Università degli Studi di Padova

Department of Information Engineering

Master Degree in Computer Engineering

Development of an interface for intuitive teleoperation of Comau manipulator robots using RGB-D sensors

Supervisor: Prof. Emanuele Menegatti
Co-Supervisor: Stefano Michieletto
Co-Supervisor: Valentina Ferrara

Candidate: Andrea Bisson

13th of October 2014

Academic Year 2013/2014

"Computers are incredibly fast, accurate, and stupid.
Human beings are incredibly slow, inaccurate, and brilliant.
Together they are powerful beyond imagination."

– Albert Einstein –

Abstract

Teleoperation of manipulator robots with RGB-D sensors is now mainly done using inverse kinematics techniques. In this work, we developed an intuitive way to teleoperate a Comau industrial robot by means of a Microsoft Kinect device, in order to control directly the manipulator joints by retargeting specific human motion. In this way the human operator has the full control of robot movements with practically no training, because of the intuitivity of this teleoperation method.
The motion remapping into the robot joints has been done by computing angles between vectors built from positions of human joints.
The system developed in this work to fullfil the teleoperation task exploits Robot Operating System (ROS) framework and the Comau C5G Open architecture. In this way, we obtained a very modular system that allows developers to change either the tracking sensor or the robot model with some small changes. Finally, the developed teleoperation system has been successfully tested on two real Comau robots, revealing to be fast and strongly reliable.

Contents

List of Figures

Chapter 1

Introduction

Teleoperation is a technique that allows human operator to move and to program a robot by simply controlling it from a certain distance. This technique is used to move robots particularly in dangerous environments and tasks, such as bombs dismantling, exploring human inaccessible sites, maintenance of nuclear facilities, underwater operations, etc. Moreover, teleoperation can be also used for robot offline programming, allowing a human operator to save a lot of time.
Teleoperation tasks can use both trivial (e.g. joysticks, keyboards, etc.), but also more complicated human machine interfaces, such as vision-based interfaces. The first ones are less intuitive to use with respect to the second ones, in fact they require some training to the human operator in order to be used properly and efficiently. The vision-based teleoperation, instead, is generally more intuitive and easier to use. In fact, it finds applications in particular for the programming of more complex robots with an high number of degrees of freedom (DoFs), such as the humanoid robots [3]. Nevertheless, recently, these techniques have been applied also to industrial manipulator robots, in order to move and program them more easily and intuitively [4, 5]. Vision-based teleoperation can be of two types, marker-based or markerless: the first is more uncomfortable because it requires to the operator to wear additional clothes, while the second is more complex to develop, because human keypoints must be estimated via software.

In this master thesis work, we will present a technique that allows to teleoperate a manipulator robot with a markerless vision-based system, by using in particular a RGB-D sensor. The sensor chosen for this work is a Microsoft Kinect, that is a very cheap and powerful RGB-D sensor, which allows to track human movements without using any kind of uncomfortable marker. Moreover, Microsoft Kinect has a large developer community, which implies the existence of many already implemented software packages to work with.
The implementation of the whole system has been made the more modular as possible. At this purpose, Robot Operating System (ROS) middleware has been

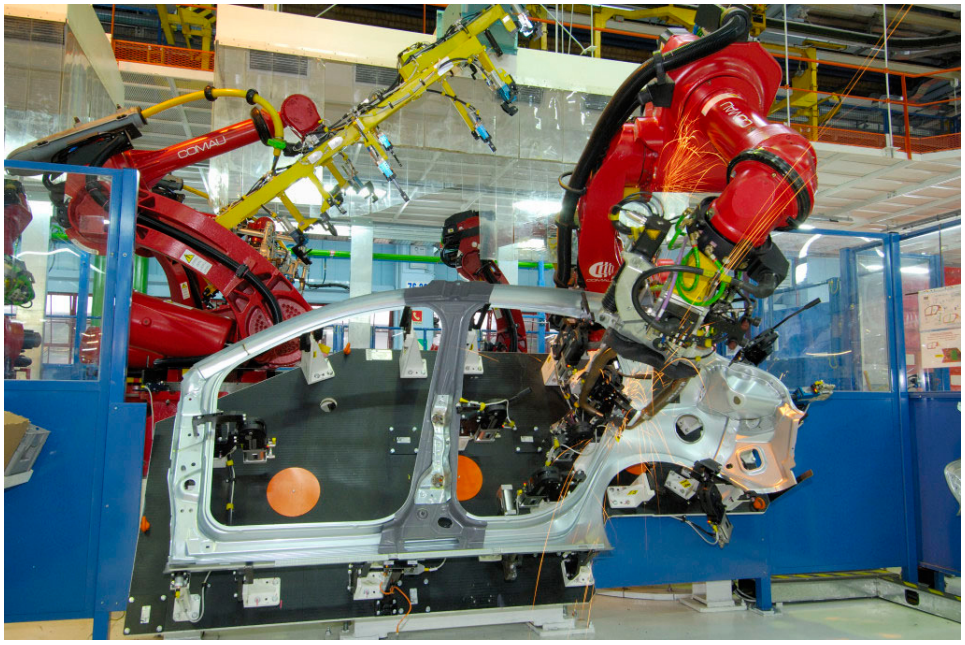

Figure 1.1: Comau manipulator robots on a assembly line.

used as framework to connect the developed modules one to each other. The development approach adopted allows developers to change almost effortlessly both the sensor and the robot model used for this work.
ROS middleware is spreading even more in both academic and industrial environments. While for the first, ROS has a great potential allowing to test new algorithms on robot on high level programming; for the second it is seen as a platform that allows to program industrial manipulator robots in order to create complex applications for their customers in less time.
This master thesis work has been done in collaboration with Comau Robotics R&D department, which has provided a wide support about the C5G Open architecture used for the teleoperation on real Comau robots. Thanks to this collaboration we were able to test the developed system in both virtual and real environment. Gazebo has been used as simulator through an accurate modelization of kinematic and dynamic characteristics of a Comau Smart5 SiX, the robot mainly used during this work.

Comau S.p.A. is one of the most important manufacturer of manipulator robots in the world. Its headquarter is located in Grugliasco, near Turin. Comau produces a wide range of manipulator robots with different characteristics, which go from robots with a small payload (a few kilograms) to ones with a heavy payload (hundreds of kilograms).

This master thesis work is organized as follows:

- on Chapter 2 we introduce the state of the art for robot teleoperation,

describing its applications on both humanoid and manipulator robots;

- on Chapter 3 we present all the hardware and the software used in order to implement a working system which can manage the teleoperation of Comau manipulator robots;
- on Chapter 4 we describe how the whole system has been designed and how the teleoperation algorithm works;
- on Chapter 5 are reported conclusions and future works aimed to improve the system developed so far.

Chapter 2

State of the Art

In this Chapter we introduce different approaches for robot vision-based teleoperation already applied in literature. We start by presenting humanoids teleoperation that takes advantage of similarities between human and robot. Then, we describe most important techniques used so far to teleoperate industrial manipulator robots.

2.1 Robots Teleoperation

Robots teleoperation is an already consolidated technique used to move a robot in a dynamic or a dangerous environment, which requires human intelligence for the decision making on how to control the robot. Some examples of its application are: dismantling bombs, maintenance of nuclear facilities, exploring inaccessible sites in rescue, mining.
Also the choice of the human-machine interface is important in order to move the robot in an intuitive and natural way. In fact, some of the most known and used interfaces are: dials, joysticks, computer mouse and computer GUI. Anyway, all these interfaces could be not so intuitive to use for human operators, and they may require some training skills in order to teleoperate the robot efficiently. Nevertheless, exist also vision-based interfaces which are usually more natural and intuitive for a human operator, allowing to do more complex movements in less time.
In the next sections, we describe some important results obtained using teleoperation techniques for both humanoid and industrial manipulator robots.

2.1.1 Teleoperation of Humanoid Robots

Transferring motion from a human demonstrator[1] into humanoid robots is a studied topic since long time. This imitation based learning process aims to simplify the complex and time consuming manual programming of a humanoid robot, by simply recording the movements of an expert teacher. One of the most difficult things to deal with in retargeting algorithms are the differences in topology, anthropometry, joint limits and degrees of freedom between the human demonstrator and the robot. Other requests for the retargeting problem with humanoids are balance constraints and interactivity in dynamically changing environments, including self collision avoidance.
Nowadays, exist several off-line solutions has been developed using both markers and markerless techniques. The former methods record information from markers placed on an human actor performing a task. The latter ones are more complex and they are able to extract motion directly from video (2D or 3D) data, with no needs of any type of marker.

In [3], authors presented an online solution considering joints limit, joints velocity and self-collision constraints in a control theoretic retargeting of the robot joint motion. The algorithm acquires the motion of a human actor using only a depth sensor and then it retargets the movements on the Honda ASIMO humanoid robot[2] (Figure 2.1).

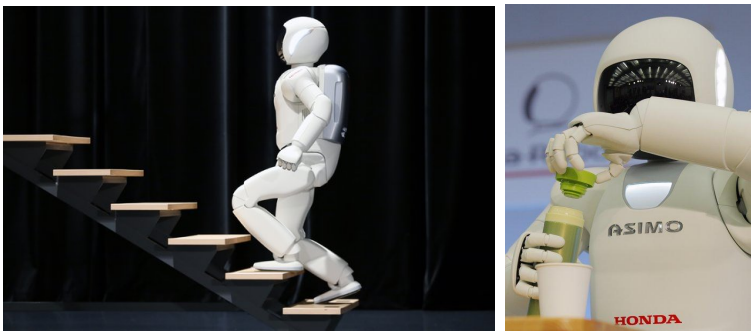

Figure 2.1: Two photos of the humanoid robot Honda ASIMO. On the left image, the robot is going up the stairs, while on the right image it is opening a bottle.

We now describe more in detail the algorithm pipeline developed in [3]:

1. **Data acquisition**: the algorithm use as input a depth image stream, obtainable by using active or passive stereo or time of flight[3] sensors providing

[1] it is also called *retargeting*

[2] Honda ASIMO (2014 version) is a humanoid robot with 57 degrees of freedom, it is 130 cm tall and it weighs 55 kg. Official website: `http://asimo.honda.com/`

[3] A time of flight sensor acquires a depth image by measuring the time that takes an emitted light signal to return back

a point cloud of the human actor. In particular, for the image acquisition, has been used a Swiss Ranger-SR 3000.

2. **Key-Point detection**: from the depth images, the next step is to detect the 8 key-points of the upper body of the person, which correspond to 3D position vectors of: waist joint, 2 shoulder joints, 2 elbow joints, 2 wrist joints and the head center. (figure 2.2)

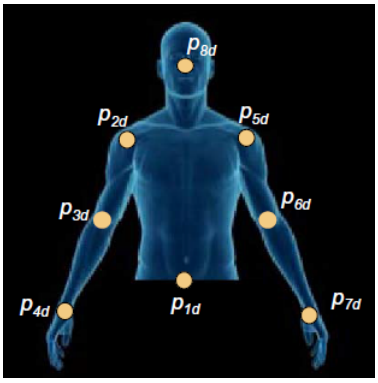

Figure 2.2: The 8 Key-Points used for the Dariush et al. retargeting algorithm

The key point detector is based on a Bayesian framework which uses both spatial context and temporal information to detect limbs and self occlusion of the body.

3. **Scaling, Filtering and Interpolation**: Once the key-points are acquired from the depth image, they are scaled to the robot dimensions, then filtered using a low-pass filter and interpolated. After these operations, it is obtained a position vector representing a descriptor on the robot model.

4. **Retargeting**: this phase is a local constrained optimization problem. In fact, before sending the motion to the robot, the difference between the Cartesian position of the reference and the descriptors must be minimized, also taking into account the robot kinematic constraints, such as joint position and speed limits and self collisions.

Similar approaches have been used also in [6–8] by using different sensors or robots. In some cases, like [6, 7], the method adopted uses a skeletal tracking technique starting from RGB-D data. While in [8] the robot stability has also been involved in the control loop.

In all the described works, the user is able to easily control a single part of the robot by moving a specific part of his body and this is a very important characteristic of the system.

2.1.2 Teleoperation of Manipulator Robots

Some vision-based teleoperation techniques have been applied also to industrial manipulator robots. The problem of retargeting the joints movements of a human actor onto the joints of a manipulator is, in a certain way, more difficult than the retargeting problem applied on humanoids. This is because, while humanoid robot movements resemble the human ones, the motion of a manipulator robot is designed for industrial tasks and it has a more limitations due to a lower number of Degrees of Freedom (DoFs), usually 6, and and it is fixed on a platform on the ground.
In this Section we describe some techniques developed to retarget movements from a human operator to a manipulator robot, presented in [4] and [5].

In [4], a marker based vision interface has been proposed, where a stereo camera system tracks three markers disposed on the right hand of a human operator, respectively on the wrist, on the thumb and on the index finger. The acquired 2D coordinates of these points are then remapped in a 3D environment and then retargeted in real-time to the end-effector of a 6 DoFs manipulator robot. During the teleoperation, the human operator has a visual feedback of the robot motion thanks to a set of cameras pointed on the robot.

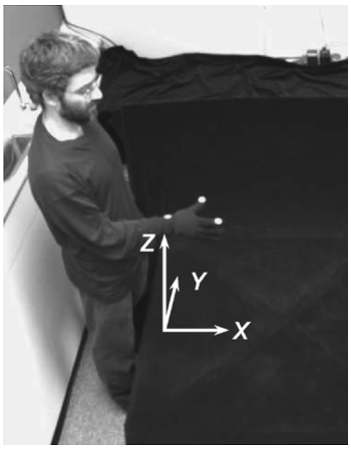

Figure 2.3: In the image we can see the human operator hand with 3 white markers and, nearby, the Cartesian reference frame used to calculate the yaw-pitch-roll angles.

The 4 points used for the retargeting are given by the 3 markers: wrist (W), thumb (T), the index finger (I), plus the midpoint (M) between thumb and index. Given these points, the rotation angles yaw, pitch and roll are computed from the directions of the vectors v_{WM} and w_{TI}, respect to the reference frame (represented in figure 2.3), in the next way:

$$\alpha_{\text{yaw}} = \arctan\left(\frac{M_{0y}}{M_{0x}}\right) \tag{2.1}$$

$$\beta_{\text{pitch}} = \arctan\left(\frac{M_{0z}}{\sqrt{M_{0x}^2 + M_{0y}^2}}\right) \tag{2.2}$$

$$\gamma_{\text{roll}} = \arctan\left(\frac{I_{2z} - T_{2z}}{I_{2y} - T_{2y}}\right) \tag{2.3}$$

where:

- M_{0x}, M_{0y} and M_{0z} are respectively the projection vectors of the point M on the axes of the initial Cartesian reference system;
- I_{2y}, T_{2y} and I_{2z}, T_{2z} are the projection points of the markers I and T respectively on the y and z axes, both on the reference frame obtained after applying α_{yaw} and β_{pitch} rotations.

