Politecnico di Torino
Master's degree course in Mechatronic Engineering
Master's Degree Thesis

Integration of anthropomorphic robots with high precision instrumentation

Candidate
Pietro Castelli

Supervisor
Prof. Marina Indri

Supervisor at
Comau S.p.A.
Ing. Stefano Pesce

October 2019

«Il bimbo ristette, lo sguardo era triste,
e gli occhi guardavano cose mai viste
e poi disse al vecchio con voce sognante:
"Mi piaccion le fiabe, raccontane altre!"»

F.Guccini, *Il vecchio e il bambino*

A mio nonno Adolfo,
sei stato la scintilla che ha acceso in me il fuoco della curiosità.

Contents

Chapter 1

Introduction

Alongside with the spread of robotic applications that is seen nowadays[**1**], tools for teaching the main concepts related to robotics are needed to familiarize new generation of students with this branch of engineering. Industrial robots are not well suited for demonstrational and teaching use due to their complexity, cost and lack of a friendly interface. To this purpose a new category of robots is being studied and designed by companies: educational robots.
Educational robots are low cost and low performance robots, meant to be used for teaching technology and related skills such as Computational thinking [**2**].
The context in which educational robotics is used fixes some design principles for this category.
For what concerns the mechanical structure, an educational robot should have a simple and representative one. Usually when dealing with mobile robotics we see wheeled robots being adopted. When it comes to manipulator instead, anthropomorphic arm with spherical wrist is often designed since in literature they are mostly used as example for the serial link manipulator category.
When designing the software for control of such robots it must be kept in mind that open and simple solutions are preferable for teaching purposes. The code should be open in the sense that students may be interested in reading, studying and modifying it, then open source solutions are preferable.
The development of high-level interfaces, that relieves the student from the knowledge of coding, is also of use when dealing with less experienced users. When designing such systems one should try to make them as much as possible likely to be expanded. Software architectures should provide solid basis, implementing the lower and most complex algorithm, from which the student

can start the design of a higher-level behavior.
In this context Comau s.p.a invested in the development of the "e.Do" project.
Comau is a company founded in Turin in 1973. Born as COnsorzio MAchine Utensili to merge all companies providing industrial supplies to build Togliattigrad VAZ plant in Russia [**3**], today it is global leader in the field of industry automation with 32 operation centers in 14 countries[**4**].
The development of the e.Do project it is of interest for the company for participating also in creation of educational and personal automation solutions. Besides commercializing the physical robot, the company provides a cloud containing resources for the user. Thanks to "eDo.cloud" the user is supported in developing projects using their robots. Tutorials, news and showcases are available together with a forum administrated by Comau staff to help user share knowledge and solve technical problems. The hardware and software architecture is open source and ROS based, so a virtual machine and the code repositories are available online. To facilitate the interaction with the user an android and web application has been developed.
This project aims at studying the capability of the robot when used outside of a teaching context. When inside an academic robotic lab such a robot is of particular interest for the study of robotic's algorithms.
The ROS architecture running on e.Do, developed by Comau as a ROS package named *edo_core_pkg* , is a solid base from which to start a complex application.
For this specific case we are interested in investigating the capabilities of e.Do to operate a pick and place task in a "multiple-system" environment.
A new ROS package to implement an interface between the robot and a digital microscope had to be designed. The digital microscope involved in the project is called mesoSPIM, currently the microscope has been installed in seven different locations. Among those we find a setup in the Neuropathology Institute of Universitat Spital of Zurich, the same institute bought e.Do and asked Comau for a jointed reasearch on the interaction between the microscope and the robot.
The objective of the interface was to implement the behavior needed for performing a series of analysis on medical samples. Such samples, stored in a holder, must be grasped and delivered to the microscope for the analysis. Once the screening is completed the samples are put back in place. These actions are then repeated on a series of different samples.
Automation of this process relieves the researcher from the manual load and unload of the samples, annoying in case of long duration analysis on a series of different samples.
The package developed extends the functionalities of *edo_core_pkg*. I have

been working along with the e.Do software team to better understand the path this project is following and where its future development will lead.
A functional collaboration helped to highlight the feature that an expert user, as an academic researcher, looks for when working with a robot of this type.
The use of a ROS environment for manipulators is little investigated by the community, unlike mobile robotics; furthermore pick and place task are usually carried out by industrial robots designed *ad-hoc* for the specific task. For these reasons this work finds little references to existing literature and relays more on practical aspects of the specific case.
In the next chapters the work done during the collaboration is presented.
Conceptually two aspects have been studied. First e.Do has been studied deeply in its code architecture and *software-hardware* structure. Typical robotics analysis, such as direct kinematics, Jacobian and so on are presented. All of this has been carried out to better understand the e.Do system, the way it works and communicates.
Then in the second part a ROS package has been developed to accomplish the pick and place task.
In Chapter 1 all the studies done on e.Do are presented. Both its software and hardware structure are explained. Kinematics features of e.DO are investigated and a representation of the workspace is discussed. The ROS *edo_core_pkg* architecture is also described.
In Chapter 2 we talk about the microscope, its features are represented and a method to overcome its physical absence it is presented.
In Chapter 3 the development of the new ros package is illustrated. Constraints and choices that motivate the project evolution are discussed.
In Appendix A some key concepts about Robotic Operating System are synthetized. In Appendix B instead some basic robotics concepts are illustrated. In Appendix C we present numerical results for direct kinematics function, Jacobian and Workspace characterization.
In the final chapter we illustrate future developments that could start from this work of thesis. Some possible improvements to the code structure are also discussed.

Chapter 2

e.Do

e.Do is the name of the educational robot developed by Comau S.p.A. The robot belongs to the category of manipulators. More specifically the robot is an anthropomorphic arm with spherical wrist. The length of the arm extended is 0.99 metres and its weight is 11.1 kg.
e.Do can lift up to one kilo of payload and deliver a torque of 4 Nm, making the robot suitable for small pick and place tasks; more detailed techincal specifications are reported in Fig. 2.1 [**5**].

Specifications		Value
Number of axis		6
Max payload		1 Kg
Max reach		478 mm
Stroke (Speed)	Axis 1	+/- 180 ° (38 °/sec)
	Axis 2	+/- 113 ° (38 °/sec)
	Axis 3	+/- 113 ° (38 °/sec)
	Axis 4	+/- 180 ° (56 °/sec)
	Axis 5	+/- 104 ° (56 °/sec)
	Axis 6	+/- 2700 ° (56 °/sec)
Total weight		11,1 kg
Robot arm weight		5,4 kg
Structure material		Ixef 1022
Power source		Universal external power source with 12V power adapter
Connectivity		1 external USB port 1 RJ45 Ethernet 1 DSub-9 Serial Port
Motherboard		Raspberry Pi running Raspbian Jessie
ROS		Kinetic Kame
Control Logic		Proprietary open-source **e.DO**
Additional Features		External emergency stop button

Figure 2.1: e.Do techincal specification provided by Comau

2.1 Hardware analysis

The joints are actuated by six electric motors. The three motors located at the base have bigger dimensions and can bare more weight, they can rotate up to 38deg/s and deliver a static torque of 17.9 Nm. The three motors used for the spherical wrist are instead smaller and capable of less torque, they rotate at 56 deg/s and produce a static torque of 2.75 Nm.
These motors are driven by Novalabs ServoDrive boards placed on top of them. Motors are also equipped with an encoder and hall sensor to feedback their position. The boards are then connected via CAN system. A PCB finally bridges the connection from CAN to Serial to a Raspberry Pi that serves as main control unit for high level algorithms.
e.Do comes with no end effector, although it is possible to attach one to it. For this work of thesis a robot gripper with two fingers has been used, fixed by means of screws and connected to the CAN network just as one of the other motors.

2.2 Software analysis

The Raspberry Pi with which e.Do is equipped comes with the Raspian Jesse OS, a Debian-based operating system *ad-hoc* for Raspberry Pi.
Installed on the operating system there is ROS Kinetic. ROS Kinetic is a distribution of ROS released in may 23 2016, since it's been released more than three years ago , Kinetic is now well supported and stable compared to successive releases.
It is in the ROS infrastructure of e.Do that we find the higher level control algorithms. Such algorithms are implemented by means of ROS nodes and are contained in the ROS package *edo_core_pkg* .
An outline of the e.Do comunication network si presented in Fig. 2.2.
The *edo_core_pkg* contains code written in C++ to implement an Algorithm Manager, a State Machine and to handle communication from/to the motors and from/to the user.
Lower level algorithms, such as direct, inverse kinematics and dynamic model of the motors are contained in a private library named ORL. The functions of ORL used by *edo_core_pkg* are not accessible by the user and are property of Comau.

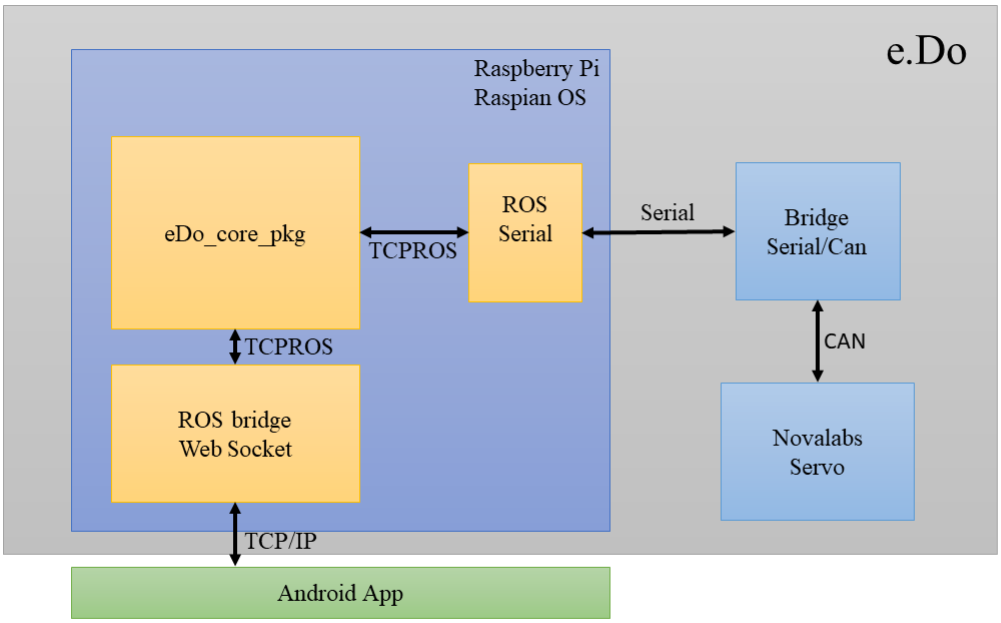

Figure 2.2: Scheme of e.Do system configuration.

2.3 e.Do core package

Comau developed a specific package for implementing a basic software infrastructure to control e.Do.
The package contains nodes to implement an interface towards the lower level control.
This package allows also to command the robot using the android application.

2.3.1 Algorith Manager

The Algorithm Manager node is the node that implements the highest behavior of the robot system, it offers the following functionalities :

- Calibration:
  When powered on e.DO needs to calibrate, which is the procedure to set the zero of the encoders that when off lose their reference. To calibrate e.Do, Comau provides an user interface with an android app. The user must connect an android device to e.Do, by means of wifi *ad-hoc*

mode connection, and perform calibration. This feature is implemented through Algorithm Manager methods.

- Feedback signal:
  Position, velocity and current values are fedback by the motors. These datas are collected by AlgorithmManager and passed to ORL functions.

- User Commands:
  Messages sent by the user to move e.Do have a specific structure, defined by Comau, and different from the ROS standard.
  User messages are interpreted by the AlgorithmManager node and fed to the ORL functions.
  To send messages to e.Do the user must publish on topic *bridge_move* a message of the type MoveCommand whose structure is showed in Fig. 2.3 and Fig. 2.4, the message must be filled with the following fields:

Topic : /bridge_move	
msg type : MoveCommand	
Type	Field
uint8	move_command
uint8	move_type
uint8	override
uint8	delay
uint8	remote_tool
uint8	cartesian linear speed
Point	target
Point	via
Frame	tool
Frame	frame

Figure 2.3: Structure of MoveCommand defined in *edo_core_msgs*

  - Command Type:
    It is the type of command we are about to send; it can be
    - MOVE: Message sent will be a request of movement between current position and target position.
    - JOG: Message sent will increment gradually one of the joint variables.
    - CANCEL: Cancel execution of current action.
    - PAUSE: Pause execution of current action.
  - Move Type:
    In case we are sending a MOVE command we must specify what

Type	Field
msg type: Point	
uint8	data_type
CartesianPose	cartesian_data
uint64	joint_mask
float32[ ]	joints_data
msg type: CartesianPose	
float32[ ]	[x, y, z, a, e, r]
string	config_flag
msg type: Frame	
float32[ ]	[x, y, z, a, e, r]

Figure 2.4: Structure of Point, CartesianPose and Frame types defined in *edo_core_msgs*

type of move we want to perform.
At the time the project was developed there were two options. The first one is a move of the JOINT type; it means that we are performing a linear move in the joint space, that results in a non-linear motion of the end-effector.
A second option was to send a move of the LINEAR type, this mode creates a linear motion in the cartesian space, that results in a motion of the end effector along a straight line.
In newer realeses of *edo_core_pkg* a new move has been implemented, the CIRCULAR.

