Robot workspace sensing and control with Leap Motion Sensor

Guillem Solé Bonet

LUNDS UNIVERSITET

Department of Automatic Control

Msc Thesis
ISRN LUTFD2/TFRT--9999--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck
Lund 2014

Abstract

The present thesis deals with the design and testing of an appropriate software interface that allows a user to control a robot using a Leap Motion Sensor while defining and keeping a safe workspace for the robot to operate.

The Leap Motion Sensor is a small device able to sense human hands above it and to keep track of them. Hence, when controlling the robot, the user will feel an interesting touch-free control experience.

Distinct control modes, such as linear commandment, joint-by-joint control or specular imitation have been studied and implemented. The basis for a future teaching mode, where the robot could remember user actions and play them accordingly to fulfill a complex task has also been settled.

At the same time, a precise definition of the workspace, creating a safe environment for both the robot and the user, and the contemplation on how to avoid undesired situations have been consciously considered.

Acknowledgements

These people and organizations helped me immensely with my work and I would like to mention them while thanking them for everything they have done:

- To my supervisor Anders Robertsson and to my examiner Rolf Johansson, for being there while I required anything and for providing me with valuable advice every time I found difficulties. To Lund University in general, for letting me use the robotics laboratory and for supplying me with a Leap Motion Sensor.
- To the ErghisTech people, Örs-Barna and Holger, for their manifest interest in my work, their technical support and their valuable advice.
- To my laboratory colleagues, from which I would like to specially mention Patrik, Jorge, Alberto, Dani, Victor and Silvan, for those long debates about how to approach next stages of my work and the incommensurable help at some key points of my work.
- To my family, who in spite of the distance I have felt very close at all times. Their support has been fundamental to keep going during the hardest times in which I was not sure to finally succeed in even moving the robot.
- To my friends in Barcelona and my new friends in Lund, who showed interest in my project and whose cheers helped a lot and became an extra motivation to keep on working.

To all of them I wish the best in their future, professionally and personally, and hope to see, and maybe work with, in another occasion.

Contents

Preface

Once, reading *Physics of the Impossible* [1], by Michio Kaku (San José, California, USA, 1947), I got surprised when along a full chapter the author reveals how far intelligent robots are from current human possibilities.

Besides the not of little interest discussion about the possibility of achieving or not a real artificial intelligence, where the fundaments of the very concept of thinking is revised and the "Turing test"[1] presented, I found specially interesting the dissection of robots apparently insurmountable limitations and their causes.

He uses the example of a fruit fly that, with a tiny brain containing only 250.000 neurons, is able to easily fly around an unknown three-dimensional environment, avoiding all possible obstacles while performing complex flight paths effortlessly. On the other hand, the descendants of SHAKEY (the first robot able to navigate around the "real world") still suffer to navigate around a two-dimensional space with only square and triangular obstacles, in spite of the fact that they equivalently have a much bigger computational power than the fly.

At that point I realized how much methods need to change to bring robotic sciences to a next level I thought was around the corner. If one thinks, there is no way within the mathematics or the infinitesimal calculus able to express such truths as than a rope can pull but not push, animals do not like pain or than time cannot go backwards.

Kaku talks about two different teaching strategies. He distinguishes between a top-down approach, where the programmer implements into the robot the knowledge he wants it to have as a predefined set. In the case of the fly, it would

[1] The "Turing test" ends with fruitless philosophical discussions about whether a machine can think or not and states that if a man cannot see the difference between the answers of a man and a machine when talking to them, the machine passes the test. The chapter, and the whole book, is highly recommendable to anyone that could be interested in such topics.

consist on designing an algorithm with recognition patterns to make the robot able to identify obstacles, and it would simply work or not.

In a bottom-up approach, on the contrary, the robot is given no knowledge but the means to learn by itself. In the previous case it would learn how to avoid obstacles by virtue of colliding lots of times with them (as a baby that is learning how to walk).

Both approaches have shown virtues and defects to this day. To find a balance between these two perspectives may be the key for powering robot capabilities in the future society.

However, when thinking of teaching a robot that has room for learning the question "How?" arises. Given the nature of the bottom-up approach, intuitive natural ways for humans to teach robots are required, since a pure programmatic set of knowledge to be implanted in the robot has few potential to evolve (would be close to the top-down approach instead).

With all these ideas in my mind and a brand-new Leap Motion Sensor the University of Lund put in my hands, I decided to explore the interesting world of robotics from this teaching-learning approach, ready to encounter the difficulties that by all means one can expect when getting into a new field of study but also with the inherent thrill this causes.

***

Therefore, the aim of the present thesis is to explore the possibilities a Leap Motion Sensor can offer when trying to control a robot and, furthermore, to teach it. Such an undertaking venture must be properly approached and a step analysis of goals to accomplish can be helpful.

Hence, prior to any move, the forthcoming efforts are guided to fulfill the following, by order:

- (Preliminaries) To do a research about previous works on this direction. To explore existing robot control options and teaching methods. To study the components at disposal and their possibilities.
- To create a network among the components at disposal through which data can flow from one to another.
- To develop an interface in order to control the robot in real-time using the before mentioned sensor. To define a safe workspace before performing any test.

- To approach the teaching concept creating ways for the system to store movements as basic actions and making it able to combine them in order to obtain more complex operations.

Obviously, when studying thoroughly each point, difficulties appear and new sub-objectives to achieve must be settled in order to progress. An accurate explanation of each case can be found within the respective section in the following chapters.

1. Preliminaries

Hereunder a short background analysis and components introduction can be found. This gives an idea on how the present work, despite being innovative, has not been the first step in this direction and how this thesis has relied on existing components and previous studies to depart.

1.1 Background

There is quite literature regarding ways of intuitive and natural human control of a robot. The use of cameras, specially the popular KINECT, has been widely tested [2; 3; 4]. These attempts have been based on improving the computational vision of the robot and getting data from the user through his body and the motions he performs with it. Thus, several gestural commands can be understood by the robot and specular imitations, for example, can intuitively be performed.

A deep understanding of natural human gesticulation by a robot is difficult to achieve however. This occurs because the segmentation of human gestures can be ambiguous (i.e., the switch between two consecutive gestures may carry transient human movements that may modify the interpretation of the two genuine gestures or even become a third one) and also because the spatio-temporal variability involved in all human actions, even in gestures made by the same user [5].

Nevertheless, interesting attempts have been made to classify, distinguish and interpret human gestures. These can be categorized, going from the more spontaneous and natural to the more socially regulated ones, in the following categories: gesticulation, languagelike gestures (which may replace words during speech), pantomimes, emblems and sign language. The first one, which is based on hands and arms movements, consists of about the 90% of total human gestured communication [5].

Regarding the learning capability of a robot, Paul M. Yanik et al. [2] define this feature as the mechanism by which some manner of feedback is used to

improve the future responses of this robot. In their work, they try to avoid labeling and classification of gestures in benefit of a correct learning by means of trustable user feedback.

In the search for this learning capability several options appear, most of them conceived as the seeking of a proper *policy* able to maximize a *reward* function. Different ideas can be found in [6], where examples of robots that actually learned tasks through several trial-and-error methods are depicted. It is interesting as well the Latent Space Policy Search presented by Kevin Luck et al. [7], where complex humanoid redundant robots learn to pose on one leg staying balanced by these learning methods.

While the robot is assimilating knowledge through these procedures, the Neural Gas (NG) algorithm by Martinetz and Schulten [8] and the improved Growing Neural Gas (GNG) by Fritzke [9] can literally build a knowledge net similar to the way the human brain learns, settling the new information by means of connecting existing and new nodes one to another, thus expanding the web.

On the other hand, professor Rolf Johansson et al. [10] have focused their studies on this field on the segmentation part of knowledge acquisition. The so called autonomous segmentation works with the idea of making robots able to perform the segmentation of the taught actions by themselves. They illustrate this with the example of cooking rice: imagine a robot that is intended to learn how to cook rice. The point would be that when copying from the "teacher", it could distinguish sub-actions in this task of cooking rice, such as pouring water, stirring the pot or scooping rice with a spoon. This would have tremendous potential because it could easily be asked to repeat any of these sub-actions or even to change the order of some of them, obtaining thus a huge potential of new derived actions to be performed.

1.2 Components

To perform the desired control in the present work, several components have been used. All of them were preexisting and their usage in this thesis has been courtesy of the Lund University Department for Automatic Control, which ceded them to the disposal of the author.

The control action here contemplated required at least three elements:

- A sensor: used to transform user gestures into computational data.
- A robot: used to see the results of the control system.

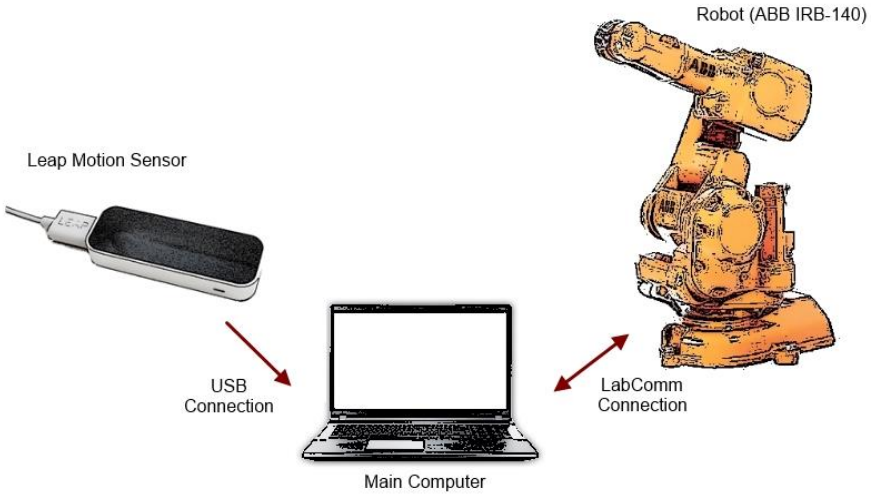

Figure 1.1. Preliminary conception of system composition and data flow direction.

- A main computer: to manage all signals, compute outgoing data and schedule threads on both sides.

In order to connect all these components a proper channel of communication was also required. In this work the LabComm connection has been used to connect the Main Computer and the robot.

A careful explanation of each of these components can be found in the following points of this report. A basic scheme with the components that have been used in this project and the direction in which data can flow among them can be seen in Figure 1.1.

Although these components are great tools by themselves, here has been intended to show that sometimes a combination of items can give place to something bigger than the mere addition of the parts. The usage of these components as parts of the control interface has sought this idea the whole time.

Leap Motion Sensor

The sensor role has been assumed by the Leap Motion Sensor. It is a desk device developed by Leap Motion, Inc. consisting on a small light sleek sensor able to detect human hands above it, alongside with their 10 fingers.

It physically occupies a small space on the desk but it is able to sense a notable volumetric space above itself. It is thought as a human-machine

interaction tool, ready to help in such different ambits as neat PC commandment, gaming or design.

The most notable features[2] of the sensor are:

- Precision: It is tremendously precise. It can keep track of all 10 fingers of both hands with a precision of $1/100^{th}$ of mm. It is definitely much more accurate than other existing motion control technologies.
- Workspace: The spherical conic space above the sensor extends to 150° and 650 mm(3) in the vertical z-axis. It gives a huge 3D space where hands and movements are detected. The total volume of this enabled workspace is 226.5 dm^3.
- Real-time interaction: The data analyzed and sent at a ratio of 200 frames per second, what gives a natural sense of instantaneous response.

It is remarkable that SDK packs exist, letting the user perform, adapt and use all the potential the tool has, taking profit of the allowed access to the primary code behind the commercially developed applications.

The physical connection between the sensor and the computer takes place via USB 2.0 (microUSB 3.0 connectors) cable.

The comparison between the Leap Motion Sensor and the popular KINECT cameras becomes inevitable. The main advantage of the first one lays on its precision, much higher than in the KINECT. On the other hand, the KINECT has a way bigger sensed workspace, including the whole human body and its near surroundings while the Leap Motion Sensor has to limit its sensing to the user's hands space.

One could agree that combining both components their full potential can arise. In a first control stage, making a robot walk long distances or move in a big scale scenario, the KINECT cameras could be used. When a more precise control was required, with the robot screwing a lightbulb or inserting a key into a bolt, then it would be time for the Leap Motion Sensor to take the leading role.

In spite of that, this thesis has studied the Leap Motion Sensor case only.

[2] The following specifications are public and official and have been taken from [15].

[3] This maximum z-axis value is the only exception to note 2. It has been measured by own means and it has to be taken as an estimation value only.

