POLITECNICO DI TORINO

Master's Degree

in Mechatronic Engineering

Master's Degree Thesis

A modular ROS platform for teleoperation of a KUKA iiwa robot with task-specific haptic feedback

Supervisor
Prof. Cristina Bignardi

Candidate
Matteo Bermond

Supervisor at Imperial College
Prof. Ferdinando Rodriguez Y Baena
Dr. Fabio Tatti
Dr. Riccardo Secoli

December 2021

Abstract

In recent years, robot-assisted orthopaedic surgery has undergone considerable development. At the same time, novel computer vision methods and hardware, are taking on an increasingly important role in medical engineering improving the efficiency of the surgery.

This work presents the design of a ROS platform to teleoperate a 7 degree of freedom robotic arm (KUKA iiwa 7 LBR 800) using a haptic device. The teleoperation aims to realise a hole in a target position which is identified and estimated through an RGBD camera mounted directly on the KUKA's end-effector.

Analysing the advantages of using the haptic feedback and the RGBD camera for the working area setting and calibration, the goal was to maintain the accuracy of the operation, both in terms of position and hole dimension, below one millimetre.

This thesis was developed under the supervision of both Politecnico di Torino and Imperial College London. All the experiments were carried out in the Mechatronics in Medicine Laboratory at Imperial College London.

Contents

Acronyms:

EE: End-Effector

WS: Work Space

IK: Inverse Kinematics

Chapter 1

Introduction

Although it has been over 30 years since the first recorded use of a robot for a surgical procedure, the field of medical robotics is still an emerging one that has not yet reached a critical mass [1].

Although robots have the potential to improve both precision and capabilities of physicians, and maintain the working area cleaner, the number of robots in clinical use is still very small. In orthopaedic surgery, for example, fewer than 5% of surgeons is using available computer technology in their procedures [2].

Applications of robot assistance, range from neurosurgery, orthopaedics, urology, maxillofacial surgery, radiosurgery, ophthalmology, and cardiac surgery.

Compared to other branches of surgery, orthopaedic surgery benefits from the fact that bone tissue is rigid, and therefore easier to adjust, work, and predict its behaviour during the operations. This advantage has favoured the development and use of different surgical systems for joint replacement. Nowadays, all the available orthopaedic surgery systems rely on using external markers to track and recognise the patient's limb in an invasive way since it requires the fixation of the structure in the patient’s bone. It implies possible post-surgery risk of infection and fractures increased. (Figure 1.1)

Figure 1.1 External markers for tracking in knee surgery

The recent development of high accuracy RGBD cameras is a considerable opportunity to overcome this obstacle based on developing a surgical system able to scan the surrounding environment and recognize the patient's anatomy automatically and marker-less.

In this project, a first step towards the development of such a system is presented. It relies on the teleoperation through a 6 DOF haptic device of a 7 DOF robotic arm and using an RGBD camera mounted on the end-effector of the latter to identify the working area and increase the precision of the cut. The project represents the basis for a surgical application based on a markerless system for the implant of orthopaedic prosthetics. Here, the advantages of using haptic feedback to help the surgeon during the operation are considerate to increasing the level of accuracy and reducing the possibility of error.

- In **Chapter 2:** an overview of the past and present state of the art for robot-assisted surgery is presented.
- In **Chapter 3:** the objective of the project and the thesis is presented in detail.
- In **Chapter 4:** the system is generically presented both from the hardware and software point of view.
- In **Chapter 5:** the hardware components are listed and briefly analysed.
- In **Chapter 6:** the software components are described in detail by dividing the section into three main parts, each of which carefully analyses the ROS nodes dedicated to Kuka, Haptic device, and 3D camera, respectively.
- In **Chapter 7:** the experiments performed to verify the validity of the developed algorithm are depicted. A discussion of the results is provided.
- In **Chapter 8:** the conclusions are presented, with a section that discusses the further possible development of the project.

Chapter 2

Background

This section introduces a summary of the principal technologies of medical robotics with a particular focus on robot-assisted orthopaedic surgery.

With the development of minimally invasive surgical techniques in the late 1980s, surgeons no longer needed to physically place their hands within the body to perform an operation.

Minimally invasive surgery (MIS), or minimal access surgery, thus revolutionized the concept of surgical procedures. In MIS, instruments and viewing equipment are inserted into the body through small incisions minimizing the collateral surgical trauma of an access cut and results in quicker recovery [3].

Using robot to assist surgeon, involves considerable advantages. Robot's accuracy, repeatability, and the substantial increase in cleaning and reduction of error during operations, make surgical robots an "extending or enhancing human capabilities" rather than replacing humans, in contrast to the example of industrial automation. Table 1 summarizes the strengths and weaknesses of robots compared with humans in relation to surgery.

	Surgeons	Robots
Advantages	Task versatility Judgment experience Hand-eye coordination Dexterity at millimeter-to-centimeter scale Many sensors with seamless data fusion Quickly process extensive and diverse qualitative information	Repeatability Stability and accuracy Tolerant of ionizing radiation Diverse sensors Optimized for particular environment Spatial hand-eye transformations handled with ease Manage multiple simultaneous tasks
Drawbacks	Tremors Fatigue Imprecision Variability in skill, age, state of mind Inability to process quantitative information easily Ineffective at submillimeter scale	Expensive Cumbersome Large Inability to process qualitative information Not versatile Technology still in infancy

Table 1: Comparison between robots and surgeons in relation to surgery.

To develop this type of technology in order to make them more and more efficient and therefore increase their use in surgery, it is essential that there is communication and mutual understanding among surgeons, engineers, entrepreneurs, and healthcare administrators.
There are different ways to classify the existing devices. A technology-based taxonomy might have categories such as autonomous and teleoperated robots, whereas an application-based taxonomy might have such categories as cardiology and urology. Role-based classifications can be more convenient because they are easily understandable by both the technology developers and the end-users. The procedural role–based taxonomy can be divided into three discrete categories:

1. **Passive role:** The role of the robot is limited in scope, or its involvement is largely low risk.
2. **Restricted role:** The robot is responsible for more invasive tasks with higher risk but is still restricted from essential portions of the procedure.
3. **Active role:** The robot is intimately involved in the procedure and carries high responsibility and risk.

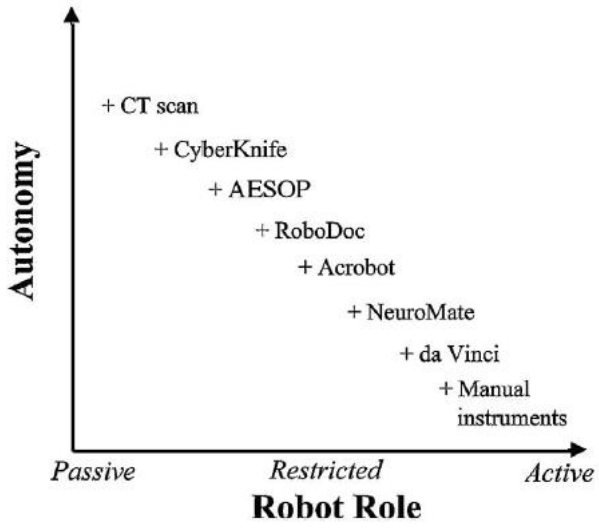


Figure 2.1 Autonomy VS Role for robot in surgery

This project focuses its attention on orthopaedic surgery in which the technology used involves an optimal trade-off between technological autonomy and action powered by the surgeon. Looking at figure 1.2, the robots used for this kind of surgery are the RoboDoc and the Acrobot

which lie in the central area of the chart. The following section presents a summary of the principal technologies and robots used nowadays in orthopaedic applications.

Orthopaedics was an early area of success in surgical robotics due to the rigid and predictable behaviour of bone. Robotic technology used in orthopaedics can be classified based on direct and indirect action and according to the mechanism of cutting, including autonomous, haptic, and boundary control. Robotic Technology in Orthopaedic Surgery [4].

A **computerized tomography** (CT) scan combines a series of X-ray images taken from different angles around your body and uses computer processing to create cross-sectional images (slices) of the bones, blood vessels and soft tissues inside your body.

RoboDoc

RoboDoc (Curexo Technology, Fremont, CA), is an active robot used for the bone-milling in total hip replacement. It is an image-guided system that preoperatively requires the surgeon to view CT images and select the appropriate implant and its placement. The system then generates the cutting path so that it may do this portion of the procedure autonomously. The surgeon must participate in the registration of the preoperative images by locating anatomical landmarks to synchronize the CT images with the physical patient. The preoperative system and the required setup and manual registration, decrease the level of autonomy [5].

Acrobot

The Acrobot Active Constraint Robotics system (Acrobot Company Limited, London, England) was developed for the technically challenging total knee replacement. It is the first generation of *haptic controlled robot* used for bone-drilling. It means that it is controlled by imposing forces on the motors of the robot joints to execute specific tasks and to constrain its motions to a region defined by preoperative images. This approach allows the surgeon to directly feel the forces of cutting but ensures that certain regions are protected from the drill [6]. This is a lower level of autonomy for the robot, and its role is like RoboDoc; however, because it uses small motors, and since the surgeon is in direct control, the system is inherently safer. It is considered to have an active role [7].

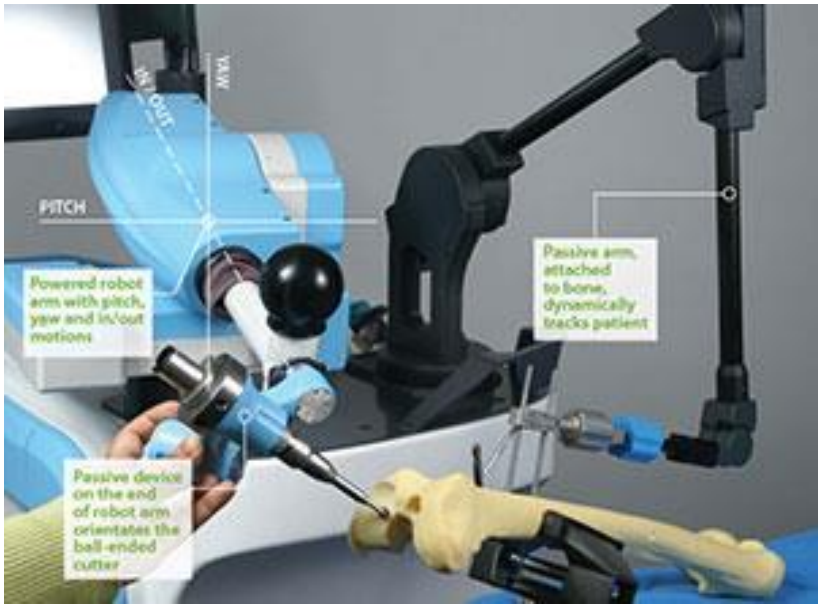


Figure 2.2 The Acrobot's surgical system

MAKO

It is developed for knee, hip, and shoulder replacement surgery. Using the company software, it allows the surgeon to pre-plan the operation and perform it by guiding the robotic arm to precisely remove bone and cartilage. It uses a CT scan to create a 3D virtual model of the unique anatomy of the patient. In addition, during the implant the robotic arm guides the surgeon within the predefined area, allowing for a more accurate and better-aligned knee replacement.

Figure 2.3 Mako unit and example of application

NavioPFS

It is a robotic-assisted knee surgery platform used for both partial and total knee replacement. Using external markers and a 3D camera, it helps the surgeon to create a highly individualized plan that is specific to the unique shape and motion of the patient's knee. The surgeon, using the company software can select the desired implant 3D model and place it on the desired position on the virtual knee reconstruction. The working area is so defined, and the software will elaborate in which part the bone can be cut imposing a specific depth.

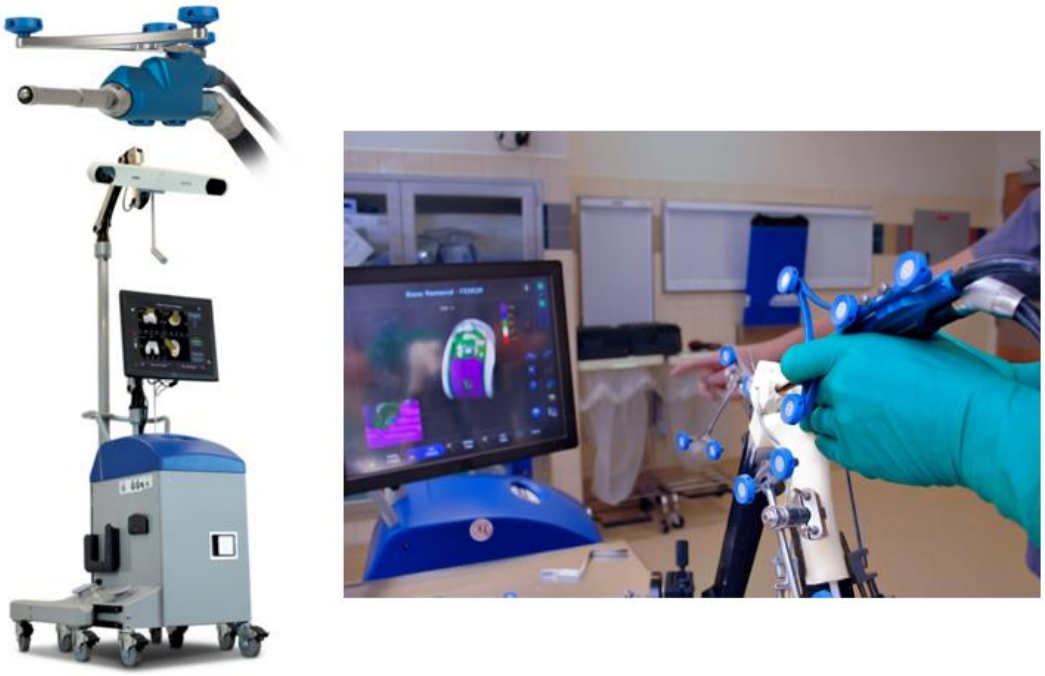


Figure 2.4 NavioPFS knee surgery platform

The platforms presented above represent the past and present of robot-assisted orthopaedic surgery. Computer vision (i.e., the use of cameras for identification, recognition, and three-dimensional reconstruction of the limb) has had a considerable expansion in this type of system.

This project wants to combine the concepts of cooperation between the computer system and the surgeon, working area constraints, haptic guide systems, and integration of camera view in a unique structure.

It is focused on creating a system that lay the foundations for a platform that can bring all these features in an application in which the recognition of the patient's limb can be done automatically by a 3D camera without the help of external markers.

Below are listed and briefly explained the main characteristics to consider during the purchase or design of a surgical robot.

- **Degrees of freedom**

  Number of independent motions of which a robot is capable. In other words, it can be considered as the number of "knobs" one can turn to control the tool tip.

- **Workspace and resolution**

  The workspace is essentially all the space that the end-effector can reach. It is obviously limited by the length of the robot's links, but it is also constrained by joints' limits.

  Resolution is related to workspace; it defines the smallest incremental movement the robot can make or measure

- **Speed, force, and back drivability**

  Every robot has a transmission system to deliver force from its actuator to its end-effector. For particular applications, it is important to bear in mind the trade-off between the force and the speed, as it is expensive to achieve both. For example, in a high force task, such as bone drilling, one should not expect the robot to move quickly.

  Back drivability is essential in case the surgeon should ever want to move the robot by hand while it is unpowered.

- **Dynamic range**

  In a particular surgical procedure, there may be some portion of the operation, such as bone drilling, that requires high force. In the same procedure, there also may be some work, like suturing, that requires a small, fine-resolution force. The ratio between this highest and lowest force is known as the force dynamic range. This is an important specification to know early on, as it can be difficult to design a robot with high dynamic range like human range.

- **Force control VS position control**

  Force control and position control are commonplace terms in the robotics world and, thus, deserve some attention. In position control, the robot attempts to follow some desired trajectory in space. This type of scheme would clearly be appropriate for a robot delivering radiation therapy according to some path in free space. Force control, on the other hand, can be used when the robot is in contact with some surface, because it is often important to control the amount of force it exerts on that surface. A bone-drilling robot might use a force control scheme to ensure smooth cutting combined with position control to stay within delimited region.

- **Bandwidth**

  If an input signal is given to the robot to move back and forth very rapidly, the robot will attempt to execute this instruction and move at the desired rate. As the input frequency is increased, the robot eventually will not be able to keep up due to limits in stiffness and inertia (i.e., the robot is too heavy and floppy, so it falls behind the command). This limiting frequency is known as bandwidth. One can easily see the importance of bandwidth in a teleoperated system.

  We can more precisely speak of bandwidth of motion (i.e., how fast can we follow a commanded position with good fidelity) as well as bandwidth in force control (i.e., how fast can the robot accurately exert commanded forces or adjust to disturbances).

Chapter 3

Objectives

The aim of this thesis is to develop an innovative platform for research in the field of robot-assisted orthopaedic surgery. The main differences between this system and the current state of the art are:

- the system does not track and recognize the work area using reflective markers that are usually implanted in an invasive way on the patient's limb, instead it is guided based on the small RGBD camera directly installed near the end-effector of the robotic arm.
- the system is not manually guided by the user at the end-effector, instead it is remotely operated via a haptic device thus ensuring greater flexibility, in terms of feedback provided to the user.

Hence, the goal of this report is to show the advantages of haptic feedback in surgical operation and how an innovative integration of a depth camera able to avoid invasive markers implantation, could improve the cut accuracy in orthopaedic surgery.

Specifically, the following technical objectives have been set:

- Design and realize, using the 3D printer, a customized end-effector to install on a commercial robot. This device has the purpose to hold both a RGBD camera in a specific position such as to observe the surgical scene, and a small cutter to execute controlled cuts.
- Design and develop a software platform able to teleoperate the robot using a haptic device providing to the user both a haptic and visual feedback to guide he/she during the specific task execution.

Since the system is intended as a research platform, this report presents a modular ROS platform to teleoperate a KUKA iiwa (7 R800 or 14 R820) robot using the Sigma.7 haptic robot. The choice to use ROS to communicate between the different parts of the system is mainly due to the need to ensure excellent flexibility and facilitates changes of codes and the system architecture in future applications.

To identify the target object in the Kuka workspace, a QR code is used; its pose with respect to the camera is read from the latter and through a series of transformations, the position of the QR code with respect to the tip of the tool is get. This procedure allows the identification of the object and of the target working area inside the Kuka workspace.
Due to time constraints, in this initial step, it was decided to renounce the automatic recognition of the anatomy of interest; however, the system was developed with a camera that would allow it in later applications. In this thesis, it was decided to use ArUco codes that allow the identification and estimation of the laying of precise QR codes to simplify the guide of the robot.
In order to demonstrate the advantage of using the haptic device and the depth camera to improve the accuracy of the cutting operation, different tests have been considered:

- Teleoperation without haptic feedback
- Teleoperation without haptic feedback using a camera view to bring the robot in the desired cut position
- Teleoperation with haptic feedback
- Teleoperation using both camera view and haptic feedback

The cut in the configurations listed above have then been compared in terms of target position accuracy, repeatability of the operation and cleanliness of the cut.

Chapter 4

System overview

In this report a modular ROS platform for the teleoperation of a KUKA iiwa 7 R800 (or 14 R820) robot manipulator guided by a haptic device, is presented.
The system is a combination of Python, C++ and Java scripts which supports the control of both the devices using the open-source software MATLAB Toolbox Server to control the Kuka and Chai3D to deal with the haptic device.