- Data Type:
  When sending a move message we must specify the type of data format we will send in the message.
  The options are two, whether we send a message containing the target joint angolar positions, JOINT data type, or we send the cartesian coordinate of the point we want to reach, CARTESIAN data type.

- Joint mask:
  If we want only specific joints to move, and the others to stay still, we can encode this information in a mask.
  For each joint we specify 1 if we want it to move, else 0. Composing these binary values, ordering them from the lower motor as LSB until the end-effector MSB we obtain a binary number that is then converted in integer and inserted in the dedicated field of the message to send.

The possible values ranges from 1 to 127.
For example, if we want only the end-effector gripper to move, we have the following mask : 1000000, converting this binary number to an integer we obtain 64. On the opposite if we want to move all joints except the end-effector the mask is 0111111, that is 63.

- Target Position:
  Once specified the data type of the position information we can specify by means of a vector of float numbers the target joint or cartesian positions.
  To close or open the end-effector the method is different. There is a unique way, for both the data types, to specify its position which is the opening width in mm, or -1 to ask for a closing untill a considerable reaction force it is sensed, that means we grasped a solid object.
- Via Point:
  Even tough at the time of the project development via points where not implemented yet, in the last releases it is possible to specify, with the same data type of the target position, a via point for our motion.
  The ORL library will generate a motion from starting point to target point with a "fly-by" motion in proximity of the via point
- Override Velocity:
  Represented as a percentage, it is the velocity at which the joints will turn. With 100 we want the motors to spin at their maximum velocity.
  It's suggested, especially when precision is needed, to take this value between 20 and 50.

Upon reception of a MOVE command the robot moves to the target position, the trajectory planning algorithms belong to the ORL library.

- State Machine comunication:
  Algorithm Manager communicates with the StateMachine node to take actions accordingly to the status of the machine state
- Error Handling:
  Algorithm Manager implements methods for the notifications of error and, when possible, initializes the error recovery procedures.

2.3.2 State Machine

edo_core_package comes with a State Machine implemented as a ROS node. The possible states are the following:

- INIT
- NOT CALIBRATE
- CALIBRATE
- MOVE
- JOG
- ERROR
- BRAKED
- COMMAND

When the robot is powered on the initial state is the INIT state. When we access the calibration procedure the state changes to the NOT CALIBRATE state. As soon as the calibration is completed the state changes to the CALIBRATE state. In this state the robot is ready to accept commands. Upon reception of a message the state changes to COMMAND. When the message has been received, depending on its type, the status of the machine changes;

- MOVE, if the message type received was MOVE
- JOG, if the message type received was JOG
- Whenever e.Do faces an internal error the status changes to MACHINE ERROR.
- In case of critical fault, like collisions, the motors are powered off and the status changes to BRAKED.

The StateMachine node subscribes to a topic named *bridge_move*, that is the topic on which the user must publish the move messages. When receiving a message on *bridge_move* the State Machines changes its status and notifies the Algorithm Manager node about the new message. The content of the message is forwarded through the topic *machine_move*. To communicate its current status to the Algorithm Manager the topic *machine_state* is used. This topic is periodically written, the states are encoded as integer numbers according to Table 2.1 . When in ERROR the message of the machine state

State	enumerate
COMMAND_STATE	255
INIT	0
NOT_CALIBRATE	1
CALIBRATE	2
MOVE	3
JOG	4
MACHINE_ERROR	5
BRAKED	6

Table 2.1: State Machine enumerate states

Operational Code	enumerate
NACK	0
JOINT_ABSENT	1
JOINT_OVERCURRENT	2
JOINT_UNCALIBRATED	3
POSITION_ERROR	4
ROSSERIAL_ERROR	5
BRAKE_ACTIVE	6
EMERGENCY_STOP	7
FENCE	8
COLLISION_ON	9

Table 2.2: State Machine enumerate operational codes

is extended with an additional field named operational code.
As shown in table 2.2 the operational code gives details about the nature of the error.

2.3.3 e.Do Recovery

A node it is implemented to allow recordings of feedback data during the execution of moves.
Upon reception of a specific message this node records position, velocity and current measured from the robot motors. These datas are then saved in a txt file located in the local memory storage of e.Do's Raspberry Pi.

During the project development, COMAU provided protected Matlab functions to load the data in Matlab, this has been useful for the validation of the correct execution of trajectories.

2.3.4 ROS Serial

Serial comunication towards the board that bridges the connection to the CAN network has been implemented with ROSSerial metapackage. ROS serial is a protocol to wrap ROS messages and send them over a serial network. Among the various solutions offered by ROSSerial meta package we find implemented the package *ROSSerial_server*. *ROSSerial_server* offers the implementation from the host-side of a serial communication, handling autonomously setup, publishing and subscribing to a rosserial-enabled device. The package offers different nodes accordingly to the number of clients that need the connection, in case of multiple clients rosserial socket is needed[**6**]. Since in this application the client is one, that is the serial-can bridging board, the rosserial node is used, designed for single-client serial connection.

2.4 Study of Direct kinematics

e.Do is an open chain manipulator, composed of an anthropomorphic arm with a spherical wrist attached on it as shown in Fig. 2.5

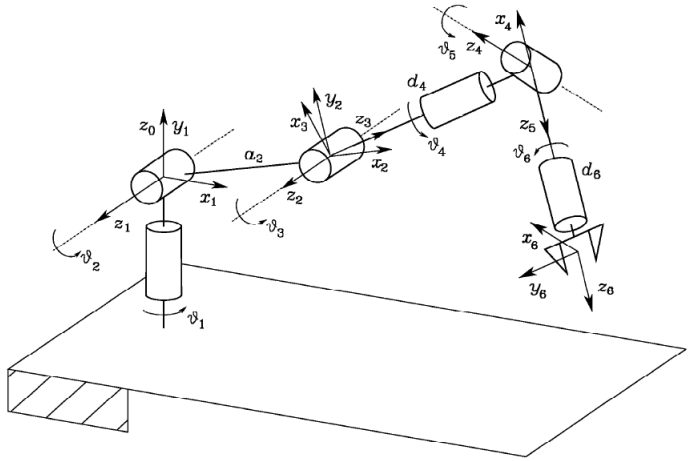

Figure 2.5: Antropomorphic arm with spherical wrist,[**7**]

For this section the presence of the seventh joint which actuates the gripper is neglected and will be discussed later on.

n	d_i [m]	a_i [m]	α_i [rad]	θ_i
1	0.337	0	$-\frac{\pi}{2}$	q_1
2	0	0.210	0	q_2
3	0	0	$-\frac{\pi}{2}$	q_3
4	0.268	0	$\frac{\pi}{2}$	q_4
5	0	0	$-\frac{\pi}{2}$	q_5
6	0.174	0	0	q_6

Table 2.3: e.Do's Denavit–Hartenberg parametrs

Instead the end effector reference frame will coincide with that one of the sixth link. The direct kinematics for open-chain structures can be obtained as a compositions of homogeneous transformation between two consecutive links [**7**].
Using the DH parameters, shown in Table 2.3, we can find the homogeneous transformations between each one of the joint's reference frames.
Composing the transformation we obtain the direct kinematics function for the end effector position and orientation.

$$T_6^0 = \prod_{i=1}^{6} T_i^{i-1}(q_i)$$

For the arm we find:

$$T_1^0 = \begin{bmatrix} cos(q_1) & 0 & -sin(q_1) & a_1 \cdot cos(q_1) \\ sin(q_1) & 0 & cos(q_1) & a_1 \cdot sin(q_1) \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$T_2^1 = \begin{bmatrix} cos(q_2) & -sin(q_2) & 0 & a_2 \cdot cos(q_2) \\ sin(q_2) & cos(q_2) & 0 & a_2 \cdot sin(q_2) \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$T_3^2 = \begin{bmatrix} cos(q_3) & 0 & -sin(q_3) & a_3 \cdot cos(q_3) \\ sin(q_3) & 0 & cos(q_3) & a_3 \cdot sin(q_3) \\ 0 & -1 & 0 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

[1]See Appendix C for T_3^0 numerical result

Composing transofrmations (2.1), (2.2) and (2.3) and substituting the values of DH parameters we obtain the following matrix[1]:

$$T_3^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \tag{2.4}$$

Equation 2.4 is the transformation that expresses the position and orientation of *arm-end* with respect to the base reference frame.
Applying the same method for the wrist we obtain:

$$T_4^3 = \begin{bmatrix} cos(q_4) & 0 & sin(q_4) & a_4 \cdot cos(q_4) \\ sin(q_4) & 0 & -cos(q_4) & a_4 \cdot sin(q_4) \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$T_5^4 = \begin{bmatrix} cos(q_5) & 0 & -sin(q_5) & a_5 \cdot cos(q_5) \\ sin(q_5) & 0 & cos(q_5) & a_5 \cdot sin(q_5) \\ 0 & -1 & 0 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$T_6^5 = \begin{bmatrix} cos(q_6) & -sin(q_6) & 0 & a_6 \cdot cos(q_6) \\ sin(q_6) & cos(q_6) & 0 & a_6 \cdot sin(q_6) \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Composing (2.5), (2.6), (2.7) and substituting the values of a_i and d_i for $i = 4, 5, 6$ we obtain the following matrix[2] :

$$T_6^3 = T_4^3 \cdot T_5^4 \cdot T_6^5 \tag{2.8}$$

that is the homogeneous transformation matrix for the wrist part.

The arm transformation is then composed with the wrist one as follows:

$$T_6^0 = T_{arm} \cdot T_{wrist} = T_3^0 \cdot T_6^3 \tag{2.9}$$

We find the change of representation matrix from the 6th reference frame to the base one, substituting the values of d_i and a_i, the origin of end-effector reference frame and the components of its versors are found[3]:

$$T_6^0 = \begin{bmatrix} \mathbf{n}^0 & \mathbf{s}^0 & \mathbf{a}^0 & \mathbf{p}^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

[2]See Appendix C for T_6^3 numerical result
[3]See Appendix C for T_6^0 numerical result

2.5 Gripper description

In this project a seventh joint is used, the one that closes and opens the two-finger gripper, shown in Fig. 2.6.
The transformation found in previous section expresses the position of the end of joint 6 with respect to the base reference frame.
To get the final expression of the direct kinematics we must add a seventh transformation which takes in account the distance between the sixth axis origin and the pick point of the gripper.
We will call this transformation T^6_{EE}, the transformation between the sixth link and the end-effector.

Figure 2.6: e.Do gripper, avaible at e.Do shop,[**8**]

Due to the mechanism structure, the length of this distance changes with the opening width of the gripper: in fact we can see how this point can move up and down as a consequence of the rotation of the mechanism that opens and closes it, as shown in Fig. 2.7 . What we expect is that also T^6_{EE} is a function of a joint variable, $T^6_{EE}(q_{ee})$. Joint seven acts as a prismatic joint, translating the end effector reference frame along the z_6 axis.
Since the exact physical dimensions of the gripper parts were not known and in order to simplify the direct kinematic functions, freeing it from a seventh variable, the following simplifying procedure has been adopted:
During the execution of the pick and place task, eachtime we need to position the gripper to pick an object, the necessary opening width is assumed to be always the same, meaning $q_{EE} = const$.
This is true for the specific application because all the picked objects have the same dimensions.
If $q_{EE} = const$ then $T^6_{EE}(q_{ee}) = T^6_{EE} = const$. The homogeneous transformation T^6_{EE} is found as a translation of a fixed distance d_{EE} along z_6.
The distance has been found using the protected inverse Kinematic contained

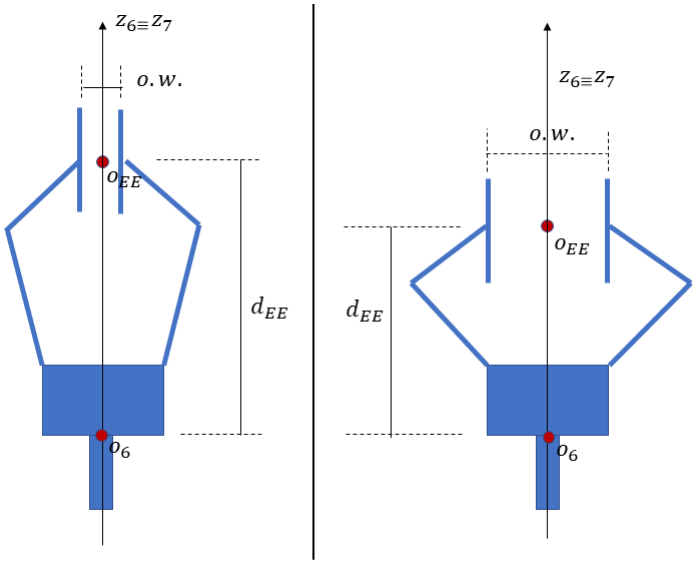

Figure 2.7: e.Do gripper schematic, different opening width, o.w., results in differents d_{EE}

in the android application provided by Comau. The robot has been configured in six joints mode, no end effector attached, all joints are moved to position zero, the robot is now aligned along the base reference frame z axis, z_0. The inverse kinematics provides the position of o_6 w.r.t[4] the base frame. The z-coordinate is a measure of the arm extension. We repeat the procedure with the robot configured with the seventh joint. Once we set the right opening width of the gripper we repeat the previous measure. Subtracting the first measure to the second measure we find d_{EE}, the distance between the pick point, at the specific opening width needed to pick the sample, and the end of sixth joint. It's now possible to find T^6_{EE} as a translation of that distance along the z axis of reference frame 6. A sketch of the procedure is illustraded in Fig. 2.8 .
Coordinates in *base* reference frame of o_6 as ${}^0\vec{o_6}$, and those of o_{EE}, as ${}^0\vec{o_{EE}}$, are reported [5] using maximum opening width, ow_{max}, closed grip, ow_c, and sample thickness, $ow_p = 26$ mm:

[4] w.r.t. with respect to
[5] Mesures are expressed in milimmiters

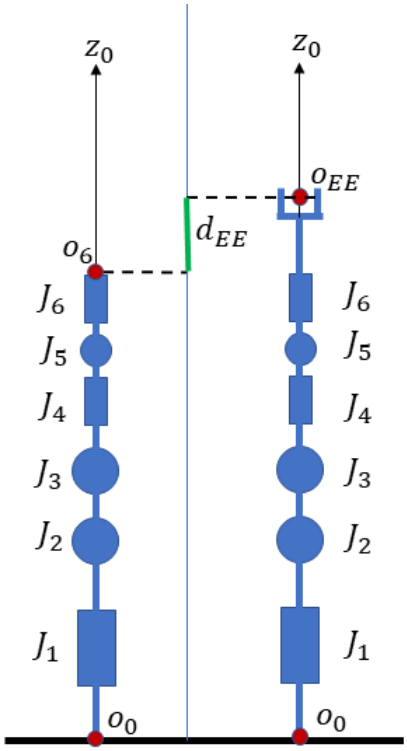

Figure 2.8: e.Do consifugration for d_{EE} measurment

$$\begin{aligned} {}^0\vec{o_6} &= \begin{bmatrix} 0 & 0 & 990 \text{ mm} \end{bmatrix} \\ {}^0\vec{o_{EE}}(ow_c) &= \begin{bmatrix} 0 & 0 & 1\,122 \text{ mm} \end{bmatrix} \\ {}^0\vec{o_{EE}}(ow_{max}) &= \begin{bmatrix} 0 & 0 & 1\,096 \text{ mm} \end{bmatrix} \\ {}^0\vec{o_{EE}}(ow_p) &= \begin{bmatrix} 0 & 0 & 1\,120 \text{ mm} \end{bmatrix} \end{aligned} \tag{2.11}$$

For the specific application we find d_{EE} as:

$$d_{EE} = \left[{}^0\vec{o_{EE}}(ow_p) - {}^0\vec{o_6} \right]_z = 130 \text{ mm} \tag{2.12}$$

2.6 Modelling e.Do kinematic chain

Peter Corke's robotic toolbox, [**9**], provided useful Matlab tools to realize an e.Do kinematic model. Thanks to this toolbox a Matlab object for simulating the robot has been created starting from the knowledge of the DH

```
edo =

eDo:: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|      0.337|          0|    -1.5708|          0|
|  2|         q2|          0|     0.2105|          0|    -1.5708|
|  3|         q3|          0|          0|    -1.5708|    -1.5708|
|  4|         q4|      0.268|          0|     1.5708|          0|
|  5|         q5|          0|          0|    -1.5708|          0|
|  6|         q6|     0.1745|          0|          0|    3.14159|
+---+-----------+-----------+-----------+-----------+-----------+
tool:    t = (0, 0, 0.13), RPY/xyz = (0, 0, 0) deg
```

Figure 2.9: e.Do's parameter in Peter Corke's Toolbox, [**9**]

parameters. In Fig. 2.9 the parameters for the model are shown. With the toolbox is possible to plot a graphic representation of e.Do, that is of use when designing a path planning algorithm poses. The grapich model is animated with a *teach* feature, Fig. 2.10.

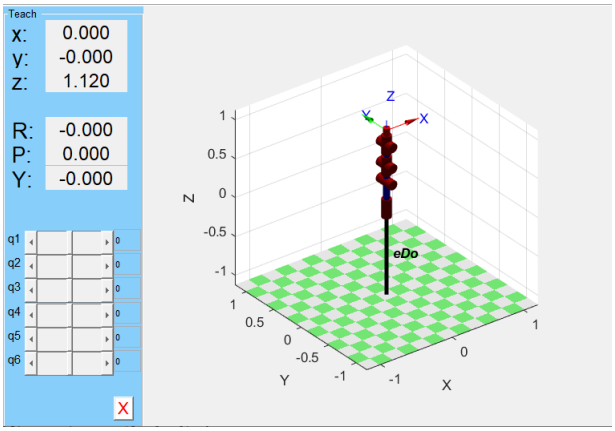

Figure 2.10: e.Do's model and teach in Peter Corke's Toolbox,[**9**]

n	d_i [m]	a_i [m]	α_i [rad]	θ_i
1	0.337	0	$-\frac{\pi}{2}$	q_1
2	0	0.2105	0	q_2
3	0	0	$-\frac{\pi}{2}$	q_3
4	0.268	0	$\frac{\pi}{2}$	q_4
5	0	0	$-\frac{\pi}{2}$	q_5
6	0	0	0	q_6

Table 2.4: e.Do's Denavit–Hartenberg parametrs with o_6 in wrist center

2.7 Computation of the Jacobian

Choosing the end effector reference frame origin at the intersection of the wrist axes, it is possible to simplify the calculation for the Jacobian and the robot singularities.
To do this we must modify the Denavit Hartneberg parameters as in Table 2.4.
The Geometric Orientation Jacobian matrix obtained is the following[6]:

$$J_o = \begin{bmatrix} {}^0\vec{z_0} & {}^0\vec{z_1} & {}^0\vec{z_2} & {}^0\vec{z_3} & {}^0\vec{z_4} & {}^0\vec{z_5} \end{bmatrix} \tag{2.13}$$

The Position Jacobian obtained is the following:

$$J_p = \begin{bmatrix} {}^0\vec{z_0} \times {}^0\vec{o_6} & {}^0\vec{z_1} \times {}^0\vec{o_6} & {}^0\vec{z_2} \times {}^0\vec{o_6} & {}^0\vec{z_3} \times {}^0\vec{o_6} & {}^0\vec{z_4} \times {}^0\vec{o_6} & {}^0\vec{z_5} \times {}^0\vec{o_6} \end{bmatrix} \tag{2.14}$$

To find singolarities we look for those $\vec{q}$ such that $det(J) = 0$.

- $q_5 = 0$, wrist singolarity
- $q_3 = 0$, elbow singolarities
- $(536 \cdot s_{23} - 421 \cdot c_2) = 0$, shoulder singolarities [7]

Singolarities given by the third element of the list above, the shoulder singolarities, have been found; Fig. 2.11 shows all possible combinations of q_2 and q_3 such that $(536 \cdot s_{23} - 421 \cdot c_2) = 0$.

[6] See Appendix C for numerical results of Geometric Jacobian
[7] $sin(q_2 + q_3) = s_{23}$

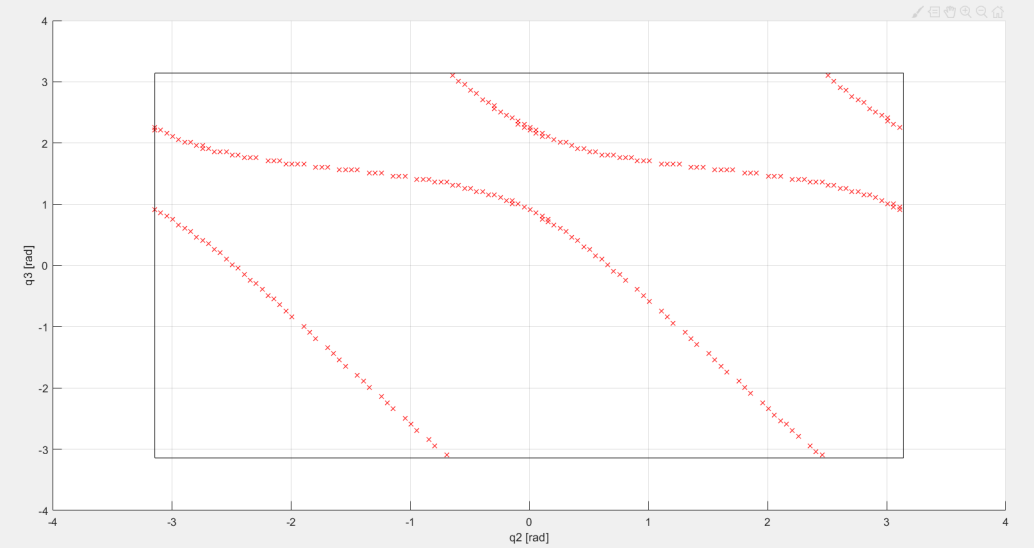

Figure 2.11: e.Do's shoulder singolarities

2.8 Workspace

A qualitative representation of the robot workspace, to analyze its reachability limits, has been studied.
In the first place a method based on repetitive calculations has been tested. The direct kinematic function was evaluated for a set of joint variable values combinations. The trade off between the density of points and the computational cost made this method fail.
Then a more efficient way to represent the robot workspace has been tested. Of crucial importance for this study is the knowledge of the joints rotational limits. Mechanical structure of e.Do doesn't allow its joints to spin continuously. Depending on the specific joint the rotational limits can vary as in table 2.5.

Starting from paper [**10**], a method to express the boundaries surfaces of the workspace, designed for the specific robot, has been found.
Since q_1, q_4, q_6 can spin at least 2π, we can simplify the problem and study, instead of surfaces, the curves, in bidimensional space, that the end effector trajectory generates when at its maximum and minimum reach. Once these curves are found, applying a rotation of 2π we can get the expression of the boundaries surfaces of the workspace.
Known $\vec{o_{EE}}(\vec{q})$ from the direct kinematic study, function from $\Re^6$ to $\Re^3$, substituting 5 of the 6 joint variables with constant values, we obtain the function $\vec{o_{EE}}(q)$, from $\Re^1$ to $\Re^3$, which describes a parametric curve in the

n	min [deg]	max [deg]
q_1	0	360
q_2	-113	+113
q_3	-113	113
q_4	0	360
q_5	-104	104
q_6	0	360

Table 2.5: Limit joint rotations

space.

- $q_1 = q_4 = q_6 = const. = 0$. We are interested in studying the problem in xz plane. Those joint variable are neglectable.
- q_2 , q_3, q_5 We keep one of them as a variable and fix the remaining two to a singularity[8] constant value. Multiple combinations are possible. Among all combinations of singularity values for q_i, $i = 2, 3, 5$ only 3 are selected. Due to the mechanical structure and rotational limits, e.Do can hit himself, in fact some of the surfaces intersect the body of the robot; on this basis some surfaces are discarded.

The limit curves choosen are the following:

$$\gamma_1(x, y, z = 0) = o\vec{E}E(0, q_2, q_{3elbow}, 0, 0, 0), q_2 \in \left\{q_{2min} : q_{2max}\right\} \tag{2.15}$$

$$\gamma_2(x, y, z = 0) = o\vec{E}E(0, q_{2max}, q_3, 0, 0, 0), q_3 \in \left\{q_{3elbow} : q_{3max}\right\} \tag{2.16}$$

$$\gamma_3(x, y = 0, z) = o\vec{E}E(0, q_2, q_{3max}, 0, q_{5max}, 0), q_2 \in \left\{q_{2max} : q_{2min}\right\} \tag{2.17}$$

Applying a rotation of a parameter v, as in equation (2.18), bounded between 0 and 2π, we obtain the corresponding parametric surfaces expressed by equation (2.19).

$$R_z(v) = \begin{bmatrix} cos(v) & -sin(v) & 0 \\ sin(v) & cos(v) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

[8]Singularity concept is extended also to mechanical joint limit since they constitute also a loss of mobility

$$
\begin{aligned}
\gamma_i(x,z) = \vec{f_\gamma}(s) \quad & f_\gamma : \Re^1 \to \Re^2 \\
S_i(x,y,z) = \vec{f_S}(u,v) \quad & f_S : \Re^2 \to \Re^3 \\
s \to u & \\
v \in [0 \ : \ 2\pi] & \\
S_i(x,y,z) = R_z(v) \cdot \begin{bmatrix} \gamma_{ix} \\ 0 \\ \gamma_{iz} \end{bmatrix} &
\end{aligned}
\tag{2.19}
$$

Once the parametric expression of the surfaces have been found they can be plotted with any math tool not needing heavy computational resources. In Fig. 2.12, Fig. 2.13 and Fig. 2.14 results for S_1, S_2, S_3 are shown[9]. In Fig. 2.15 a plot of all the surfaces and e.Do[10] is presented.