Robot

The robot that has been controlled in this thesis is an ABB IRB 140. It consists on a robotic arm with 6 degrees of freedom. It can handle a payload of 6 kg and reach up to 810 mm (to axis 5). The robot itself weights 98 kg and its TCP can achieve a maximum speed of 2.5 m/s and a maximum acceleration of 200 m/s^2, as can be consulted in the official catalogue [11].

The usual applications for which the robot is conceived are arc welding, assembly, cleaning/spraying, machine tending, material handling, packing and deburring. Although in this thesis the robot has not been intended to do any of those concretely, the possibilities the touch-free control offers can be used in some of them.

Robot space definition

The coordinate frames along the robot are established according to the Denavit-Hatemberg convention. It is a methodology that makes it possible to describe the relations between translations and rotations along subsequent elements in an articulated chain, in the present case the robot arm.

To achieve this, a Cartesian orthonormal frame (X_i, Y_i, Z_i) is set for each element in the chain, being $i = 1, 2, \dots n$ the joint number and n the number of DOF of the robot. See the methodology in Algorithm 1, based on [12].

Algorithm 1. Denavit-Hatemberg frame construction

1: Set the base coordinate frame (X_0, Y_0, Z_0) with Z_0 aligned with the articulation 1 axis and pointing out of the floor. X_0 and Y_0 can be chosen conveniently.
2: **for** $i = 0$ **until** $i = n - 1$ **do**
3: Establish the articulation axes. Align Z_i with the motion axis of joint $i + 1$, pointing toward next joints.
4: Establish the frame origin. It lays in the intersection of Z_i and Z_{i+1} or in the intersection of their common normal and Z_i.
5: Establish $X_i = (Z_{i-1} \times Z_i)/\|(Z_{i-1} \times Z_i)\|$. If both z-axis are parallels, then along their common normal.
6: Establish Y_i in order to complete the right-handed coordinate frame.
7: **end for**
8: Establish the flange frame. Set Z_n along Z_{n-1} axis direction and pointing out of the robot. Set X_n normal to both and Y_n as it is required in order to complete a right-handed coordinate frame.

The Denavit-Hartenberg convention defines a set of parameters used to illustrate the transformation from a frame at a joint to the next one [13]. Those are:

- **Link length (a_i)**: distance between the z-axis in i and the previous one.
- **Link twist (α_i)**: angular difference between z-axes in i and $i-1$ around their common normal.
- **Link offset (d_i)**: distance along Z_{i-1} from the origin of frame $i-1$ until the common normal with i.
- **Joint angle (θ_i)**: angular difference between x-axes in i and $i-1$ around the z-axis of frame $i-1$.

A clarifying summary of what has been explained until this point can be seen in Figure 1.2.

This set of parameters, properly defined for each transformation step from $i-1$ to i, can be expressed as a transformation matrix, resulting of two translations and two rotations. It is

$$\mathbf{A}_i^{i-1} = \mathbf{R}_{z_{i-1},\theta_i} \cdot \mathbf{T}_{z_{i-1},d_i} \cdot \mathbf{T}_{x_i,a_i} \cdot \mathbf{R}_{x_i,\alpha_i}$$
$$= \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)\cdot cos(\alpha_i) & sin(\theta_i)\cdot sin(\alpha_i) & a_i \cdot cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)\cdot cos(\alpha_i) & -cos(\theta_i)\cdot sin(\alpha_i) & a_i \cdot sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.1)$$

where $\mathbf{X}_{a,b}$ means a rotation (if **R**) or a translation (if **T**) around or along the a-axis of an angle a or a length b.

The global transformation matrix for a robot can be obtained multiplying these A-matrices from the base to the flange, considering the order. Forward kinematics is the transformation applied to calculate TCP position and orientation in a Cartesian workspace from a set of joint angular positions, usually expressed as q_i. It has always a unique solution: given a joint configuration there is one and only one position and orientation for robot's TCP.

The inverse calculation can theoretically also be performed. It is called inverse kinematics and intends to obtain a set of joint angular positions given a specific position and orientation of the robot's flange. It can be much harder to solve and it can have no analytical solution or this one can be not unique: for a certain position and orientation of robot's TCP, a suitable joint configuration to achieve it may not exist. Moreover, if there are several solutions, possible jumps

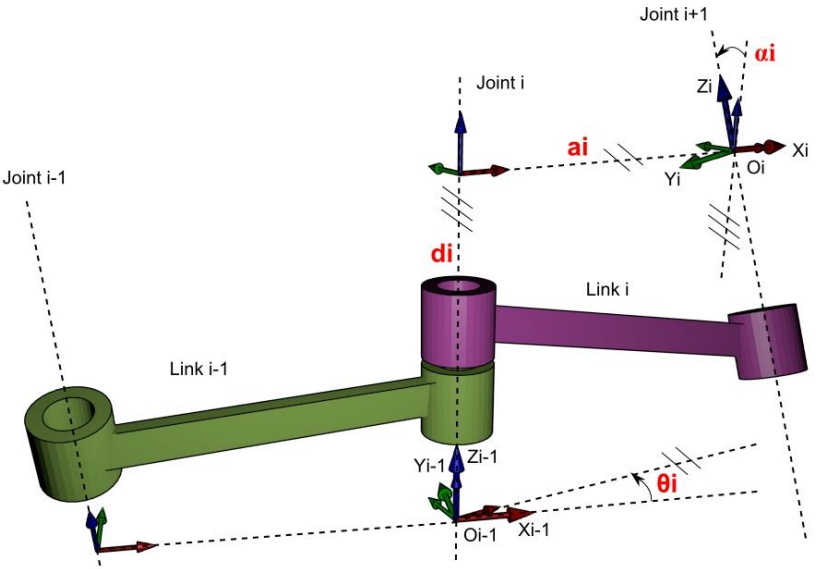

Figure 1.2. Denavit-Hartenberg visual example

in reference signals may appear [13]. For a 6 DOF robot, however, it can be demonstrated that has always at least a solution (inside the workspace) [4].

Regarding the velocities of the TCP, both linear and angular ($\boldsymbol{v}, \boldsymbol{\omega}$), the same transformations can be applied, relating them with the rotational velocities of each joint $\dot{\mathbf{q}}$ in what is known as velocity kinematics. The so called Jacobian **J** is a commonly used matrix to perform this transformation. This matrix has 6 rows and n columns (n being the number of DOF of the robot). It depends on the joint configuration **q** at each instant of time, so one can say that

$$\begin{bmatrix} \boldsymbol{v}(t) \\ \boldsymbol{\omega}(t) \end{bmatrix} = \mathbf{J}\big(\mathbf{q}(\mathrm{t})\big) \cdot \dot{\mathbf{q}}(\mathrm{t}) \tag{1.2}$$

is true [13]. In the same way as before, contemplating the problem in the other way around can be interesting if the desired flange velocity is known and one wants to calculate the required joint actions which cause it. In this case,

$$\dot{\mathbf{q}}(\mathbf{t}) = \mathbf{J}\big(\mathbf{q}(\mathbf{t})\big)^{-1} \cdot \begin{bmatrix} \boldsymbol{v}(\boldsymbol{t}) \\ \boldsymbol{\omega}(\boldsymbol{t}) \end{bmatrix} \tag{1.3}$$

but one can see that if the inversion of the matrix is not possible, the inverse velocity kinematics study will not be possible.

Studying the Jacobian it can be seen that singularities may appear for certain configurations of the joints. Specifically, they will at any time in which **J** has no

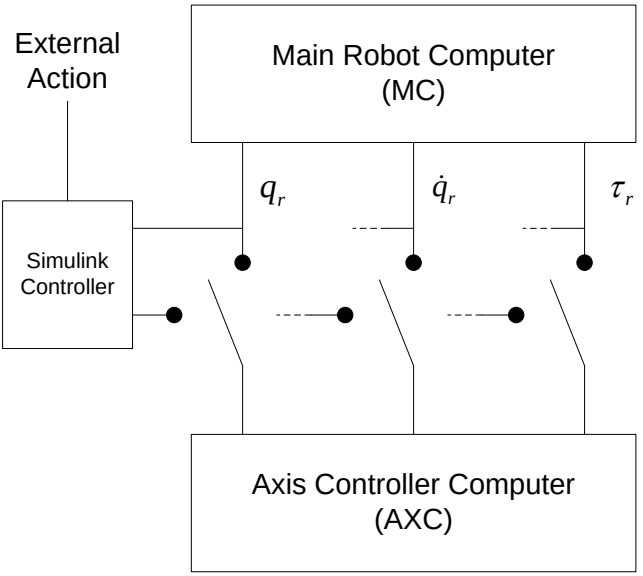

Figure 1.3. External Action Interception diagram

maximum rank; this means at any time when two axes are aligned or whenever the flange reaches an edge of the workspace. Singularities require special care because they imply that certain directions of movement are not possible and that bounded TCP velocities may cause unbounded joint velocities [13].

ExtCtrl

The robot controller is composed by two internal computers: the Main Robot Computer (MC) and the Axis Controller Computer (AXC). The first one is in charge of the high level control of the robot (e.g., path planning and feed-forward control), while the second one is just intended to perform the low level control, acting on each joint.

ExtCtrl is a protocol conceived to intercept the internal data in the robot controller, actuating thus in its behavior from an external computer. When the MC sends data to the AXC the interception of this communication takes place and allows external control, q.v. Figure 1.3.

This middle step actuation is defined through a Simulink model compiled by Real-Time-Workshop [4]. On the one side, it gets references sent by the MC, sensory data concerning the current state of the robot and, if required, LabComm predefined outgoing variables. On the other side, it delivers expected signals such

as torque, velocity and position for the robot (among others) and, if required, LabComm predefined incoming variables.

Main Computer

The essential communication between the Leap Motion Sensor and the robot to be controlled requires by all means a main computer as an intermediate stage, which is going to be in charge of understanding the Leap Motion Sensor data, computing it according to the desired control specifications and sending the appropriate signals to the robot, in order to make it behave as required. Moreover, the computer will also be responsible of managing times and ensuring a free-of-conflicts access to the exchanged data by all parts.

In this present work this main computer role has been assumed by a laptop Acer Aspire E1-571 running under Windows 8. Visual Studio[4] has been used as code editor and compiler platform and the programming language used has been C#. The nature of the created software architecture and its characteristics will be accurately explained in Chapter 2.

The understanding of the Leap Motion Sensor data can be made through its own SDK (software development kit) that, as has been told, can be easily downloaded[5]. This software transforms the Leap Motion Sensor signals into intuitive notions such as fingers, hands, positions of those or gestures, facilitating hugely the programmer work.

In order to facilitate this interpretation of user commands through the Leap Motion Sensor, part of the work of ErghisTech, a small Swedish enterprise interested in free-touch ways of communication between humans and machines, was used.

Thence, given the collaboration with ErghisTech, the software used was actually provided by this enterprise, consisting in a more evolved version of the standard SDK pack including very interesting features such as the air keyboard developed by them and their concept of a virtual sphere to be hold (see Figure 1.4).

[4] Microsoft Visual Studio Express 2012 for Windows Desktop.
[5] It can be downloaded for free at https://developer.leapmotion.com/.

Figure 1.4. Promotional image by ErghisTech where the sphere concept can be seen.

Finally, once sensory data is properly understood, the main computer has to send clear commands to the robot. To achieve so, it is necessary to establish a channel through which both the main computer and the robot can interact. This has been done with a LabComm connection, q.v. next section LabComm Connection.

LabComm Connection

The LabComm Connection protocol is a communication protocol developed by the Department for Automatic Control of the LTH, in Lund. It allows the communication between a computer connected to the local network and the controller of the robot. It gives a transmission rate suitable for a correct control.

LabComm requires explicitly stating the variables that are going to be sent through the network into an .lc file. This file is then compiled using the LabComm compiler and kept on both sides of the connection for them to be able to identify the incoming and outgoing data [14].

2. Methodology

The main goal of this work has been to create a single program for the main computer able to deal with the connection with the robot and the one with the Leap Motion Sensor, acting as a linkage and data manager.

2.1 Connecting with the robot

On the one hand, in order to connect the main computer with the server using C# language the LabComm protocol was used. This apparently simple task became a real endeavor in the present thesis. The first intuition one can have may be to establish the connection between the main computer and the robot directly, but it turns out to be impossible because of the use of C# in the first part, which has no implemented methods to deal with the procedures required in the robot side.

Hence, a third element must be introduced, assuming the client role and acting as a linkage between both entities. Its name, and the one that will be used for it from now on, is Central Station. This is a lab computer that, in Python language, starts two threads, one sending data from the main computer to the robot and the other one in the other way around. Time and master's thesis advance proved this second thread to be unnecessary as long as the returning data is not taken into consideration in the designed program.

To establish such a communication, encoders and decoders were needed. They are part of the LabComm protocol and are used to convert a variable with its values into a signal that can be sent through the network and, once it has arrived to its destination, be converted back in order to understand it.