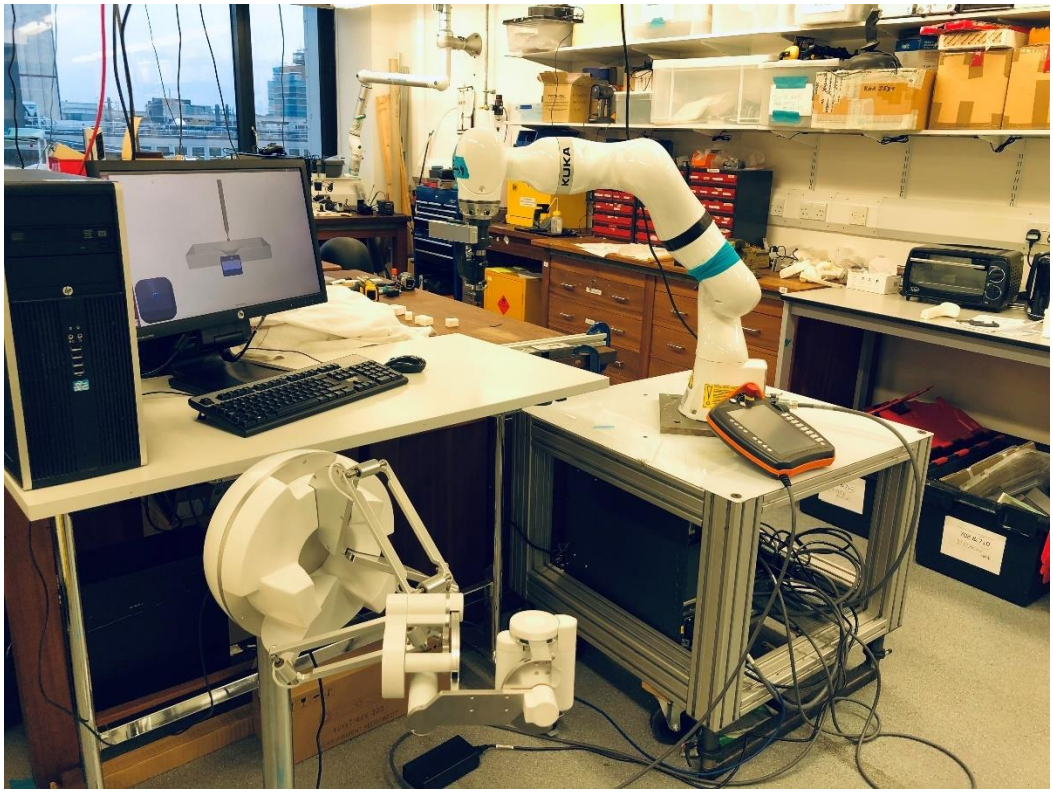


Figure 4.1 System platform: working area, Kuka iiwa, Sigma7 and CHAI3D virtual environment

The system, shown in figure 4.1, is composed by three main environments:

- CHAI3D virtual world.
- Kuka iiwa workspace.
- Sigma.7 haptic device workspace.

These have been calibrated so that the virtual drill in CHAI3D and the KUKA end-effector mirrored the movement of the haptic device guided by the user.
During the project, as mentioned in chapter 3, both the hardware and the software aspects of the system have been designed and implemented. A customized end-effector for the KUKA iiwa has been created to hold both the cutter to execute the operation and a small RGBD camera to observe the surgical scene (figure 4.2).
Below is a list of all the elements and hardware and software components that have been realized in this work; these will then be presented in detail in chapters 5 and 6.

Figure 4.2 Customized end-effector to hold both the mill and the RGBD camera

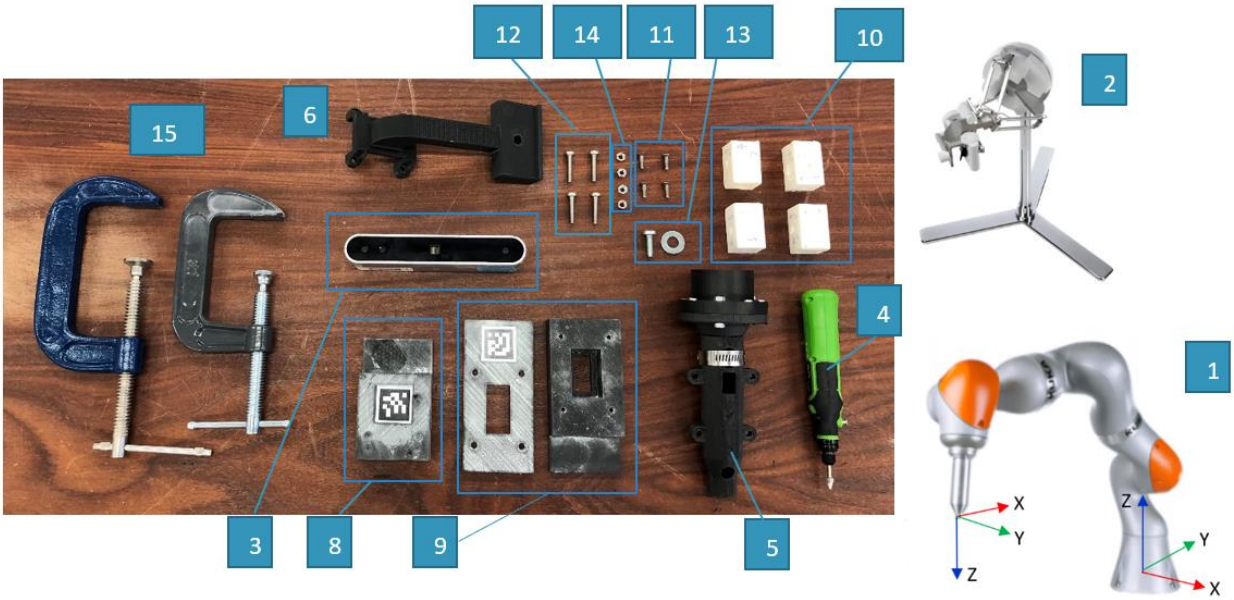


Figure 4.3 Hardware components

HARDWARE COMPONENTS

1. KUKA iiwa 7 R800 or 14 R820

   It is a 7 DOF robotic arms that guarantees an excellent level of accuracy and repeatability. Its geometric conformation allows it to reach the same target point with different poses (redundancy) and its nature as a cooperative robot ensures a disciplined level of safety for both the patient and the operators.
2. Sigma.7 Force Dimension

   A haptic device with 6 DOF. It allows the control of the Kuka's EE both in terms of position and orientation.
3. Acusense 3D camera

   It is a depth camera. It can acquire high-accuracy RGB-D image, point-cloud, and depth map whit an accuracy below one millimetre.
4. Rotary Tool Cordless, Ginour 3.7V
5. Designed terminal component to hold the cutter
6. Designed terminal component to hold the Acusense 3D camera
7. 1 Ethernet cable
8. Camera calibration - Divot
9. Tip-Camera calibration and milled material holder
10. N.4 foam prism one for each test configuration

11. N.4 countersunk screws M4 (4mm x 14mm)
12. N.4 screws M5 (5mm x 30mm)
13. N.1 screw M6 (6mm x 20mm)
14. N.4 nuts M5
15. N.2 clamps
16. N.8 rubber nuts M4 (figure 5.2)
17. Flagship Continuous Fiber Composite 3D Printer
18. 3D snapshot sensor GOCATOR® 3200 SERIES

SOFTWARE COMPONENTS

- Ubuntu Focal Fossa 20.04
- Python 3.8.10
- ROS Noetic (Full Desktop install recommended)
- Numpy 1.18.0
- Rospy
- Yaml python3
- C++
- iiwaPy3 GitHub repository
- KUKA MATLAB-Toolbox-Server on KUKA Smart Pad
- CHAI3D Linux version 3.2.0
- OpenCV 4.2.0
- OpenGL 4.6.0
- Customized ROS nodes
    - KUKA control node
    - Haptic device control node
    - Camera node

The communication between devices is implemented using ROS, an open source, meta-operating system that allows the information exchange in terms of publication and subscription to topics where specific messages have been sent. In the following picture a schematic representation of the ROS communication map is presented.

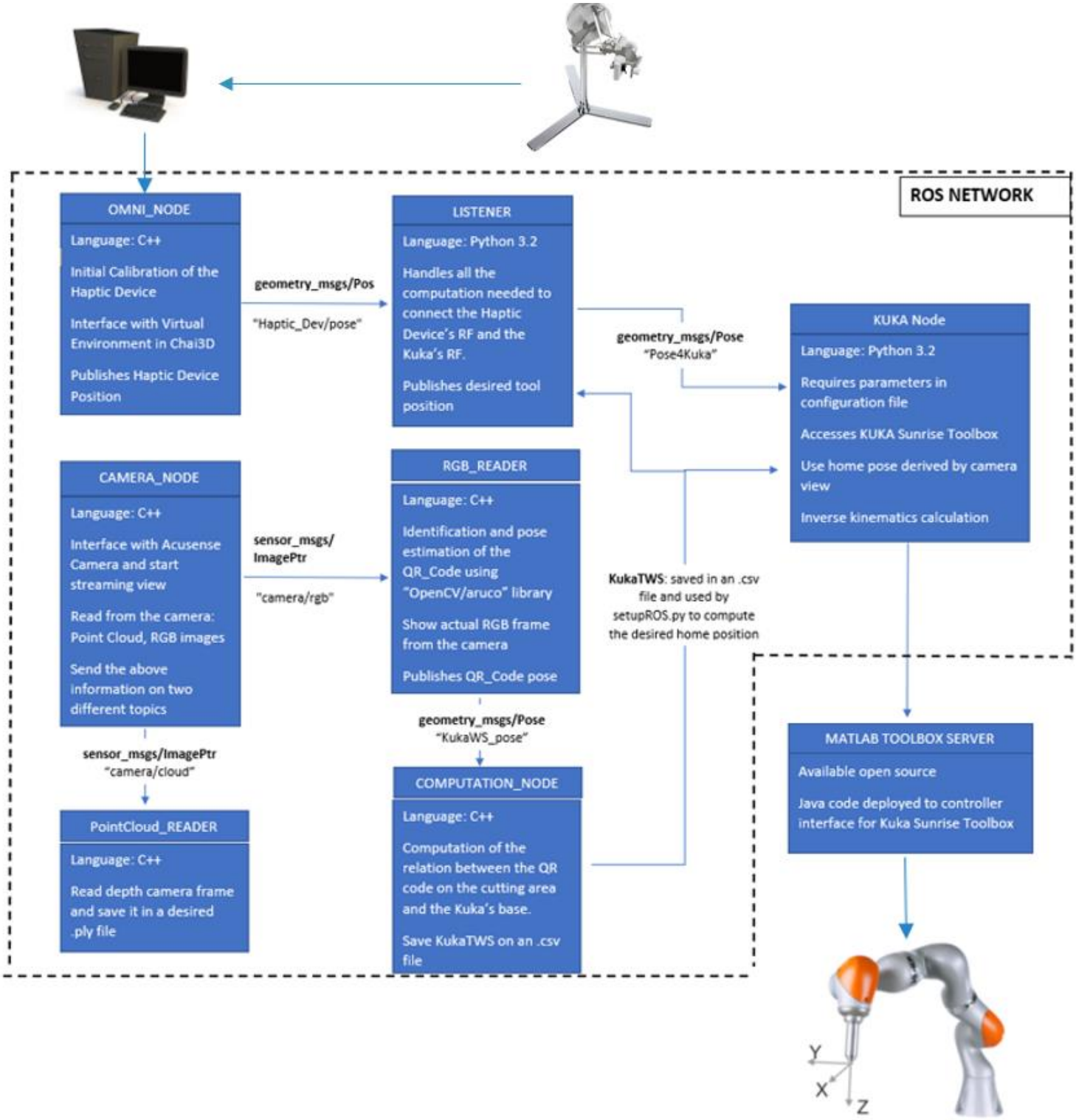


Figure 4.4 System's ROS map

Chapter 5

Hardware components

The designed components are described in the following section in which the motivations that have guided the designing process will be explained in more detail. Most of these components were made with the Mark Two 3D Printer which has been designed to use a material named Onyx™. It is a micro carbon fiber filled nylon that yields accurate parts with near flawless surface finish. It offers high strength, toughness, and chemical resistance when printed alone, and can be reinforced with Continuous fiber to yield aluminium-strength parts. [8]

- **Terminal component to hold the cutter**

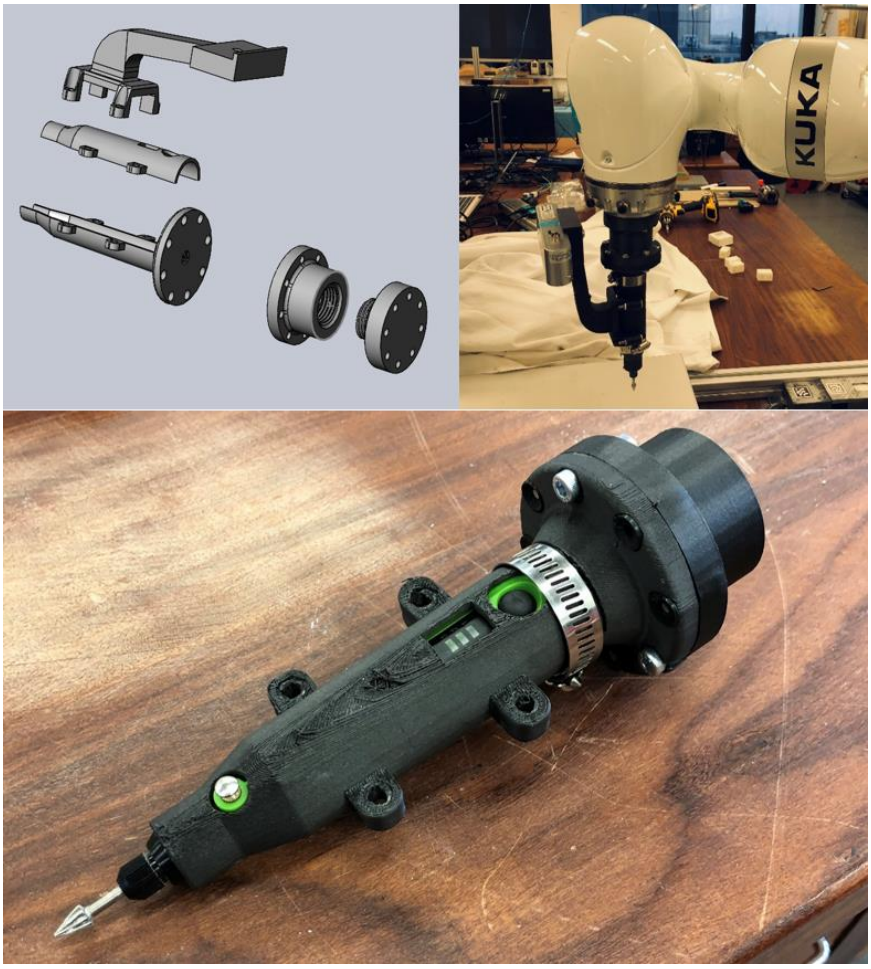


Figure 5.1 Mill holder

The idea followed during the design is to fit the perfect shape of the cutter and pinch it between the two parts of the component by tightening the screws on the appropriate side fins. On the upper parts of the element, some holes have been realized to allow the user to: change the tip, check the drill charge, and start the cutter.
The anchor to the end-effector of the Kuka, as shown in the figure above on the left, was designed with a thread in order to make it quick to fix, stable and precise.
In order to avoid fully rigid forces transmission, a mechanical dumping system has been realized. In the larger circular part of the component, have been introduced eight rubber nuts (Figure 5.2); they are designed to support the drill and to dampen the transmission of compression and torque forces both during the impact with the material and during the shear phase. In figure 5.3 the forces affecting the component and the Kuka's end-effector are highlighted.

Figure 5.2 Rubber nuts

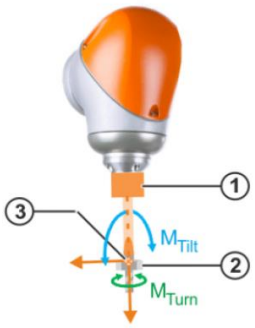


Figure 5.3 Forces and torques on Kuka's end-effector

- **<u>Designed terminal component to hold the Acusense 3D camera</u>**

The element shown in the figure above has been designed to be mounted on the drill holder, over the side fins. The inclination of the camera with respect to the longitudinal axis of the drill, is fixed; it guarantees to:

- Have a good view of the working area during the streaming to give the user the best optic feedback possible.
- A good stability of the camera during the cut, even if the rigidity of the component could lead to some image interferences due to vibration disturbances.

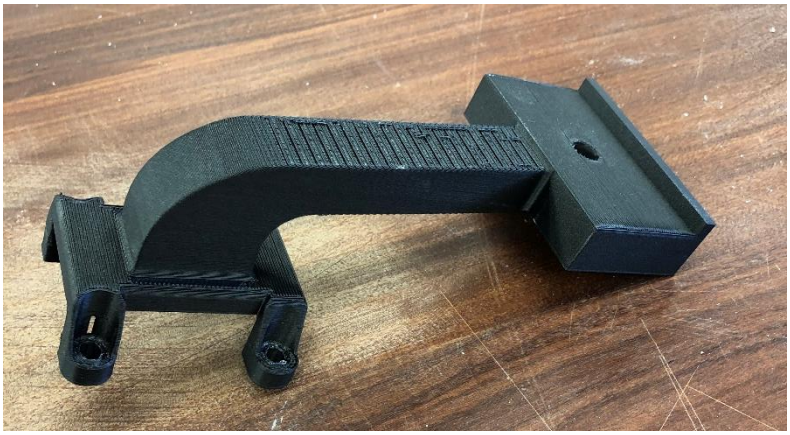


Figure 5.4 Acusense camera holder

- **Camera calibration – Divot**

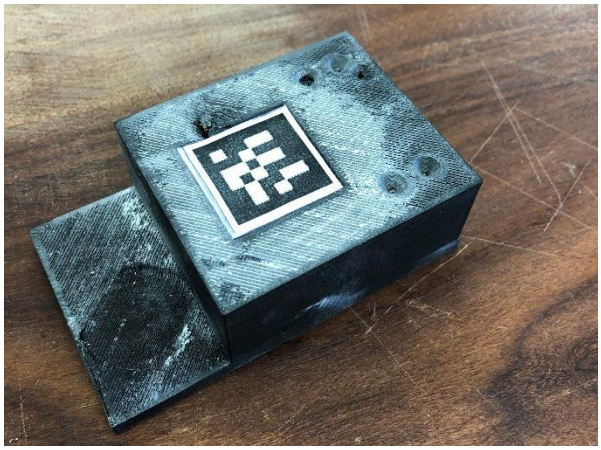


Figure 5.5 Hardware component design for the camera calibration

This element has been designed to become a fixed point in the Kuka's WS.
During the calibration process, the Kuka has been teleoperated to bring the cutter over each divot (conic hole with specific and perfectly known position with respect to the center of the QR code). Afterward, the global position of each divot in the Kuka's RF has been saved in a CSV file and used in a MATLAB script to the calibration process. The calibration procedure will be further analysed in Appendix D.

- **Tip-Camera calibration and milled material holder**

This part aims to define a relationship in terms of orientation and position of the component to be cut with respect to the QR code of the previously analysed component (Divot).

The foam prism is wedged into this component and closed by a cover that prevents it from moving during the impact with the cutter edge.

As shown in figure 5.6 from the 3D model section, the lid has been fixed incorporating, in the printing phase, 4x4M steel nuts.