The orientation of the end-effector on the teleoperated robot is computed by applying the three rotation angles obtained by the orientation of the operator hand in the above order. Moreover, the translations of the end-effector along the 3 axes are calculated from the translations of the markers with respect to their starting position.

In [5], instead, the authors have developed a vision system based on the use of Microsoft Kinect as a tracking sensor for human arms, in order to teleoperate the manipulator robot used for the gripping task. The implementation is based on ROS in order to obtain a modular system and integrate the OpenNI package to get human joints positions from the Kinect sensor.
In this work, the operator has to use both arms to control position, orientation of the robot end-effector as well as the opening and closure of the gripper, since OpenNI tracker cannot achieve any type of information regarding hands orientation or gesturing.
A ROS node saves the initial position of both operator hands and end-effector once the teleoperation starts. The difference between the initial and the current positions of the right hand is used to calculate the position to be sent to the robot. Instead, the left hand controls the end-effector position and the gripper. In fact, if the left hand is extended:

- up, the gripper is open;
- down, the gripper is closed;
- left, the end-effector pose controller is used;

- right, the position controller is used.

If the position controller is activated, the robot can be moved freely by the right hand of the human operator, who has a visual feedback of the robot movements. When the pose controller is active, the orientation of the end-effector is a constraint and it is forced to point down, while the operator can move only its position with the right hand. Moreover, the pose controller allows the operator to move the end-effector more slowly, with finer movements.
Both the position and the pose controllers use inverse kinematics[4] in order to calculate joints speeds $\dot{\theta}$ in the form:

$$\dot{\theta} = (J_{\text{task}})^{\dagger}\, u(k) \tag{2.4}$$

where $(J_{\text{task}})^{\dagger}$ is the damped least squares pseudo-inverse of J_{task}, the task jacobian, while $u(k)$ is the signal computed by a discrete PID controller:

$$u(k) = K_p\, e(k) + K_i \sum_{j=0}^{k} e(j) + K_d \frac{e(k) - e(k-1)}{T} \tag{2.5}$$

where K_p, K_i, K_d are matrices containing PID gains for the $u(k)$ elements, T is the sampling period and $e(k) = x(k) - x_{\text{ref}}(k)$. $x(k)$ and $x_{\text{ref}}(k)$ are column vectors representing respectively the actual and the desired positions of the robot end-effector. In particular, depending on the system state, the meaning of the components in equations 2.4 and 2.5 is the next:

- if the pose controller is active:
  - $J_{\text{task}} = J_A$, where J_A is the analytical jacobian in dual quaternion[5] space [9].
  - $x(k) = \underline{\mathbf{x}}(k)$ and $x_{\text{ref}}(k) = \underline{\mathbf{x}}_{\text{ref}}(k)$, where $\underline{\mathbf{x}}(k)$ and $\underline{\mathbf{x}}_{\text{ref}}(k)$ are both dual quaternions
- if the position controller is active:
  - $J_{\text{task}} = J_{GL}$, where J_{GL} is the linear velocities related part of the geometrical jacobian [2].
  - $x(k) = p(k) = (p_x\ p_y\ p_z)^T$ and $x_{\text{ref}}(k) = p_{\text{ref}}(k) = (p_{\text{ref}x}\ p_{\text{ref}y}\ p_{\text{ref}z})^T$, where $p(k)$ and $p_{\text{ref}}(k)$ are the current and the desired end-effector positions extracted from the dual quaternion position $\underline{\mathbf{x}}(k)$

[4] for a complete description of the inverse kinematics problem, look at [1, 2]

[5] for a more complete description on techniques and advantages of using dual quaternion control, we suggest respectively the reading of [9] and [10]

Chapter 3

Hardware and Software

In this Chapter we describe all the hardware used, such as the Comau's open controller architecture (that includes a B&R industrial PC and a Comau manipulator robot with its controller) and the Microsoft Kinect motion sensor. Moreover, we present the adopted software framework, such as ROS and several types of libraries.

3.1 Hardware

3.1.1 Comau's Open Controller Architecture

Such as every industrial manipulator, a Comau working cell consists in a robot and its control unit. This controller is a cabinet that contains some industrial PCs that drive and control the robot. To this control unit is connected the Teach Pendant (TP), that is an I/O device useful to move and program the robot and also to make some diagnostics.

Figure 3.1: A photo of the fifth version of the Comau Teach Pendant

As an optional module, Comau sells to its customers an open version of the robot controller: the name of this architecture is *C5G Open*. This type of system allows to control the robot motion, using an external industrial PC connected directly to the robot controller by a Powerlink Ethernet (these components will be described in details respectively in sections 3.1.1.1 and 3.1.1.2). This technology allows to software developers to work with industrial manipulators by using high level programming languages, such as C or C++. The C5G Open architecture is essential in order to move Comau robots using external sensors, such as vision or force sensors.
This type of controller is a sort of bridge between the academic and the industrial robotics worlds: in fact it allows to develop and test complex algorithm on Comau manipulator robots with the additional use of external sensors.
The modalities of robot controlling supplied by this architecture are:

- **Default modality** is the basic communication modality. The switching to every other modality requires to pass through this one.
  - CRCOPEN_LISTEN
- **Position modalities** allow users to control the robot by assigning respectively absolute, relative or additive position targets.
  - CRCOPEN_POS_ABSOLUTE
  - CRCOPEN_POS_RELATIVE
  - CRCOPEN_POS_ADDITIVE
- **Speed modalities** allow the user to control the robot using velocity targets. (Not implemented yet)
- **Acceleration modalities** allow the user to control the robot using acceleration targets. (Not implemented yet)

An interesting feature of the C5G Open system is the simulation mode, with which we can control the manipulator from the external industrial PC without moving the real robot, but only the simulated one[1]. In simulation mode, in fact, the robot controller works normally, with the exception that it doesn't send any current to the motors. This is really useful for testing motion algorithms avoiding to damage the real manipulator.

[1]Comau supplies also a visualizer of their robots for testing purposes

3.1.1.1 The B&R Industrial PC

The computer supplied by Comau for the C5G Open is a B&R's APC 910 model (called LPC) with the next technical characteristics[2]:

- **CPU**: Intel i7 2.5 GHz with 4 MB of L2 cache
- **RAM**: 4 GB
- **HDD**: 500 GB
- **I/O**: 1 standard Ethernet + 1 Powerlink Ethernet ports (this one must be connected to the robot controller in order to work in C5G Open modality), 5 USB ports, PCI standard + PCI Express slots, RS232
- **OS**: Linux Ubuntu 10.04 32bit with PREEMPT real-time kernel, 2.6 version

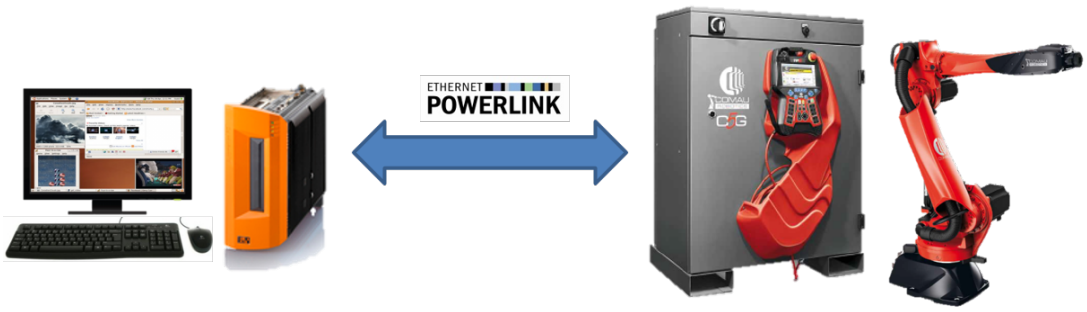

Figure 3.2: The Comau's open controller architecture: on the right side we have the robot with its C5G controller and on the left we have the B&R industrial PC. This two components are interconnected between them by a Powerlink Ethernet.

3.1.1.2 The Powerlink Ethernet

Ethernet Powerlink is a deterministic real-time protocol introduced by B&R in 2001 for standard Ethernet. It is an open protocol managed by the Ethernet Powerlink Standardization Group (EPSG)[3].

This type of Ethernet is based on a broadcast protocol, in which there is a Master node that gives a rigid temporization, avoiding collisions and ensuring a hard real-time communication. In this protocol we can distinguish three communication phases:

1. **Start Signal**, during this phase, the Managing Node (MN) sends a synchronization message Start of Cycle (SoC) to all Controlled Nodes (CN)

[2] `http://www.br-automation.com/it/prodotti/pc-industriali/automation-pc-910/`
[3] `http://www.ethernet-powerlink.org/en/powerlink/technology/`

2. **Isochronous Phase**, the Managing Node calls each node to transfer time-critical data for process or motion control by sending (always in broadcast) the Poll Request frame (PReq). Then, only the addressed node answers with the Poll Response frame (PRes).

3. **Asynchronous Phase**: the Managing Node grants the right to one particular node for sending ad-hoc data by sending out the Start of Asynchronous frame (SoA). The addressed node will answer with Async Data. Standard IP-based protocols and addressing can be used during this phase.

A Powerlink network is used within the robot controller, between:

- the computer[4] (called APC) that manages the trajectory generation, the dynamic model calculation, the position, speed, current loop, etc.;
- the servo drivers.

Moreover, in the C5G Open architecture, another Powerlink communication is used to exchange packets between the APC and the LPC every 400 μs during the Isochronous phase.

In both Powerlink networks, independent each other, the Managing node is always the APC inside the robot controller.

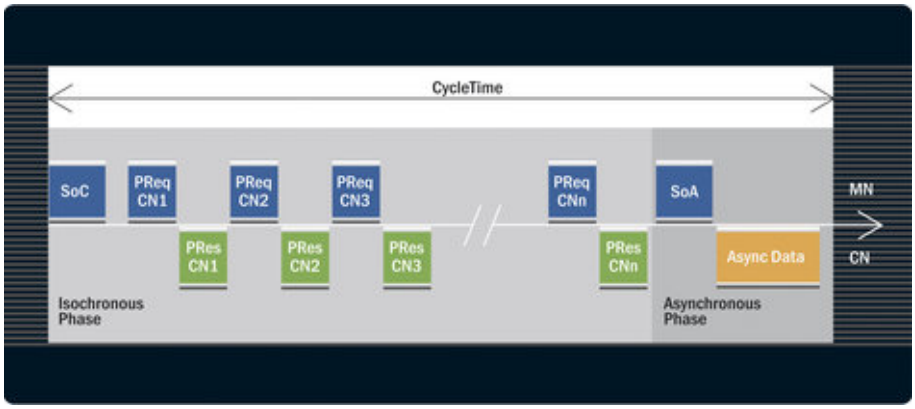

Figure 3.3: Diagram of a cycle of the Powerlink protocol

[4]This computer is a B&R APC model 820

3.1.1.3 The Comau Smart5-SiX Robot

The industrial manipulator robot Comau Smart5-SiX has been used in this work, that is actually the smallest robot manufactured by Comau[5]. This industrial manipulator belongs to the Comau Smart family, which robots are particularly suitable for all operations that require fast movements and a high degree of repeatability. These kind of manipulators can perform many small load tasks, such as: arc welding, assembling, handling, packaging, sealing and polishing.

Figure 3.4: Photo of the Comau Smart5-Six manipulator robot

Comau Smart5-Six robot has the next technical characteristics and performance:

Characteristic	
Type	Anthropomorphous
Number of axes	6
Weight	160 kg
Repeatability (ISO 9283)[6]	0.05 mm
Load at wrist	6 kg
Max horizontal reach	1.4 m

In the next table, we report the main robot kinematic characteristics[7]:

[5] `http://www.comau.com/eng/offering_competence/robotics_automation/products/small_payload_robots/Pages/smart_5_six.aspx`

[6] Repeatability is the ability of a manipulator robot to return exactly to a previously reached position. Formally, repeatability is the standard deviation of the same position reached by a robot at maximum speed and at maximum payload. This ability is important when performing repetitive industrial tasks.

[7] The stroke limits and the speeds are of public domain on the Comau's website, while the axes accelerations have been privately supplied by Comau

Axis	Min limit	Max limit	Speed	Acceleration
1	$-170°$	$+170°$	$140°/s$	$280°/s^2$
2	$-85°$	$+155°$	$160°/s$	$320°/s^2$
3	$-170°$	$0°$	$170°/s$	$340°/s^2$
4	$-210°$	$+210°$	$450°/s$	$700°/s^2$
5	$-130°$	$+130°$	$375°/s$	$750°/s^2$
6	$-2700°$	$+2700°$	$550°/s$	$1100°/s^2$

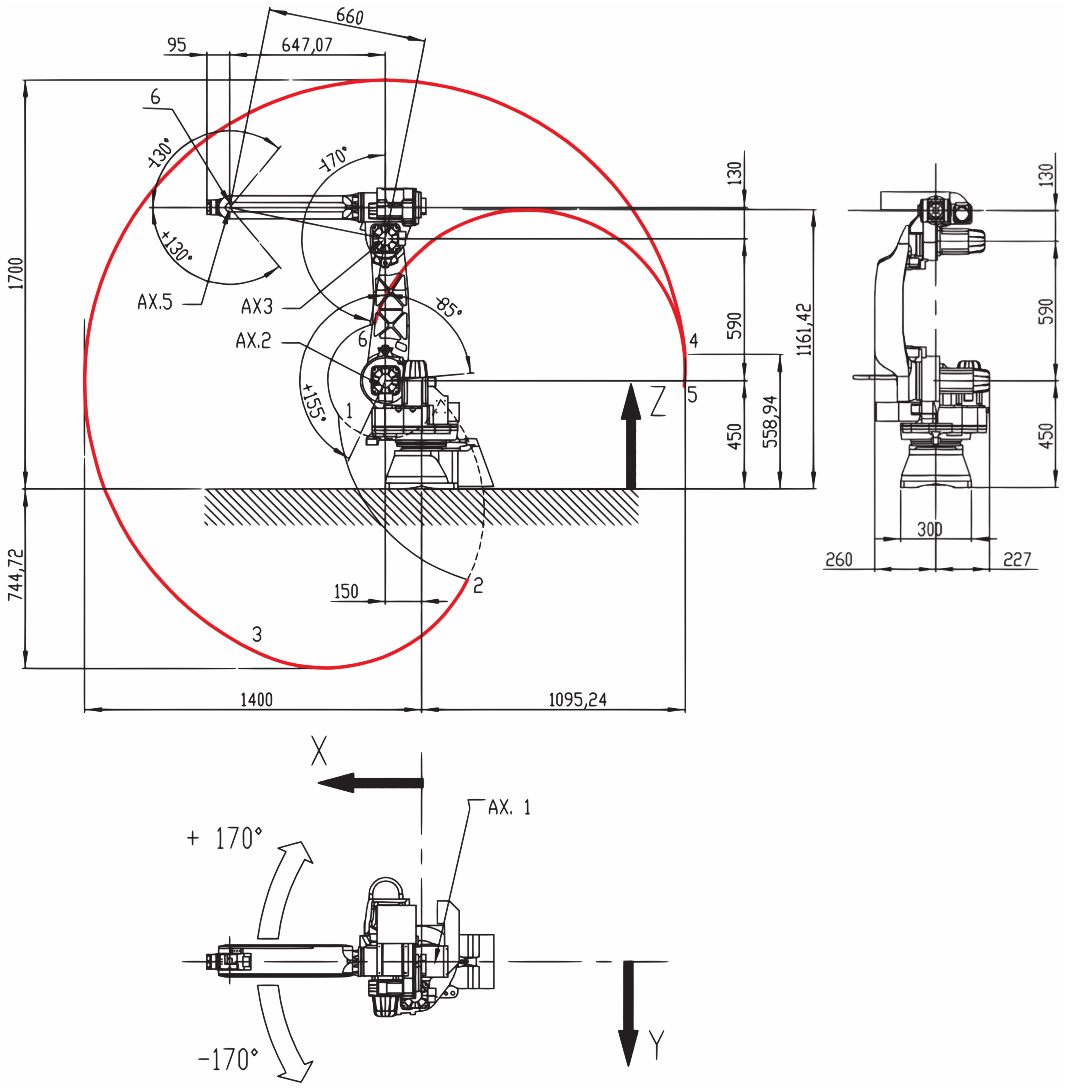

Figure 3.5: This plot shows the dimensions of the axes of the Comau Smart5-Six. We can see the robot working area delimited by the red line, that represents the furthest positions reachable by the robot end-effector.