The last problem with this connection was the existence of two LabComm versions. On the one side, the Central Station and the main computer use a late version of the protocol (2013) while, on the other side, Central Station and the robot controller use the old version (2006). Given therefore the existence of two

different .lc-compiled files, one for each version, a careful use of signatures after coding and decoding became fundamental to achieve a correct communication.

The basic .lc file, called *leap_labcomm.lc* states as follows:

```
sample struct {
double data[7];
int jogbyjoint;
} leap2extctrl;

sample struct {
double data[7];
int jogbyjoint;
} extctrl2leap;
```

The distinction between *leap2extctrl* and *extctrl2leap* is merely a convention to clarify and organize data to keep clear in which direction it flows. As has been said, *extctrl2leap* was not used and has only been maintained in the program for possible future uses.

2.2 Connecting with the Leap Motion Sensor

The connection with the Leap Motion Sensor could be performed with the standard Leap Motion Sensor software, but in this case was made through the software provided by ErghisTech. This slightly differs from the standard one developed by Leap Motion Sensor, having differences such as variable names and function declarations. On the one hand, this makes impossible to consult general Leap Motion Sensor developers' forums which talk in terms of standard methods, but on the other hand, the methodology designed by ErghisTech and their sphere conception can be used.

In general, data acquisition can be made through polling (to ask for data whenever the application is ready to deal with it) or through callbacks (the sensor sends the data as soon as it is updated). In the present work the second choice was used, in concordance with ErghisTech methods. A listener was implemented and the function *OnErghisFrame()* used. This comes to mean that the Leap Motion Sensor will set the data flow ratio and the application will never receive either two samples for the same frame or skip a frame (this can happen in polling).

Callbacks require threads. In this case, given that only one source was reading from this thread, no semaphores were needed. New data was saved into a

local variable in *Main Program* as soon as it was received and then sent to *Data Management* to be interpreted.

2.3 Understanding new data

Once data could be clearly read from the sensor and clearly sent to the robot, control could begin.

First of all it is fundamental to clearly define what is the robot required to do and how is the user going to be able to make the robot do it. When thinking of these desired responses and possible user commands it has to be kept in mind that the control object will be an ABB IRB-140 arm robot and the control tool a Leap Motion Sensor.

Robot responses required

It is intended to take profit of all robot capabilities. At the most basic level, for an arm robot, this means to make the robot:

- Translate its TCP in all 3 directions within its limits.
- Rotate its TCP in all 3 directions within its limits.
- Lock one or some directions to make the robot translate or rotate, for example, along a single relevant axis or a defined plane.
- Implement specular imitation. This is, make the robot able to follow the position and orientation of user's hands in a mirror-like way.
- Use joint-by-joint motion, meaning to translate and rotate the TCP as a result of the motion of a particular robot joint, within its limits.
- Grab and release the robot gripper.
- Change the speed at which the robot makes all the above listed actions.

From a higher level it can be very interesting, and is actually a goal of this thesis, to make the robot learn specific actions, composed by one or several of the above listed. Thus, the ability to record actions, to store them and to let the user navigate through them is also contemplated.

In the same way, it is desired to make the user feel a comfortable and natural control over the robot, so live continuous commands are preferred, rather than predefined numerical definitions or discrete modes.

To achieve such a wide range of objectives it is required to create a menu-like navigation system. The robot has to know where it is and act properly. This

comes to mean that the same user action will not cause always the same robot response.

Some of the menu levels have to be:

- Main Menu: The first stage the user fronts. It lets him choose what he wants to do.
- Motion Modes: Different options for the user to control the robot.
- Teaching mode: The user can start recording an action or play, rename or delete an already recorded action.

User control possibilities

The Leap Motion Sensor, with the SDK pack, can obtain a lot of information about the hands it has over it. It can get:

- The number of hands and fingers exposed (maximum 2 and 10, respectively) and their identification.
- The position and rotation of a given hand (in all 3 axes), within its volumetric range.
- Recognition of several gestures, like a hand folding into a fist or finger-tapping.

An interesting gesture that was not implemented was the clap. It was created by own means considering the two most representative characteristics of a clap: the hands contact at the end of the gesture and they have a certain velocity in their way there. Thus, the method *isAClap()* checks at any time the variation on the hand-to-hand distance. When this becomes bigger than a certain speed boundary a second clause is triggered, beginning to check if this distance gets closer than a certain distance boundary during the next half second. If it gets, the method return true (a clap has occurred). If it does not, the program gets back to its original speed monitoring state. This conception prevents fast hand movements, where hands are far from each other, or slow hand approximations to be mistaken as claps. Between two consecutive claps a certain time is required to avoid multiple positives because of a single clap.

Control guide

The control guide is the definitive matching solution adopted to solve the control problem here studied. It relates each robot response requirement, previously stated, with a certain user' commands combination to achieve it.

Several options were studied and contemplated, because there are lots of possible solutions to the matching problem. The final a decision was made taking into account aspects as compactness and coherence of the solution, easiness to understand and follow by the user and easiness to be implemented.

When possible, the program intends to let the user decide. He can choose between different control ways for him to feel content. For example, several ways of controlling the speed are thought to be appealing. In this case, the author's intention is not to decide one of these options and discard the others but to let the user choose according to his preferences and working circumstances.

Converting data

Once the data captured by the Leap Motion Sensor is understood following the control guide it has to be transformed into something that can be sent to the robot in order to make it behave as expected.

The Simulink controller has room for user-defined variables, specified in the .lc file. In this case those variables are a vector of 7 components (6 DOF + gripper) and an indicator of whether this vector refers to joint values or to absolute linear speed values. In the second case, it is also needed to specify if the speed comes from Linear Commandment or from Specular Imitation, because in the second one, filtering will be required. The software that has been developed in this thesis accomplishes this point, storing data in the variables called *data* and *jogbyjoint*. Both are the variables to be encoded and sent.

2.4 Safety considerations

The Leap Motion Sensor is an accurate device, but in spite of that, the flow of information from the user hands to the sensor is not free of errors and it can mistake values sometimes, therefore completely wrong instructions may be sent. It is supposed to detect hands, but it is thought to be shown only hands also. If a distracted user, for example, exhibits his chin to the sensor it will take some value for it, as if it was a finger, so undesired control action may occur.

Moreover, is a basic principle of this thesis to consider always the worst case, so even if a nearly perfect sensor performance could be assured, safety considerations would still remain stated as fundamental, ready to deal with unexpected distortions.

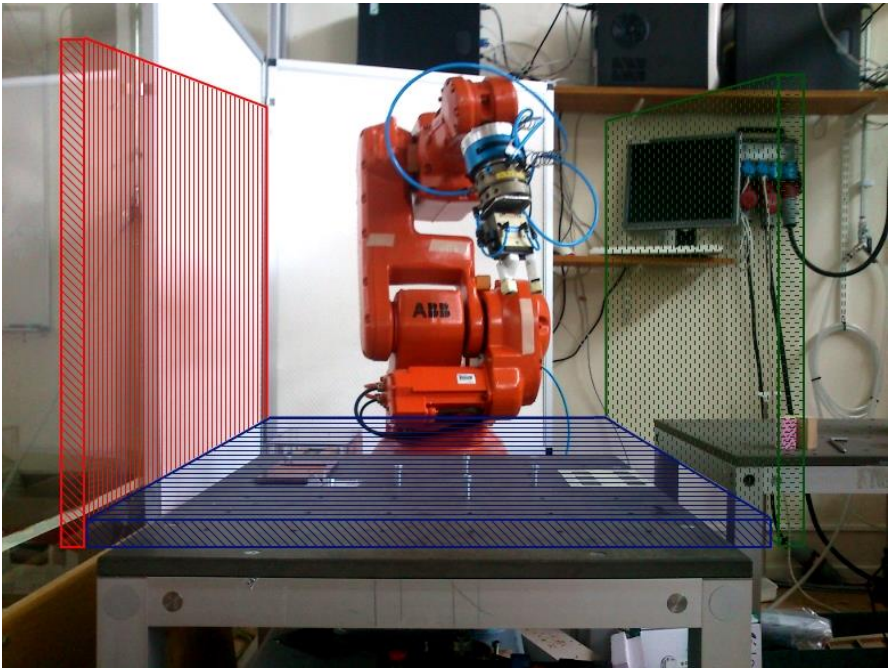

Figure 2.1. Workspace virtual walls.

Here safety considerations refer to the prevention of hits between the robot and the user, the robot and its surroundings and the robot with itself, which may result in human, robotic or material damages.

Positional saturation

The first kind of hits may be easily avoided keeping the user executing the control out of the reach of the robot, given that no direct interaction (contacts) are required in this work.

The second kind of collisions is very important, because it has to be noted that the current emplacement of the robot in the laboratory leaves objects inside its area of influence, hence a bad control (due to user distraction or sensor misunderstanding) can easily result in a clash. It is overcome by defining invisible walls the robot can by no means breach. The location of these walls in the working environment of the robot can be seen in Figure 2.1. The green wall was created with testing purposes only.

What the wall actually does is to saturate the robot's linear speed to zero in the outgoing direction in function of its TCP position. Moreover, a linear

interpolation of this saturation has been performed to avoid abrupt clashes with the invisible walls, converting them into soft walls. Once in the wall, the TCP can only be moved back to the workspace or along the wall.

Note that it would be extremely easy to define alternative walls in another working environment.

Regarding "autocollisions" or the ones between the robot and itself, once they are produced it is important to stop the movement there; otherwise motor damages can appear. One can distinguish two degrees on them:

- 1st degree autocollisions: Those produced between two consecutive links. They can occur when a joint angle gets too small and the rigid parts at each side collide. To deal with them, joint speed saturation has been performed in function of the angle at each joint. If a user forces a joint to its limit, the controller will ignore further changes and allow only releasing joint velocities.
- 2nd degree autocollisions: Those produced between two non-consecutive links. Much harder to prevent, and also much more uncommon, have been left unrestrained given the complexity of their management. Users must be asked to avoid them.

Velocity saturation

Although there is no planned way to give the robot an uncontrolled velocity, their final control is still a must. The incoming velocity always passes through a saturation block in the controller where it is assured to be within safe boundaries.

In addition, the ultimate joint velocity is checked to be within boundaries again, preventing cases such as near-singularity movements (q.v. Singularities management, below).

Singularities management

As exposed in the introductory part of this paper (q.v. Robot space definition), given some joint configurations singularities may appear. So far, this thesis has protected the workspace environment edges, but another cause of these singularities may lay in the alignment of links.

In such a configuration, the inversion of the Jacobian tends to infinite, resulting in huge velocities. This would not be problematic given that final joint speed is properly saturated, but if a joint reaches the total alignment can become stuck and be unable to leave that state unless giving a direct joint action.

To avoid such a case, a baptized *Singularity Crosser* has been performed. It is intended to act when the user is in either Specular Imitation or Linear

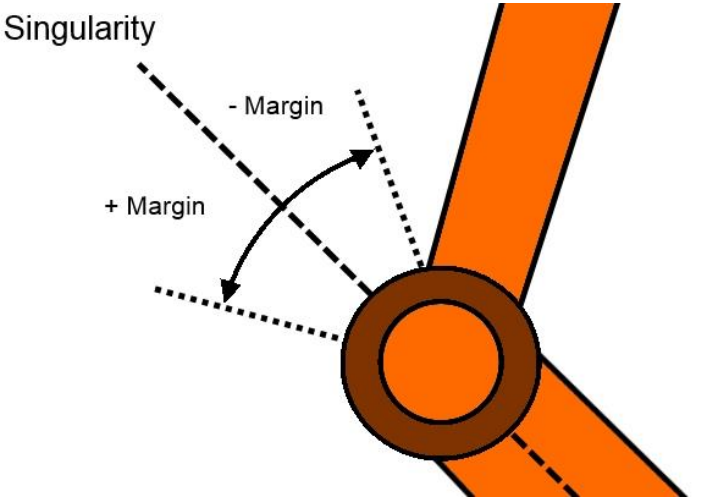

Figure 2.2. Singularity by alignment of axes contemplation.

Commandment modes and the robot is going to cross a singular links-aligned position.

The crosser keeps control of the angular position of joints 3 and 5, the ones that may cause link alignments. When the angle lies within predefined small margins covering the singular position, it takes control of the speed of the joint and forces it to be high enough to ensure it exits the singular zone in the next sample of time. This speed value is none other than two times the half-margin (see Figure 2.2). Thus it is impossible for a joint to rest at a singular position.

Hands surveillance

Two main problems may be identified when thinking about following the track of the user's hands the whole time:

Entering and leaving the workspace without disturbances:

A problem arises when thinking of the needed exposure and retreat of the hands above the Leap Motion Sensor and how to make the sensor distinguish between a voluntary command and an inevitable motion for the user to place his hands over it or to retreat them.