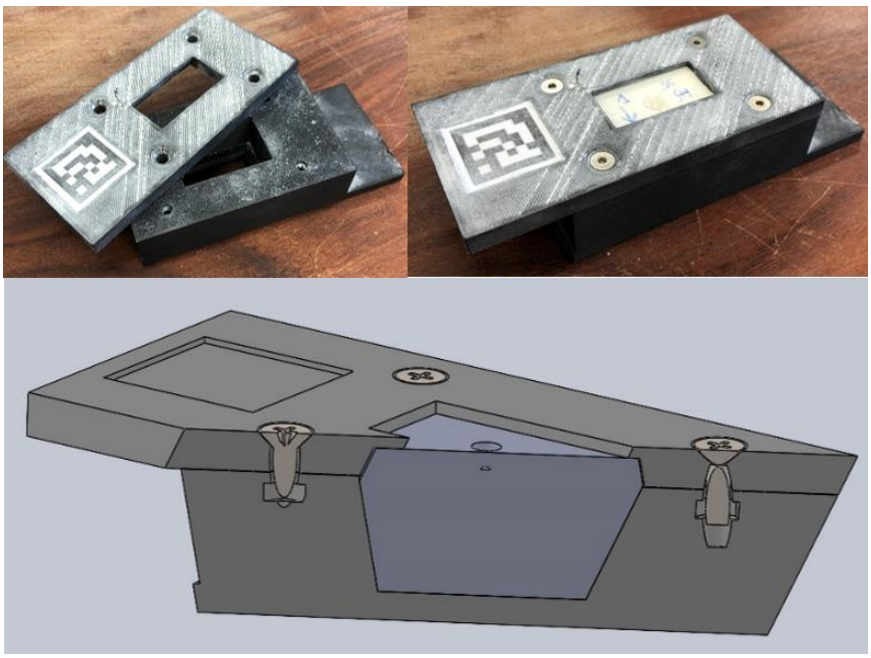


Figure 5.6 Designed hardware to identify the working area position and hold the material to mill

Chapter 6

Software components

In this section, all the software parts of the system designed are analysed in detail.
Here, it is accurately discussed the functionality and structure of the principal ROS nodes with particular attention to the usage of the RGBD camera analysed in Chapter 6.3. To give the user a better understanding of how the connection between the different parts of the system has been made, it presents in Appendix A, an overview of the reciprocal and geometrical connection through rigid roto-translations of every part of the system.

6.1. KUKA control node

The KUKA control node essentially provides a ROS wrapper of the Kuka Sunrise Toolbox. The Kuka Sunrise Toolbox is a Python version of a MATLAB Toolbox which interfaces with a Java application called “MATLAB Toolbox Server” on the KUKA robot controller.
The toolbox provides intuitive functions to control the robot both in PTP (Point To Point) and RT (Real Time) motion configuration.
Figure 6.1 shows an overview of the KST architecture.

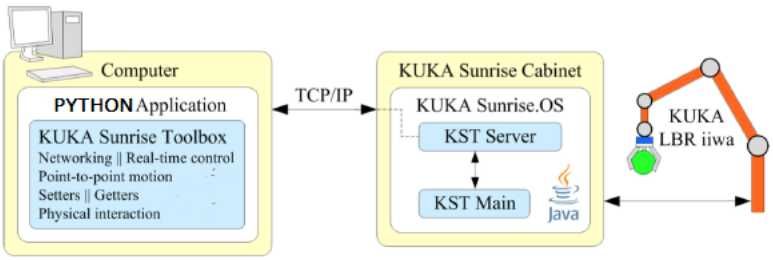


Figure 6.1 KUKA Sunrise Toolbox communication scheme

6.1.1. Script overview

The Kuka control script can be divided in two main nodes:

- **kuka_test.py:** This script is dedicated to the Kuka connection and to send it the input to execute the real-time movement. The control of the robot is a position control, so no forces or joints torques have been considered in this work. The workspace limits and physical constraints are imposed through the haptic feedback of the Sigma.7 robot and imposed by the script setupROS.py.
- **listener.py:** it includes all the necessary computation and Reference Transformations needed to pass from the haptic device to the KUKA reference frame.

Note: The choice to implement a specific node devoted to perform all the necessary computation is due to the need to guarantee a rapid and easy debug of the code and speed up the computation process to avoid or eventually reduce communication lag between the two robots during the teleoperation.
In each of the main nodes, the functions of the iiwaPy3 repository are used both for computation, control and to get information from the Kuka.
In the following table, the list of necessary functions with their main purposes are presented.

Script name	Functionality
iiwaPy3	Connection and communication with the robot
getConstrainedPosition	Verification and eventually modification of the Kuka's EE position to lead within the WS
getScaledPosition	Scaled the position of the Kuka from the WS_center with respect to the position of the Haptic Device in its RF. It allows to have big movement on the Haptic Device side mirrored in small and precise movement of the Kuka's EE, or vice versa.
QuaterniontoRotation	Transform a quaternion in form $[q_w, q_x, q_y, q_z]$ Into its respective 3x3 rotation matrix.
RPY2Quaternions	Transform the orientation information of the Kuka's EE derived from the function **getEEPos()** include in **iiwaPy3.py** into the respective quaternion vector: $[q_w, q_x, q_y, q_z]$
Rot2Quaternions	It has the opposite purpose of the function previously described **QuaterniontoRotation.**
setupROS	It is the function needed to compute and verify the **WS dimensions, center and boundaries** besides compute and impose the **home position** of the Kuka. Here the information (homogeneous matrix) describing the relationship between the QR codes and the Kuka, are used to define the WS center and the home position of the joints in order to maintain a desired configuration.

Table 2 KUKA node scripts overview

6.1.2. KUKA configuration file

All the information about the Kuka and the desired workspace are taken from a configuration file located in **src/kuka/src/config/kuka_params_rt.yaml.**
In this file it is possible to modify the following information about the Kuka and workspace:

Feature name	Functionality
robot	The type of the Kuka used is required in order to use the correct parameters for the Denavit-Hanterberg convention both in the Direct and inverse kinematics computations
flange	Specify the type of flange mounted on the Kuka End-Effector
ip	IP address of the KUKA case. It is possible to get this information on the Kuka smart PAD
center_line	Center line in the Kuka work area with reference to the workspace will be created. **Z must be 0.**
opening_angle	Opening angle of the workspace
z_limits	Vertical constraints of the desired WS
sphere_limits	Inner and Outer limit of the WS that define the WS thickness
tool_length / tool_thicness	Information of the tool used to execute the cut
velocity_ptp	Define the speed used to move the robot in PTP motion when it need to reach the WS_center. Range (0,1).
home_pos	The home position can be different from the WS_center and can be specified in this

	parameter giving the desired angle (in degrees) of each joint
use_home_pos	It is a Boolean parameter used to define if the user wants to use the specified home_position rather than the WS_center afterwards computed.

Table 3 Configuration properties for KUKA node

The range of each of the previous parameters are specified at the top of the same file, so that the user can have a precise idea of what each parameters means and what are the specific and feasible range for the device.
In the same repository, opening the file named **KukaTWS.csv**, it is possible to observe the homogeneous matrix that relate the cutting area RF with respect to the Kuka's RF both from the orientation and position point of view.
The following sections explain how these two files are used to impose the workspace boundary and limit the robotic arm's mobility to ensure the safety of both the user and the patient during the surgical operation.

6.1.3. Workspace definition

Since the working envelope of the KUKA robot follows the shape of a hollow sphere (grey area in figure 6.6) the workspace boundary has been set as a segment of this spherical shape.
As enounced in Table 3 above, the workspace of the robot is defined by the following parameters: center_line, opening_angle, z_lower_limit, z_upper_limit, inner_sphere_limit and outer_sphere_limit.
Figures 6.5 and 6.6 show an overview of what the individual workspace parameters mean.
The tool tip (center of frame) always coincides with the workspace center when, unless otherwise specified in the configuration file, the robot reaches its home position.
In figure 6.3 and figure 6.4, two different dimensions of the workspace are presented; those are simply obtained by modifying the parameters of the configuration file listed above.

The figure 6.2 shows the convention used to determine the orientation and the sense of rotation of each Kuka's joint and in figure 6.7, an exact representation of each joint's reference frame, which are used throughout the software, is presented.

The KUKA manual describes the joint convention as follows. "The positive direction of rotation of the robot axes can be determined using the right-hand rule. Imagine the cable bundle which runs inside the robot from the base to the flange. Mentally close the fingers of your right hand around the cable bundle at the axis in question. Keep your thumb extended while doing so. Your thumb is now positioned on the cable bundle so that it points in the same direction as the cable bundle runs inside the axis on its way to the flange. The other fingers of your right-hand point in the positive direction of rotation of the robot axis" [1, p. 87].

For labelling axes, the standard "XYS_RGB" convention is used in this report.

- X axis corresponds to red
- Y axis corresponds to green
- Z axis corresponds to blue

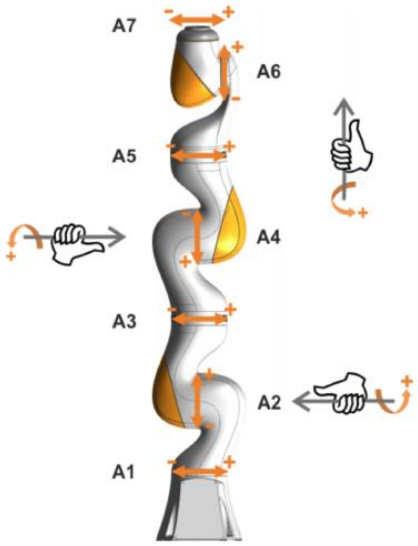


Figure 6.2 Joint convention

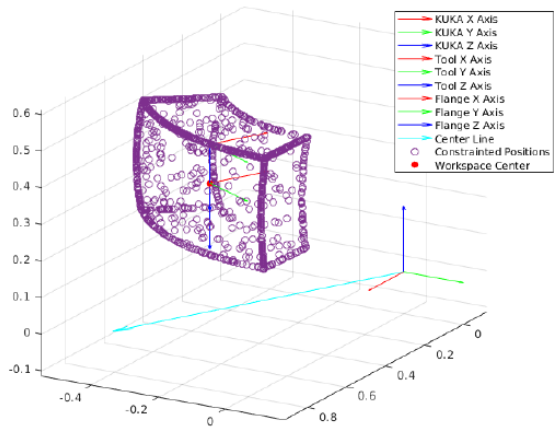


Figure 6.3 Workspace plot (center line [2, -1,0], opening angle 50 deg)

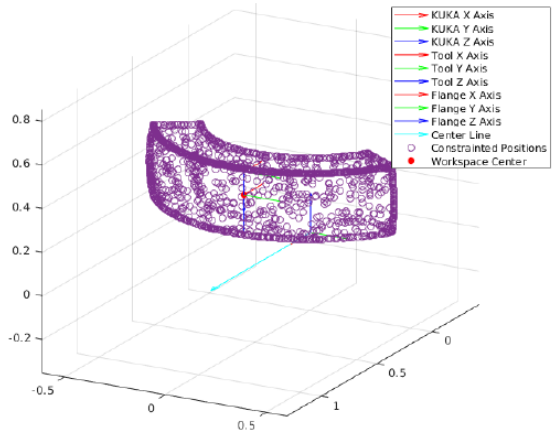


Figure 6.4 Workspace plot (center line [1,0,0], opening angle 120 deg)

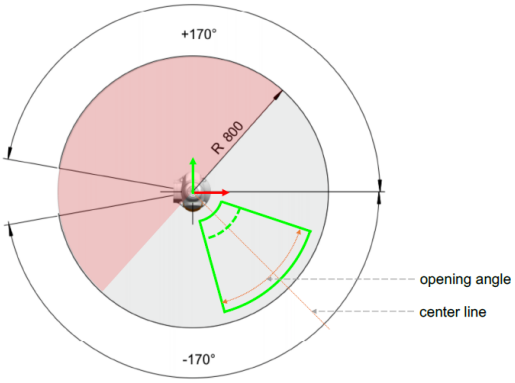


Figure 6.5 Workspace parameters (top view)

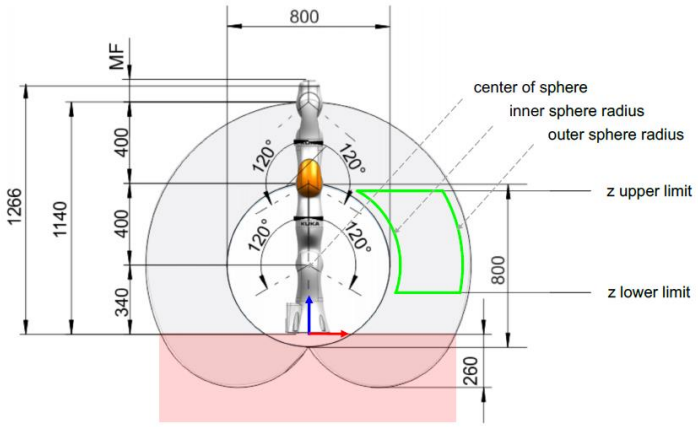


Figure 6.6 Workspace parameters (side view)

6.1.4. Enforcing workspace boundaries

During the execution of the computational script (listener.py), the getConstrainedPosition.py function is used to limit the position of the tool. This function is provided by the GitHub repository iiwaPy3 [9] but it was properly modified to consider the geometry of the tool mounted on the Kuka end-effector to execute the cut.

This function is divided in four main parts and guarantees that the end-effector of the Kuka never exits from the WS previously defined; this is done by updating in real time the position of the end-effector.

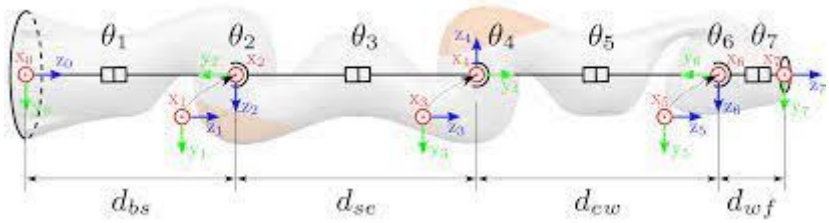


Figure 6.7 Coordinate system

1. If the desired tool position is above/below the allowed Z limit the constrained position is projected down/up along the shortest path to the allowed Z limit. The shortest path is always a vertical projection along z axis. The range of motion on the XY plane is not restricted unless no other boundary constraint is enforced.

2. If the desired tool position is further from/close to the origin of joint 2, than the allowed spherical limits is reached. The position is projected back inside the WS boundaries along a vector pointing perpendicular from the z axis to the desired position.

3. If the desired tool position is beyond the planes defined by the center_line and the opening_angle, the position will be projected down to the closer valid point in the WS boundary along a normal vector of this planes.

4. If the tool position lies within any of the red area in figure 6.5 and 6.6 (i.e., in negative center line direction or in negative z direction) an error is thrown. This is a safety precaution as positions that lie within these areas are unfeasible and might potentially be dangerous. If the control script recognises such an error, the application is shutdown.

Real Time control loop algorithm

```
while true do
    if exit_app_emergency then
        return;
    else if exit_app then
        break;
    if control active then
        get tool transform;
        scale position;
        enforce workspace boundaries;
        solve inverse kinematics;
        send joint position to Kuka;
        real-time motion to calculated joint position;
    else if control inactive then
        PTP motion to home position;
```

6.2. Haptic device control node

The Haptic device control node provides a C++ interface feasible for any haptic device (figure 6.8) and the CHAI3D virtual environment. Here, the haptic feedbacks designed to improve the precision of the cut in terms of quality and position, are created. CHAI3D is an open-source library for computer haptics, visualisation and interactive real-time simulation that perfectly match the needs of this project since it is feasible with most of the haptic device on the market.

Figure 6.8 Haptic devices, from left to right: Sigma.7 (6DOF), Falcon (3DOF), Omni (4 DOF)

The haptic feedbacks are provided by the virtual environment in CHAI3D and by the collision of the virtual drill with the virtual objects.
However, into the code has been pre-arranged all the functions that allow the user to directly impose forces to the haptic device joints.
Note: the coexistence of the two haptic feedbacks (the one provided by CHAI3D and the one directly imposed by the user) is not supported by the software due to interference and conflict between the two modalities; this issue led to instability and vibration of the haptic device manipulator.
For this reason, only the feedback of the virtual environment is considered in this report.
The haptic feedback implemented to guide the user during the cut, are three and are respectively linked to:

- A 3D model of the object to cut with the desired holes and modifications that the user wants to introduce. This element presents a high stiffness so that represents for the Kuka a real constraint on its workspace.
- A haptic guide that allows the user to bring the virtual drill and then the Kuka in the exact position in which the cut must be realized. In order to guide the user, the stiffness of this object is set to a high value too, so that represents for the Kuka a physical guide on its workspace.
- A viscosity sphere that fulfils the environment. This haptic feedback has been implemented as a safety parameter to limit the maximum speed of the robot; a higher viscosity allows to move the Kuka simply through a position control avoiding a more complex impedance control along all the process. In this way, the speed and the impact force applied to the Kuka's joints are reduced imposing a safety limit constraint, indeed.

In figure 6.9 it is possible to observe a schematic overview of the code.

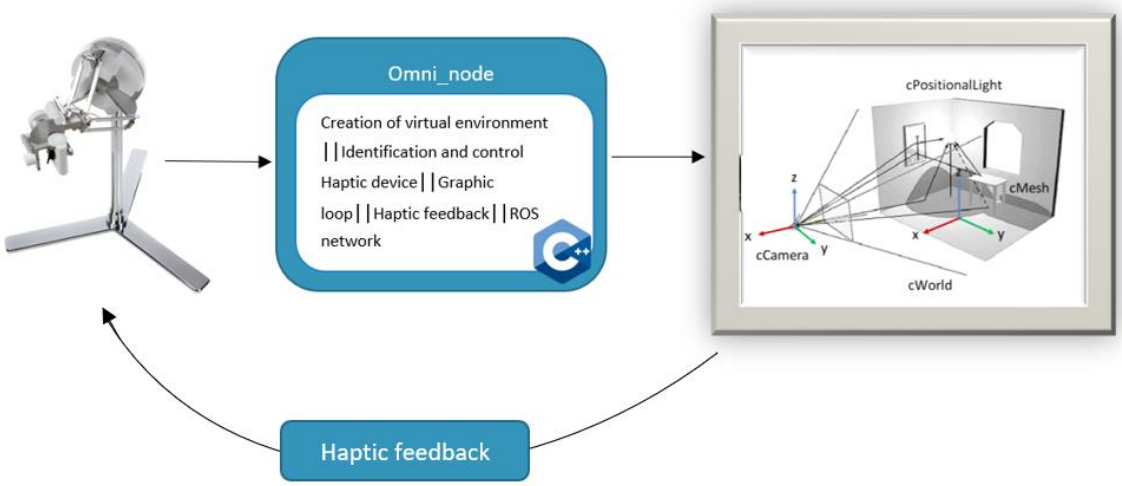


Figure 6.9 Haptic node communication scheme

6.2.1. Script overview

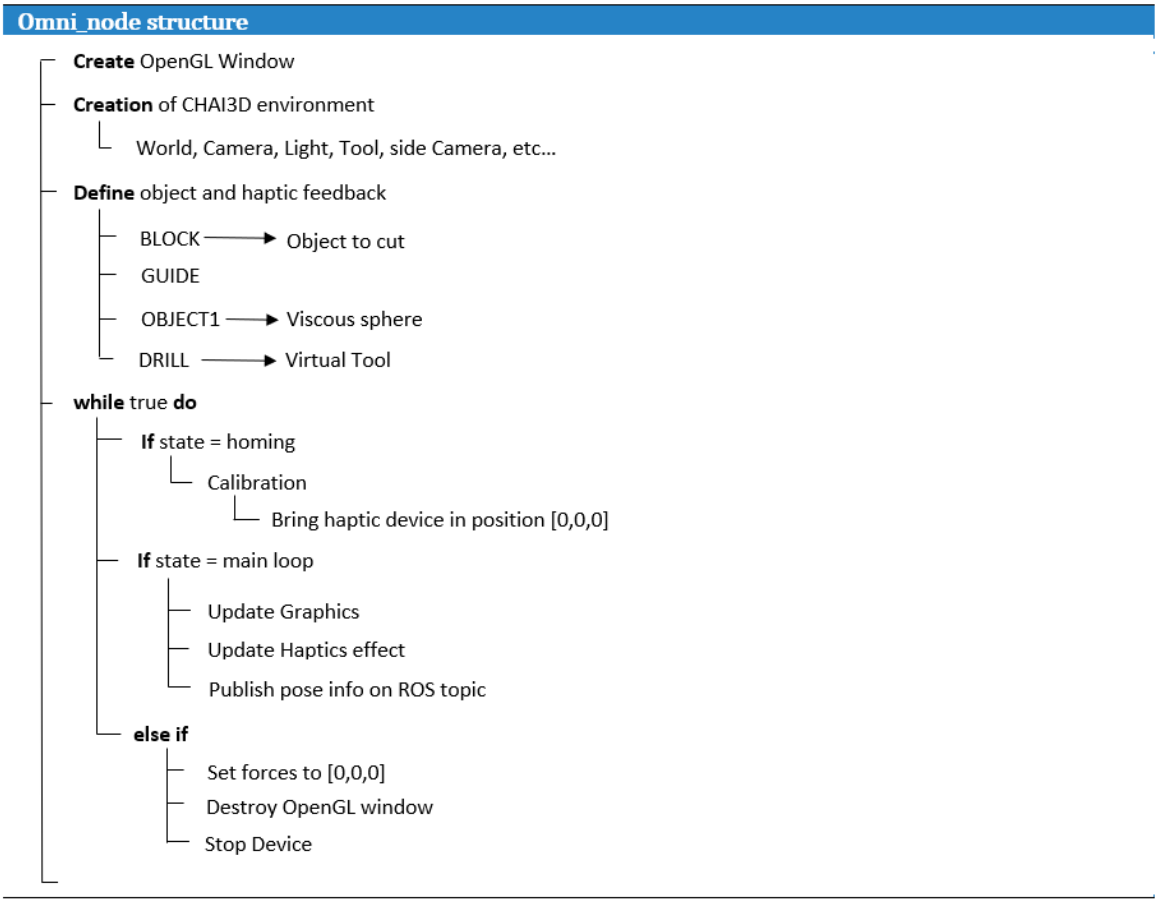
The omni_node is structured in three files whose functionalities are reported in the following table.