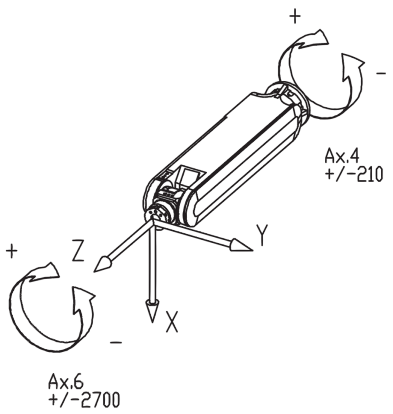

Figure 3.6: This plot shows the axes of the Comau Smart5-Six forearm.

3.1.2 The Microsoft Kinect Sensor

The Microsoft Kinect sensor, originally known as Project Natal, released the 4th of November 2010 for Microsoft Xbox 360 gaming console. It was developed by Microsoft and Primesense, which is an Israeli company that originally designed this device. This input sensor allows to a user in front of it to control and play with Xbox without holding or wearing anything.

The Microsoft Kinect potential was noticed not only by gamers, but also by the developers community, especially for academic research purposes. In fact many unofficial drivers for Windows and for Linux, such as OpenNI, were realized and published after the launch. In the middle of 2011, Microsoft released the official SDK and drivers only for Microsoft Windows for the next languages: C#, C++ and Visual Basic 2010.

Figure 3.7: Photo of the Microsoft Kinect sensor

3.1.2.1 Technical Details

The Microsoft Kinect is a RGB-D device composed by many types of sensors[8], positioned according to the figure 3.8:

[8] http://msdn.microsoft.com/en-us/library/jj131033.aspx

1. **IR Emitter and Receiver**: the emitter emits infrared light beams while the receiver, an IR camera, captures the IR beams that are reflected back. In this way, it is possible to get distance informations of every object in front of the Kinect, generating a depth map at 30 fps with a maximum resolution of 640 × 480 (see section 3.1.2.2 for a formal explanation). The viewing angles are: 43° for the vertical and 57° for the horizontal. The typical working range is from 0.8 m to 4 m[9].

2. **Color sensor**: it is an RBG camera that acquires scene images at a maximum frame rate of 30 with a resolution of 640 × 480. It can also work at the maximum resolution of 1280 × 960.

3. **Microphone array**: with this array of 4 microphones is possible to record audio and also to find the location of the sound source and the direction of the audio waves.

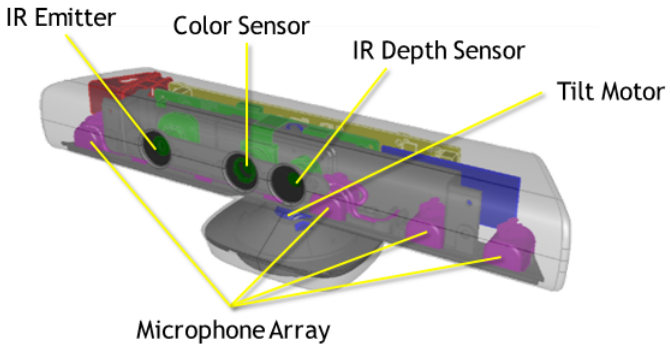

Figure 3.8: Disposition of the various sensors within the Microsoft Kinect

3.1.2.2 Mathematical Model

In this Section, we describe how Microsoft Kinect builds a depth image using the IR emitter and the IR depth camera [15].

Given:

- the Kinect Z axis, orthogonal to the image plane towards the object;
- the X axis, parallel to the baseline b, which connects the IR camera to the IR emitter;
- the Y axis points downward, from the depth camera to the ground, following the right hand rule.

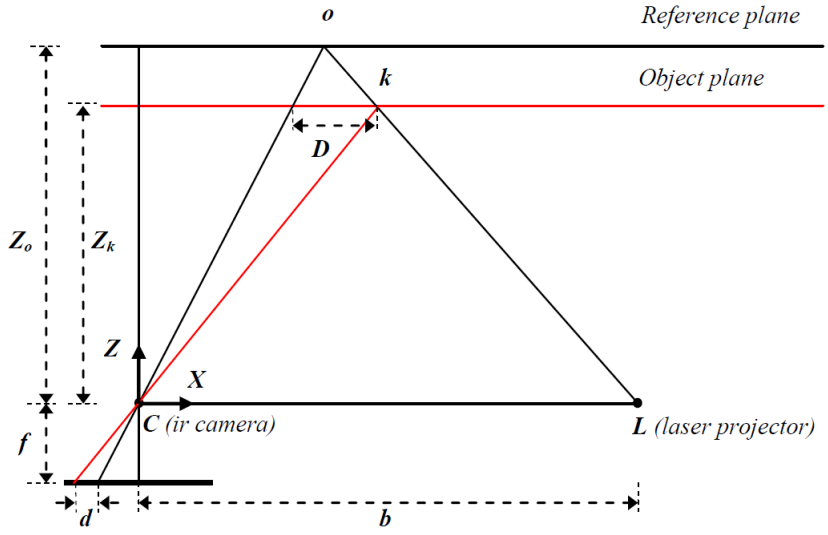

Figure 3.9: In this image are drawn the lines and the triangles representing the relationships between IR beam emitted from the emitter and reflected back from an object to the depth camera.

From figure 3.9 we can take a specific point of an object, at a known reference distance Z_o from the sensor. We consider now the same object closer to Kinect, at distance Z_k. In this way the IR beam will be shorter and it will result shifted along the X axis of a distance D. Using the similarity of triangles we obtain the next proportions:

$$\frac{D}{b} = \frac{Z_o - Z_k}{Z_o} \tag{3.1}$$

$$\frac{d}{f} = \frac{D}{Z_k} \tag{3.2}$$

where:

- Z_o and Z_o are the depth values of the same point of the considered object;
- D is the shift value along the X axis of the IR beam in object space;
- d is the disparity value of the object in the image space.

Now, substituting D from equation 3.2 to equation 3.1, we obtain the value of variable Z_k:

[9] `http://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges`

$$Z_k = \frac{Z_o}{1 + \frac{Z_o}{f \cdot b} d} \tag{3.3}$$

With the equation 3.3, it is possible to calculate the depth map starting from the disparity map acquired from the IR camera of Microsoft Kinect.

Starting from the depth image data, the skeleton tracker[10] will then search for human silhouttes in order to start tracking human movements.

Figure 3.10: This image shows the various steps of skeleton tracking in a Microsoft Kinect sensor: on the left the scene viewed from the RGB sensor, on the center the image from the depth sensor and on the right the identified human links.

[10] in our case the skeleton tracker used is NiTE

3.2 Software

In this Section we describe every software used in this work, from the ROS middleware to the libraries used for the communication with the robot controller and the movement interpolation.

3.2.1 ROS

The Robot Operating System (ROS)[11] is an open source middleware for robotics. It is a collection of libraries and softwares useful to simplify the design and the implementation of the code that will be used for moving robots and get data from the sensors. One of the main point of interest of ROS is the abstraction layer given by the APIs, that supplies to developers a common interface for robots and sensors, which guarantees a high software modularity. The last stable version of ROS at the time of writing is Indigo Igloo.

The fundamental ROS bricks are the *nodes*, which are independent system processes working on top of the ROS layer. They can be implemented in several languages, such as C++ and Python. These nodes can communicate between them using a publisher-subscriber communication protocol. In this protocol, a node publish a type of message on a *topic*[12] and all the other nodes that are subscribed to the same topic can receive and read the message sent by the publisher. Each node is unaware of the existence of the other nodes which are publishing or are subscribed to the same topic.

The ROS architecture can be graphically viewed as a directed graph, where the nodes are the vertices and the topics are the directed edges. The source codes of nodes and the messages are stored in project folder called *packages*.
To better understand the potentiality of the ROS architecture, we can use a practical example. Given a system composed by two ROS nodes, one for data acquisition from a sensor which also sends this data (through a message on a topic) to the other node that manages the movements of a mobile robot, so that it can plan a safe motion based on the information received from the sensor node. The most important thing to highlight on this toy example is that we can change the sensor device or the robot model: this will change the source code that allows the respective devices to work, but it will not change the ROS nodes architecture and behavior.

3.2.1.1 TF - Transform Frames

TF is a ROS package which main function is to keep track of multiple coordinate frames of the robot links over time. These frames are stored in a tree structure

[11] ROS official website: `http://www.ros.org/`
[12] A topic is a word that identifies the communication channel

buffered in time, to allow users to do all the desired operations between the coordinate frames.

3.2.1.2 PCL - Point Cloud Library

Point Cloud Library (PCL)[13] is a powerful tool for the point clouds processing. This cross platform library supplies a lot of state of the art algorithms for feature estimation, surface reconstruction, registration, model fitting and segmentation. PCL is a library included in ROS, and it is used for computer vision tasks (e.g. world exploration, object recognition, people tracking), which are essential for robots moving within dynamic environments.

3.2.1.3 OpenNI and NiTE

OpenNI, which is the acronym for Open Natural Interaction, is an open source framework containing softwares, drivers and libraries useful for developing applications which use RGB-D sensors, such as Microsoft Kinect. The human detection and tracking functionalities of OpenNI are contained in a package called NiTE, which provides all the APIs and algorithms useful to estimate the positions of human joints (see figure 3.11) from a depth image (as seen on figure 3.10). NiTE skeletal tracker works at 30 Hz and gives the best results when the user distance from the sensor is in the range included between 1.2 and 3.5 meters. Moreover, when the NiTE tracker is initialized, it requires that the person starts moving in front of the RGB-D sensor in order to detect and start tracking him.

3.2.1.4 Gazebo

Gazebo[14] is a multi-robot simulator. It can simulate the interaction between several objects, sensors and obviously the robots movements inside a three-dimensional world with physics. Gazebo has been a ROS package, until the Groovy version of ROS. Starting from ROS Hydro version, Gazebo has become a standalone robot simulator though it has kept a good degree of integration with it.
An important Gazebo's feature is the possibility for developers to create plugins. These plugins allow to bring into Gazebo new robot models and new sensors in order to make them working in a simulated environment.

[13]PCL official website: `http://www.pointclouds.org/`
[14]Gazebo official website: `http://gazebosim.org/`

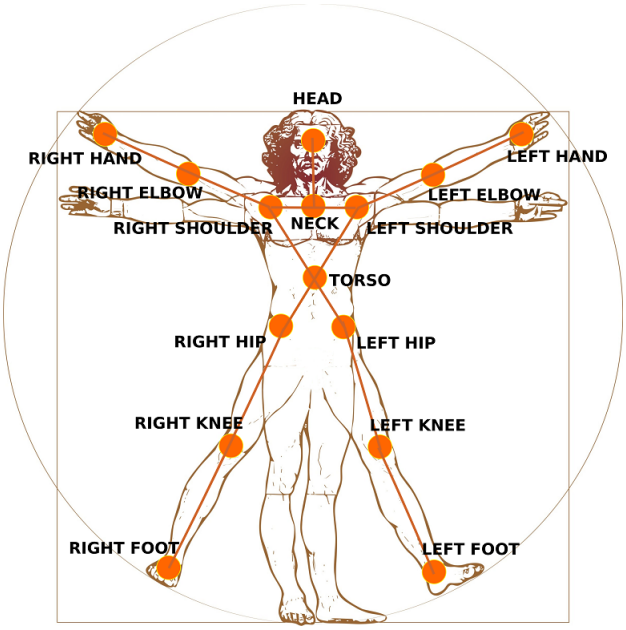

Figure 3.11: Positions and names of the 15 human joints estimated by NiTE skeletal tracker

3.2.2 External Libraries

3.2.2.1 eORL

The Enhanced Open Realistic Robot Library (eORL) is a library supplied by Comau, that allows programmers to develop applications for working on Comau robots, exploiting the potentiality of the C5G Open architecture. This library allows to do the next kind of operations using a real Comau robot (or a virtual one) inside the user application:

- the initialization of a virtual Comau robot on Linux (complete Comau robot family), starting from a configuration file (*.c5g)
- the computation of Direct and Inverse Kinematics
- the calculation of Comau's robot trajectory, using the eORL interpolator
- the computation of the Dynamic Model and of the Jacobian

eORL supplies a simple interface for developers, in fact this library hides all the real time functions needed in order to communicate with the real robot, delegating this part to a callback that is automatically called by eORL library every 400 μs (or 2, 4, 8, 16 ms). The user has the freedom to fill this callback to do what he wants in order to use the Comau manipulator robot with external sensors, using one of the C5G open modalities described in section 3.1.1. The

library is installed by default inside the B&R LPC, but it can be installed also into other Linux PCs.

3.2.2.2 Reflexxes

Reflexxes[15] is a German engineering company specialized in motion generation software for robots and servo drive controllers. It has been founded in 2010 as a spin-off of the Robotics Institute at TU Braunschweig (Germany), where a team of researchers including Torsten Kroeger developed an algorithm for computing robot motion trajectories from arbitrary initial states on-the-fly.
The company offers both commercial and free software implementation of their online trajectory generation algorithms. It also recently joined Google.

The type II version of the Reflexxes motion libraries has a LGPL software license and it can be freely downloaded from the official website[16]. Its strength is the capability to generate instantaneously new targets obtained from data acquired with external sensors. This library has a C++ API and it allows to generate both position and velocity based trajectory.