To prevent such confusion, a certain gesture is decided to be used to define the start and end of the motion time. This gesture is thought to be a clap. Moreover, this clap can additionally be used to define a reference system adjusted to the user's most comfortable starting position.

Hence, a clap will first scan the user's hands' position and set is as the reference origin point and next trigger the sending of non-zero data to the robot, making it move. A second clap will finish this transit, stopping any motions and allowing the user's hands to retreat in a safe way.

Confronting interruptions:

Another problem may appear when realizing that at some points during the control action, the Leap Motion Sensor suddenly loses track of the user's hands, thus sending suddenly a zero value for all moves. This sort of error can be caused most likely by inappropriate room lighting or simply because the user left the workspace without noticing.

The reacquisition of hand's track has proved to cause speed peaks that by all means must be avoided.

To solve this, a *Softener* is implemented. Its mission consists in linearly modulate the intensity of all sent signals between zero and its true value during the first two seconds after any loss of track. Thus, peaks are avoided.

If the loss lasts more than three seconds, the signal is considered lost and a new clap is required to restart the movement.

3. Results

The final results of this thesis are presented below.

3.1 Computer program

The previously presented possibilities-requirements problem, introduced in section 2.3 , is finally solved with the hereunder presented control guide and an interface thought for the users to easily understand where they are and what can they do in each mode. The hierarchical structure of the program can be seen in Figure 3.1.

To navigate throughout the different levels of the menu the user has to tap

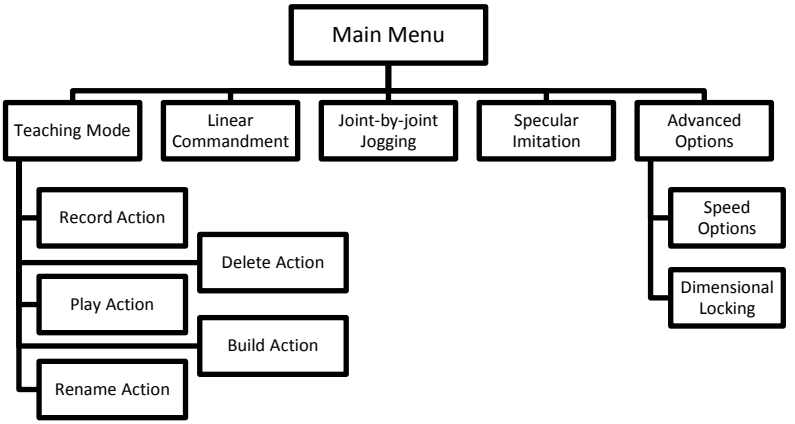

Figure 3.1. Hierarchical organization of modes scheme

using his fingers onto the virtual sphere. At each mode, he will see in the screen all the modes he can access from that point and which finger triggers which mode. When leaving the main menu stage, a back button will appear for him to level up to the previous step if he wants to deepen into another branch instead.

Once being said that Main Menu is only a gate to access the different modes available, the rest of the modes Figure 3.1 shows will be explained below this.

Teaching Mode

This mode aims to be a tool for making the robot to learn. Using action segmentation concepts, the robot is meant to be able to store sub-actions taught by the user one by one and to let this user build new, maybe more complex, actions using these small ones.

With further studies, and taking full profit of the Autonomous Segmentation theory, the robot would even be able to distinguish these sub-actions from a big taught action by itself. Moreover, it would be very interesting to let the robot try to compose new actions under its own judgment, using for example some policy-reward algorithms to make it reach a specific objective.

So far, the Teaching Mode has only available the following commands:

- Record Action: The program is ready to start recording the next motion performed by the user, being the following two claps the start and end of this recording. The new action is going to be saved in a text file with the present date and time as filename.
- Play Action: The robot reads a previously recorded action from a stored text file, repeating identically the action the user performed before when recording.
- Rename Action: The user is able to change the default name of a file (present time) for a more representative one of what the action does actually do, e.g., “Waving”.
- Delete Action: The user can remove a previously recorded action from the actions folder if he wants to discard it.
- Build Action: The user can build a new action using previous actions as pieces. He can create a sequence of sub-actions combining them in whichever order, repeating several times a certain one and adding rest periods in between. Note that this functionality was not completely implemented in the final version given time constraints.

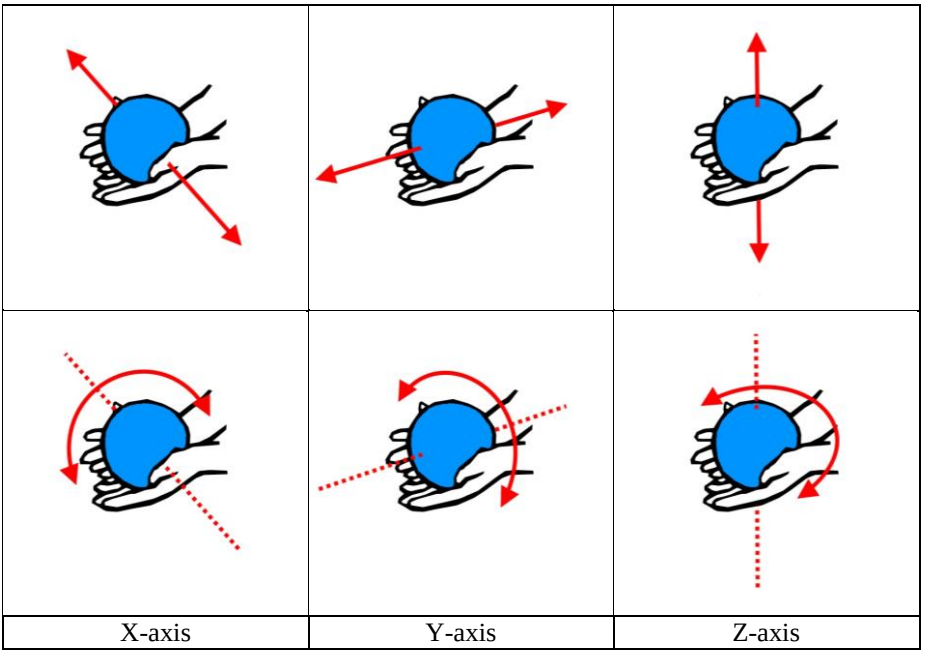

Figure 3.2. Linear Commandment mode user instructions for translating and rotating along and around all three axes; with axis definitions.

Motion Modes

The so called Motion Modes are those where the user does actually move the robot in a direct way. They are named Linear Commandment, Joint-by-joint Jogging and Specular Imitation. All this modes require a clap to start sending data and habilitate the motion of the robot.

Linear Commandment

In order to rotate and translate linearly the TCP of the robot, the user has to rotate and translate his hands over the Leap Sensor Motion.

Given a reference origin point, a centered natural position for the user over the sensor that is defined after clapping, when the user's sphere center reaches a certain distance from this origin, along one or several axes, the program will understand "translation" in those several axes in the specific direction the user has moved his hands towards.

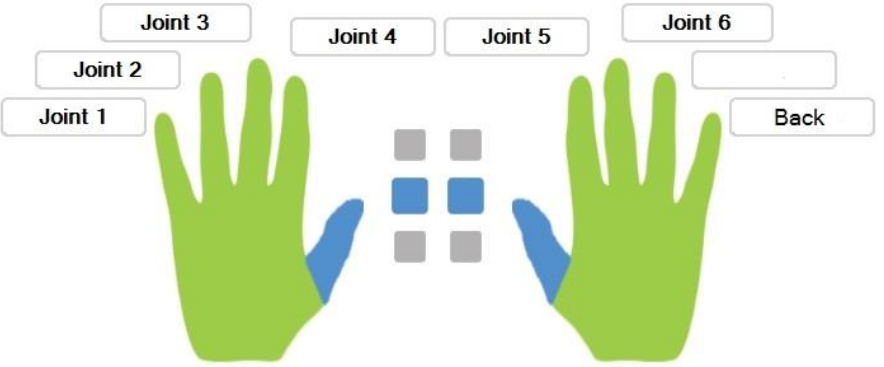

Figure 3.3. Joint-by-joint mode menu

When the user's hands reach a certain inclination respect the original position, around one or several axis, the program will understand "rotation" around those axes in the specific direction the user has rotated his hands around.

The velocity of these movements will not depend on the distance from the origin to the current position but from the predefined speed and the size of the virtual sphere at every instant of time (this is further explained in Speed options, later in this chapter). In Figure 3.2 the motion along the 6 DOF of space can be seen. The blue sphere represents the virtual sphere the user is holding the entire time.

Joint-by-joint Jogging

This mode is used to move a specific joint individually Given that there are 6 joints and the user has 8 tapping fingers, an ordered numbering is made hence the user can select which joint he wants to move by tapping the chosen joint. The last finger exits this mode. A visual example can be found in Figure 3.3.

Once in Joint-by-joint Jogging, the tilting of hands up and down (around the x-axis, q.v. Figure 3.2) will make the robot move around that joint in one or other direction.

Specular Imitation

In this mode the robot will follow the user's hands' position in a mirror-like way. It is possible to choose among three levels of scaling, depending on if it is desired to perform a more or less precise move.

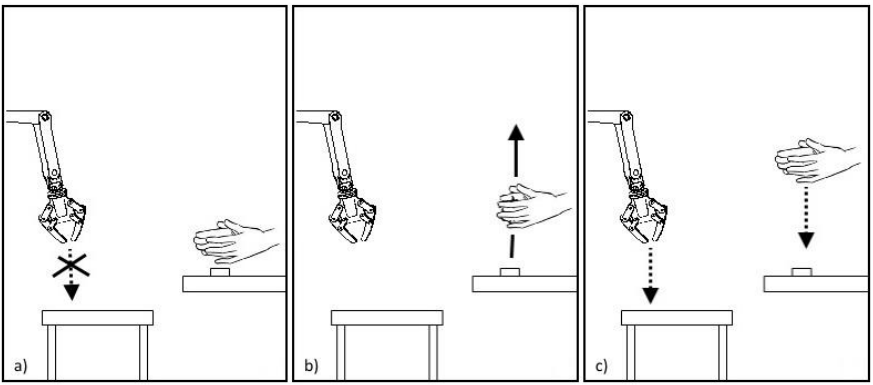

Figure 3.4. Scrolling example: a) the user would like the robot to move lower, but he has no room above the sensor. b) he turns his right hand into a fist and moves up, the robot ignores him and stays still. c) the user unfolds his right hand and gains control of the robot again, now having room to move it lower.

Specular Imitation works as follows: at each frame, the Leap Motion Sensor captures the current position (center of the sphere's position) but stores the last known position, so the difference can be calculated and the velocity obtained. Sending this velocity rather than the position to the controller avoids the calculation of inverse kinematics (required for saturations) and simplifies the Simulink model (no new branches have to be created). The only noticeable thing is that a low-pass filter will be required to soften the robot's motion; otherwise it results noisy and brusque.

In order to give more – actually unlimited – space for the user to perform movements, the concept of *scrolling* appears. In this context, to scroll means to change the reference system from one point to another within the workspace without moving the robot. If the user wants to scroll he has to close his right hand into a fist, move or rotate to the new position, open it back resetting the reference and continue with the motion. The concept of scrolling can be easily understood looking at the example in Figure 3.4.

Advanced Options

This static mode allows the user to define how he wishes to perform the control of the robot. Speed options and dimensional locking can be adjusted, always by direct tapping.

Speed options

When shifting the left thumb, the user can turn the Sphere Modulation on or off, decide the basis speed and decide the basis scaling.

The Sphere Modulation is related to the size of the virtual sphere the user is holding at all times. It is used to modulate the speed at which the robot moves in real-time: for small precise and slow movements, both hands have to be close from each other, holding a small sphere, while for big fast movements both hands must be separated, holding a bigger sphere. This is an analog process with a continuous modulation, thought for the user to experience a real control sensation in a very intuitive way.

While running under Specular Imitation mode, the Sphere Modulation will not change the speed but the working scale, what will result also in a very intuitive control feeling.

By default this option is on, but if the user prefers to deactivate it, he can easily do it tapping onto the switch in this Advance Options mode.

The basis speed is used to predefine what kind of motion the user intends to perform. He can choose among Slow, Medium or Fast speeds.

In a similar way, the user can decide the scaling factor used in Specular Imitation, among Precise, Normal or Big scales.

The effect of both Sphere Modulation and predefined speed and scaling factors is combined into the final resulting speed as a multiplication of factors.

Dimensional locking

The user has full control over the dimensions in which the motion is allowed. He can explicitly impede translations or rotations along or around certain axes if he wants, for example, to work over a plane or a straight line.

