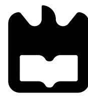
Universidade de Aveiro Departamento de Engenharia Mecânica
2014/2015

Tiago Gomes Moura

Desenvolvimento de um Sistema Robótico de Dois Braços para Imitação Gestual

Development of a Dual-Arm Robotic System for Gesture Imitation



Universidade de Aveiro Departamento de Engenharia Mecânica
2014/2015

Tiago Gomes Moura

Desenvolvimento de um Sistema Robótico de Dois Braços para Imitação Gestual

Development of a Dual-Arm Robotic System for Gesture Imitation

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Automação Industrial, realizada sob a orientação científica do Doutor Filipe Miguel Teixeira Pereira da Silva, Professor Auxiliar do Departamento de Electrónica,Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Paulo Miguel de Jesus Dias, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Pedro Nicolau Faria da Fonseca
Professor Auxiliar da Universidade de Aveiro

vogais / committee

Doutor Miguel Armando Riem de Oliveira
Investigador do Instituto de Engenharia Electrónica e Telemática de Aveiro
(Arguente)

Prof. Doutor Filipe Miguel Teixeira Pereira da Silva
Professor Auxiliar da Universidade de Aveiro
(Orientador)

agradecimentos / acknowledgements

Em primeiro lugar, sendo estes os mais importantes, quero deixar os meus agradecimentos aos meus pais por todo o apoio que me proporcionaram ao longo de todo o meu percurso académico. Sem eles nada do que alcancei seria possível e, por isso, é a eles que dedico este trabalho.
Em segundo, e não menos importante, quero agradecer aos meus orientadores, Doutor Filipe Silva e Doutor Paulo Dias, pela ajuda e dedicação prestadas ao longo deste projeto e pelo enorme conhecimento que me transmitiram. Foi com o seu apoio que consegui os resultados alcançados. São orientadores sempre presentes e dispostos a ajudar, seja qual for o desafio. Sem a sua ajuda seria impossível a realização deste projeto, por isso o meu muito obrigado aos dois.
Um especial agradecimento também ao João Torrão, investigador do LAR, que se prestou disponível para fazer uma revisão mecânica aos dois braços robóticos, sendo assim possível a continuidade do trabalho visto que as peças anteriores, em plástico, partiram e foram substituídas por peças em alumínio desenhadas pelo João Torrão. Por último, mas não menos importante, quero agradecer à minha namorada, Daniela Simões, pela ajuda na revisão da escrita principalmente no inglês e na construção das frases. Foi muito importante a sua opinião e ajuda pois tem uma perpectiva diferente por ser da área de línguas.

Palavras Chave

Imitação gestual, controlo cinemático, Jacobiano, manipulação bi-manual, sistema de captura de movimento, sensor Kinect, estrutura ROS.

Resumo

A investigação dedicada à área de robótica tem vindo a desempenhar um papel fundamental no que diz respeito à interação humano-robot. Esta interação tem evoluído em aspetos como reconhecimento de voz, caminhar, imitação gestual, exploração e trabalho cooperativo. A aprendizagem por imitação traz várias vantagens em relação aos métodos de programação convencionais, pois possibilita a transferência de novas habilidades ao robot através de uma interação mais natural. O trabalho desenvolvido pretende a implementação de um sistema robótico para imitação gestual que sirva como base para o desenvolvimento de um sistema capaz de aprender recorrendo à imitação gestual de um humano. As demonstrações foram adquiridas recorrendo a um sistema de captura de movimento humano baseado no sensor Kinect. O sistema desenvolvido permite reproduzir os movimentos capturados num robot humanoide composto por dois braços Cyton Gamma 1500 em tempo real, respeitando as restrições físicas e de espaço de trabalho do robot bem como prevenindo possíveis colisões. Os braços robóticos foram fixados numa estrutura mecânica, similar à estrutura do torso humano, desenvolvida para o efeito. Foi estudada a cinemática do manipulador com o objetivo de desenvolver algoritmos base de controlo. Estes foram desenvolvidos de forma modular de modo a criar um sistema que permite vários modos de funcionamento independentes. Foram elaborados testes experimentais com o intuito de avaliar o desempenho do sistema em diferentes situações. Estas estão relacionadas com limitações físicas associadas à imitação, como por exemplo: limites físicos das juntas, limites de velocidade, limites do espaço de trabalho, configurações singulares e colisões. Foram assim estudadas e implementadas soluções que permitem resolver estas situações.

Keywords

Gesture imitation, kinematics control, Jacobian, dual-arm manipulation, motion capture, Kinect sensor, ROS framework.

Abstract

Research in robotics has been playing an important role in human-robot interaction field. This interaction has evolved in several areas such as speech recognition, walking, gesture imitation, exploring and cooperative work. Imitation learning has several advantages over conventional programming methods because it allows the transfer of new skills to the robot through a more natural interaction. The work aims to implement a dual-arm manipulation system able to reproduce human gestures in real-time. The robotic arms are fixed to a mechanical structure similar to the human torso developed for this purpose. The demonstrations are obtained from a human motion capture system based on the Kinect sensor. The captured movements are reproduced in a two Cyton Gamma 1500 robotic arms assuming physical constraints and workspace limits, as well as avoiding self-collisions and singular configurations. The kinematics study of the robot arms provides the basis for the implementation of kinematics control algorithms. The software development is supported by the Robot Operating System (ROS) framework following the philosophy of modular and open-ended development. Several experimental tests are conducted to validate the proposed solutions and to evaluate the system's performance in different situations, including those related with joints physical limits, workspace limits, collisions and singularity avoidance.

Contents

Contents i

List of Figures iii

List of Tables vii

1 Introduction 1
- 1.1 Motivation and Objectives . . . 2
- 1.2 Dissertation Structure . . . 3

2 State-of-the-Art 5
- 2.1 Anthropomorphic Upper Body Humanoids . . . 5
  - 2.1.1 Armar *III* . . . 6
  - 2.1.2 Tombatossals . . . 7
  - 2.1.3 Justin . . . 7
- 2.2 Imitation Learning . . . 8
- 2.3 Motion Capture Systems . . . 11
  - 2.3.1 Mechanical and Magnetic Systems . . . 11
  - 2.3.2 Optical Systems . . . 12

3 Experimental Setup 15
- 3.1 Cyton Manipulator Arms: Technical Features . . . 16
- 3.2 Torso Design . . . 19
  - 3.2.1 Design Options . . . 19
  - 3.2.2 Workspace Analysis . . . 21
- 3.3 Software Development Tools . . . 26
  - 3.3.1 Robot Operating System . . . 27
  - 3.3.2 OpenNI . . . 31

4 Kinematics and Robot Motion 35
- 4.1 Kinematic Analysis . . . 36
  - 4.1.1 Direct Kinematics . . . 36
  - 4.1.2 Inverse Kinematics . . . 38

4.1.3 Differential Kinematics 39
4.1.4 Kinematics Control for Gesture Imitation 40
4.2 Point-to-Point Control Mode: Implementation and Evaluation 42
4.3 Continuous Control Mode: Implementation 47
4.3.1 Inverse Jacobian Solution 48
4.3.2 Closed-Loop Inverse Kinematics 50
4.3.3 Performance Evaluation 53

5 Software System Integration **61**
5.1 Motion Capture and Kinematic Mapping 62
5.2 Dual-Arm Robot Control 64
5.2.1 Overcome Physical Aspects 66
5.2.2 Singularity Robust Inverse Kinematics 67
5.3 Self Collision Detection and Avoidance 69
5.4 Overall ROS-Based Software Architecture 75

6 Results and Evaluations **83**
6.1 Physical, Velocity and Workspace Limits 83
6.2 Singularity Avoidance 89
6.3 Self Collision Detection 91
6.4 Final Demonstration 93

7 Conclusions **99**
7.1 Results Discussion 99
7.2 Final Conclusions 101
7.3 Future Work 102

References **103**

Appendices **106**

A Cyton hardware API public functions **107**

B Actin Simulator **111**

C Structural analysis **113**

D Linear Trajectory Planning **117**

E Launch Files **119**

F User Guide **123**

List of Figures

2.1 *Robonaut R2 by NASA (Diftler et al., 2011).* 6
2.2 The humanoid robot ARMAR III (Asfour et al., 2006a). 6
2.3 The Universitat Jaume I humanoid torso *Tombatossals* (del Pobil et al., 2013). 7
2.4 German Aerospace Center (DLR) *Rollin' Justin (Borst et al., 2009).* 8
2.5 Imitation model approach by (Weber et al., 2000). 10
2.6 Application example of mechanical motion capture system (Metamotion, 2012). 11
2.7 Basic Vicon MX architecture (Miedema, 2010). 13
2.8 Example of optical system used for motion capture (Dinis, 2011). 13
2.9 Microsoft Kinect Xbox 360. 13
2.10 Kinect depth system overview (left).Representation of three phases of human motion capture algorithm used in the Kinect sensor (right) Biomechanics. . . . 14

3.1 Overall system architecture. 16
3.2 Cyton Gamma 1500 dimensions. 19
3.3 3D model of a solution to the structure that allows to adjust the angle and the distance between the two arms assembled in sideways of the torso. 20
3.4 3D model of a solution to the structure that allows to adjust the distance between the two arms assembled vertically to the torso. 21
3.5 Length of body parts as function of body height (Drillis et al., 1964). 22
3.6 Representation of the three simulated configurations in a 3D plot. Green blocks represent the torso, black lines represent the robotic arms and black cylinders are the used joints. a) Vertical configuration, b) Sideways configuration and c) Sideways configuration with an angle of 45 degrees. 23
3.7 Workspace representation of right arm (blue) and left arm (red) assembled to torso (green) in a vertical configuration. 23
3.8 Workspace representation of right arm (blue) and left arm (red) assembled to torso (green) in a lateral configuration. 24
3.9 Workspace representation of right arm (blue) and left arm (red) assembled to torso (green) in a side mounted configuration with 45º. 24
3.10 Final hardware setup. 26
3.11 Example of three topics exchanging messages. 28

3.12 Services communication mechanism. 28
3.13 Action Server and Action Client interaction in ROS. 29
3.14 End-effector control ROS architecture. 30
3.15 Joint level control ROS architecture. 31
3.16 OpenNI architecture (Avancini, 2012). 32
3.17 NITE calibration pose. The user should stay in front of the sensor with a pose similar to the one presented in the figure (Avancini, 2012). 33

4.1 Coordinate frames of 3 DOF Cyton arm. 37
4.2 Curve of relation between input and real joint velocities. Joint 1 (top), joint 2 (middle) and joint 3 (bottom). 43
4.3 Joint space control flowchart. 44
4.4 ROS architecture of joint control algorithm. 45
4.5 Graphics of the behavior of joint angles over time (top) and velocity profiles (bottom). 46
4.6 Inverse kinematics algorithm using Jacobian inverse with estimated joint positions. 49
4.7 Velocity control ROS-based implementation. 50
4.8 Block diagram of the inverse kinematics algorithm with Jacobian inverse (Sciavicco and Siciliano, 1996). 51
4.9 Inverse kinematics algorithm using Jacobian inverse with feedback of joint positions. 52
4.10 Comparison between the desired Rd, estimated Rest and effective Rread positions for a movement with execution time of 5s (Top) and 10s (Bottom) along the x-axis. 54
4.11 Comparison between the desired Rd, estimated Rest and effective Rread positions for a movement with execution time of 5s (Top) and 10s (Bottom) along the z-axis. 55
4.12 Comparison between the desired Rd, estimated Rest and effective Rread positions for a movement with execution time of 5s, k=1.2 and k=2.2 (Top) and 10s, k=1.2 and k=2.2 (Bottom) using control feedback algorithm. 57
4.13 Error between the desired and real position in Cartesian space measured in the x-axis for the x-axis movement (top) and z-axis for the z-axis movement (bottom). 58

5.1 Capture of depth and RGB images and transformation (left) and a capture with just transformations using Rviz. 62
5.2 Trajectory generator of hand tracking ROS implementation. 64
5.3 Hardware implementation and representation of global (black axis) and local (white axis) reference frames. 65
5.4 Physical aspects representation. 66
5.5 Singularity simulation using Matlab. The robotic arm links are represented as red lines and its joints as black cylinders. The singularity region is marked as a dotted cylinder, the executed trajectory as a blue line and the final desired position as a black asterisk. A perspective view (left) and top view (right) of the system are represented. 69

5.6 Two line intersection in 2 dimensional plane and the associated vectors. 70
5.7 Collision detection algorithm diagram. 72
5.8 Sequence representing a movement that causes collision between the two robotic arms. Starting position (top left), intermediate position (top right) and intersection position and collision detected (bottom center). 73
5.9 Collision detection ROS implementation simplified. 74
5.10 Imitation Algorithm block diagram. 76
5.11 Imitation algorithm ROS diagram with both arms. l and r refer to left and right respectively. 77
5.12 *XML* code of launch file. 80
5.13 Command window of openni_tracker (left) and cyton lauch (right). 81
5.14 Command window to control left and right robotic arms. 81

6.1 Graphical representation of the comparison of Cartesian position of human hand and end-effector (top), joint angles (middle) and the error amplitude (bottom) in function of time. The Cartesian position of the hand is already filtered. The vertical black lines represent the instants of time that a joint reaches its physical limit and the consequent rise of the error amplitude. 84
6.2 Graphical representation of the comparison of Cartesian position of human hand and end-effector (top), joint velocities (middle) and the error amplitude (bottom) in function of time. The Cartesian position of the hand is already filtered. The horizontal lines represents the joint velocities limits (red for joint 3 and yellow for joints 1 and 2) and the vertical lines represents the instant of time where the joints exceeded the velocity limits, consequently increasing the error amplitude. 86
6.3 Graphical representation of the comparison of Cartesian position of human hand and end-effector (top), joint angles (middle) and the error amplitude (bottom) in function of time. The Cartesian position of the hand is already filtered. The vertical black lines represent the instants of time that the human hand exceeded the workspace limits of the robot and, consequently increasing the error amplitude. 88
6.4 Top view representation of the left human hand and right end-effector trajectory passing through the singularity region (dotted cylinder), starting from the asterisk position. 90
6.5 Joint angles during the experiment when the end-effector passed through the singularity region (between the two vertical black dotted lines). 90
6.6 Representation of a Matlab simulation of self collision detection using experimental data. 91

6.7 Graphical representation of the comparison of Cartesian position of the human hand and end-effector (top) and joint angles (bottom). The left plots represent the position of the left human hand and right end-effector as well as the joint angles of the right robotic arm. The right plots represent the right human hand and left end-effector as well as the joint angles of the left robotic arm. The vertical lines represent the time instant when collisions occurred. 92
6.8 Human demonstrator imitation acquisitions. Sequence from top to bottom, left column first and then right column. 94
6.9 Comparison between left end-effector and right human hand trajectories in x-axis (top), y-axis (middle) and z-axis (bottom). 95
6.10 Comparison between right end-effector and left human hand trajectories in x-axis (top), y-axis (middle) and z-axis (bottom). 96
6.11 Comparison between the left robotic elbow and right human trajectories (top) and respective error (bottom). 97
6.12 Comparison between the right robotic elbow and left human trajectories (top) and respective error (bottom). 98

B.1 Actin simulator interface and manipulator configuration window. 112

C.1 Frame analysis of torso structure applying bending moments (yellow curved arrows) to the aluminum bars that support the robotic arms, considering that the structure is fixed to the base and also the gravitational force applied in the center of the structure as well. The bending moment is equivalent to the force applied in the end-effector of the arm as if it was carrying a load of 1,5Kg. The values presented are related to displacement from the initial position of the structure. . 115
C.2 Torsion analysis of the structure applying the same bending moments of previous figure. 115

D.1 Curves of position, velocity and accelaration. 117

E.1 Launch file related to left robotic arm. 121
E.2 Launch file related to right robotic arm. 122

List of Tables

3.1 Cyton Gamma 1500 general technical specifications. 17
3.2 Cyton Gamma 1500 joint specifications. 18
3.3 Dynamixel servos specifications. 18
3.4 Workspace specifications for vertical configuration 23
3.5 Workspace specifications for lateral configuration. 24
3.6 Workspace specifications for lateral configuration with 45º. 24

4.1 Denavit parameters of 3 DOFs Cyton arm 37
4.2 Maximum error between desired and real positions for a movement performed along the x-axis. 56
4.3 Maximum error between desired and real positions for a movement performed along the x-axis. 59

5.1 Topic names and specification. 78
5.2 Node names and specifications. 79

A.1 Public member functions of hardwareInterface class. 109

Chapter 1

Introduction

Over recent years, the interest in robotic systems dedicated to complex tasks has increased remarkably influenced by the use of state of the art supporting technologies, the demand for innovative solutions and the search for new areas of potential application (Siciliano and Khatib, 2008), (Bekey and Sanz, 2009). The field of robotics is rapidly expanding into human environments and particularly engaged in its new challenges: interacting, exploring, and working with humans. In order to work in the environment of humans, this new generation of robots should have a human-like behavior in terms of motion skills, advanced adaptation and learning capabilities. At the same time, it will be increasingly important to provide multimodal, natural and intuitive modes of communication, including speech and gesture.

Robot learning is a fundamental and unavoidable step towards more robust and autonomous robot systems, when comparing with those with built-in knowledge (*i.e.*, with explicit pre-programmed behaviors) (Billard et al., 2007), (Argall et al., 2009), (Billard and Grollman, 2013). In this context, robot learning from demonstration is a powerful approach in which the robot acquires training examples from human demonstrations. Acquiring teacher demonstrations is a key step when transferring skills from humans to robots through imitation that can be accomplished in many different ways: recording the demonstrations provided by the human teacher's own motion using vision or motion capture systems, recording state-action pairs whilst the robot is passively tele-operated by the human teacher, or, alternatively, using more user-friendly interfaces, such as kinesthetic teaching in which the human moves directly the robot's parts (Asfour et al., 2006b).

This dissertation work was proposed by the Institute of Electronics and Telematics Engineering of Aveiro (IEETA) in the scope of current activities aiming to apply robot learning approaches to the specific domain of manipulation. The work involves technical and scientific knowledge in numerous areas of engineering such as mechanics, robotics, control and programming.

1.1 Motivation and Objectives

There is an increased interest in the specific problems of dual-arm manipulation (Asfour et al., 2006b), driven by recent advances in both anthropomorphic robots (Asfour et al., 2006b), (Asfour et al., 2006a), (Bekey and Sanz, 2009), (Borst et al., 2009), (del Pobil et al., 2013), imitation learning (Schaal, 1999), (Billard, 2001), (Nehaniv and Dautenhahn, 2002), (Schaal et al., 2003), (Alissandrakis et al., 2007), (Argall et al., 2009), (Englert et al., 2013), and bi-manual industrial manipulators (Hermann et al., 2011), (Smith et al., 2012), (Warren and Artemiadis, 2014). Since the beginning of the study of humanoid robots there has been an effort to approximate the behavior of humanoids to the human behavior. This approach has been improved in several aspects such as walking, manipulating objects, visual perception, voice recognition and gesture imitation. The study of gesture imitation in humanoid robots is growing even more because of the idea of the robot operating alone is vanishing due to the new concept of human-robot iteration. This new vision of programming robots using gesture imitation can be very useful in various areas such as medical surgery, bomb disposal, search outside the planet, among others. If teleoperation is combined with gesture imitation, the interaction can be much more powerful and can help solve situations that are more difficult with conventional modes of programming.

Recently, there have also appeared a few dual-arm systems proposed for industrial use (for instance: SDA10 Motoman, FRIDA ABB dual-arm robot, bi-handed Cyton, etc) with the motivation that dual-arm systems are more compact and cheaper than two single-arm units. Most of the developments of dual-arm setups for research are motivated by human-like appearance (outer shape similarity), high degree of task space redundancy and increased manipulability. Despite the wide range of existing hardware platforms, there are several common problems to be solved from low-level control to high level task planning and execution. When comparing with single-arm manipulation, the higher complexity of dual-arm manipulation means more advanced system integration, high level planning and viable control approaches, even more when coordination between arms is a prerequisite.

Some of these arguments served as motivation for this dissertation aiming the development of a dual-arm robotic system able to perform gesture imitation of human demonstrations. These demonstrations are obtained through a motion capture system based on a Kinect sensor. The human-robot interface should allow the real-time reproduction of the captured movements (as closed as possible), while respecting physical constraints such as joints limits, joints velocities and available workspace. At the same time, problems associated with self collision avoidance and singular configurations will also be addressed. This work was developed from scratch based on two Robai Cyton Gamma 1500 robotic arms available in the laboratories. It can be seen as a first development phase before the research in visuomotor coordinating and imitation learning can be conducted. In the pursuit of this goal, the dissertation work was organized in the following main sub-goals:

- Design of the upper-body torso to install the robotic arms bearing in mind similarity to

human's kinematics structure, workspace and range of motion, while keeping it simple;

- Kinematics analysis of the manipulator in order to develop algorithms for robot kinematic control;
- Development of the overall system's software architecture with the integration of modular processes on a stand-alone basis;
- Evaluation of the system's performance through experiments conducted in different scenarios and supported by adequate metrics.

1.2 Dissertation Structure

This dissertation is divided into seven chapters supported by six appendixes. Chapter 1 presents an introduction to the developed work and a brief explanation of the studied topic. Chapter 2 presents a review on humanoid torsos, some imitation learning techniques and approaches and also the usual motion control systems available and suitable for this dissertation. Chapter 3 provides an overall overview of the experimental setup, presenting some specifications of the Cyton Gamma robotic arms and also a study of the workspace of the all system. It then shows some solutions for the torso structure design and describes the software used to control the robotic arms as well as to obtain the skeleton data from the Kinect sensor. Chapter 4 provides a kinematic analysis of the manipulator and an explanation of the processes used to execute the robot control. Chapter 5 describes the software implementation, the problems associated with gesture imitation and the adopted strategies to solve them and then the overall software implementation. In Chapter 6 the experimental results are discussed and an evaluation of the system facing different limit situations it is presented. Finally, Chapter 7 presents some conclusions obtained during this project and also some possible ideas to improve in future work.

Chapter 2

State-of-the-Art

This chapter presents a brief overview of recent developments on topics of interest for this work, such as upper-body humanoid robot designs, imitation learning algorithms and human motion capture systems. Current anthropomorphic robots are a useful source of information to draw ideas in terms of torso design (structure and physiognomy), hardware and software architectures or control algorithms. Although this work is focused on a recording and reproducing perspective, the main concepts and challenges of imitation are also addressed. Finally, the most common motion capture systems are described, mainly the gold-standard VICON system, whose data is often used as a ground-truth measurement and the much cheaper Kinect sensor adopted in this work.

2.1 Anthropomorphic Upper Body Humanoids

Humanoid research is a topic of interest for researches for long ago both in locomotion and manipulation. However, the research in locomotion overcome to dexterous manipulation research. Locomotion was simpler to start studying due to the usage of wheels to move the robot. Wheeled mechanisms have been known for a long time and it was not difficult to implement them in robots. On the other hand, dexterous manipulation had some more complex aspects to take into account such as: knowing the environment to know where the object is, have a gripper capable to grab it and, at most, 6 DOFs to execute dexterous manipulations. Nowadays, some work has been developed in this area and there are some interesting projects related to the upper limbs of humanoid robots.

Robonaut, developed by NASA, is maybe the most powerful in this area. The project began in 1997 and the main goal was to develop a humanoid robot that could assist astronauts in their tasks. The first prototype was R1 that could perform maintenance tasks or explore the surface of the moon or Mars. In 2010 a more dexterous, faster and more technologically advanced humanoid robot ever made was revealed. The *Robonaut* R2, presented in Figure 2.1, have 42 DOFs in total, more than 350 sensors and 38 power processors and is primarily composed by aluminum and non-metallic materials (Diftler et al., 2011).

Figure 2.1: *Robonaut R2 by NASA (Diftler et al., 2011).*

At research and academic level, there are some important cases studies among which the ARMAR III, the Baxter, the Tombatossals, the Justin, the Bi-Handed Cyton, the ROMAN. Those humanoid torso robots are mostly used to manipulate objects in a restricted workspace without moving its body position. In the next sub-sections the most relevant humanoid torso robots are outlined.

2.1.1 Armar *III*

ARMAR III, presented in Figure 2.2, was designed to closely mimic the sensory and sensory-motor capabilities of a human. It should be able to perform different activities in a household environment and to deal with the objects encountered in it. This robot has 14 DOFs in both arms and 3 in the torso being each arm equipped with a five-fingered hand with 8 DOFs.

Figure 2.2: The humanoid robot ARMAR III (Asfour et al., 2006a).

ARMAR III has some perception skills that allows it to recognize objects of interest and localize them to perform the grasping. It has also motor skills provided by different inverse

kinematics algorithms required to execute manipulation tasks. The mechanical joint limits are avoided by the arm redundancies, which allows to generate human-like manipulations. This robot also has a collision avoidance module, which means that tasks are only executed if there is no risk of collision (Asfour et al., 2006a).

2.1.2 Tombatossals

Tombatossals, presented in Figure 2.3, is a less known humanoid robot but its characteristics are interesting for the topic of this dissertation. It has 2 robotic arms with 7 DOFs each and uses the Kinect sensor to obtain a three-dimensional reconstruction of the scene. The main goal of the robot is to grasp and move the target object, known as *pick and place.* Its software architecture is based in the Robot Operating System (ROS) framework.

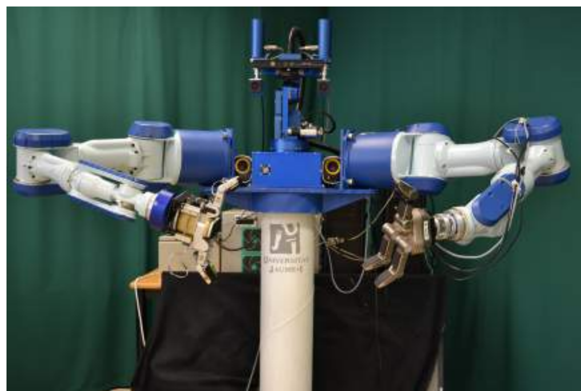


Figure 2.3: The Universitat Jaume I humanoid torso *Tombatossals* (del Pobil et al., 2013).

2.1.3 Justin

The concept of the *Justin* humanoid torso is based on two modular 7 DOFs arms and two four-fingered hands. It was designed to perform two-handed manipulation and should be able to reach objects up to 2m height as well as objects on the floor. At first, this robot only had the upper body, however, later, a mobile platform it was implemented that allowed the robot to interact with humans in different kinds of tasks. Figure 2.4 presents the *Rollin' Justin*, the upper torso combined with the mobile platform.

Figure 2.4: German Aerospace Center (DLR) *Rollin' Justin (Borst et al., 2009).*

Rollin' Justin main features are the visual tracking that gives it the capability to track and grasp freely moving objects in 6 DOFs. The robot also allows commands via speech recognition and dual arm path planning (Borst et al., 2009), (Zacharias et al., 2010).

2.2 Imitation Learning

"Imitation is the ability of an agent to observe a demonstrator and act like it. This is an open-ended definition since "acting like" can be defined in many ways."

(Kurt, 2005)

Human gesture imitation is a complex problem that can be identified by who, when, what and how to imitate problems and how the success of the imitation can be evaluated.

Imitation learning, or programming by demonstration, has become a relevant topic of study in robotics and it is related with some research areas such as machine learning, human-robot interaction, machine vision and motor control (Billard et al., 2007). This technique is a natural interaction with a robot that, along time, will be accessible to all people in daily tasks. One of the objectives of the many studies around this technique is to make learning faster than other learning methods such as reinforcement learning.

Currently there are three well-established types of approaches for teaching new skills to robots that are: direct programming, imitation learning and reinforcement learning papa-rameters (Kormushev et al., 2013). Those approaches are all still in use but each one has its advantages and disadvantages. Direct programming is the lowest-level approach and it is considered a programming method because actually it is not a learning method. However, this approach is actively used, usually in industrial environments, where the workspace is well structured and the movement actions are well defined.

Reinforcement learning is the process of learning from trial-and-error that uses a reward function algorithm which acts differently depending on the performance of the robot relatively to the desired goal. The motivation of this approach is to offer new abilities that other approaches did not offer. First, the robot is able to learn new tasks which cannot be demonstrated or directly programmed by teacher. Secondly, the robot is able to learn how to optimize the achievement to goals even if it has no analytic formulation or known solution. Finally, it is able to adapt a skill to a new previously unseen task by itself, without the need of readjust parameters.

Imitation learning, also known as programming by demonstration (Billard et al., 2007) or learning from demonstration (Argall et al., 2009) uses three main methods (Billard et al., 2007):

- Kinesthetic teaching: This consists in moving the robot's body manually and record its motion. This method is usually used in lightweight robots and the teaching can be performed in a continuous way or recording discrete parts of the trajectory. This method usually needs a gravity compensation controller with aim to reduce the error introduced by the weight of the robot.
- Teleoperation: This method consists in remotely control the robot using an input device. The main difference between this and Kinesthetic method is that with teleoperation the teacher can be located in a different geographically location. Distance can be an issue because of the time delay, also teleoperation makes it more difficult to feel the limitations and capabilities of the robot than using Kinesthetic approach. On the other hand, the setup of teleoperation can be simpler and the devices used can give the teacher different information, such as forces rendered by haptic devices.
- Observational or imitation learning: In this case, the movement is demonstrated by the teacher's own body and it is observed using motion capture stereo cameras systems. This method has the disadvantage of correspondence problem that is related to the difficulty of mapping teacher and robot kinematics.

Related to imitation learning are associated some problems such as: pose estimation, movement recognition, pose tracking, body correspondence, coordinate transformations, resolution of redundant DOFs, modularization of motor control among others (Schaal et al., 2003). Each one of those topics is complex and deserves to be carefully studied to obtain successful results. In this dissertation the main focus was to develop computational algorithms to control the motors in order to perform human gesture imitation. However, were considered some of the refereed topics were considered, such as pose tracking, body correspondence, coordinate transformations and resolution of redundant DOFs that are going to be explained along the document. From the point of view of motor control, according to (Schaal et al., 2003), there are some statistical and mathematical approaches in terms of control policies: imitation by learning policies from demonstrated trajectories, imitation by model-based policy learning and matching of demonstrated behavior against existing movement primitives.

Imitation by learning policies from demonstrated trajectories was studied in order to get a stable approach to imitation with a small amount of information. In this approach the task goal is known and the demonstrated movement is used as feedback to improve the imitation and reduce the error. It is usual that a human demonstrates the task and its movement trajectory is stored with motion capture system. The robot aims to demonstrate the trajectory in task space based on position of the end effector and tries to adopt an arm posture as similar as possible to the demonstrator's one. The desired end-effector trajectory is approximated by via-points and it can generate some perturbations caused by possible gaps in trajectory which can cause huge motor velocities to catch up the movement.

Another approach, pursued by (Billard, 2001), is more biologically inspired, uses joint angular trajectories of the human demonstration which are segmented using zero velocity points. In it a second order differential equation is used to approximate the joint movement to the activated muscles and a trajectory with a bell-shape velocity profile is generated (Schaal et al., 2003).This imitation system allows to perform very complex movements.

Imitation by model-based policy learning was first introduced by (Atkeson and Schaal, 1997). It is based on learning a task model and using an optimization criterion for that task. This approach was normally used in cases where the robot may not be doing exactly the same task as the demonstrator.

Another important approach is matching of demonstrated behavior against existing movement primitives (Schaal et al., 2003). It is based mapping the perceived behavior onto a set of existing primitives. To do this there are two main approaches, according to (Schaal et al., 2003), matching based on kinetic and kinematic outputs. The first uses outputs such as forces and torques but they are hard to get from demonstrations. In the second approach the demonstrated movement can be directly compared with the primitives. (Weber et al., 2000), use perceptual motor primitives that combine the perceptual and motor routines with the function of mirror neurons in primitives. The Figure 2.5 represents the main algorithm used by (Weber et al., 2000) which consists of motion extraction, matching to combinations of innate primitives and reconstruction.

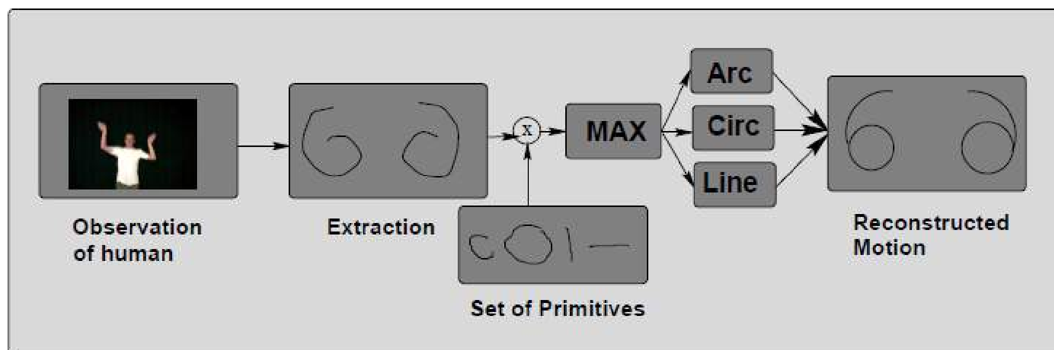


Figure 2.5: Imitation model approach by (Weber et al., 2000).

2.3 Motion Capture Systems

Motion capture system consists of a mechanism that is able to measure the position and orientation of an object in the environment. The captured data is mapped in Cartesian space and can be performed with different processes.

There are some motion capture systems that use different techniques to capture the data. The well known and most used to capture human motion are: mechanical, magnetic and optical systems. Below these different motion capture systems are briefly explained.

2.3.1 Mechanical and Magnetic Systems

Mechanical systems used in human motion capture are usually composed by a set of potentiometers placed in the articulations that will be captured. This system has the disadvantage of being intrusive since the human needs to have all the equipment attached to its body. However, it has the advantage of being an equipment of absolute measure, with low latency, and it is not affected by magnetic fields or undesired reflections. In Figure 2.6 an application example of mechanical motion capture system is shown.

Figure 2.6: Application example of mechanical motion capture system (Metamotion, 2012).

Magnetic systems use a set of sensors attached to the human articulations that measures their position and orientation relatively to a transmitting antenna. This system is affordable and its computational processing cost is lower but, in contrast, it has the disadvantage of being affected by external magnetic signals. This system is also connected to the antenna by cables. Nevertheless, this disadvantage has been remedied with wireless magnetic systems.

2.3.2 Optical Systems

Optical systems usually use markers attached to the demonstrator body, which can use a especially designed suit. The system is composed of high-resolution cameras strategically positioned to track the markers during the demonstration. Using different triangulation strategies, depending on the system, the two dimensional image of each camera is computed by software obtaining the 3D coordinates of the markers.

In most cases these systems are accurate, easy to change markers configuration and use wireless markers. However, data post processing is usually more intense, the system is expensive relatively to others and the motion capture needs to be done in controlled environments, without reflection points to avoid interferences and the markers cannot be occluded.

Optical systems can use passive or active markers.

- Passive markers: These markers are made of retro-reflective material in order to reflect the light emitted by the cameras. A calibration is needed so only the markers can be identified.

- Active markers: These kind of markers uses LED technology powered by small batteries instead of reflecting material.

Optical systems can also capture human motion without using markers. This kind of systems is usually called markerless motion capture systems and they do not need special equipment to track the demonstrator movement. The movement is captured using cameras, stereo or infrared, and the motion capture process is completely done via software. This has the advantage of being non intrusive, although more advanced algorithms are needed to track the articulation positions. A good example of a passive marker optical system is Vicon, which is a very accurate system but also expensive. On the other hand, a very well known markerless system is the Kinect sensor by Microsoft, which has far less precision but also a far less price and a much easier and fast setup. Below these two motion capture systems are presented with aim to describe some of its specifications and strategies to capture human motion (Nogueira, 2011).