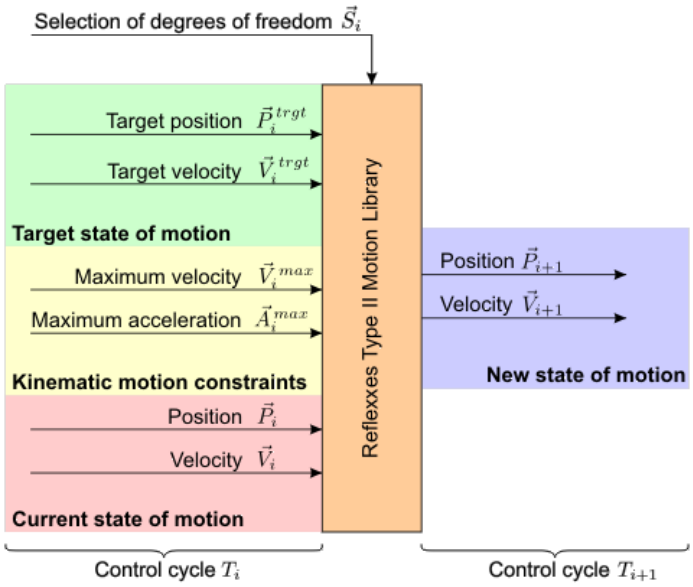

Figure 3.12: Input and output values of the Reflexxes position-based online trajectory generation algorithm

[15]Reflexxes official website: `http://www.reflexxes.com/`
[16]`http://www.reflexxes.com/products/overview-and-download`

Chapter 4

Implementation

In this Chapter we describe in detail the architecture and the retargeting algorithm implemented in this master thesis work. At the beginning we introduce the physical system, how it is composed and how it works. After that, we present the algorithm pipeline, examining in detail the functioning of each block.

4.1 System Architecture

The architecture developed in this work to manage the vision-based teleoperation system can be partitioned in two macro-blocks (see figure 4.1 for a complete overview of the architecture):

1. **The acquisition and retargeting subsystem** consists in a Microsoft Kinect sensor connected to a computer with ROS, which contains all the ROS packages needed for the retargeting of the robot joints, starting from some human operator movements.

2. **The C5G Open subsystem** composed by a B&R LPC, which is a component of the C5G Open architecture, that allows to teleoperate the real Comau Smart5-Six robot.

The described parts of the system can communicate one with each other by using a standard Ethernet network, as we can see in figure 4.1.
It has not been possible to use a unique computer, because ROS cannot be installed on the industrial B&R LPC, responsible of the real robot teleoperation via Powerlink Ethernet. This is because this computer has been tested to work with a real-time kernel and the Powerlink Ethernet only with Linux Ubuntu 10.04, while ROS Hydro[1] requires at least Ubuntu 12.04 installed on the PC.
Nevertheless, also installing on the LPC a newer version of Ubuntu, compliant

[1]ROS Hydro is the lowest ROS version on which the implemented ROS nodes can run.

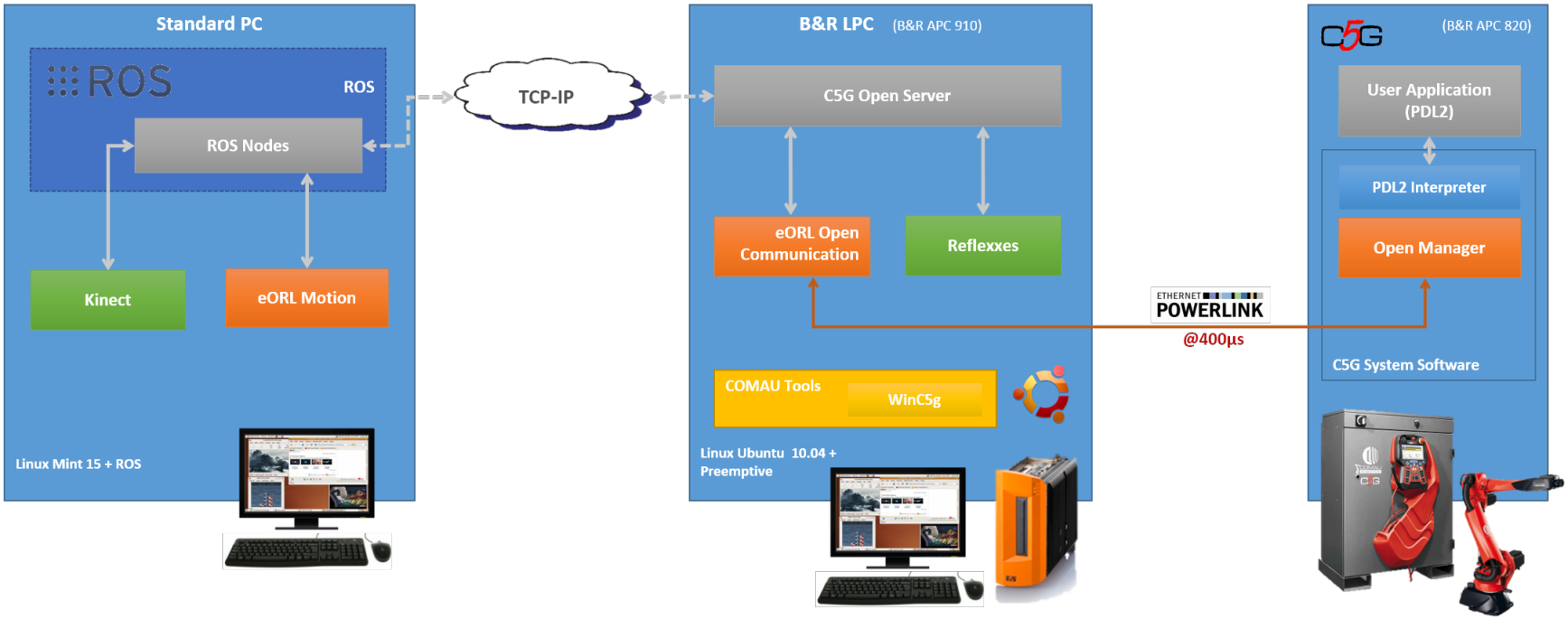

Figure 4.1: This diagram shows the whole system architecture developed in this work, in order to allow the teleoperation of the Comau manipulator robot using the Microsoft Kinect. The blue squares on the center and on the right of the image represent the C5G Open subsystem, while the left blue square represents the PC on which runs all the ROS nodes.

with ROS Hydro, would not have solved the problem. In fact the Powerlink Ethernet drivers do not work reliably on Ubuntu 12.04 running the same real-time 2.6 PREEMPT kernel used for Ubuntu 10.04. So, while Comau engineers will not find a solution to this problem, the simplest way was to implement both the human tracking and the joints remapping into a separate PC with Ubuntu 12.04 and ROS Hydro installed. This computer will then communicate the calculated targets to the B&R LPC via a standard Ethernet network, that will forward these targets to the robot via the Powerlink Ethernet.

4.2 Algorithm Pipeline

This Section presents how the whole algorithm works within the architecture discussed in section 4.1, explaining the implemented software on both PCs. We describe the various ROS nodes, all implemented in C++, from the tracking of the human joints to the retargeting of human motion into the robot joints. After that, we give an overview of the software developed on the B&R LPC, responsible of the motion of the real Comau robot. In particular, we describe in detail also how works the communication between the two computers.

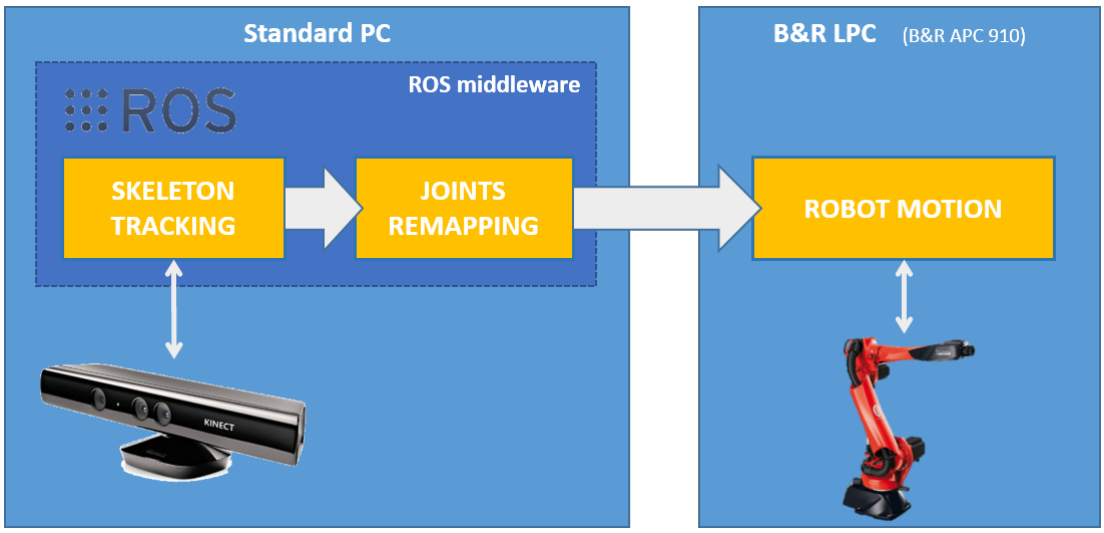

Figure 4.2: The yellow blocks represent the steps of the algorithm pipeline: the first two blocks work within ROS framework on the Standard PC, while the last block runs on the B&R LPC.

4.2.1 Skeleton Tracking

The ROS package responsible of the skeleton tracking of a user in front of the Kinect sensor is *nite2tf*. This package contains the node that must be launched in order to start tracking the human joints. This ROS node publishes the position and the orientation of these joints as TFs, disposed as in figure 4.3. All the frames of the human model are computed from the reference coordinate frame called *camera_rgb_optical_frame*, located on the RGB camera of the Kinect sensor.

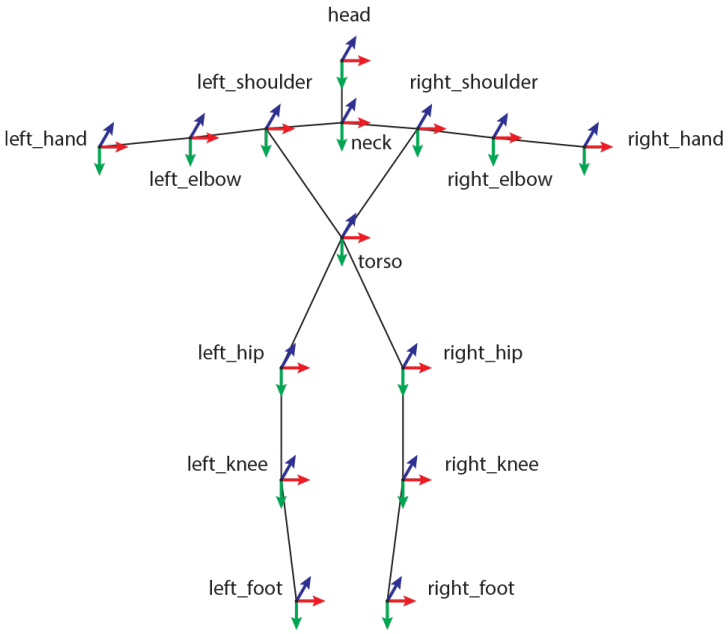

Figure 4.3: On this image are represented the joints and the links of the human model tracked by Kinect during a frontal acquisition with the *nite2tf* tracker. Notice that this model is flipped horizontally, in fact the TFs of the left side of the body are on the right and vice versa.

In order to acquire the precise orientation of the user's hands, requested for a precise remapping of human hands motion onto the robot wrist[2]. The package *nite2tf* has been extended, by adding two more TFs to each arm: wrist and fingertips. This is because the standard information of *nite2tf* package was not enough to acquire a precise orientation of the hand, so it has been necessary to add other custom TFs to the already existing *left_hand* and *right_hand*, which positions correspond to the centroid of the respective hands.
In order to do this, we had to work with the human point cloud given by OpenNI, which is built from depth informations acquired by Microsoft Kinect.
Once acquired the human point cloud, the first thing to do is to segment the point

[2]see section 4.2.3.1 for a detailed discussion on this topic

clouds of each hand, from fingertips to wrist. This segmentation is computed after estimating the positions of both hands centroids, allowing to obtain two clusters of points representing the two human hands.
Now, in order to extrapolate the orientation of both hands, we exploit some results taken from Principal Component Analysis (PCA). Given a set of points distributed in a three dimensional space, using PCA theory, we can find principal directions of the points cluster, which consists in finding the vectors[3] along which the variance of the points cluster is maximum. (see figure 4.4)

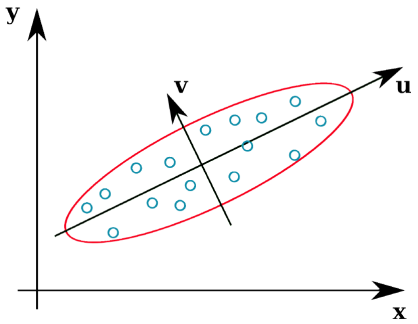

Figure 4.4: This is a graphical example of a bi-dimensional Principal Component Analysis, in which it has been computed the principal components, **u** and **v**, of the blue points cluster on the XY plane.

Formally, let $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$ be a set of points $\mathbf{x}_i \in \mathbb{R}^3$ representing the hand point cloud. The first step to do is to calculate the normalized covariance matrix Σ:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{pmatrix} \tag{4.1}$$

where

- $\boldsymbol{\Sigma}_{ij} = \boldsymbol{\Sigma}_{ji} = \dfrac{1}{n-1} \displaystyle\sum_{k=1}^{n} (\mathbf{x_k}_i - \boldsymbol{\mu}_i)(\mathbf{x_k}_j - \boldsymbol{\mu}_j)^T$
- $\boldsymbol{\mu} = \dfrac{1}{n} \displaystyle\sum_{k=1}^{n} \mathbf{x_k}$, instead, is the mean point in $\mathbb{R}^3$ of the $\mathbf{X}$ vector, i.e. the centroid of the hand point cloud.

The next step is to compute the 3 eigenvectors of the $\boldsymbol{\Sigma}$ matrix. These vectors will be the principal components of our hand point cloud, allowing us to obtain all the information needed in order to achieve the hand orientation. So, computing roots of the characteristic equation:

$$\det(\boldsymbol{\Sigma} - \lambda \mathbf{I}) = 0 \tag{4.2}$$

[3]also called principal components

we obtain the three eigenvalues λ. Once obtained also the three associated eigenvectors, we take the one associated with the greatest eigenvalue. This eigenvector will be the component vector v_{WF} going from wrist to fingertips.
Now, before publishing our new TFs, we must first compute their origin position and orientation. Taking the direction given by v_{WF} we add and subtract half of the hand length from the hand centroid, obtaining the positions respectively of the hand wrist and the middle fingertip.
For the TF orientation, instead, we calculate the needed quaternion by using the angle between the Microsoft Kinect z axis and the v_{WF} vector.