Moreover, he is supposed to be able to restrict motions to a single dimension or a plane online (while performing in a Motion Mode), by shifting his right thumb or both right and left, respectively. This was conceived to give fast access to usual restrictions, improving the control experience.

However, these modes have proved to fail a bit in some lighting conditions, because of sensor misunderstandings. In any case, the user has freedom to turn these functionalities off if he wants.

Gripper control

The gripper can be controlled while the user is in a Motion Mode turning the left hand into a fist. Then, tilting the right open hand up and down will make the gripper respectively grab and release.

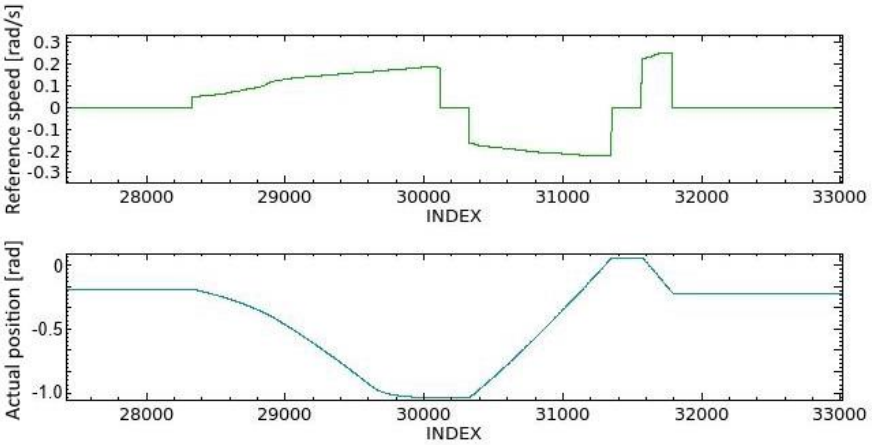

Figure 3.5. Reference speed vs. actual position for Joint 1

3.2 Robot response analysis

Although the robot followed well the user's commands during trials at lab, a finer analysis was made to clearly see until what point it was following the reference properly.

Joint-by-joint mode was used and reference speed and actual position were plotted (see Figure 3.5).

It can be seen that no significant response delays appear and that saturation due to invisible walls work well (position is attenuated around -1 radians even though reference speed was increasing).

It is also apparent that there is a mismatch in rotation directions between reference and response (a positive speed makes the robot move towards a negative direction). However, it has no effect in the user's perception of motion and is left as it is.

4. Discussion

The achieved results are satisfactory and positively accomplish this thesis' objectives, but it must be said that the full potential of this control and learning interface has not been used.

Due to time constraints, final tests to perfect the control and the interface (e.g., reduce sensor mistaking signals and deal with them when they appear, fully implement the Build Action method or improve final screen interface shown to the user) were not performed.

Nevertheless, in general lines, the program works well and accomplishes the basic objectives that were thought for the control experience. Hereunder, all program modes performances will be discussed.

Teaching Mode:

So far the program records actions by storing all sent data into a text file and it then plays the action by reading from the file instead of listening to the sensor. It would be interesting to think of a different way of storing that could be meaningful, like calculating the described trajectory and the velocity at each instant of time.

Some problems related to the dynamical declaration of commands in the ErghisTech definition software impeded to satisfactory choose which action is to be played, renamed or deleted. The correction of this error should not be much complicated.

Finally, to create the Build Action method would have been very interesting and, furthermore, to implement new concepts for the robot to try Autonomous Segmentation over a recorded action would be a must in future research on this topic.

Motion Modes:

Motion Modes have proved to be useful and the wide range of adjustments allowed regarding speed or scaling makes them easily adaptable to several kinds of situations and working conditions.

Improving the sensor detection capability to avoid the occasional loss of track of the hands or at least soften this empty frames' effect could be a line of study. Moreover, it could be room for improvement in filtering the noise of the robot response in Specular Imitation mode.

Advanced Options:

The program allows the user to configure settings according to his preferences in order to obtain a tailored control system. Moreover, the implementation of new features would be relatively easy and the program could be adapted following user's demands.

Another point of discussion is the balance between the control made in the program, allocated in the Main Computer, and the control performed in the Simulink Controller, allocated in the Central Station. The final result uses both resources to take care of the several aspects the control cares about, that have been detailed in the previous chapters.

However, it is thought that in order to facilitate the implantation of this software into another robots rather than the one in the Robotics Lab that have been used so far, it would be interesting to concentrate all the control efforts in one side only, preferably in the Main Computer, leaving for the Simulink Controller only the task to deliver received signals to the robot.

So far, the Simulink Controller is additionally responsible of computing forward kinematics and Jacobian matrices in order to perform safety saturations as explained in section 2.4 , as well as filtering the Specular Imitation signal to reduce noise and performing the integration from the reference velocity to the reference position. This tasks could be integrated into the Visual Studio solution in the Main Computer, resulting into a more compact and portable software.

5. Conclusions

Once the work is finished and time gone, it is interesting to have a look at the initial objectives and see what has been accomplished and what not.

- A working network was satisfactorily created in the Lab, linking for the first time a Windows 8 laptop with C# language code with the robots through the LabComm protocol.
- The Leap Motion Sensor has proved to be a useful device very capable of allowing human-robot communication in a natural way.
- A control guide was developed, proving to be useful when making the robot perform its most basic actions such as translate, rotate, jog by joints and follow a positional reference.
- Basis for task teaching have been settled. Although being true the final result is quite far from the initial expectations about making the robot learn, a clear path can be seen, at least, until the robot creating new action by composing stored sub-actions.
- Other future research lines and improvements for the interface to become a potentially useful tool for users to control robots in nowadays market can be descried; those that have been discussed in the previous section.

6. Appendices

6.1 Computer code

In this section, the whole code that lies beyond the final program is presented. It is divided into the natural different files it is composed by.

Main Computer (MyProject)

The entire code is written in C#.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel;
using System.Threading;
using System.Windows.Forms;
using Erghis;

namespace ErghisExternal
{
    public class MainProgram
    {
        public static double[] outgoingData = new double[7];
        private static Thread leapThread;

        //Internal Data parameters
        public static Data data = new Data(); //last frame of leapData
        public static Data olddata = new Data(); //previous frame of leapData

        public static bool firsttime = true;
        public static DateTime startingTime = DateTime.Now;
        public static bool justLost = false;
        public static DateTime exitTime = DateTime.Now;

        public static void Main()
        {
            //Present the program
```

```
            Console.WriteLine();
            Console.WriteLine("Leap Motion Sensor Robotic Control. v1.0");
            Console.WriteLine("=======================================");
            Console.WriteLine(DateTime.Now);
            Console.WriteLine();

            ErghisExternal.Program.Setup();

            //Starts getting data
            //in a new thread, to ensure that gui is not locked
            leapThread = new Thread(run);
            leapThread.Start();

            //Establish a connection:
            ExtCtrl.LabCommManagerServer ls = new ExtCtrl.LabCommManagerServer();

            //Holds the program
            Console.ReadLine();
        }

        //The Erghis launchment
        private static void run()
        {
        Application.Run(new Erghis.Form1(ErghisExternal.Program.EC));
        }
    }
}
```

Code 1. MainProgram.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Threading.Tasks;
using System.Windows.Forms;
using Erghis;

namespace ErghisExternal
{
    public class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        ///
        public static ErghisController EC;
        public static ExternalListener listener;
        public static Commands commands;

        [STAThread]
        public static void Setup()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
```

```
            //Create a new ErghisController
            EC = new ErghisController();

            //Create an instance of your listener
            listener = new ExternalListener((ErghisOrbStyle)EC.Settings.GetStyle(), EC);
            EC.SetListener(listener);

            //If you want to define custom commands for the keymaps, do so here

            //Create a new class to hold the code
            commands = new Commands(EC);

            //Register a new command in the keymap
            //Parameters:
            //Name as it appears in the keymap
            //Delegate of the method to be run on that tap
            //How the command will appear in the GUI
            //Default keymap:
            EC.RegisterCommand("[KEYMAP]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("mainmenu")), "Robot");

            //Main menu keymap:
            EC.RegisterCommand("[Teach Mode]", new
Keymap.VoidDelegate(()=>commands.SwitchKeymap("teachingmode")), "Teaching Mode");
            EC.RegisterCommand("[Jog by joints]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("jogbyjoint")), "Jog by Joints");
            EC.RegisterCommand("[Jog linearly]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("freejogging")), "Linear Commandment");
            EC.RegisterCommand("[Options]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("options")), "Advanced Options");
            EC.RegisterCommand("[Tracking]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("tracking")), "Specular Imitation");

            EC.RegisterCommand("[Back to menu]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("mainmenu")), "Back to Menu");

            //Jogbyjoint keymap:
            EC.RegisterCommand("[Joint 1]", new Keymap.VoidDelegate(() =>
commands.SwitchJoint(1)), "Joint 1");
            EC.RegisterCommand("[Joint 2]", new Keymap.VoidDelegate(() =>
commands.SwitchJoint(2)), "Joint 2");
            EC.RegisterCommand("[Joint 3]", new Keymap.VoidDelegate(() =>
commands.SwitchJoint(3)), "Joint 3");
            EC.RegisterCommand("[Joint 4]", new Keymap.VoidDelegate(() =>
commands.SwitchJoint(4)), "Joint 4");
            EC.RegisterCommand("[Joint 5]", new Keymap.VoidDelegate(() =>
commands.SwitchJoint(5)), "Joint 5");
            EC.RegisterCommand("[Joint 6]", new Keymap.VoidDelegate(() =>
commands.SwitchJoint(6)), "Joint 6");

            //Options Keymap
            EC.RegisterCommand("[Allow speed modulation ON/OFF]", new
Keymap.VoidDelegate(() => commands.SpeedModulation()), "Hands Modulation");
            EC.RegisterCommand("[Slow]", new Keymap.VoidDelegate(() =>
```

```
commands.ChangeSpeed("Slow")), "Slow");
            EC.RegisterCommand("[Medium]", new Keymap.VoidDelegate(() =>
commands.ChangeSpeed("Medium")), "Medium");
            EC.RegisterCommand("[Fast]", new Keymap.VoidDelegate(() =>
commands.ChangeSpeed("Fast")), "Fast");

            EC.RegisterCommand("[Precise]", new Keymap.VoidDelegate(() =>
commands.ChangeScaling("Precise")), "Precise");
            EC.RegisterCommand("[Normal]", new Keymap.VoidDelegate(() =>
commands.ChangeScaling("Normal")), "Normal");
            EC.RegisterCommand("[Big]", new Keymap.VoidDelegate(() =>
commands.ChangeScaling("Big")), "Big");

            EC.RegisterCommand("[Allow single dimension lock mode ON/OFF]", new
Keymap.VoidDelegate(() => commands.SingleDimLock()), "Single Dimension Lock");
            EC.RegisterCommand("[Allow plane control mode ON/OFF]", new
Keymap.VoidDelegate(() => commands.PlaneControl()), "Plane Control");
            EC.RegisterCommand("[Rot X]", new Keymap.VoidDelegate(() =>
commands.NeglectDim(3)), "Rot X");
            EC.RegisterCommand("[Rot Y]", new Keymap.VoidDelegate(() =>
commands.NeglectDim(4)), "Rot Y");
            EC.RegisterCommand("[Rot Z]", new Keymap.VoidDelegate(() =>
commands.NeglectDim(5)), "Rot Z");
            EC.RegisterCommand("[Trans X]", new Keymap.VoidDelegate(() =>
commands.NeglectDim(0)), "Trans X");
            EC.RegisterCommand("[Trans Y]", new Keymap.VoidDelegate(() =>
commands.NeglectDim(1)), "Trans Y");
            EC.RegisterCommand("[Trans Z]", new Keymap.VoidDelegate(() =>
commands.NeglectDim(2)), "Trans Z");

            //Teaching keymap:
            EC.RegisterCommand("[Record]", new Keymap.VoidDelegate(() =>
commands.Record()), "Record Action");
            EC.RegisterCommand("[Rename]", new Keymap.VoidDelegate(() =>
commands.doNothing()), "Rename Action");
            EC.RegisterCommand("[Delete]", new Keymap.VoidDelegate(() =>
commands.doNothing()), "Delete Action");
            EC.RegisterCommand("[Play]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("playactions")), "Play Action");

            EC.RegisterCommand("[Back to teach]", new Keymap.VoidDelegate(() =>
commands.SwitchKeymap("teachingmode")), "Back");

            OwnFunctions.updateFiles();

        }
    }
}
```