Vicon

Vicon motion capture system is known to be one of the most accurate three-dimensional record systems, especially for acquisition of human body motion. The system uses reflective markers, placed in articulations of the human body. The most basic Vicon architecture, schematized in the Figure 2.7, consists of 8 cameras and an Ethernet module responsible to send the data to a host computer. The goal of the *Vicon* system is to track and reconstruct the position of these markers in 3D space and it is able to form each marker trajectory and represent its path.

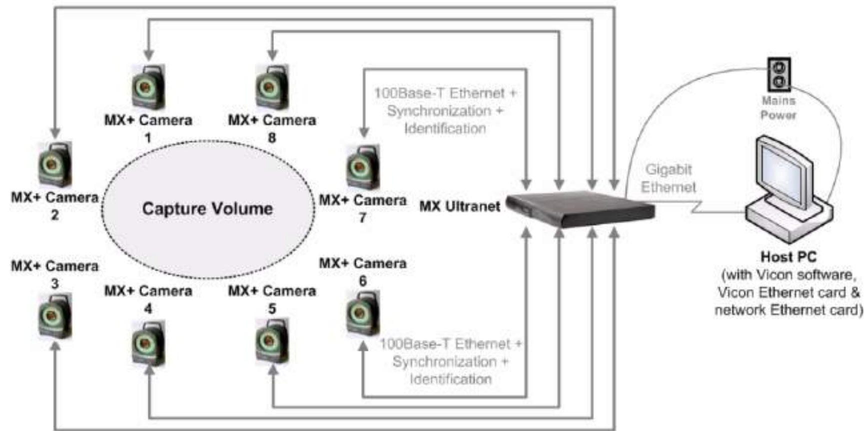


Figure 2.7: Basic Vicon MX architecture (Miedema, 2010).

Figure 2.8: Example of optical system used for motion capture (Dinis, 2011).

Kinect Sensor

For a long time, robots and computers were able to analyze images that are provided by cameras and, with hard effort, they could extract information about objects by processing two dimensional images. The computational cost was high and the precision and quality of the measures were not so good. The 3D sensors, such as the *Kinect* shown in Figure 2.9, changed all these with the implementation of depth sensors along with RGB cameras.

Figure 2.9: Microsoft Kinect Xbox 360.

It can be considered a special case of 3D sensors based on light coding. Actually, the technology used in the Kinect sensor is active triangulation with structured light. Kinect sensor have an infrared projector that projects a speckle dot pattern and an infrared camera receives the pattern and the data is processed and a depth map is created, as schematized in Figure 2.10 (left).

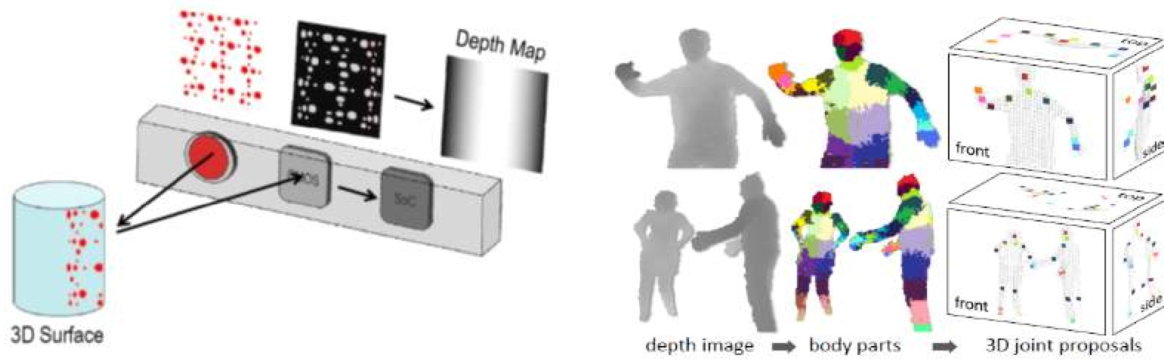


Figure 2.10: Kinect depth system overview (left).Representation of three phases of human motion capture algorithm used in the Kinect sensor (right) Biomechanics.

The human motion capture performed by Kinect sensor is based on two processes: firstly computing a depth map, then inferring the body position of the subject. The body part recognition algorithm implemented on Kinect sensors is based in a large database of motion capture of human actions. This database consists of approximately 500 000 frames that covers a wide variety of poses in a few hundred sequences of driving, dancing, kicking, running, navigating menus, etc. This classifier uses no temporal information, uses only static poses, because the changes in pose from frame to frame are almost insignificant. The algorithm defines several localized body part labels that cover the body and some of them define a skeletal joint and others fill the gaps and can be used to predict other joints. Figure 2.10 (right) represents a general overview of this process, representing the depth image, the defined body parts and the 3D joint proposal (Shotton et al., 2011).

Chapter 3

Experimental Setup

This chapter describes the hardware and software technologies used to develop the dual-arm robotic system for gesture imitation. As in most systems, the technology used deeply influences the design options that have to be taken. From the very beginning of the project a few assumptions underlying the work were defined: First, the 7 DOFs of the robot arms should be reduced to a 3 DOFs solution, promoting the similarity with the human arm kinematic structure (shoulder, elbow and wrist), although with less DOFs, and its natural workspace. Secondly, this application requires developing a real-time human motion capturing system that works without special devices or markers. The choice of a Kinect sensor resulted from several factors, including a compromise among accuracy, sensing range and price. At this level, it is assumed that the 3D motion capture system is stable by acquiring full information about the upper-body without the occurrence of self-occlusions. Thirdly, the software architecture should be distributed supporting the implementation of modular processes running on a stand-alone basis. The key element for this software architecture is the Robot Operating System (ROS) that provides an extensive list of libraries and tools to help create robot applications.

The software architecture is based on the Fuerte version of the ROS framework under Linux, using C/C++ programming environment. Figure 3.1 illustrates the overall system's architecture both at the hardware and software levels. The main hardware components consist of the anthropomorphic robotic arms, a Kinect sensor and the central processing unit (PC-based). The main software components include the ROS framework and libraries, the OpenNI driver and the actionlib libraries. In the line with this, the following sections describe the design of the torso, where the two arms and the Kinect sensor will be installed, and the software development environments employed.

The communications protocol between the Linux computer and the robot system is established through USB controllers, with a speed limit of approximately 4.5 Mbps, using the functions provided by the Cyton hardware API (see Appendix A for more details) and action goals, from *actionlib,* to send the commands. In general, the goal function takes as input argument a compact vector with information for all robot's joints. All actuators are connected

through a single, daisy-chained cable.

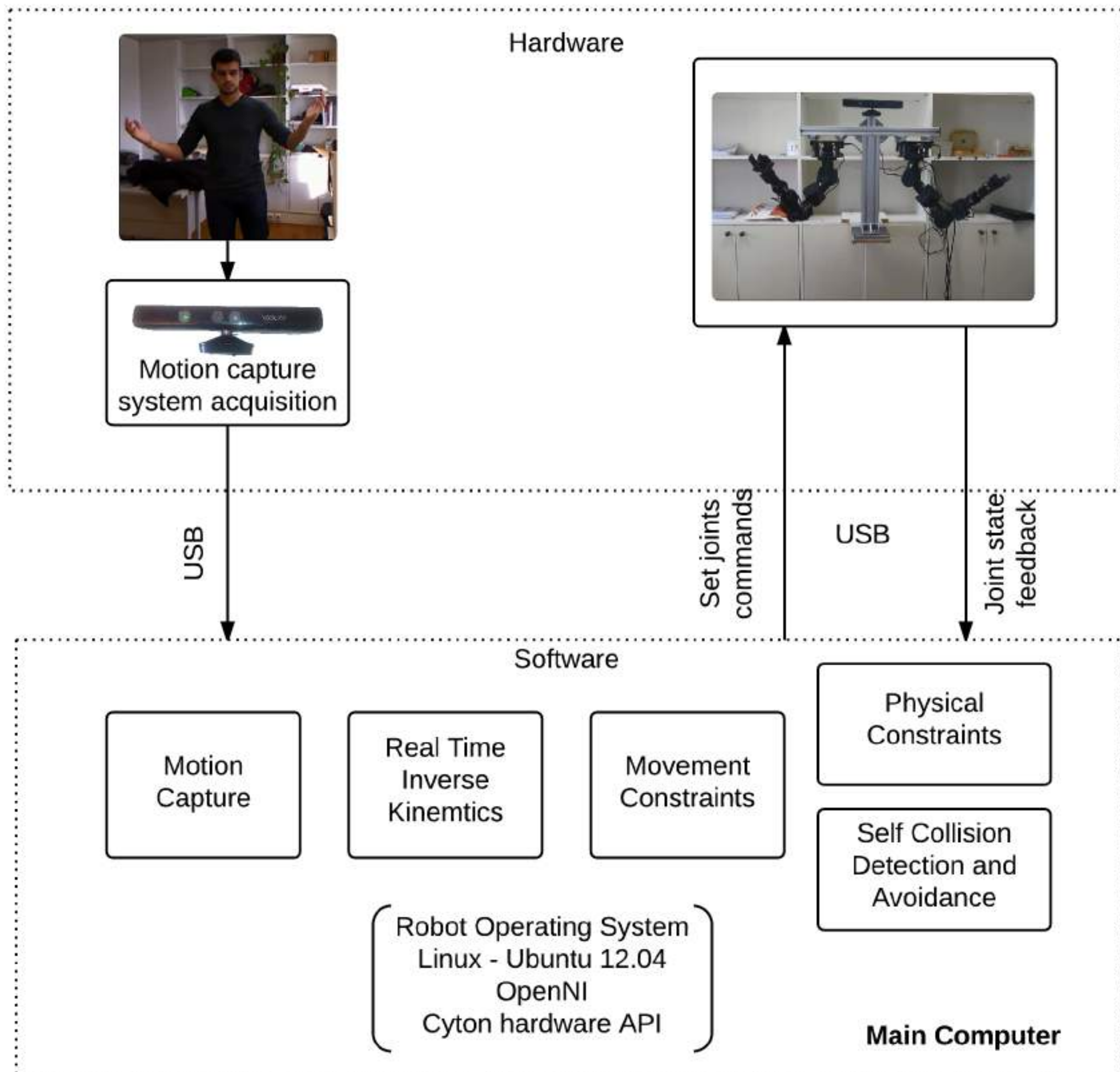


Figure 3.1: Overall system architecture.

3.1 Cyton Manipulator Arms: Technical Features

Cyton Gamma 1500 was introduced by Robai corporation and this model offers increased joint torques compared to other versions. Each robotic arm has 7 DOFs and also a motor to control the gripper. These arms have kinematic redundancy that enables the placement of the end effector at a position and orientation in a variety of different ways. It can be interesting to avoid obstacles or to reach the goal in different configurations but it turns the kinematics of the arm more difficult. Each joint actuator can provide position, speed, load, voltage and temperature feedback information and the user is able to configure these

parameters through a XML[1] file. In Table 3.1 some technical specifications[2] of the Cyton Gamma 1500 manipulator are described. They are related to physical characteristics such as its weight and length, to its performance such as its maximum linear speed and repeatability and also to its electrical and control interface.

Table 3.1: Cyton Gamma 1500 general technical specifications.

Specifications	
Total Wight	2 Kg
Maximum Payload	1500g at full range
Reach from base to tip	76.15 cm
Maximum linear arm speed	5 cm/sec
Repeatability	+/- 0.5 mm
Input Voltage	100-240V AC or 12 DC 2A battery
Current	2.5 A max in normal use
Control interface	USB or RS485
Total independent joints	7

Related to joint specifications, presented in Table 3.2, are described its joint angle and velocity limits as well as the correspondent servo.

[1]eXtensible Markup Language
[2]http://outgoing.energid.info/Robai/cyton_gamma_1500.pdf

Table 3.2: Cyton Gamma 1500 joint specifications.

Joint	Angle limits (degrees)	Velocity limits (degrees/s)	Servo model
Shoulder roll (joint 0)	-150 to 150	75	MX-64
Shoulder pitch (joint 1)	-105 to 105	75	MX-64
Shoulder yaw (joint 2)	-105 to 105	75	MX-64
Elbow pitch (joint 3)	-105 to 105	65	MX-28
Wrist yaw (joint 4)	-105 to 105	110	MX-28
Wrist pitch (joint 5)	-105 to 105	330	MX-28
Wrist roll (joint 6)	-150 to 150	330	MX-28

The servo specifications[3,4] are described in Table 3.3. The servo motors were developed by Dynamixel and the robotic manipulator has two different models, MX-64 and MX-28. Both use the PID controller as a main control method, gives position, temperature, load and input voltage feedback and they run from 0 to 360 degrees with endless turn.

Table 3.3: Dynamixel servos specifications.

Servo Model	Servo resolution (degrees)	Servo gear reduction ratio	Position Sensor	Stall torque N.m
MX-64	0.088	200 : 1	Contactless absolute encoder (12bit, 360 degrees)	6.0 (at 12V, 4.1A)
MX-28	0.088	193 : 1	Magnetic Potentiometer (12bit, 360 degrees)	2.3 (at 12V, 1.5A)

Figure 3.2 shows a representation of the Cyton Gamma 1500, its joints and also its dimensions. As explained before, in this project were just used 3 DOFs, corresponding to joints 0,1 and 3 of the robotic arm.

[3]http://support.robotis.com/en/product/dynamixel/mx_series/mx-64.htm
[4]http://support.robotis.com/en/product/dynamixel/rx_series/mx-28.htm

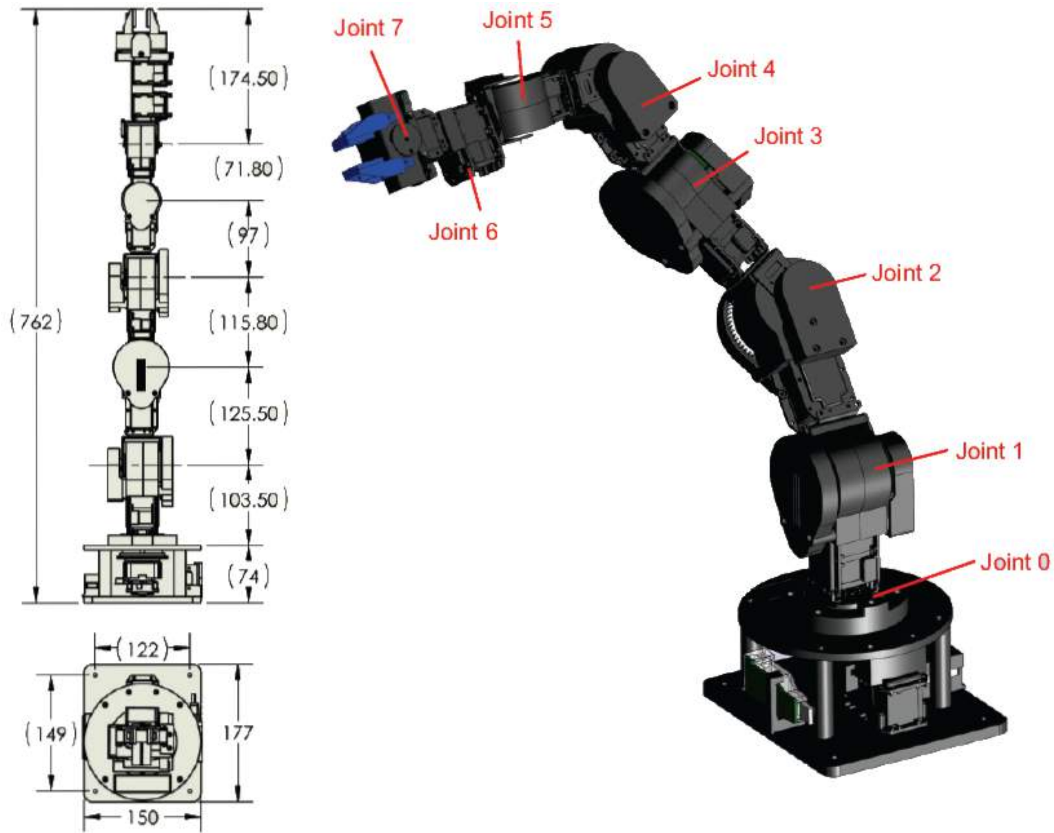


Figure 3.2: Cyton Gamma 1500 dimensions.

3.2 Torso Design

To install the two robotic arms as well as the Kinect sensor in a single system it was necessary to develop the torso structure. That required a mechanical structural design and study, presented in Appendix C. For the design of the structure, were taken in account the mounting orientation and position of both arms that represent a similar configuration and workspace of a human torso. In this section the design of two viable options to install the robotic arms in two different configurations, vertical and lateral, are presented.

3.2.1 Design Options

At this stage it was necessary to meet the physical aspects of the arms as well as the functional aspects desired for the structure. Thus, the structure should be light, flexible, strong and easily portable. To design a structure with these features aluminum components was mainly used, going of against the material used in the humanoid robots available in the market. Another predefined specification was that the distance between both arms should be adjustable, thus, guidance systems were designed.

At this stage a couple of structures were proposed and designed to met the different configurations initially desired. Figure 3.3 shows a solution that allows to fix the arms in sideways of the torso with possibility to adjust the angle between the base of the arms and the torso, as well as the distance between both arms.

Figure 3.3: 3D model of a solution to the structure that allows to adjust the angle and the distance between the two arms assembled in sideways of the torso.

In this solution, the change of the distance between the two arms is done through the guidance system of the horizontal aluminum profile. The setting of the angle between the arms base and torso is done by the “half moon” shape profile. This solution presents little robustness due to the hinges of the angle adjustment and it is also a heavy solution.

In the natural posture of a human being both arms are oriented vertically. Thus, a solution was designed to try and keep the humanoid structure as close as possible to a human. The proposed solution for the torso structure is presented in Figure 3.4.

Figure 3.4: 3D model of a solution to the structure that allows to adjust the distance between the two arms assembled vertically to the torso.

This solution is lighter than the solution in Figure 3.3 because it does not need an intermediate plate between the arms base and the torso. The robotic arms are assembled with its bases fixed to the smaller section aluminum profiles, which allows to adjust the distance between the arms.

With the aim to choose the solution that best fits to the human gesture imitation workspace a study of it was done using both robotic arms installed in different configurations, based on the previously presented solutions.

3.2.2 Workspace Analysis

With the goal to develop and design a humanoid torso structure similar to a human, a preliminary study of the body parts length relations as well as the system workspace were required. In this study were taken into account the distance between both arms as well as the height of the torso. In accordance with biometrical relations, studied in anthropometry, there is a length estimation of each part of the body as function of the individual height. Figure 3.5 represents a schematic of estimated relations defended by (Drillis et al., 1964).

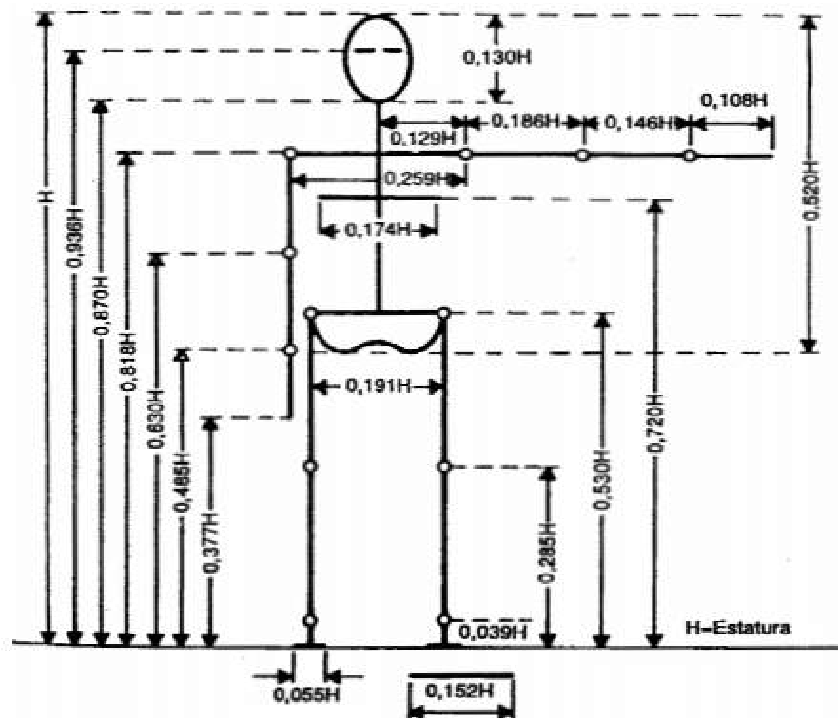


Figure 3.5: Length of body parts as function of body height (Drillis et al., 1964).

Based on specifications of the Cyton Gamma 1500, its length, from the shoulder roll joint to the end-effector, is about 688mm. Therefore, using this value for the arms length, the estimation for the distance between the two shoulders is about 404mm and the torso height is about 450mm.

Taking these measures into account a study of the workspace was made using different mounting configurations of both arms. A simulation was developed using Matlab in order to set the robotic arms orientation in the structure that gives the workspace that best fits to the project situation. In it was represented 2 arms mounted in some different configurations which approximates to the configuration of a human torso. At this point, 3 kinds of configurations were defined :

- Vertically mounted arms, Figure 3.6 a);
- Sideways mounted arms, Figure 3.6 b);
- Sideways mounted arms, with 45 degrees angle between the base and the structure, Figure 3.6 c).

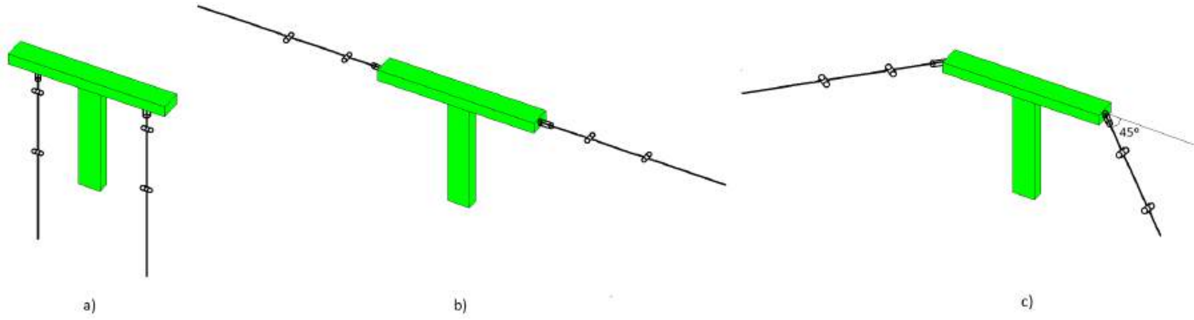


Figure 3.6: Representation of the three simulated configurations in a 3D plot. Green blocks represent the torso, black lines represent the robotic arms and black cylinders are the used joints. a) Vertical configuration, b) Sideways configuration and c) Sideways configuration with an angle of 45 degrees.

Adopting the previous calculated lengths for the height of the torso and the distance between both arms, a simulation of each different mounting configuration workspace in a 3D plot was represented. These simulations were based in iterative cycles that run through joint angles, using just 3 DOFs, form lower to higher limits. The point cloud of every end-effector position was stored and plotted, representing all the possible end-effector poses.

Figure 3.7 displays the workspace of the two arms assembled in a vertical configuration. Table 3.4 outlines the workspace dimensions of the two arms assembled in a vertical configuration.

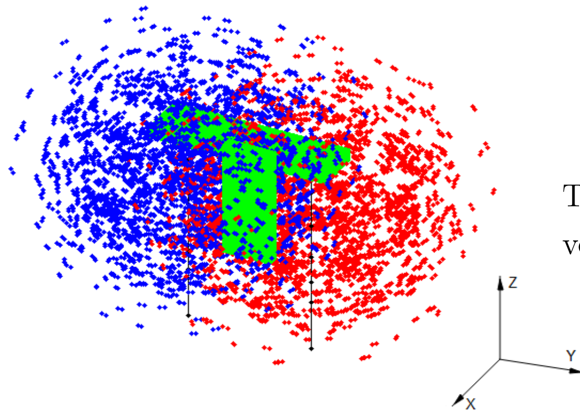


Figure 3.7: Workspace representation of right arm (blue) and left arm (red) assembled to torso (green) in a vertical configuration.

Table 3.4: Workspace specifications for vertical configuration

Axis	Min. [mm]	Max. [mm]
X	-51.1858	51.1858
Y	-71.6950	71.6950
Z	-19.0411	65.6570

It was also tested a simulation with both arms assembled in sideways of the torso. The point cloud of the workspace of this configuration is displayed in the Figure 3.8.The workspace

dimensions of the two arms assembled in a lateral configuration are described in the Table 3.5.

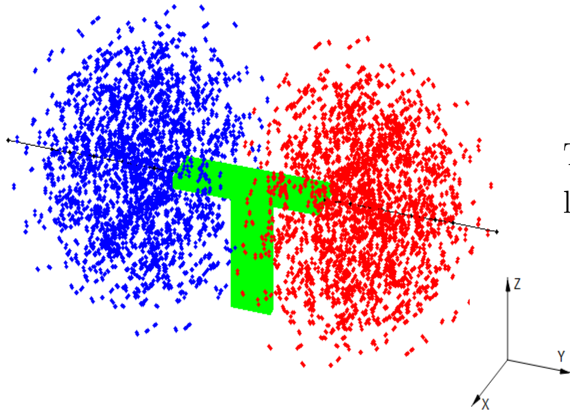


Figure 3.8: Workspace representation of right arm (blue) and left arm (red) assembled to torso (green) in a lateral configuration.

Table 3.5: Workspace specifications for lateral configuration.

Axis	Min. [mm]	Max. [mm]
X	-51.1858	51.1858
Y	-88.0911	88.0911
Z	-9.4950	93.4950

It was also made a simulation with an intermediate angle, 45º. The point cloud that represents its workspace is shown in Figure 3.9. The respective dimensions of the two arms workspace, assembled in lateral configuration with 45º, are described in the Table 3.6.

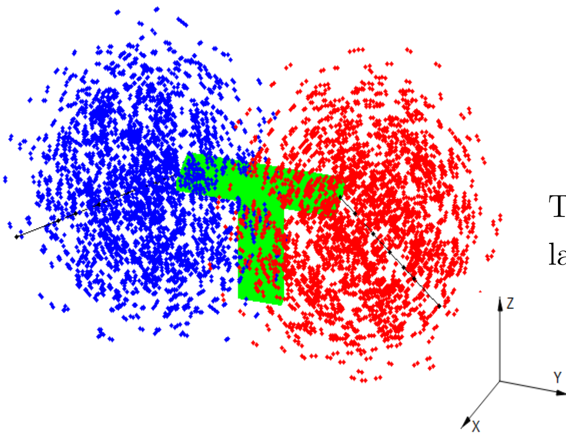


Figure 3.9: Workspace representation of right arm (blue) and left arm (red) assembled to torso (green) in a side mounted configuration with 45º.

Table 3.6: Workspace specifications for lateral configuration with 45º.

Axis	Min. [mm]	Max. [mm]
X	-41.3066	60.9748
Y	-90.0248	90.0248
Z	-9.1858	93.1858

It was defined that the workspace of the robot should be, preferentially, in front of robot torso and as far forward as possible. By the graphical comparison, which gives a visual perspective of the workspace, and by the workspace dimensions analysis of each configuration, it can be concluded that the vertical configuration is the one that best fits the workspace specifications. Furthermore, that configuration has a large volume of workspace where both arms can reach simultaneously. That can be useful if it is intended the robot to perform grasp movements with both end-effectors at the same time.

Given this features, the adopted configuration was the one with the two arms assembled in a vertical configuration. Hereupon it was developed a structure that supports both arms with the dimensions previously mentioned and with variable distance between the arms as well as the Kinect sensor, as illustrated in Figure 3.10.

Figure 3.10: Final hardware setup.

3.3 Software Development Tools

In this section is presented one of many programmatic control interfaces for Cyton Gamma 1500 as well as the package used to acquire the data from the Kinect sensor. Cyton has C++ SDK, TCP commands, LabView or Robot Operating System (ROS) interfaces available. The most used are the C++ SDK, which uses the Actin simulator interface, and ROS. In next subsections are presented some features of the used interface, ROS, and also the OpenNI package used to acquire the human motion data. ROS was chosen for this project because it is

more versatile, scalable and easier to use. Its mechanism allows to create individual processes that communicate with others through messages and each process can be programmed uing different programming languages. Thus, if it is necessary to add new functionality or features to the system it is just necessary to create a new process which subscribes messages from other processes and publishes its owns. OpenNI package was used due to the fact that it allows the integration with NITE middle-ware, which allows skeleton tracking, explained in subsection 3.3.2.

The software was developed and tested in Ubuntu 12.04, under ROS Fuerte version using a PC with Intel Core i5 - 2.6GHz processor. The applications were developed using C++ programming language resorting to ROS libraries and an external library, *dlib*. This library allows to manipulate matrices in a simple way, which were very used during the project. The user can interact and introduce commands through a simple command line interface.

3.3.1 Robot Operating System

With the large increase of the community in research and development of robotic systems it was remarkable the several difficulties developing software applications for robotic systems. This is mostly due to the fact that each robot have a particular communication protocol, each camera has a specific image format and the need that can be acquired data from sensors running in different computers are some of the many difficulties experienced when there is the need to develop robot software. In order to solve or minimize this difficulties Robot Operating System was developed. The official definition of ROS is:

> *ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.*[5]

ROS provides interesting and useful features and services such as hardware abstraction, simple mechanisms of communication between processes, package management, software reuse, easy and rapid testing and language independence. ROS framework was already implemeted in C++, Python and Lisp and has some experimental libraries in Java, Lua and Matlab. Those features made the ROS a powerful tool for robotics software development.

ROS programs and libraries that perform a certain function are grouped into packages. Inside each package are stored source code, libraries, binaries, *manifest.xml* file, where are the declaration of the packages dependencies of other packages and the *CMakeList.txt* file ,which contains instructions for the *CMake* compilation.

[5] *http://wiki.ros.org/ROS/Introduction*

A program is a group of *nodes* that are running at the same time. The communication between nodes are provided by the ROS *master* and it is done by exchanging messages with one another. The primary mechanism that nodes use to communicate is sending *messages*. Messages are published onto topics and a node that wants to share a certain message publishes it on the appropriate *topic* or subscribes the topic that is interested in to receive the message. A single node can subscribe and publish different topics.

Figure 3.11 presents a simple example of message exchanging between three nodes. In it, nodes are represented by ellipses and topics are signaled by rectangles. This example shows three nodes, A, B and C and two topics, A and B. Node A does not subscribe any message and it is publishing messages on topic A. Node B and C are subscribing to messages of topic A. Node B is publishing messages on topic B and node C is subscribing those messages. Each node can be programmed to publish a certain topic at a specific frequency and it can be different in all nodes of the program. This shows that the exchange of complex messages can be easily managed by ROS framework and it is easy to implement.

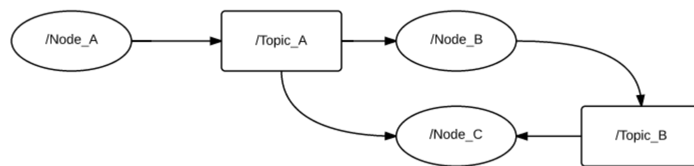


Figure 3.11: Example of three topics exchanging messages.

This is the primary method of communication between nodes but it has some limitation. First it is unidirectional and the messages are published to any node that wants to subscribe it and there is no response. Alternatively, *services* are bi-directional and implements one-to-one communication. Thus, one node sends information to another and waits for response, this method is based in server/client topology as shown in the diagram of the Figure 3.12.

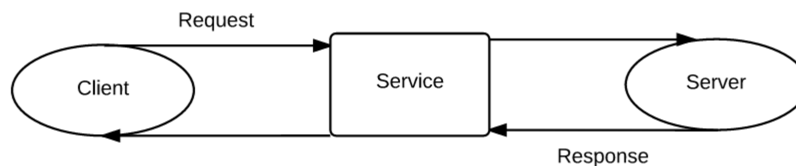


Figure 3.12: Services communication mechanism.

Related to Cyton Gamma, used in this dissertation context, ROS has the ROS-Cyton Module that provides a ROS interface for Energid's actinSE, which is a robotic simulator with brief explanation presented in Appendix B, and Cyton manipulators. This interface allows to perform a direct and real-time control of the Cyton robotic arms in joint space and end-effector modes using *ActionServers* and *ActionClients* protocols. These are similar

to services communications but they use ROS action protocol, which is built on top of ROS messages. It provides a simple API for users to request goals (client) and execute goals (server) via function callbacks, Figure 3.13. In the specific case of controlling Cyton Gamma it allows to send goals that contains the joint commands information, which is composed by:

- position - End-effector coordinates or joint values;
- rate - Joint rates;
- time - Simulation time (used in Actin simulator, briefly described in Appendix B);
- eeindex - End-effector type;
- home - Home flag to move Cyton to home position;
- gripper_value - Gripper joint angle for controlling gripper separately
- gripper_rate - Gripper joint rate

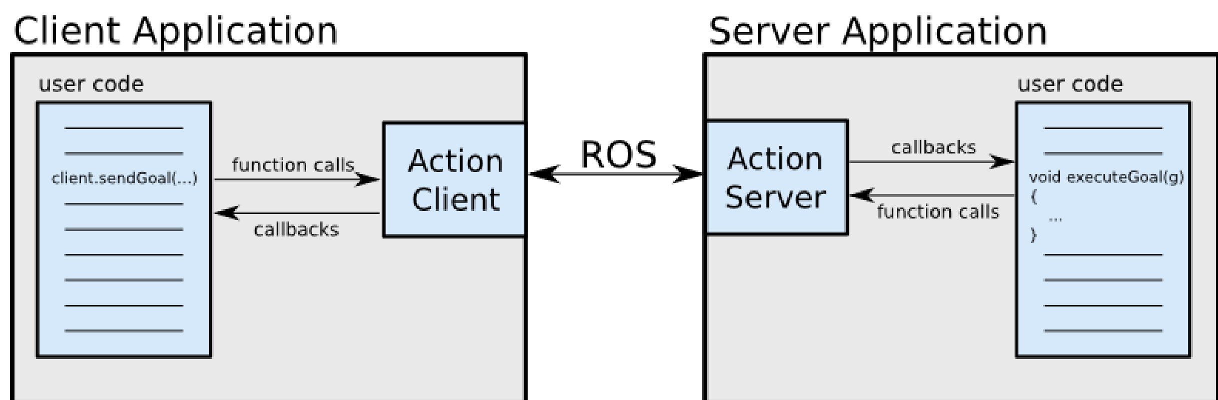


Figure 3.13: Action Server and Action Client interaction in ROS.

Figure 3.14 presents the ROS architecture provided to control Cyton Gamma end-effector in task spade using the 7 DOFs of the arm.