In synthesis, the algorithm for the generation of custom TFs is the next:

1. Segment the hand point cloud starting from the user point cloud
2. Compute the hand point cloud centroid $\boldsymbol{\mu} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x_k}$
3. Calculate the normalized covariance matrix $\boldsymbol{\Sigma}$ of the hand point cloud, given its centroid $\boldsymbol{\mu}$
4. Compute eigenvalues and eigenvectors of matrix $\boldsymbol{\Sigma}$ and get the eigenvector associated with the greatest eigenvalue. It will be called v_{WF} and it is a vector going from wrist to fingertip
5. Take the centroid $\boldsymbol{\mu}$ and add and subtract half the hand length along the direction given by vector v_{WF}, obtaining the origin position of custom TFs
6. Get TFs quaternions by taking the angle between the z axis and v_{WF} vector.

After applying this procedure for both left and right hands point clouds, we can publish our new custom TFs, which names are respectively: *left_wrist* and *left_fingertip* for the right hand, and *right_wrist* and *right_fingertip* for the left hand. The new human model obtained with the modified skeleton tracker can be seen on figure 4.5.

Applying these changes to the skeleton tracking node contained in *nite2tf* ROS package, allow us to easily remap human movements on the wrist joints of the manipulator robot. Before considering joints remapping, we explain basic concepts about the virtual robot model used within ROS and Gazebo, in order to better understand the implemented remapping functions.

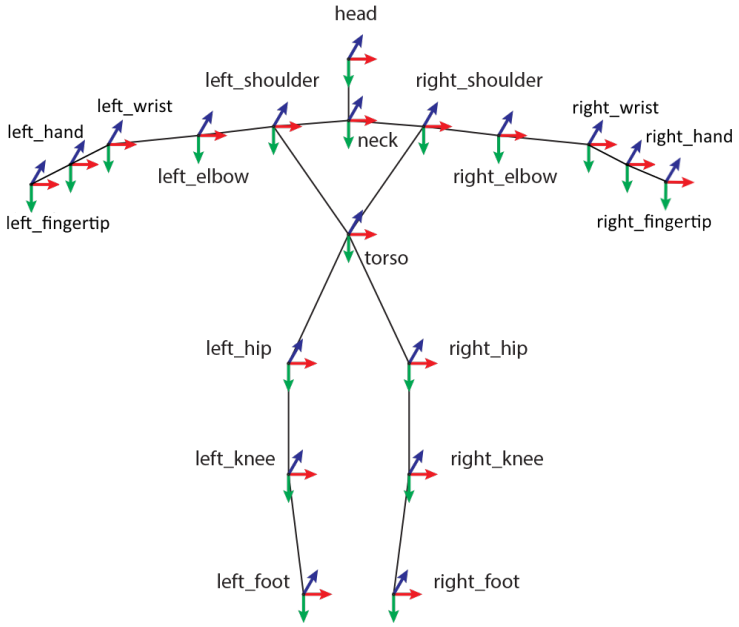

Figure 4.5: This image represents the new human model (in a frontal view) published by the modified skeleton tracker. Respect to the original tracker, the total number of published TFs is now increased by four (two new TFs for each hand).

4.2.2 Robot Model

A ROS package, namely *comau_ model*, has been created to represent the Comau Smart5 SiX through a virtual model compliant with ROS. This package contains the meshes[4] and all the kinematic and dynamic data[5] of the robot, useful to simulate it within Gazebo. So, in this package we can find:

- a *.urdf*[6] file containing kinematic and dynamic data and other informations on the robot links and joints. Its syntax is similar to a XML file;
- the files containing the meshes of the robot links with *.dae* extension.

In a URDF file, are declared both links, which describe the rigid bodies which compose the robot, and joints, which are the components that connect links to each other creating a hierarchy. This file describes formally the virtual robot and it must contain the next informations:

[4]Downloadable freely on Comau website: `http://www.comau.com/eng/offering_competence/robotics_automation/products/low_medium_robots/Pages/smart_5_six.aspx`

[5]Part of this data, such as joints limits and velocities, is freely accessible from the page of Comau website. Other informations as the accelerations and the inertia matrices of robot links have been privately supplied by Comau.

[6]acronym of Unified Robot Description File

1. *Links*[7], declared through tags *<link>*, containing the next tags:
   - *<visual>*, which contains the graphic informations of the link that can be either a URDF primitive(e.g. parallelepiped, sphere , cylinder, etc.) or a mesh in *stl* or *dae* format. This tag must have also the origin point and the RPY angles of the inserted visual element.
   - *<geometry>*, which describes collision bounds of the model link by inserting its geometry. It must also contain an origin and a RPY angle, like *<visual>* tag
   - *<inertial>*, in which must be present the mass, the inertia matrix and origin and angles of the link.
2. *Joints*[8], declared through tags *<joint>*, containing the next tags:
   - *<type>* which can be one of the next typologies:
     - *revolute*: describes a hinge joint that rotate along a specified axis with upper and lower limits
     - *continuous*: analogous to revolute but without upper and lower limits
     - *prismatic*: that is a sliding joint along a specified axis with a limited range
     - *fixed* is a joint with no degrees of freedom, so it cannot move
     - *floating*: is a joint with 6 DoFs
     - *planar*: allows motion in a plane perpendicular to the axis
   - *<origin>*: specifies the origin point and the angles of the joint, the same as links. The joint is located on the origin of the child link
   - *<parent>* indicates the name of the parent link
   - *<child>* indicates the name of child links
   - *<axis>* indicates the rotational axis for revolute joints, translation axis for prismatic joints or the normal of a plane for a planar joint
   - *<limit>* specifies upper and lower values for joints

[7]further informations on links can be found on `http://wiki.ros.org/urdf/XML/link`
[8]further informations on joints can be found on `http://wiki.ros.org/urdf/XML/joint`

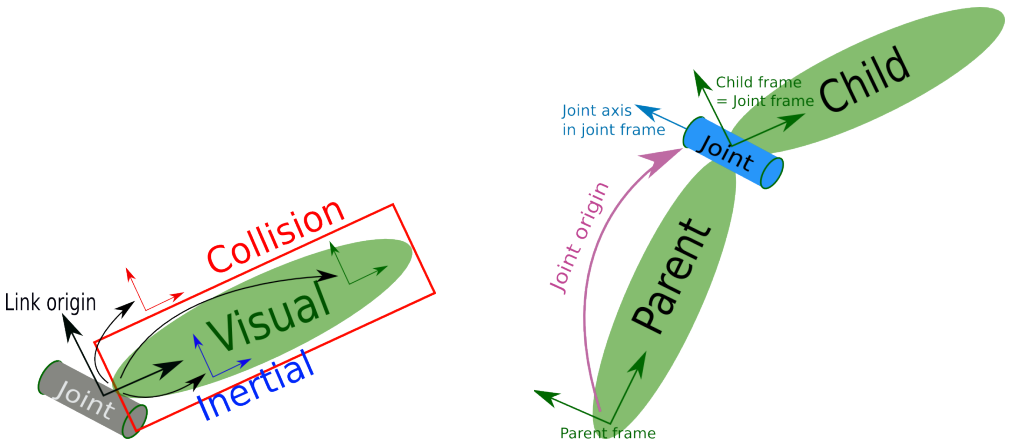

Figure 4.6: Graphical link and joint representations used in the URDF file.

In particular for the modeling of Comau Smart5 Six robot, its URDF file present in *comau_model* package is composed by 8 links:

1. *world*
2. *base_comau*
3. *axes_1*
4. *axes_2*
5. *axes_3*
6. *axes_4*
7. *axes_5*
8. *axes_6*

where the first is a dummy link representing the environment and the other 7 are links of the robot, each one with its mesh. These links are connected between them by 7 joints: 1 fixed joint connecting the *world* link with the *base_comau* link, in order to fix the robot model on the ground, and 6 revolute joints that give to robot the degrees of freedom which compose it. On table 4.2.2 we can see how links are interconnected by joints and what is the Cartesian axis along which they rotate.

Joint Name	**Parent Link**	**Child Link**	**Rot. Axis**	**Direction**
joint_1	base_comau	axes_1	Z	CW
joint_2	axes_1	axes_2	Y	CCW
joint_3	axes_2	axes_3	Y	CW
joint_4	axes_3	axes_4	X	CW
joint_5	axes_4	axes_5	Y	CCW
joint_6	axes_5	axes_6	X	CW

The resulting virtual robot model obtained with the URDF file implemented for the Comau Smart5-Six robot can be seen on figure 4.7b.

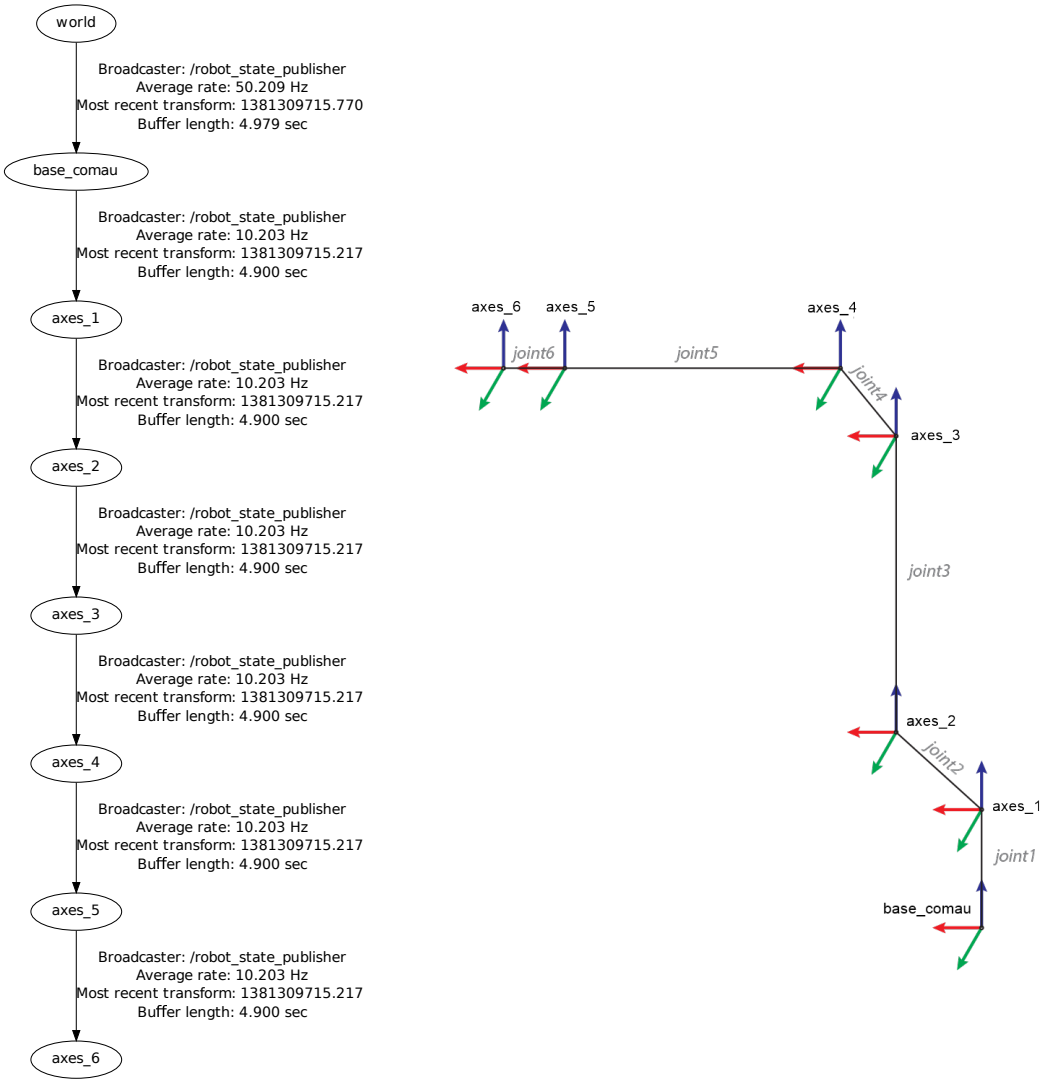

Figure 4.7: **(a)** Represents the TFs tree of the robot model viewed using *view_frames* ROS command.
(b) Shows the TFs position on the URDF robot model implemented within *comau_model* package.

The *comau_model* package allows developer to change the manipulator robot model, by simply changing the robot meshes and the values of the URDF file.

4.2.3 Motion Remapping

Now we can talk about the ROS nodes implemented in order to remap, in real-time, the human joints motion into the robot joints. The nodes responsible of this task are inside the *comau_sim* package and they manage the remapping algorithms in order to transform the joints motion coming from the TFs[9], received from tracker within *nite2tf* package, into joints angles of the virtual robot model[10].
Two ways has been implemented to remap the human movements into the robot using a Microsoft Kinect sensor:

1. **direct joints remapping**: this technique is aimed to remap the motion of the human body parts into the robot links in the most intuitive way for the human actor. This is the main remapping technique.

2. **theory of inverse kinematics**: this technique has been implemented for comparison purposes with the direct remapping method. The theory of inverse kinematics allows to calculate the robot joints angles by acquiring only the hand Cartesian position of the human operator.

4.2.3.1 Direct Joints Remapping

One of the main goals of this work has been to find an intuitive way to move the Comau manipulator robot by simply acquiring the motion of the person, which moves its body parts as if they were robot links. In order to accomplish this objective, it has been created a remapping function for each of the first five robot joints, i.e. joints 1, 2, 3, 4 and 5. Movements for joint 6 has not been considered because its remapping would have not been intuitive to implement, and also because it would have required the acquisition of finer movements by the sensor, such as complex hands movements, which a Microsoft Kinect sensor cannot supply.
For the remapping of all the joints we must first consider a change of Cartesian coordinate system, because human and robot TFs have different coordinate frames. So to pass from the ones of figure 4.5 to the others of figure 4.7b, we have to do the next transformation:

$$\begin{aligned} x &\longleftarrow -z \\ y &\longleftarrow \ \ x \\ z &\longleftarrow -y \end{aligned}$$

From now on, all the coordinates frames of the human skeleton will be considered already transformed in this way.

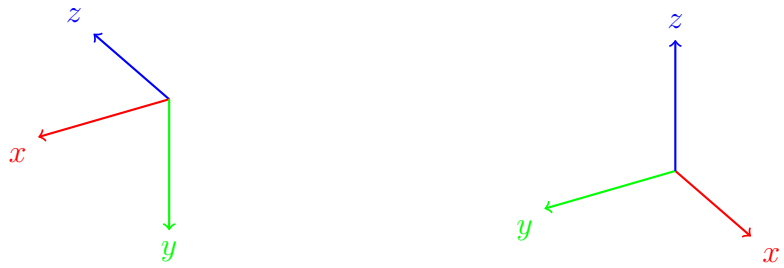

Figure 4.8: On the left the Cartesian coordinate system of the human model and on the right the coordinate system of the robot model.