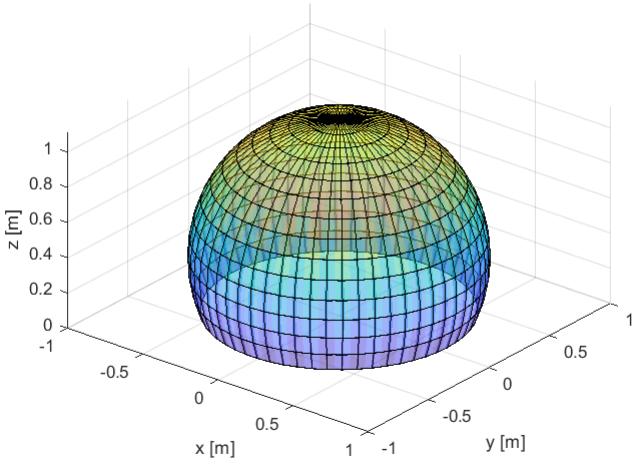

Figure 2.12: S_1

[9]For numerical results see Appendix C, Workspace section
[10]Peter Cork's Robotic Toolbox graphical representation

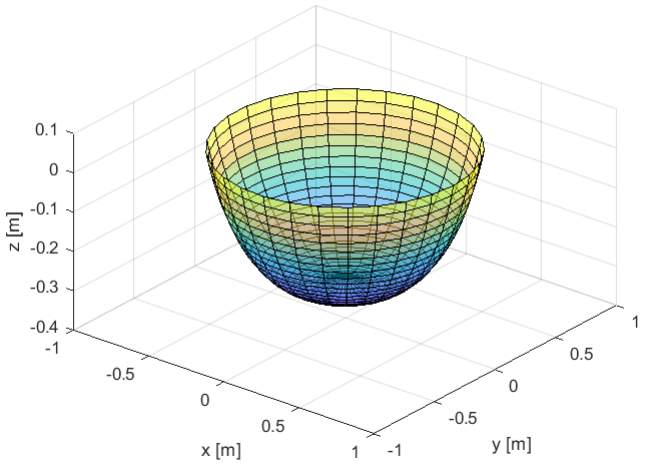

Figure 2.13: S_2

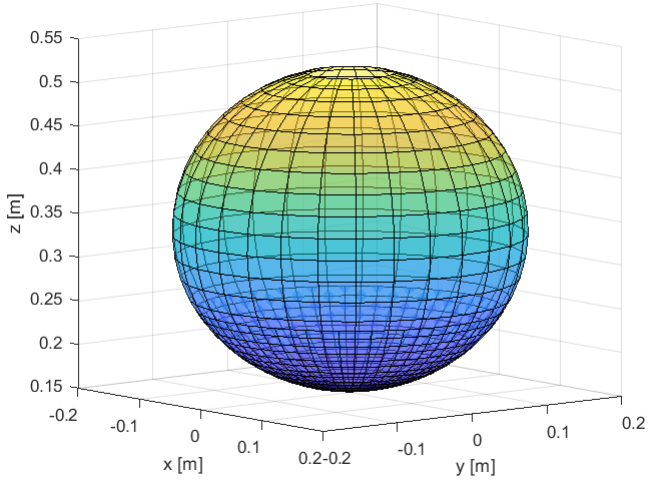

Figure 2.14: S_3

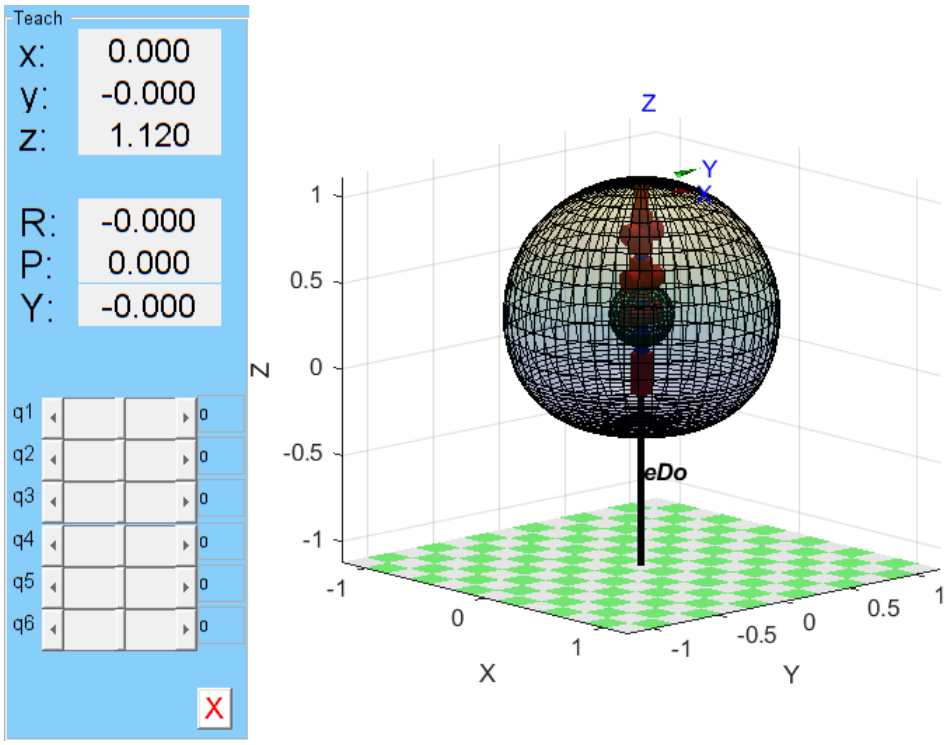

Figure 2.15: e.Do and its boundaries

Chapter 3

mesoSPIM digital microscope

MesoSPIM, Fig. 3.1, is a digital microscope. The software it runs is open source, developed in python language and available in github, [**11**]. Since the microscope was not physically available during the project development, we procede highlighting the main features of interest for this project.

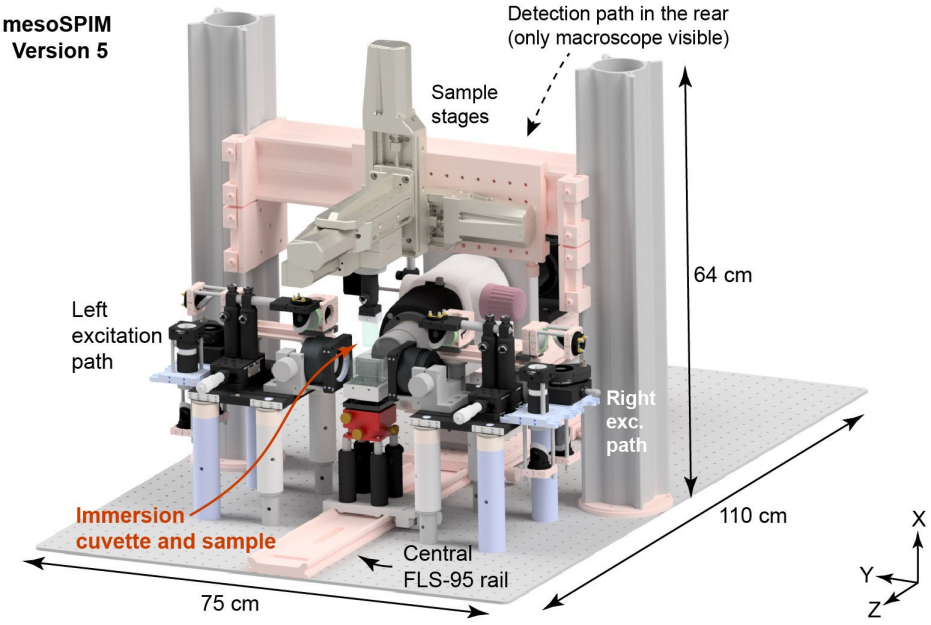

Figure 3.1: The digital microscope mesoSPIM, [**12**]

3.1 mesoSPIM in project context

What is mostly relevant from a structural point of view is the space available for delivering the samples, that has been measured from a CAD, provided by the designer of the microscope. Another relevant aspect of the structure of the microscope is the mechanism with which the sample is held. mesoSPIM accepts samples by means of an electro magnet, positioned above the immersion container. Samples have a magnet located on the top. When delivering the sample the electro magnet activates and attracts the samples.
Since during the project development it has not been possible to visit the labs in which mesoSPIM is located, the functioning of the microscope code has been outlined reading the available git-hub.
In the first place it is important to highlight that the software for the control of the microscope, called *mesoSPIM_control*, is meant to run in a Windows environment; this played an important role in the design of the interface.
The most relevant feature about mesoSPIM's algorithm is the presence of a variable to store the status of the microscope and a method to start the actions to perform an analysis. From the knowledge of this two aspects it is possible to design a simple interface between mesoSPIM and e.Do.
To overcome the absence of the physical microscope, the property of supporting code spread across multiple machine of ROS is exploited. The microscope is implemented as a node. The behavior of this node is to communicate the microscope state to the interface and accept from it requests of analysis begin, Fig. 3.2.

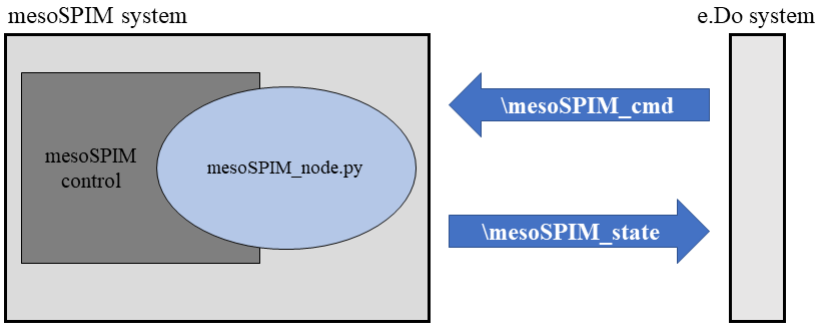

Figure 3.2: mesoSPIM node in its system

On the topic *mesoSPIM_cmd* requests of beginning analysis are accepted by the microscope; this is achieved sending a string message containing the "START" word. On the topic *mesoSPIM_state* instead mesoSPIM comunicates its status. The possible state words, sent as string messages, are:

- Init : Microscope not ready to perform analysis.
- Ready: Microscope ready to accept "START" command.
- Busy: Microscope currently performing analysis.

To replace the real execution of the task, upon reception of a START message, the node simulates a BUSY state for a time $T_{analysis}$, then becomes READY again, Fig. 3.3.
During the first phase of application development, the microscope was meant

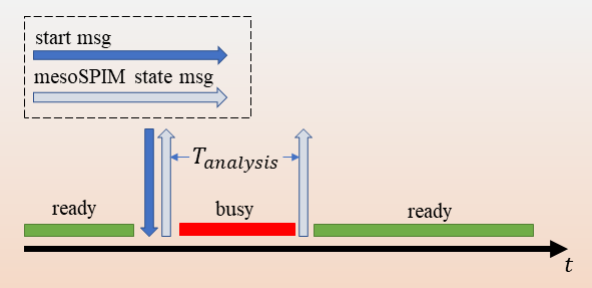

Figure 3.3: mesoSPIM simulated behaviour

to be simulated on the same machine of eDo. In a second phase the node ran on another machine connected to the ROS network, setting manipulator's ROS process as the master. In the final design phase the node was then moved outside the context of a ROS environment. With the use of specific libraries the same behaviors were implemented in a system running Windows O.S.

Chapter 4

Project Development

4.1 Introduction

Objective of this project was to develop an interface to accomplish a pick and place task. Part of the pick and place (task) involved the digital microscope mesoSPIM. It was then needed that the interface coordinates e.Do's action with those of mesoSPIM.
The manipulator has to pick a sample from a sample holder, deliver it inside the microscope, wait for it to be analyzed and then put the sample back in its place, as shown in Fig. 4.1.

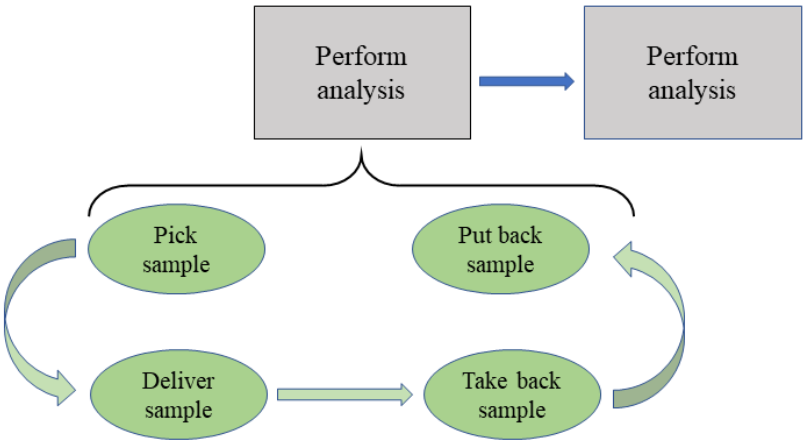

Figure 4.1: Multiple analysis tasks, each one can be decomposed in lower level tasks

This series of action has to be repeated for a series of different samples, all

located in the holder.