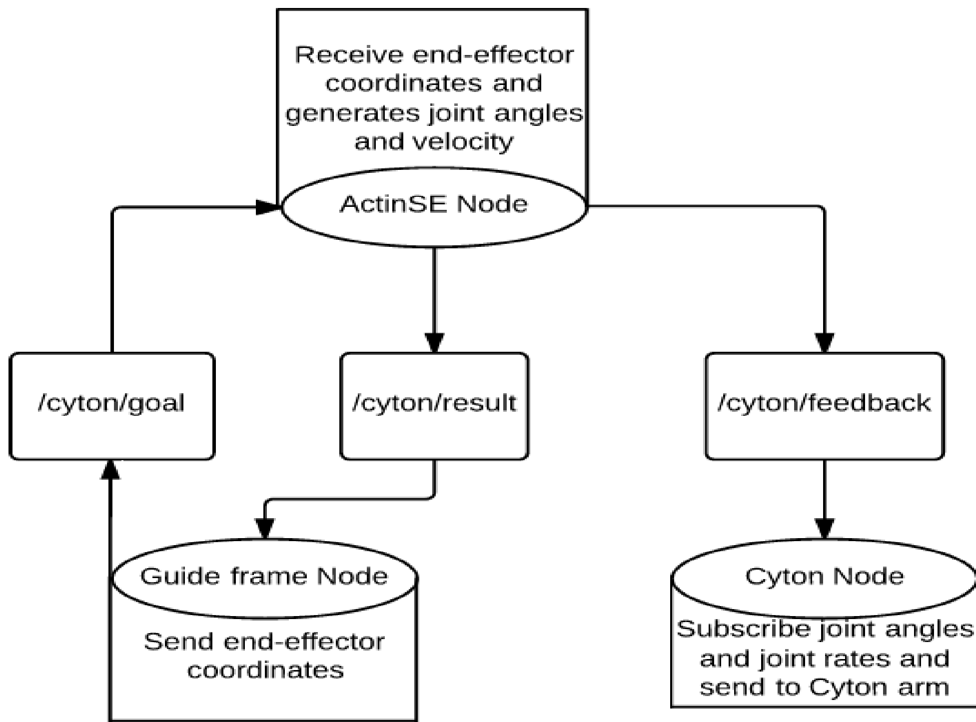


Figure 3.14: End-effector control ROS architecture.

ActinSE node is the server and uses ActinSE Cyton inverse kinematics engine to convert the subscribed end-effector Cartesian coordinates, published by Guide frame node, to joint space angles and velocities. After, ActinSE publishes the result to the */cyton/feedback* topic and the Cyton node subscribes it and sends a command to the robotic arm. It allows to control the arm in operational space resorting to ActinSE IK algorithm that uses all 7 DOFs of the arm.

Another available mode is joint level control of the arm. Figure 3.15 represents the ROS architecture of this mode.

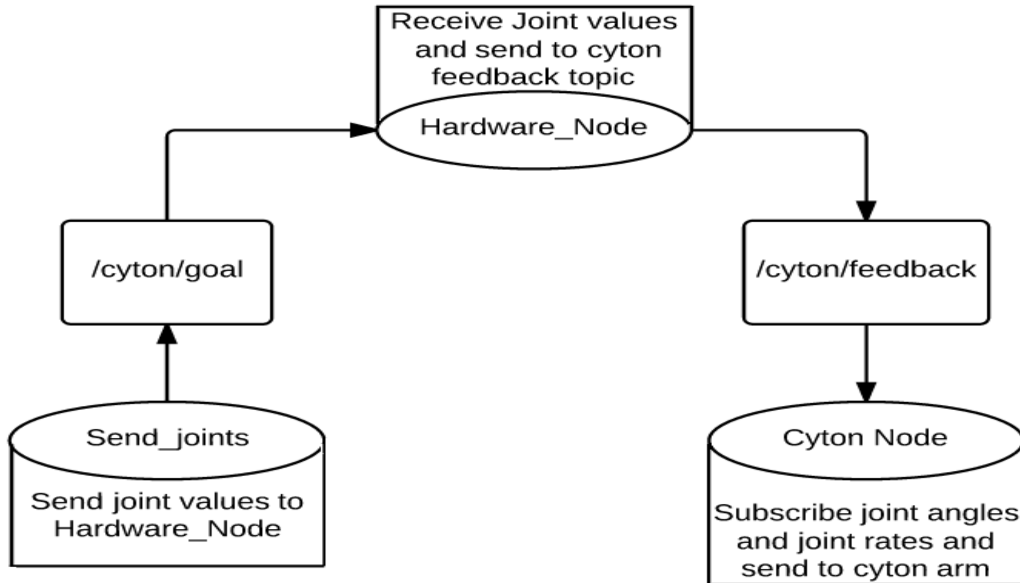


Figure 3.15: Joint level control ROS architecture.

In joint level control, Hardware node is the server and receives joint values from Send joints node. Joint angles and velocities are published to */cyton/feedback* topic, which is subscribed by Cyton node that sends commands to the robotic arm.

It is possible to use the nodes as they are arranged or reuse them for other purposes as well as modify the functionality of each one of them. The communication with the robotic arm was performed resorting to the Cyton Hardware API[6], which has available the hardwareInterface class. It contains the public functions presented in Appendix A.

3.3.2 OpenNI

Open Natural Interaction (OpenNI) is a non-for-profit organization created by Primesense, Willow-Garage, Side-Kick, Asus and Appside in 2010. It has the aim to certify the compatibility of NI devices such as the Kinect sensor. OpenNI API is an abstract layer that provides the interface to the physical devices and middle-ware components (Avancini, 2012). Its architecture is shown in Figure 3.16.

[6]http://www.robai.com/content/docs/HardwareInterface_docs/index.html

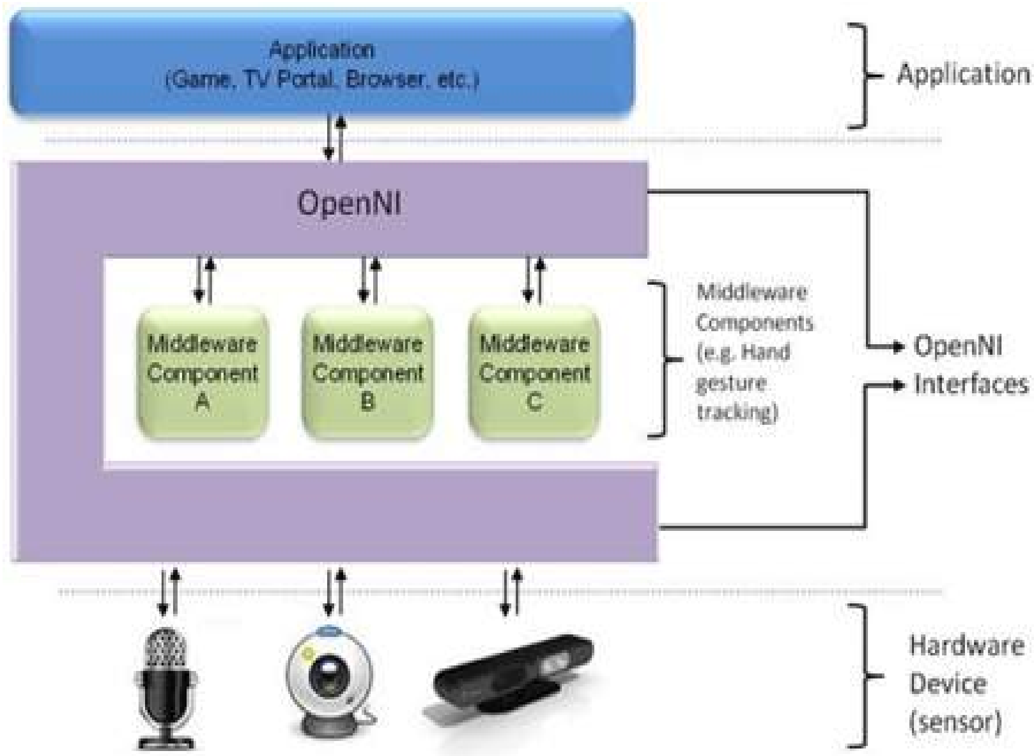


Figure 3.16: OpenNI architecture (Avancini, 2012).

In this dissertation was also used the Natural Interaction Technology for End-user (NITE) middle-ware to perform the skeleton tracking. It acts as the detection engine that is able to detect the user body and offers features like user identification, movement detection and skeleton tracking. Using the NITE algorithms to perform skeleton tracking requires the user to stay in “surrender” pose as indicated in Figure 3.17.

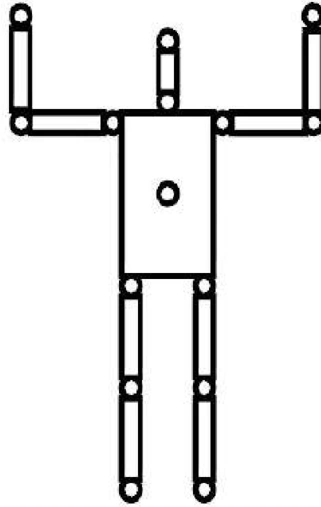


Figure 3.17: NITE calibration pose. The user should stay in front of the sensor with a pose similar to the one presented in the figure (Avancini, 2012).

It was used openni tracker[7] package to get the skeleton from the Kinect sensor using ROS interface. It is an independent node that does not require other nodes to run and broadcasts the OpenNI skeleton frames using transformations (tf). Tf is a tree structure containing the relationship between coordinate frames in time. It allows to transform points, vectors or matrices between two coordinate frames. The human pose is published as a set of tf with the following frame names: head, neck, torso, left_shoulder, left_elbow, left_hand, right_shoulder, right_elbow, right_hand, left_hip, left_knee, left_foot, right_hip, right_knee, right_foot.

[7]http://wiki.ros.org/openni_tracker

Chapter 4

Kinematics and Robot Motion

The main objectives of this chapter are twofold. First, to derive the mathematical model of the 3-DOF manipulator arms in terms of forward, inverse and differential kinematics, and then to present the computational algorithms that generate the reference inputs to the low-level motion control system. These algorithms comprise both trajectory planning techniques and inverse kinematics solutions. Before addressing the level of functionality that is required for gesture imitation, a more detailed explanation is needed in what concerns the internal control units that are part of the robot's actuators. The Dynamixel servomotors provide control actions in a typical position mode using the actuator's built-in microcontroller. Each servo receives as input the desired angular position and, internally, performs the trajectory planning in the joint space between the current and target angular positions, using the velocity parameter to derive the movement runtime. To accomplish all the necessary operations, three control modes were developed as independent ROS modules that can be selected according the user's intention. This control interface provides the following modes:

1. **Joint control mode** in which a vector of desired joint angles are specified as the input to be sent to the robot. The joint-space trajectory planning implemented in the actuator's control unit generates the time sequence of target values for the low-level controller.

2. **Point-to-point control mode** in which the desired end-effector position is specified in task-space as input and an inverse kinematic algorithm, running in the main computer, calculates the desired joint angles to be sent to the robot. The resulting end-effector path is not predictable due to the nonlinear mapping between joint and Cartesian spaces.

3. **Continuous control mode** in which the desired end-effector trajectory is specified in task-space as input and an inverse kinematics algorithm, running in the main computer, calculates in real-time the joint velocities to be sent to the robot.

4.1 Kinematic Analysis

To control a robotic manipulator is essential to know its kinematics. In this sub-section are presented the Direct Kinematics (DK), Inverse Kinematics (IK) and Differential Kinematics of the 3 DOFs robotic arm. DK is used to obtain the operational position of the end-effector related to the coordinate frame on the base of the robotic arm by knowing its joint angles. It is very helpful to apply feedback control and also to know where the position of the robotic arm in the operational space. On the other hand, IK is used to know the configuration and joint angles that the robotic arm needs to take to move to a specific position, regardless the path. Finally, the Differential Kinematics is used when it is important to know the end-effector trajectory. It implies controlling its path but also each joint velocity during the movement.

4.1.1 Direct Kinematics

The DK of a robotic arm is the determination of end-effector position and orientation as a function of the joint angles. To simplify the kinematics and the control of the robotic arm, in the dissertation context, it was constrained some joint variables, and considered only 3 joints.

In order to study robotic manipulators, Denavit and Hartenberg developed a convention to obtain the transformation matrix that represents the end-effector related to the global reference (Sciavicco and Siciliano, 1996).

Figure 4.1 represents the coordinate systems of each joint to the 3 DOFs simplification of Cyton arm, and in the Table 4.1 are the respective *Denavit-Hartenberg* parameters. From the Denavit parameters it is possible to determinate the Cartesian position of end-effector in terms of joint angles ($\theta 1$,$\theta 2$ and $\theta 3$).

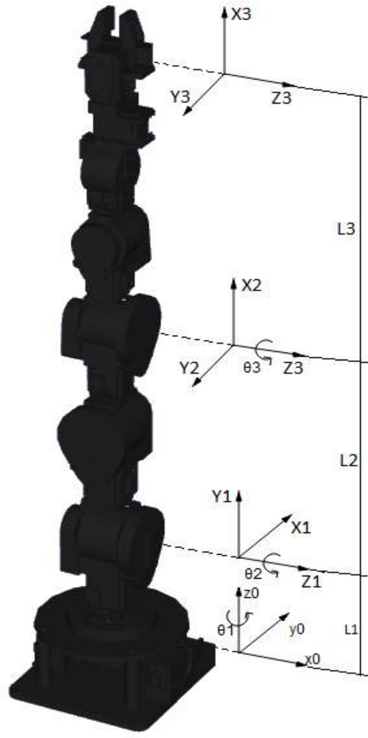


Figure 4.1: Coordinate frames of 3 DOF Cyton arm.

Table 4.1: Denavit parameters of 3 DOFs Cyton arm

i	θi	Li	di	αi
1	90º+θ1	0	L1	90º
2	90º+θ2	L2	0	0
3	θ3	L3	0	0

It can be done by the successive multiplication of the transformation matrices, each one corresponding to the respective link. The transformation matrix applied to each link is given by the following expression4.1.

$$A_i = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & L_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & L_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Multiplying all the transformation matrices results the expressions for Cartesian coordinates of the end-effector in order to joint angles, which are shown in the equations 4.2, 4.3 and 4.4.

$$X = sin(\theta_1) \cdot (L_2 \cdot sin(\theta_2) + L_3 sin(\theta_2 + \theta_3)) \tag{4.2}$$

$$Y = -cos(\theta_1) \cdot (L_2 \cdot sin(\theta_2) + L_3 sin(\theta_2 + \theta_3)) \tag{4.3}$$

$$Z = L_1 + L_2 \cdot cos(\theta_2) + L_3 \cdot cos(\theta_2 + \theta_3) \tag{4.4}$$

4.1.2 Inverse Kinematics

The IK consists of the determination of the joint variables corresponding to a given end-effector position and orientation. The IK may has multiple solutions or it can even has no solution, depending on the complexity and number of DOFs of the manipulator. It is one of the reasons of the simplification to 3 DOF that was made.

Usually, the IK is determined by manipulating the DK expression. The expression of $\theta 1$, presented in the equation 4.6, can be obtained by dividing x by y as follows:

$$\frac{X}{Y} = \frac{sin(\theta_1) \cdot (L_2 \cdot sin(\theta_2) + L_3 sin(\theta_2 + \theta_3))}{-cos(\theta_1) \cdot (L_2 \cdot sin(\theta_2) + L_3 sin(\theta_2 + \theta_3))} \tag{4.5}$$

$$\theta_1 = arctan\left(\frac{X}{-Y}\right) \tag{4.6}$$

The $\theta 2$ expression can be obtained by manipulating x and z expressions.

$$\begin{cases} \frac{X}{sin(\theta_1)} = & L_2 sin(\theta_2) + L_3 sin(\theta_2 + \theta_3) \\ Z - L_1 = & L_2 cos(\theta_2) + L_3 cos(\theta_2 + \theta_3) \end{cases}$$

From the manipulation of the results:

$$2L_2 \cdot \frac{Y}{sin(\theta_1)} \cdot sin(\theta_2) + 2L_2 \cdot (Z - L_1) \cdot cos(\theta_2) = -L_3^2 + L_2^2 + \frac{X^2}{sin(\theta_2)^2} + (Z - L_1)^2 \tag{4.7}$$

That result is in the form of $K_1 \cdot sin(\theta) + K_2 cos(\theta) = K_3$ that has a analytical know solution.

But when $\theta 1$ tends to zero, $\frac{x}{sin(\theta_1)}$ tends to infinite, so it was necessary to use the y expression instead of the y in this situation. Thus, K_2 remains the same because it does not depend on $\theta 1$. K_1 , $K_2$and K_3 takes the equations 4.8, 4.9 and 4.10, respectively.

$$K_1 = 2L_2 \cdot \frac{Y}{sin(\theta_1)} \vee 2L_2 \cdot \frac{Y}{-cos(\theta_1)} \tag{4.8}$$

$$K_2 = 2L_2 \cdot (Z - L_1) \tag{4.9}$$

$$K_3 = -L_3^2 + L_2^2 + \frac{X^2}{sin(\theta_2)^2} + (Z - L_1)^2 \vee -L_3^2 + L_2^2 + \frac{Y^2}{cos(\theta_1)^2} + (Z - L_1)^2 \tag{4.10}$$

The solution for $\theta2$ is given by the equation 4.11.

$$\theta_2 = atan\left(\frac{K_1}{K_2}\right) \pm atan\left(\frac{\sqrt{K_1^2 + K_2^2 - K_3^2}}{K_3}\right) \tag{4.11}$$

To obtain $\theta3$ the same method can be used, but squaring both x and z and y and z expressions. Thus, K_1, K_2 and K_3 are defined in equations 4.12, 4.13 and 4.14, respectively.

$$K_1 = 0 \tag{4.12}$$

$$K_2 = 2L_2L_3 \tag{4.13}$$

$$K_3 = \left(\frac{X}{sin(\theta_1)}\right)^2 + (Z - L_1)^2 - L_2^2 + L_3^2 \vee \left(\frac{Y}{-cos(\theta_1)}\right)^2 + (Z - L_1)^2 - L_2^2 + L_3^2 \tag{4.14}$$

The expression of $\theta3$ is given by the equation 4.15.

$$\theta_3 = \pm arctan\left(\frac{\sqrt{K_2^2 - K_3^2}}{K_3}\right) \tag{4.15}$$

4.1.3 Differential Kinematics

The differential kinematics aims to describe the arm movement between two configurations. It gives the relationship between the joint velocities and the corresponding end-effector linear velocity. This relation is obtained by the arm Jacobian matrix. It can be defined by the equation 4.16.

$$d\vec{r} = J \cdot d\vec{q} \tag{4.16}$$

where $d\vec{r}$ is a vector with Cartesian position of end-effector, $d\vec{q}$is a vector with joint angles and J is the Jacobian matrix, given by equation 4.17.

$$J = \begin{bmatrix} \frac{\partial r1}{\partial q1} & \frac{\partial r1}{\partial q2} & \cdots & \frac{\partial r1}{\partial qn} \\ \frac{\partial r2}{\partial q1} & \frac{\partial r2}{\partial q2} & \cdots & \frac{\partial r2}{\partial qn} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial rm}{\partial q1} & \frac{\partial rm}{\partial q2} & \cdots & \frac{\partial rm}{\partial qn} \end{bmatrix} \tag{4.17}$$

where m represents the number of Cartesian position variables and n the number of joints.

For the 3 DOF simplification there are 3 joints and the Cartesian variables are x, y and z, despising the orientation. With these considerations, the Jacobian of the arm is presented in equation 4.18.

$$J = \begin{bmatrix} c_1 \cdot (L_2 s_2 + L_3 s_{23}) & s_1 \cdot (L_2 c_2 + L_3 c_{23}) & L_3 s_1 \cdot c_{23} \\ s_1 \cdot (L_2 s_2 + L_3 s_{23}) & -c_1 \cdot (L_2 c_2 + L_3 c_{23}) & -L_3 c_1 \cdot c_{23} \\ 0 & -L_2 s_2 - L_3 s_{23} & -L_3 s_{23} \end{bmatrix} \tag{4.18}$$

The IK can be derived by the inverse matrix of Jacobian and the expression that represents the joint velocities as function of end-effector Cartesian coordinates is shown in equation 4.19.

$$d\vec{q} = J^{-1} \cdot d\vec{r} \tag{4.19}$$

The calculation of an inverse of a matrix can be computationally slow, and that could be crucial when all parts of the arm control are running at the same time. So this matrix was calculated analytically and it is expressed by the equation 4.20.

$$J^{-1} = \begin{bmatrix} \frac{c_1}{L_2 s_2 + L_3 s_{23}} & \frac{s_1}{L_2 s_2 + L_3 s_{23}} & 0 \\ \frac{-s_1 \cdot s_{23}}{-L_2 \cdot s_3} & \frac{c_1 \cdot s_{23}}{-L_2 \cdot s_3} & \frac{-c_{23}}{-L_2 \cdot s_3} \\ \frac{s_1 \cdot (L_2 s_2 + L_3 s_{23})}{-L_2 L_3 s_3} & \frac{-c_1 \cdot (L_2 s_2 + L_3 s_{23})}{-L_2 L_3 s_3} & \frac{L_2 c_2 + L_3 c_{23}}{-L_2 s_3} \end{bmatrix} \tag{4.20}$$

4.1.4 Kinematics Control for Gesture Imitation

Manipulation tasks are usually specified in terms of the end-effector trajectories defined in the Cartesian space (or task-space), while the robot arms are actuated at the joint level. Therefore, inverse kinematics algorithms are required to map from the task-space trajectory into a suitable joint-space trajectory. An effective approach to the motion control problem is

the so-called kinematic control based on an inverse kinematics transformation which feeds the reference values corresponding to an assigned end-effector trajectory to the joint servos. For a given trajectory in task space $r(t)$, the kinematics control problem can be formulated as to find a joint space trajectory $q(t)$ such that $f(q(t)) = r(t)$ is satisfied. Most of the approaches proposed in the literature Sciavicco and Siciliano (1996) are derived at the velocity level through the use of an online control solution based on the manipulator's Jacobian matrix (Jacobian-based techniques), as follows:

$$\dot{q} = T(q) \cdot \dot{r} \tag{4.21}$$

where T is a suitable control (transformation) matrix based on the Jacobian matrix. The simpler solution is to use the inverse of the Jacobian matrix:

$$\dot{q} = J^{-1}(q) \cdot \dot{r} \tag{4.22}$$

This solution is attractive since a close-form solution is available avoiding numerical computations. Furthermore, it is well-suited for the practical implementation of on-line gesture imitation in which the end-effector trajectory is continuously modified based on sensory feedback information provided by the Kinect sensor. However, using this formulation for gesture imitation is more challenging that it may appear at first glance due to two main problems. First, kinematic singularities are not avoided: singularities occur when the matrix J, at some configuration q, loses rank (one or more degrees-of-freedom). In this case, the manipulator loses its ability to move along some direction of the task space. In order to overcome this drawback, a solution is to use a damped least-square inverse of the Jacobian matrix (Sciavicco and Siciliano, 1996), (Buss, 2009) in the form of $J^* = J^T(JJ^T + \lambda I)^{-1}$ that is non-singular in the whole workspace. This modified Jacobian provides an approximate inverse kinematics solution that can be used in the neighborhood of singular configurations by selecting a suitable value for the damping factor λ. This specific solution will be discussed in more detail in Chapter 5 when addressing the problem of singularity avoidance.

Secondly, it must be emphasized that the solution 4.22 is open-loop in what concerns the task-space, causing unavoidable drifts during the task execution. Solutions to overcome the problem are based on the use of feedback corrections: close-loop inverse kinematics (CLIK) algorithms. A closed-loop version of the previous solution can be obtained if the task space vector $\dot{r}$is replaced by $\dot{r} = \dot{r}^d + K \cdot e$, where $e = r^d - r$ denotes the error between the desired and the actual task trajectories (the actual trajectory can be computed from the real robot joint sensor measurements via direct kinematics algorithm) and K is a positive definite (diagonal) matrix that shapes the error convergence (Sciavicco and Siciliano, 1996), (Buss, 2009). In addition, if a computationally less intensive solution is desirable, a solution based on the transpose of the Jacobian can be devised as follows:

$$\dot{q} = \alpha J^T \cdot (K \cdot e) \tag{4.23}$$

An inherent advantage of this solution is that it may avoid the typical numerical instabilities which occur at kinematic singularities.

4.2 Point-to-Point Control Mode: Implementation and Evaluation

The position control, or point-to-point control, is usually used when it is necessary to move the end-effector to a certain position but the intermediate points of the path are not taken into account. This kind of control is usually used when the goal is to position the end-effector in a certain position regardless the path. The position control was implemented independently so the user can choose the desired Cartesian position and the end-effector will move to that position using a synchronous algorithm so that all joints start and finish the movement at the same time, based on equation 4.24.

$$\dot{Q} = \frac{Q_f - Q_i}{T_{exec}} \tag{4.24}$$

where, $\dot{Q}$represents the joint velocity, $Q_i$and Q_f are the initial and desired joint angles, respectively and T_{exec} is the desired execution time.

Joint velocities are given by the difference of joint value in the final position and initial position divided by the maximum execution time. The joint values of the final position were given by the inverse kinematics of the robotic arm knowing the operation space position introduced by the user. It was also implemented a function that verifies if the introduced position is inside or not the workspace volume. During the experiments it was noticed that the velocities sent to the robotic arm were not the velocities actually performed. The velocity command needed to be adjusted and the scale factor was verified empirically.A simple test was made which consists of sending each joint from a specific angle to another, using a relatively long range, and measuring the execution time using a chronometer. It was a low accurate method of measure but it reveled to be enough to have good results. Thus, it was possible to calculate the velocity that each joint assumed and plot a graph with those values, as shown in Figure 4.2. The collected points were approximated by a linear trend line. With it is possible to define the line equation that was used to rectify the input velocities. It was noticed that the obtained value from the approximated line for the intersection with the Y-axis was not correct since when the input velocity was inputted as zero the arms still moved. It was used the value obtained empirically when the input velocity was zero and the arm remained still. Thus, the equations relative to joint 1, 2 and 3 are presented in equations 4.25, 4.26 and 4.27 , respectively.

$$dq_{1,input} = 0.6529 \cdot dq_{1,real} + 0.0086 \tag{4.25}$$

$$dq_{2,input} = 0.6360 \cdot dq_{2,real} + 0.0084 \tag{4.26}$$

$$dq_{3,input} = 0.7060 \cdot dq_{3,real} + 0.0085 \tag{4.27}$$

Using these equations it was possible to have a better control of the arm since it was important to guarantee the velocities imposed to each joint.

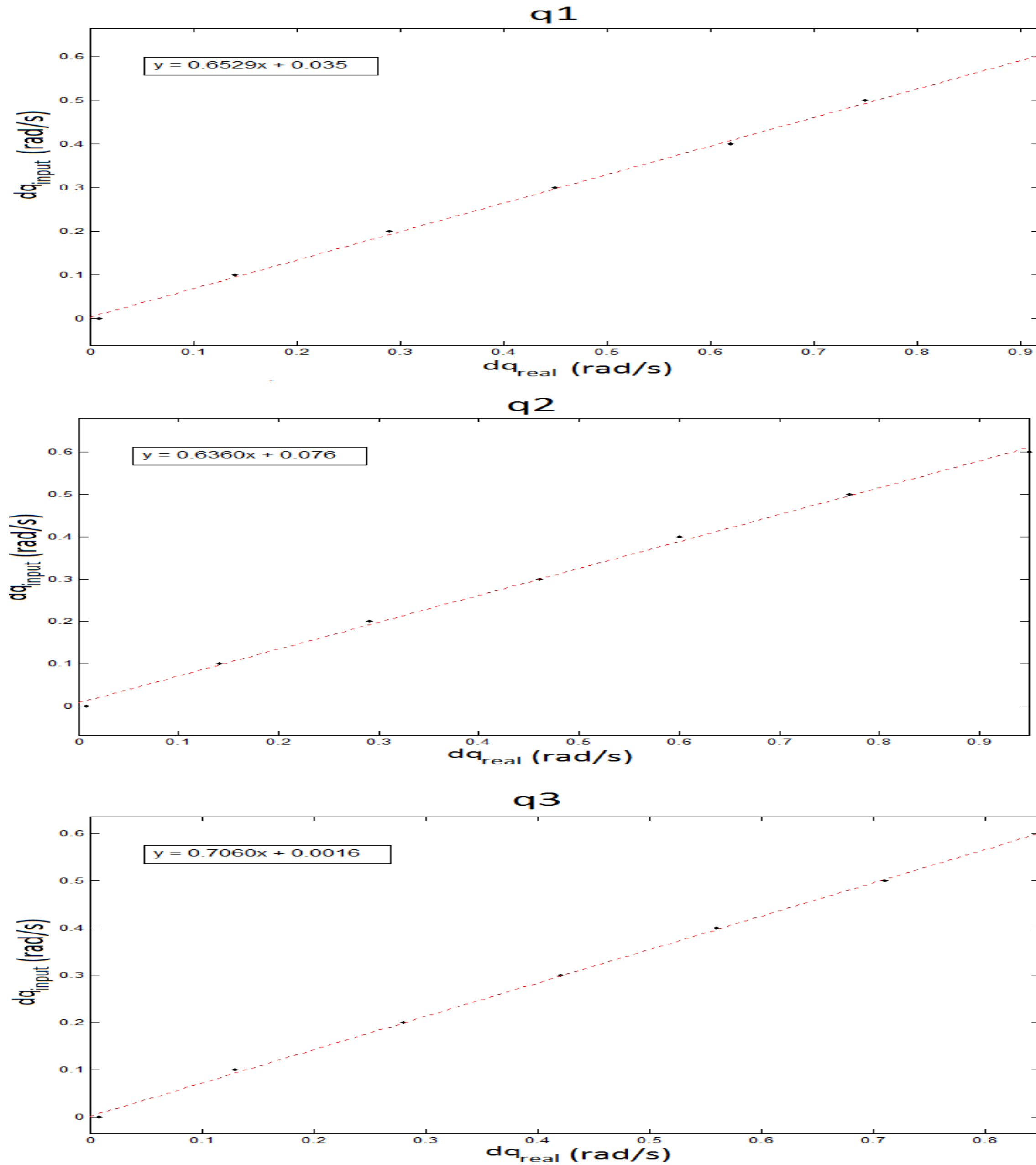


Figure 4.2: Curve of relation between input and real joint velocities. Joint 1 (top), joint 2 (middle) and joint 3 (bottom).

The block diagram of the position control implementation is presented in Figure 4.3.

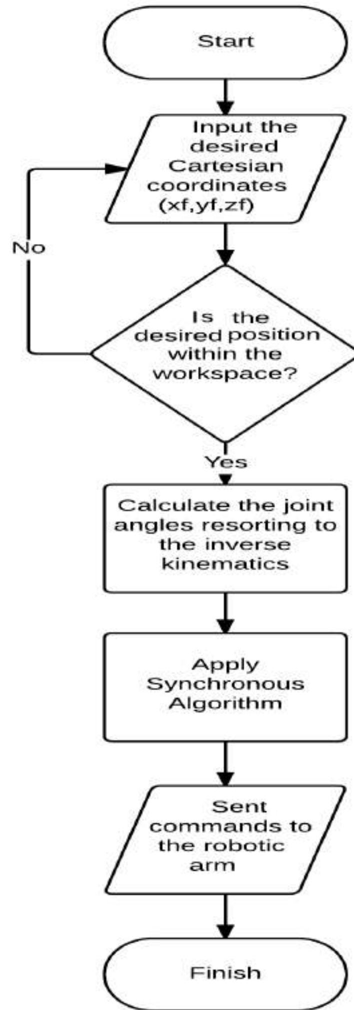


Figure 4.3: Joint space control flowchart.

First, the user introduces the desired Cartesian coordinates of the end-effector through a console interface. Secondly it is verified if they are within the workspace of the robotic arm. If not, it is asked again to input the new coordinates. If the coordinates are within the workspace volume it is applied the inverse kinematics to the introduced position. Then it is applied the synchronous algorithm to guarantee that all joints start and finish at same time and the commands are send to the robotic arm.

The implementation of the position control algorithm was based on the provided nodes and functions available to control the Cyton Gamma 1500, hardware_node and move_hardware nodes. The first acts as an Action Server application, which receives a goal containing the desired command sent by an Action Client application. The second node subscribes to the topic published by the Server and as the function of sending the command

to the robotic arm. In addition it was implemented a function to this node responsible to get the joint states from the robotic arm and publish them.

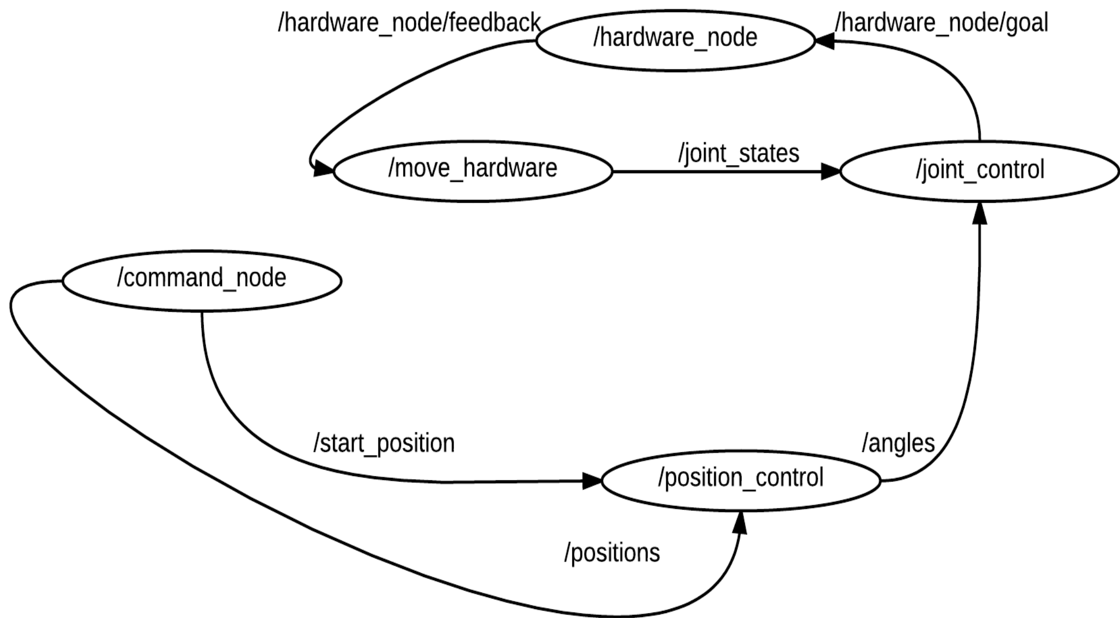


Figure 4.4: ROS architecture of joint control algorithm.

In terms of ROS processes (nodes), the algorithm is divided into five different ones. The interface with user is all done in the command node, where it's possible to input the desired Cartesian position of the end-effector. When the positions are inserted the command node will publish them in the /positions topic and the position control process will subscribe to it. This node receives the positions and applies the inverse kinematics algorithm. If the position is within the robotic arm workspace this node publishes the joint angles to the /angles topic. The joint control node is waiting for the joint angles values and when it receives them it will apply the synchronous algorithm and the joint angles and synchronized velocities are published to the /hardware_node/goal topic. The hardware node is responsible to receive the goal and with joint values and publish them to /feedback topic. The move hardware node is subscribing this topic and when receives new values it sends them to the robotic arm. In order to control the processes some control messages were used, published in the /set_home and /start_position with the objective to define when the process starts. This control mechanism was used in all processes implemented along the dissertation.

The algorithm was tested using one of the robotic arms to ensure that the control is correct and to check if the synchronous control works. In the Figure 4.5 four graphics are presented that represent the behavior of each joint over time, applying and not the synchronism of

joint velocities, as well as the velocity profiles of each joint for the same initial and desired positions.