File name	Function
Omni_device.h	Header file containing the declarations of all the variables, states, and functions used in the other files. It includes moreover the chai3d.h library needed to interface with the virtual environment.
Omni_device.cpp	Definition of all the functions declared in omni_device.h.
Omni_node.cpp	It is the main file that contains all the commands to create the virtual environment, upload the virtual object and generate the Haptic and Graphic threads needed.

	In this file, the ROS publisher and subscribed are initialized so that the node can communicate with the other parts of the system. In the main thread, the position of the Haptic device is published on the **HapticDev/pose** ROS topic and read by the node **listener.py** of the Kuka node.

Table 4 Haptic node scripts overview

A schematic and intuitive description of the structure of the omni_node is presented in the following algorithm.



6.2.2. CHAI3D

CHAI3D [10] (Computer Haptics and Active Interface) is an open-source set of C++ libraries for computer haptics, visualization, and interactive real-time simulation. Written entirely in C++, CHAI3D was designed to make it easier and more intuitive for developers to create applications that combine 3D modelling with force-feedback rendering capabilities. By supporting different types of force-feedback devices, CHAI3D offers a unique interface to easily design and deploy advanced computer haptic applications.

CHAI3D has grown to become one of the most popular open-source multi-platform haptics rendering frameworks and has been used in many researches and production projects, in such diverse areas as games, simulators, educational software, interactive art, scientific visualization, and *medical applications.* The reason why CHAI3D is perfect for this project since it supports various commercially available three-, six- and seven-degree-of-freedom haptic devices and makes it simple to support new custom force-feedback devices.

The framework also provides the necessary data structures required to create multi-level scene graphs which carry static, dynamic, and articulated bodies. A lightweight OpenGL-based graphics engine provides the foundations for easy rendering of virtual environments using dedicated 3D graphic acceleration hardware.

Object meshes, implicit shapes, volumes, surface materials, and texture properties are all represented in well-organized classes that can easily be extended by the programmer to incorporate more advanced or application-specific features. In figure 6.10 a schematic representation of the virtual environment is shown.

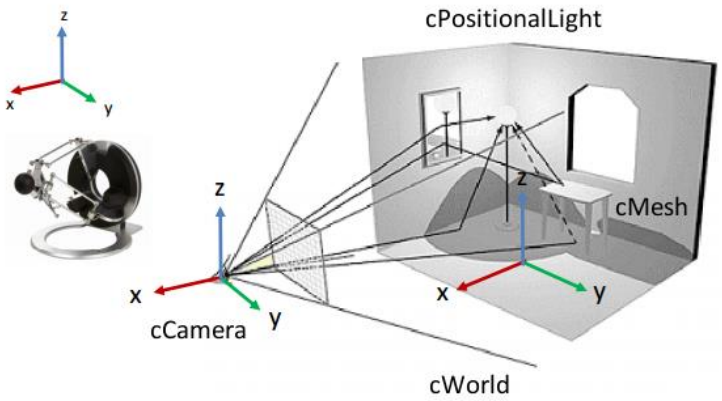


Figure 6.10 schematic representation of a virtual environment in CHAI3D

Figure 6.11 represents instead some examples of possible objects and fields in which CHAI3D can be used.

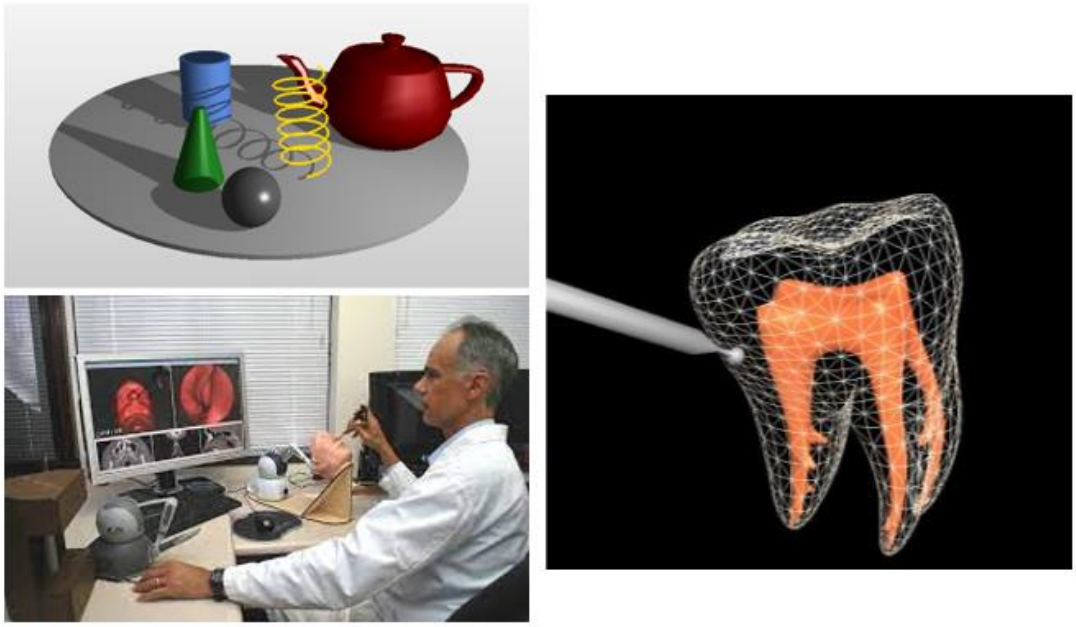


Figure 6.11 Some examples of CHAI3D applications

CHAI3D involves, as a software, the use of several global variables that allow to use both the definition and creation of the virtual environment into the main thread and run separately different threads for the graphic, haptic, and forces update loops.
The first part of the omni_node.cpp file is dedicated to the definition of all the functions threads and variables that need to develop and work in the CHAI3D environment.
Afterward, inside the main thread, all the world, object, and haptic effects were defined; in particular, the following elements have been created:

- World
- Camera
- Light
- Upload designed Mesh object:
    - Element to cut (foam prism)
    - Virtual, transparent, and viscous system to slow down the contact between the cutter and the material
    - Virtual and semi-transparent guide to increasing the precision of the cut

- Physical features have been imposed on the objects:
    - Material
    - Stiffness
    - Viscosity effect
    - Drag effect
    - Magnetic effect
    - Etc…
- A virtual pointer that mirrors the haptic device end-effector movements
- Drill to simulate the cutter
- A second camera that shows what the tip of the tool is facing

In the last part of the code, all the threads are defined to:

- Manage the haptic feedbacks
- Detect collisions between virtual objects and pointer
- Manage the graphic and haptic loops

The haptics effect and the graphic view are constantly updated in separate and independent threads once the haptic device has finished the synch with the Kuka iiwa. Once both the robots are in the desired starting pose, giving input from the keyboard and pressing the first button of the haptic device, these two threads are started, and the user can visualize and feel the effect and object previously described.

6.2.3. Haptic interface

The Haptic interface, as previously mentioned, has been developed through the CHAI3D functions. Hence, the KUKA iiwa is connected directly with the virtual environment without imposing specific forces on the haptic device.
This section presents an accurate explanation of how the features of the virtual objects have been set. In this way, the reader can have a practical idea of what the user feels during the operation.

The objects created or uploaded in CHAI3D are virtually introduced in the haptic device workspace so that the user can practically feel, through haptic effects precisely, the objects as physical elements. In this project, three different models are introduced in the virtual environment:

- *Foam prism:*

  It is the 3D model of the object supposed to be cut. It has been designed using Solidworks and, on its upper face, a hole with the desired position, diameter sand depth has been placed. Since the one goal of this project is to cut the material anywhere but in the desired position, its stiffness has been set to a value fifteen times higher than the maximum factory imposed by the haptic device founded by the software.

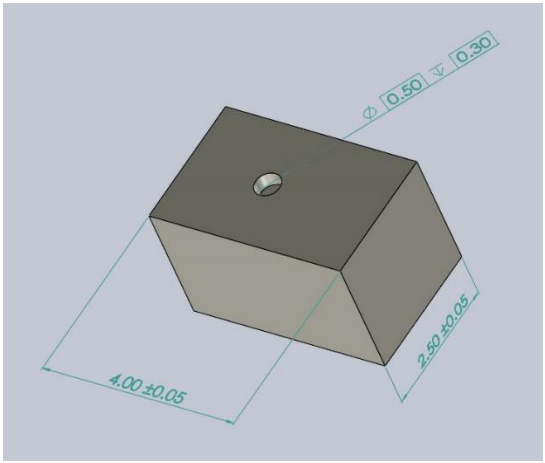


Figure 6.12 Foam prism 3D model with hole in target position

- *Virtual Guide:*

  It is a model designed to help the user to reach the desired position where the hole is supposed to be done.
  A high stiffness has been imposed, and the shape has been thought to keep the cutter two centimetres higher than the foam prism's upper surface so that cuts in unwanted areas of it are avoided.
  The reverse cone shape allows the user to get closer to the target position avoiding excessively high speed and sudden movements of the cutter.

On the top of the cone, a hole of the same dimension as the one realized on the foam block has been placed so that a perfect match between the two parts is guaranteed and the user can't cut the material outside the target position.

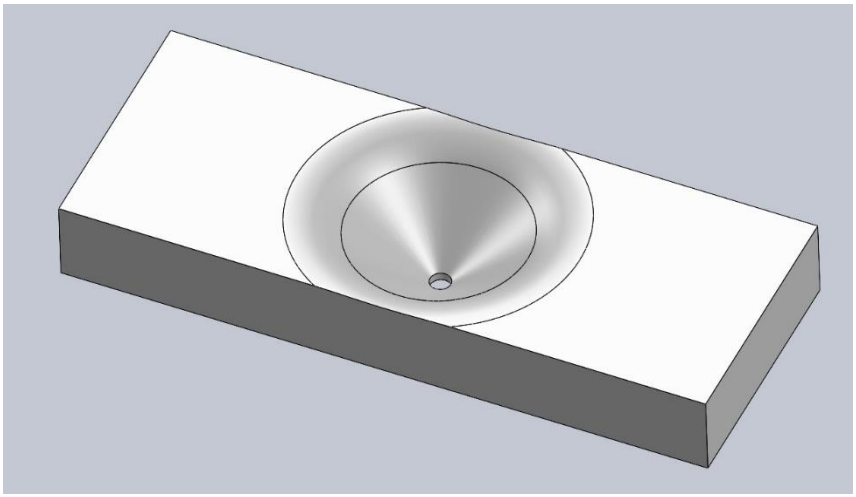


Figure 6.13 3D model of the haptic guide

- *Viscous sphere:*

The dimension of this component, introduced as a Chai3D object, has been set to fulfil all the workspace. High transparency has been set on this component so that it does not introduce any visual disturbance to the user.
The aim of this component is to slow down the movements of both the manipulator of Sigma.7 and the end-effector of the Kuka iiwa consequently. As already mentioned in the previous chapters, this feature allows the control of the robots only in terms of position completely bypassing the more complicated force/impedance control.

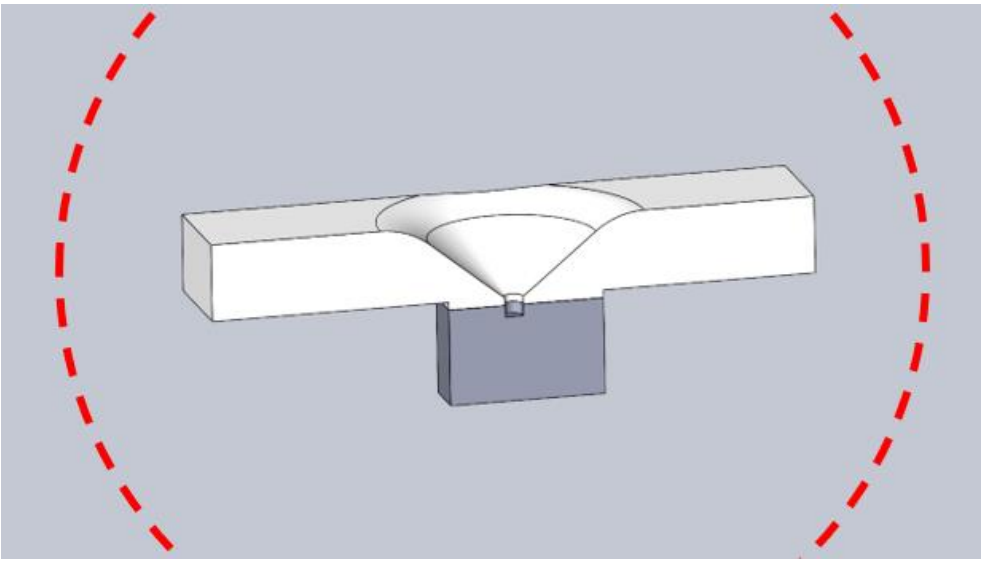


Figure 6.14 3D model in CHAI3D with schematic representation of the viscous sphere for slow down the robot's movements

As previously mentioned, these three components apply haptic feedback communicating with the haptic device through CHAI3D functions without directly imposing any force or torque on the Sigma joint due to input interference. Anyway, since the software has been designed to work in different conditions and to be as more general as possible, all the functions needed to impose directly a force on the manipulator of the haptic device, have been set. These functions are used during the calibration process to bring and maintain the Sigma in the desired position to match the center of its WS with the Kuka's one.

- When the sigma is in *state homing*, that is when the two WS (Kuka and Sigma) are synchronized, on the haptic device are directly set forces that keep the manipulator in the position [0,0,0]. These forces are elastic forces of the form:

$$F_x = (x - x_0) * k - \beta * v_x$$
$$F_y = (y - y_0) * k - \beta * v_y$$
$$F_z = (z - z_0) * k - \beta * v_z$$

Where:

- x, y, z are the actual coordinate of the Sigma manipulator
- x_0, y_0, z_0 are the coordinate of the desired position in which the manipulator must be maintained (in this project set to [0,0,0]).

- k, β are respectively the elastic and damping coefficient set to 500 N/m and 80 Ns/m.

- When the haptic device is in *state main_loop*, the forces and feedback imposed by CHAI3D are used. Only in this state, therefore, are activated the adjustment and updating of the graphics and the haptic so that they do not generate any interference with the forces set in state homing.

6.3. Camera node

The camera node represents the interface between the visual feedback provided by the Acusense 3D Camera and the global system. This node gives the user both the possibility to have visual feedback in real-time of what is happening in the working area and to increase the cut precision using the camera view to update and set the position of the Kuka's end-effector.

6.3.1. Acusense 3D camera

Acusense is an original depth camera created by Revopint®, based on stereo vision and proprietary structured light technology. It can acquire high-accuracy RGB-D image with full-colour texture, point-cloud and depth map at a working distance ranging from 0.2 to 2 meters whit an accuracy that led down the millimetre. [11]

In the Appendix E, an in-depth presentation of the individual types of images produced by the Acusense camera is provided.

As it is shown in figure 6.15 Acusense is designed small and compact, easy to be integrated with robots for diverse applications such as machine vision, security systems and manufacturing.

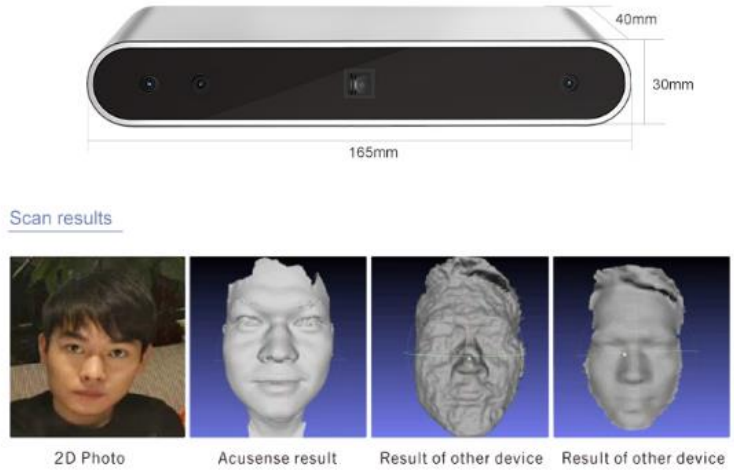


Figure 6.15 Acusense camera

The 3D depth camera adopts stereo vision technology to obtain depth information. With the MEMS micro mirror, coded structured light can be projected fast and precisely, and the infrared laser light causes no damage to human eyes.

Clear and precise texture data is output by the high-resolution RGB sensors,
together with accurate depth data acquired with the proprietary algorithm and chips.
The camera can be connected to computers or smart phones via USB port to capture high-resolution RGB-D image in real time. It is an ideal tool for engineers to develop depth perception.
In figure 6.16 are highlighted the cameras mounted on the device; it is equipped with:

- N.2 IR cameras at the edges
- N.1 RGB Sensor
- N.1 Projector in the middle which aims to grab the point cloud to display a three-dimensional (3D) object on a two-dimensional (2D) surface. These projections rely on visual perspective and aspect analysis to project a complex object for viewing capability on a simpler plane.

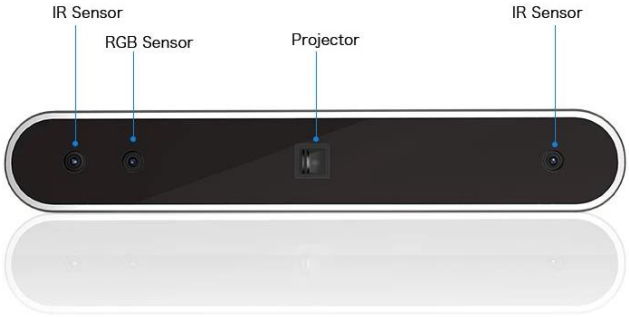


Figure 6.16 Camera details

This device can work both with depth images, point cloud and RGB images. For readers who are novices in the camera view field, a small introduction to these different image formats is presented below.