4.2 Task analysis

A top-down decomposition of the task is presented to better understand the objective.

- High level: a series of analysis must be performed, each of them involves a different sample among those located in the holder. Samples are identified by a number, to each number corresponds a sample in a specific holder position. Higher level task then comes in the form: Perform analysis of sample 1, then sample 2 and so on.
- Medium level: In order to accomplish an analysis of a sample, a series of actions are requested. These actions are:
  - Pick the desired sample.
  - Deliver the sample to mesoSPIM.
  - Wait for mesoSPIM to perform the analysis.
  - Take back the sample from the microscope.
  - Put the sample back in its holder place.

  Once completed the above series of actions, we repeat for the successive requested sample.
- Low level: Each one of the medium level actions has been decomposed in motions between known points.

First of all three zones of interest are identified as follows:

- Initial position zone: Location in which the robot is upon finishing the calibration.
- Sample Holder zone: Location in which the sample holder is positioned.
- mesoSPIM zone : Location in which the microscope is positioned

We look for remarkable points in each of the zones to define the robot elementary moves.
In the initial position zone we identify:

- Init point: In this point the robot is positioned as soon as the calibration procedure is completed.
- Home point: The point, away from singularity initial position, where we first move the robot.

In the Sample Holder zone we identify:

- Holder point: Point positioned above the holder, this is the point from which we start/end moves from/to the holder of samples.
- Sample points: These points are the target ones for picking and releasing of the samples. The number of points is equal to the number of the different samples.
- Sample-via points: Above each one of the sample points we choose a via point. Via points are located on top of each sample. Their choice is such that motion will happens on a straight vertical line when moving between via points and sample points.

In the mesoSPIM zone we identify:

- mesoSPIM deliver point: Target point inside the microscope. Here we need to position the sample for the analysis.
- mesoSPIM proximity point: This point, located in front of the mesoSPIM point, is the one from where we start/end the movements to/from the microscope delivery point. This point is also used as the position where to wait for the sample to be analyzed. Its choice is made to constrain the movements from and to the microscope on a straight line.

4.3 Constraints

When designing the path planning and the application itself some constraint had to be respected.

4.3.1 Pose constraints

- Deliver: The body of the microscope during the pick and place from and to it constrains the physical trajectory of the end effector to a straight line. The deliver point is in fact contained inside a cube approximately of 10 cm edge, where five of the six faces are occupied by the microscope

components. Then when approaching the microscope the robot must move in a straight line orthogonal to the microscope frontal plane, Fig. 4.2.

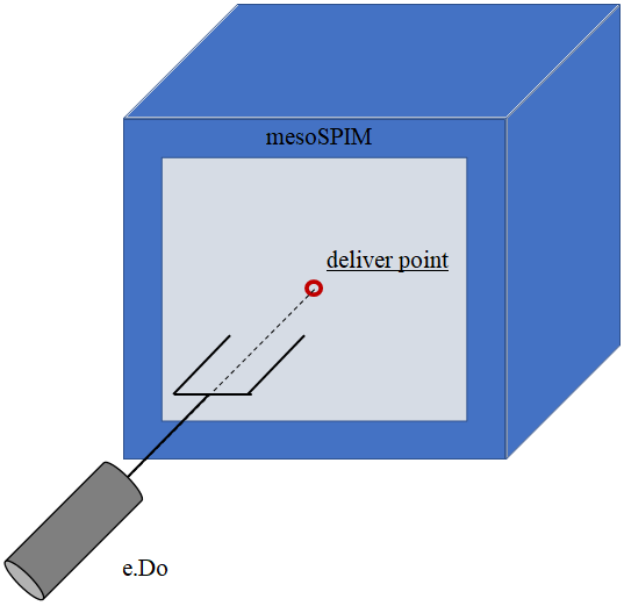

Figure 4.2: Trajectory for reaching the delivery point

- Sample pick: Samples are placed in a stair-shaped holder. In order to pick samples the gripper moves on top of them and then slides from above to grasp. The stairs allow to position the gripper in order not to damage the nearby samples, Fig. 4.3.

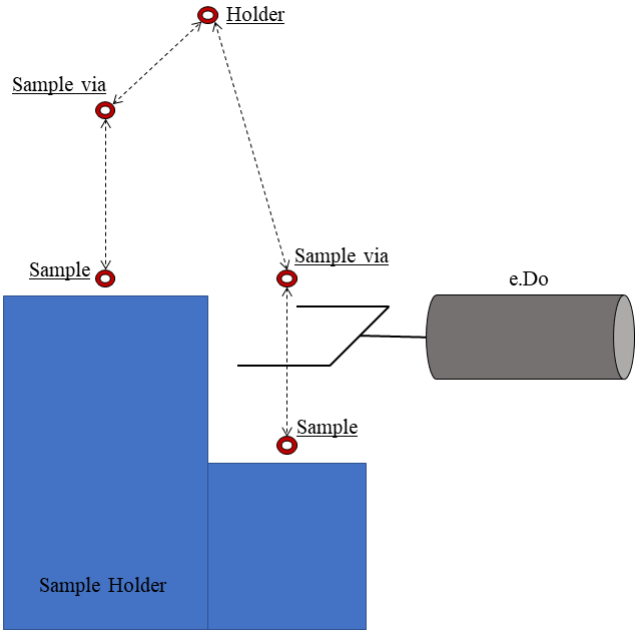

Figure 4.3: Trajectory for reaching the sample points

4.3.2 Software Constraints

- Move Type: Publishing on */bridge_move* the messages mentioned before allows to move the robot from one point to another. When communicating a target point the user must decide first what format to use for the coordinates, then the desired trajectory.
  Trajectory generation is handled currently in two ways, with a linear planning in the joint space, or else a linear planning in the cartesian space.
  During the development of the project both solutions were tested. Joint space planning resulted to be robust and singularity-proof. It was chosen for long distances, in which the end effector could pass by singularity points, and when exact trajectory of the end effector is not needed to be imposed.
  Cartesian space planning was still under development. Among the two data format possible, the joint one proved to be more efficient.
  As a consequence, when moving the robot using the cartesian linear planning, joint coordinates of target points must be computed first.
  This problem is solved by the inverse kinematic function. Due the fact that inverse kinematics is contained in the private library function its solution will not be discussed.

Cartesian space planning was used when it was needed to impose a specific straight line in the space to avoid collisions with environment.

- Decoupling of algorithm: When designing the application two approaches were taken into consideration: A first one in which an application designed for the specific task would have been produced. A second one aimed at making easier a possible integration with future upgrades of e.Do system.
  Considering the evolutionary state of e.Do project solution two was adopted. Project application proved to be compatible through three consecutive releases, 2.3 2.4 2.5. This choice lead to design a decoupled system, preferring to separate the implementation of the methods and functionalities between them.
  The use of a multi-node structure in the computational graph was adopted and will be illustrated in Section 4.6.

4.4 Path Planning design

4.4.1 Move Type Choice

When deciding the movement types a mixed solution, between type joint and type cartesian moves, has been adopted.
This choice is justified by the need of having a robust application without fixing the microscope and sample holder position with respect to the robot. Due to the evolutionary situation in which this project has been developed, the position of the microscope and holder in the space were not known, additionally it was assumed that they could vary in time. Even though cartesian planning is preferable, since the generated trajectory has a predictable shape, it can't be used for long distance movements that may involve singularity close passage.
When addressing all the remarkable points in the space we grouped them in zones of interest. Point belonging to the same zone are reasonably close and the trajectory between them occurs along a straight line, thus it's easy to calculate. For this category of move commands is then chosen the cartesian planning type. Defined geometric constraints, that will ensure the correct execution, are less likely going to change in future developments since they depend on the structure of microscope and holder.
When performing instead a move between different zones we can expect to explore a wider part of the workspace, as showed in Fig. 4.4.
For this reason joint type planning has been chosen to move across different zones.

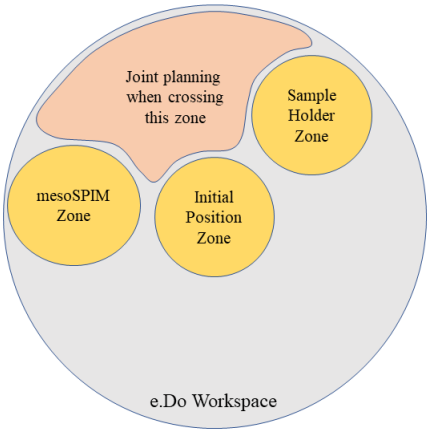

Figure 4.4: Identified zones in the workspace to group target points

Zone	Target Point
Initial	Init
Initial	Home
Sample Holder	Holder
Sample Holder	Sample
Sample Holder	Sample-Via
mesoSPIM	Deliver
mesoSPIM	Proximity

Table 4.1: Target points and their belonging zone

Samples store delicate materials and they cant't be flipped. When moving with joint linear planning, where the cartesian trajectory of the end effector is hardly predictable, samples risk to be damaged.
Start and end point of these moves are taken with closest possible orientation for pitch and roll angles. Small variation of the end effector roll/pitch orientation during moves proved the joint algorithm planning safe for the samples.

4.4.2 Path definition

The path for the considered task is then defined as a series of movements between known points, Table 4.1, with different types of moves. Table 4.2 reports the sequence moves between the target points in a typical analysis by the microscope. The final path planning results in a series of consecutive

Starting Point	Ending Point	Move type
Init	Home	Joint
Home	Holder	Joint
Holder	Sample-via[i]	Cartesian
Sample-via[i]	Sample[i]	Cartesian
Sample[i]	Sample-via[i]	Cartesian
Sample-via[i]	Holder	Cartesian
Holder	Proximity	Joint
Proximity	Deliver	Cartesian
Deliver	Proximity	Cartesian
Proximity	Holder	Joint
Holder	Sample-via[i]	Cartesian
Sample-via[i]	Sample[i]	Cartesian
Sample[i]	Sample-via[i]	Cartesian
Sample-via[i]	Holder	Cartesian

Table 4.2: Moves between target points for a single analysis

moves, some of them planned in joint space, some planned in cartesian space. An example is illustrated in Fig. 4.5.

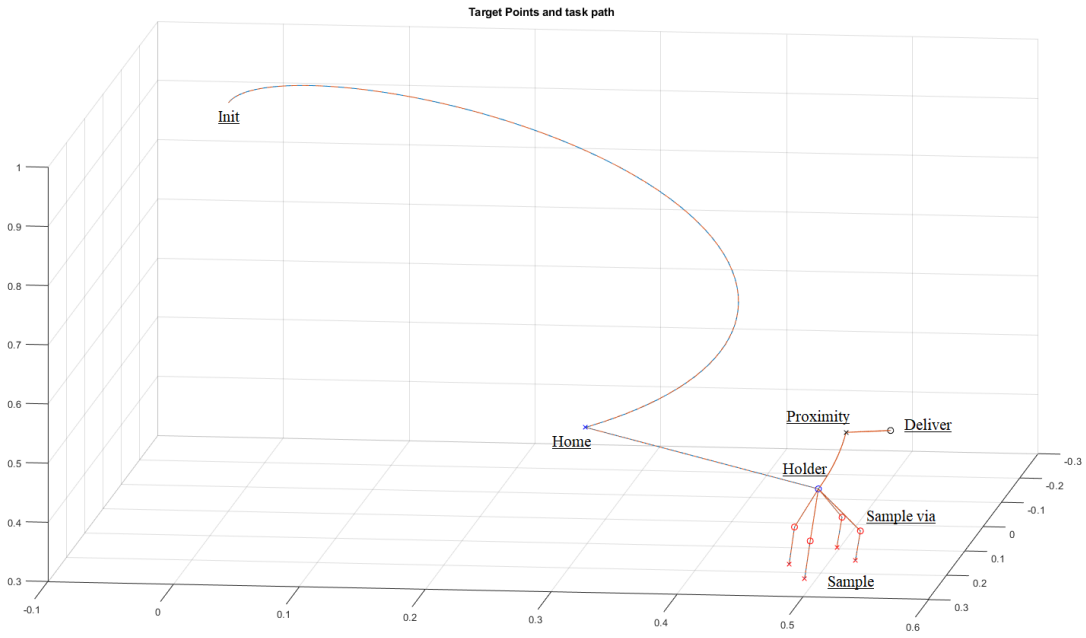

Figure 4.5: Multiple analysis tasks, each one can be decomposed in lower level tasks

4.5 Community resources

During the development of this project ROS packages by the community were integrated in the application.

4.5.1 edo_core_pkg and _msg

The algorithms provided by Comau, mentioned in Section 2.3, that has been used for the control of the system, come as a ROS package. Alongside *edo_core_pkg*, the package *edo_core_msgs* has been used. This package serves as a container for all the message definitions of *edo_core_pkg*.

4.5.2 SMACH

Executive SMACH is a metapackage made available by the community[1].
The tools provided by this metapackage are those needed when designing a Machine State compliant with ROS.

[1]http://wiki.ros.org/smach

Using ROS_SMACH a finite-state machine was designed, thanks to the functionalities provided, states could integrate ROS methods like publish and subscribe.