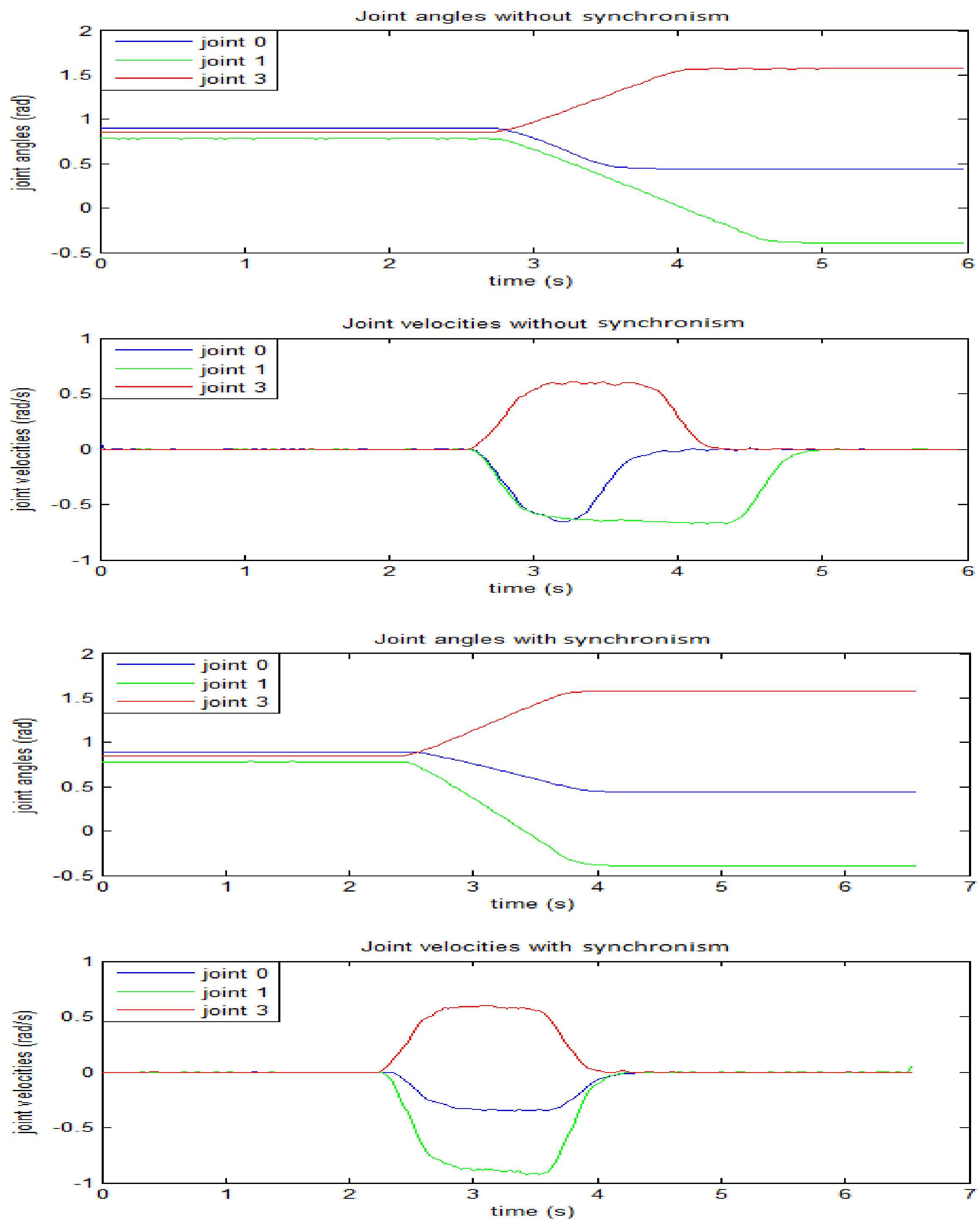


Figure 4.5: Graphics of the behavior of joint angles over time (top) and velocity profiles (bottom).

Observing Figure 4.5 it is possible to see that, when the synchronism to the joint velocities was not applied, the joint movement did not finish at the same time. On the other hand, when the synchronism to the joint velocities was applied, the joint movement finished at the same time. It is also evident the velocity profile applied by the servomotors as a trapezoidal profile.

4.3 Continuous Control Mode: Implementation

The main interest of a manipulator or, in this case, two robotic arms, is the capability to perform a movement from an initial to a final posture in the operational space. It is necessary to implement algorithms that take into account the end-effector position, velocity and/or acceleration. These algorithms are known by trajectory planning and aim to define the behavior of the end-effector, not only in terms of the path that the end-effector has to follow but also define suitably trajectories. Velocity control has the advantage to control the end effector trajectory during time, which was not possible using position control.

The aim of this section was to create a sub-program to test the operational space control using Cyton Gamma 1500 in order to study its performance using a linear trajectory planning to move from the starting to the goal positions in the operational space. After the study of the relationship between operational space and joint space either in geometric (direct and inverse kinematics) and kinetic (differential kinematics), it was necessary to proceed to the trajectory planning. It includes a set of studies and methods that defines the velocities behavior of each joint, in order to meet the movement goal. When the end-effector movement is as simple as moving from one point to another no matter what path it does, it's called point-to-point planning. On the other hand, when it is necessary for the end-effector to execute a well-defined path in the operation space, obeying to precise temporal criteria, it is called path motion planning.

In this trajectory planning, the end-effector follows a straight line from the initial position to the final one. This planning defines the evolution of each Cartesian coordinate of the end-effector through time so that the position, velocity and even acceleration can be verified. The movement should start at time instant ti and finish at tf starting from initial position Xi, Yi and Zi to the final position Xf, Yf and Zf. In the simplest case there is a start and finish position and the start and finish velocities are null. The simple way to do that is to use a polynomial function of t. Thus, the velocity expression is a second order polynomial equation because it has two roots and, consequently, the position expression is a third order equation.

In line with this, the implementation of the continuous control mode would imply the numerical integration of the joint velocities and the accurate knowledge of current joint angles. The strategy adopted in this work was to implement the more natural velocity control mode. In this sense, the algorithm devised can be implemented in three steps: First, the direction of rotation is obtained from the signal of the desired angular velocity. Secondly, the angular position is set to the limits values considering that direction of rotation. Thirdly, the angular

velocity is set to the target value. It should be noted that the second step is only executed when a change of direction occurs.

4.3.1 Inverse Jacobian Solution

As seen before, the inverse kinematics of a manipulator can be difficult to determine and it only gives the relation between end-effector position and the joint angles. The arm Jacobian can be useful, among other things, to determine the inverse kinematics algorithms. In the following sections two algorithms to implement the IK using the Jacobian matrix are presented.

As seen before, in section 4.1.3, the Jacobian matrix gives a relation between the joint velocity space and the operational velocity space. The equation 4.28 shows this relation.

$$d\vec{q} = J^{-1}(q) \cdot d\vec{r} \tag{4.28}$$

The Jacobian matrix depends on the current configuration of the arm, so as the initial posture q0 is known, joint positions can be estimated resorting to a simple technique based on the Euler integration method. If the integration interval is given by Δt and the joint positions and velocities at time t are known, the joint positions at time t+Δt are given by the expression presented in equation 4.29.

$$q(t + \triangle t) = q(t) + \dot{q}(t) \cdot \triangle t \tag{4.29}$$

With this technique, a first approach of IK using differential kinematics can be applied. The algorithm used to apply the IK is described in Figure 4.6.

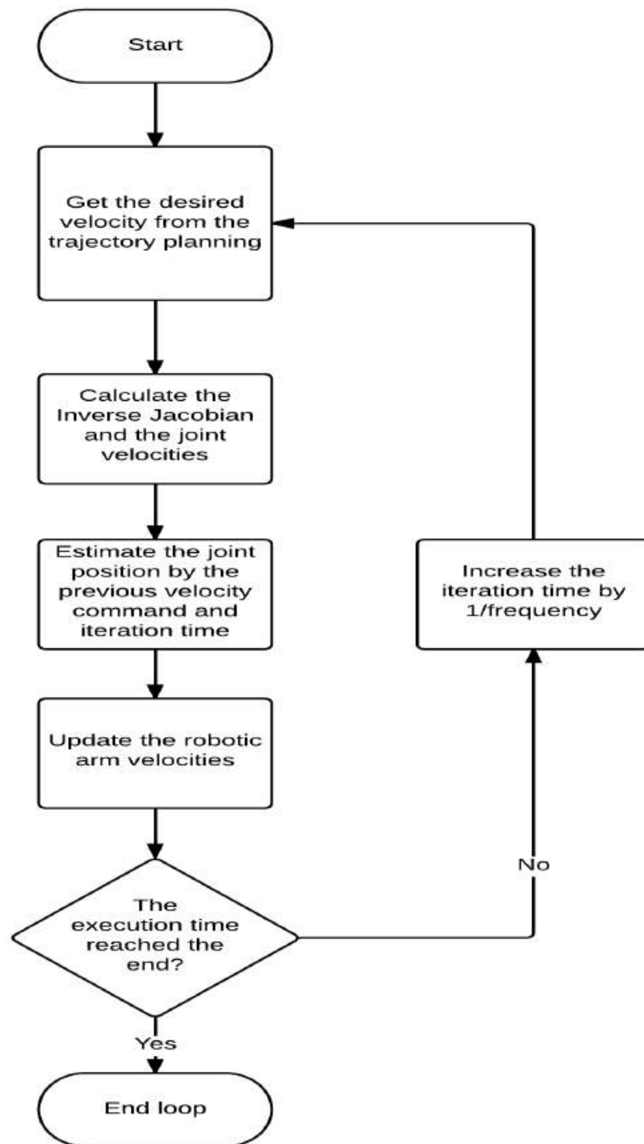


Figure 4.6: Inverse kinematics algorithm using Jacobian inverse with estimated joint positions.

Regarding to ROS implementation, Figure 4.7, two individual processes were developed in addiction to the existing ones. One of those had the objective to perform the trajectory planning for a linear movement. It had as inputs the pretended execution time as well as the desired Cartesian positions. It calculates the trajectory imposing null initial and final velocities and publishes both calculated position and velocity. The velocity control node is responsible to convert the desired velocity in the operational space in joint space velocities and send them to the process responsible to control the robotic arm.

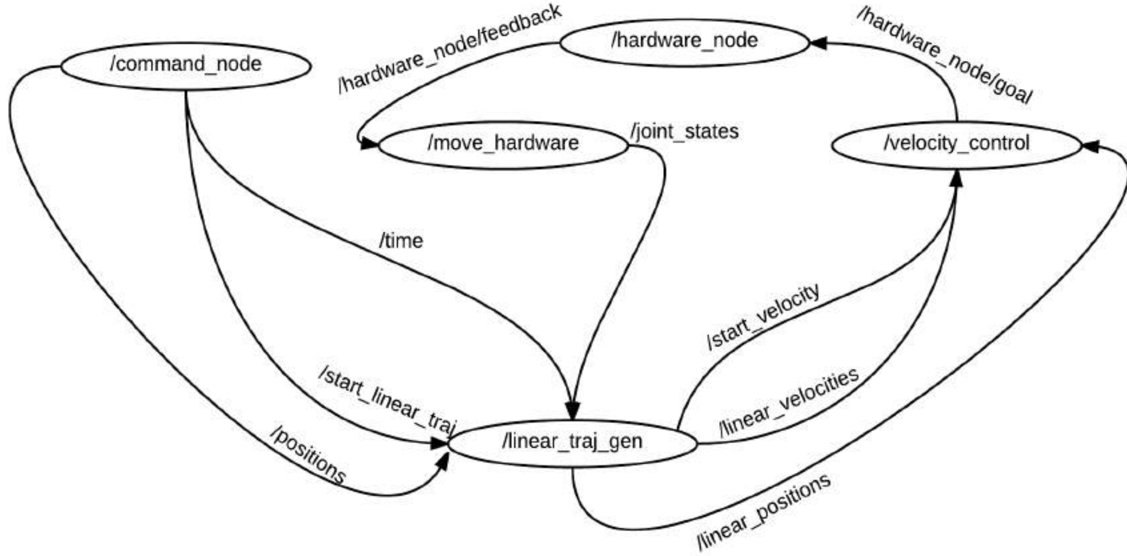


Figure 4.7: Velocity control ROS-based implementation.

4.3.2 Closed-Loop Inverse Kinematics

To guaranty that the defined trajectory is followed with minimal error it was necessary to apply a control algorithm, first using the Jacobian inverse. Taking in account the operational space error e between the desired $\vec{r_d}$ and the real $\vec{r}$ positions of the end-effector, given by the expression 4.30 .

$$e = \vec{r_d} - \vec{r} \tag{4.30}$$

According to (Sciavicco and Siciliano, 1996), considering a square and non singular Jacobian matrix, the relation between the joint velocity space and the operational velocity space can be described as shown in the equation 4.31, considering the error measure e.

$$\dot{q} = J^{-1}(q) \cdot (\dot{r} + K \cdot e) \tag{4.31}$$

The error converges to zero depending on the K and the sampling time. The block diagram of the inverse kinematics algorithm is presented in the Figure 4.8.

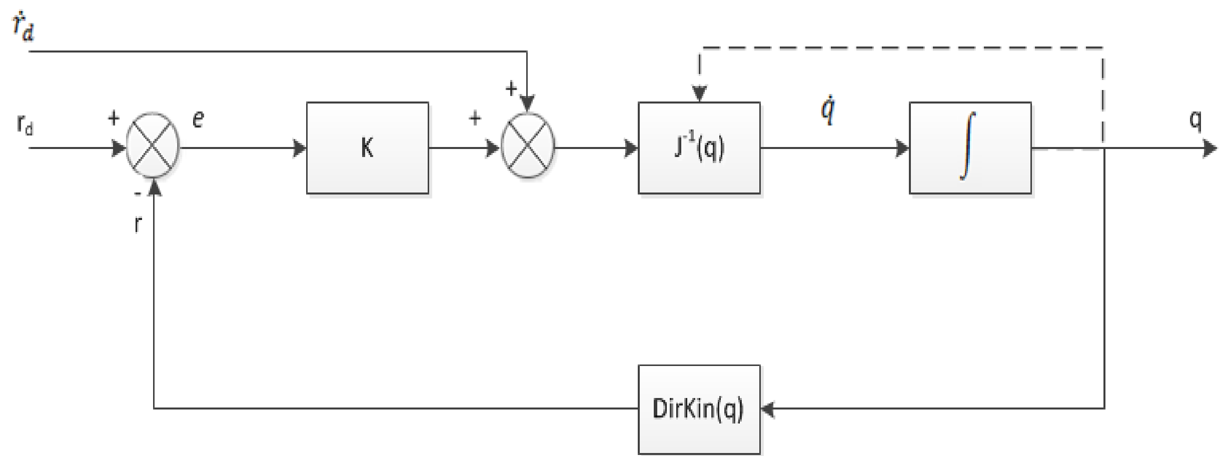


Figure 4.8: Block diagram of the inverse kinematics algorithm with Jacobian inverse (Sciavicco and Siciliano, 1996).

The DirKin refers to the direct kinematics of the arm and is used to calculate the error between the real and the desired positions. This algorithm enables the end-effector to follow the desired trajectory with more precision than the previous algorithms and is described in the Figure 4.9. This algorithm can be computationally slow due to the use of the Jacobian inverse, which could require some computational time. This problem was bypassed using the analytical Jacobian inverse, so it is not necessary to calculate it in real time.

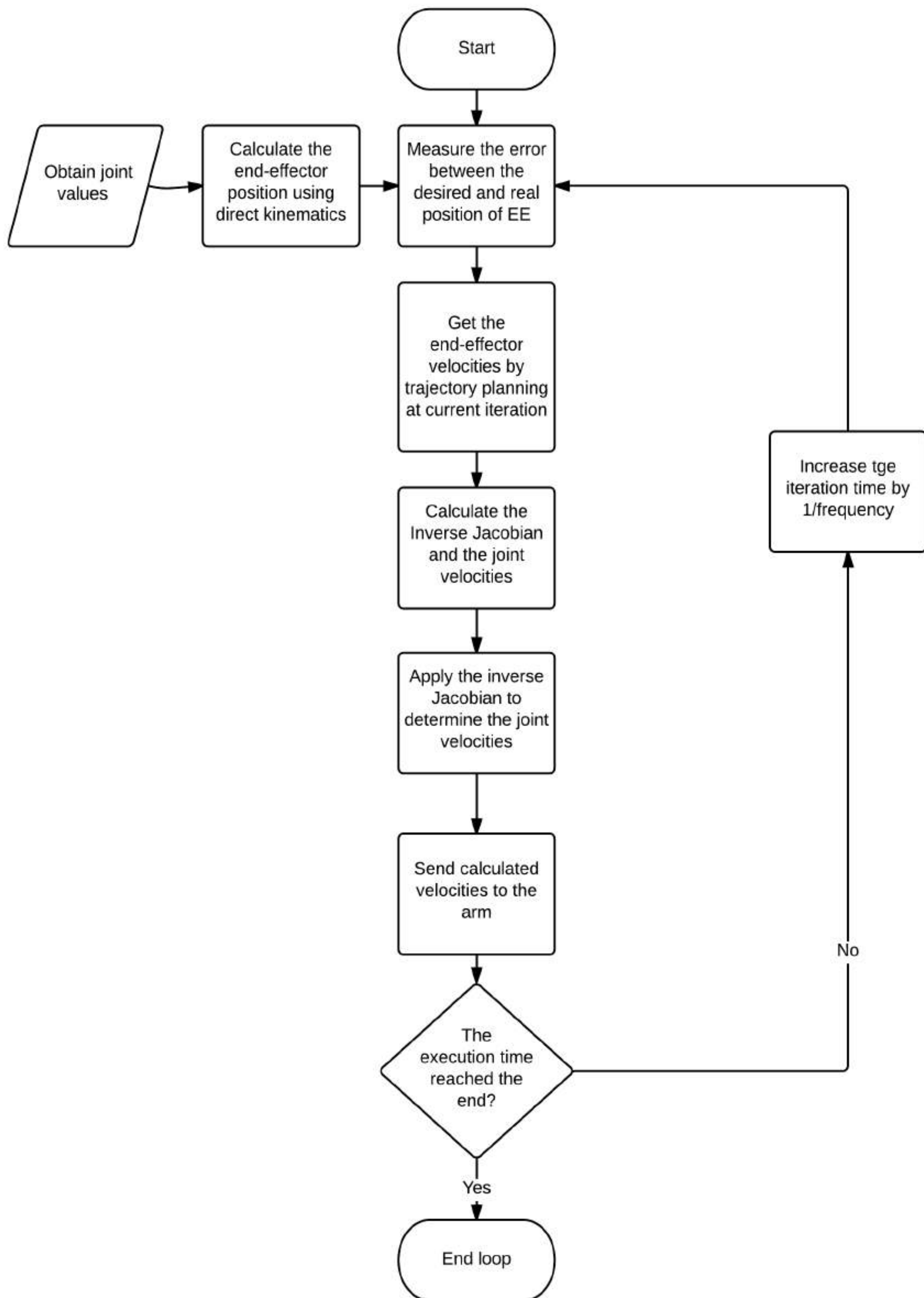


Figure 4.9: Inverse kinematics algorithm using Jacobian inverse with feedback of joint positions.

4.3.3 Performance Evaluation

The experimental procedures to test the methods previously shown as well as the obtained results will be described here. First are presented the results for the implementation using estimated joint values. Using it, it was expected some error between the desired and the actual positions of the end-effector during the movement due to error introduced by the joint angles estimation. The introduced error may be derived by motor control, rating oscillations, encoders accuracy, etc. In order to confirm and quantify that error some tests were made with one of the robotic arms by performing some movements in the operational space. Tests with different parameters were made in order to verify its influence in the performance of the movement. Thus, different velocities of the movement were imposed , *i.e*, with different execution time (5 and 10 seconds) and the movements were done along two different Cartesian axis, x-axis and z-axis. The results of the comparison between the desired position Rd, the estimated position Rest and the effective position read from the robotic arm encoders Rread taken for the same movement performed along the x-axis are represented in Figure 4.10 and performed along the z-axis are represented in Figure 4.11.

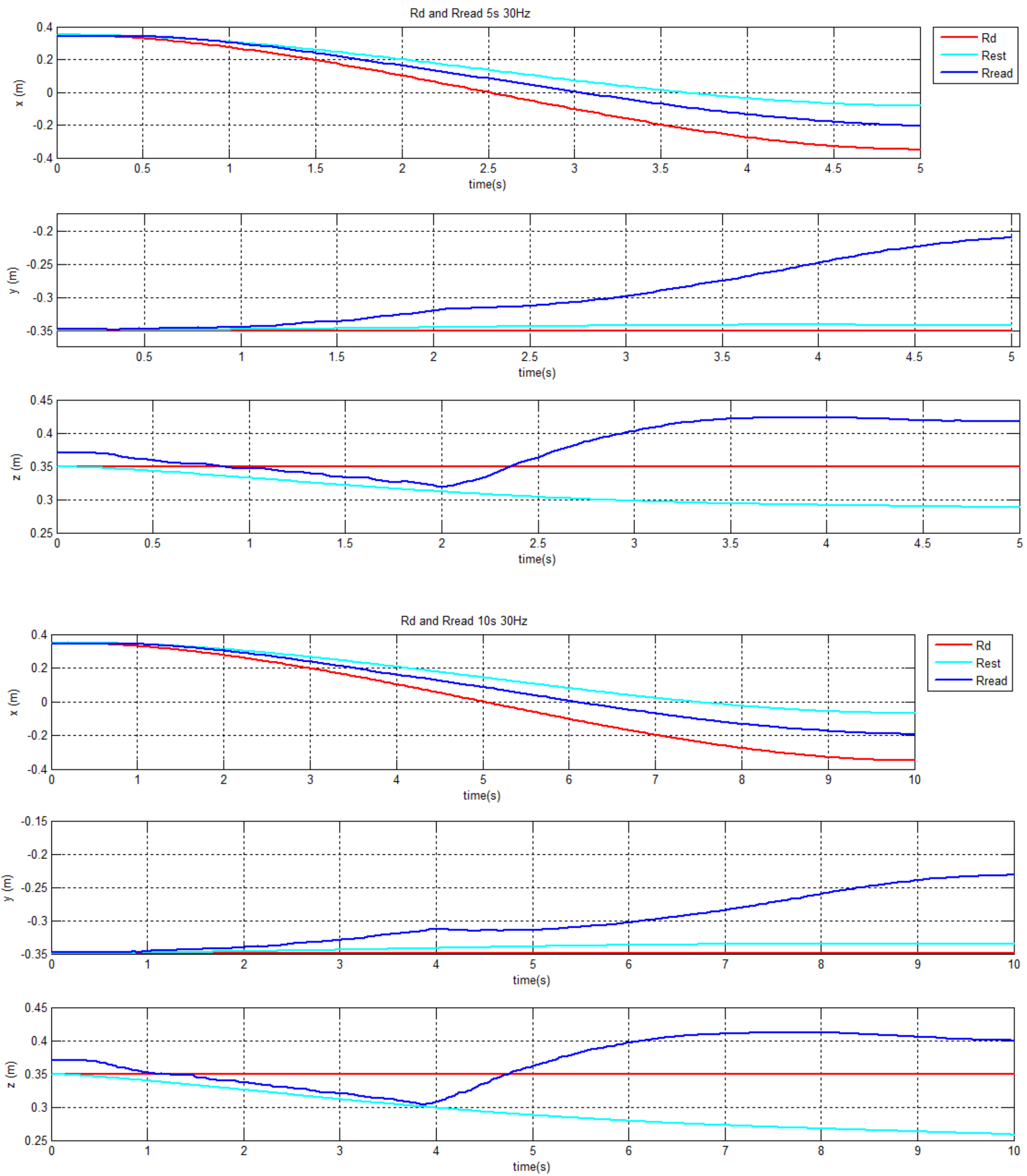


Figure 4.10: Comparison between the desired Rd, estimated Rest and effective Rread positions for a movement with execution time of 5s (Top) and 10s (Bottom) along the x-axis.

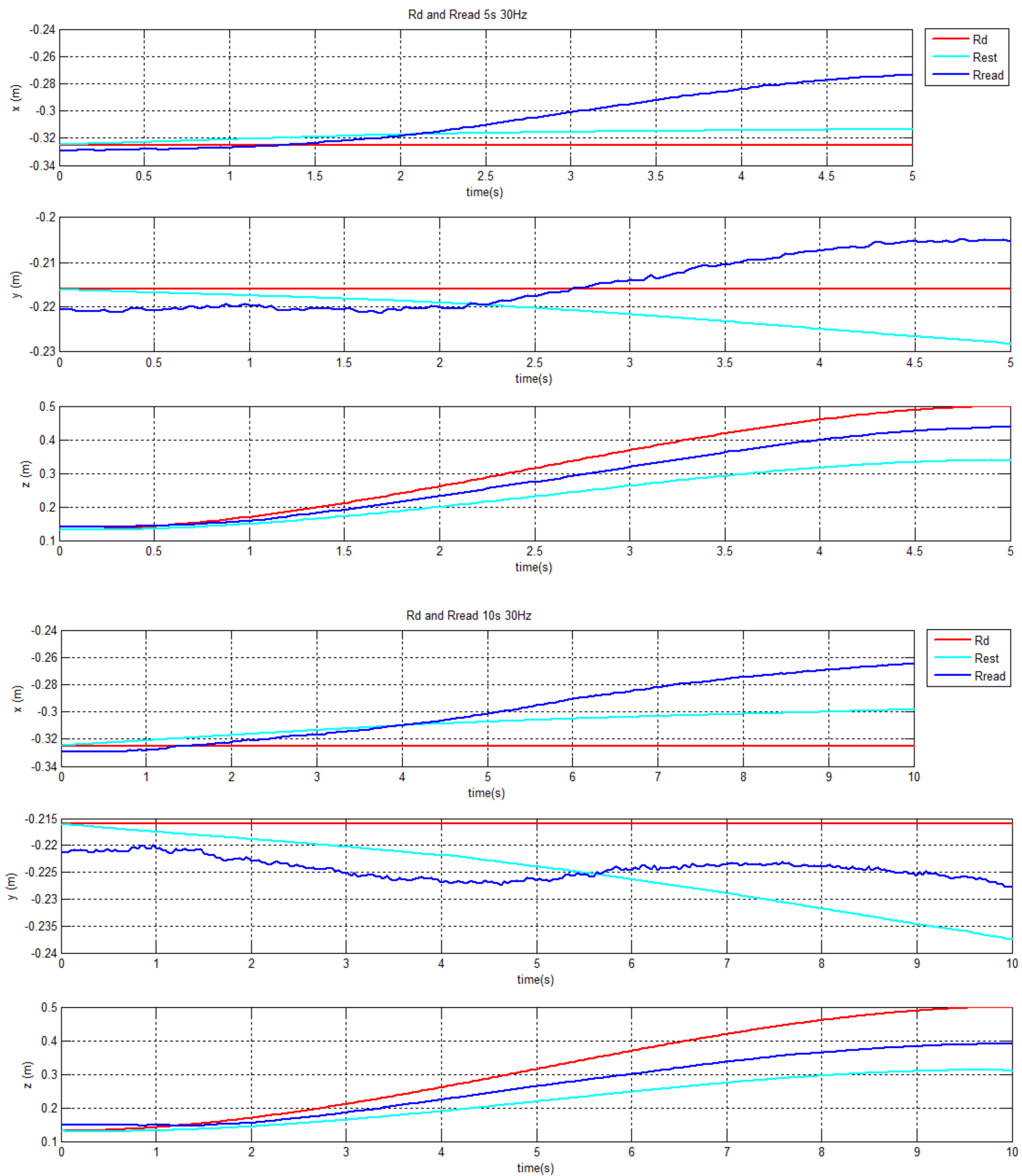


Figure 4.11: Comparison between the desired Rd, estimated Rest and effective Rread positions for a movement with execution time of 5s (Top) and 10s (Bottom) along the z-axis.

The graphics show that there is a deviation between the desired path, estimated and performed by the arm. The Table 4.2 represents the maximum error and the standard deviation between the desired and real positions during the movements performed along the X and Z axis.

Table 4.2: Maximum error between desired and real positions for a movement performed along the x-axis.

Movement axis		**Execution time (s)**			
		5		10	
		max. error (mm)	standard deviation (mm)	max. error (mm)	standard deviation (mm)
X-axis	**x**	148.287	84.163	158.273	85.786
	y	139.676	36.587	118.162	36.462
	z	73.119	27.610	61.674	37.358
Z-axis	**x**	51.554	14.881	60.113	23.378
	y	11.113	4.483	11.688	8.292
	z	64.212	40.350	110.246	5.1068

From the table it can be seen that the error and the standard deviation is considerably high for the application of this dissertation and it increases with the execution time of the movement.

So, it was necessary to apply a control algorithm. The first approach, that cannot be considered control because it does not measure the error between the desired and the real positions, was to get the joint angles feedback from the motors encoders and introduce them in the inverse Jacobian matrix. This approach only allows to have the real joint positions instead of the estimated ones, which improves the followed trajectory, but it does not ensure that the end-effector is following the defined trajectory.

In order to verify that the error is lower using the control algorithm, some tests were made, similar to the ones whose results were presented without using control. Figure 4.12 shows the results of the comparison of the desired (Rd) and the real (Rread) positions obtained for the same movement, performed along the x-axis adopting different constants for the applied controller. Many tests were performed using different velocities and configurations and it was noticed that the values that the constant (k) should take, maintaining the robotic arm behavior stable, were $k>1$ and $k<2.5$, approximately. In this test the values $k=1.2$ and $k=2.2$ were chosen in order to see the impact of the constant in the error.

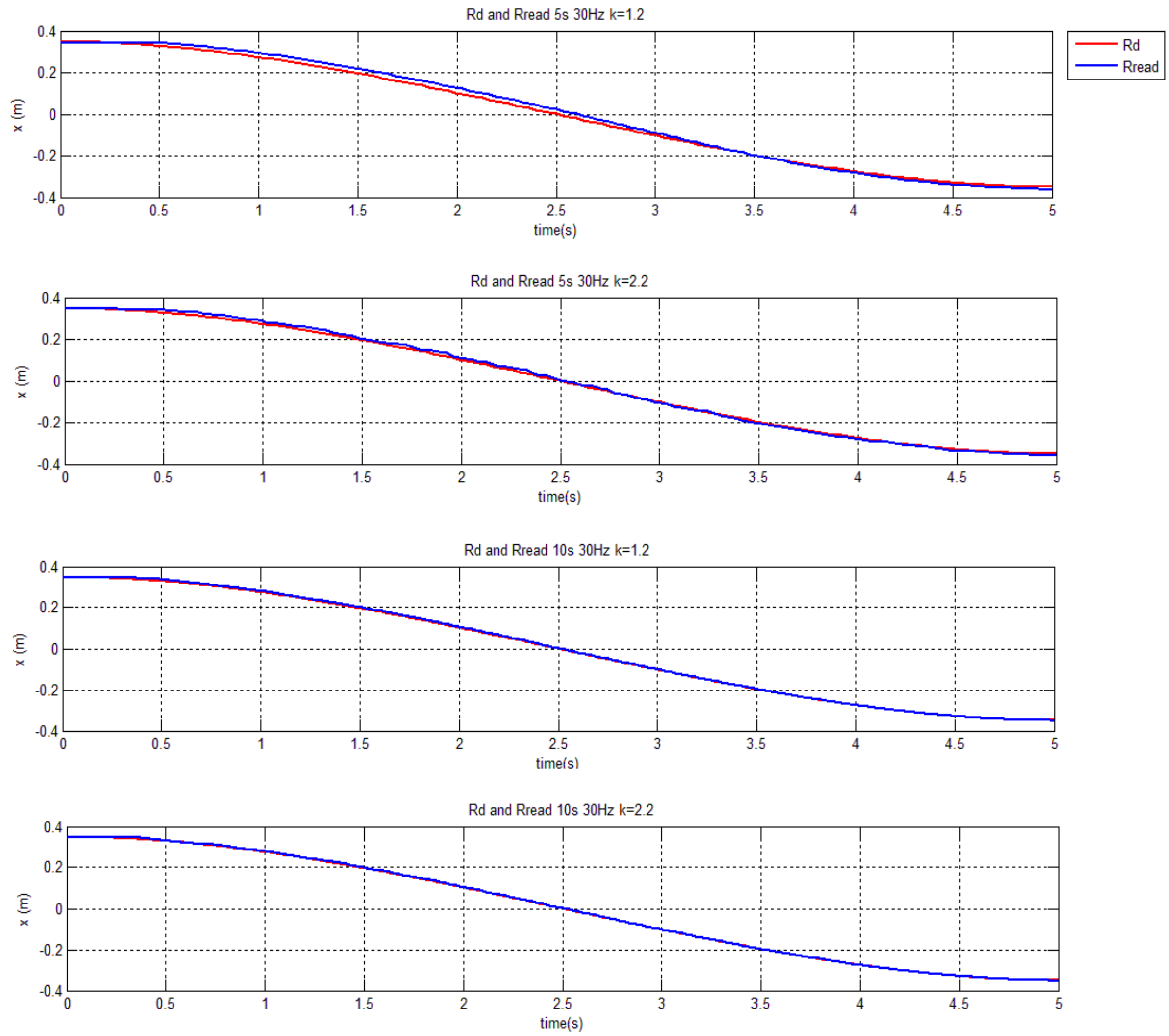
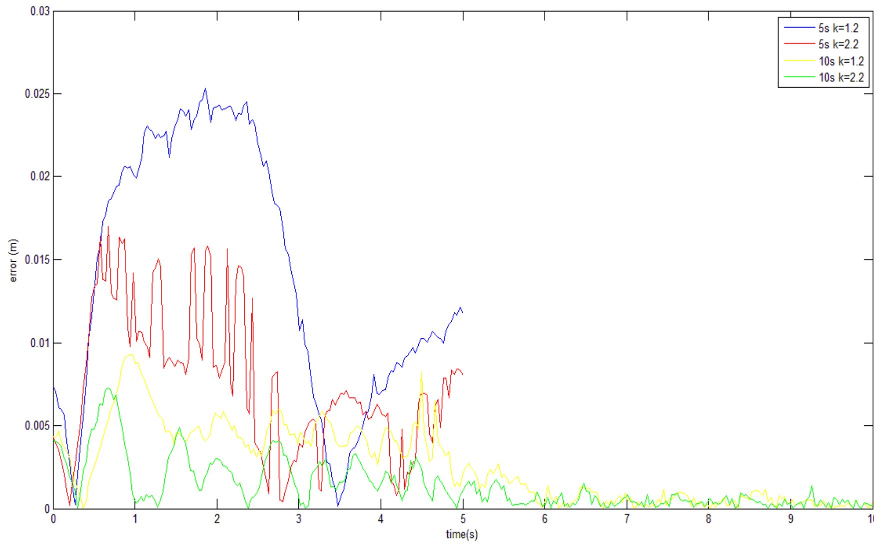


Figure 4.12: Comparison between the desired Rd, estimated Rest and effective Rread positions for a movement with execution time of 5s, k=1.2 and k=2.2 (Top) and 10s, k=1.2 and k=2.2 (Bottom) using control feedback algorithm.

From the graphics of Figure 4.12 it is possible to observe that the desired and real position are very close to each other but it is not clear what situation introduces more error. So, a graphic was plotted with the error between the desired and real positions during time in Figure 4.13.