Colour digital images are made of pixels, and pixels are made of combinations of primary colours represented by a series of codes. A channel in this context is the grayscale image of the same size as a colour image, made of just one of these primary colours. For instance, an image from a standard digital camera will have a red, green, and blue channel. A grayscale image has just one channel [12].

6.3.2. Features

The choice to use the Acusense was due to the fact that it is very versatile and therefore suited very well to the distinctive features of the project.
The main features of Acusense are listed and briefly described below.

1. Cross platform

   The camera is suitable both for Linux and Windows operating systems. It provides valuable and useful elements to improve the quality of the grabbing allowing to set and modify different specifications; for this project this software has not been used and all the parameters and preferences have been directly set in the C++ code.

2. Software development kit available

   The camera supports and can interface with common platforms and languages as python and C++. In this work, Visual Studio and C++ have been used.

3. Agile data transfer

   Data from the camera can be sent to the computer using a Wi-Fi connection or a USB 3.0. The latter was the option used in this project.

4. Post processing software

   The camera developers provide an easy tool to process points cloud grabbed from the camera by stitching, meshing, chipping, or filling holes.

6.3.3 Technical data

In this section, the technical data of the camera are presented in order to give the reader an overview of the kind of device that can be suitable to substitute the Acusense in the application analysed.

The most important parameter taken into consideration for the specific application is that, as shown in figure6.17, the Acusense is able to grab images with good quality from a distance of 250 mm which is quite low compared to the equivalent cameras on the market.

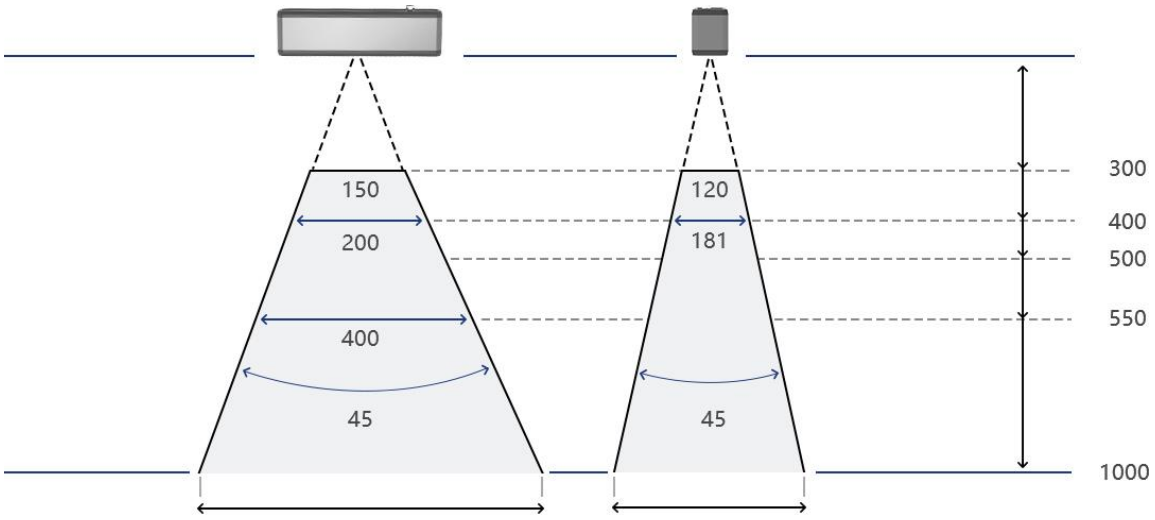


Figure 6.17 Comparison of focusing and distance width between Acusense and general 3D camera

Product Model	Acusense Q1
Technology	Dual camera infrared structure light
Depth range (m)	0.2~2
Power input	5V, 1A
RGB resolution and Frame rate	2560 x 1600 @ 15fps (8M pixel)
	1600 x 1200 @ 15fps (2M pixel)
	800 x 600 @ 15fps (2M pixel)
	640 x 400 @ 15fps (8M pixel)
Depth resolution & Frame rate	1280 x 800 @ 2fps
	640 x 400 @ 10fps
Dimension (mm)	154.6 x 25.6 x 38.2
Accuracy (mm)	±0.1 @ 200
	±0.5 @ 500
	±1 @ 1000
RGB Sensor FOV	H51° x V32° (8M pixels)
	H42° x V32° (2M pixel)
Depth FOV	H49° x V29°
RGBD alignment	Support alignment of depth image and RGB image
RGBD sync	Software sync
Operating system	Linux (Ubuntu 18.04), Windows 8/10
USB interface	USB 3.0
Wireless transfer	Bluetooth 4.0, Wi-Fi 2.4G/5G
Power supply	USB, Hirose 6-pin connector
Laser safety	Class 1 laser, 820~860 nm
Operating environment	Indoor only
Trigger	External trig in/out
Multiple camera control	Wired or wireless; successive control

Table 5 Acusense technical data

6.3.4. Application

The camera is mounted on the end-effector of the Kuka and is held by a hardware device specially created for the cutting modes used in this application as shown in figure 6.18.
Two versions of the device designed to support the camera have been made to better adapt to the use conditions:

- Figure 6.18 shows the first configuration in which the camera is kept vertically with respect to the cutting surface. This configuration is ideal for the calibration of the system with the camera view since allows to visualize the QR codes perfectly from the top minimizing the effect of the prospective issues.
  It is not feasible to observe what is happening during the cut since the orientation
- The second configuration, shown in figure 6.18, has been designed to help the user during the cutting operation so that visual feedback is available. This kind of hardware proved to be ineffective for the calibration because, being the Acusense camera sensitive to perspective, the error of measurement of the mutual distance between the two QR codes' reference systems increased in a conspicuous way.

Figure 6.18 Different configuration of the camera holder mounted on the KUKA's end-effector: first configuration (top right), second configuration (bottom)

The camera, programmed using a C++ code, has been connected to the computer through a USB 3.0 port and communicate with the other part of the system using ROS topic.

6.3.5. Script overview

The Camera node provides a C++ interface used to set the camera parameters, get the different images, and send it on dedicated script to get the necessary information from them.
The node consists of four different codes listed in the table below.

File name	Function
PointCloud_reader.cpp	Get the point cloud images from a ROS topic named "camera/cloud" and save it in a file .ply located in "/home/matteo/catkin_ws". The images can then be saved for different purposes as calibration, estimation, data analysis, etc..
RBG_reader.cpp	Get the RGB images and work on them using the Aruco library to identify the QR codes; estimate their position and orientation; plot their respective RF on an OpenGL window; send the relative position of the QR code on the working area with respect to the fixed QR code to the computation.cpp script to get the WS center in the Kuka work environment.
computation.cpp	Contain and get from ROS topics all the matrices derived from the calibration process and from the pose estimation of the QR codes. In this script, these matrices are used to evaluate the position of the hole in the Kuka RF. This position is saved on a .csv file ("/home/matteo/catkin_ws/src/kuka/src/config/KukaTWs.csv") that is used from setupROS.py to compute the Kuka over the target position.
camera_node.cpp	Ensure the connection with the Acusense camera, set all the images parameters (depth range, RGB format, width & hight,

	auto exposure, frame per seconds, etc…) and send the images to the previous described scripts through dedicated ROS topics.

Table 6 Camera node scripts overview

In the end a fifth script named create_markers.cpp has been used to create the Aruco QR code. This code contains all the information related to the markers that must be reported and inserted in the code camera_node.cpp; if the information used in these scripts do not match, the two QR code will not be identified by the Aruco library when the latter codes is running.
In figure 6.19 a schematic representation of the camera control node is presented to provide the reader an easier overview of the structure and interconnection of each part of it.

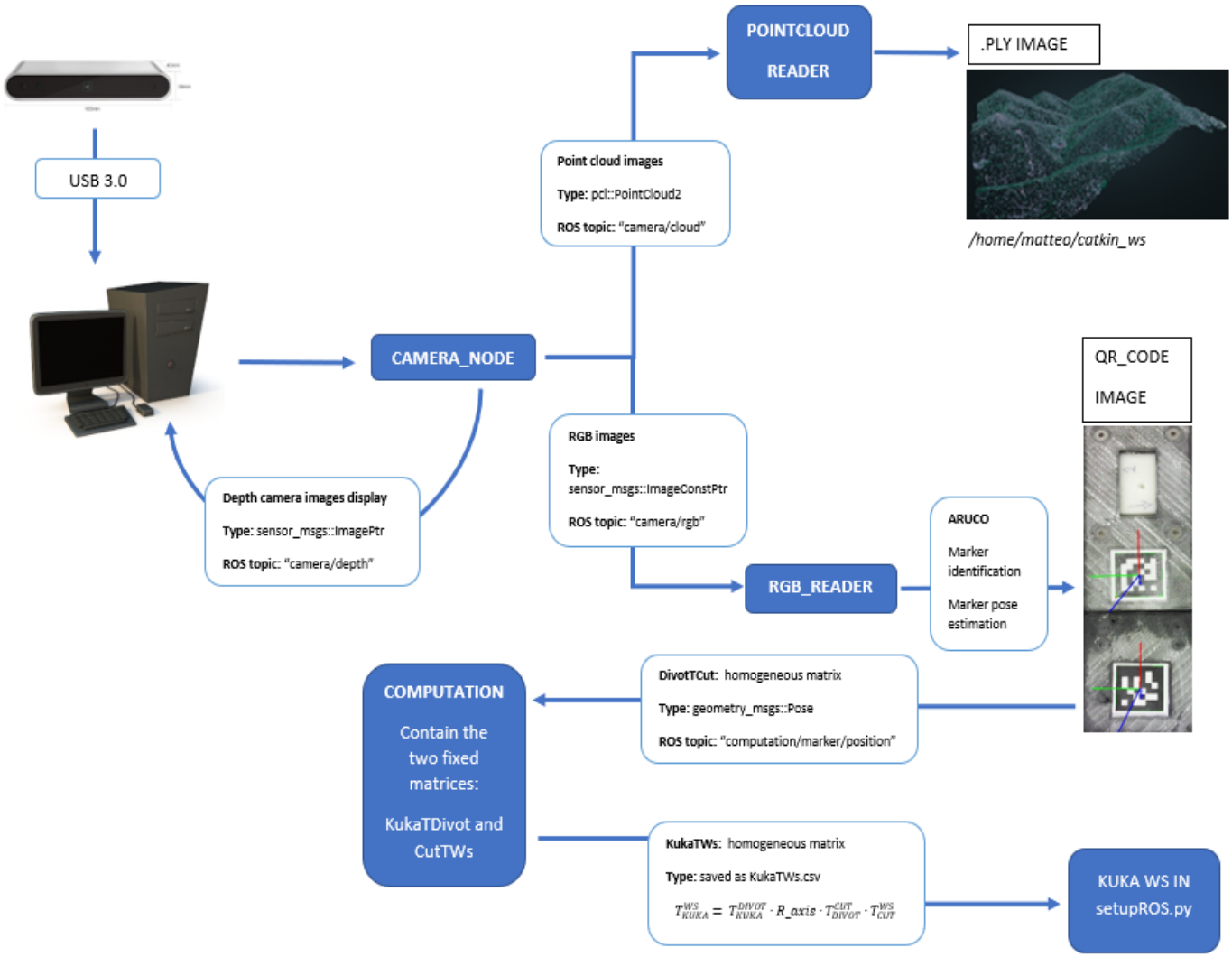


Figure 6.19 Camera node communication scheme

6.3.6. Data collection and computations

In this chapter has been highlighted how all the matrices previously described are used to evaluate the position of the hole with respect to the Kuka RF. To provide a more efficient and rapid consultation of the data transmission between the different codes and software, a graphic representation of the communication has been implemented. (Figure 6.19)

The result of the calibration process consists, as shown in the figure above, in the computation of the homogeneous matrix that connects the reference system of the Kuka iiwa to the position where the cut processing must be start. This matrix has been calculated as follows:

$$T_{KUKA}^{WS} = T_{KUKA}^{DIVOT} \cdot R\_axis \cdot T_{DIVOT}^{CUT} \cdot T_{CUT}^{WS}$$

All the components have been extensively presented in the Appendix B and Appendix C in which an in-depth presentation of the calibration process is provided.

Chapter 7

Experiments and results

The performance analysis of the system has been developed in the form of four tests. Each test is focused on the impact that the two main features of the project (i.e., haptic feedback and camera view) have on the quality, precision, and repeatability of the cut.
To evaluate this influence, on each test has been added or removed one of the previously mentioned characteristics.
The analysis, takes into consideration different aspects:

- Hole size
    - Depth
    - Diameter
- Hole position relative to the centre of the QR code
    - analysis of the error with respect to the expected position.
- Repeatability of the cut
    - for each test, four trials have been carried out to analyse the repeatability of the cut under the same conditions. The evaluation of this parameter has been done through the analysis of *mean*, *standard deviation*, and *standard deviation of the mean* of each work configuration. The standard deviation (SD) measures the amount of variability, or dispersion, from the individual data values to the mean, while the standard error of the mean (SEM) measures how far the sample average of the data is likely to be from the true population mean.

7.1. Tests

The four tests which took place during the experiment have been listed below:

- Cut *without* haptic feedback and *without* camera view (*NCNH*)
- Cut *without* haptic feedback and *with* camera view (*SCNH*)
- Cut *with* haptic feedback and *without* camera view (*SHNC*)
- Cut *with* haptic feedback and *with* camera view (*SHSC*)

Deactivating the haptic feedback means that the user does not have any response from the haptic device and CHAI3D. When he/she touches the guide or the foam block model (look figure in the chapter of CHAI), no stiffness, resistance, or impedance is detected.
The viscous feedback, which is used to reduce the speed of movement of the robotic arm in its workspace, is not eliminated to ensure safety during remote operation.

Deactivating the camera view calibration, the WS centre (a.k.a. the position of the hole in the Kuka RF) has been computed using only the information derived by the CAD models i.e., the distance between the two QR codes and position of the centre of the hole with respect to the QR code RF on the working block. *(*Look appendix B for more information*)*

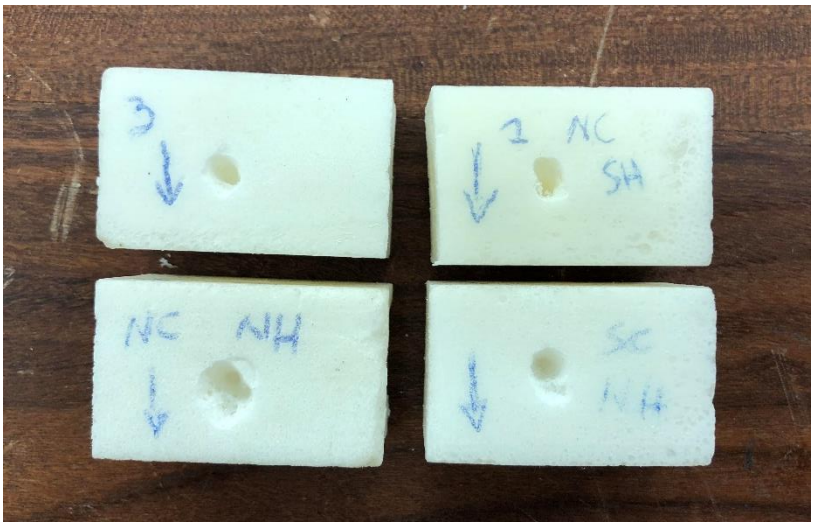


Figure 7.1 Drilling results for each configuration: SHSC (top-left), NCSH (top-right), NCNH (bottom-left), SCNH (bottom-right)

The figure 7.1 above shows the different outcomes for every configuration.
As it is possible to observe the cases in which the haptic feedback has been removed, have imperfections in the cut. This phenomenon is due to the fact that by removing and tightening the cutter during the operation, for example, to remove the milled material from the hole, it is particularly difficult to return to the same position because there is nothing that guides the user and constrains his/her movements. The result is that the dimensions of the hole increase and the cutting surfaces are jagged and discontinuous.

7.2. Results

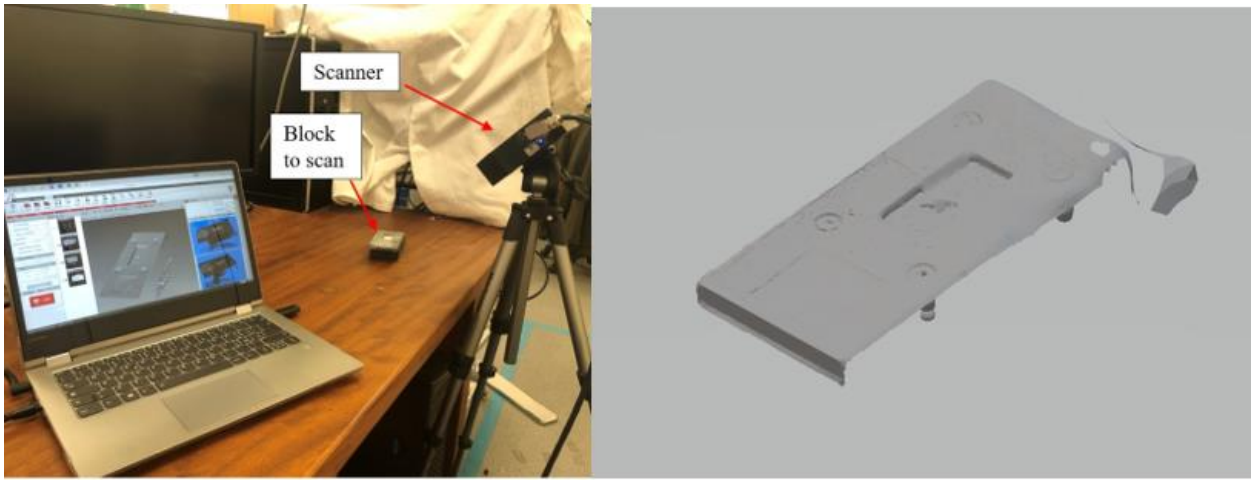


Figure 7.2 Scan procedure and hardware (left) and result model (right)

For each work configuration, the hole is realized on the foam prism teleoperating the Kuka. After every cut, the entire working block (figure) has been removed and scanned using a 3D snapshot sensor (GOCATOR® 3200 SERIES) shown in figure 7.2. The outcome is an STL file that has been processed using a MATLAB script in which the following criteria have been evaluated:

- The position of the centre of the QR code.
- The coordinates of the hole with respect to the QR code.
- The depth of the hole.
- The diameter of the hole.

In the following charts are shown the errors on the average of four tests for each work configuration. In table 7, has been highlighted the standard deviation of the measurements to observe the repeatability of the cut.

The general consideration of the results shows that removing the haptic feedback strongly compromises the quality and repeatability of the cut.

On the other hand, the omission of the camera calibration strongly influences the precision of the hole in terms of position maintaining high repeatability of the operation.

Figure 7.3 Global position error compared with desired result for each configuration

The chart above shows the error on the mean of the global distance of each configuration over four tests. Here the error bars represent the standard deviation of the four samples.

It is possible to observe that in each configuration, the position accuracy remains very high so that the maximum error is presented in the NCSH and is equal to 0.265 mm (definitely less than the target of 1 mm). Even if the maximum error has been obtained in a configuration in which the haptic feedback was active, it is evident that the tests no-haptically assisted, present very high standard deviation underlining the low repeatability of the cut in these conditions.