Code 2. Program.cs

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Erghis;

namespace ErghisExternal
{
    public class Commands
    {
        ErghisController EController;

        public Commands(ErghisController ec)
        {
            EController = ec;
        }


        public void SwitchKeymap(string keymap)
        {
            EController.SwitchKeymap(keymap);
            switch (keymap)
            {
                case "freejogging":
                    Console.WriteLine();
                    Console.WriteLine("Linear Jogging Mode");
                    Console.WriteLine("--------------------");
                    Console.WriteLine("Move your sphere around the origin to transmit
linear velocity and/or rotations to the robot.");

                    break;
                case "jogbyjoint":
                    Console.WriteLine();
                    Console.WriteLine("Jog-by-joint Mode");
                    Console.WriteLine("-----------------");
                    Console.WriteLine("Jog the robot joint by joint tilting up and down.
Set a joint to begin.");
                    break;
                case "options":
                    Console.WriteLine();
                    Console.WriteLine("Advanced Options");
                    Console.WriteLine("----------------");
                    Console.WriteLine("Choose velocity settings for jogging, tracking
scale or dimensional locking options.");
                    break;
                case "tracking":
                    Console.WriteLine();
                    Console.WriteLine("Tracking mode");
                    Console.WriteLine("-------------");
                    Console.WriteLine("Move the sphere along the path you want the robot
to follow.");
                    break;
            }
        }

        public void SwitchJoint(int joint)
```

```
        {
            DataManagement.jogbyjoint = joint;
            Console.WriteLine("Active Joint: {0}", joint);
        }

        public void SpeedModulation()
        {
            if (DataManagement.handmod)
            {
                DataManagement.handmod = false;
                Console.WriteLine("Hands Modulation OFF");
            }
            else
            {
                DataManagement.handmod = true;
                Console.WriteLine("Hands Modulation ON");
            }
        }

        public void ChangeSpeed(string speed)
        {
            switch (speed)
            {
                case "Slow":
                    DataManagement.SF = 0.5;
                    DataManagement.SFJ = 0.5;
                    Console.WriteLine("Speed: Slow");
                    break;
                case "Medium":
                    DataManagement.SF = 1;
                    DataManagement.SFJ = 1;
                    Console.WriteLine("Speed: Medium");
                    break;
                case "Fast":
                    DataManagement.SF = 3;
                    DataManagement.SFJ = 1.5;
                    Console.WriteLine("Speed: Fast");
                    break;
            }
        }

        public void ChangeScaling(string scale)
        {
            switch (scale)
            {
                case "Precise":
                    DataManagement.SFT = 0.5;
                    Console.WriteLine("Tracking scale: Precise");
                    break;
                case "Normal":
                    DataManagement.SFT = 1;
                    Console.WriteLine("Tracking scale: Normal");
                    break;
                case "Big":
                    DataManagement.SFT = 1.3;
                    Console.WriteLine("Tracking scale: Big");
```

```
                break;
            }
        }

        public void SingleDimLock()
        {
            if (DataManagement.singdimlock)
            {
                DataManagement.singdimlock = false;
                Console.WriteLine("Single Dimension Lock OFF");
            }
            else
            {
                DataManagement.singdimlock = true;
                Console.WriteLine("Single Dimension Lock ON");
            }
        }

        public void PlaneControl()
        {
            if (DataManagement.planectrl)
            {
                DataManagement.planectrl = false;
                Console.WriteLine("Plane Control OFF");
            }
            else
            {
                DataManagement.planectrl = true;
                Console.WriteLine("Plane Control ON");
            }
        }

        public void NeglectDim(int i)
        {
            if (DataManagement.neglections[i] == 1)
            {
                DataManagement.neglections[i] = 0;
            }
            else
            {
                DataManagement.neglections[i] = 1;
            }
            Console.WriteLine("Dimension Control Status:");
            Console.WriteLine("C: Considered, N: Neglected (permanently)");
            for (int j = 0; j < 6; j++)
            {
                Console.Write(DataManagement.dictionary[j]);
                if (DataManagement.neglections[j] == 1) Console.Write(": C, ");
                else Console.Write(": N, ");
            }
            Console.Write("\n");
        }

        public void Record()
        {
```

```
            //createTarget
            int i = 0;
            if (DataManagement.recordingIntention || DataManagement.recording)
            {
                DataManagement.recordingIntention = false;
                DataManagement.recording = false;
                Console.WriteLine("The rocording has been cancelled.");
            }
            else
            {
                while (i < DataManagement.storage.Length - 1 && i < 30)
                {
                    i++;
                }
                if (i == 31)
                {
                    Console.WriteLine("Storage capacity full. Remove a recording before
starting a new one.");
                }
                else
                {
                    DataManagement.currentTarget = "Rec" +
DateTime.Now.ToString("HHmmss");
                    DataManagement.storage[i] = DataManagement.currentTarget;
                    DataManagement.recordingIntention = true;
                    Console.WriteLine("The recording will begin after you clap your
hands.");
                }
            }
        }

        public void Play(string file)
        {
            //Console.WriteLine(ind);
            //string file = DataManagement.storage[Convert.ToInt32(ind)];
            Console.WriteLine(file);
            DataManagement.currentTarget = file;
            DataManagement.playing = true;
            DataManagement.recording = false;
            DataManagement.Play();
        }

        public void doNothing()
        {

        }
    }
}
```

Code 3. Commands.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using Erghis;


namespace ErghisExternal
{
    public class ExternalListener : ErghisListener
    {

        ErghisController EController;

        public ExternalListener(ErghisOrbStyle style, ErghisController ec)
            : base((ErghisOrbStyle)style)
        {
            EController = ec;
        }

        public override void OnErghisFrame(Data data)
        {
            base.OnErghisFrame(data);

            if (data.Hands!= null)
            {
                if (MainProgram.firsttime)
                {
                    MainProgram.olddata = data;
                    MainProgram.firsttime = false;

                }
                else
                {
                    MainProgram.olddata = MainProgram.data;
                }
                MainProgram.data = data;

                //Console.WriteLine("Hands detected!");

                string keymap= ErghisExternal.Program.EC.CurrentKeymap.filename;
                bool motionMode = (keymap == "freejogging" || keymap == "jogbyjoint" ||
keymap == "tracking");

                //Turn ON/OFF the sending of data because of a clap
                if (ErghisExternal.Clap.isAClap())
                {
                    if (!motionMode)
                    {
                        Console.WriteLine("Enter a Motion Mode before starting to send
data. Clap again to restart.");
                    }
                    else if (DataManagement.on)
                    {
                        DataManagement.on = false;
                        Console.WriteLine("Stoped");
                        if (DataManagement.recording)
                        {
```

```
                        DataManagement.recording = false;
                        DataManagement.recording = false;
                        Console.WriteLine("Recording has ended.");
                        OwnFunctions.updateFiles();
                    }
                }
                else
                {
                    Console.WriteLine("Started. Capturing hands reference...");
                    System.Threading.Thread.Sleep(1000);
                    DataManagement.setOrigin = true;
                    DataManagement.on = true;
                    if (DataManagement.recordingIntention)
                    {
                        DataManagement.recording = true;
                        DataManagement.recordingIntention = false;
                    }
                }
            }

            //Turn OFF the sending of data because of mode
            if (DataManagement.on && !motionMode)
            {
                DataManagement.on = false;
                Console.WriteLine("Stoped");
                if (DataManagement.recording)
                {
                    DataManagement.recording = false;
                    Console.WriteLine("Recording has ended.");
                    OwnFunctions.updateFiles();
                }
            }
            MainProgram.justLost = true;
            //Act consequently
            if (DataManagement.on) DataManagement.Manage();
        }
        else
        {
            //Console.WriteLine("No hands detected!");
            MainProgram.startingTime = DateTime.Now;
            if (DataManagement.on)
            {
                if (MainProgram.justLost)
                {
                    MainProgram.exitTime = DateTime.Now;
                    MainProgram.justLost = false;
                }

                if ((DateTime.Now - MainProgram.exitTime).TotalMilliseconds > 2500)
                {
                    DataManagement.on = false;
                    Console.WriteLine();
                    Console.WriteLine("Hands track lost.");
                }
            }
            DataManagement.SendZero();
```

```
                MainProgram.firsttime = true;
            }
        }
    }
}
```