Movement along x-axis



Movement along z-axis

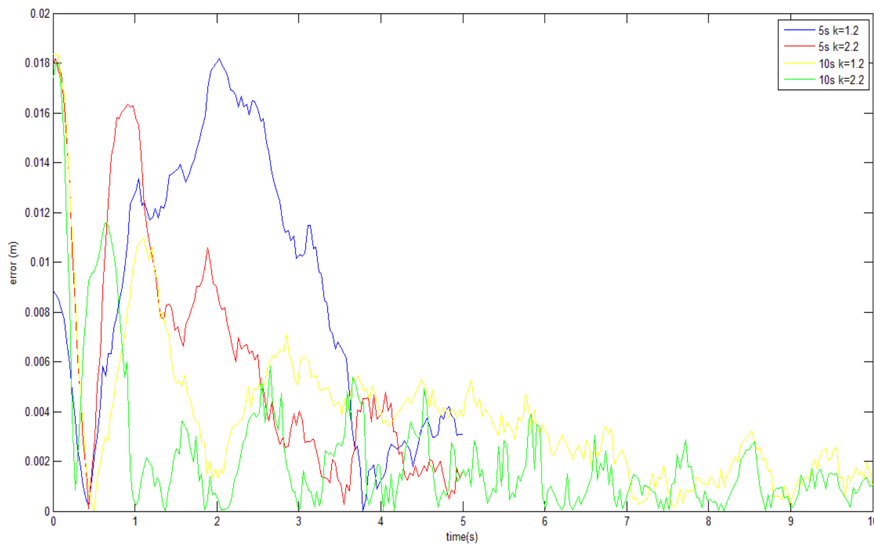


Figure 4.13: Error between the desired and real position in Cartesian space measured in the x-axis for the x-axis movement (top) and z-axis for the z-axis movement (bottom).

From this graphic it can be noticed that the error decreases during time in all situations. It means that, to larger velocities the error will be greater. It is also visible that the higher the constant *k,* the lower the error. Thus, it was assumed a value for the constant around *k=2* for the practical experiments. Table 4.3 shows absolute values for maximum error and standard deviation in the three different components (x,y and z) for movements with 5 and 10s of execution time, control constants of 1.2 and 2.2 and executed along X-axis and Z-axis.

Table 4.3: Maximum error between desired and real positions for a movement performed along the x-axis.

Movement axis		Execution time (s)							
		5				10			
		k=1.2		k=2.2		k=1.2		k=2.2	
		max. error (mm)	standard deviation (mm)	max. error (mm)	standard deviation (mm)	max. error (mm)	standard deviation (mm)	max. error (mm)	standard deviation (mm)
X-axis	x	25.348	12.148	16.979	6.842	9.270	2.081	7.295	0.824
	y	26.214	15.587	14.471	8.034	4.794	1.501	3.857	0.702
	z	24.662	8.658	13.852	4.901	13.852	3.080	10.045	1.435
Z-axis	x	9.217	6.572	5.240	2.759	4.948	1.302	4.685	0.587
	y	4.988	1.318	5.412	0.694	5.588	1.255	5.412	0.842
	z	18.165	8.711	18.194	4.522	18.377	3.395	17.982	1.290

Chapter 5

Software System Integration

This chapter describes how the previous ideas and concepts are integrated in order to create the desired dual-arm robotic system for gesture imitation. As mentioned before, the hardware components of the system are the dual-arm robot, the Kinect sensor and central control unit. Another indispensable element of the gesture imitation system is the human subject that, placed in front of the Kinect sensor, provides the upper-limb movements to be replicated in real-time by the robotic arms. From the software point of view, a major concern is to ensure that the robot arms respect their physical constraints, such as joints limits, joint velocities and available workspace. In line with this, it is possible that during certain periods of time the motion tracking is only approximate or even fails. In these cases, the robot system recovers, automatically, the normal operation as soon as the tracking is possible. Collision detection/avoidance and singularity avoidance are two crucial problems also addressed in this chapter. Besides the outlined problems, there are many others out of the scope of this dissertation not taken into account, such as free objects collision avoidance or self-occlusions in motion capture.

To accomplish the desired level of functionality, the robotic system is provided with low-level functions (*i.e.*, hardware or sensor oriented) and intermediate level functions (*i.e.*, application oriented), as follows:

- Sensorial feedback: this comprises all the low-level modules related with the sensorial information about the robot state (*e.g.*, joint angles, Cartesian coordinates) and the human-motion capture, including trajectory acquisition and pre-processing of the raw-data.

- Robot control: all the intermediate modules generating control actions to the robotic system contribute to this function, including the control modes discussed in Chapter 4.

- Movement constraints: this intermediate function is divided in several modules specifically associated with the robots physical limits, the collision detection/avoidance and the singularity avoidance algorithms.

The following sections describe the algorithms implemented to solve the aforementioned problems and give more details about the developed modules and their dependencies under the ROS framework.

5.1 Motion Capture and Kinematic Mapping

The human-motion capture system was based on a Kinect sensor, the OpenNI drivers and the openni tracker package. This package broadcasts the OpenNI skeleton frames using transformations that maintain the relationship among coordinate frames in a tree structure buffered in time. Therefore, it is possible to transform points and vectors between any two coordinate frames at any desired time. In the proposed approach, the Kinect skeleton coordinates are extracted relative to the reference frame located in the neck. The Rviz ROS package provides a useful 3D visualization environment to observe the transformations published by the openni tracker. Figure 5.1 shows the overlap of depth and RGB data and the corresponding coordinate frames distributed along the skeleton, including the coordinate frame of the Kinect sensor (closest one). Openni tracker needs an initial pose calibration. To recognize the skeleton, the subject should face the sensor for a short period of time, standing in a "surrender" pose until a message of user tracking is sent.

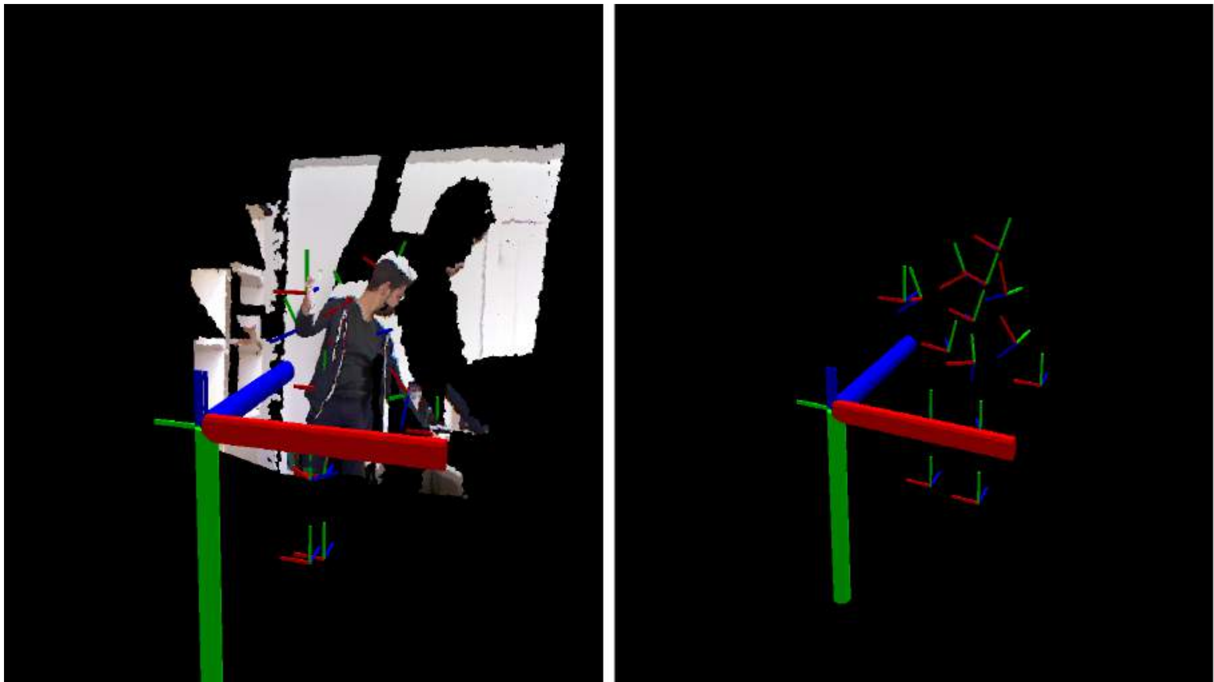


Figure 5.1: Capture of depth and RGB images and transformation (left) and a capture with just transformations using Rviz.

In order to define the connection between the human and the robot arms, it is important to remember that the two systems are kinematically different from the point of view of number

of DOFs, link lengths and available workspace. As result, connecting them at a joint level does not seem to be appropriate or even feasible. Instead, it is preferable to connect them at their Cartesian positions in a position-position mode. This is a typical control mode for stationary robots with limited workspace in which the human arm displacements are interpreted as desired positions. In this context, the human and robot arms can be connected at their tips (*i.e.*, hands) in a position-position mode according to the following general expression:

$$x_r = \mu \cdot x_h + x_{offset} \tag{5.1}$$

where x_r is the robotic arm tip position measured in the reference coordinate system S_r, μ represents the motion scale, x_h is the human arm tip position measured in S_r and x_{offset} is the offset between the two tips. On one hand, the scale is set to either map the two workspaces as best as possible or to provide different resolutions for each workspace. On the other hand, the offset between the two systems means that they are not constrained to start at the same location.

In this work, the movements of the human's hands are directly mapped onto desired positions of the robot's end-effectors. This kinematic mapping is implemented by a ROS module that subscribes to the openni tracker transformations and generates the control actions to the robot arms (see Figure 5.2). However, the motion capture data provided by the Kinect sensor (skeleton data) may be unsuited if directly transferred into control actions to be applied to the robotic arms. This sensor does not have sufficient resolution to ensure consistent accuracy of the skeleton tracking data over time. This problem manifests itself as the data seems to vibrate around their positions. Therefore, low-pass filtering of the raw-data uses a moving average to attenuate noise and vibrations, smoothing the actions sent to the robot controller, as follows:

$$R_{filttered}\left[i\right] = \frac{1}{N} \sum_{j=0}^{N-1} R_{kinect}\left[i+j\right]$$

where $R_{filtered}$ is the output signal, R_{kinect} is the input data from the Kinect sensor and N is the number of points in the average filter. After filtering, the ROS module publishes the desired coordinates of the two end-effectors and the corresponding velocities estimated numerically using the time stamp vector. After several experiments receiving data from the Kinect sensor was established a fixed value for the number of samples of moving average $N = 15$, based on a relation between the quality of the data and the introduced delay.

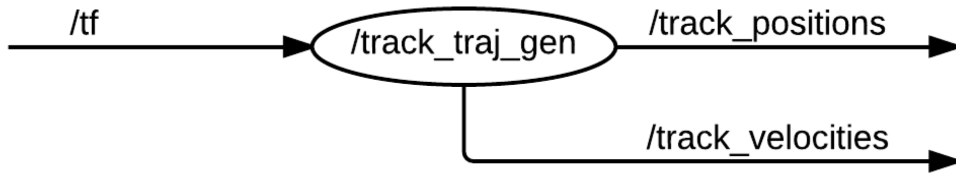


Figure 5.2: Trajectory generator of hand tracking ROS implementation.

5.2 Dual-Arm Robot Control

The human gesture imitation performed by robots leads to some problems. Some are related to physical aspects of the robotic system, others related to possible collisions that may happen during the performance and others related to cinematic of the arms. This sub-section presents the imitation problems that were taken into account in the software implementation as well as the adopted strategies to solve them. Solving them allows to obtain a more reliable and robust system and to avoid some damage that the robot can cause. Besides the outlined problems there are many others that were not taken into account such as free objects collision avoidance, motion capture joint occlusions, etc, which are out of the scope of this dissertation. In previous sections, most of the experiments were executed using just one of the robotic arms since the behavior of the other one is similar. In this section, the aspects presented take into account using both robotic arms. Thus, a global reference frame in the center of the torso was defined, as shown in Figure 5.3, as well as a homogeneous transformation matrix to do the correspondence of each robotic arm local reference frame related to the global one as follows:

$$T = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & d \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where T is the homogenous transformation matrix to convert the coordinates from the local to global reference frame, d is the distance from the origin of the global reference frame to the origin of the local reference frame of each are, assuming positive values for left robotic arm and negative value for the right robotic arm. T^{-1}can be used to do the inverse transformation if it is necessary to convert the global coordinate to the local reference frame.

Figure 5.3: Hardware implementation and representation of global (black axis) and local (white axis) reference frames.

5.2.1 Overcome Physical Aspects

The physical morphology and kinematics of a humanoid can be a problem in human gesture imitation. It is highly probable that the morphology and kinematics of the humanoid is not quite similar to the human's. The correspondence problem is related to the links length, the joint velocities and the mapping of the position of the end-effector related to the human hand position.

The adopted strategy to do the mapping between the demonstrator and imitator was based in a relation between the operational space from the point of view of each one. Thus, the acquired data from the human hand is the position of it related to the human coordinate system and then it is transposed to a similar coordinate system in the humanoid. It solves the problem of mapping of human joints with the robot. Although it introduces other important aspects to the imitation. Not using a direct mapping means that the human and robotic arm can have different lengths. It can cause that the human hand position exceeds the humanoid workspace. In this section the strategies adopted are discussed when the desired trajectory implies that the robot exceed physical joint limitations, velocities as well as its workspace limits.

The strategy adopted in order to approximate the morphology of the robotic arm to the human's arm was to limit the elbow pitch angle to just take positive values, as presented in Figure 5.4. As the human arm elbow cannot invert its direction, this constrain closely approximates both human and robot morphology during the imitation.

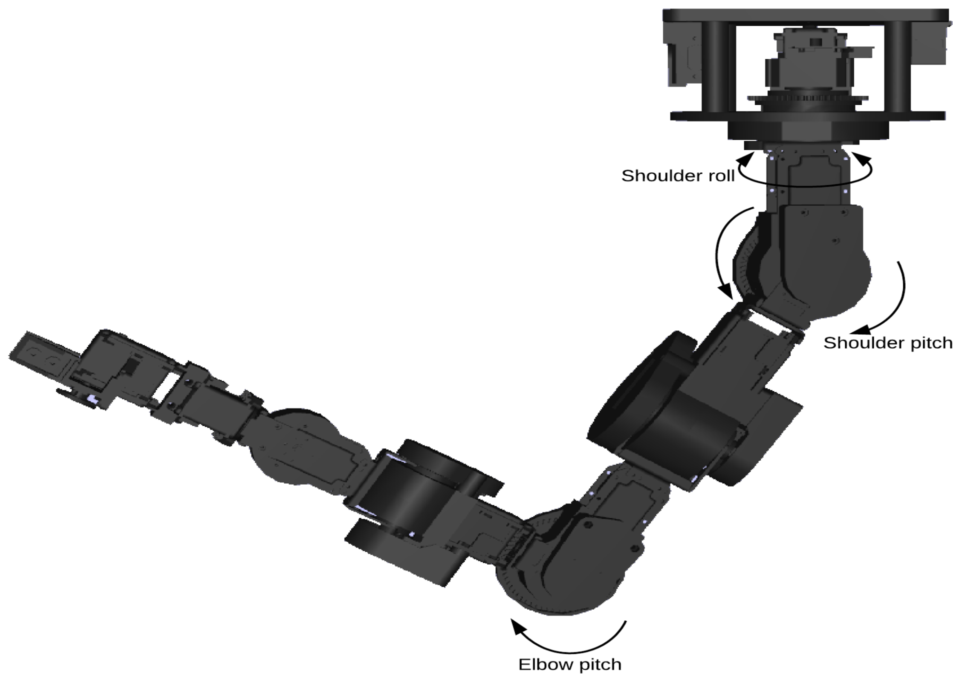


Figure 5.4: Physical aspects representation.

By limiting the elbow pitch angle just approximates both robot and human morphologies but the physical joint limits still exist. The strategy adopted when a joint reaches its physical limit during the imitation was based on fixing that joint to its limit and continue sending the desired joint values to the other joints. It was expected that the error would gradually increase and the joint velocities increased as well. During the experiments, it was noticed that the robotic arm had a good performance during this particular case.

As the human movements are done resorting to muscles and they react much faster than the robotic motors, it was expected that in some movements the robotic arm joints were not able to follow the velocities imposed by the human demonstration. Thus it was necessary to implement a strategy to solve this problem. It consisted in maintain the limit velocity of the joint that exceeded it and continue sending the desired velocities to the other joints. It was expected that the controller adjusts the joint that that did not reach the velocity limit to compensate the loss of one degree of mobility until it can recover.

It was also adopted a strategy for the cases when the human hand exceeds the humanoid workspace for some reason. It consisted in stopping the robotic arm when the workspace limit was reached and waiting until the human hand returns into the workspace. It can introduce some error to the controller and, when the human hand returns into the workspace, if the human hand position is considerably distant from the end-effector it can result in large joint velocities until continuing the tracking. The experimental results related to these problems can be seen in Section 6.1.

5.2.2 Singularity Robust Inverse Kinematics

Differential kinematics is used because of its simplicity and efficiency. However, it is necessary to deal with its singularities. Singularities can be split into two categories (Craig, 2005):

1. Workspace-boundary singularities, which occur when the manipulator is fully stretched out and the end-effector is near to the boundary of the workspace.

2. Workspace-interior singularities, which occur inside the workspace and are caused by special configurations of the manipulator, for instance when two or more joint axes are aligned.

When a manipulator is in a singular configuration it loses DOFs and the mobility of the structure is reduced. When it happens, infinite solutions to the inverse kinematics may exist and in the neighborhood of the singularity, small velocities in Cartesian space may cause large velocities in the manipulator joints (Sciavicco and Siciliano, 1996).

There are several methods to solve IK problems in robot control. Three of the most popular Jacobian-based methods are: Jacobian transpose, Pseudo-Inverse, and Damped Least Square methods (Buss, 2009), (Tee et al., 2010). The Jacobian transpose method is based on using the Jacobian transpose matrix instead of the inverse matrix of it. Although the

transpose is not the same as inverse but its use can be justified in terms of virtual forces (Buss, 2009). This method is faster than others since it does not need to calculate any inverse matrix. Pseudoinverse method is usually used when the Jacobian matrix is non square, which is not possible to use the ordinary inverse. This method tends to have stability problems near to singular configurations (Buss, 2009). On the other hand, damped least squares method, explained in (Buss, 2009), avoids many of these problems and it can give a numerically stable method to calculate the joint velocities. According to (Buss, 2009) and (Tee et al., 2010), the most efficient method near singular configurations is the damped least square. (Duleba and Opalka, 2013) affirm that the damped least squares method is more computationally expensive than Jacobian transpose but it occasionally loses the convergence property.

In this dissertation the option was implementing a simpler solution which serves as preliminary method to solve the singularities of the robotic arm used. Its implementation took into account some aspects, important to mention:

1. All singular configurations were solved by limiting the joint angles to avoid them, when the robotic arm is fully extended, except the singular configuration that occurs when the end-effector passes near by the shoulder roll axis;

2. The verification if the end-effector was or not near to a singular configuration was done by delimiting a cylindrical region aligned with shoulder roll joint;

3. The solution intends to not stop the motion when the singular configuration occurs but to operate inside the singularity region;

4. The motion inside the singularity region was limited to a plan defined by should roll angle, which introduce error in the executed trajectory;

5. There are three main situations that can be considered in this approach: Starting outside the singularity region and enter it; Starting inside the singularity region and operate inside or leave it; Starting outside the singularity region and pass through it.

Taking these aspects into account it was developed a Matlab simulation to implement and test the algorithm. The algorithm is based in two fundamental steps:

- When it is detected that the end-effector is inside the singularity zone, the shoulder roll joint is fixed assuming the direction given by the Cartesian space velocity vector in the previous instant, which defines a parallel plan to the one where was the end-effector trajectory. Related to this three possible solutions were assumed: Continuously adjusting the shoulder roll joint to the angle given by the velocity vector; Fixing the shoulder roll joint to the initial value and keep it fixed until the end-effector leaves the singularity zone; Fixing the shoulder roll joint but defining angular intervals where the joint is adjusted. The first solution can cause jitter during the movement so it did not seem a good solution. On the other hand, fixing the joint angle to the initial value until the end-effector leaves the singularity region seems to constrain too much the

movement. Consequently, the more acceptable solution was to adjust the joint angle for specific predefined angular intervals;

- The other two joints are controlled while fixing should roll joint but it was necessary to use a new Jacobian matrix, based on a 2 DOFs manipulator operating in the plan defined by shoulder roll joint;

The plot of Figure 5.5 shows two plots from the Matlab simulation with the objective to give an idea of the strategy behavior. The left side of Figure 5.5 presents a perspective view of the robotic arm and the singularity region, defined by the cylinder. The right side of Figure 5.5 represents the top view of the system and the trajectory of a tested movement, starting outside and then entering the singularity region. It is possible to observe the shoulder roll joint aligning to a parallel plan to the one defined by the previous trajectory and then the movement does not finish in the desired point, introducing some error as mentioned before. This algorithm worked as expected in simulations but it still had some limitations that were not solved.

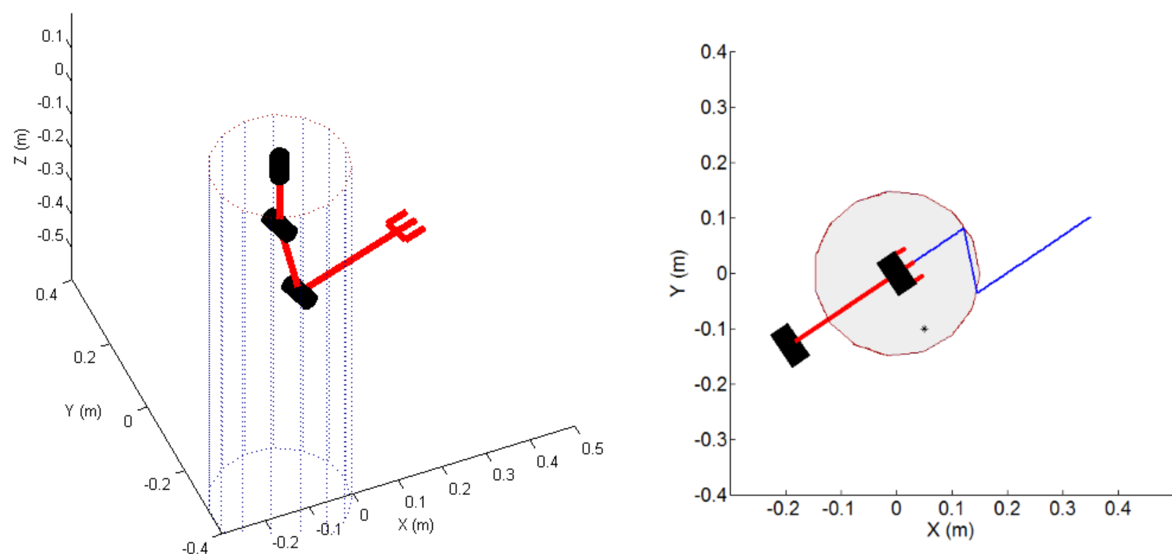


Figure 5.5: Singularity simulation using Matlab. The robotic arm links are represented as red lines and its joints as black cylinders. The singularity region is marked as a dotted cylinder, the executed trajectory as a blue line and the final desired position as a black asterisk. A perspective view (left) and top view (right) of the system are represented.

5.3 Self Collision Detection and Avoidance

The safety of an automated robotic system is one important aspect that should be taken into account. The system must be projected in order to protect the people around it as well as the system itself. In this project it was implemented a simple collision detection

algorithm that allows the detection of collision between both arms and between each arm and the torso structure or another predefined structure or object. This algorithm was based in intersection of geometric primitives more specifically in line to line intersection. The main idea of this algorithm is to create bounding boxes, composed by four rectangular faces, around the different parts and see if they intersect each other.

This algorithm has as input the set of the vertices from two faces that can collide. Then it is calculated the normal vectors of each face and then the line of intersection of the planes defined by the two faces. The parametric equation of a line defined by two points, P_0 and P_1 can be defined as shown in equation 5.2.

$$P(s) = P_0 + s(P_1 - P_0) = P_0 + su \tag{5.2}$$

Where u is the line direction vector and s is a real number between 0 and 1 which represents a point on the finite segment and its value represents a fraction of that segment. If its value is lower than 0 or higher than 1 it means that the point is outside the segment in P_0 or P_1 direction, respectively. In each pair of faces there are one normal line to each face, thus it can be defined two parametric equations.

$$P(s) = P_0 + su \tag{5.3}$$

$$Q(t) = Q_0 + tv \tag{5.4}$$

It is possible to see if the two lines are parallel and it means that they are coincident or do not intersect. It happens when their directions are collinear, when u and v are linearly related as $u = av$, this is equivalent to the condition $u_1 v_i - u_i v_1 = 0$.

If the segments are not parallel they intersect in a single point. If they intersect in 3 dimensional space then their linear projections into 2 dimensional planes will also intersect. This restriction was used to calculate the intersection of each face edge and the planes intersection line. The intersection point of two lines and the associated vectors are represented in Figure 5.6.

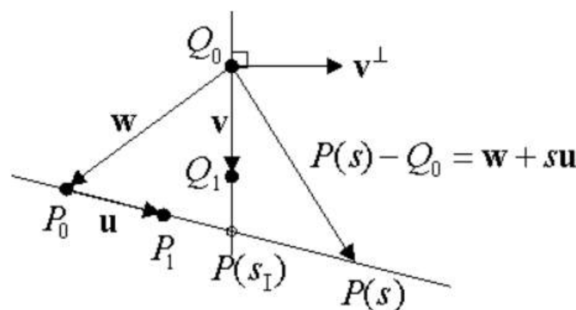


Figure 5.6: Two line intersection in 2 dimensional plane and the associated vectors.

It is necessary to calculate the intersection point I at $P(s_I)$ and $Q(t_I)$. At the intersection it's verified that $P(s_I) - Q_0$ is perpendicular to $v^\perp$ and that $v^\perp \cdot (w + s_I u) = 0$. By solving these equations along with equation it's obtained the s_I value, equation, and similarly the t_I value, equation.

$$P(s) - Q_0 = w + su \tag{5.5}$$

$$s_I = \frac{v_2 w_1 - v_1 w_2}{v_1 u_2 - v_2 u_1} \tag{5.6}$$

$$t_I = \frac{u_1 w_2 - u_2 w_1}{u_1 v_2 - u_2 v_1} \tag{5.7}$$

If both lines are segments, like in this case, both t_I and s_I must be between 0 and 1 for the segments to intersect. The implemented algorithm is explained in form of diagram in Figure 5.7.

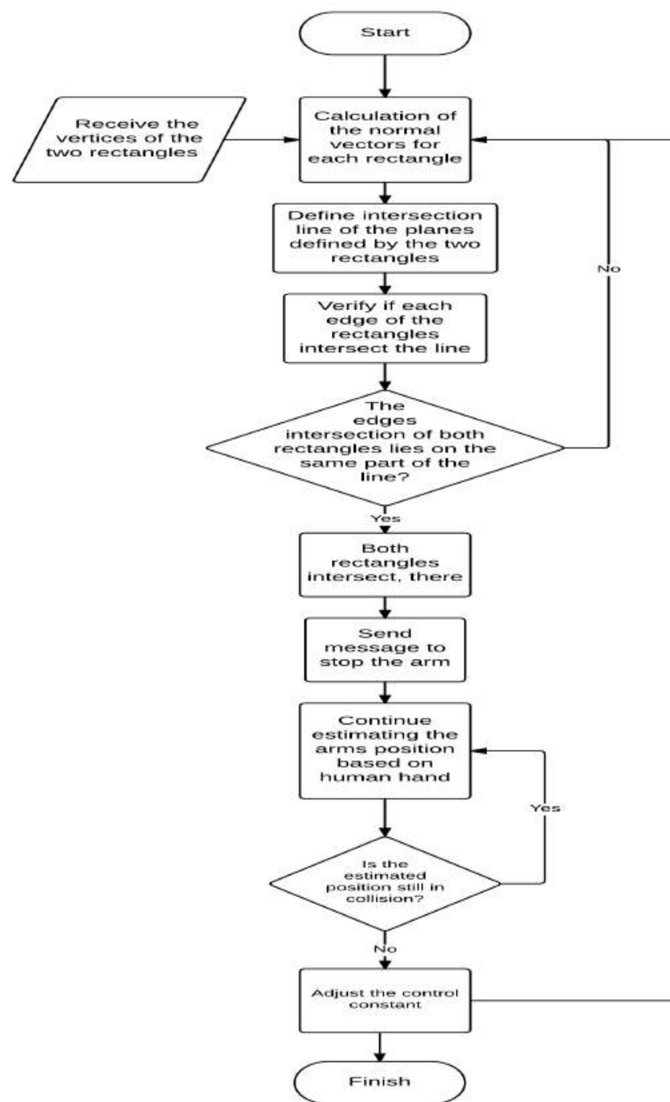


Figure 5.7: Collision detection algorithm diagram.

This is a simple algorithm but it is robust and light because it was optimized to this specific situation. It means that the geometry of the bounding boxes is simpler, parallelepiped, it uses just vertices and segments instead of meshes that are much computationally heavier, the volume of the bounding boxes can be adjusted to improve the performance and the comparison between the bounding boxes faces can be arranged in order not to use unnecessary processing. Those advantages restrict the algorithm to some applications but it proved to be suitable to this project.

In order to prove and test the algorithm it was made a Matlab script to perform some tests. It was used the direct kinematics of the robot to draw it and to move the arms in a simple way. Figure 5.8 shows the representation of the humanoid torso structure as well as the bounding boxes of the link between the elbow and shoulder and the link between the shoulder and end-effector as well as the bounding box of the torso.

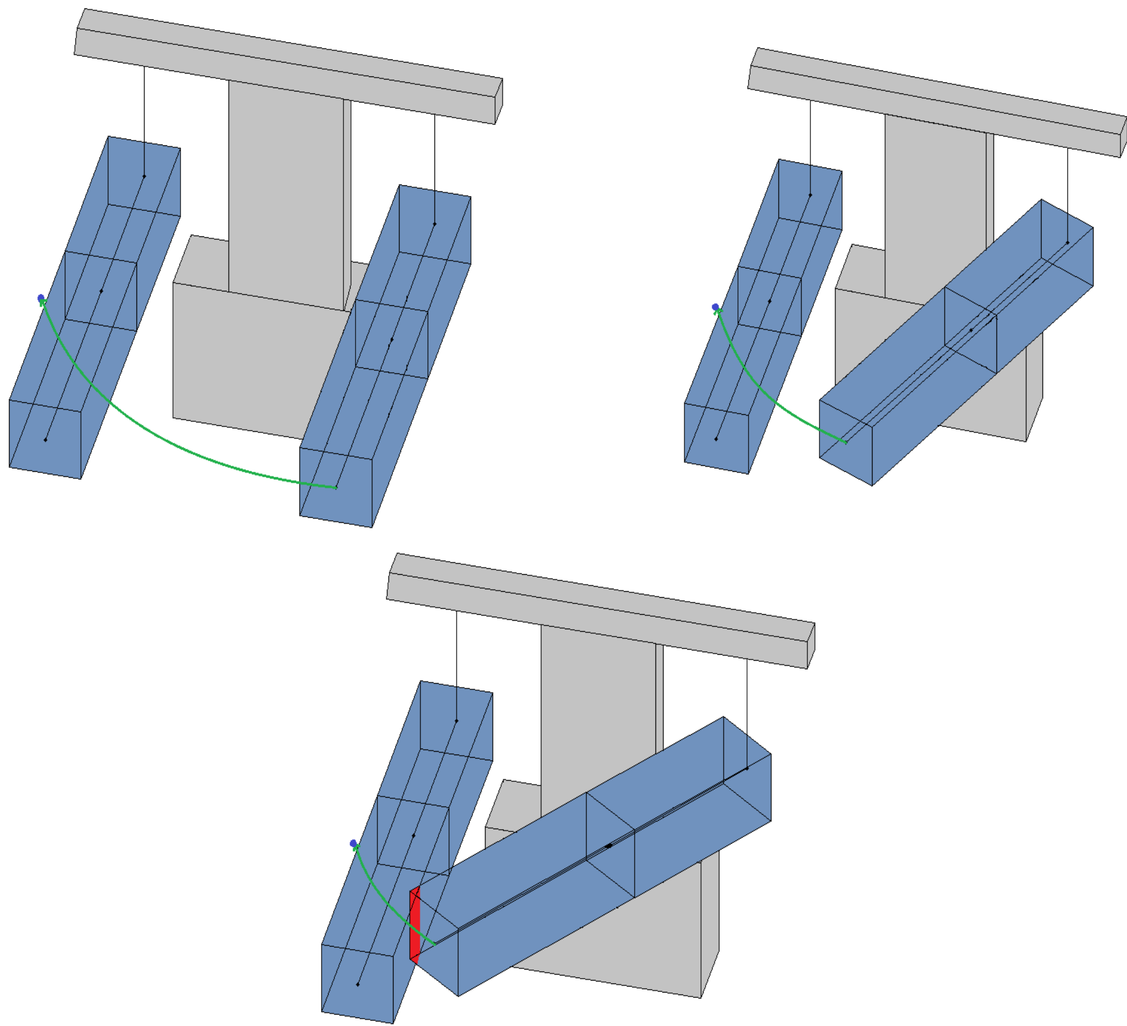


Figure 5.8: Sequence representing a movement that causes collision between the two robotic arms. Starting position (top left), intermediate position (top right) and intersection position and collision detected (bottom center).

Using this simulation script it was possible to test the algorithm in several situations to make it reliable and to guarantee that it will work in real situation.

In terms of implementation it was developed an independent process that had the goal to evaluate if both robotic arms were in collision with each other and with the torso structure. A simplified representation of the ROS implementation is represented in Figure 5.9.

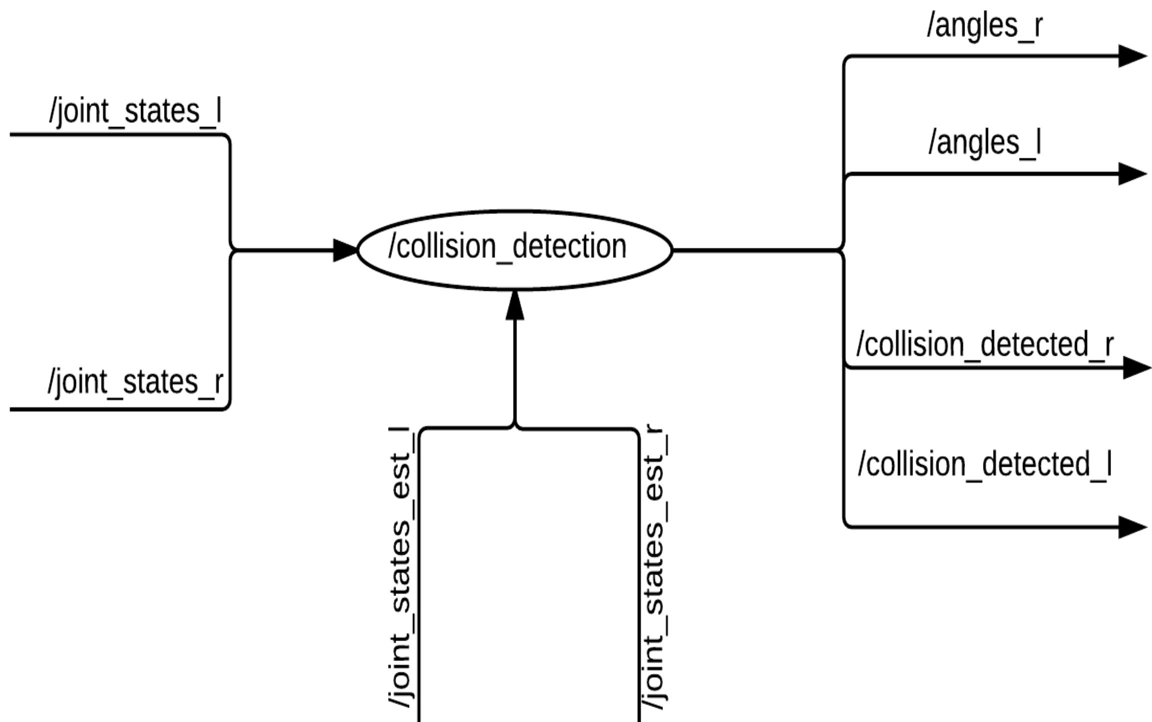


Figure 5.9: Collision detection ROS implementation simplified.