To give the reader an idea of the difference in terms of data dispersion for each case, these have been remarked in table 7.

CONFIGURATION	POSITION ERROR [mm]	STANDARD DEVIATION [mm]	STANDARD DEVIATION OF THE MEAN [mm]
NCNH	0.127	1.211	0.303
NCSH	0.266	0.561	0.140
SCNH	-0.177	0.957	0.239
SCSH	0.153	0.251	0.063

Table 7 Numerical results for global position error

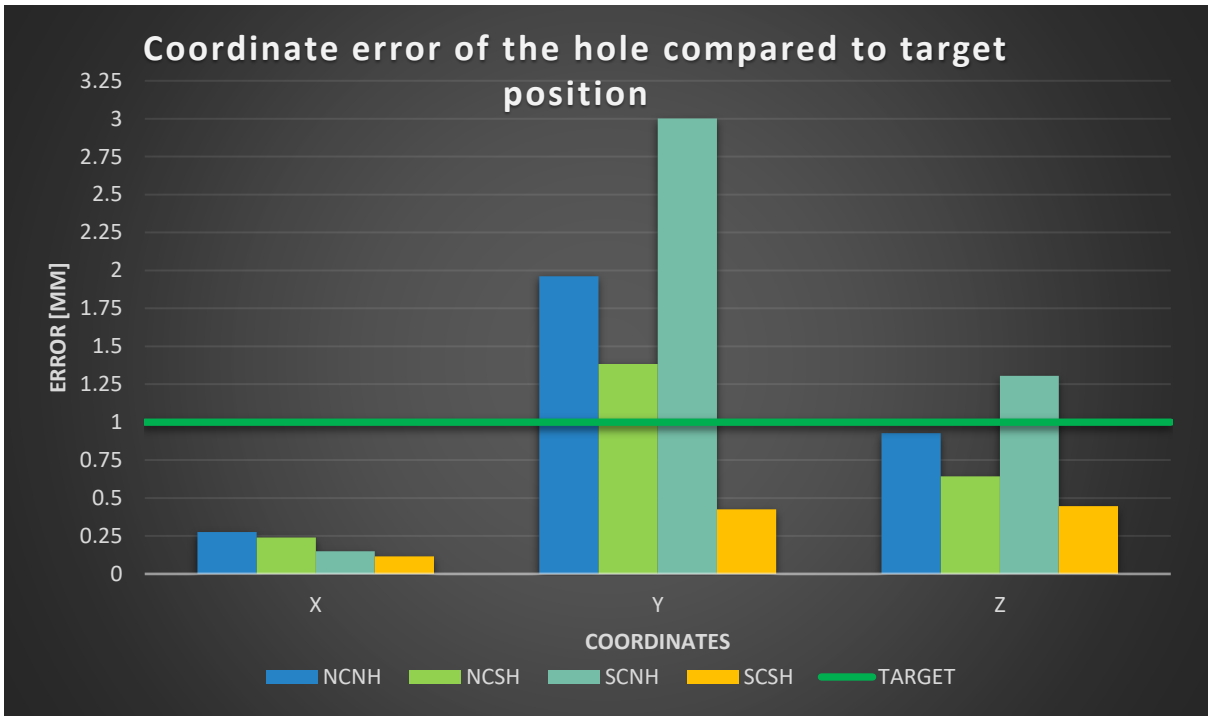


Figure 7.4 Breakdown of error on individual coordinates

Figure 7.4 shows how the error is distributed on the single coordinates for each particular configuration.

It can be said that the desired accuracy is roughly respected in all configurations. The excessive peak in the case of Y and the outlier on the Z are justified as follows:

- *NCNH* test is strongly influenced by the fact that the user was guided only by the CHAI3D visual feedback, and he couldn't feel any active constraint on the working area.
- In the *SCNH* case, the error linked to the absence of the haptic guide has been increased by the camera calibration in which the estimation and the matrix reconstruction was strongly influenced by the camera perspective. This topic has been explained in detail in Appendix D and chapter 6.3.

The unexpected result is the outlier in the *NCSH* test; indeed, the combination of the haptic feedback and the CAD model calibration should have guided the milling in the target position with the desired accuracy. After several considerations, this outlier has been linked to the positioning error of the objects in the virtual environment. The three workspaces calibration lies to an offset along the Y-axis, indeed.

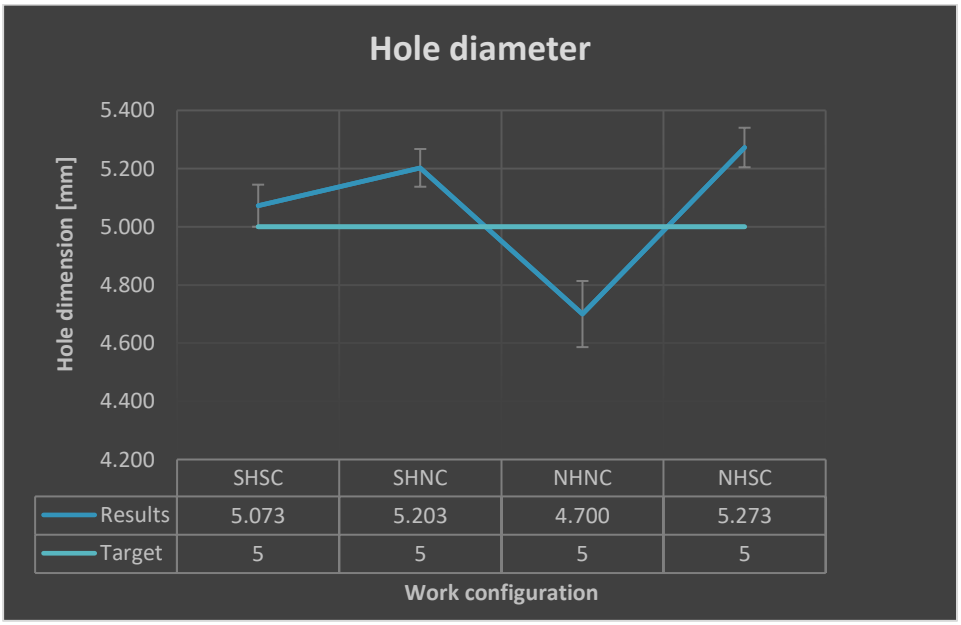


Figure 7.5 Diameter dimension error for each configuration

The last analysis takes into consideration the hole diameters with respect to the target of 5 mm.

The results listed in table and graphically represented in figure shown that the haptic assisted tests present a smaller error up to 0.2025 mm and a quite good repeatability. As expected, the combination of the camera calibration and the haptic guide lie to the best result.

CONFIGURATION	DIAMETER ERROR [mm]	STANDARD DEVIATION [mm]	STANDARD DEVIATION OF THE MEAN [mm]
NCNH	0.300	0.455	0.114
NCSH	0.203	0.259	0.065
SCNH	0.273	0.272	0.068
SCSH	0.073	0.289	0.072

Table 8 Numerical output for the diameter error analysis

Chapter 8

Conclusions

8.1. Conclusions

This report presents an innovative platform to teleoperate a 7 DOF robotic arm (KUKA iiwa 7 LBR 800) with task-specific haptic feedback. Here a novel use of RGBD camera to detect and estimate the position of the working area has been implemented. The goal was to demonstrate that haptic feedback and the use of an RGBD camera directly mounted on the robot's end-effector could significantly increase the efficiency of certain specific tasks that can be replicated in the operating room for orthopaedic surgery.

The application provides both haptic and visual feedback using an open-source software named Chai3D. Here a designed virtual environment guides the user to milling the target object in the desired position and with good repeatability. The sought accuracy, in terms of position, was fixed to lie under 1 mm. The test made on the platforms shows that by implementing both the camera calibration and the haptic feedback, the task accuracy sits under 0.5 mm. This goal is not reachable in other work configurations in which one of the main features have been deactivated, therefore the presented platform and thesis is valid.

8.2. Future developments

The next step of the project aims to merge this platform with innovative software that allows to identify and estimate the working area completely markerless. Using a 3D camera mounted on the Kuka end-effector, the new software will detect and build a model of the patient’s knee and compute the workspace to impose to the Kuka iiwa. Integrating this software with the system shown in this report, it will be possible to automate all the calibration processes with high accuracy in terms of position. The use of the haptic feedback provided by Chai3D

and the haptic device, will increase the performance, efficiency, and cleanliness of the surgery.

This novel platform will allow the disposal of reflective marker devices that today are the basis of robotic-assisted orthopaedic surgery, thus bringing significant advantages in terms of time savings and post-operative rehabilitation of the patient.

In any case, to allow an optimal configuration with this new software, it is necessary to implement some modifications to the current system. These are, therefore, listed and briefly commented.

- The Acusense camera, as already mentioned several times, is strongly affected by the perspective. Hence, it is suggested to use a different camera more stable and tested.
- The cutter used is of low cost and power, making it unsuitable for an orthopaedic application in which you are likely to mill a very rigid and resistant bone tissue. It is, therefore, possible to design and realize a cutter specific for the system and the needs of the project or change the current hardware thus, it can accommodate and support a commercial surgical mill.
- The actual hardware designed to hold the camera and the cutter present the following issues:
    - The stiffness of the rubber nuts used to dump the system was insufficient to sustain the tilting moment and the shear forces. Therefore, it turned out necessary to add a rigid bolt-nut connection.
    - The thickness of the lower part on which lie the side fins is too low and leads to excessive flexibility of the component. For this reason, two range adjustable clamps have been used to stiffen the structure.

Since the tensions mentioned above are strongly present when combining the cutting of a material with a planar or three-dimensional movement to follow a specific geometry, it is required in the following application to rebuild the damping system and increase the thickness of the lower part of the holder.

Figure shows a possible solution for the dumping system where the external case is continuous and rigid, and the dumping is given by the rubber disk highlighted in the picture.

Chapter 9

Appendix

A. Coordinate systems and transformation

In this section all the transformation matrices and the RF relationship are analysed. The analysis is implemented dividing the different homogeneous matrices in two main categories:

- Homing: in which the relationships needed to bring the burr tip in the desired initial position are implemented
- Main loop: in which the relationships between the different devices used during the Kuka's teleoperation are discussed.

For each matrix the following criteria are described: what it represents, how it is obtained and where and when it has been used in the computation. At the end of this chapter, a picture showing every relation in the system is reported (figure A.8) to clarify the interior mathematical connection between each device/part.

- **Homing:**

This category, describing the position of the WS center with respect to the reference frame of the Kuka, contains the three matrices listed below whose are all used in **catkin_ws/src/camera/src/computation.cpp**.

- **QR_Code_Divot->Kuka:**

  It provides the pose of the fixed point represented by the center of the QR code placed on the Divot block with respect to the reference frame of the Kuka.

  In order to obtain this matrix, a calibration process explained into the details in Appendix D has been used.

$$T_{KUKA}^{QRCode\_DIVOT} = \begin{bmatrix} 0.7851 & 0.6194 & 0 & 563.5 \\ -0.6194 & 0.7851 & 0 & -153.83 \\ 0 & 0 & 1 & 88.8575 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

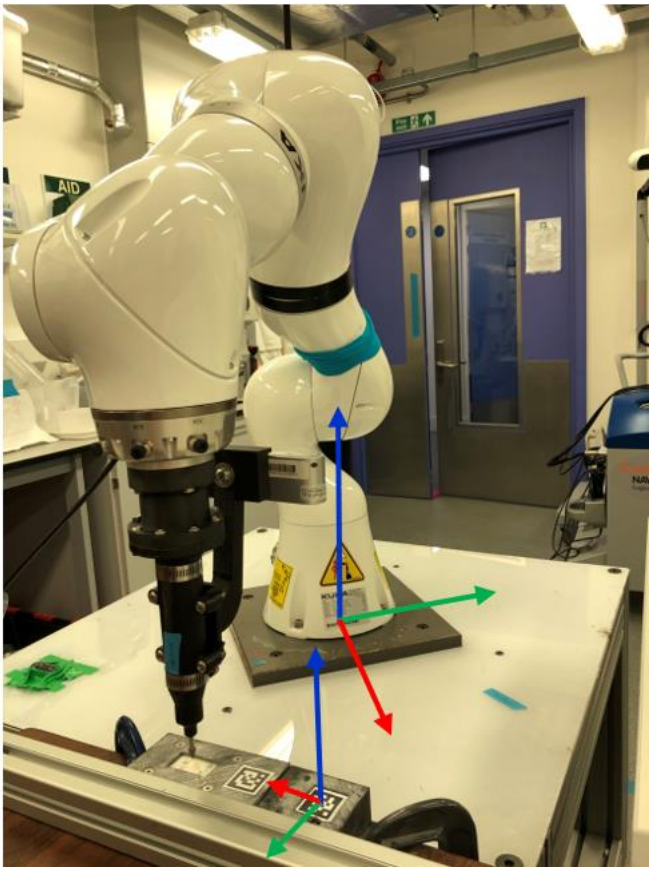


Figure A.1 Coordinate reference frame of the Kuka and of the fixed QR code

- **QR_Code_Divot -> QR_Code_Cut:**

  This matrix represents the position of the QR code placed on the block in which the material to be milled is embedded in the RF of the QR on the Divot block.

  It has been obtained in two different ways to compare the precision reached using the camera view or fixing the matrices as expected from the CAD models.

  1. **CAD models:**

     Here the needed matrix is obtained directly from the configuration of the cutting area and computed using the CAD models. As shown below, the expected matrix should be a simple translation along the X axis of the QR code on the Divot block.

$$T_{QRCode\_DIVOT}^{QRCode\_CUT} = \begin{bmatrix} 1 & 0 & 0 & 65 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

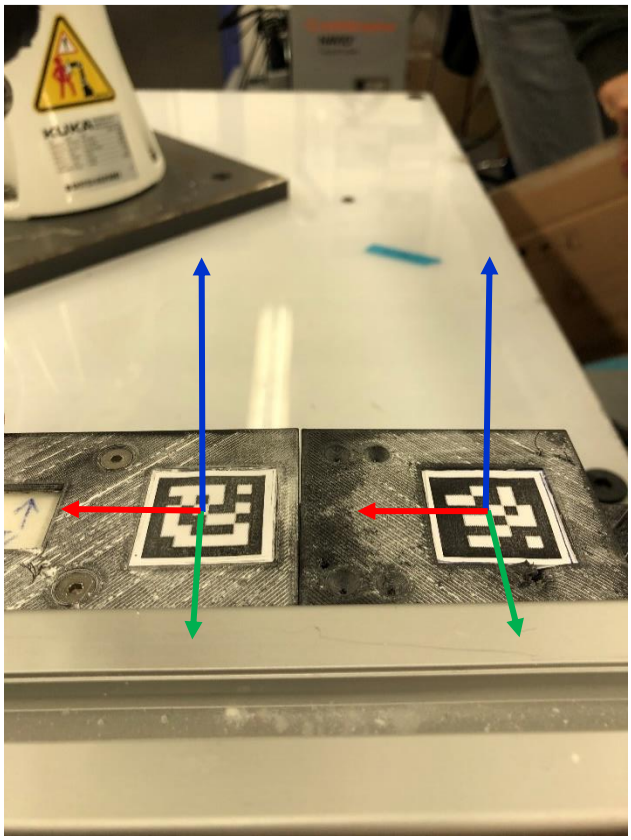


Figure A.2 Reference frame of the two QR codes

2. Camera calibration:

In this case, the camera, placed perfectly above the two QR codes, grab 10 different frames and, for each of them, computes the pose of both the QR codes using the aruco library of OpenCV (See Appendix D for more details).

Inside the RGB_reader.cpp code, the relation between this two RFs is computed and sent to the computation.cpp node to join the WS position computation process.

$$T_{QRCode\_DIVOT}^{QRCode\_CUT} = \begin{bmatrix} 1 & 0 & 0 & 56.96 \\ 0 & 1 & 0 & 15.09 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As it is possible to observe there is a small mismatch between the two matrices. The reason of it has been found in a systematic error since the Acusense camera is strongly influenced by the prospective. To deal with this issue has been decided to fix the calibration position of the Kuka and to measure the entity of this difference. What had been obtained is that the transformation matrix obtained using the Acusense camera data presents a rotation around the z axis of 4°. In order to solve this problem a rotation matrix has been implemented.

$$R\_axis = \begin{bmatrix} \cos(4) & \sin(4) & 0 & 0 \\ \sin(4) & \cos(4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note: This systematic correction is possible only if the relative position between the two QR code and the Kuka pose used to the calibration remain constant.

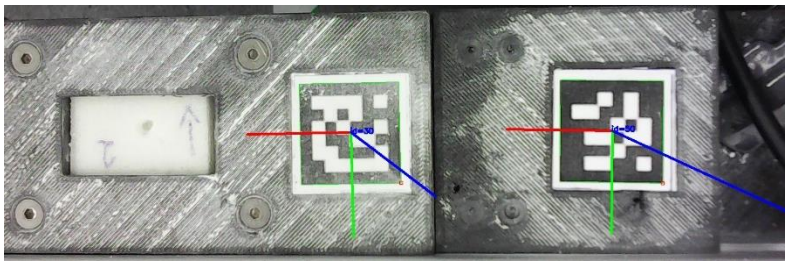


Figure A.3 Reference frame of the two QR codes and relative position of the material to cut (top view)

- **QR_Code_Cut -> WS:**

  It shows the position of the WS center in the RF of the QR placed on the block in which the material to be milled is trapped. This position has been obtained directly from the CAD model.

$$T_{QRCode\_CUT}^{WS\_center} = \begin{bmatrix} 1 & 0 & 0 & 51.92 \\ 0 & 1 & 0 & 0.05 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It shows that the desired position of the hole with respect the QR code must be related to an offset along X of 58.01 mm and an offset of 0.05 mm along Y.

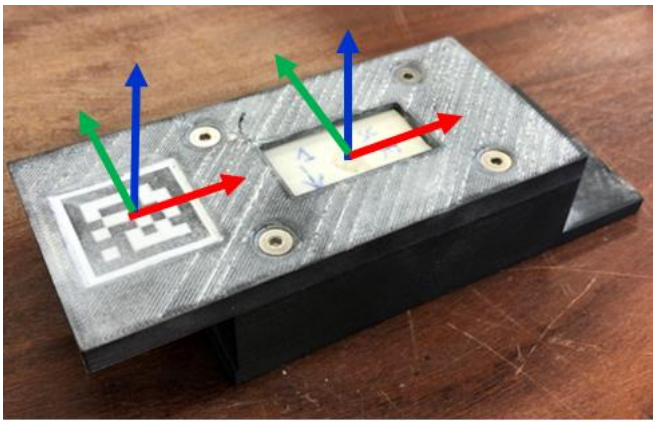


Figure A.4 Reference frame of the QR code and the hole on the working area hardware

All these matrices are then used to compute the position of the WS center in the Kuka's RF as specified in the following equation.

$$T_{KUKA}^{WS\_center} = T_{KUKA}^{QRCode\_DIVOT} \cdot R\_axis \cdot T_{QRCode\_DIVOT}^{QRCode\_CUT} \cdot T_{QRCode\_CUT}^{WS\_center}$$

- **Main loop:**

This category contains all the necessary reports to coordinate the movements of virtual drill in CHAI3D, Sigma.7 manipulator and Kuka's EE. These matrices, deeply analysed below are respectively used in the **catkin_ws/src/omni/src/omni_node.cpp** and **catkin_ws/src/kuka/src/kuka/listener.py** codes.