Code 4. ExternalListener.cs

```
using System;
using System.IO;
using System.Linq;
using System.Threading;
using Erghis;

namespace ErghisExternal
{
    public class DataManagement
    {

        //Outgoingcommands
        public static double[] outgoingData = { 0, 0, 0, 0, 0, 0, 0 }; //to be send to the
robot

        //Jog by joint
        public static int jogbyjoint;

        //Clapping start
        public static bool on = false;

        //Translations and rotations:
        public static int[] mov = { 0, 0, 0, 0, 0, 0, 0 };
        public static string[] dictionary = { "TransX", "TransY", "TransZ", "RotX",
"RotY", "RotZ", "Gripper" };

        //Origin
        public static double[] ori = { 0, 0, 175, 0, 0, 0 };
        public static bool setOrigin = false;

        //Speed:
        const double TransSpeed = 40;
        const double RotSpeed = 0.1;
        const double JointSpeed = 0.2;

        public static double SF = 1;
        public static double SFJ = 1;
        public static double SFT = 1;

        public static bool handmod = true;

        //Tracking
        const double transScale = 40;
        const double rotScale = 10;

        //Other modes
        public static bool singdimlock = true;
```

```
        public static bool planectrl = true;
        public static int[] neglections = { 1, 1, 1, 1, 1, 1, 1 };

        //Teaching Mode
        public static string path = @"C:\Users\Guillem\Documents\LTH\Master Thesis\My
Project\Project1\Project1\bin\Debug\recorded\";
        public static bool recording = false;
        public static bool recordingIntention = false;
        public static bool playing = false;
        public static string currentTarget;
        public static string[] storage = Directory.GetFiles(DataManagement.path, "*.txt")
                                        .Select(path2 =>
Path.GetFileNameWithoutExtension(path2))
                                        .ToArray();

        //Management
        public static void Manage()
        {

            double[] handmov = new double[6];
            double[] handmov2 = new double[6];

            handmov[0] = (MainProgram.data.Hands[0].Position.Z +
MainProgram.data.Hands[1].Position.Z) / 2; //centerX
            handmov[1] = (MainProgram.data.Hands[0].Position.X +
MainProgram.data.Hands[1].Position.X) / 2; //centerY
            handmov[2] = (MainProgram.data.Hands[0].Position.Y +
MainProgram.data.Hands[1].Position.Y) / 2; //centerZ
            handmov[3] = (MainProgram.data.Hands[0].ZRotation +
MainProgram.data.Hands[1].ZRotation) / 2;  //rotX
            handmov[4] = (MainProgram.data.Hands[0].XRotation +
MainProgram.data.Hands[1].XRotation) / 2;  //rotY
            handmov[5] = -(MainProgram.data.Hands[0].YRotation +
MainProgram.data.Hands[1].YRotation) / 2;  //rotZ (requires sign to be correct)

            Console.WriteLine(handmov[5]);

            handmov2[0] = (MainProgram.olddata.Hands[0].Position.Z +
MainProgram.olddata.Hands[1].Position.Z) / 2; //centerX
            handmov2[1] = (MainProgram.olddata.Hands[0].Position.X +
MainProgram.olddata.Hands[1].Position.X) / 2; //centerY
            handmov2[2] = (MainProgram.olddata.Hands[0].Position.Y +
MainProgram.olddata.Hands[1].Position.Y) / 2; //centerZ
            handmov2[3] = (MainProgram.olddata.Hands[0].ZRotation +
MainProgram.olddata.Hands[1].ZRotation) / 2;  //rotX
            handmov2[4] = (MainProgram.olddata.Hands[0].XRotation +
MainProgram.olddata.Hands[1].XRotation) / 2;  //rotY
            handmov2[5] = -(MainProgram.olddata.Hands[0].YRotation +
MainProgram.olddata.Hands[1].YRotation) / 2;  //rotZ (requires sign to be correct)

            //Set Origin
            if (setOrigin)
            {
                for (int i = 0; i < 6; i++)
                {
                    ori[i] = handmov[i];
```

```
                }
                setOrigin = false;
            }

            double[] interval = { 200, 200, 200, 1, 1, 1 };
            double[] upplimits = new double[6];
            double[] lowlimits = new double[6];
            for (int i = 0; i < 6; i++)
            {
                upplimits[i] = ori[i] + interval[i] / 2;
                lowlimits[i] = ori[i] - interval[i] / 2;
            }

            double[] trackScale = { transScale * SFT, transScale * SFT, transScale * SFT,
rotScale * SFT, rotScale * SFT, rotScale * SFT };
            double[] speed = { TransSpeed * SF, TransSpeed * SF, TransSpeed * SF, RotSpeed
* SF, RotSpeed * SF, RotSpeed * SF, 0 };

            double scalefactor = Math.Sqrt((Math.Pow(MainProgram.data.Hands[0].Position.X
- MainProgram.data.Hands[1].Position.X, 2) +
                                            Math.Pow(MainProgram.data.Hands[0].Position.Y
- MainProgram.data.Hands[1].Position.Y, 2) +
                                            Math.Pow(MainProgram.data.Hands[0].Position.Z
- MainProgram.data.Hands[1].Position.Z, 2))) / 150;
            if (!handmod) scalefactor = 1;

            //Teaching mode
            string target = path + currentTarget + ".txt";

            string keymap = ErghisExternal.Program.EC.CurrentKeymap.filename;
            switch (keymap)
            {
                //LINEAR MODE
                case "freejogging":
                    jogbyjoint = 0;
                    //Free Control => Obtain trans and rots of the TCP
                    //Detected trans and rots
                    for (int i = 0; i < handmov.Length; i++)
                    {
                        if (handmov[i] > upplimits[i])
                        {
                            mov[i] = 1;
                        }
                        else if (handmov[i] < lowlimits[i])
                        {
                            mov[i] = -1;
                        }
                        else
                        {
                            mov[i] = 0;
                        }
                    }

                    //Preparing the outgoing
                    for (int i = 0; i < mov.Length; i++)
```

```
                    {
                        outgoingData[i] = mov[i] * speed[i] * scalefactor * neglections[i]
* OwnFunctions.softener();
                    }
                    break;

                //JOG BY JOINT MODE
                case "jogbyjoint":

                    //Jog by joint => Obtain motion for each joint (1-6)
                    if (jogbyjoint > 0)
                    {
                        //if both hands are Up/Down => move mov[jogbyjoint] (jogbyjoint =
#joint)
                        if (handmov[4] > upplimits[4]) mov[jogbyjoint - 1] = 1;
                        else if (handmov[4] < lowlimits[4]) mov[jogbyjoint - 1] = -1;
                        else mov[jogbyjoint - 1] = 0;

                        //Preparing the outgoing
                        SendZero();
                        outgoingData[jogbyjoint - 1] = mov[jogbyjoint - 1] * JointSpeed *
scalefactor * SFJ * OwnFunctions.softener();

                        //the gripper and can't be controlled from jog-by-joint
                    }
                    else
                    {
                        SendZero();
                    }
                    break;

                //ADVANCED OPTIONS
                case "options":
                    SendZero();
                    break;

                //TRACKING
                case "tracking":

                    jogbyjoint = -1;
                    //Sends the velocity to the robot
                    for (int i = 0; i < handmov.Length; i++)
                    {
                        outgoingData[i] = (handmov[i] - handmov2[i]) * trackScale[i] *
neglections[i] * scalefactor * OwnFunctions.softener() / 0.035;
                    }

                    if (MainProgram.data.Hands[1].isClosed)
                    {
                        SendZero();
                    }
                    break;

                default:
                    SendZero();
                    break;
```

```
            }

            //Single-dimension control case (neglect everything but the major)
            //If right thumb shifted => consider only the bigger movement
            if (MainProgram.data.RightThumbShifted && singdimlock)
            {
                double major = 0;
                for (int i = 0; i < outgoingData.Length-1; i++)
                {
                    if (Math.Abs(outgoingData[i]) > major)
                    {
                        major = Math.Abs(outgoingData[i]);
                        for (int j = 0; j < i; j++)
                        {
                            outgoingData[j] = 0;
                        }
                    }
                    else
                    {
                       outgoingData[i] = 0;
                    }
                }
            }
            //Plane control
            //if leftthumbs are folded => neglect all rotations and the smallest
translation
            if (MainProgram.data.LeftThumbShifted && planectrl)
            {
                //find "smallest" trans
                double minor = Math.Abs(outgoingData[0]);
                int index = 0;
                for (int i = 0; i < 3; i++)
                {
                    if (Math.Abs(outgoingData[i]) < minor)
                    {
                        minor = Math.Abs(outgoingData[i]);
                        index = i;
                    }
                }
                //cancel minor and rots
                for (int i = 0; i < outgoingData.Length; i++)
                {
                    if (i == index || i > 2) outgoingData[i] = 0;
                }
            }

            //Gripper control
            //if left hand is a fist => open/close gripper according to the tilting of the
right one
            bool motionMode = (keymap == "freejogging" || keymap == "jogbyjoint" || keymap
== "tracking");
            if (MainProgram.data.Hands[0].isClosed && motionMode)
            {
                Console.WriteLine("gripper!");
                SendZero();
```

```
                if (MainProgram.data.Hands[1].XRotation > upplimits[4]) outgoingData[6] =
1;
                else if (MainProgram.data.Hands[1].XRotation < lowlimits[4])
outgoingData[6] = -1;
            }

            //Recording actions
            if (recording)
            {
                OwnFunctions.printLive("Recording");
                string dataToStore = String.Join(" ", outgoingData.Select(p =>
p.ToString()).ToArray()) + " " + jogbyjoint;
                OwnFunctions.writeData(dataToStore, target);
            }

            //Send data
            MainProgram.outgoingData = outgoingData;

            ////Print if desired
            //for (int i = 0; i < 7; i++)
            //{
            //    Console.Write("{0} ", outgoingData[i]);
            //}
            //Console.WriteLine();
        }

        public static void SendZero()
        {
            for (int i = 0; i < mov.Length; i++)
            {
                MainProgram.outgoingData[i] = 0;
            }
        }

        public static void Play()
        {
            if (playing)
            {
                string target = path + currentTarget + ".txt";
                OwnFunctions.printLive("Playing");
                using (StreamReader sr = new StreamReader(target))
                {
                    string line;
                    while ((line = sr.ReadLine()) != null)
                    {
                        string[] fields = line.Split(new char[] { ' ' });

                        for (int i = 0; i < 7; i++)
                        {
                            outgoingData[i] = Convert.ToDouble(fields[i]);
                        }

                        DataManagement.jogbyjoint = Convert.ToInt32(fields[6]);
                        MainProgram.outgoingData = outgoingData;

                        //Print if desired
```

```
                        for (int i = 0; i < 7; i++)
                        {
                            Console.Write("{0} ", outgoingData[i]);
                        }
                        Console.Write(jogbyjoint);
                        Console.WriteLine();
                        Thread.Sleep(20);//can be modulated to change speed
                    }
                    Console.WriteLine("\n Playing has ended.");
                }
            }
        }
    }
}
```

Code 5. DataManagement.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using se.lth.control.labcomm;
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace ExtCtrl
{
    class LabCommManagerServer : extctrl2leap.Handler, leap2extctrl.Handler
    {
        private LabCommEncoderChannel enc;
        private LabCommDecoderChannel dec;
        private TcpListener listener;
        private TcpClient clientSocket;

        private Boolean run = false;
        private Boolean close_socket = false;
        private int nbr_of_connections = 0;

        public LabCommManagerServer()
        {
            listener = new TcpListener(IPAddress.Any, 9513);

            while (true)
            {
                if (close_socket)
                {
                    clientSocket.Close();
                    listener.Stop();
                    close_socket = false;
                }
                while (!run)
                {
                    listener.Start();
```

```
                    Console.Out.WriteLine("The program is listening for connections...");
                    clientSocket = listener.AcceptTcpClient();  // Blocking, waiting for
connection(s)
                    IPEndPoint endPoint = (IPEndPoint)clientSocket.Client.RemoteEndPoint;
                    IPAddress ipAddress = endPoint.Address;
                    new Thread(encoderThread).Start();
                    new Thread(decoderThread).Start();
                    nbr_of_connections += 1;
                    Console.Out.WriteLine();
                    Console.Out.WriteLine("Connected to: " + ipAddress+" #:
"+nbr_of_connections);
                    Console.Out.WriteLine();
                    run = true;
                    close_socket = false;
                }
            }
        }


        private void encoderThread()
        {
            enc = new LabCommEncoderChannel(clientSocket.GetStream());
            leap2extctrl.register(enc);
            while (run)
            {
                leap2extctrl coord = new leap2extctrl();
                coord.data = ErghisExternal.MainProgram.outgoingData;
                coord.jogbyjoint = ErghisExternal.DataManagement.jogbyjoint;
                try
                {
                    leap2extctrl.encode(enc, coord);
                    Thread.Sleep(10); //to be disminished
                }
                catch (System.IO.IOException)
                {
                    Console.Out.WriteLine("Connection lost.");
                    Console.Out.WriteLine();
                    run = false;
                    close_socket = true;
                }
                catch (Exception e)
                {
                    Console.Out.WriteLine("Unexpected exception occurred: \n" + e);
                    run = false;
                    close_socket = true;
                }
            }
        }

        private void decoderThread()
        {
            dec = new LabCommDecoderChannel(clientSocket.GetStream());
            extctrl2leap.register(dec, this);
            try
            {
                //dec.run();
```

```
            }
            catch(Exception e)
            {
                Console.Out.WriteLine("Ending decoder thread... Exception: " + e);
                run = false;
                close_socket = true;
            }
        }

        public void handle(extctrl2leap value)
        {
            Console.WriteLine("Got something: {0} {1} {2} {3} {4} {5} {6}",
value.data[0], value.data[1], value.data[2], value.data[3], value.data[4], value.data[5],
value.data[6]);
        }
        public void handle(leap2extctrl value)
        {
            Console.WriteLine("Got sometsdsdsdhing: qp[0] = " + value);
        }
    }
}
```

Code 6. LabcommManagerServer.cs

```
using Erghis;
using ExtCtrl;
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Text;
using System.Threading.Tasks;

namespace ErghisExternal
{
    public class Clap
    {
        //Clap recognision parameters
        static double minSpeed = 35;
        static double maxDistance = 50;

        //Time within the start of the motion and the clap
        static long clapStart;
        static bool clapStarted = false;
        static double maxDuration = 0.4;

        //Time within two consecutive claps
        static DateTime clapEnd;
        public static bool inAClap = false;
        static double interClapsTime = 500;

        public static bool isAClap()
        {

            //Speed of Hands calculation
```

```
            double leftHandSpeed =
Math.Sqrt((Math.Pow(MainProgram.data.Hands[0].Position.X -
MainProgram.olddata.Hands[0].Position.X, 2) +
                                                Math.Pow(MainProgram.data.Hands[0].Position.Y
- MainProgram.olddata.Hands[0].Position.Y, 2) +
                                                Math.Pow(MainProgram.data.Hands[0].Position.Z
- MainProgram.olddata.Hands[0].Position.Z, 2)));
            double rightHandSpeed =
Math.Sqrt((Math.Pow(MainProgram.data.Hands[1].Position.X -
MainProgram.olddata.Hands[1].Position.X, 2) +
                                                Math.Pow(MainProgram.data.Hands[1].Position.Y
- MainProgram.olddata.Hands[1].Position.Y, 2) +
                                                Math.Pow(MainProgram.data.Hands[1].Position.Z
- MainProgram.olddata.Hands[1].Position.Z, 2)));
            double globalHandSpeed = leftHandSpeed + rightHandSpeed;

            //Distance between Hands calculation
            double distance = Math.Sqrt((Math.Pow(MainProgram.data.Hands[0].Position.X -
MainProgram.data.Hands[1].Position.X, 2) +
                                            Math.Pow(MainProgram.data.Hands[0].Position.Y -
MainProgram.data.Hands[1].Position.Y, 2) +
                                            Math.Pow(MainProgram.data.Hands[0].Position.Z -
MainProgram.data.Hands[1].Position.Z, 2)));
            //Console.WriteLine(globalHandSpeed);
            //Console.WriteLine(distance);

            bool clap = false;

            if (globalHandSpeed > minSpeed && !inAClap && !clapStarted)
            {
                clapStarted = true;
                clapStart = DateTime.Now.Second;
            }
            if (clapStarted && !inAClap)
            {
               if (DateTime.Now.Second - clapStart > maxDuration) clapStarted = false;
               if (distance < maxDistance)
                        {
                            clapStarted = false;
                            clap=true;
                            inAClap = true;
                            clapEnd = DateTime.Now;
                        }
            }
            if (inAClap)
            {
                if ((DateTime.Now - clapEnd).TotalMilliseconds > interClapsTime) inAClap =
false;
            }
            return clap;
        }
    }

    public class OwnFunctions
    {
        public static void writeData(string data, string target)
```