Joint 1

Joint 1 allows robot to rotate about its axis z. For moving this joint using the informations[11] acquired by Microsoft Kinect, it has been used the orientation of the human actor in front of the sensor.
In order to do this, first we must take the TFs positions of the neck $\mathbf{w}_n$ and of the left hand $\mathbf{u}_{rh}$[12]. Formally:

$$\mathbf{w}_n = (x_n, y_n, z_n) \qquad \mathbf{u}_{rh} = (x_{rh}, y_{rh}, z_{rh})$$

and now from these two points we calculate the vector going from neck to left hand

$$\mathbf{v} = \mathbf{u}_{rh} - \mathbf{w}_n$$

From this vector we can now get the orientation of $\mathbf{v}$ along z axis, by calculating the angle α between the vector projection on XY plane and the x axis (as shown in figure 4.9), using the next formula:

$$\alpha = \begin{cases} \arccos\left(\frac{x_{\mathbf{v}}}{\rho_{XY}}\right) & \text{if} \quad y_{\mathbf{v}} \geq 0 \\ -\arccos\left(\frac{x_{\mathbf{v}}}{\rho_{XY}}\right) & \text{if} \quad y_{\mathbf{v}} < 0 \end{cases} \tag{4.3}$$

where $x_{\mathbf{v}}$ and $y_{\mathbf{v}}$ are respectively the x and y components of the vector $\mathbf{v}$, while $\rho_{XY} = \sqrt{x_{\mathbf{v}}^2 + y_{\mathbf{v}}^2}$ is the length of its projection on the XY plane. α corresponds to the joint 1 rotational angle.

[9] showed on figure 4.5
[10] illustrated on figure 4.7b
[11] reference frames on figure 4.5
[12] we remember that on skeleton model the right is flipped with the left side of the body

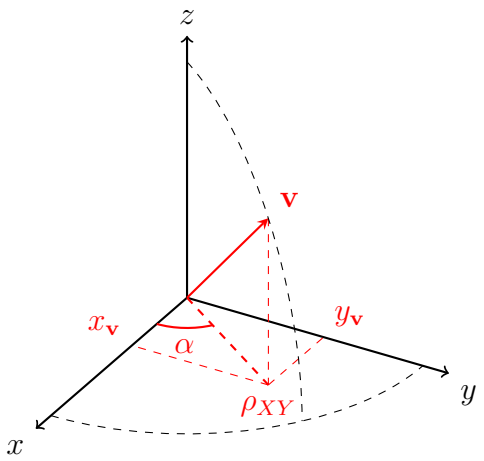

Figure 4.9: Graphical representation of angle α, used for the remapping of joint 1.

Joint 2

This joint allows to rotate the robot about the y axis. This type of movement allows the robot to make a longitudinal advance along the direction of joint 1. The angle needed for the remapping of joint 2 is obtained from the relative position of the human demonstrator respect of its initial tracking position.
Before calculating this angle, that we call γ, we have to get the direction of the person in front of the sensor by using the orientation of the human shoulders. Let

$$\mathbf{v}_{ls} = (x_{ls}, y_{ls}, z_{ls}) \qquad \mathbf{u}_{rs} = (x_{rs}, y_{rs}, z_{rs})$$

be the vector representing the positions respectively of right and left shoulders. Now, we can take the vector

$$\mathbf{w} = \mathbf{v}_{ls} - \mathbf{u}_{rs}$$

that represents the initial direction of the person respect to the z axis. In order to change the coordinate system from the initial to the actual $\mathbf{w}$ orientation about the z axis, we use the next rotation matrix:

$$R_z = \begin{pmatrix} \beta & \alpha & 0 \\ -\alpha & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

where $\alpha = \frac{x_{\mathbf{w}}}{||\mathbf{w}||}$ and $\beta = \frac{y_{\mathbf{w}}}{||\mathbf{w}||}$, with $x_{\mathbf{w}}$ and $y_{\mathbf{w}}$ x and y components of vector $\mathbf{w}$ and $||\mathbf{w}||$ its L^2 norm.

Inverting the matrix R_z we get the matrix

$$R_z^{-1} = \begin{pmatrix} \frac{\beta}{\alpha^2+\beta^2} & -\frac{\alpha}{\alpha^2+\beta^2} & 0 \\ \frac{\alpha}{\alpha^2+\beta^2} & \frac{\beta}{\alpha^2+\beta^2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.5}$$

which can transform the coordinate system given by the vector $\mathbf{w}$, representing the actual orientation of the shoulders, into the coordinate system of their starting orientation.
Once known the shoulder orientation, we must compute the shift of the person respect to its initial position. To do this we take a shift vector $\mathbf{s}$ going from the initial position of the neck to its actual position:

$$\mathbf{s}' = \mathbf{r}_{n,a} - \mathbf{r}_{n,i}$$

where $\mathbf{r}_{n,a}$ and $\mathbf{r}_{n,i}$ are respectively the actual and the initial position of the neck. Now, we have to rotate $\mathbf{s}'$ using rotation matrix R_z^{-1} obtaining a vector $\mathbf{s}$ referred to the starting coordinate system (as shown in figure 4.10):

$$\mathbf{s} = R_z^{-1}\,\mathbf{s}'$$

To convert this vector into an angle for joint 2, we calculate the arccosine of the ratio given by the projection of $\mathbf{s}$ with the length of axis 2 of Comau Smart5 Six, which is 0.59 meters. To this quantity we have to add a $\frac{\pi}{2}$ corrective factor due to the robot model, giving us the final rotation angle γ:

$$\gamma = \frac{\pi}{2} - \arccos\left(\frac{x_{\mathbf{s}}}{\text{axis 2 length}}\right) \tag{4.6}$$

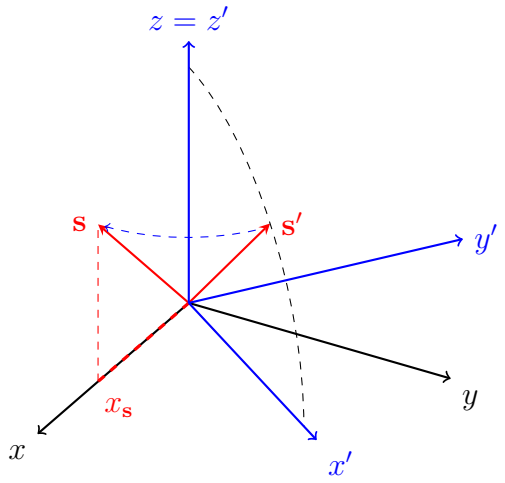

Figure 4.10: Graphical representation of vectors $\mathbf{s}$ and $\mathbf{s}'$ with their respective coordinate systems.

Joint 3

This joint rotates about axis y and is responsible of moving up and down the robot forearm. The angle for this joint has been computed looking the direction of the left arm of human actor respect to his torso. Moreover, because the rotational axes of joints 2 and 3 are the same we have to consider also the inclination γ for joint 2, in order to guarantee a correct angle remapping for joint 3.
Let's take the vector $\mathbf{v}$ already used for joint 1:

$$\mathbf{v} = \mathbf{u}_{rh} - \mathbf{w}_n$$

that represents the vector going from neck to left hand.
Calculating the vertical inclination of this vector respect to the XY plane we get the angle φ' (shown in figure 4.11):

$$\varphi' = \begin{cases} \arccos\left(\frac{\rho_{XY}}{||\mathbf{v}||}\right) & \text{if} \quad z_{\mathbf{v}} \geq 0 \\ -\arccos\left(\frac{\rho_{XY}}{||\mathbf{v}||}\right) & \text{if} \quad z_{\mathbf{v}} < 0 \end{cases} \tag{4.7}$$

where ρ_{XY} is the length of $\mathbf{v}$ projection on XY plane, $||\mathbf{v}||$ is the vector L^2 norm and $z_{\mathbf{v}}$ its component along z axis.
Taking into account the previous considerations about the inclination of joint 2, the final angle will be:

$$\varphi = \gamma - \frac{\pi}{2} - \varphi' = \begin{cases} \gamma - \frac{\pi}{2} - \arccos\left(\frac{\rho_{XY}}{||\mathbf{v}||}\right) & \text{if} \quad z_{\mathbf{v}} \geq 0 \\ \gamma - \frac{\pi}{2} + \arccos\left(\frac{\rho_{XY}}{||\mathbf{v}||}\right) & \text{if} \quad z_{\mathbf{v}} < 0 \end{cases} \tag{4.8}$$

where $\gamma - \frac{\pi}{2}$ is the angle computed for joint 2 without the corrective factor.

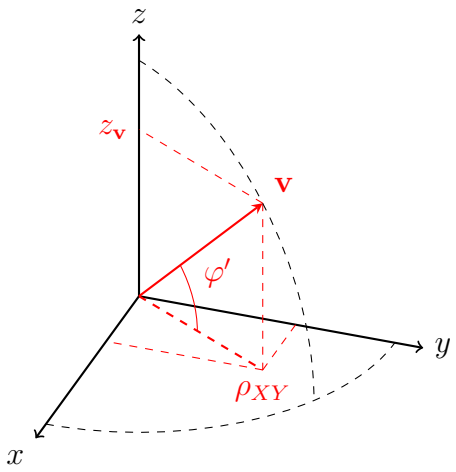

Figure 4.11: Graphical representation of the φ' angle used for the remapping of joint 3.

Joint 4

The joint 4 allows to rotate clockwise or counterclockwise the robot forearm. To rotate this joint like a human forearm, we get the required informations by tracking the human hand rotation about the axis given by the forearm direction. To do this, we can exploit the quaternion contained in the TFs of the hand wrist, but first we have to convert the quaternion to Euler angles, using the Z-Y-X convention. Given the quaternion $\mathbf{q}$ of the left hand wrist[13]:

$$\mathbf{q} = (q_x, q_y, q_z, q_w)$$

we convert it to Euler angles:

$$\begin{cases} \varphi_q &= \text{arctan2}(2(q_z q_w + q_x q_y), (q_x^2 - q_y^2 - q_z^2 + q_w^2)) \\ \theta_q &= \text{arcsin}(-2(q_y q_w - q_x q_z)) \\ \psi_q &= \text{arctan2}(2(q_y q_z + q_x q_w), (q_x^2 + q_y^2 - q_z^2 - q_w^2)) \end{cases} \tag{4.9}$$

where $-\frac{\pi}{2} \leq \theta_q \leq \frac{\pi}{2}$.

The final angle θ for joint 4 has been obtained using the next equations:

$$\theta = \begin{cases} \theta_q & \text{if} \quad \psi_q \geq 0 \\ -\theta_q & \text{if} \quad \psi_q < 0 \end{cases} \tag{4.10}$$

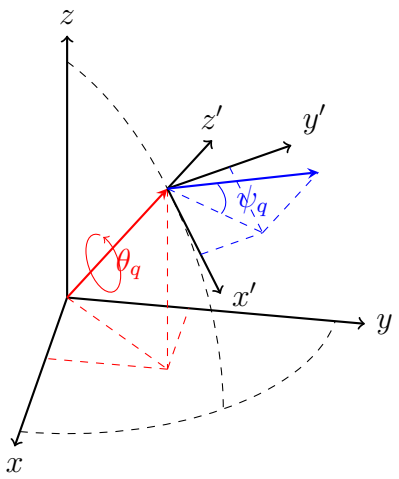

Figure 4.12: This plot represents the angles used for the remapping of the joint 4. The red arrow represents the rotational axis (arm) about which the hand can rotate. The blue vector, instead, represents the hand.

[13] a description on how this quaternion has been computed, look at section 4.2.1

This two cases are necessary in order to correct the direction of the joint rotation (clockwise or counterclockwise) respectively when the hand is pointing up or down.

Joint 5

Joint 5 is the one that moves the wrist of the robot. It can rotate about the y axis and its motion is really similar to the one of human wrist. For the calculation of the angle for this joint, in fact, we will use the inclination of the hand respect to the wrist.
Formally we take:

$$\mathbf{t}_{rs} = (x_{rs}, y_{rs}, z_{rs}) \qquad \mathbf{u}_{ls} = (x_{ls}, y_{ls}, z_{ls})$$

$$\mathbf{v}_{rw} = (x_{rw}, y_{rw}, z_{rw}) \qquad \mathbf{w}_{rf} = (x_{rf}, y_{rf}, z_{rf})$$

which are respectively the vectors of the left and right shoulder, the left wrist and the left fingertips. From these we calculate:

$$\mathbf{a} = \mathbf{u}_{rw} - \mathbf{t}_{rs} \qquad \mathbf{s} = \mathbf{t}_{rs} - \mathbf{u}_{ls} \qquad \mathbf{h} = \mathbf{w}_{rf} - \mathbf{v}_{rw}$$

where $\mathbf{a}$, $\mathbf{s}$ and $\mathbf{h}$ are vectors representing respectively the directions of the left arm (from shoulder to wrist), the shoulders (from the left to the right) and the left hand (from fingertips to wrist).
In order to obtain the orientation of the hand respect to the arm, we have to calculate the angle ψ' between the two respective vectors in the next way:

$$\psi' = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{h}}{||\mathbf{a}||\,||\mathbf{h}||}\right) \tag{4.11}$$

where $\cdot$ is the dot product, and $||\mathbf{a}||$ and $||\mathbf{h}||$ are the L^2 norms of the respective vectors.
Now we have to get the direction for ψ' angle, in order to remap it to the joint 5. For this purpose, we take the vector obtained from the cross product between $\mathbf{a}$ and $\mathbf{s}$. This vector, in fact, is pointing to the side of the plane given from $\mathbf{a}$ and $\mathbf{s}$, in which the angle ψ' should be negative. In order to add the information of the direction, we do:

$$\psi = \begin{cases} \psi' & \text{if} \quad \mathbf{h} \cdot (\mathbf{a} \times \mathbf{s}) \geq 0 \\ -\psi' & \text{if} \quad \mathbf{h} \cdot (\mathbf{a} \times \mathbf{s}) < 0 \end{cases} \tag{4.12}$$

where $\times$ is the vector product. In other words, equation 4.12 correct the sign of ψ' angle, by looking if the hand vector $\mathbf{h}$ has the same direction of vectors given

by $\mathbf{a} \times \mathbf{s}$. If their direction are the same (the dot product is positive), the joint angle ψ is positive, otherwise it is negative.
Now the final angle ψ can be sent to the robot.