The collision detection process is responsible for determining if both robotic arms are in collision with each other and with the torso structure taking as input the joint values of both arms. These values are published in /joint_states_l and /joint_states_r topics for left and right arm, respectively. The collision detection node verifies if any collision occurs and, as soon as it happens, it sets a Boolean flag as true and publishes it in the /collision_detected topics as well as a start flag in the /start_joint_l and /start_joint_r topics. The start joint topics indicates that the arm is sent to the angles received in /angles_l and /angles_r topics that correspond to the actual joint angles with the objective of stopping the robotic arm movement. The collision detected flag indicates to velocity control node that the movement must stop and that it must start to publish estimated joint values. These values are based in the human hand trajectory, but instead of being sent to the robotic arm, they are sent to the collision detection node.It allows the collision detection process to know if the human hand is still in collision or not. When the hand leaves the collision zone the collision detection process sets the flag to false and the velocity control process sends the values to the robotic arm again, adjusting the controller constant due to possible error accumulation if the position of the hand in the instant that leaves the collision is very different from the end-effector position.

5.4 Overall ROS-Based Software Architecture

The gesture imitation control was based on the combination of the different modules presented before. It was implemented a well-defined procedure in order to perform the robotic gesture imitation. It consists of acquiring the Kinect sensor data of the human hands and neck, filtering it and planning the trajectories of the robot arms. However, it must be highlighted that the two devices are not always connected in a tracking behaviour. For example, when the system is turned on the two systems do not need to adopt the same configuration. During this phase, the coordinates extracted online from the human hands are the target input of the point-to-point control algorithm that moves the arms with high velocity. The gesture imitation system only enters in the normal tracking mode when the error between the target and the real coordinates is below a predefined threshold. At this time, the control changes from the point-to-point to the continuous mode in which the end-effector velocities try to follow the linear velocities of the human hands. A similar procedure is used whenever some of the arms (or both) cannot follow the desired captured motion due to physical limits. The flowchart that represents this strategy is shown in Figure 5.10.

Figure 5.11 shows a visual graph of the modules developed in ROS and their dependencies. It is possible to observe all the processes responsible to perform the gesture imitation control for both left (top) and right (bottom) robotic arms, including the transform /tf that is the topic that publishes the skeleton positions of the captured human by the Kinect sensor. This architecture of processes allows the system to perform the human gesture imitation. Table 5.1 presents the complete list of topics used, as well as the type of message published in each, its publishing rate and the parameters of each message. The loop rates are limited by the sensors' maximum rate: the Kinect sensor operates at 30 Hz and the encoders of the robotic arms can provide information at a rate of 50 Hz. Therefore, it was established a frequency rate of 30 Hz for all topics since it was, experimentally, verified that it does not have much influence on the control performance. Table 5.2 lists the implemented nodes, the input parameters and the main function of each one (the input parameters are described on the bottom).

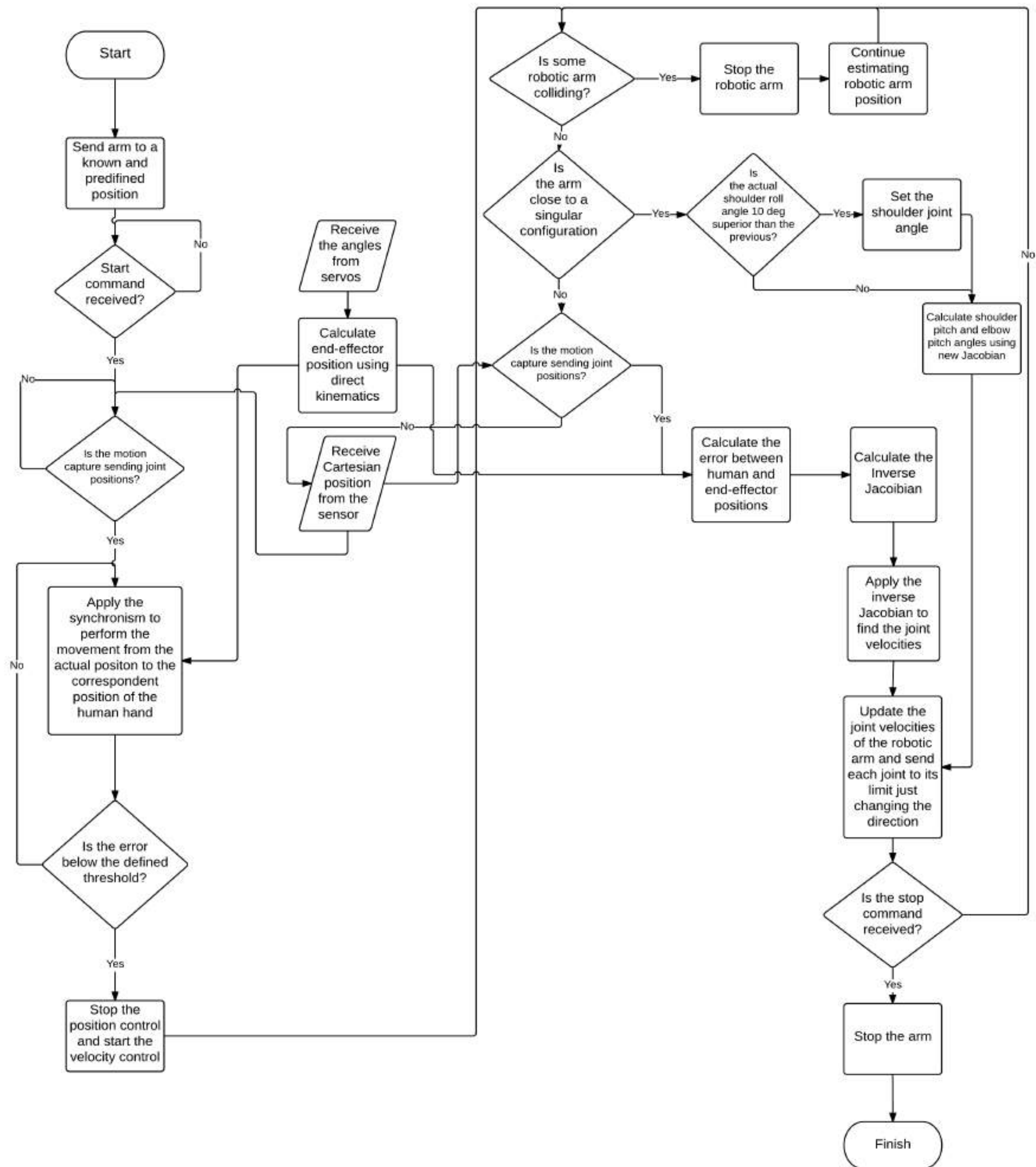


Figure 5.10: Imitation Algorithm block diagram.

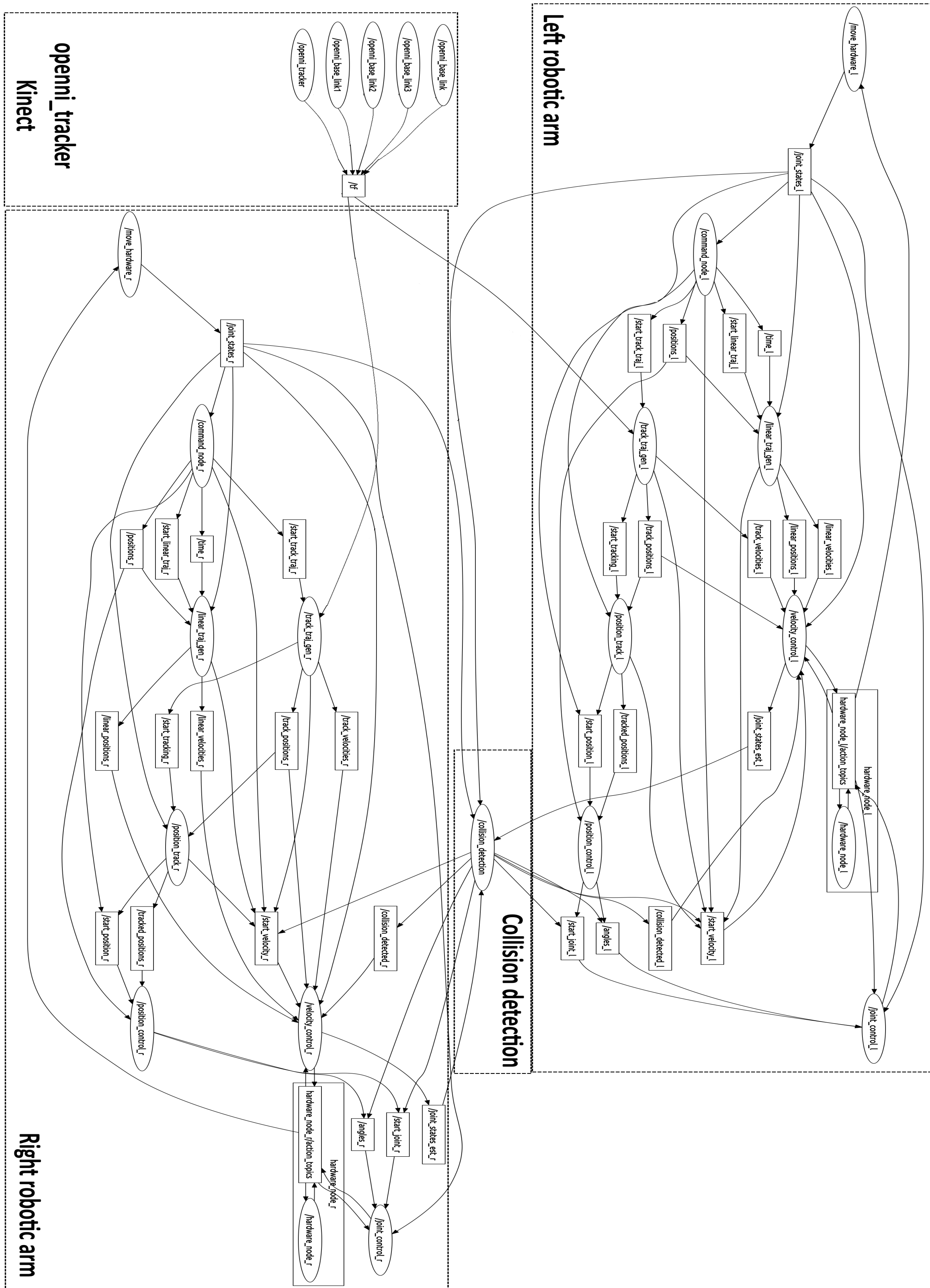


Figure 5.11: Imitation algorithm ROS diagram with both arms. l and r refer to left and right respectively.

Table 5.1: Topic names and specification.

Topic name	Message type	Rate (Hz)	Parameters
/start__track__traj	std__msgs/Bool	30	bool data
/start__position	std__msgs/Bool	30	bool data
/start__linear__traj	std__msgs/Bool	30	bool data
/start__velocity	std__msgs/Bool	30	bool data
/start__tracking	std__msgs/Bool	30	bool data
/start__joint	std__msgs/Bool	30	bool data
/collision__detected	std__msgs/Bool	30	bool data
/positions	geometry__msgs/Point	30	float64 x float64 y float64 z
/tracked__positions	geometry__msgs/Point	30	float64 x float64 y float64 z
/track__positions	geometry__msgs/Point	30	float64 x float64 y float64 z
/linear__positions	geometry__msgs/Point	30	float64 x float64 y float64 z
/angles	sensor__msgs/JointState	30	string[] name float64[] position float64[] velocity float64[] effort
/joint__states	sensor__msgs/JointState	30	string[] name float64[] position float64[] velocity float64[] effort
/track__velocities	geometry__msgs/PointStamped	30	std__msgs/Header header geome- try__msgs/Point point
/linear__velocities	geometry__msgs/PointStamped	30	std__msgs/Header header geome- try__msgs/Point point
cyton/goal	cyton/cytonAction	30	float32 position float32 rate float32 time int32 eeindex uint32 home float32 gripper__value float32 gripper__rate
cyton/feedback	cyton/cytonAction	30	float32 position float32 rate float32 time float32 gripper__feed__value float32 gripper__feed__rate

Table 5.2: Node names and specifications.

Node name	Input parameters	Function
/command__node	d	Command line interface for input commands
/hardware__node	none	Subscribe to cyton/goal to receive joint values and publish them into cyton/feedback topic
/move__hardware	none	Subscribe joint angles and rates and send to cyton robotic arm
/joint__control	none	Subscribe joint angles, apply the synchronism publish the joint angles and velocities into cyton/goal topic
/position__control	d	Subscribe to the operational space position, verify if it is inside the workspace, apply inverse kinematics and publish the correspondent joint angles
/position__track	d	Subscribe to human hand position and publish it to position__control node until the error between human and end-effector positions are closed
/velocity__control	d K	Subscribe to the operational space position and velocity form the trajectory generator, apply the inverse Jacobian to calculate the correspondent joint velocities and publish them into cyton/goal topic
/linear__track__gen	d	Subscribe to the desired operational space positions and the execution time, calculate the operational space velocity based on a linear trajectory and publish them into the /linear__velocities and /linear__positions topics
/track__traj__gen	d C	Subscribe to transform (/tf) containing the frames of the human hand and neck, calculate the position, as well as its velocity, of the hand relative to the neck applying a moving average filter and publish them into /track__positions and /track__velocities topics
/collision__detection	none	Subscribe to joint states to know the robotic arm position applying DK. Publish a flag when the collision is detected and the joint angles correspondent to that position to stop it. When collision occurs the robotic arms position is estimated to determine the moment of recovering to normal tracking.
d	Distance between the center of the robotic arm base and the global frame of the system. Positive is related to the left arm Negative is related to the right arm	
K	Constant of the proportional controller	
C	Number of samples of the moving average filter	

Launching all nodes at the same time for both robotic arms can be done resorting to ROS launch arguments, which can be passed through the command window or assume a default value. The system was divided in three launch files, two of them responsible for launching left and right arm processes independently and the third one responsible for launching the other two launch files at the same time and also the collision detection process. It was also launched the processes responsible to change the *XML* files that specify the USB port of each robotic arm. The *XML* code related to both left and right arms launch file is presented in Appendix E. The *XML* code related to the launch file that launches the other two is presented in Figure 5.12.

```
1  <launch>
2
3   <!-- args USBL and USBR defines the USB port number for LEFT and RIGHT robotic arm respectivly-->
4
5     <arg name="collision_freq" default="30" />
6     <arg name="USBL" default="0" />
7     <arg name="USBR" default="1" />
8
9   <!-- LEFT ROBOTIC ARM -->
10    <arg name="dl" default="0.202"/>
11    <arg name="median_cl" default="15"/>
12    <arg name="Kl" default="15"/>
13
14    <node name="change_xml_file_l" pkg="cyton" type="change_xml_file" args="$(arg USBL)"/>
15    <include file="$(find cyton)/launch/cyton_l.launch">
16        <arg name="d" value="$(arg dl)" />
17        <arg name="median_c" value="$(arg median_cl)" />
18        <arg name="K" value="$(arg Kl)"  />
19    </include>
20
21  <!-- RIGHT ROBOTIC ARM -->
22    <arg name="dr" default="-0.202"/>
23    <arg name="median_cr" default="15"/>
24    <arg name="Kr" default="15"/>
25
26    <node name="change_xml_file_r" pkg="cyton" type="change_xml_file" args="$(arg USBR)"/>
27    <include file="$(find cyton)/launch/cyton_r.launch">
28        <arg name="d" value="$(arg dr)"  />
29        <arg name="median_c" value="$(arg median_cr)"  />
30        <arg name="K" value="$(arg Kr)" />
31    </include>
32
33  <node name="collision_detection" pkg="cyton" type="collision_detection" required="true" args="$(arg collision_freq)"/>
34
35 </launch>
```

Figure 5.12: *XML* code of launch file.

In launch files it is possible instantiated predefined arguments or they can be passed as arguments. In the file of Figure 5.12 it can be observed the declaration of the input arguments, marked with *arg* tag, the launching of independent nodes, marked with *node* tag and also the launching of other launch files, marked with *include* tag. These included launch files are related to left and right arm set of nodes. The user can specify the input arguments through a command window console interface and then, when the system starts up, the prompt consoles of Figure 5.13 are shown.

Figure 5.13: Command window of openni_tracker (left) and cyton lauch (right).

The left side of Figure 5.13 represents the messages shown when it is launched the openni_tracker is lauched. It gives the information of the user index, the calibration phase and the confirmation that the user has been tracking. The right side of the image shows the messages shown when the Cyton launch file is lauched, which includes both left and right control. The control of each arm is controlled independently and, after appearing the previous command windows, two new command windows are shown to control left and right robotic arm as shown in Figure 5.14.

Figure 5.14: Command window to control left and right robotic arms.

Chapter 6

Results and Evaluations

The main objective of this chapter is to present experimental results and evaluate the performance of the implemented system facing different situations. The goal is to observe the performance of the robotic arms facing limit situations as well as its performance recovering from them. To do that some experimental tests were performed, each one targeting a specific situation and also a final demonstration where the goal is to show the performance of the system in a normal situation. All the results were evaluated calculating the mean square error, considering the difference between the desired and actual positions. It is important to mention that this error cannot be conclusive analytically since the trajectories are affected by a delay caused by the moving average filter. Thus, the error here is used to give an idea of the performance of the movements and also to observe its behavior facing difference situations.

6.1 Physical, Velocity and Workspace Limits

As already mentioned before, some problems can affect the performance during the imitation control. The results presented here are obtained for three different limit situations. The first is related to physical limits of the robotic arm joints. The principal limitation of the arm is related to the elbow pitch joint, which is very constrained. Figure 6.1 represents the performance of the left robotic arm executing a movement where the elbow pitch joint reaches its physical limits. It happened two times in this experiment, delimited by two vertical lines in Figure 6.1.

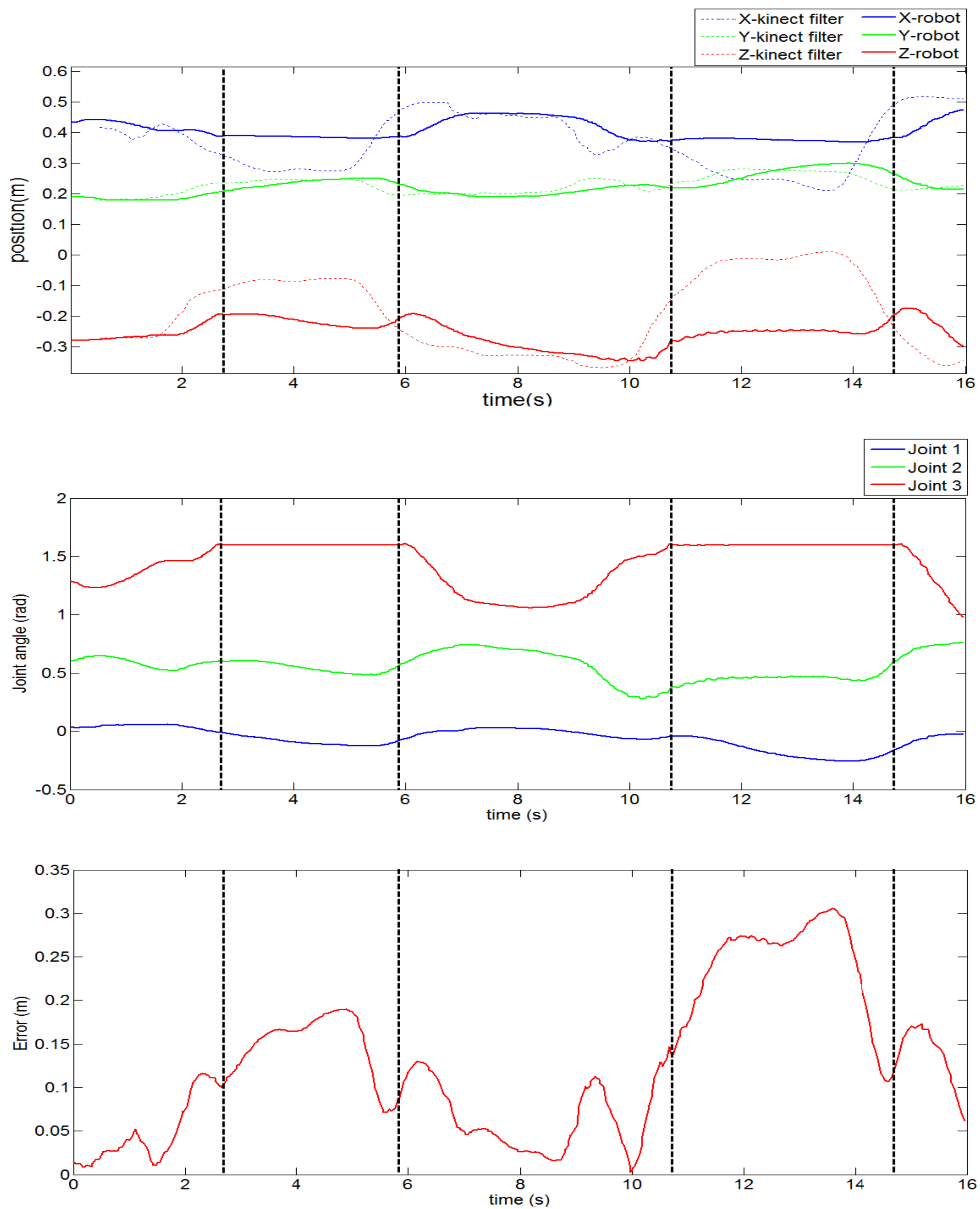


Figure 6.1: Graphical representation of the comparison of Cartesian position of human hand and end-effector (top), joint angles (middle) and the error amplitude (bottom) in function of time. The Cartesian position of the hand is already filtered. The vertical black lines represent the instants of time that a joint reaches its physical limit and the consequent rise of the error amplitude.

From the graphic it is possible to observe the performance of the tracking (top), the behavior of robotic arm joints (middle) and also the error between desired and actual position (bottom). There are two stages where joint 3 reaches its limits. When it happens the tracking starts to diverge and, as consequence, the error starts to rise up. It could be problematic if it was applied an integral controller that used accumulation error. As it was used just a proportional controller, which depends on the actual error, this situation had a smooth response in practice.

One of the predictable limits to happen during human gesture imitation was related to joint velocities. As the human arms are moved by muscles, its accelerations could be almost instantaneous and its velocities extremely large in a short period of time. And so, it was predictable that the robotic arm could not track correctly the human gesture movement during phases where the joint velocities limits were reached. To prove the purposed approach an experiment was executed where the main goal was to test this situation. Thus, the demonstrator performed a movement where, during two phases in this experiment, the operational velocities were high, as presented in Figure 6.2.

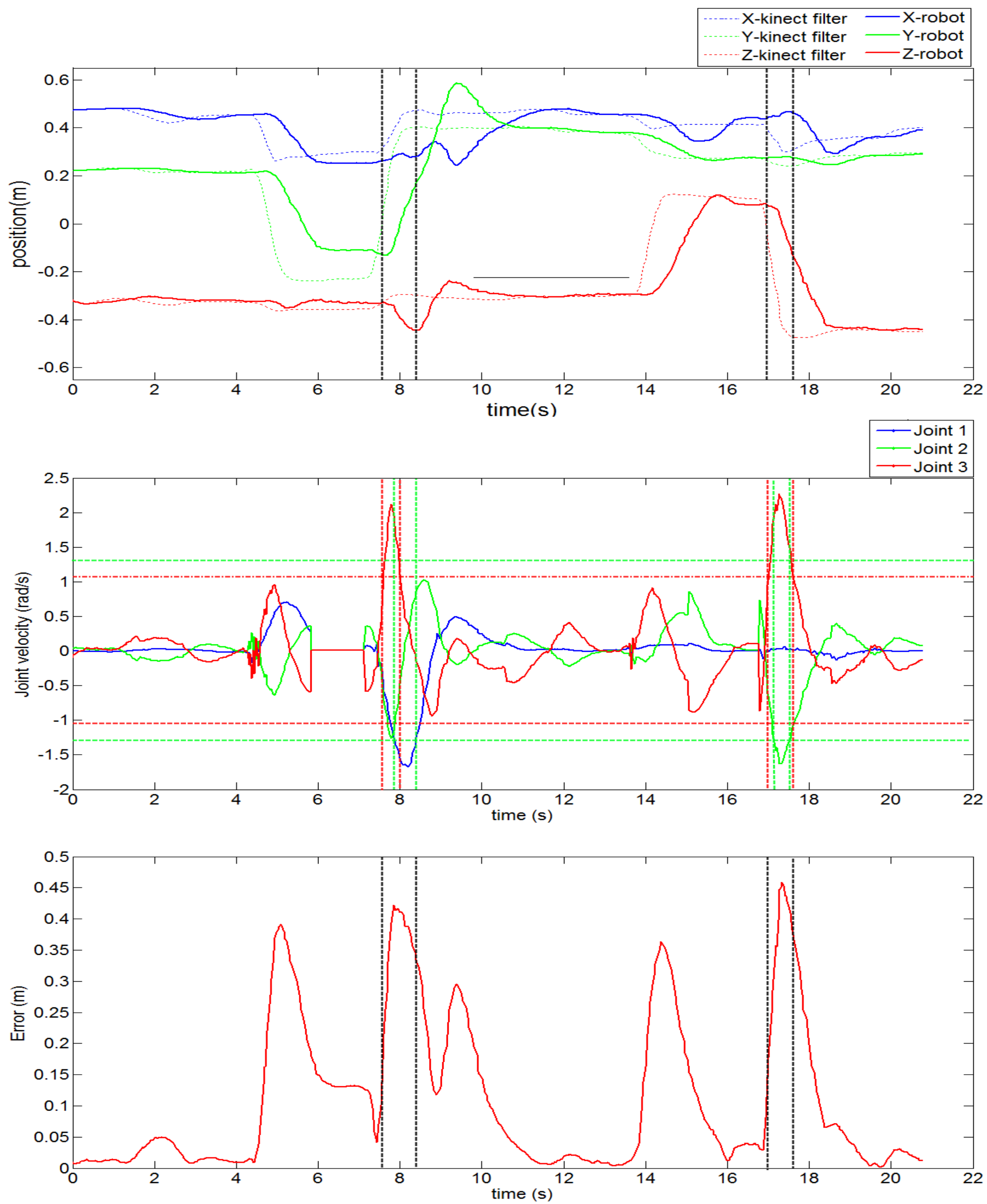


Figure 6.2: Graphical representation of the comparison of Cartesian position of human hand and end-effector (top), joint velocities (middle) and the error amplitude (bottom) in function of time. The Cartesian position of the hand is already filtered. The horizontal lines represents the joint velocities limits (red for joint 3 and yellow for joints 1 and 2) and the vertical lines represents the instant of time where the joints exceeded the velocity limits, consequently increasing the error amplitude.

It can be observed that the joint 3 reached its velocity limit two times during the movement (marked as red vertical lines in plot). Joint 1 also reached its velocity limits (marked as yellow vertical lines in plot). During these phases it is possible to see that, when joint velocities reach or are closed to their limits, the tracking starts to diverge and, consequently, the error between desired and actual positions increases. Although, when the human hand velocity produced joint velocities that were far within limits, the tracking starts to recover and the tracking starts to converge again.

The situations when the robotic arms reached their workspace limits were similar to the ones when they reached their physical limits. This experiment was intended to represent a situation where the human hand position represented a position outside the robotic system workspace in its own referential. Figure 6.3 represents the results obtained for an experiment that happened during the performed movement.

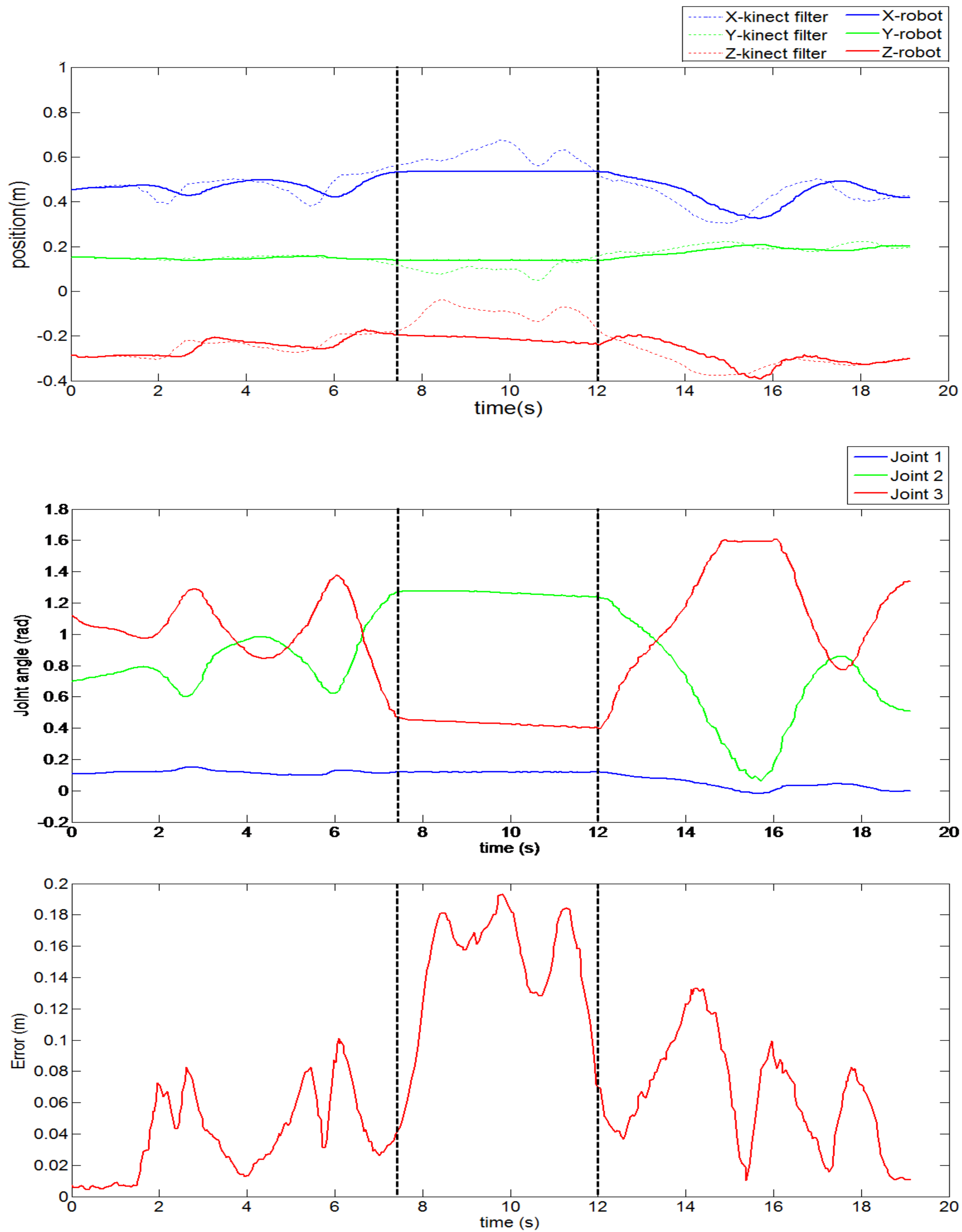


Figure 6.3: Graphical representation of the comparison of Cartesian position of human hand and end-effector (top), joint angles (middle) and the error amplitude (bottom) in function of time. The Cartesian position of the hand is already filtered. The vertical black lines represent the instants of time that the human hand exceeded the workspace limits of the robot and, consequently increasing the error amplitude.

Performing this experiment the workspace limit was reached during a period of time (represented as vertical lines in plot). During this phase, all joints remained still as shown in middle plot. As expected, the error between desired and actual position increased but, when the human hand returned inside operational workspace, the tracking was recovered. It is important to mention that, if the position of the human hand when it returns to operational workspace is too far from the end-effector position, the error will be larger and joint velocities can be larger too. To smooth this situation, when it happens, the proportional control constant was adjusted to a lower value. It smooths the movement when the error is too high.

6.2 Singularity Avoidance

This section has the objective of presenting a preliminary implementation of the singularity avoidance strategy purposed before. The practical experiments toke into account the following aspects:

- Some experiments were made in order to set a value of the cylinder radius that defines the singularity region, fixing a 15 cm value. Relatively to this region it is important to mention that this radius should be adjusted dynamically, depending on the end-effector velocity;
- It was assumed that, when the end-effector enters the singular region, the shoulder roll joint should be fixed when the difference between the new angle and the previous did not exceed 10 degrees. If the new angle was higher the shoulder roll joint adjusted to the desired angle. Thus, some jitter that occurs when the angle was always adjusted was eliminate and the quality of the movement improved relatively to the method of fixing the joint angle to the initial direction as long as the end-effector stayed in singularity region;
- To perform the movements inside the singularity region new inverse Jacobian, based on a 2 DOFs manipulator was used a. Here it was not applied any kind of feedback control since for the pretended implementation was not a priority and also because the singularity region is not large enough to affect the error too much.

Hereupon the practical experiments proceeded to test this strategy. The experiment consisted of starting off the singularity region, entering it and then returning to the outside adjusting the shoulder roll twice (in entrance and when the desired trajectory imposed angle was higher than 10 degrees). Figure 6.4 represents the top view of the left human hand and end-effector of the right robotic arm trajectory in Cartesian space. It is possible to observe the two trajectories diverging inside the singularity region (marked with a dotted cylinder) due to the fact that the shoulder joint was fixed, loosing one DOF, and to the fact that no kind of feedback control was used.

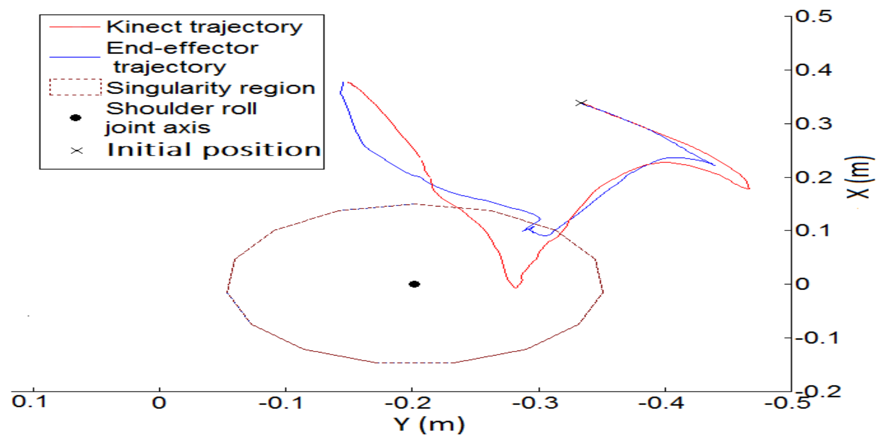


Figure 6.4: Top view representation of the left human hand and right end-effector trajectory passing through the singularity region (dotted cylinder), starting from the asterisk position.

In this experiment it is also interesting to observe the behavior of the joint angles during time, as shown in Figure 6.5. Here can be seen the period of time where the end-effector passed through the singularity region (period between the two vertical black dotted lines). It is possible to observe joint 1 adjusting when the end-effector entered the singularity region and also some instants before it left the region.