- **CHAI3D->Sigma.7:**

It represents the pose of the CHAI3D virtual drill in the Sigma.7 RF. It allows to synchronize the movement of the drill shown in the graphic OpenGL window with the manipulator of the haptic device.

$$T_{CHAI3D}^{HAPTIC} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

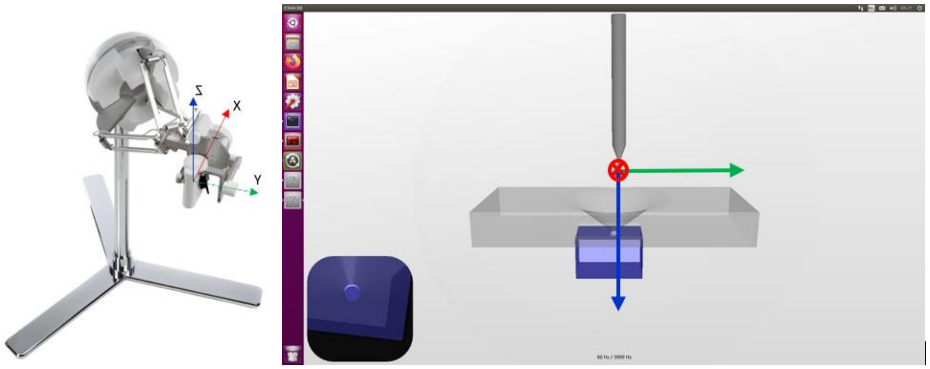


Figure A.5 Relation between CHAI3D and Sigma7 reference frame

- **Sigma.7->Kuka_EE:**

It is the homogeneous matrix that links and synchronizes the movement of the haptic device manipulator and of the Kuka EE. It is used in the **catkin_ws/src/kuka/src/kuka/listener.py** code to get the desired joints position through the IK.

$$T_{HAPTIC}^{KUKA\_EE} = \begin{bmatrix} -0.905 & 0.426 & 0 & 0.48011 \\ 0.426 & 0.905 & 0 & -0.24437 \\ 0 & 0 & -1 & 0.34355 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

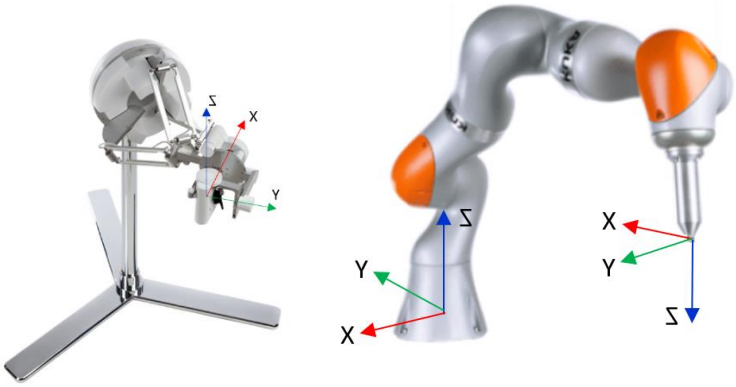


Figure A.6 Relation between Sigma 7 and KUKA reference frames

- **Kuka_EE->Tool_tip:**

  This is a relation that gives the position of the tip of the tool with respect to the Kuka's RF.

  Due to the cutter holder geometry, it is just represented by a vertical offset equal to the tool length, indeed. The length of the tool is specified in the **src/kuka/src/config/kuka_params_rt.yaml.** file**.**

$$T_{HAPTIC}^{KUKA\_EE} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 249.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure A.7 End effector - tool tip relation

To give the reader a better knowledge of how the sub-systems and devices are connected, a graphic representation is presented in figure A.8.

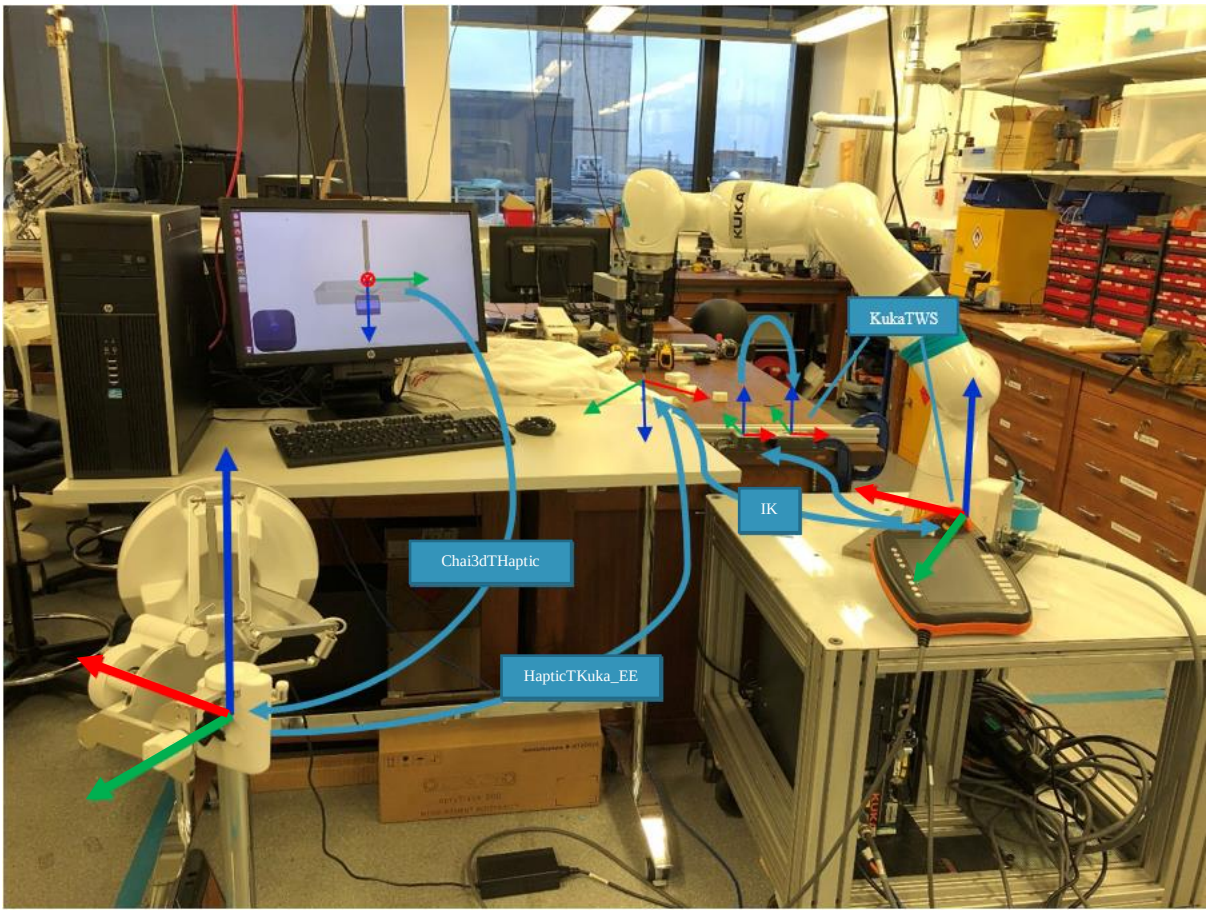


Figure A.8 Schematic representation of the geometrical transformation implemented

B. Kuka's home position computation

In this software, the home pose is always the idle position the robot reaches when the application kuka_test.py is started. The zero position is the position in which the arm is fully vertically stretched out (i.e., joints positions [0, 0, 0, 0, 0, 0, 0]).

Note that the home pose is not computed if the parameter **use_home_pos** is set to *true.* If it is equal to true, the system will take whatever home position is specified by the **home_pos** parameter.

The given home position must lead to a tool position that lies within the defined workspace; if it is not, a message to inform the user is displayed.

The workspace center parameter is then set to the cartesian tool home position.

If the **use_home_pos** parameter is set to *false*, the home position will be computed from the workspace definition and corresponding to the constraints listed below:

- Tool position must coincide with the workspace center plus a desired object of 15 mm to match the graphical representation on Chai3D and the tip position with respect to the cutting surface
- Tool z axis must point downwards
- Tool x axis must be aligned with the workspace center line and point in the Kuka's base direction
- All joints must lie in a plane spanned by the center line and the Kuka z axis.

In order to simplify the computation of the inverse kinematics, three joints (i.e., joint 3,5,7) are set to 0°. This allows to compute a joint position avoiding any possible issue derived by the redundancy of the Kuka's geometry and guarantee a considerable simplification; at the same time though, it greatly limits the possible configurations that the robotic arm can use to reach a specific target position or, in any case, lead to configurations not ideal for the application studied in this project.

To avoid problems due to peculiar joints configuration of the Kuka and excessively abrupt movements to reach them, which could hit operators in the vicinity of the co-robot or damage the latter due to collisions with surrounding objects, it was decided to use the previously explained procedure just to compute the starting joints configuration in order to use it in the designed Inverse Kinematics formulation explained in depth in Appendix C. Obviously, this configuration is not optimal, because definitely increases the computation volume; the choice to use this formulation is that the simplified inverse kinematics leads to a joints configuration that perfectly reflects the desired work laying for the application studied in this project.

In the following section, a detailed description of how the workspace is calculated if the **use_home_pos** parameter is set to *false,* is presented.

The workspace center computation is shown in figure where it is highlighted that:

$$z_{WS_{center}} = z_{lower_{limit}} + (z_{upper_{limit}} - z_{lower_{limit}})/2$$

$$Offset_{J2} = inner_{sphere_{limit}} + (outer_{spere_{limit}} - inner_{sphere_{limit}})/2$$

Where $Offset_{J2}$ determine the sphere radius centred in J2 on which the WS_center lies.

The function **getJointsHome.py** computes the home pose from the workspace center and the above constraints.

Joint 1's position is computed such that the rotation axis of joint 2 is perpendicular to the simplification plane in figure

B.2. The simplification plane is spanned by the specified center line and the Kuka z axis. The position of Joints 2 and 4 are calculated such that joint 6 lies above the WS_center. Joint 6 is set such that the tool's z axis face downwards.

The script provides an error message when the given workspace parameters would lead to a non-reachable home pose. A reason might be that the specified tool length is too big.

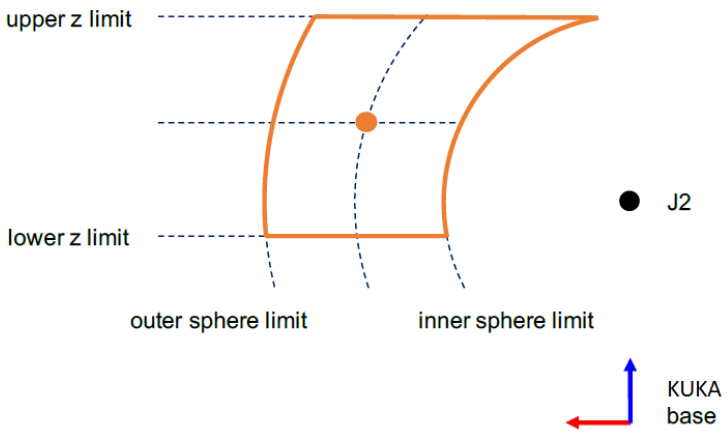


Figure B.1 KUKA's workspace geometry

Since it is possible to work in both sides of the Joint 1 range (± 170°), two cases are distinguished (figure B.2). If the center line points in the positive X direction (including [0, 1, 0] and [0, -1, 0]) the home position will resemble the left picture. If the center line position lies on the negative Kuka X direction, the home position will resemble the right illustration instead. Depending on which home pose configuration is initially computed (tool X axis points towards or away from base frame) the operator needs to publish the correct and desired tool frame transform to the system.

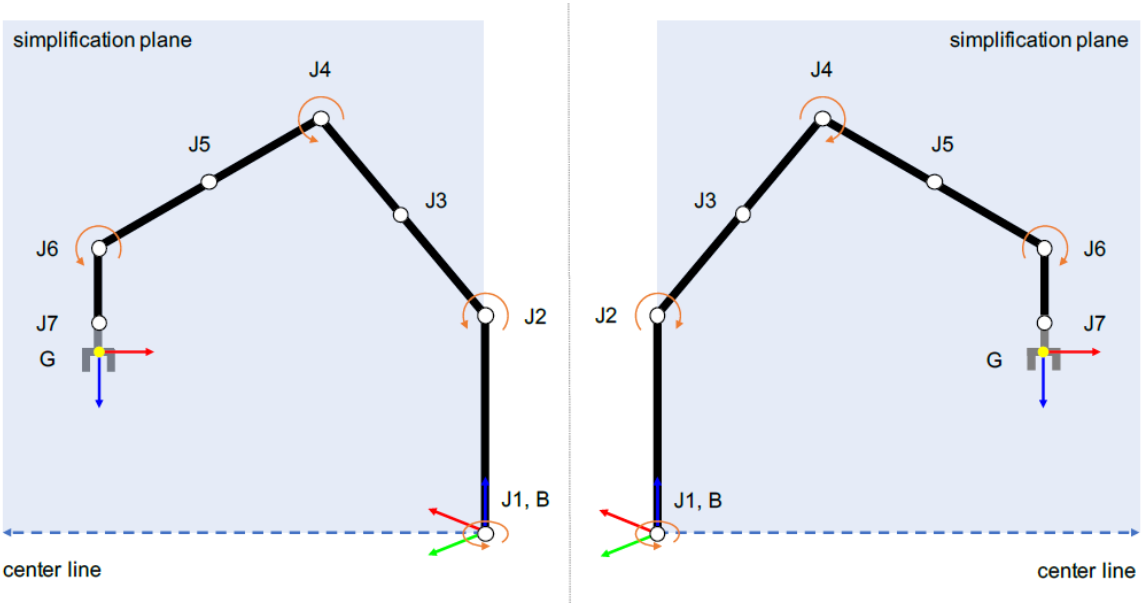


Figure B.2 Calculation of home position

C. Kuka control loop

This section provides the information of what happens in the main control loop for both the scripts **listener.py** and **kuka_test.py.**

For each script is provided the main goal (as refresh of what enounced in the introduction of chapter 3), and the main function used to reach it.

1. listener.py

The first part of the script is dedicated to upload all the information from the configuration file, compute the home position and the workspace, initialize the ROS node and the publisher that will send the desired Kuka's end-effector position to the **kuka_test.py** script. The script is subscribed to the ROS topic "Haptic_dev/pose" which send the pose of the Haptic device in its workspace and allow the transformation between the latter and the tool in the Kuka's workspace.

For this reason, a transformation matrix T is created; it is derived from 3 rotation matrices and a vector base explained below:

- *Vector Base:*
    - X_tool: set the x axis of the tool mounted on the Kuka in the same direction of the center line.
    - Z_tool: set the orientation of the z axis of the tool downwards.
    - Y_tool: complete the normalized right-handed tern.

- *R_centerline:*

  Use the previous defined tern to create a rotation matrix that connect the Haptic device Reference Frame and the Tool one.

$$R_{centerline} = [x_{tool}; y_{tool}; z_{tool}]$$

- *R_HK:*

  Provide the necessary information to obtain the desired link between the Haptic device movement and the Kuka end-effector. This is done so that the movement of the Kuka EE, the virtual drill of Chai3d and the Haptic device EE coincide.

$$R_{HK} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

  It is possible to observe that in order to obtain a perfect match between the three software, the Kuka and effector with respect to the Haptic device must move as follow:

  - $X_{Haptic} \rightarrow Z\_tool$
  - $Y_{Haptic} \rightarrow -Y\_tool$
  - $Z_{Haptic} \rightarrow X\_tool$

- *R_HK_pos:*

  It is the rotation matrix needed to obtain the relation between the Haptic device and the Tool position in the relative Reference Frames. It is obtained combining R_{HK} and $R_{centerline}$:

$$R_{HK_{pos}} = R_{HK} * R_{centerline}$$

Using this information, the pose of the Tool with respect to the Kuka's RF, is computed in a *while loop* in which the computed pose of the Tool is checked and eventually modified to remain inside the WS boundaries using the function **getConstrainedPosition.**

Finally, the desired and eventually corrected Tool pose is send through the ROS topic "/Pose4kuka".

2. kuka_test.py

The first part of this code is dedicated to upload all the information regarding the configuration file, hence the desired workspace, initialize the ROS node and the publisher and subscriber through which the robot can communicate with the Haptic device.

In this section, the connection with the robot is established too.

In the second part, the Kuka is moved in the home position using the function **movePTPJointSpace.** This allows the Kuka to synchronise its workspace center with both the haptic device and the Chai3D world. From now on, all the Kuka's movement will be related to the WS center of the haptic device and eventually scaled imposing the desired scaling factor.

The PTP motion uses a Java script in which, depending on the actual position of each joint, the shortest path to the target point is computed and followed; obviously, due to the Kuka geometry usually this path is not a straight line as shown in figure B.3.

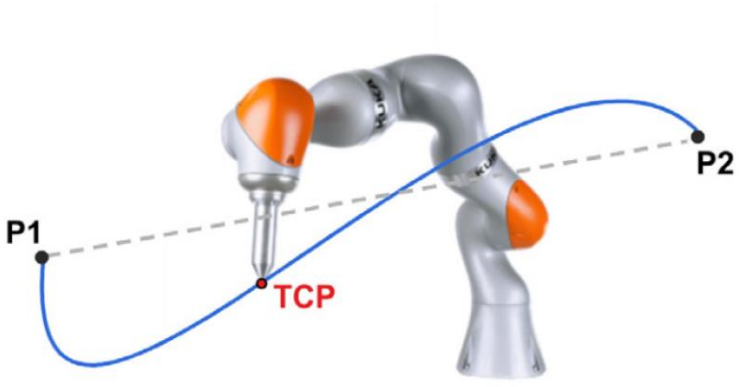


Figure C.1 Point to Point motion

Finally, the communication in Real Time between the code and the robot is started; it is executed in a while loop where the target position derived from the *listener.py* code is updated, and the joint position need to reach it is computed using the function *InverseKinematics.*
Due to Kuka's features, to avoid redundancy and singularity problem in the joints position computation, a local Inverse Kinematics computation has been implemented; it means basically that the computed joint position is strongly connected with the starting pose of the Kuka and can change every time. The *InverseKinematics* function, hence, takes as input three different values:

- A *homogeneous matrix* that defines the target pose.
- A vector that contains the actual joint position from which the function will compute locally the best configuration to reach the desired pose.
- A 6x1 vector named *w_task* that contains the information regarding which features the user wants to take into consideration during the computation.
  The first 3 elements regard the position of the Kuka's end-effector while the last three elements are referred to the orientation of the end-effector.
  The possible values are 0 (that means "don't take into account) or 1 (which means "take into consideration") so the user can basically choose if in his/her application both position and orientation are important or if the orientation can be neglected.

The large number of degrees of freedom (DOF) increases the Kuka versatility but brings the kinematic redundancy problem too. Redundancy means the ability of the robot to reach the same position in the space with its end-effector but in more than one joints configuration.
In order to create an optimization method to compute the inverse kinematics and solve the redundancy issue, the method of "Projection in the Null Space" was used.
Due to the redundancy, the dimension of the task "*m"* is less than the number of joints "*n"* (m<n), hence the Jacobian matrix $J(q) \in R^{mxn}$ is rectangular.
The kinematic control can be then written as:

$$\dot{\boldsymbol{q}} = \boldsymbol{J_{pinv}} * \boldsymbol{e} + \boldsymbol{N_J \lambda}$$

Where:

- $\dot{\boldsymbol{q}} =$ joint velocity to control
- $\boldsymbol{J_{pinv}} =$ Jacobian pseudo-inverse
- $\boldsymbol{e} =$ position error
- $\boldsymbol{N_J = I - (J_{pinv} * J)}$
    - Basic matrix of the null space of *J(q)*
    - $\boldsymbol{N_J} \in R^{nx(n-m)}$
    - $\boldsymbol{J * N_J = 0}$

- $\boldsymbol{\lambda} =$ value chosen ad-hoc to satisfy some desired performance.