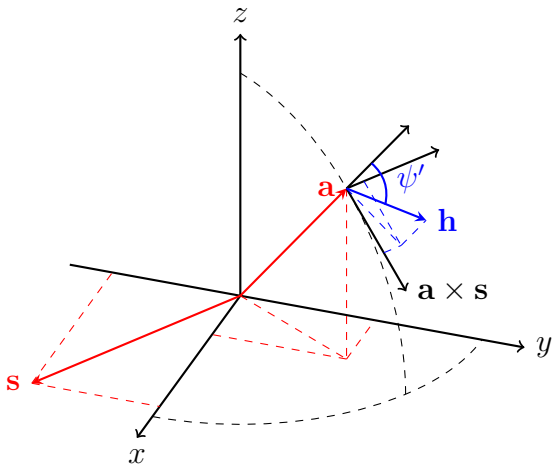

Figure 4.13: This plot represents the angles used for the remapping of joint 5. We can see the vectors **a**, **h** and **s** representing respectively the arm, the hand and the shoulders.

Before sending these 5 angles on topic *joint_ states*, we have first to filter the output values in order to smooth them and to eliminate all the noise coming from the human tracker and from tremors of human movements. In section 4.2.3.2 we will describe the technique adopted for values filtering.

4.2.3.2 Values filtering

In order to eliminate the noise coming from the tracker and from natural tremors of human movements, a smoothing filter has been developed. This filter takes the angles calculated with the direct joints remapping method, computing the arithmetic mean of their last n values. Formally, the equation used in the implemented smoothing filter is the next:

$$\bar{\xi}_k = \frac{1}{n} \sum_{i=0}^{n-1} \xi_{i,k} \tag{4.13}$$

where

- n is the number of values of a single joint angle to be smoothed by the filter;
- k is the number representing the remapped joint index (from 1 to 5)

Using the output values $\bar{\xi}_k$ of this filter instead of using directly the remapped angle values, we can notice a smoother and a less noisy motion on the real robot. the best value of n has been found experimentally by testing different values, in the final version of the code we fixed the value to $n = 20$. This value revealed to be an optimal compromise between a well smoothed remapped motion and a less responsive robot control.
Now, the smoothed joints angles $\bar{\xi}_k$ obtained from this filter can be sent on topic *joint_ states*.

4.2.3.3 Inverse Kinematics Remapping

The package *comau_ sim* allows the human demonstrator also to move the manipulator robot using a retargeting mode based on Inverse Kinematics. This kind of retargeting does not allow to control directly each joint, while it moves the manipulator arm by simply remapping the human hand position into the robot end-effector position. The Inverse Kinematics theory, in fact, can compute the angle of each joint, knowing only the Cartesian position (XYZ coordinates plus Euler angles) of the robot end-effector.
Formally, we take the position of the left human hand from the respective TF:

$$\mathbf{v}_{rh} = (x_{rh}, y_{rh}, z_{rh})$$

Now, in order to convert this position into the robot end effector position, we must first change the coordinate system with the canonical transformation already seen in Joint Remapping section (4.2.3.1):

$$\begin{aligned} x_{rh} &\longleftarrow -z_{rh} \\ y_{rh} &\longleftarrow x_{rh} \\ z_{rh} &\longleftarrow -y_{rh} \end{aligned}$$

and then we have to scale and translate them in the following way:

$$\begin{cases} x_{\text{end-effector}} &= x_{rh} \times 1000 + 2000 \\ y_{\text{end-effector}} &= -y_{rh} \times 1000 \\ z_{\text{end-effector}} &= z_{rh} \times 1000 + 1200 \\ \phi_{\text{end-effector}} &= 0^\circ \\ \theta_{\text{end-effector}} &= 135^\circ \\ \psi_{\text{end-effector}} &= 0^\circ \end{cases} \tag{4.14}$$

where the three coordinate x, y and z have been scaled from meters to millimeters and all the three Euler angles has been fixed. Moreover the two offset values of

2000 and 1200, set respectively on x and z axes, shift onward and upward the coordinate system in order to translate the base frame of the robot in front of the Kinect sensor. This ensures that the robot end-effector will move inside the working area, in front of the robot, avoiding the singularities near the robot base frame.

After these transformations, the 6 values of the equation 4.14 are sent on topic *endEffectorPosition*. Every node that will be subscribed to this topic, will have to calculate the inverse kinematics of these values, in order to get the joints angles of the robot.

4.2.3.4 Comparison between Direct Joints and Inverse Kinematics Remapping techniques

Once implemented these two remapping techniques, we have done some experiments on usability of these two retargeting methods. The main differences noticed during the usage of both retargeting methods are the next:

- the inverse kinematics has more limited movements respect to the direct joints remapping method, because its working area must be bounded in order to avoid singularities;
- with the direct joint remapping method, sometimes it can be difficult to control the end effector position with a certain accuracy;
- with the direct joint remapping method, the human operator can control the angle of each robot joint in a natural and intuitive way: this freedom of control can be particularly useful in crowded industrial environments;
- with the inverse kinematics method, the human operator can control only the end-effector position. This could be a problem in some crowded industrial environments.

4.2.3.5 TCP Server

This ROS node is responsible of forwarding the target values, sent by the nodes within the *comau_sim* package, by creating a TCP server in order to communicate the new target positions via a TCP/IP communication channel to the B&R LPC.
Whenever a message containing the new retargeting values arrives from one of the ROS topics *joint_states* or *endEffectorPosition*, the server creates a TCP packet (57 bytes long) composed in the next way:

- 1 field of 8 bytes (1 unsigned long) for the sequential number of the packet;

- 6 fields of 8 bytes (6 doubles) for the joints values in case of direct joints remapping, or the Cartesian positions in case of the inverse kinematics remapping;
- 1 field of 1 byte (1 char) which indicates the type of remapping chosen. This field will contain the character:
  - `j` for direct joints remapping;
  - `i` for inverse kinematics remapping.

This packet will then be sent by this TCP server node, where it will be received by a TCP client running on the B&R LPC, which is responsible of actuating the robot movements based on the new targets.

4.2.3.6 ROS simulated environment

In this Section, we describe the ROS package created for testing both the remapping methods within a simulated environment. This ROS package has been called *comau_gazebo_plugin* and it includes the code needed to simulate, within Gazebo, the movements of the virtual robot contained in the *comau_model* package.
The *comau_gazebo_plugin* package implementation is modular. In fact, it includes a *ComauGazeboPlugin* class which implements the methods `Load` and `OnUpdate` which are respectively needed for loading and moving the virtual robot inside Gazebo. Moreover, an abstract class has been implemented to expose an interface enabling developers to create and use their own controller for moving the simulated robot inside Gazebo. This interface is called *RobotMovementInterface*, and it contains the next virtual methods:

1. `setMove`: this method is for setting a new desired target movement on the simulated robot;
2. `getNextInterpolationAngles`: it calculates the interpolation steps for each joint of the virtual robot, from a starting position to a final position, in order to simulate the robot motion. This method is called once for each interpolation period (e.g. 2 ms);
3. `getNextInterpolationVelocities`: this method has been created in order to compute the interpolated velocities of each joint, from a starting position to a final position. This method has not been used, but it has been created for future works that will require a velocity controller.

The *comau_gazebo_plugin* package already contains two implemented controller classes which extend the *RobotMovementInterface* abstract class (see figure 4.14):

- *RobotMovementPID*: this class computes the interpolated angles using a PID controller;
- *RobotMovementComauLib*: this computes the interpolated angles using the Comau eORL library.

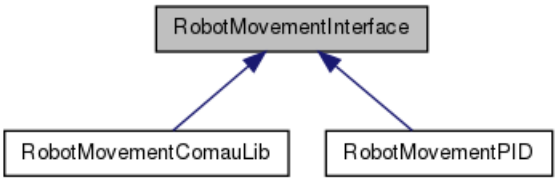

Figure 4.14: Inheritance of the *RobotMovementInterface* abstract class

These controllers are mutually exclusive. In fact during compiling phase, the system checks if the eORL library is installed: if it is installed it will use *RobotMovementComauLib* class to simulate the robot movements within Gazebo, otherwise it will use a simple PID controller included in *RobotMovementPID*. This double controller has been created in order to allow also who has not a license of eORL library to use the remapping methods previously described in a simulated environment.

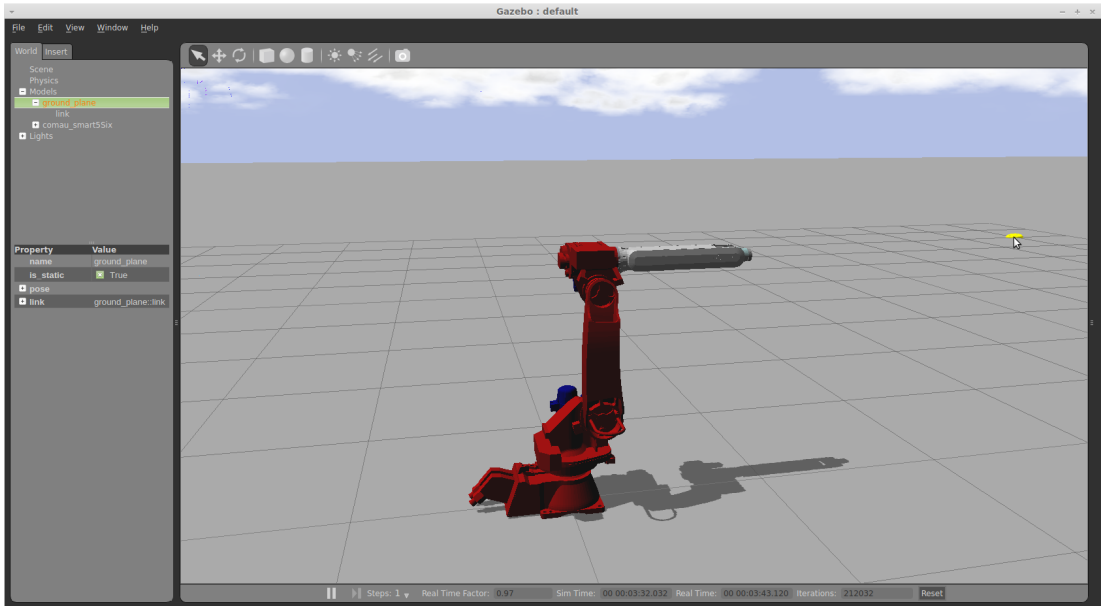

Figure 4.15: Screenshot of Comau Smart5 Six robot simulated within Gazebo.

4.2.3.7 ROS Node Graph

Now we show graphically the various ROS nodes implemented and how they are interconnected between them with topics. We remember that all these nodes have been implemented in C++ and they all work on the same computer, as it was illustrated in Figure 4.2.

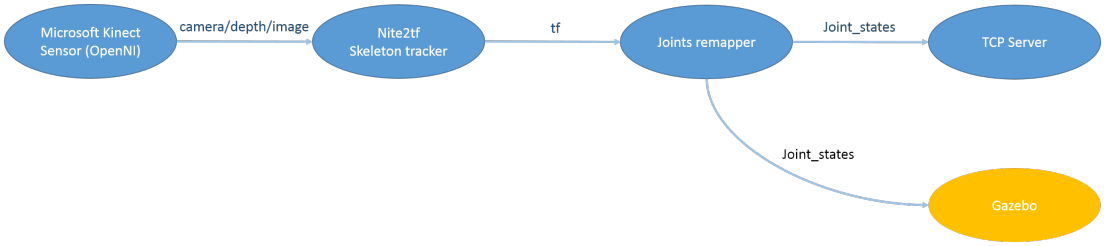

Figure 4.16: In this graph are represented the ROS nodes that allow to do the direct joints remapping. The arcs connecting the various nodes represent the topics on which nodes communicate each other. The node *Gazebo* is yellow because it is not necessary in order to make the system work.
In case of inverse kinematics remapping, the only thing that changes on this graph is the topic *Joint_states* that becomes *endEffectorPosition*, but the structure of the graph remains the same.

4.2.4 Robot Motion

The task of moving the Comau robot is entirely done by an application, written in C++, that runs on the B&R LPC. As previously said on the TCP server section, this application works as a TCP client receiving the new joints from the computer on which is running ROS with the TCP server node.
This software, that we call *C5G Open server*, uses the eORL library:

- for the real time communication methods of in order to communicate with the Comau robot;
- to compute the interpolation steps;
- to calculate the inverse kinematics.

This program has a simple structure. It is essentially composed by two methods: a main and a callback. In the main method are initialized all the variables needed, and are also established the next two connections:

- one as a TCP client toward the ROS PC;
- one as a real-time Powerlink communication server toward the robot controller.

The callback is a hard real time method called every two milliseconds, that does the next steps:

1. it checks if a new TCP packet with a new target has arrived. If the type of the remapping chosen in the packet is `j` then the new joints values received are saved, otherwise (if the remapping mode is `i`) it computes the inverse kinematics of the Cartesian position received and then it saves the calculated joints values.
2. if the controller is in *CRCOPEN_POS_ABSOLUTE* modality, the new joint values arrived from the TCP server are saved as a new target.
3. If a new target has arrived, it is assigned as a new movement to the interpolator and it is computed the next interpolation step. The new target is set also if there is an old pending movement.
4. The last interpolation step computed is finally sent to the robot.

In order to make the robot controller go in *CRCOPEN_POS_ABSOLUTE* modality, the user must run a PDL2 program that loads this C5G Open modality[14] on the C5G controller, allowing to make the robot work with the implemented system.

[14] described in section 3.1.1

It is important to guarantee that the computational time of the operations performed during the callback must stay under 2 ms, otherwise the robot controller will not receive the packet in the established time, generating a timeout error.

During first tests of this architecture, we noticed that sending a new target to the robot controller automatically aborted the precedent one, causing a very slow and rough motion. This behavior is due to the interpolator available through the eORL library. In fact, the robot is required to stay still before eORL can communicate it a new target. This implies that the robot is continuously slowed down when a new target is assigned.
Moreover, since the ROS retargeting system previously described publishes a lot of new target during the human actor movement, this problem is even more noticeable and the only possible solution is to change the used interpolator.

4.2.4.1 Use of the Reflexxes Library

In order to solve this problem, we exploited interpolation algorithms present within the Reflexxes library, instead of the one inside eORL. In fact, the main feature of Reflexxes library is the real-time computation of interpolation steps during a continuous assigning of new targets.
In order to use the Reflexxes interpolator algorithm, in place of the eORL interpolator, we have first to initialize all the variables, the maximum velocities and the maximum accelerations of the robot axes[15], and also to set the interpolation step period. After, we can finally substitute all the calls of the eORL interpolator with the equivalent methods present in the Reflexxes library.
Once these changes have been done on the C5G Open server application, we obtain a system that can manage all the new targets coming from the ROS PC, avoiding any slowing down of the robot motion.

The implemented C5G Open Server application allows to use the described retargeting techniques with all the Comau robot models. In fact, it is enough to change the values of maximum velocities and maximum accelerations of the axes of the robot model in order to use the C5G Open Server with another robot.

[15] taken from table 4.2.2

Chapter 5

Conclusions and Future Works

5.1 Conclusions

The programming of Comau industrial robots can be currently done using only TPs or computer keyboards. Nevertheless this robot programming technique can take a long time to move a robot in specific positions. For this purpose, the teleoperation technique presented in this master thesis allows to move an industrial manipulator robot, controlling directly the joints motion, just using a markerless based vision sensor that does not constrain the human operator to hold or to wear any physical devices.