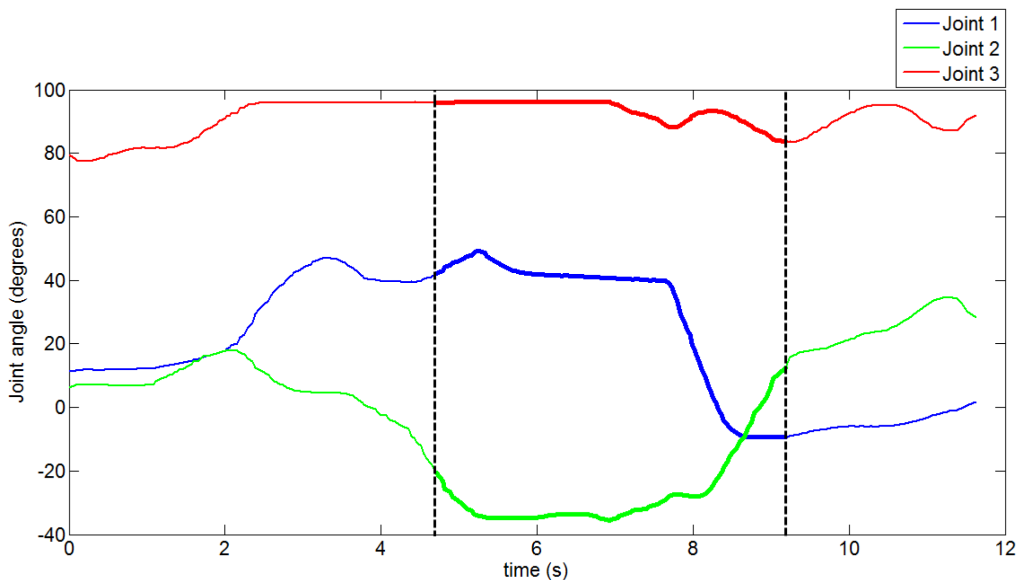


Figure 6.5: Joint angles during the experiment when the end-effector passed through the singularity region (between the two vertical black dotted lines).

The used strategy to avoid singularities was not deeply tested and it did not solve the problem for all situations. It was noticed that the movement inside is not as accurate as

outside the singularity region, possibly due to the fact that no kind of feedback control was used. It is important to refer that it was a preliminary strategy and that it was not deeply tested but it seems to work and solve this problem, just needing to be adjusted and tested to every possible situation.

6.3 Self Collision Detection

The system was also projected with a self collision detection mechanism. This mechanism allows detecting and acting when both robotic arms are in collision with each other or with torso structure. In order to test and evaluate this mechanism an experiment was performed where both arms were against the torso and then in the opposite direction. In it both right and left arms were used, testing the collision detection using each one against the torso. Figure 6.6 represents a Matlab simulation using experimental data acquired during this experiment. The goal is to show the behavior of the mechanism by representing the predefined bounding boxes (unfilled boxes in figure) during the movement.

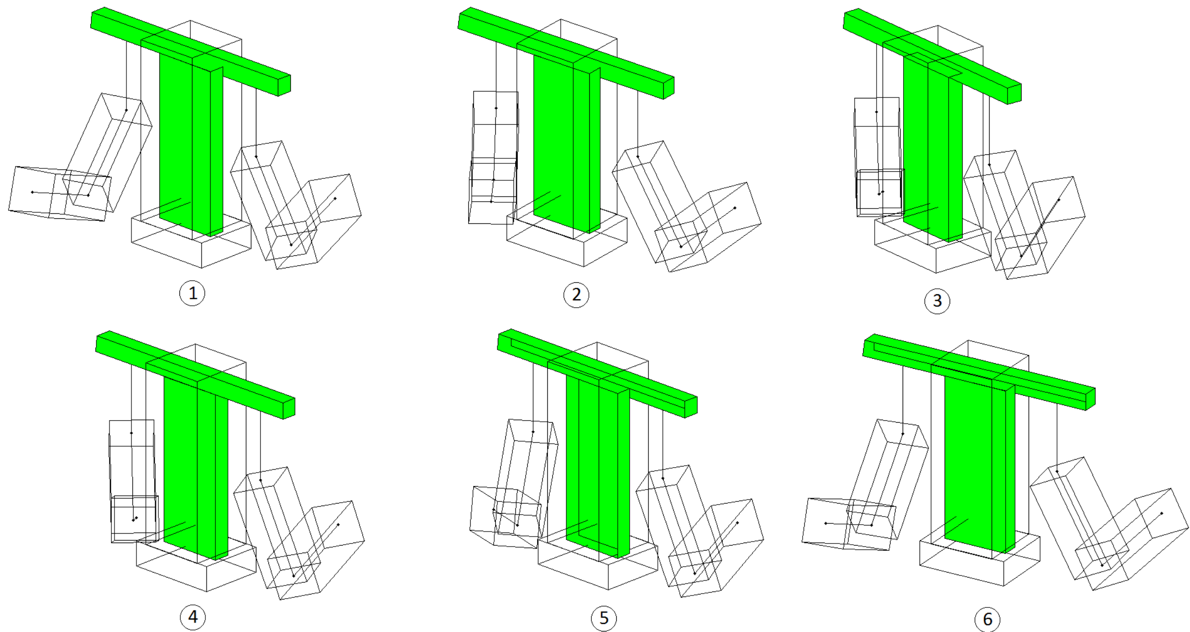


Figure 6.6: Representation of a Matlab simulation of self collision detection using experimental data.

Figure 6.6 represents the first stage of the experiment where the left robotic arm was positioned away from torso by the demonstrator and, simultaneously, the right robotic arm was moved against the torso and then moved away from the torso. The subsequent movement was similar to the one represented in Figure 6.6 but performed using the left robotic arm. In Figure 6.7 are plotted the trajectory (top) executed by both left and right human and robotic arms during all the experiment as well as its joint behavior (bottom).

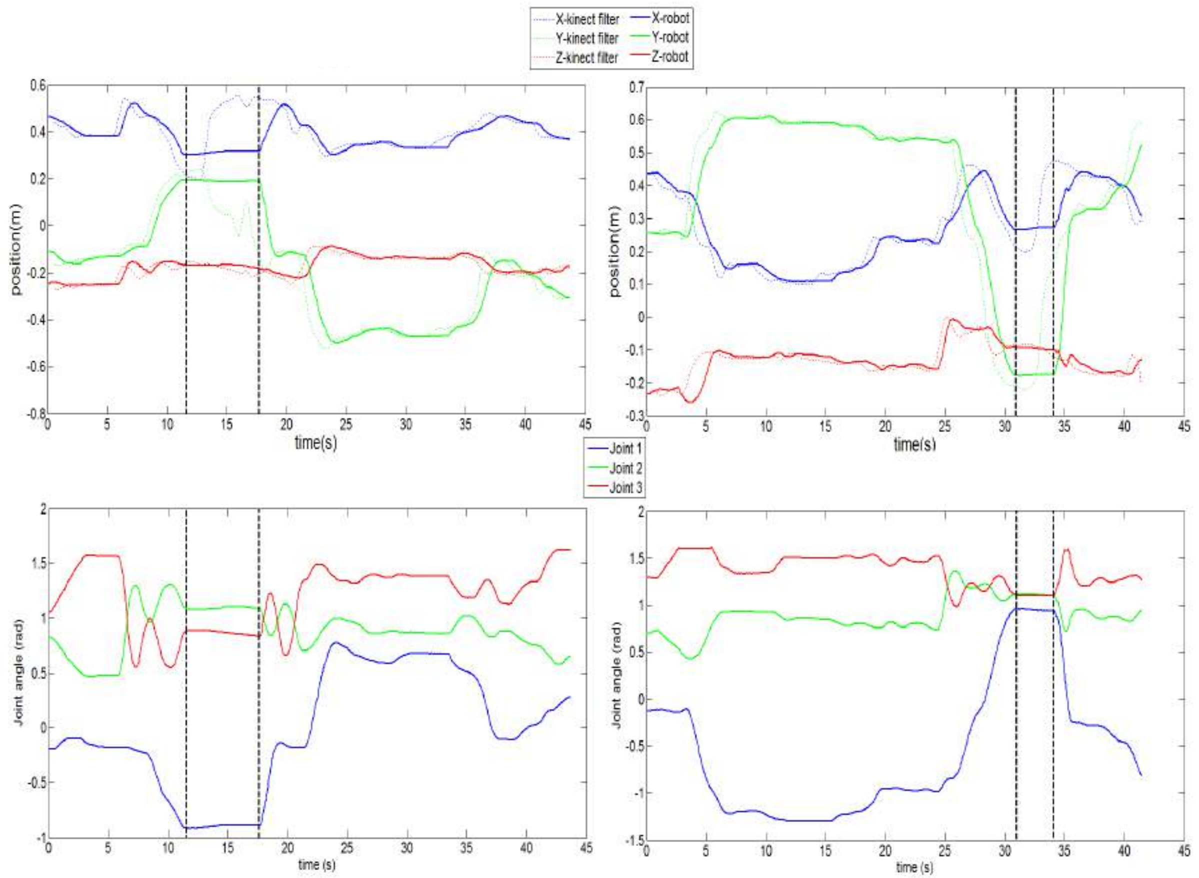


Figure 6.7: Graphical representation of the comparison of Cartesian position of the human hand and end-effector (top) and joint angles (bottom). The left plots represent the position of the left human hand and right end-effector as well as the joint angles of the right robotic arm. The right plots represent the right human hand and left end-effector as well as the joint angles of the left robotic arm. The vertical lines represent the time instant when collisions occurred.

The graphic represents the movement (top) and also the joint angles (bottom) of both left human hand movement (left) and right human hand movement (right). As can be seen, all joint of each robotic arm remain still when the collision is detected (marked as vertical lines in plot). During this period of time the end-effector position remains still as long as the human hand position is still inside the collision zone. As mentioned before, when the collision is detected, the position of the robotic arm is estimated by the actual human hand position so it can be possible to know if the entire arm is still in collision or not. When the estimated position gets out of the collision zone the robotic arm recovers the human hand tracking. It is possible to observe that the response time is reasonable. However, it was possible to conclude that the mechanism was not robust enough when high velocities were reached. It could be solved by adjusting the bounding boxes according to actual velocities of the robotic arms.

After executing some experiments in order to test and evaluate this mechanism it was possible to conclude that the collision between both robotic arms was not reliable due to the limitations of the Kinect sensor. It is related to occlusions of human joints when both arms are overlapping.

6.4 Final Demonstration

During the previous sections the main objective was to show the performance of the system facing different problem situations. This section goal is to show the behavior of the robotic system in a demonstration where the demonstrator tried to execute movements naturally. The execution of this demonstration was intended to prove and test the system in a common usage and also to show some practical pictures captured during the experiment. Figure6.8 represents some frames captured during the final demonstration[1] and they are sorted from top to bottom, left column first and then right column.

[1]Full video: https://www.youtube.com/watch?v=WTaToIOV49I&feature=youtu.be

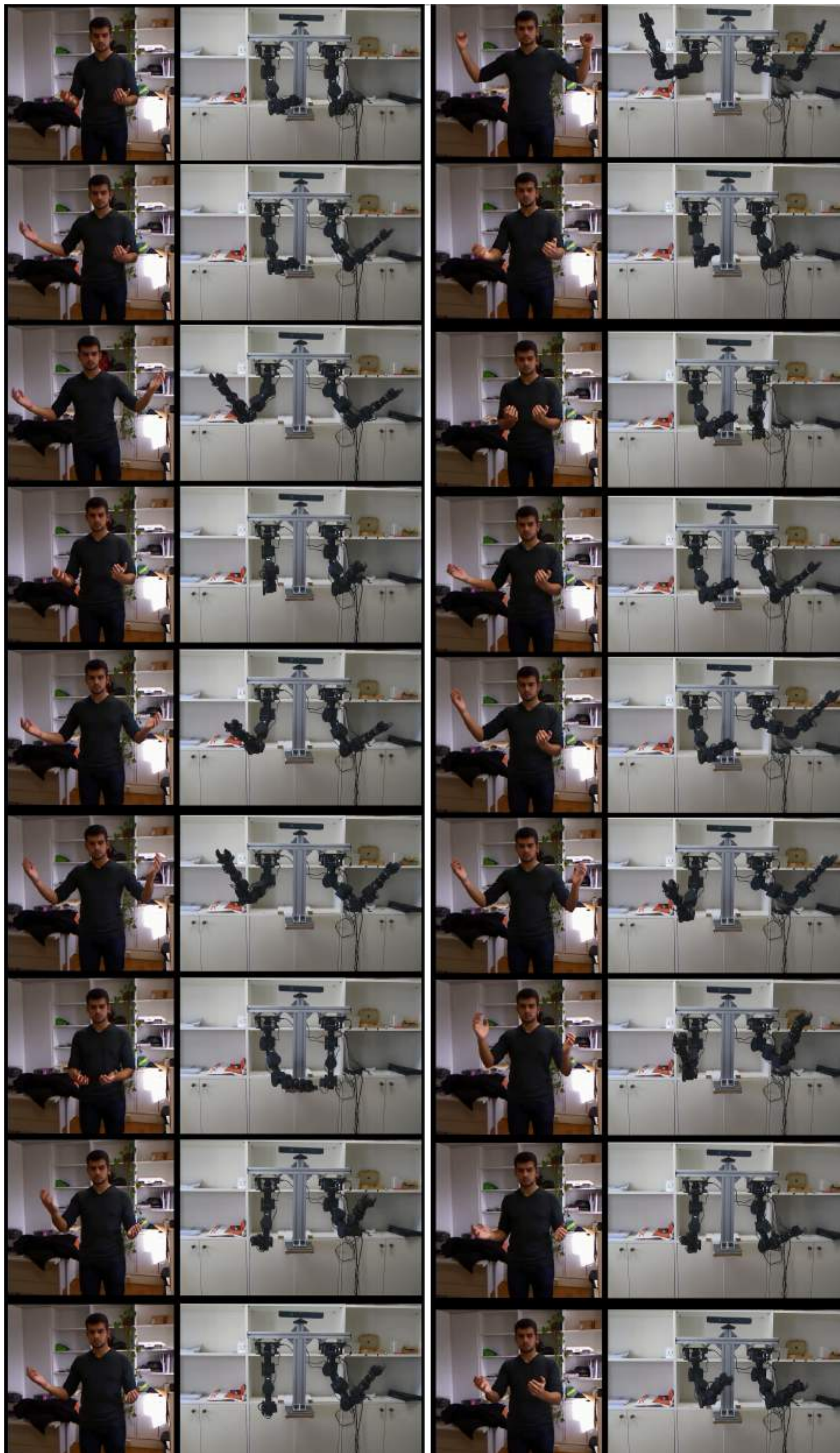


Figure 6.8: Human demonstrator imitation acquisitions. Sequence from top to bottom, left column first and then right column.

These frames just represent a visual perception of the experiment. From it, it is possible to observe that the robotic arms execute the human gesture imitation, mirroring its movement. The strategy used for the motion capture allows the demonstrator to move around the scene during the imitation.

In order to show a more detailed information of the imitation performance two graphics were plotted, as shown in Figure 6.9 and Figure 6.10, representing the path executed by human hand overlaid with the path performed by end-effector of both left and right robotic arm, respectively.

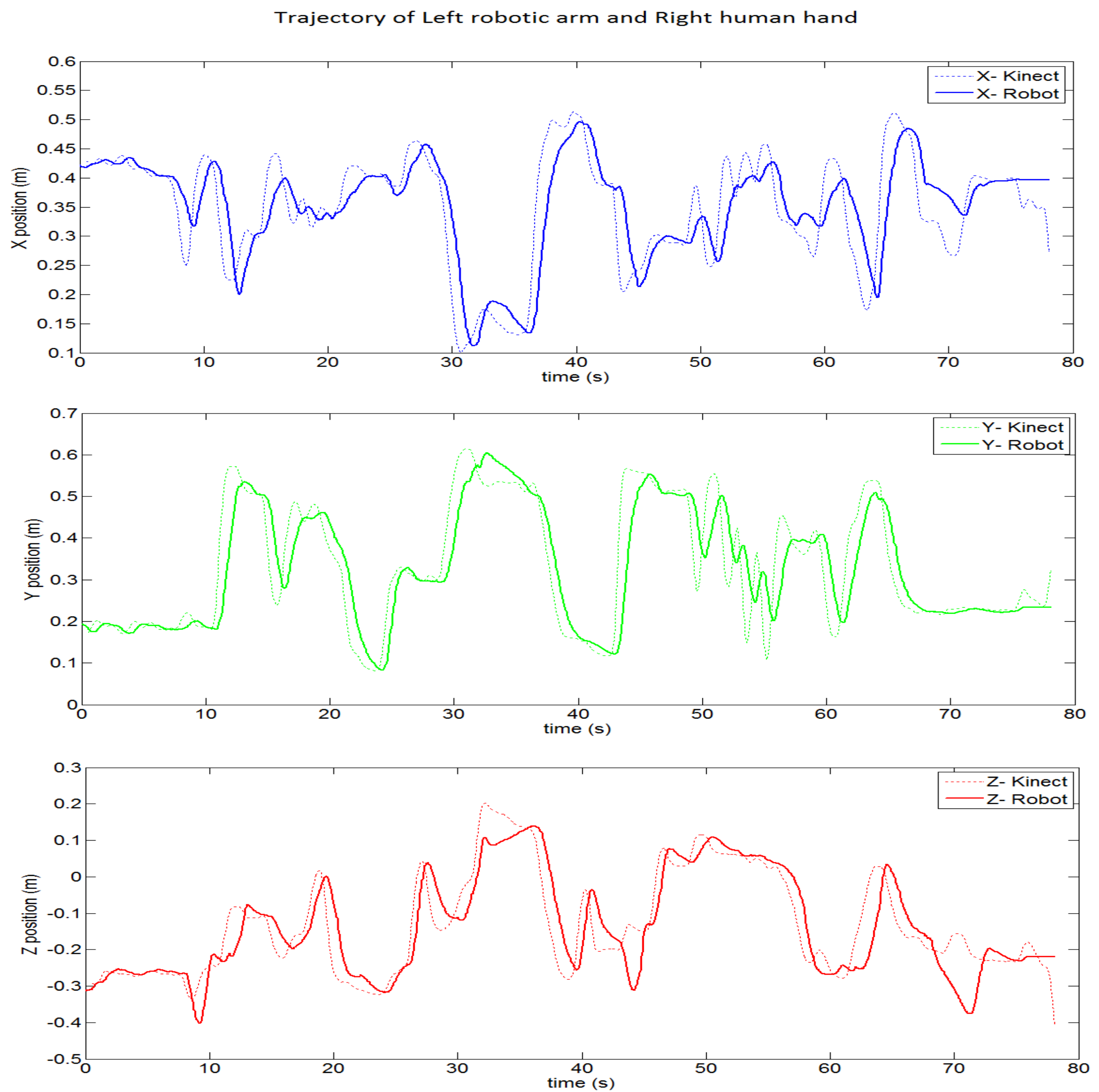


Figure 6.9: Comparison between left end-effector and right human hand trajectories in x-axis (top), y-axis (middle) and z-axis (bottom).

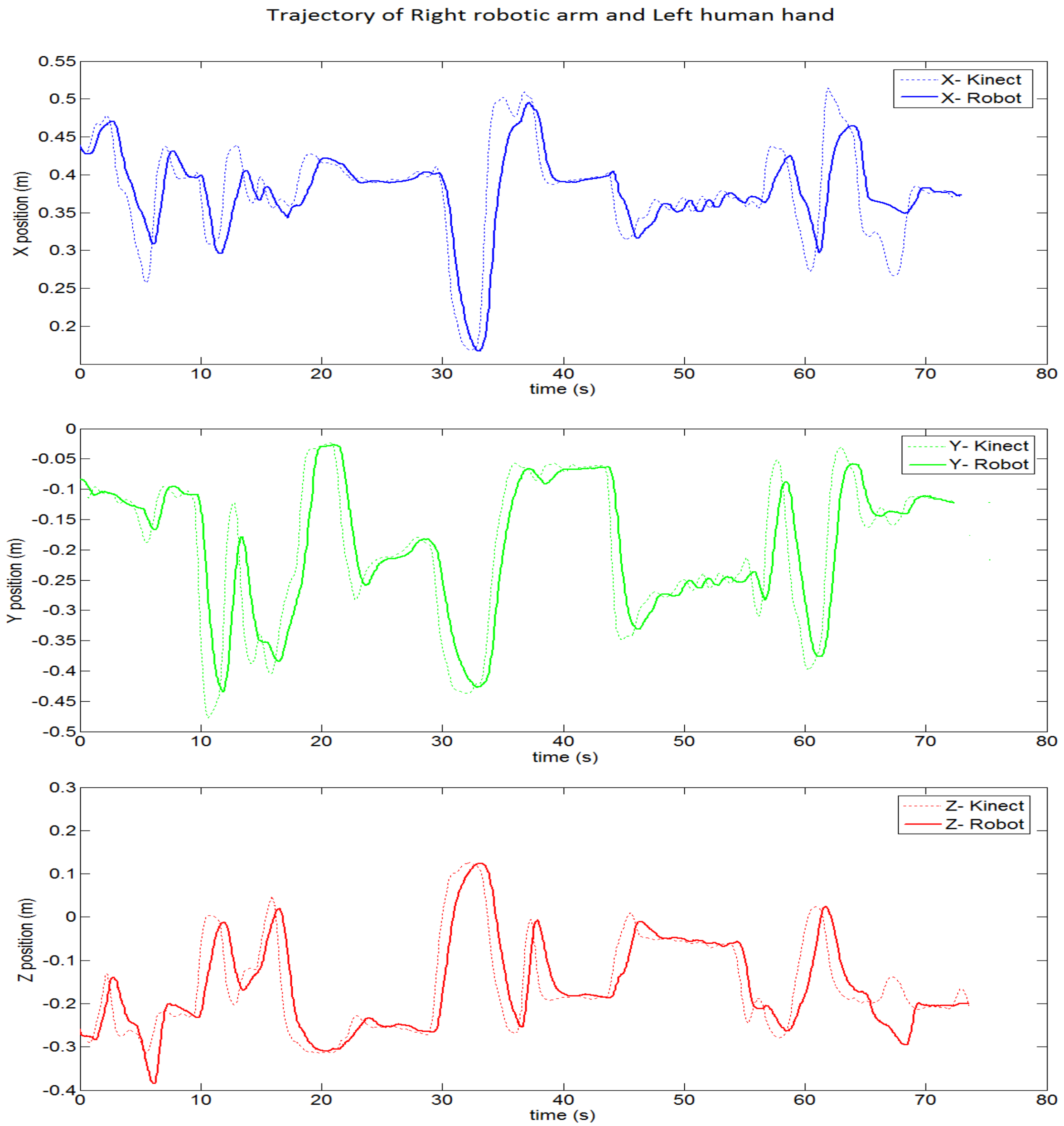


Figure 6.10: Comparison between right end-effector and left human hand trajectories in x-axis (top), y-axis (middle) and z-axis (bottom).

The adopted strategy imitated the human hand trajectory regardless the position of the robotic elbow relatively to the human elbow. The DOFs of the robotic arm were reduced to 3 and the elbow joint angle was also limited to take just positive values in order to approximate the human and robot physiognomy. The graphics of Figure 6.11 and Figure 6.12 represent a comparison between the robotic and human elbows trajectories during the experiment for left and right robotic arm, respectively.

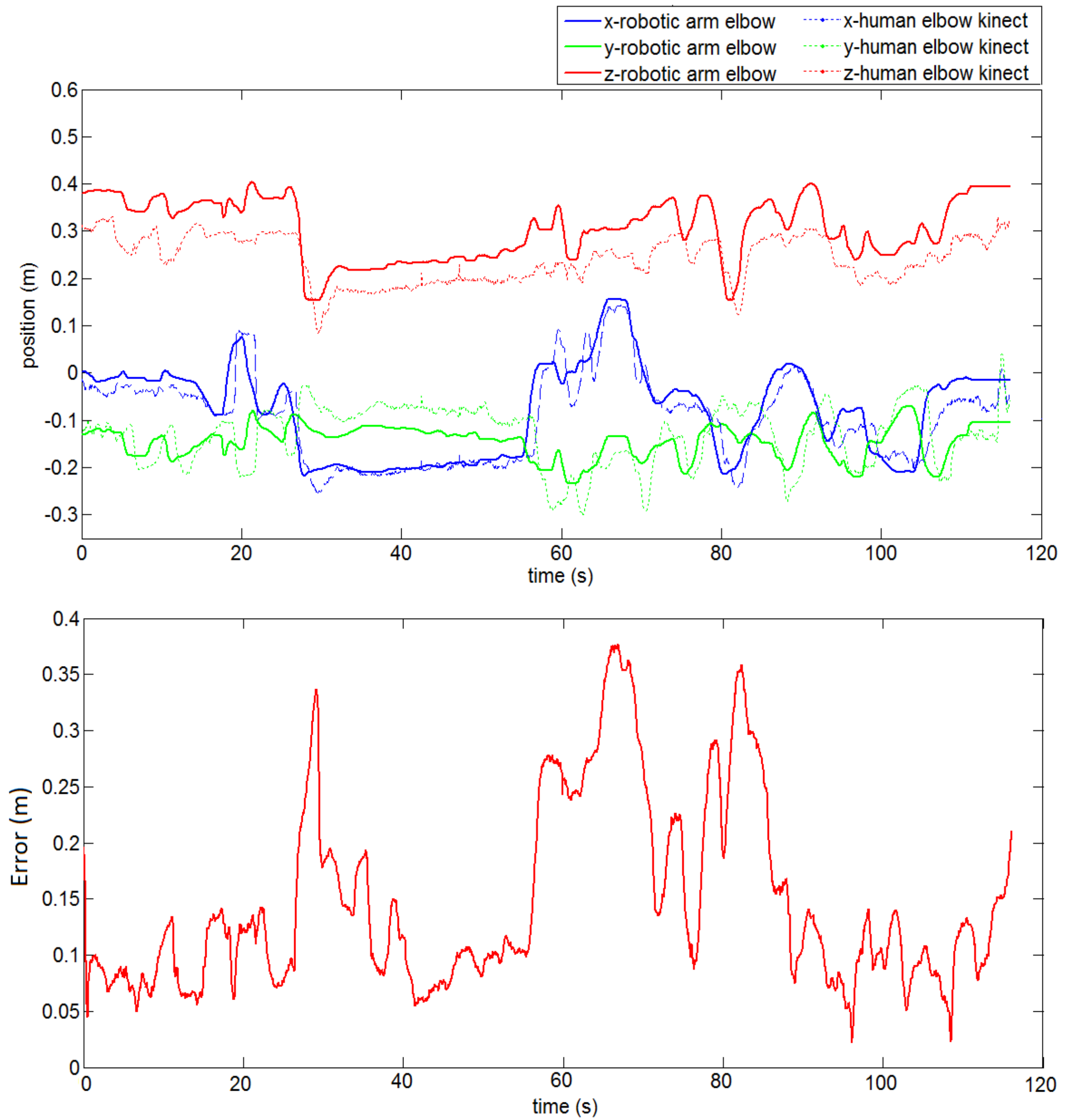


Figure 6.11: Comparison between the left robotic elbow and right human trajectories (top) and respective error (bottom).

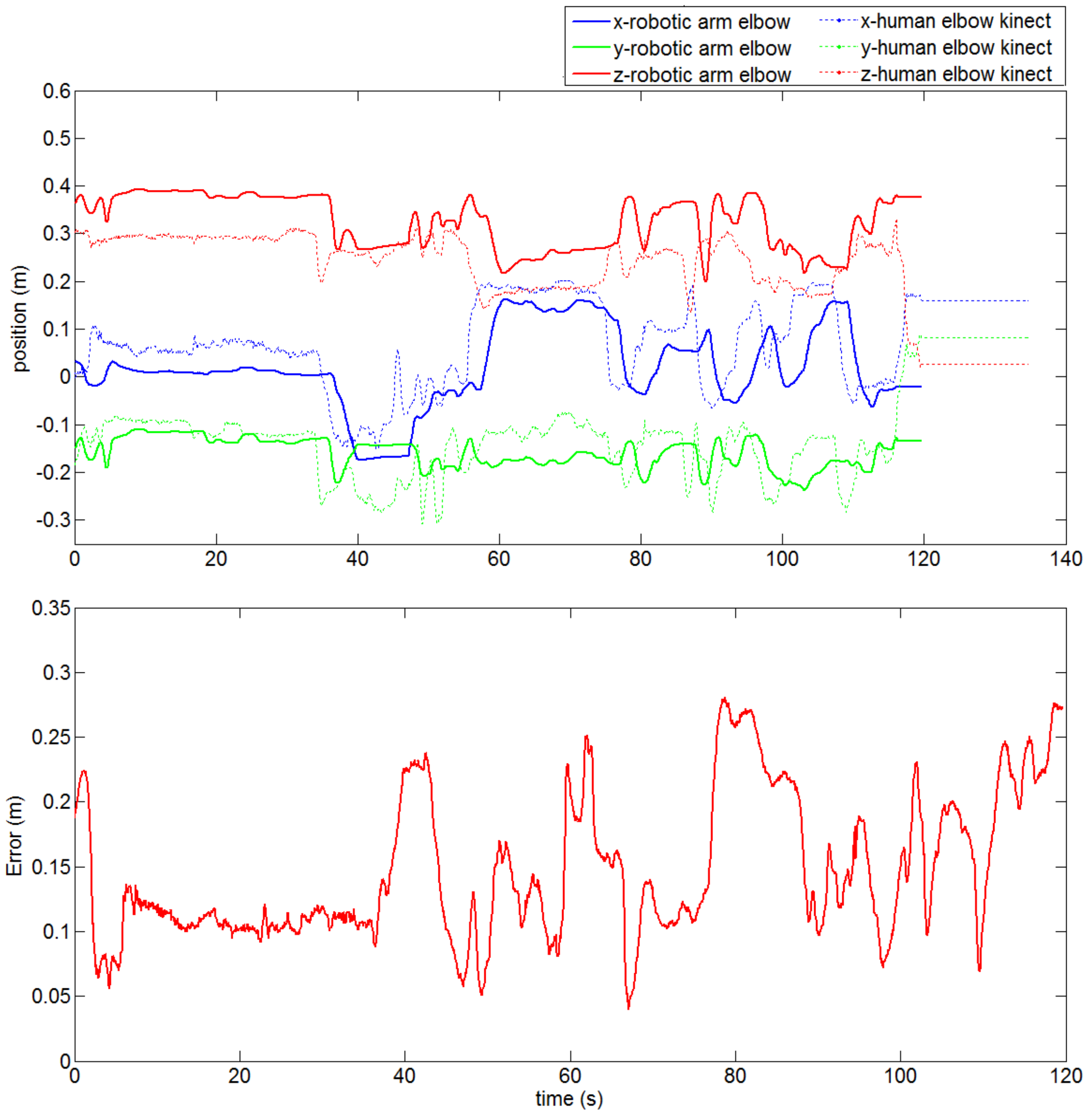


Figure 6.12: Comparison between the right robotic elbow and left human trajectories (top) and respective error (bottom).

From the plots it is possible to observe that the trajectory executed by the robotic elbow did not match perfectly the one executed by the human elbow. Although, taking into account that nothing was done to force the elbow tracking the results are acceptable.

Chapter 7

Conclusions

This final chapter is dedicated to discuss the significance of the main results achieved, and to present the final conclusions of the work and perspectives of future developments.

7.1 Results Discussion

The main objective of this work was to develop computational algorithms to control a humanoid platform in order to execute human gesture imitation. The work was divided into three fundamental parts: first, the need of a structure to create the system and to execute the experiments; second, the need of low level control of the robotic arms that allow to control them using different approaches using just the 3 DOFs; and finally the development of a software architecture that allows to execute human gesture imitation.

Structural design required a brief study of system workspace and a study of anthropomorphic relations. The final result worked well during the experiments and proved to be robust enough to hold both robotic arms even when they reached large velocities.

During the development of low level motion control software was noticed that ROS package, provided by Cyton to control the robotic arms, had some limitations and flaws. The first one was related to the process the function to send commands to the robotic arm. It had not implemented the functionality to change the desired joint velocity. If another process sent a velocity command it would assume a default one. Thus, it was implemented in that process the functionally to send joint velocities command to the arm using low level Cyton Hardware API functions. The second problem was related to the actual joint velocities executed by the robotic arm. The specified joint velocity sent to the robotic arm did not match with the actual velocity. In order to solve that, a relation between the input velocity and real velocity was determined so it could be possible to execute the desired movements. A final problem was related with hardware communication. The low level Cyton Hardware API has a function which allows specifying the path to a *XML* file, where the *USB* port where the robotic arm is attached can be specified. Thus, the idea was to create two different *XML* files, each one specifying the USB port of the correspondent robotic arm and when the function was called

it selected the correct file. However, it did not work and the function assumed always the default path. To solve this problem a process was developed responsible to replace the default *XML* file for a predefined one with a specific USB port each time that it was necessary to initiate the hardware. This process receives the USB port number as argument which needs to be specified by the user.

All processes are launched using a launch file. It allows to launch many processes at the same time with a single command and to remap the nodes and topic names so it is possible to use the same processes to control both arms independently. The launch command launches all processes at same time, or at least it is not known an order of launching. It can introduce a limitation to the strategy presented before to change the XML files. If the XML file changing does not happen in the right order, the hardware initialization will fail and it is needed to re-launch the system. This situation happened rarely during the experiments but it is an aspect to take into account in future works using this system.

During the experiments were also noticed some limitations of the Kinect sensor. First, when a demonstrator is executing a movement in front of the sensor joint occlusions can occur, which can result in inappropriate data. It introduced some limitations during the self collision experiments between the two robotic arms. When there is collision between both arms the Kinect sensor might confuse them and publish inappropriate joint values. This situation was not exhaustively tested and it was not studied the Kinect data in those situations, which could confirm and evaluate the data form the sensor during joint occlusions. Another problem while using the Kinect sensor was related to the openni_tracker package. It provides the skeleton data but also identifies the subject with an index number. If there is more than one subject in scene or the same subject leaves and returns to the scene again a new index might be assigned . It is hard to solve this problem because the index value is not provided by openni_tracker; thus, it is not easy to know which subject to imitate. Also related to this, if the system is performing imitation and the subject leaves the scene, the response will be unpredictable since the openni_tracker can continue to send inappropriate joint values and it is not easy to know if the imitation should continue or not. Kinect sensor also has distance limitations, the subject cannot operate further than about 4 meters away form the sensor and also not closer than about 80cm.

The experiments showed some results obtained for different situations. Those resulted as consequence of limitations of robotic arms and imitation problems. The results show that the system can react fairly well when the robotic arms reach their physical, velocity and workspace limits. The implemented solution was always based on recovering from the limit situation using the same control as during gesture imitation. As described before this can result in large velocities in some cases due to the fact of, in some situations, the error can be high when the arm returns from the limits. It could be solved using other control method, depending on the limit situation, such as position control to guide the robotic arm to the human hand position and then start the imitation again.

It was also implemented a self collision detection to prevent possible damage of the system.

The results shows that the implemented strategy worked in arm-torso collisions but did not work well in arm-to-arm collisions due to the occlusion problem described before. This strategy also did not work well with large velocities where the system cannot react fast enough to stop the robotic arms before they collide with torso. This problem could be solved by adjusting the bounding boxes created around each arm and torso depending on the arm velocity. There are other existing solutions that could be considered to execute the collision detection. *MoveiT!* is a software for mobile manipulation able to do motion planning, manipulation, 3D perception, kinematics, control and navigation. It also allows to checking self collision and collision with other objects in space, although it just runs in ROS Hydro and Indigo versions, which are more recent than ROS Fuerte used during this project. It did not seem viable to change ROS version and it was not found easy software to use collision detection so it was decided to create the algorithm previously described based on bounding boxes and line to line intersection.

7.2 Final Conclusions

Globally, the results obtained are very satisfactory and relevant, even if some refinements, improvements and extensions are required to improve the system's performance. Having this in mind, the following conclusions can be drawn:

1. The mechanical design of the torso structure and the options adopted proved suitable for the concrete application of gesture imitation.