When the application is shutdown, the Kuka returns in *home position*.

A schematic and intuitive description of the control loop is presented in the following algorithm.

D. Calibration process

During the first experiments a mismatch between the RF used from the depth camera and the one used from the RGB was detected, so a calibration process become necessary.
In the first instance, an attempt was made to find a transformation matrix that would link these two reference systems. The method chosen consisted in grabbing a series of images from which, using a 3D visualization software, the centre of the QR code in the depth camera reference system was estimated; this was compared to the same estimate in the RGB image provided by the aruco library. These two guesses were compared in a MATLAB file in which the homogeneous matrix of transformation between the two systems was subsequently evaluated. This method of estimation did not produce satisfactory results since the evaluation of the mutual distance of the two QR codes was strongly influenced by the camera's perspective. To manage this issue, it was decided to opt for a more static and systematic approach. It has been deeply described in the following section.

D.1. Fixed point calibration

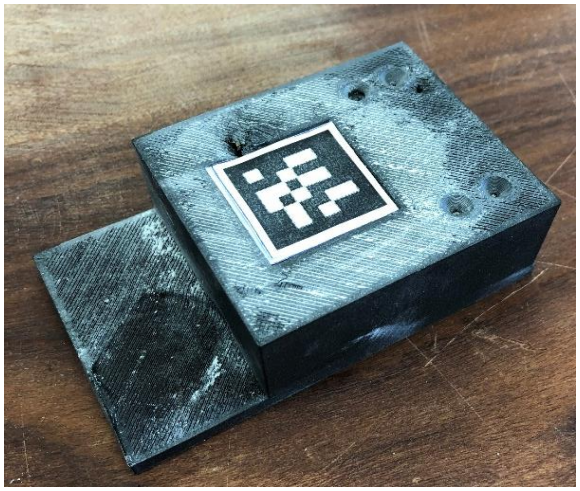


Figure D.1 Fixed point QR code and Divot calibration hardware

The fixed point is represented by the block in figure D.1. This is placed in a fixed position of the Kuka WS and represents a reference to calculate where the center of the work area is; this target position is identified by the second QR code and will be calibrated with the procedure explained in section D.2.

To identify the position of the center of the QR code with respect to the Kuka's RF, the following procedure has been followed:

- Using a clamp, the divot block shown in figure has been fixed in a position that lie inside the reachable working area of the iiwa robot.

- The Kuka has been brought in a home position that is close to the block with the orientation of the end effector always facing the negative z.

- The code omni_node.cpp, listener.py, and kuka_test.py have been run to teleoperate the robotic arm using the haptic device.

- In the end the tip of the tool has been placed in each divot (conic reversed shape) and the global position of the end-effector in the Kuka RF has been read from the smart Pad. In this way, two different matrices have been created:

    - The first containing the coordinate [X,Y,Z] of each divot whit respect to the Kuka RF.

$$Kuka\_Divot = \begin{bmatrix} 563.35 & -186.39 & 337.56 \\ 558.06 & -193.22 & 335.59 \\ 531.78 & -161.88 & 336.76 \\ 526.52 & -168.02 & 337.52 \end{bmatrix}$$

    - The second that contains the position of each divot in form [X,Y,Z] with reference to the center of the fixed QR code.

$$QRcode\_Divot = \begin{bmatrix} 20 & -25 & -2 \\ 20 & -35 & -2 \\ -20 & -25 & -2 \\ -20 & -35 & -2 \end{bmatrix}$$

The position of the same point in two different RF have been obtained so that the relation between these two can be computed using the MATLAB script *calibration.m*.

Here the function *estimateGeometricTransform3D* has been used to evaluate the rigid transformation between the two reference frames.

The output of this computation has been saved in the file `kukaTqr.csv` and the resulting estimated matrix (shown below) has been used in the file *computation.cpp* has explained in section 6.3.6.

$$T_{CUT}^{WS} = \begin{bmatrix} 0.7851 & 0.6194 & 0 & 563.51 \\ -0.6194 & 0.7851 & 0 & -153.825 \\ 0 & 0 & 1 & 88.857 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note: the element in third row and firth column will be modified in setupROS.py and increased of the tool length so that:

$$T_{CUT}^{WS} = 88.857 + 249.7 = 338.557\ mm$$

This matrix is imported and maintained unchanged for computation since its value changes only if the reciprocal position between the Kuka and the QR code is changed. In this case, the calibration procedure shall be repeated.

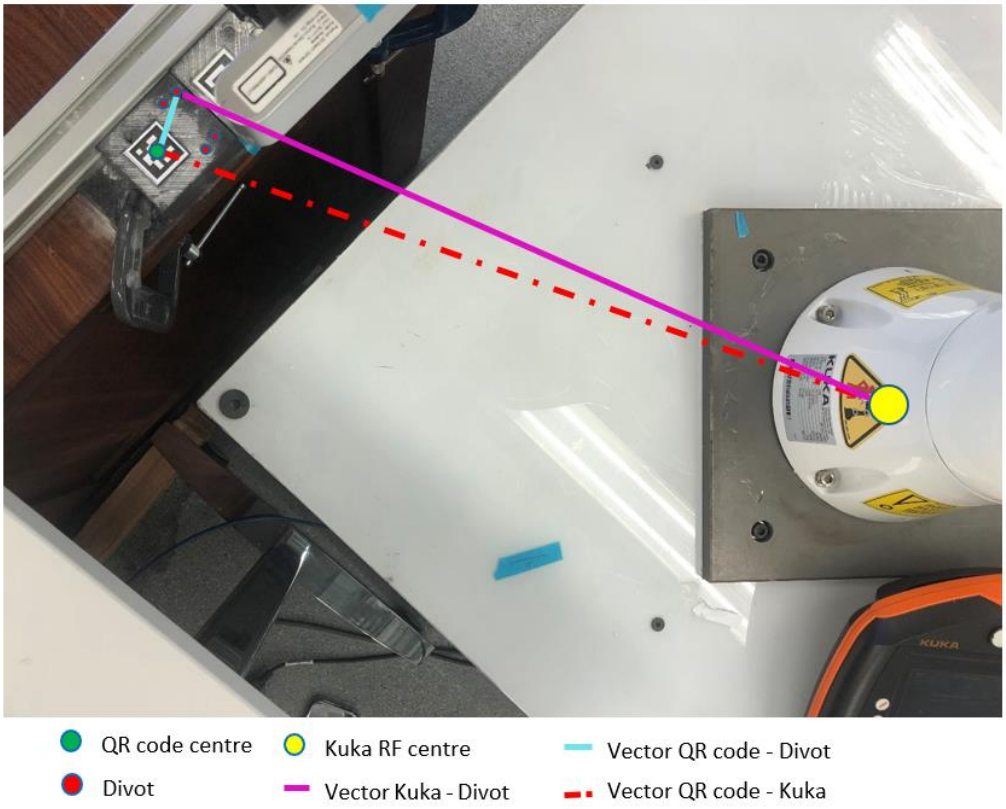


Figure D.2 KUKA base, QR code and Divot relation

D.2. Working area calibration

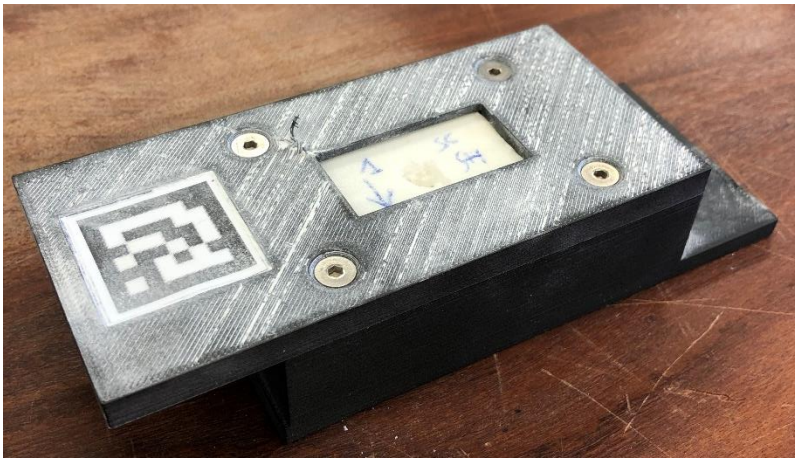


Figure D.3 Working area block

Once the position of the fixed QR code is known, the calibration to estimate the working area position has been implemented. The goal of this calibration is to evaluate the relative distance between the QR code shown in figure D.1 and the one represented in figure D.3, to move the robotic arm over the target position of the hole.

To evaluate this relation, the following steps have been followed:

- The Kuka is moved in a position in which the camera can perfectly see both the QR code as vertically as possible to reduce the prospective disturbances. It can be made using the code:
  /home/matteo/catkin_ws/src/kuka/src/kuka/calibration.py in which the user can impose a target/desired position of the Kuka's end-effector in millimetres; the orientation of the tip of the tool always facing the negative z.

- The camera is started running the code camera_node.cpp.
  From now on the grabbing of the images is started and the information are sent on the dedicated ROS topic.

- The file RGB_reader.cpp is run; in this way, the identification and estimation of the QR code begin. Here the relationship between the two QR codes is computed and sent to computation.cpp.
  To keep the estimation easier and guarantee a good accuracy of the estimation maintaining the camera not too far from the detection area, the two QR codes have been located as shown in figure D.4.

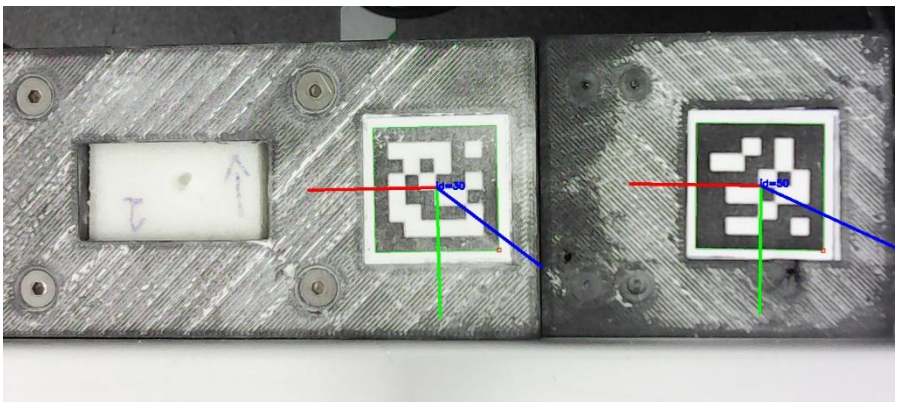


Figure D.4 QR code calibration (top view)

The resulting matrix of this configuration is presented below.

$$T_{DIVOT}^{CUT} = \begin{bmatrix} 1 & 0 & 0 & 56.96 \\ 0 & 1 & 0 & 15.09 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To get the position of the hole with respect to the QR code on the block shown in figure D.5, it has been obtained directly from the 3D assembly and imposed in the computation.cpp file as the following homogeneous matrix.

$$T_{CUT}^{WS} = \begin{bmatrix} 1 & 0 & 0 & 58.1 \\ 0 & 1 & 0 & 0.05 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

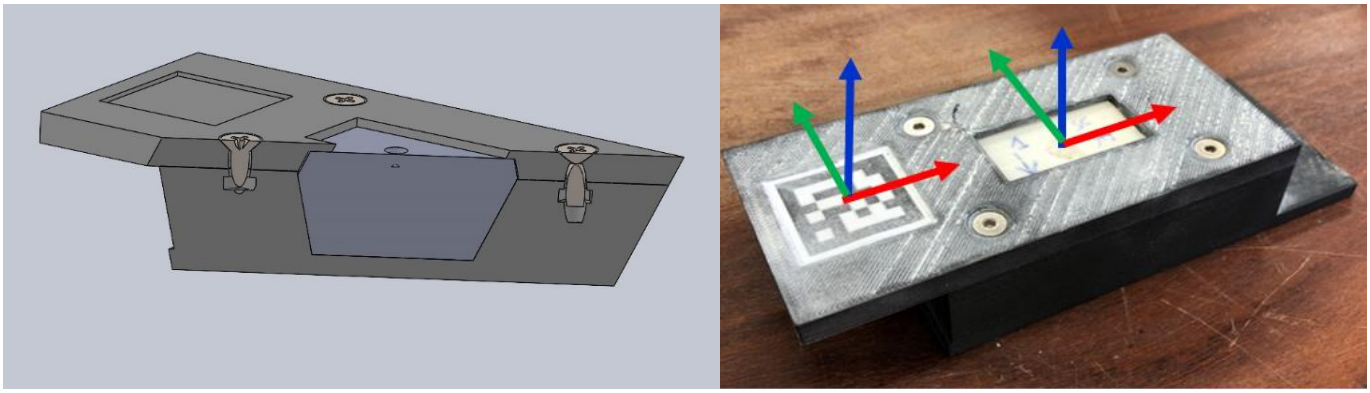


Figure D.5 Working area block: Solidworks model (left), QR code - Hole reference frame relation (right)

E. Image type overview

RGB image

An RGB image has three channels: red, green, and blue. RGB channels roughly follow the colour receptors in the human eye and are used in computer displays and image scanners.

If the RGB image is 24-bit, each channel has 8 bits, for red, green, and blue. In other words, the image is composed of three images (one for each channel), where each image can store discrete pixels with conventional brightness intensities between 0 and 255. If the RGB image is 48-bit (very high colour-depth), each channel is made of 16-bit images.

Figure E.1 RGB image

Depth maps

In 3D computer graphics and computer vision, a depth map is an image or image channel that contains information relating to the distance of the surfaces of scene objects from a viewpoint. Depth maps have several uses, including:

- To provide the distance information needed to create and generate auto stereograms and in other related applications intended to create the illusion of 3D viewing through stereoscopy.
- Subsurface scattering - can be used as part of a process for adding realism by simulating the semi-transparent properties of translucent materials such as human skin.

- In computer vision, single-view or multi-view images depth maps, or other types of images, are used to model 3D shapes or reconstruct them. Depth maps can be generated by 3D scanners or reconstructed from multiple images.
- In Machine vision and computer vision, to allow 3D images to be processed by 2D image tools.
- Etc.

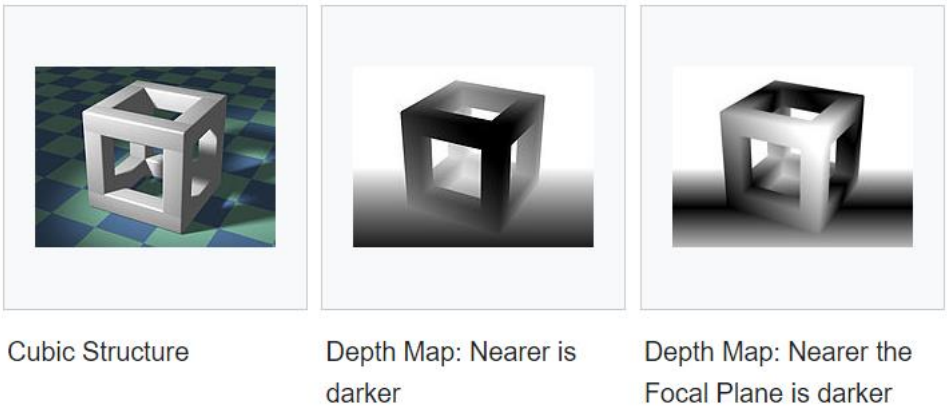


Figure E.2 Depth map

Two different depth maps can be seen in figure, together with the original model from which they are derived. The first depth map shows luminance in proportion to the distance from the camera. Nearer surfaces are darker; further surfaces are lighter. The second depth map shows luminance in relation to the distances from a nominal focal plane. Surfaces closer to the focal plane are darker; surfaces further from the focal plane are lighter.

Point clouds

A point cloud is a set of data points in space. The points may represent a 3D shape or object. Each point position has its set of Cartesian coordinates (X, Y, Z).

Point clouds are generally produced by 3D scanners or by photogrammetry software, which measures many points on the external surfaces of objects around them. As the output of 3D scanning processes, point clouds are used for many purposes, including to create 3D CAD models for manufactured parts, metrology and quality inspection, and a multitude of visualization, animation, rendering, and mass customization applications.

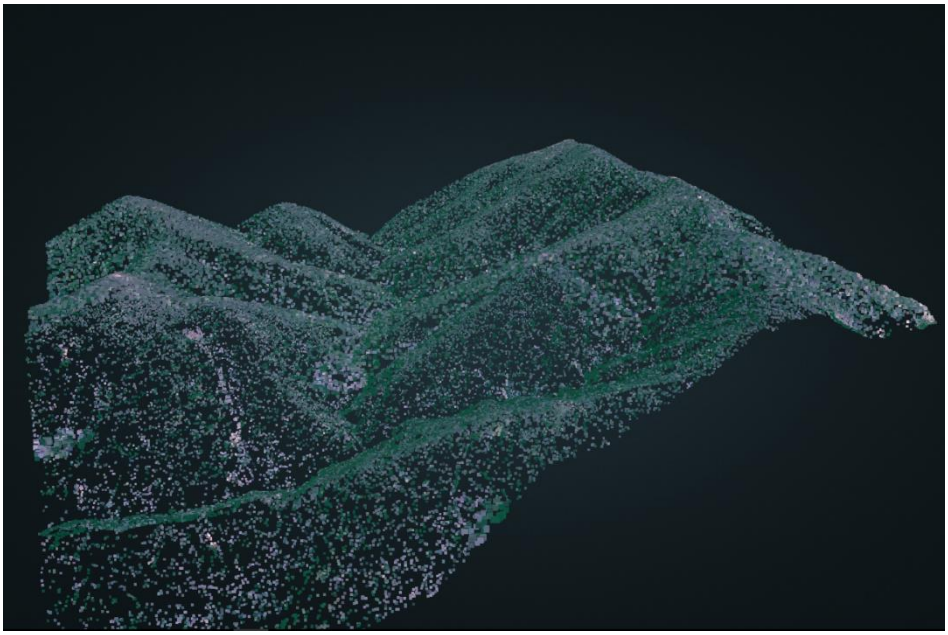


Figure E.3 Point cloud

Bibliography

[1] Robotic technology in surgery: past, present, and future. David B. Camarillo, M.S., Thomas M. Krummel, M.D., J. Kenneth Salisbury, Jr., Ph.D.

[2] [Computer assiste d orthopae dic surgery: Past, present, and future. Frederic Picard, AngelaHelen Deakin, PhilipE. Riches, Kamal Deep, Joseph Baines.

[3] From Camarillo DB, Krummel TM, Salisbury JK Jr. Robotic technology in surgery: past, present, and future. Am J Surg. 2004 Oct;188(4A Suppl):2S-15S. doi: 10.1016/j.amjsurg.2004.08.025. PMID: 15476646.

[4] Chen, Antonia F. MD, MBA1; Kazarian, Gregory S. BA2; Jessop, Galen W.2; Makhdom, Asim MD, MSc, FRCSC2

[5] Taylor RH. Robots as surgical assistants: where we are, wither we are tending, and how to get there. Lecture Notes in Artificial Intelligence.1997;1211:3–11

[6] Davies B. A review of robotics in surgery. Proc Inst Mech Eng[H] 2000;214:129–40

[7] Jakopec M, Harris SJ, Rodriguez y Baena F, Gomes P, Cobb J,Davies BL. The first clinical application of a “hands-on” robotic knee surgery system. Comput Aided Surg 2001;6:829–39.

[8] https://markforged.com/3d-printers/mark-two

[9] https://github.com/Modi1987/iiwaPy3

[10] https://www.chai3d.org/download/doc/html/wrapper.html

[11] https://3dcamera.revopoint3d.com/html/acusense/index.html

[12] https://en.wikipedia.org/wiki/RGB_color_model

List of Figure

List of Tables