4.5.3 ROSLIBPY

Microscope operating system is not compatible with ROS. Messages between e.Do and the microscope had to be exchanged across a mixed o.s. infrastructure. For this need ROSLIBPY[**13**] provided already implemented solutions. Basic functionalities typical of a ROS system, as subscribe publish and server parameter communication, are supported.

4.5.4 ROS Bridge

ROS Bridge is a ROS metapackage providing "a JSON API to ROS functionality for non-ROS programs"[**14**]. Many front-end for the comunication are available, inside the already existing e.Do system a node for a Websocket is present.
The Web socket allows comunication with the Android App. To facilitate system design to comunicate with mesoSPIM, a *non-ROS* system, the same Websocket is used. Roslibpy allows to comunicate from the microscope side connecting to the Websocket basic ROS methods such as publish and subscribe. In Fig. 4.6 the network configuration between the two system is sketched,

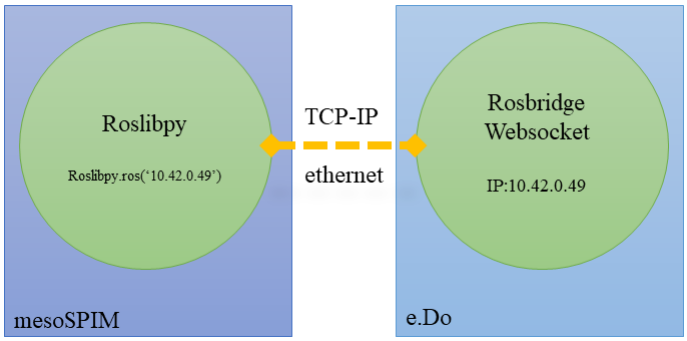

Figure 4.6: Network connection between mesoSPIM and e.Do

4.6 Package development

To implement all the features discussed above a ROS package was created. The final structure comes in the form of an algorithm to control the two different systems. At this purpose a finite state machine was designed, collecting the state of each of the two subsystems, a series of actions are requested. Those actions come in the form of move requests for the manipulator and sample analysis request for the microscope.
When writing a ROS code one must decide whether to use Python language or Cpp.
Python was preferred since dynamically interpreted[**15**], this feature plays a crucial aspect during the development phase of a project of this nature.
The use of Python allowed to save time when testing different solutions of the code. Another Python feature that led to choose it as main language for this asplication is its reduced complexity in writing structures suitable for high level system implementation.
The interface had to function as a high level layer controlling two subsystems, each one already provided with its control software.
The outline of the whole system is shown in Fig. 4.7.

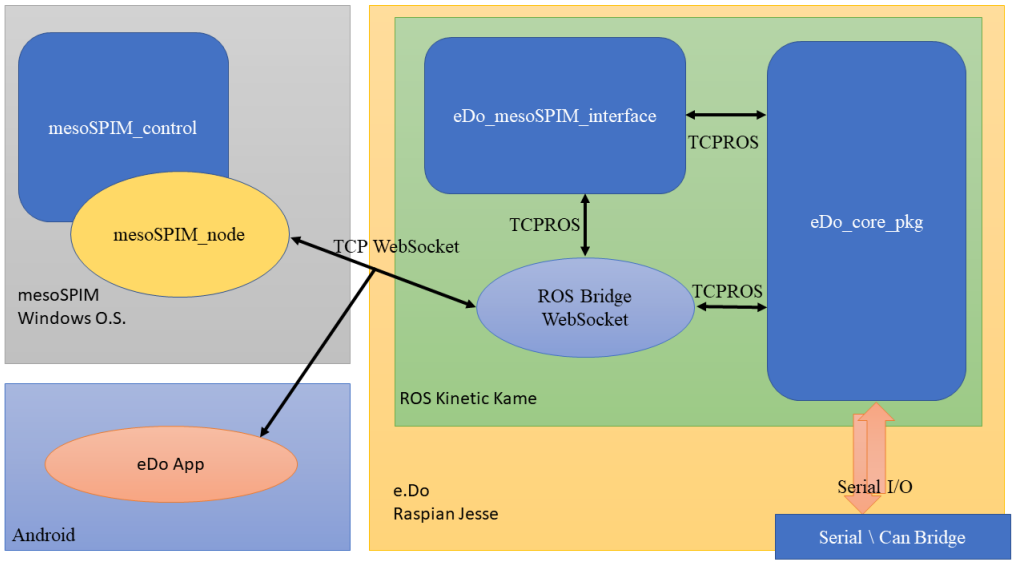

Figure 4.7: Systems interaction scheme

4.6.1 Package resources

In the package directory the following resources are contained:

- src/ : Folder containing all the .py files that are used by the system.
- param/ : Folder containing the param.yaml file in which static parameters for the application are defined. Values stored in parameters are constant variables needed across the application.
- launch/ : Folder containing the launch files.
  Those types of files are needed when a molteplicity of nodes and services are requested to run by the same application.
  Additionally the launch file loads the parameters defined values at startup, making them available to the nodes. Launch file comes with two versions. A first one to launch the program with joints coordinates as default, a second one to launch cartesian coordinates as default. Second option is yet not fully supported, *edo_core_pkg* proved not to be enough robust when dealing with cartesian coordinate algorithm.
- readme.txt : text file containing instruction for installing and configuring the package.
- package.xml : Manifest of package.

4.6.2 State Machine

To coordinate the actions of the two subsystems a state machine is implemented,whose states are shown in Fig. 4.8. Evolution of the system is accomplished upon completion of different tasks. The states defined are:

- Init:
  This is the initial state when the robot is powered on. This state monitor the topic /system_state; when receiving a READY message from there the system evolves to the next status.
- Build queue:
  In this state the queue of samples to analyse is loaded from the parameter server. Sample's ID are saved as an array of integers. Upon completion of analysis the queue will be emptied.
- Publish:
  After the queue is loaded its first element is popped out. From this information we know what sample to analyse. If no elements are in the queue, the task is completed and the application terminates. Once the sample that has to be analysed is known, the first move message is sent.

Publish state contains the sequence of all the move messages to send in order to complete one analysis. To take record of the current state of the analysis a counter for the movements is implemented. When the message is published the state evolves to Wait move.

- Wait move:
  This state serves as a synchronizing state. Wait moves implement no specific actions but to wait for e.Do to start moving after receiving the move message.
- Wait ready:
  As wait move, wait ready contains no specific actions. In this state the system waits for both e.Do and mesoSPIM to be ready. We want the task execution to procede only if e.Do is ready, meaning it is not moving or in error state, and mesoSPIM is ready to accept a new sample to analyse.

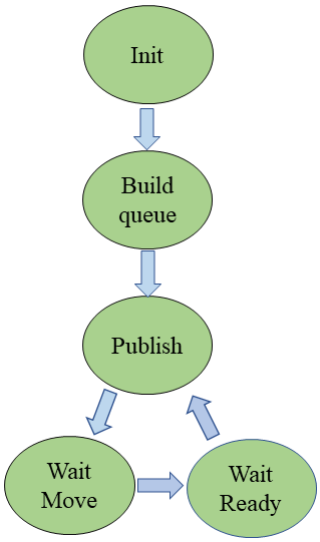

Figure 4.8: Interface State Machine to coordinate subsystem actions

4.6.3 State Interpreter

To deal with the interpretation of the manipulator microscope states and to extrapolate a higher-level information, the state interpreter has been designed.

e.Do original state	e.Do reduced state
COMMAND_STATE	not implemented
INIT	INIT
NOT_CALIBRATE	INIT
CALIBRATE	READY
MOVE	MOVE
JOG	MOVE
MACHINE_ERROR	ERROR
BRAKED	ERROR

Table 4.3: State Machine enumerate states

This node subscribes to *machine_state* topic, published by e.Do's state machine, and to *meso_state*, topic in which the microscope publishes its state. Information sent across *machine_state* resulted to be redundant for the high level action planning.
The state interpreter changes the method with which subsystems' states are communicated to the interface state machine; from being periodically published to be published only in case of their change. A mapping between e.Do S.M.[2] states to a new and less numerous set of states is implemented to reduce algorithm complexity, Table 4.3.

From the knowledge of the subsystem states it is possible to define a new conceptual state that is the one of the system in its whole. A mapping to obtain this information is presented in Fig. 4.9.

		e.Do reduced state			
		Init	Ready	Move	Error
mesoSPIM state	Init	Init	Init	Error	Error
	Ready	Init	Ready	Busy	Error
	Busy	Error	Busy	Busy	Error

Figure 4.9: Interface State Machine to coordinate subsystem actions

In future *edo_core_pkg* releases, having the State Interpreter to decouple the interface state with that one of their subsystems, will facilitate to adapt to changes of the code.

[2]S.M. : State Machine

4.6.4 Command Interpreter

With the same principle used in the state interpreter we adopt a specific node, the command interpreter, to send and handle assembly of messages.
The Interface state machine communicates to the command interpreter a message containing the start point and the end point of the move to accomplish.
Two Different command interpreters were realized in order to test separately cartesian coordinates format and joint coordinate format.

4.7 Testing and results evaluation

During the last steps of the project development, a simulation of the procedure to set up the interface and the task itself has been realized. The execution was videotaped for demonstrational purposes.
Reproducing a scenario more likely as possible to the real one helped understanding what weaknesses the interface had and what could be improved.

4.7.1 Scene Objects

For what coencerns the environment with which e.Do has to interact during the pick and place, reproduction of the samples, the microscope and the sample holder have been built.

- Sample : Four identical reproductions of a sample have been 3-d printed. A real sample is a box of dimensions 46x26x18 mm. Printed samples have the same dimensions as the real ones. These are the objects whom will be picked by e.Do.

- Sample Holder: The sample holder is the place in which samples are stored. From here e.Do takes samples and put them back. The shape choosen for the design is such that e.Do can easily grab samples without damaging those nearby. The number of samples used for the simulation is 4, then the designed holder can hold the same number. This design is easily expandable in the number of samples that can be held adding more stairs or more cells in line.

- mesoSPIM: When trying to reproduce the microscope we look for having the same space for delivering the sample that mesoSPIM offers. Using the CAD project of the microscope dimension were taken and then a similar shape has been 3d printed. A box of 120x75x78 mm

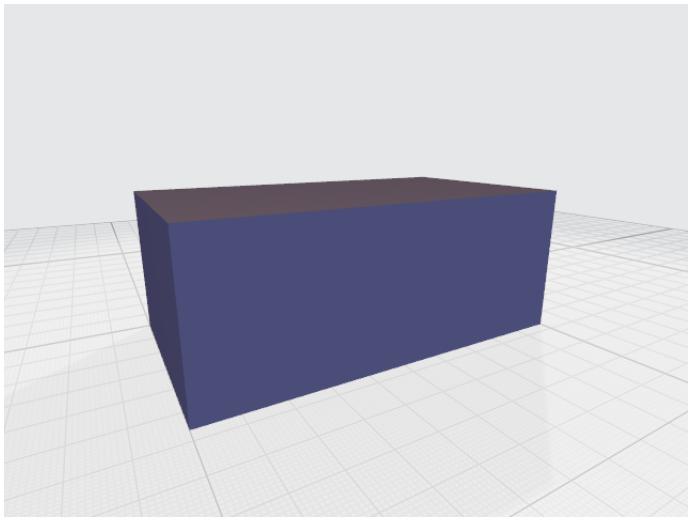

Figure 4.10: Sample 3d model

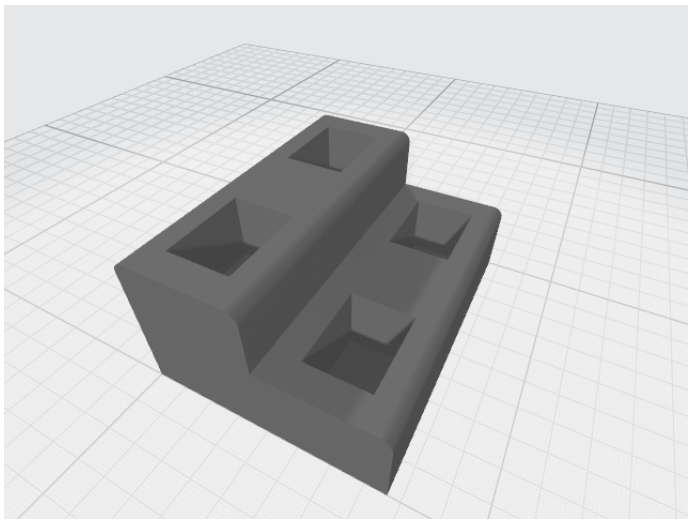

Figure 4.11: Sample holder unloaded

with two zones to unload the sample, one on the floor, and one on the ceiling, meant to be used with a magnet is shown in Fig. 4.12.