In this work, we have implemented a system which allows to teleoperate, in real time, a Comau industrial robot in an intuitive way using a RGB-D sensor, which in our case is a Microsoft Kinect sensor. The implemented software packages exploit the potentialities and the modularity of both ROS and the C5G Open architecture, allowing to apply the developed retargeting algorithm on each model of the Comau robots family.
The implemented system revealed to be strongly reliable and fast on a Comau Smart5 Six model in all the tests performed on real Comau robots. Moreover, the system has been successfully tested also on a Comau 7-axes prototype robot, whose tests revealed to be as stable as those performed with the Smart5 Six model.

Thanks to this modular architecture, developer can easily modify for example the retargeting algorithms or the tracking sensor. In the next Section, in fact, we will propose and describe some of the future works that could be done in order to improve the actual retargeting system.
Part of the software developed during this master thesis work has been used as base in some recently published works [17, 18, 21].

5.2 Future Works

In this Section, we present some possible improvements that could be brought to the system implemented during this master thesis work.

5.2.1 Integration of the C5G Open Server within ROS

An important simplification that could bring to the developed system architecture is the usage of a single computer for the robot teleoperation (see Figure 5.1). As hinted on Section 4.1, this can be achieved by porting Powerlink Ethernet drivers in Ubuntu 14.04 Linux with a PREEMPT real-time kernel. In this way it will be possible to install ROS Indigo and to import all the ROS packages implemented for the teleoperation task, in order to run them on the LPC. Once this real-time communication will work, it will be possible to control the real Comau robot directly from a ROS node which will implement the C5G Open Server, avoiding the use of two different computers.
Some tests has already been done, but until now, the Powerlink drivers revealed not to be reliable yet on Ubuntu 14.04. However, further investigations will be done in collaboration with Comau in order to solve these real-time communication problems with the newer Ubuntu distributions.

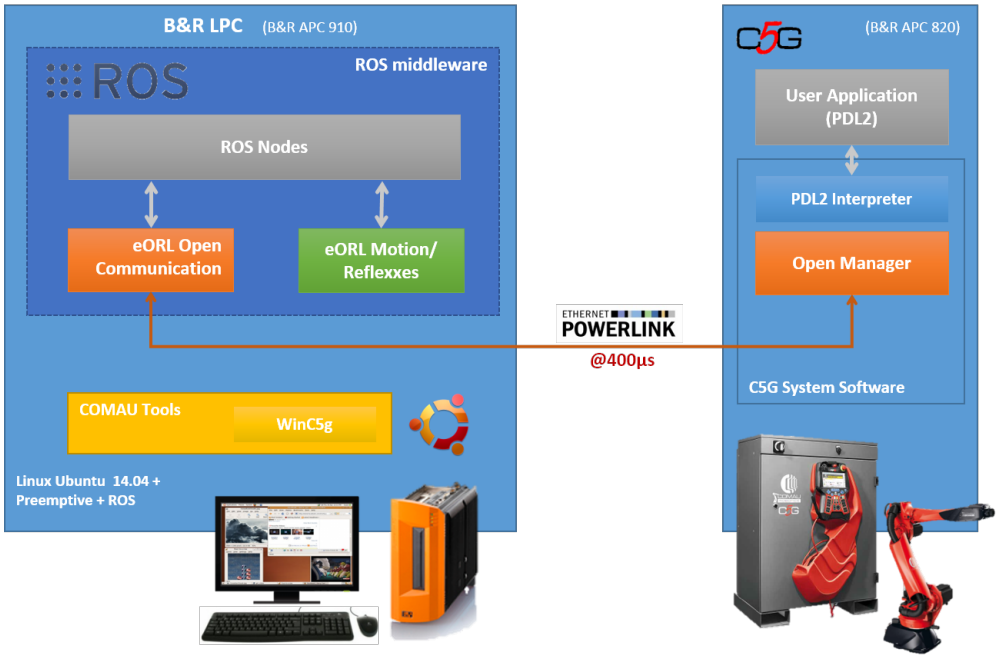

Figure 5.1: In this image we can see a simplified version of the architecture described on section 4.1, where ROS nodes and the C5G Open Server are running on the same B&R LPC.

5.2.2 Collision Avoidance

An interesting improvement for the implemented retargeting system is its integration with a collision avoidance system. Comau has developed a real-time distributed system that allows to modify in real-time the override of their robots in order to decelerate the axes motion, avoiding collisions between the manipulator and the other environmental components located inside the same working cell [20].
This system has been developed as a client-server architecture: in this way a PC connected through TCP/IP to the C5G controller unit can get, at each instant, the updated position of the robot, allowing to properly perform collision avoidance. At this purpose, V-REP simulator[1] has been used in order to calculate the distances between all complex objects located in the working cell. In fact, V-REP contains a fast algorithm able to calculate minimum distances between the various meshes present in the scene. In synthesis, the algorithm works as follow:

1. update the robot position inside V-REP, according to the real position given from the C5G controller, via TCP/IP;

2. calculate the minimum distances between the virtual robot and all the other objects on the scene;

3. compute the override to be sent to the real robot.

The robot velocity override is computed based on the next law:

$$v_k = \begin{cases} v_{k-1} & \text{if} \quad d > d_{max} \\ v_{k-1} \cdot \frac{d - d_{min}}{d_{max} - d_{min}} & \text{if} \quad d_{min} < d < d_{max} \\ 0 & \text{if} \quad d < d_{min} \end{cases} \tag{5.1}$$

where v_k and v_{k-1} are respectively the override values at times k and $k-1$. In equation 5.1, the override decrements linearly its value if the robot is approaching to a warning zone[2] at distance d_{min} from an obstacle. If the robot is inside the warning zone, the robot motion is stopped. While, in case the robot is moving away from an obstacle, the override is linearly incremented to reach $v_k = 100$.

This collision avoidance module has already been tested on the C5G Open architecture and would be easily integrable on our system.

[1] V-REP official website: `http://www.coppeliarobotics.com/`
[2] a warning zone is a tridimensional space that encloses an item

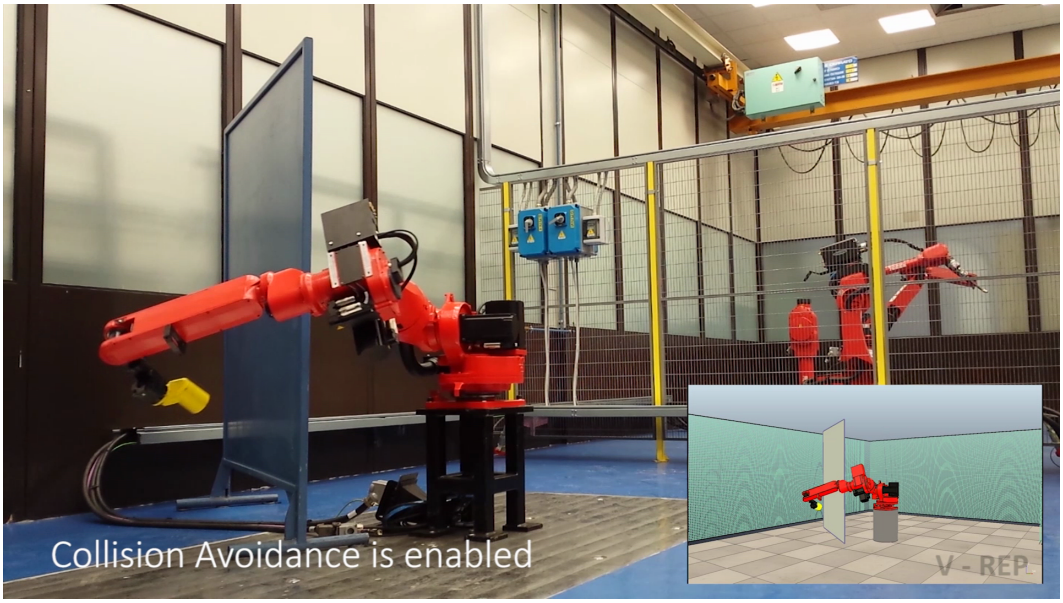

Figure 5.2: In this photo there is a working cell with a Comau robot moving near a vertical panel. On the bottom right of the image is represented the same working cell within V-REP simulator.

5.2.3 Use of Leap Motion Sensor

For the teleoperation of the Comau robot, and in particular for the wrist movements, a Leap Motion sensor could be used instead or along with the Microsoft Kinect.
Leap Motion sensor is a small and cheap device that allows to track complex hand gestures. It can manage the movements of all ten human hand fingers simultaneously with an accuracy of 0.01 mm and a frame rate of 300 fps. This device is composed by three IR led emitters disposed in a row, with two IR cameras in the middle. This characteristics allow to track a surface area of 24 cm^2 with a wide field of view up to 150° [19].

Thanks to an already existing ROS package[3], it would be easy to integrate the use of Leap Motion in order to teleoperate Comau robots, by interfacing the Leap Motion sensor with the already implemented ROS packages. Moreover, such a small device could be integrated on the Comau Teach Pendant in order to allow the programming of Comau manipulator robots using this sensor.

[3] `http://wiki.ros.org/leap_motion`

Figure 5.3: The Leap Motion sensor.

Bibliography

[1] Siciliano B., Khatib O.: *Handbook of Robotics*, Springer, (2008)

[2] Siciliano B., Sciavicco L., Villani L., Oriolo G.: *Robotics - Modelling, Planning and Control*, Springer, (2009)

[3] Dariush B., Gienger M., Arumbakkam A., Zhu Y., Jian B., Fujimura K., Goerick C.: *Online transfer of human motion to humanoids*, International Journal of Humanoid Robotics, (2009)

[4] Kofman J., Wu X., Luu T. J., Verma S.: *Teleoperation of a Robot Manipulator using a Vision-Based Human-Robot Interface*, IEEE, (2005)

[5] Marinho M. M., Geraldes A. A., Bò A. P. L., Borges G. A.: *Manipulator control based on the dual quaternion framework for intuitive teleoperation using Kinect*, Brazilian Robotics Symposium and Latin American Robotics Symposium, IEEE, (2012)

[6] Ibrahim A. R., Adiprawita W.: *Analytical Upper Body Human Motion Transfer to Naohumanoid Robot*

[7] Bisson A., Busatto A., Michieletto S., Menegatti E.: *Stabilize Humanoid Robot Teleoperated by a RGB-D Sensor*, PAI AI*IA, pages 97 – 102, Citeseer, (2013)

[8] Stanton, C., Bogdanovych A., Ratanasena E.: *Teleoperation of a humanoid robot using full-body motion capture, example movements, and machine learning*, Proc. Australasian Conference on Robotics and Automation, (2012)

[9] Pham H.L., Perdereau V., Adorno B. V., Fraisse P.: *Position and Orientation Control of Robot Manipulators Using Dual Quaternion Feedback*, International Conference on Intelligent Robots and Systems, IEEE, (2010)

[10] Kenwright B.: *A Beginners Guide to Dual-Quaternions*, The 20th International Conference on Computer Graphics, Visualization and Computer Vision, WSCG, (2012)

[11] Jeong D. H., Ziemkiewicz C., Ribarsky W., Chang R.: *Understanding Principal Component Analysis Using a Visual Analytics Tool*, Charlotte Visualization Center, Purdue University

[12] Smith L.: *A tutorial on Principal Component Analysis*, Department of Computer Science, Otago University, (2002)

[13] Michieletto S., Chessa N., Menegatti E.: *Learning how to approach industrial robot tasks from natural demonstrations*, ARSO, (2013)

[14] Werber K.: *Intuitive Human Robot Interaction and Workspace Surveillance by means of the Kinect Sensor*, Lund University, (2011)

[15] Khoshelham K., Elberink S. O.:*Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications*, Sensors, (2012)

[16] Romanelli F., Ferrara V.: *C5GOpen: The latest generation of the Industrial Robots Open Control System for University and SMEs*, Comau, (2014)

[17] Munaro M., Antonello M., Moro M., Ferrari C., Clemente G., Pagello E., Menegatti E.: *FibreMap: Automatic Mapping of Fibre Orientation for Draping of Carbon Fibre Parts*, In IAS-13 Workshop Proceedings: Workshop on ROS-Industrial in European Research Projects, pp. 272 – 275, Padova, Italy, (2014)

[18] Michieletto S., Tosello E., Romanelli F., Ferrara V., Menegatti E.: *ROS-I Interface for COMAU Robots*, in Simulation, Modeling, and Programming for Autonomous Robots, Lecture Notes in Computer Science, volume 8810, pages 243 – 254, Springer, (2014)

[19] Bassily D., Georgoulas C., Güttler J., Linner T., Bock T.: *Intuitive and Adaptive Robotic Arm Manipulation using the Leap Motion Controller*, International Symposium on Robotics (ISR) - Robotik, (2014)

[20] Fenucci A., Indri M., Romanelli F.: *A Real Time Distributed Approach to Collision Avoidance for Industrial Manipulators*, (2014)

[21] Tosello E., Michieletto S., Bisson A., Pagello E., Menegatti E., *A learning from demonstration framework for manipulation tasks*, ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of, pages 1 – 7, VDE, (2014)

Acknowledgements

Voglio ringraziare innanzitutto la mia famiglia per il sostegno, sia economico che motivazionale, datomi in questi anni di carriera universitaria.

Grazie al mio relatore, prof. Emanuele Menegatti, e al mio correlatore, Stefano Michieletto, per la passione trasmessami per la computer vision e la robotica e per avermi appoggiato nella decisione di fare il tirocinio in Comau, nonostante l'interminabile burocrazia.

Grazie a tutti i dottorandi e post-doc per la simpatia e per gli utili insegnamenti dati durante questo anno e mezzo di permanenza in IAS-LAB.

Grazie a tutti i miei amici: Anna, Baso, Beppe, Chiara, Gian, Jack, Matteo, Pull, Valentina e Zigio per la compagnia e per allietare le serate nei fine settimana.

Grazie a tutti gli amici e compagni di corso conosciuti in questi anni di università.
In particolare un grazie a Fabio B., Fabio G. e Filippo per i divertenti momenti di pausa tra una lezione e l'altra al DEI, fin dai tempi del secondo anno della triennale.
Grazie a Elena e Gian per il supporto durante questo ultimo anno di magistrale e per il gruppo del buongiorno mattutino.
Grazie ad Andrea, a Marco e a tutti gli amici conosciuti durante la magistrale per i momenti divertenti passati assieme.

Grazie a tutti i miei futuri colleghi in Comau, in particolare a Valentina, per avermi dato la possibilità di fare la tesi in azienda, per gli utili consigli sul lavoro di tesi e per la fiducia ripostami durante tutto il periodo di stage.

Grazie anche a tutti i laureandi del Robolab per le conversazioni interessanti e per i momenti di relax alle macchinette del caffè.