2. The implementation of the different control modes provides the required level of functionality. In particular, the closed-loop inverse kinematics method proved to be robust for real-time tracking of the target trajectories.

3. The extension of the motion control algorithm in order to deal with physical constraints, self-collisions and singularities ensures a more natural and efficient human-robot interface.

4. The software architecture based on ROS offered several desired features in terms of software development and it proved its added value.

5. The global performance of the dual-arm system reproducing the human gestures is as expected, taking into account the specific limitations of both the robot arms and the motion capture system.

7.3 Future Work

During the execution of this project was noticed that the system could be improved by addressing to new features or improving the implemented ones. Given the current state of development, the perspectives of future work point in the following directions:

1. Migrating from ROS Fuerte version to a new one, which can provide more features and a better performance;

2. Using the later version of Kinect sensor and/or OpenNI drivers can improve the quality of the acquired data. The new Kinect sensor has improved body, hand and joint orientation, which can be useful to execute a better imitation of the human gesture;

3. Developing a strategy to initiate the imitation tracking in a more practical way, for instance using voice commands or a predefined gesture;

4. Creating a new graphical interface or implementing the developed software along with Actin simulator. It provides C++ plugins, which can ease the implementation, and can be useful to observe what is happening in real time. This simulator has collision detection strategies as well and built-in inverse kinematics algorithms;

5. Combining the human gesture imitation along with tele-operation can be very useful to control humanoids regardless the distance using TCP/IP sockets to transfer the data from the motion capture system to the computer unit responsible to control the robot;

6. Defining metrics to evaluate the human gesture and creating learning methods through imitation. It can be very powerful to transfer new skills to the robot in a more natural and intuitive way.

References

Aris Alissandrakis, Chrystopher Nehaniv, and Kerstin Dautenhahn. Correspondence Mapping Induced State and Action Metrics for Robot Imitation. *IEEE Transactions on systems,man, nad cybernetics - part B cybernetics*, 37(2):299 to 307, 2007.

Brenna Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 to 483, 2009.

T. Asfour, K. Regenstein, P. Azad, J. Schroder, A. Bierbaum, R. Dillmann, and N. Vahrenkamp. ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. *IEEE-RAS International Conference on Humanoid Robots*, page 169 to 175, 2006a.

Tamim Asfour, Florian Gyarfas, Rudiger Dillmann, and Pedram Azad. Imitation Learning of Dual-Arm Manipulation Tasks in Humanoid Robots. *IEEE-RAS International Conference on Humanoid Robots*, page 40 to 47, 2006b.

Christopher Atkeson and Stefan Schaal. Learning tasks form a single demonstration. *IEEE International Conference on Robotics and Automation*, 2:1706 to 1712, 1997.

Mattia Avancini. *Using Kinect to emulate an Interactive Whiteboard.* PhD thesis, University of Trento, 2012.

G Bekey and P Sanz. Robotics: State of the Art and Future Challenges. *Robotics & Automation Magazine, IEEE*, 16(1):116, 2009.

A Billard and D Grollman. Robot learning by demonstration. *Scholarpedia*, 8(12):3824, 2013.

Aude Billard. Learning motor skills by imitation: A biologically inspired robotic model. *Cybernetics and Systems*, 32:155 to 193, 2001.

Aude Billard, Sylvan Calinon, Ruediger Dillmann, and Stefan Schaal. Robot Programming by Demonstration. In *Handbook of Robotics*, chapter 59. 2007.

Depth Biomechanics. How The Kinect Works. URL `http://www.depthbiomechanics.co.uk/?p=100`.

Christoph Borst, Thomas Wimbock, Florian Schmidt, Matthias Fuchs, Bernhard Brunner, Franziska Zacharias, Giordano, Paolo Robuffo, Konietschke, Rainer Wolfgang, Sepp Fuchs, Stefan Christian Rink, Alin Albu-Schaffer, and Gerd Hirzinger. Rolling Justin - Mobile Platform with Variable Base. *IEEE Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 1597 – 1598, 2009.

Samuel R. Buss. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods. 2009.

John J. Craig. *Introduction to Robotics Mechanics and Control*. Third edition, 2005.

A P del Pobil, A J Duran, M Antonelli, J Felip, A Morales, M Prats, and E. Chinellato. Integration of Visuomotor Learning, Cognitive Grasping and Sensor-Based Physical Interaction in the UJI Humanoid Torso. *Designing Intelligent Robots: Reintegrating AI II*, 2013.

M.A. Diftler, J.S. Mehling, M.E. Abdallah, N.A. Radford, L.B. Bridgwater, A.M. Sanders, R.S. Askew, D.M. Linn, J.D. Yamokoski, F.A. Permenter, B.K. Hargrave, R. Platt, R.T. Savely, and R.O. Ambrose. Robonaut 2: The First Humanoid Robot in Space. 2011.

José Dinis. Manual de captura de movimentos usando o sistema óptico da Vicon e o software iQ. Technical report, Laboratório MovLab, Universidade Lusófona, 2011.

R Drillis, R Contini, and M Bluestein. Body Segment Parameters; a Survey of Measurement Techniques. *Artificial limbs*, 25:44 to 66, 1964. ISSN 0004-3729. URL `http://www.ncbi.nlm.nih.gov/pubmed/14208177`.

Ignacy Duleba and Michal Opalka. A Comparison of Jacobian-Based Methods of Inverse Kinematics for Serial Robot Manipulators. *Int. J. Appl. Math. Comput. Sci*, 23(2):373 to 382, 2013.

Peter Englert, Alexandros Paraschos, Jan Peters, and Marc Deisenroth. Addressing the correspondence Problem by Model-based Imitation Learning. *ICRA Workshop on Autonomous Learning*, 2013.

A Hermann, Zhixing Xue, S.W Ruhl, and R Dillmann. Hardware and software architecture of a bimanual mobile manipulator for industrial application. *Robotics and Biomimetics (ROBIO)*, page 2282 to 2288, 2011.

Petar Kormushev, Sylvain Calinon, and Darwin Caldwell. Reinforcement Learning in Robotics: Applications and Real-World Challenges. *Robotics*, 2(3):122–148, July 2013. ISSN 2218-6581. doi: 10.3390/robotics2030122. URL `http://www.mdpi.com/2218-6581/2/3/122`.

Baris Kurt. *Imitation of human arm movements by a humanoid robot using monocular vision.* 2005.

Metamotion. Gypsy Motion Capture System Workflow, 2012. URL `http://www.metamotion.com/gypsy/gypsy-motion-capture-system-workflow.htm`.

Vítor M.F. Santos. *Robótica Industrial*. Aveiro, 2004.

Rob Acronius Miedema. Improve motion capturing by using a movie camera and new markers. Technical report, Centro de Technologia da Informação Renato Archer, Campinas, 2010.

Chrystopher Nehaniv and Kerstin Dautenhahn. The correspondence problem. In *Imitation in Animals and Artifacts*, chapter chapter 2. 2002.

Pedro Nogueira. Motion Capture Fundamentals: A Critical and Comparative Analysis on Real-World Applications. 2011.

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3:233–242, 1999.

Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational Approaches to Motor Learning by Imitation. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences*, 358(1431):537–547, 2003.

Lorenzo Sciavicco and Bruno Siciliano. *Modeling and Control of Robot Manipulators*. Naples, 1996. ISBN 0-07-114726-8.

Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. 2011.

B. Siciliano and O. Khatib. Springer Handbook of Robotics. *Robotics & Automation Magazine, IEEE*, page 110, 2008.

Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V. Dimarogonas, and Danica Kragic. Dual Arm Manipulation - a survey. *Robotics and Autonomous Systems*, 60(10):1340 to 1353, 2012.

Keng Peng Tee, Rui Yan, Yuanwei Chua, and Zhiyong Huang. Singularity-Robust Modular Inverse Kinematics for Robotic Gesture Imitation. In *IEEE International Conference on Robotics and Biomimetics*, pages 14–18, China, 2010.

Stephen Warren and Panagiotis Artemiadis. On the Control of Human-Robot Bi-Manual Manipulation. *Journal of Intelligent & Robotic Systems*, 2014.

Stefan Weber, Maja J Mataric, and Odest Chadwicke Jenkins. Experiments in Imitation Using PerceptuoMotor Primitives. *Autonomous Agents*, pages 136–137, 2000.

Franziska Zacharias, Daniel Leidner, Florian Schmidt, Christoph Borst, and Gerd Hirzinger. Exploiting Structure in Two-armed Manipulation Tasks for Humanoid Robots. 2010.

Appendix A

Cyton hardware API public functions

Table A.1: Public member functions of hardwareInterface class.

Function	Description	Parameters	Returns
hardwareInterface (const EcString &pluginName, const EcString &configFile="")	Constructor. Does not initialize hardware.	**pluginName** Name of hardware plugin to utilize **configFile** Optional hardware configuration file	
~hardwareInterface ()	Destructor. Shuts down device driver if loaded.		**EcStringVector** A vector of string representing the port names of the devices available. Platform dependent. Empty list returned if not available, or plugin not loaded.
setPort (const EcString &port)	Specify a port to use for the connection to the hardware.	**port** String name of port to use. Platform dependent	
availablePorts () const	Examine current hardware configuration to list available ports.		**EcStringVector** A vector of string representing the port names of the devices available. Platform dependent. Empty list returned if not available, or plugin not loaded.
setInitAndShutdownMode (const InitAndShutdownMode mode)	Override config defaults and specify a new set of init and shutdown options. Options are bit-ORed together.	**mode** Specified override parameters	
initAndShutdownMode () const	Accessor to retrieve state of init and shutdown parameters.		**InitAndShutdownMode** ORed bitfields of modes
init ()	Initialize hardware, which includes reading in configuration file, opening the port and resetting hardware to a known good state.		**EcBoolean** Success or failure of initialization
reset ()	Send a reset command to the hardware to move joints back to resting position.		**EcBoolean** Success or failure of reset command
shutdown ()	Unloads plugin device driver.		**EcBoolean** Success or failure of shutdown command
setTorque (const EcBoolean enabled)	Controls overall torque control. When disabled, servo motors will be free-spinning. Use with caution as this may damage hardware if not supported.	**enabled** Desired torque state	**EcBoolean** Success or failure of set command
setVelocityFromDeltaTime (const EcReal deltaT)	This method is used to set a time difference that will be used when subsequent setJointCommands calls are made to calculate an appropriate set of joint velocities. The time given also represents the amount of time (approximate) that the hardware will take to reach the desired joint state.	**deltaT** Time difference used to calculate joint rates	
convertUnits (const EcRealVector &jointAngles, StateType stateType) const	Performs unit conversion from the specified type and returns the values in radians.	**jointAngles** Input angles to process **stateType** Format that the input angles are in	**EcBoolean** Success or failure of set command

setJointCommands (const EcRealVector &jointCommands, const EcRealVector &jointVelocities=EcRealVector())	Sends commands to Cyton hardware to move joints to a specified location. If the joint velocities are not specified, then the velocity values will be calculated based upon the previous call to setVelocityFromDeltaTime(). All input values are in radians (and radians-per-degree). In the case where the delta time has never been explicitly set, the value of 20 seconds will be used.	**jointCommands** Vector of joint angles to move servos to **jointVelocities** (Optional) Vector of joint velocities	**EcBoolean** Success or failure of set command
setJointCommands (const EcReal timeNow, const EcRealVector &jointCommands, const StateType stateType=JointAngleInRadians)	DEPRECATED This command has been replaced with the following commands: setVelocityFromDeltaTime(time) convertUnits(jointCommands, stateType) - Optional setJointCommands(jointCommands, jointVelocities) The time difference calculation for joint rates is now explicitly done in a separate call. Also there is a separate convenience command for performing units conversion.Sends commands to Cyton hardware to move joints to a specified location. A time difference is calculated from the previous command to determine the rate at which to move the joints.	**timeNow** Current time **jointCommands** Vector of joint angles to move servos to **stateType** Optional unit conversion for input jointCommands	**EcBoolean** Success or failure of set command
getJointStates (EcRealVector &jointStates, const StateType stateType=JointAngleInRadians) const	Retrieve servo information. Depending on the stateType parameter it will return the last commanded position (default) or any of the configuration parameters for the servos (joint bias, min angle, max angle, reset angle, max joint rate, joint scale).	[out] **jointStates** Vector of returned values **stateType** Type and unit of requested values	**EcBoolean** Success or failure of query command
waitUntilCommandFinished (const EcU32 timeoutInMS) const	Wait for the last command to finish, up to a specified maximum time in milliseconds.	**timeoutInMS** Maximum time to wait in milliseconds before failing	**EcBoolean** Success or failure of wait command
numJoints () const	Retrieve the number of joints currently configured.		**EcU32** Number of joints in the loaded system
hardwareInterface (Ec::Plugin *plugin)	Examine current hardware configuration to list available ports.		**EcStringVector** A vector of string representing the port names of the devices available. Platform dependent. Empty list returned if not available, or plugin not loaded
plugin ()	Retrieve a handle to the loaded plugin.		**Ec::Plugin*** The loaded plugin

Appendix B

Actin Simulator

One of the available development and simulation software to control the Cyton arms is the Actin simulator, provided by ENERGID technologies. This simulator is a general simulator, not used particularly for Cyton, and it is a powerful tool to test and control all kind of robotic arms. This software allows to control fixed or mobile robots with up to 100 independent moving parts and uses internal inverse kinematics algorithms to perform the specified motion. In a general overview, the Actin software capabilities are:

- Mathematical and Geometrical Tools : includes a number of tools for mathematical and geometric calculation such as three-dimensional vector math and matrix routines and various orientation methods like Quanternion, Euler angles, angle-axis and so forth.
- Automatic Kinematic Control : calculates the joint rates and positions to reach the desired end-effector position. The inverse kinematics algorithm is automatically done based only on the manipulator model description and it gives to Actin the ability to control almost any robotic manipulator, independently of the number of links, the number of bifurcations and the type of joints and end-effectors.
- Rendering : provides manipulator visualization through an easy-to-use interface with the motion animation. Any number of manipulators can be shown and this tool provide the capability for intuitive debugging and to create human-machine interfaces.
- Machine vision : includes methods for capturing images with USB or fire-wire cameras as well as algorithms for analyzing the captured images and use the results to perform actions with the manipulator.
- Network Communications : includes C++ classes for network communications with sockets implemented for TCP/IP and UDP/IP communications. It allows remote supervision and teleoperation.

The interface of this simulator is presented in the Figure B.1, and it is displaying the Cyton Gamma 1500 as well as the manipulator configuration window.

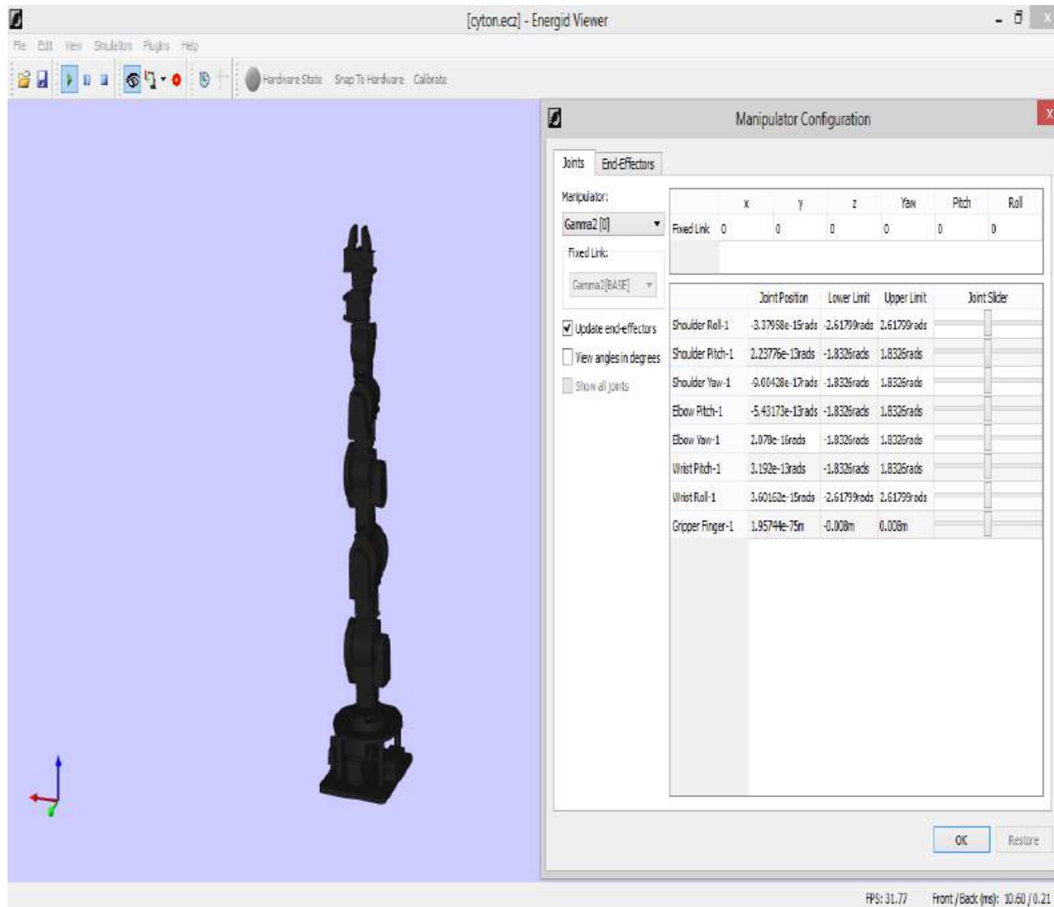


Figure B.1: Actin simulator interface and manipulator configuration window.

The manipulator configuration window allows the user to control each joint and gripper of the manipulator, to change the position and orientation of the coordinate frame related to the base of the manipulator and to change the Cartesian position and orientation of the end effector. When the position of the end-effector is changed, as well as when the guide mode is used which allows the user to control the position of the end-effector with mouse or keyboard using the visual interface, the simulator uses an inverse kinematic algorithm that deals with the redundancies of the arm.

This simulator also provides an Assistive Mode, which allows the user to hold the robotic arm and guide it through the set of target positions, while all joint motors had their torque disabled. Then it is possible to playback the guided motion, providing a more intuitive and natural programming control.

Appendix C

Structural analysis

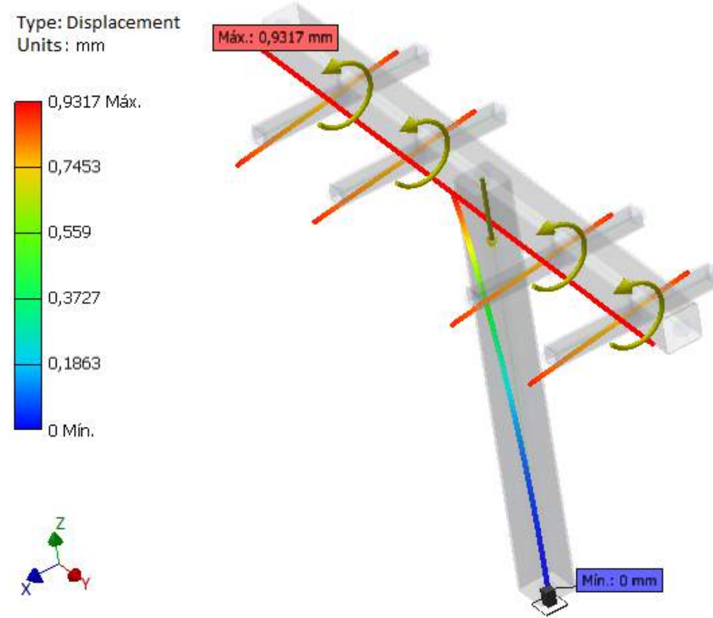


Figure C.1: Frame analysis of torso structure applying bending moments (yellow curved arrows) to the aluminum bars that support the robotic arms, considering that the structure is fixed to the base and also the gravitational force applied in the center of the structure as well. The bending moment is equivalent to the force applied in the end-effector of the arm as if it was carrying a load of 1,5Kg. The values presented are related to displacement from the initial position of the structure.

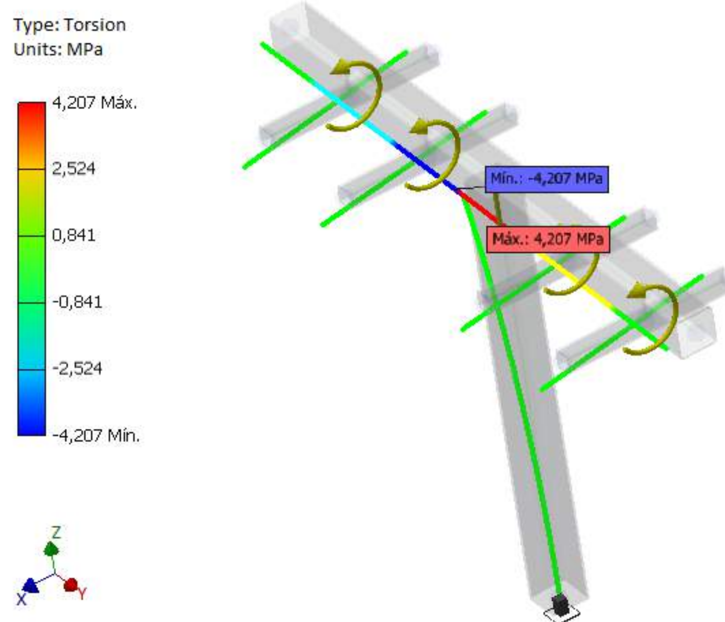


Figure C.2: Torsion analysis of the structure applying the same bending moments of previous figure.

Appendix D

Linear Trajectory Planning

Considering p a generic Cartesian position, $\dot{p}$ the velocity of this position and $\ddot{p}$ its acceleration, the general expressions of position, velocity and acceleration are presented in equations D.1, D.2 and D.3.(M.F. Santos, 2004)

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{D.1}$$

$$\dot{p}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{D.2}$$

$$\ddot{p}(t) = 2a_2 + 6a_3 t \tag{D.3}$$

The expressions D.1, D.2 and D.3 can be represented graphically as shown in the Figure D.1.

Considering the terms imposed in the expression D.4, and combining the equations D.1,D.2 and D.3, it can be shown that the equations D.5, D.6 and D.7 represent the linear position movement with null initial and final velocities.

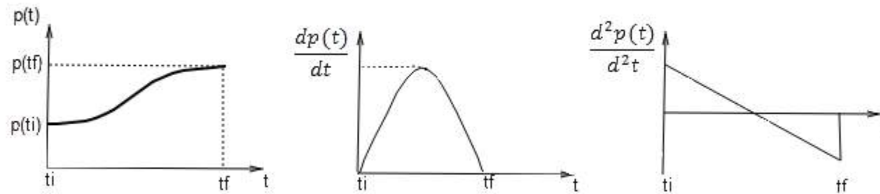


Figure D.1: Curves of position, velocity and accelaration.

$$p(t_i) = 0; p(t_f) = pf; \dot{p}(0) = p_i; \dot{p}(t_f) = \dot{p}_f \tag{D.4}$$

$$p(t) = -\frac{2}{t_f^3}(p_f - p_i) \cdot t^3 + \frac{3}{t_f^2}(p_f - p_i) \cdot t^2 + p_i \tag{D.5}$$

$$\dot{p}(t) = -\frac{6}{t_f^3}(p_f - p_i) \cdot t^2 + \frac{6}{t_f^2}(p_f - p_i) \cdot t \tag{D.6}$$

$$\ddot{p}(t) = -\frac{12}{t_f^3}(p_f - p_i) \cdot t + \frac{6}{t_f^2}(p_f - p_i) \tag{D.7}$$

Appendix E

Launch Files

```
1  <launch>
2
3    <!-- arg d defines the arm that will be controlled. positive-> LEFT ROBOTIC arm negative-> RIGHT ROBOTIC | arm (distance in meters) -->
4
5      <arg name="d" />
6      <arg name="K" />
7      <arg name="median_c" />
8
9      <remap from="joint_states" to="joint_states_l"/>
10     <remap from="hardware_node/goal" to="hardware_node_l/goal"/>
11     <remap from="hardware_node/feedback" to="hardware_node_l/feedback"/>
12   <node name="move_hardware_l" pkg="cyton" type="move_hardware" required="true" args="$(arg d)" cwd="node" />
13     <remap from="hardware_node" to="hardware_node_l"/>
14   <node name="hardware_node_l" pkg="cyton" type="hardware_node" required="true" args="$(arg d)" />
15     <remap from="start_joint" to="start_joint_l"/>
16     <remap from="angles" to="angles_l"/>
17     <remap from="joint_states" to="joint_states_l"/>
18   <node name="joint_control_l" pkg="cyton" type="joint_control" required="true"  />
19     <remap from="angles" to="angles_l"/>
20     <remap from="start_joint" to="start_joint_l"/>
21     <remap from="start_position" to="start_position_l"/>
22     <remap from="positions" to="positions_l"/>
23     <remap from="tracked_positions" to="tracked_positions_l"/>
24   <node name="position_control_l" pkg="cyton" type="position_control" required="true" args="$(arg d)"/>
25     <remap from="start_velocity" to="start_velocity_l"/>
26     <remap from="track_positions" to="track_positions_l"/>
27     <remap from="tracked_positions" to="tracked_positions_l"/>
28     <remap from="start_position" to="start_position_l"/>
29     <remap from="start_tracking" to="start_tracking_l"/>
30   <node name="position_track_l" pkg="cyton" type="position_track"  args="$(arg d)" required="true" />
31     <remap from="hardware_node/goal" to="hardware_node_l/goal"/>
32     <remap from="hardware_node/feedback" to="hardware_node_l/feedback"/>
33     <remap from="linear_positions" to="linear_positions_l"/>
34     <remap from="linear_velocities" to="linear_velocities_l"/>
35     <remap from="track_positions" to="track_positions_l"/>
36     <remap from="track_velocities" to="track_velocities_l"/>
37     <remap from="joint_states" to="joint_states_l"/>
38     <remap from="joint_states_est" to="joint_states_est_l"/>
39     <remap from="start_velocity" to="start_velocity_l"/>
40     <remap from="collision_detected" to="collision_detected_l"/>
41   <node name="velocity_control_l" pkg="cyton" type="velocity_control" required="true" output="screen" args="$(arg d) $(arg K)"/>
42     <remap from="start_position" to="start_position_l"/>
43     <remap from="time" to="time_l"/>
44     <remap from="positions" to="positions_l"/>
45     <remap from="start_linear_traj" to="start_linear_traj_l"/>
46     <remap from="start_track_traj" to="start_track_traj_l"/>
47   <node name="command_node_l" pkg="cyton" type="command_node" required="true" output="screen" args="$(arg d)" launch-prefix="xterm -e"/>
48     <remap from="time" to="time_l"/>
49     <remap from="positions" to="positions_l"/>
50     <remap from="linear_positions" to="linear_positions_l"/>
51     <remap from="linear_velocities" to="linear_velocities_l"/>
52     <remap from="joint_states" to="joint_states_l"/>
53     <remap from="start_linear_traj" to="start_linear_traj_l"/>
54     <remap from="start_velocity" to="start_velocity_l"/>
55   <node name="linear_traj_gen_l" pkg="cyton" type="linear_traj_gen" required="true" args="$(arg d)"/>
56     <remap from="start_track_traj" to="start_track_traj_l"/>
57     <remap from="track_positions" to="track_positions_l"/>
58     <remap from="start_tracking" to="start_tracking_l"/>
59     <remap from="track_velocities" to="track_velocities_l"/>
60     <remap from="start_velocity" to="start_velocity_l"/>
61   <node name="track_traj_gen_l" pkg="cyton" type="track_traj_gen" required="true" args="$(arg d) $(arg median_c)"/>
62
63 </launch>
```

Figure E.1: Launch file related to left robotic arm.

```
1 <launch>
2
3   <!-- args defines the arm that will be controlled. positive-> LEFT ROBOTIC arm negative-> RIGHT ROBOTIC | arm (distance in meters) -->
4
5     <arg name="d" />
6     <arg name="K" />
7     <arg name="median_c" />
8
9     <remap from="joint_states" to="joint_states_r"/>
10    <remap from="hardware_node/goal" to="hardware_node_r/goal"/>
11    <remap from="hardware_node/feedback" to="hardware_node_r/feedback"/>
12  <node name="move_hardware_r" pkg="cyton" type="move_hardware" required="true" args="$(arg d)" cwd="node" />
13    <remap from="hardware_node" to="hardware_node_r"/>
14  <node name="hardware_node_r" pkg="cyton" type="hardware_node" required="true" args="$(arg d)" />
15    <remap from="start_joint" to="start_joint_r"/>
16    <remap from="angles" to="angles_r"/>
17    <remap from="joint_states" to="joint_states_r"/>
18  <node name="joint_control_r" pkg="cyton" type="joint_control" required="true"  />
19    <remap from="angles" to="angles_r"/>
20    <remap from="start_joint" to="start_joint_r"/>
21    <remap from="start_position" to="start_position_r"/>
22    <remap from="positions" to="positions_r"/>
23    <remap from="tracked_positions" to="tracked_positions_r"/>
24  <node name="position_control_r" pkg="cyton" type="position_control" required="true" args="$(arg d)"/>
25    <remap from="start_velocity" to="start_velocity_r"/>
26    <remap from="track_positions" to="track_positions_r"/>
27    <remap from="tracked_positions" to="tracked_positions_r"/>
28    <remap from="start_position" to="start_position_r"/>
29    <remap from="start_tracking" to="start_tracking_r"/>
30  <node name="position_track_r" pkg="cyton" type="position_track"  args="$(arg d)" required="true" />
31    <remap from="hardware_node/goal" to="hardware_node_r/goal"/>
32    <remap from="hardware_node/feedback" to="hardware_node_r/feedback"/>
33    <remap from="linear_positions" to="linear_positions_r"/>
34    <remap from="linear_velocities" to="linear_velocities_r"/>
35    <remap from="track_positions" to="track_positions_r"/>
36    <remap from="track_velocities" to="track_velocities_r"/>
37    <remap from="joint_states" to="joint_states_r"/>
38    <remap from="joint_states_est" to="joint_states_est_r"/>
39    <remap from="start_velocity" to="start_velocity_r"/>
40    <remap from="collision_detected" to="collision_detected_r"/>
41  <node name="velocity_control_r" pkg="cyton" type="velocity_control" required="true" output="screen" args="$(arg d) $(arg K)"/>
42    <remap from="start_position" to="start_position_r"/>
43    <remap from="time" to="time_r"/>
44    <remap from="positions" to="positions_r"/>
45    <remap from="start_linear_traj" to="start_linear_traj_r"/>
46    <remap from="start_track_traj" to="start_track_traj_r"/>
47  <node name="command_node_r" pkg="cyton" type="command_node" required="true" output="screen" args="$(arg d)" launch-prefix="xterm -e"/>
48    <remap from="time" to="time_r"/>
49    <remap from="positions" to="positions_r"/>
50    <remap from="linear_positions" to="linear_positions_r"/>
51    <remap from="linear_velocities" to="linear_velocities_r"/>
52    <remap from="joint_states" to="joint_states_r"/>
53    <remap from="start_linear_traj" to="start_linear_traj_r"/>
54    <remap from="start_velocity" to="start_velocity_r"/>
55  <node name="linear_traj_gen_r" pkg="cyton" type="linear_traj_gen" required="true" args="$(arg d)"/>
56    <remap from="start_track_traj" to="start_track_traj_r"/>
57    <remap from="track_positions" to="track_positions_r"/>
58    <remap from="start_tracking" to="start_tracking_r"/>
59    <remap from="track_velocities" to="track_velocities_r"/>
60    <remap from="start_velocity" to="start_velocity_r"/>
61  <node name="track_traj_gen_r" pkg="cyton" type="track_traj_gen" required="true" args="$(arg d) $(arg median_c)"/>
62
63 </launch>
```

Figure E.2: Launch file related to right robotic arm.

Appendix F

User Guide

User Guide

Installation and Usage guide to control the two Cyton Gamma 1500 robotic arms and Kinect sensor in point-to-point control mode, continuous control mode and human gesture imitation mode.

Tiago Gomes Moura

Institute of Electronics and Telematics Engineering of Aveiro (IEETA)

University of Aveiro

2015

Installation Guide

In this section are outlined the installation procedures assuming the software that was used for testing (new versions can be used with the appropriate modifications).

Software requirements:

- Linux Operating System – Ubuntu 12.04
- Robot Operating System (ROS) – ROS Fuerte version
- ROS-Cyton interface
- OpenNI Unstable build for Ubuntu 12.04 – v1.5.4
- PrimeSense NITE Unstable build for Ubuntu 12.04 - v1.5.2.21

Installation:

- Install the ros-cyton module:
- Installation tutorials can be found here: https://code.google.com/p/cyton-ros-pkg/downloads/list
  1. Install ROS-Fuerte from: http://wiki.ros.org/fuerte/Installation/Ubuntu
  2. Copy Cyton folder (ros-cyton interface) to the work folder
  3. Add environmental variables to the bash file:

     CYTON_INC :=include directory path inside the cyton folder
     CYTON_LIB :=lib folder path inside the cyton folder
     CYTON_BIN :=bin folder path inside the cyton folder
     CYTON_EE_FILE :=EE file path,which stores a series of EE position
  4. Add ROS_PACKAGE_PATH :

     export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:[path]
  5. Edit the existing Makefile and replace all content with:

     include $(shell rospack find mk)/cmake.mk
  6. Using a terminal inside cyton folder build using rosmake command
  7. Install module using rosmake –rosdep-install
- Install OpenNI:
- Download OpenNI_NITE_Installer-Linux32-0.27
- Extract the file and navigate to *OpenNI-Bin-Dev-Linux-x86-v1.5.4.0* folder (version can be different if it was installed a recent version)
- Use sudo ./install.sh
- Navigate to *NITE-Bin-Dev-Linux-x86-v1.5.2.21* folder
- Use sudo ./install.sh
- Navigate to *Kinect* folder then *Sensor-Bin-Linux-x86-v5.1.2.1*
- Use sudo ./install.sh

Testing the installation:

- Connect the Kinect sensor and verify if green Led is blinking
- Navigate to:
  OpenNI_NITE_Installer-Linux32-0.27-> OpenNI-Bin-Dev-Linux-x86-v1.5.4.0->Samples->Bin->x86-Release
- Use ./NiViewer
- An image of the Kinect sensor should appear

Usage Guide

This usage guide is related to the system composed by two Cyton Gamma 1500 and a Kinect sensor.

- Plug the USB cables of the two Cyton robotic arms (preferentially left robotic arm first) and then the Kinect sensor
- Enable permission to USB (if not enabled already): sudo chmod 777 /dev/ttyUSBx (x is the USB port number)
- Open a terminal (preferentially a multi terminal window)
- Run the next command to launch the openni_tracker package:
  roslaunch openni_launch openni.launch camera:=openni
- In a different window run the next command to launch the cyton nodes to control both arms:
  roslaunch cyton cyton.launch
- If is pretended to control just one of the arms run:
  roslaunch cyton cyton_l.launch (Left robotic arm)
  roslaunch cyton cyton_r.launch (Right robotic arm)
- Two command windows will appear, one to control the left robotic arm and another one to control the right robotic arm
- Follow the instructions shown in each command window

Cautions:

- Always plug the power cables of the Cyton robotic arms to a socket equipped with an emergency button to cut the power
- If the robotic arm behaviour seems to be unstable press the emergency power and restart the system
- The system was tested in many situations but it is not guaranteed that it is 100% robust and without problems
- The Kinect sensor can provide unexpected data, which can result in an unstable behaviour of the system, in cases when the user leaves the scene, when another user enters the scene or when there are occlusions or the user is too far away or too close from the sensor