4.7.2 Set Up of Interface

To set up the interface the user must teach the target points discussed in the previous section.
Teaching of targets point is done using the JOG function on the android

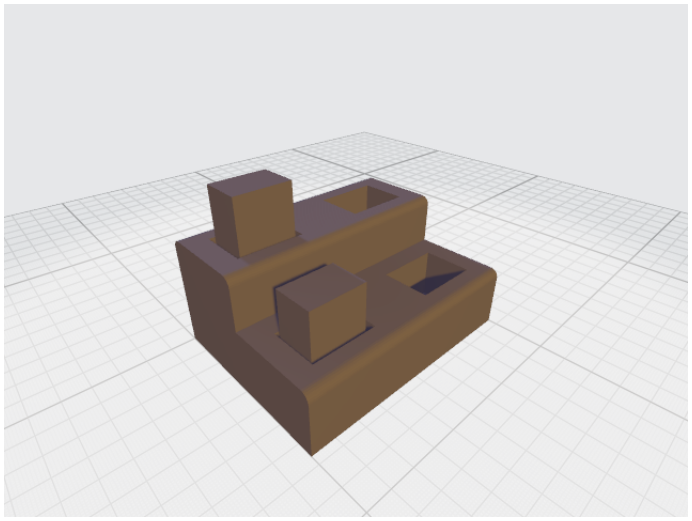

Figure 4.12: Sample holder loaded with two samples

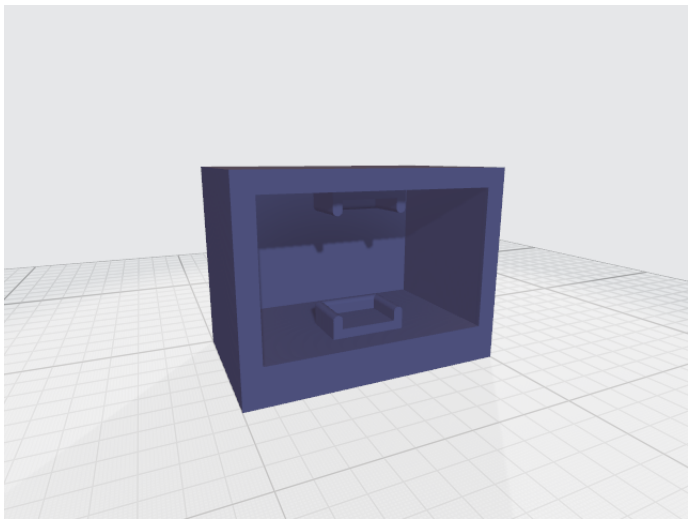

Figure 4.13: mesoSPIM reproduction

application. The end effector is moved on each one of the following target points : samples, deliver, and home points.
Once these points are reached we register the corresponding joint coordinates. From the knowledge of these taught points the equivalent cartesian coordinates for the holder, sample via, and proximity target points, that re-

[3]Inverse kinematic was not accessible, then if a point is known in cartesian coordinates we must use protected inverse kinematic algorithm contained in the android application to convert to joints coordinates

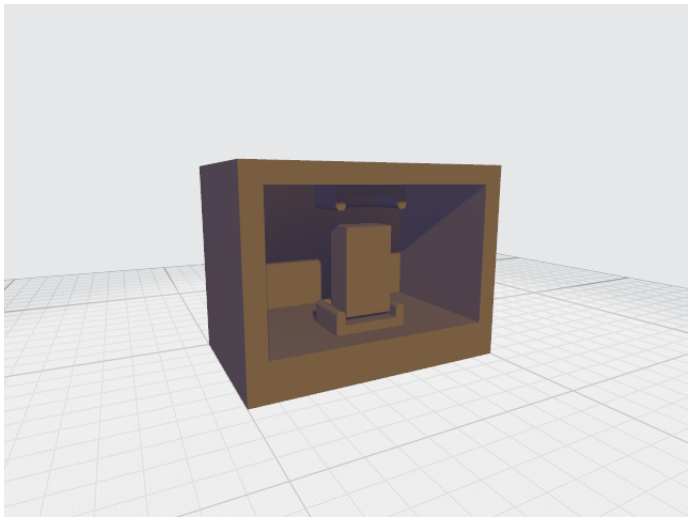

Figure 4.14: mesoSPIM reproduction loaded with a sample

spect the previous discussed contraints, are found with a Matlab script, in Fig. 4.15 the previous discussed poses are showed.
To finish with this procedure we must now move the robot to the previous found cartesian positions and read the corresponding joint values[3].
Once this has been done we can copy the joints coordinates of all the target points in the interface parameters
This procedure highlighted a major weakness of the Interface: Due to the fact that e.Do must be calibrated every time is switched on, taught points must be taken again every time we power on the robot. Calibration performed a previous time may differ slightly from the current, then teaching could be imprecise.

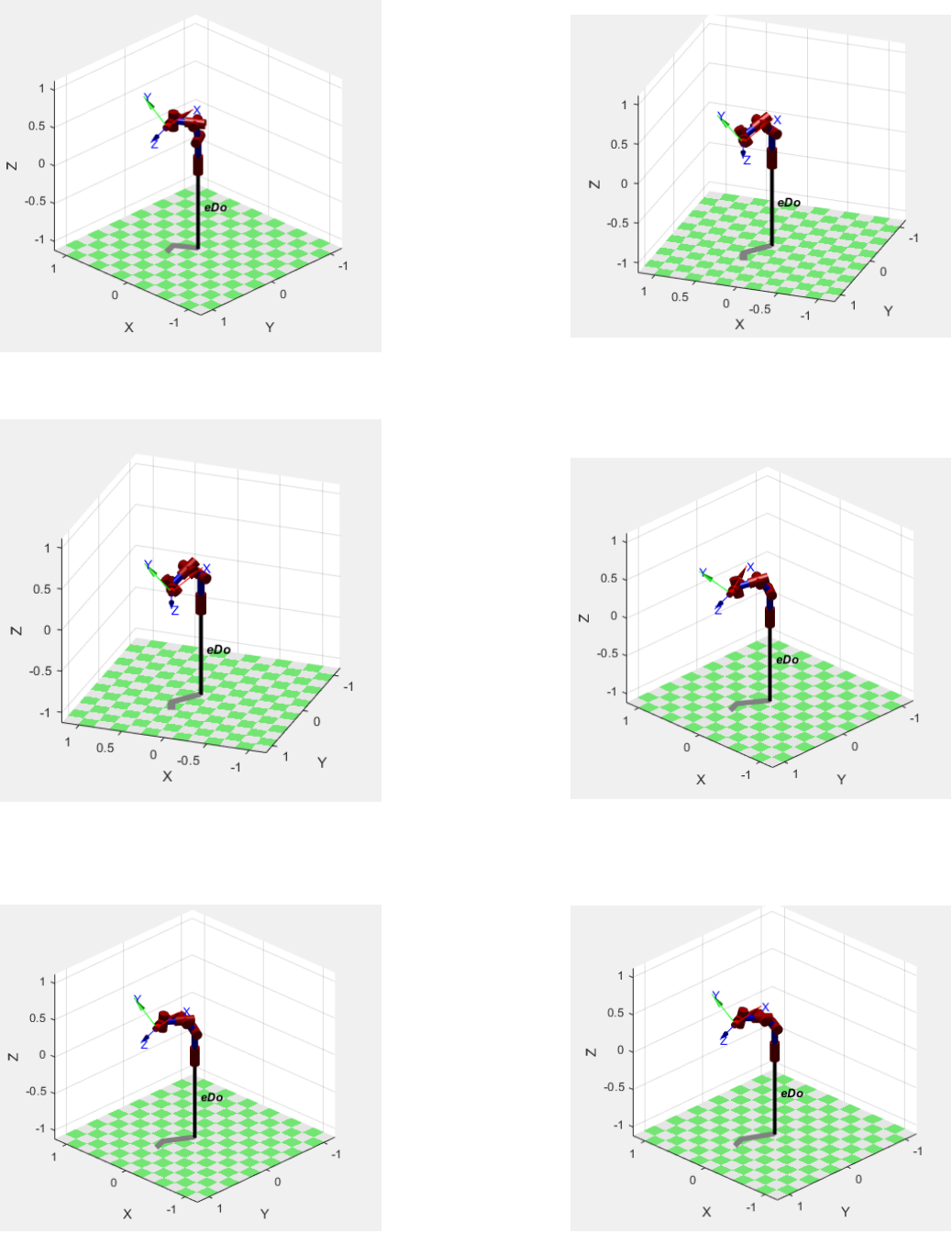

Figure 4.15: From Left to Right and Top to Bottom : Home, Sample-Via, Sample, Holder, mesoSPIM proximity and deliver poses

4.7.3 Results evaluation

Using *edo_recovery* node's methods the feedback signals coming from all of the six joints were collected.

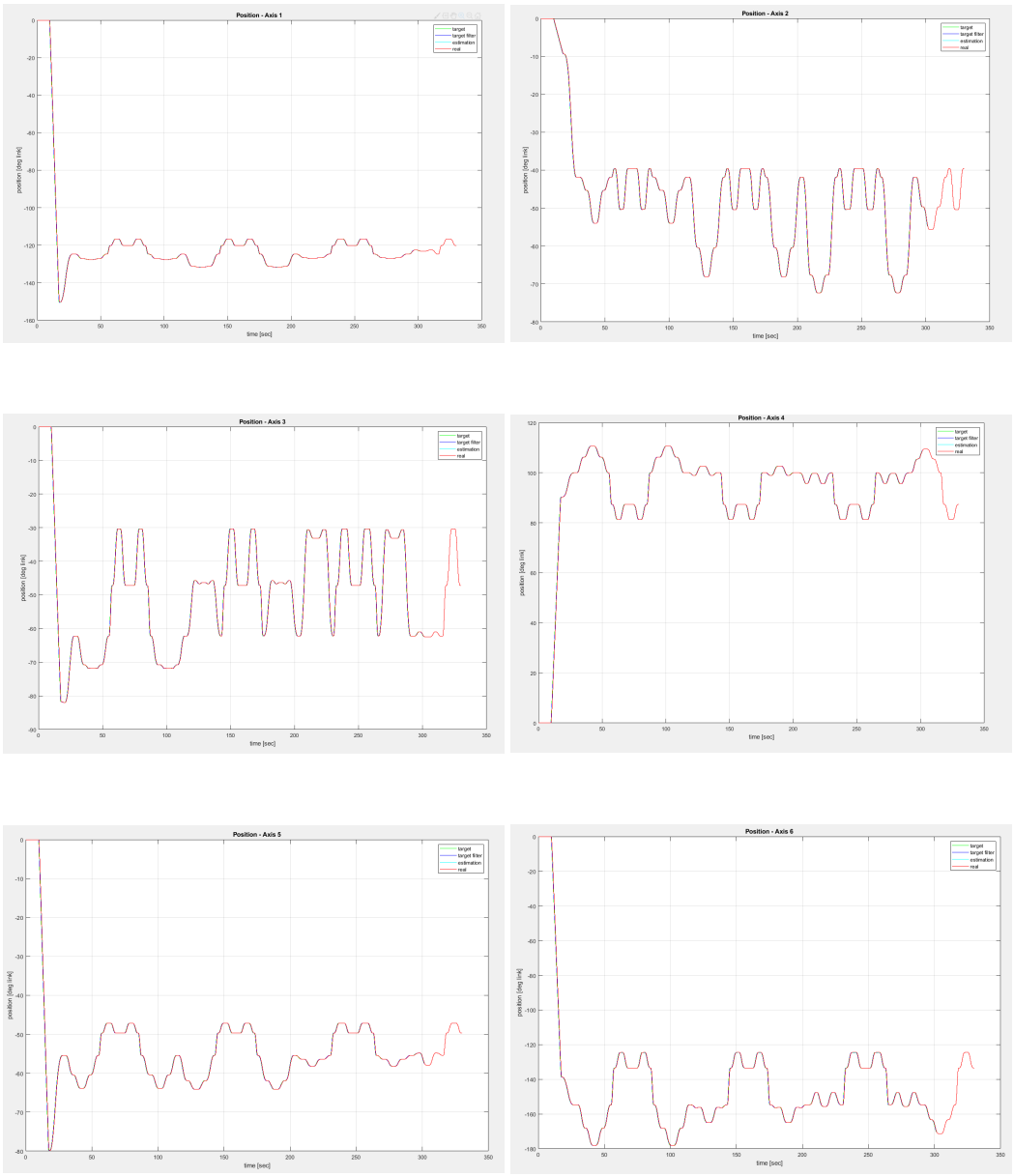

Figure 4.16: Axis target and measured values during the task simulated

In Fig. 4.16 target and measured values are showed for each one of the six joint variables. To evaluate the quality of the trajectory execution we

define the error as the difference between the joint's target positions signals $J_i^{targ}(t)$ and the joint's measured positions signals $J_i^{meas}(t)$:

$$e_i(t) = J_i^{targ}(t) - J_i^{meas}(t) \tag{4.1}$$

Once these errors have been found we can calculate the Root Mean Square Level for the error signals ($RMSE_i^J$) as follows:

$$RMSE_i^J = \sqrt{\frac{1}{N}\sum_{n=1}^{N} e_i(n)^2} \tag{4.2}$$

where N is the length of the error signals.
In Table 4.4 the results obtained are reported.