```
        {
             using (System.IO.StreamWriter file = new System.IO.StreamWriter(target,
true))
                {
                    file.WriteLine(data);
                }
        }

        public static void printLive(string w)
        {
            Console.SetCursorPosition(0, Console.CursorTop);
            if (DateTime.Now.Second % 2 == 0)
            {
                Console.Write(w + " ...");
            }
            else
            {
                Console.Write(w + "    ");
            }
        }

        public static double softener()
        {
            double y;
            if ((DateTime.Now - MainProgram.startingTime).TotalMilliseconds/20000>1) y=1;
            else y=(DateTime.Now - MainProgram.startingTime).TotalMilliseconds/20000;

            return y;
        }

        public static void updateFiles()
        {
            DataManagement.storage = Directory.GetFiles(DataManagement.path, "*.txt")
                                      .Select(path2 =>
Path.GetFileNameWithoutExtension(path2))
                                      .ToArray();
            for (int j = 1; j <= 30; j++)
            {
                if (j <= DataManagement.storage.Length)
                {

                    Program.EC.RegisterCommand("[File" + j + "]", new
Keymap.VoidDelegate(() => Program.commands.Play(DataManagement.storage[j - 1])),
DataManagement.storage[j - 1]);
                    //Program.EC.RegisterCommand("[File1]", new Keymap.VoidDelegate(() =>
Program.commands.Play(Convert.ToString(1))), DataManagement.storage[j - 1]);
                    //Program.EC.RegisterCommand("[File2]", new Keymap.VoidDelegate(() =>
Program.commands.Play(Convert.ToString(2))), DataManagement.storage[j - 1]);
                    //Program.EC.RegisterCommand("[File3]", new Keymap.VoidDelegate(() =>
Program.commands.Play(Convert.ToString(3))), DataManagement.storage[j - 1]);
                }
                else
                {
                    Program.EC.RegisterCommand("[File" + j + "]", new
Keymap.VoidDelegate(() => Program.commands.doNothing()), " ");
```

```
                }
            }
        }
    }
}
```

Code 7. Definitions.cs

```
using System;
using se.lth.control.labcomm;
/*
sample struct {
  double data[7];
  int jogbyjoint;
} leap2extctrl;
*/

public class leap2extctrl : LabCommSample {

  public double[] data;
  public int jogbyjoint;

  public interface Handler : LabCommHandler {
    void handle(leap2extctrl value);
  }

  public static void register(LabCommDecoder d, Handler h) {
    d.register(new Dispatcher(), h);
  }

  public static void register(LabCommEncoder e) {
    e.register(new Dispatcher());
  }

  private class Dispatcher : LabCommDispatcher {

    public Type getSampleClass() {
      return typeof(leap2extctrl);
    }

    public String getName() {
      return "leap2extctrl";
    }

    public byte[] getSignature() {
      return signature;
    }

    public void decodeAndHandle(LabCommDecoder d, LabCommHandler h) {
      ((Handler)h).handle(leap2extctrl.decode(d));
    }
  }

  public static void encode(LabCommEncoder e, leap2extctrl value) {
    e.begin(typeof(leap2extctrl));
```

```
    {
      int i_0_max = 7;
      for (int i_0 = 0 ; i_0 < i_0_max ; i_0++) {
        e.encodeDouble(value.data[i_0]);
      }
    }
    e.encodeInt(value.jogbyjoint);
    e.end(typeof(leap2extctrl));
  }

  public static leap2extctrl decode(LabCommDecoder d) {
    leap2extctrl result;
    result = new leap2extctrl();
    {
      int i_0_max = 7;
      result.data = new double[i_0_max];
      for (int i_0 = 0 ; i_0 < i_0_max ; i_0++) {
        result.data[i_0] = d.decodeDouble();
      }
    }
    result.jogbyjoint = d.decodeInt();
    return result;
  }

  private static byte[] signature = new byte[] {
    // struct { 2 fields
    17,
      2,
      // array [7] 'data'
      4,
      100, 97, 116, 97,
      // array [7]
      16,
        1,
        7,
        38,
      // }
      // int 'jogbyjoint'
      10,
      106, 111, 103, 98, 121, 106, 111, 105, 110, 116,
      35,
    // }
  };
}
/*
sample struct {
  double data[7];
  int jogbyjoint;
} extctrl2leap;
*/

public class extctrl2leap : LabCommSample {

  public double[] data;
  public int jogbyjoint;
```

```
public interface Handler : LabCommHandler {
  void handle(extctrl2leap value);
}

public static void register(LabCommDecoder d, Handler h) {
  d.register(new Dispatcher(), h);
}

public static void register(LabCommEncoder e) {
  e.register(new Dispatcher());
}

private class Dispatcher : LabCommDispatcher {

  public Type getSampleClass() {
    return typeof(extctrl2leap);
  }

  public String getName() {
    return "extctrl2leap";
  }

  public byte[] getSignature() {
    return signature;
  }

  public void decodeAndHandle(LabCommDecoder d, LabCommHandler h) {
    ((Handler)h).handle(extctrl2leap.decode(d));
  }
}

public static void encode(LabCommEncoder e, extctrl2leap value) {
  e.begin(typeof(extctrl2leap));
  {
    int i_0_max = 7;
    for (int i_0 = 0 ; i_0 < i_0_max ; i_0++) {
      e.encodeDouble(value.data[i_0]);
    }
  }
  e.encodeInt(value.jogbyjoint);
  e.end(typeof(extctrl2leap));
}

public static extctrl2leap decode(LabCommDecoder d) {
  extctrl2leap result;
  result = new extctrl2leap();
  {
    int i_0_max = 7;
    result.data = new double[i_0_max];
    for (int i_0 = 0 ; i_0 < i_0_max ; i_0++) {
      result.data[i_0] = d.decodeDouble();
    }
  }
  result.jogbyjoint = d.decodeInt();
  return result;
```

```
  }

  private static byte[] signature = new byte[] {
    // struct { 2 fields
    17,
      2,
      // array [7] 'data'
      4,
      100, 97, 116, 97,
      // array [7]
      16,
        1,
        7,
        38,
      // }
      // int 'jogbyjoint'
      10,
      106, 111, 103, 98, 121, 106, 111, 105, 110, 116,
      35,
    // }
  };
}
```

Code 8. leap_labcomma.cs

Central Station

This code is written in Python language.

```
#!/usr/bin/env python
'''

'''
#!/usr/bin/env python
'''
    orca_labcomm_basic.py --- Created by Patrik Cairen, LTH, April 2014


    TODO: part .-lc-file in LabComm 2013 and labcomm


'''
from  orca.connection import writer
import orca
import threading
import socket
import leap_labcomm # v2006
import leap_labcomma # v2013
import time
import sys
import labcomm # v2006
import LabComm # v2013
import ast
```

```
class lc_leap_data(object):
    def __init__(self, data, jogbyjoint):
        self.data = data
        self.jogbyjoint = jogbyjoint

class stream_reader():
    def __init__(self, stream):
        self.stream = stream

    def start(self, decoder, version):
        other_version = decoder.decode_string()
        if version != other_version:
            raise Exception("labcomm version mismatch %s != %s" %
                            (version, other_version))

    def read(self, count):
        result = self.stream.read(count)
        if len(result) == 0:
            raise EOFError()
        return result

    def mark(self, value, decl):
        pass

class stream_writer():
    def __init__(self, stream):
        self.stream = stream

    def start(self, encoder, version):
        encoder.encode_string(version)

    def write(self, data):
        self.stream.write(data)

    def mark(self):
        self.stream.flush()
'''
    Note: Needs LabComm --- v.2006 & v.2013
'''
class extc2win( threading.Thread ):
    def __init__(self, encoder=None, decoder=None, signature=None):
        threading.Thread.__init__(self)
        self.encoder    = encoder
        self.decoder    = decoder
        self.signature  = signature
        self._stop      = threading.Event()

    def stop(self):
        self._stop.set()

    def stopped(self):
        return self._stop.isSet()

    def run(self):
        while not self.stopped():
            (recv,_) = self.decoder.decode()
            if recv <> None:
        #print "something received from extctrl"
```

```
                self.encoder.encode(recv, self.signature)
        #print "something sent to Win"


'''
    Note: Needs LabComm --- v.2006 & v.2013
'''
class win2extc( threading.Thread ):
    def __init__(self, encoder=None, decoder=None, signature=None):
        threading.Thread.__init__(self)
        self.encoder    = encoder
        self.decoder    = decoder
        self.signature  = signature
        self._stop      = threading.Event()

    def stop(self):
        self._stop.set()

    def stopped(self):
        return self._stop.isSet()

    def run(self):
        while not self.stopped():
            (recv,_) = self.decoder.decode()
            if recv <> None:
        #print "something received from Win"
        print "Sent: "
        print recv.data
        new = lc_leap_data(recv.data,recv.jogbyjoint)
        self.encoder.encode(new, self.signature)
        #print "something sent to extcrl"


if __name__ == "__main__":
    print "Setting up connections..."
    # Setup ORCA-connection
    o = orca.connection('turing', 2000)
    s = o.open_and_select(['leap2extctrl'], 1, o._directory.input,
o._select_input)
    extc_sign = leap_labcomm.leap2extctrl.signature
    extc_enc = labcomm.Encoder(writer(s))
    extc_enc.add_decl(extc_sign)
    extc_dec = o.open_output(['extctrl2leap'])

    # Setup TCP/IP-connection
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('130.235.83.221', 9513))


    win_enc = LabComm.Encoder( stream_writer(sock.makefile('w', 100)) )
    win_enc_sign = leap_labcomma.extctrl2leap.signature
    win_enc.add_decl( win_enc_sign )


    win_dec = LabComm.Decoder(stream_reader(sock.makefile('r', 100)))
    win_dec.add_decl( leap_labcomma.leap2extctrl.signature)
```

```
    print "Creating and starting threads..."
    a = extc2win(encoder   = win_enc,
                 decoder   = extc_dec,
                 signature = win_enc_sign)

    b = win2extc(encoder   = extc_enc,
                 decoder   = win_dec,
                 signature = extc_sign)
    a.start()
    b.start()

    while not a.stopped() and not b.stopped():
    try:
            time.sleep(0)
        except KeyboardInterrupt:
            b.stop()

    sys.exit(0)
```

Code 9. Client.py

```
#!/usr/bin/python
# Auto generated leap_labcomma

import labcomm

class leap2extctrl(object):
    signature = labcomm.sample('leap2extctrl',
        labcomm.struct([
            ('data', labcomm.array([7],
                labcomm.DOUBLE())),
            ('jogbyjoint', labcomm.INTEGER())]))

class extctrl2leap(object):
    signature = labcomm.sample('extctrl2leap',
        labcomm.struct([
            ('data', labcomm.array([7],
                labcomm.DOUBLE())),
            ('jogbyjoint', labcomm.INTEGER())]))

typedef = [
]
sample = [
    ('leap2extctrl', leap2extctrl.signature),
    ('extctrl2leap', extctrl2leap.signature),
]
```

Code 10. leap_labcomma.py

7. References

[1]. **Kaku, Michio.** *Physics of the Impossible.* New York : Doubleday, 2008. 978-0-385-52544-2.

[2]. **Yanik, Paul M., et al.** *Use of Kinect Depth Data and Growing Neural Gas.* Clemson, South Carolina, USA : s.n., 2012.

[3]. **Masse, Jean-Thomas, et al.** *Human Motion Capture using Data Fusion of Multiple Skeleton Data.* Toulouse, France : s.n., 2013.

[4]. **Werber, Klaudius.** *Intuitive Human Robot Interaction and Workspace Surveillance by means of the Kinect Sensor.* Lund : s.n., 2011. 0280-5316.

[5]. **Mitra, Sushmita and Acharya, Tinku.** *Gesture Recognition: A Survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews.* 2007.

[6]. **Kober, Jens, Bagnell, J. Andrew and Peters, Jan.** *Reinforcement Learning in Robotics: A Survey.*

[7]. **Luck, Kevin Sebastian, et al.** *Latent Space Policy Search for Robotics.*

[8]. **Martinetz, T. and Schulten, K.** *A "Neural-Gas" Network Learns Topologies.* Amsterdam, The Netherlands : Elsevier Science Publishers, 1991.

[9]. **Fritzke, B.** A Growing Neural Gas Network Learns Topologies. *Advances in Neural Information Processing Systems 7.* 1995.

[10].**Lee, Sang Hyoung, et al.** *Learning Basis Skilles by Autonomous Segmentation.* Osaka : s.n., 2012.

[11]. **ABB.** [Online] http://www05.abb.com/global/scot/scot241.nsf/veritydisplay/98ba43a906331fec48257c6f00374818/$file/PR10031EN%20R15_En.pdf.

[12]. **Sanfeliu Cortés, Alberto.** *Curso de Robótica Industrial.* Barcelona : s.n., 2011.

[13]. **Freidovich, Leonid B.** *Control Methods for Robotic Applications - Lecture Notes.* 2014.

[14]. **Blomdell, Anders and Robertz, Sven Gestegård .** *Labcomm tech report.* Lund : s.n., 2014. p. 8.

[15]. **Leap Motion, Inc.** [Online] 2014. https://www.leapmotion.com/product.