If we observe the target and measured signals we notice that the measured

Joint	RMSE [deg]
1	0.8931
2	0.6626
3	1.0108
4	0.6880
5	0.5341
6	1.1126

Table 4.4: RMSE for joint variables

are delayed with respect to the targets, Fig. 4.17, this delay is taken in account in the RMSE. If we repeat the computation of the RMSE but with a smaller interval where both the signals are constant, Fig. 4.18, we obtain new measures for the RMSE without the delay error; we will call this ${}^*RMSE_i^J$, in Table 4.5 its values are reported. Values obtained for ${}^*RMSE_i^J$ are really small for a robot of this type, precision of e.Do is actually worst due to mechanical backlash happening in the joints that are not seen by motors encoders.

Starting from the target and measured values of the joint variables, using Comau kinematics function, the target and measured values have been found for the End Effector pose. In Fig. 4.19 the End Effector trajectory, measured and target, during the task, is showed. If we repeat the computation of the ${}^*RMSE$ for the pose coordinates of the end-effector we obtain the values showed in Table 4.6.

Joint	*RMSE [deg]
1	0.0109
2	0.0064
3	0.0075
4	0.0057
5	0.0016
6	0.0127

Table 4.5: RMSE without delay

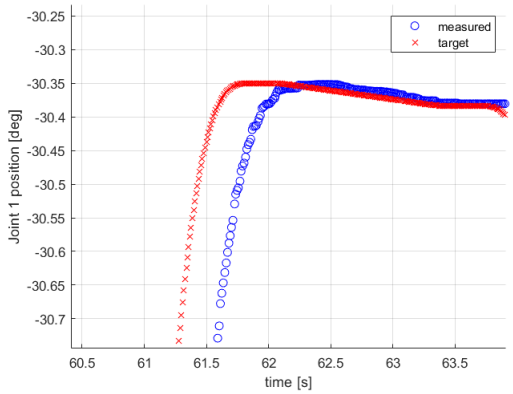

Figure 4.17: Target and Measured signal for Joint 1

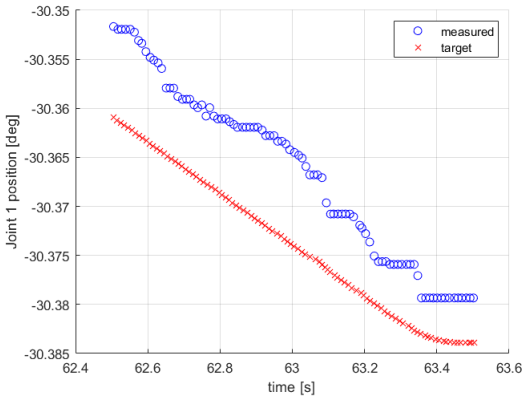

Figure 4.18: Target and Measured signal for Joint 1

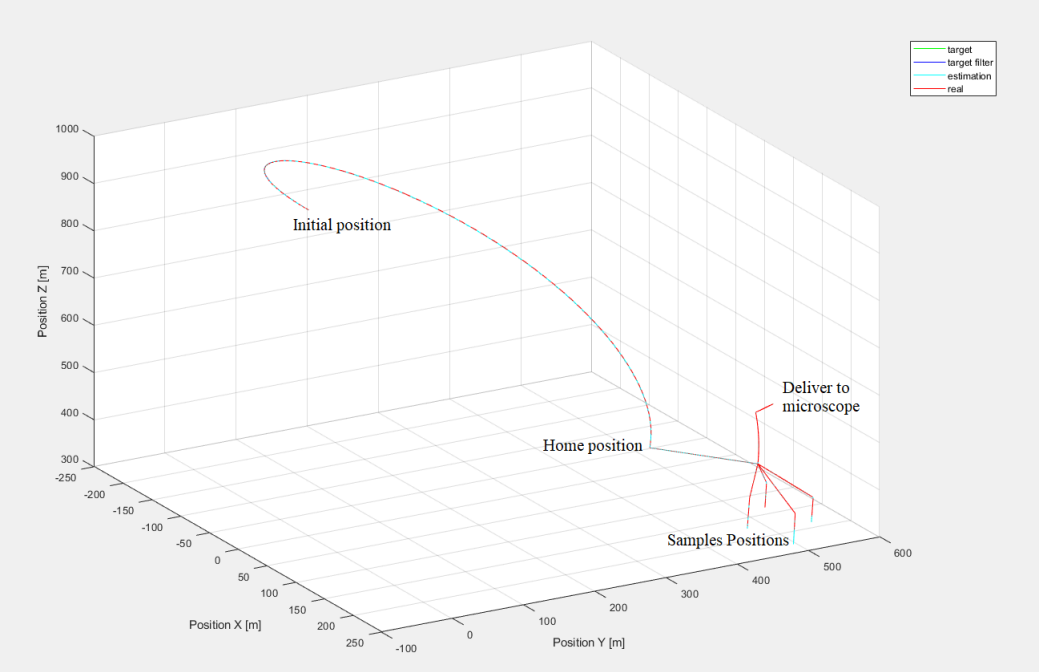

Figure 4.19: End Effector position in cartesian space, measured and target values

Pose Coordinate	*RMSE [deg]
x	0.0932 mm
y	0.0394 mm
z	0.0229 mm
angle x	0.0095 deg
angle y	0.0039 deg
angle z	0.0096 deg

Table 4.6: RMSE for pose coordinates

To prove consistency of the trajectory generated with the pose contraints, defined in Chapter 4.3.1, images from the demo are showed in Fig. 4.20.

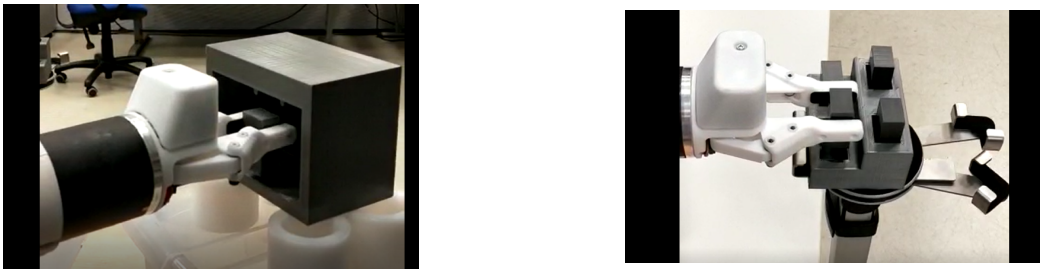

Figure 4.20: Pose Constraints

Of importance are also the x and y angles orientations during the path; we want them to change as little as possible in order not to damage the materials inside the samples. In Fig. 4.21 and Fig. 4.22 variations about x and y base reference frame axis angles are reported. As it can be seen, except for the initial part, in which the robots moves from the initial position, where it is extended, to the home position, the variation of the orientations angles around x and y are limited.

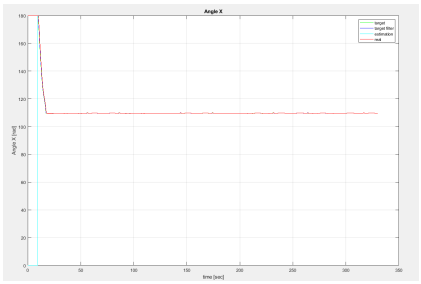

Figure 4.21: End Effector x-orientation measured and target values

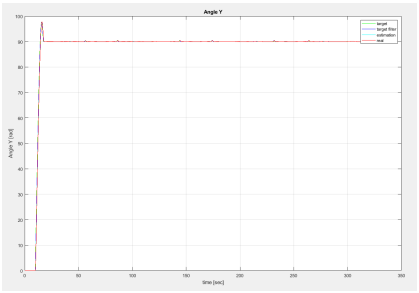

Figure 4.22: End Effector y-orientation measured and target values

Chapter 5

Conclusions and future works

During this work of thesis we aimed at developing an interface capable of letting e.Do coordinate its actions with a high precision instrumentation such as mesoSPIM digital microscope.
The interface had to work on top of the lower level control algorithms developed by Comau, as it concerns e.Do, and by the mesoSPIM Initiative, for what concerns mesoSPIM.
Since the microscope was not available to study we investigated deeply the e.Do system and focus the project development on this robot.
e.Do has been studied in it's hardware and software architecture, typical robotics analysis, such as Direct Kinematics, Jacobian and Workspace studies, alongside with a detailed analysis of the software running e.Do have been produced.
Features of mesoSPIM microscope were abstracted from available resources, such as code repositories and CAD projects.
Exploiting ROS environment, in which hardware simulation and code spreading are easily implemented, a package to fulfill the task of performing autonomous analysis on different medical samples has been developed.
Due to the context in which the project has been carried out, that saw mostly e.Do as a protagonist, the interface from the side of the microscope has yet to be implemented and the resulting application developed is characterized by a general interaction behavior that in the future can be shaped both on mesoSPIM but also on any digital instrumentation that share common features with the microscope.
Future development for this project involve two different evolutions.
First we discuss future of e.Do itself. The robot proved to be capable of precise enough positioning suitable for the interaction with such delicate instrumentation. Control algorithms that drives e.Do guarantee good reference tracking and accuracy in positioning within millimeters. In future releases

collision detection will be implemented to help the robot operate in delicate environments.
Major weaknesses of e.Do resulted to be two: impossibility for the user to create personalized trajectories, and calibration procedure, that in case of applications that require teaching of target points, makes the set up procedure obsolete every time we switch off the robot.
To overcome these problems the following solutions are proposed: first to generate a personalized trajectory we could bypass *edo_core_pkg* and write joint reference signal directly on the serial communication. Private Inverse Kinematics and Trajectory planning functions by Comau could be replaced by open source algorithms available from the community. Then to overcome calibration problems we could substitute the teaching procedure with a Computer Vision system capable of deriving target points position from image processing.
As it concerns the future interface what has to be done is to hook it up to the mesoSPIM system. Test in medical labs in which mesoSPIM is present could be developed in the future to implement the interaction from the microscope side.

Chapter 6

Appendix A: Robotic Operating System, ROS

6.1 Introduction

ROS is an open-source meta-operating system designed to be used for robotics applications. It provides functionalities typical of an operating system such as hardware abstraction and low level device control. Moreover it provides tools and libraries to build, write and run code spread across multiple machines [**16**].
The primary goal of ROS is to allow code reuse among the community. ROS is a distributed framework of processes that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed.
The core of ROS is licensed under the standard three-clause BSD license. This is a very permissive open license that allows for reuse in commercial and closed source products.
"Over the past several years ROS has grown to include a large community of users worldwide. Historically, the majority of the users were in research labs, but increasingly is seen adopted in the commercial sector, particularly in industrial and service robotics" [**17**].
These features, and many others, make ROS the perfect tool for the study and design of complex robotic systems. Dealing with complex robotic systems means to deal with many different technological aspects. The support from the community allows the user to integrate resources from others in his project, relieving him from the development of the system in its whole.

6.2 Three levels of concepts

ROS structure has been designed and divided in three sections of concepts, [**18**]:

- File System
- Computation Graph
- Community

6.2.1 File System

The file system level concepts describe the resource that are typically found on a ROS system storage, [**18**].

- Package: A package is the main unit to organize software in ROS. Generally speaking a package can contain runtime processes (nodes), libraries, configuration files or any other resource needed to be organized together.
- Metapackage: A metapackage is a specialized package meant to represent a group of related packages.
- Package Manifest: A manifest defines metadata about a package such as: author name's, license of the package, dependencies and so on.
- Repository: A repository is a collection of packages that share the same version control (VCS) .
- Message Type: It describes a type of message, defining its data structure.
- Service Type: It describes a type of service, defining the request and response data structure of the service.

6.2.2 Computation Graph

The computation graph is the p2p network configuration of ROS processes, [**19**].
Communication occurs through direct links, called topics, between different terminals, called nodes.

- Node: A node, generally speaking, is a process performing computation. In a complex robotic system we can address different tasks to different nodes, in such a way we can decompose the code of the whole in smaller modular pieces, [**19**].

- Master: When setting up a ROS environment, occasionally spread across multiple machines, the ROS Master provides name registration and lookup for the rest of the computational Graph elements, this allows nodes to locate each other in the network and exchange messages. The Master API network functionality is implemented through XML-RCP. XML-RCP is a remote procedure call (RCP) protocol which uses XML to encode its calls and HTTP as a transport mechanism. In XML-RPC a client performs an RCP by sending an HTTP request to a server that implements XML-RCP and receives the HTTP response. The API provides methods for register and *unregister* services, subscription and publish will. Clients use libraries like rospy and roscpp to access methods of the Master API.
  ROS Master provides a service to allow nodes to be aware of the presence of other nodes with whom to communicate. When a node has data to publish, it communicates the master the will to send a message on a specific topic, Fig. 6.1. When a node wants to subscribe to a topic it communicates to the Master its interest, Fig. 6.2. By matching this informations the master notifies publisher and subscriber about reciprocal existence, Fig. 6.3, [**20**].

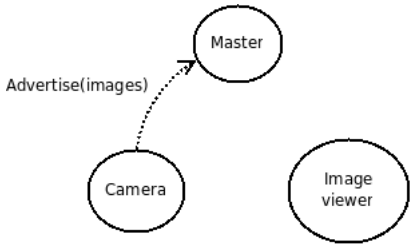

Figure 6.1: Camera advertise Master the will to publish on topic Images, [**20**]

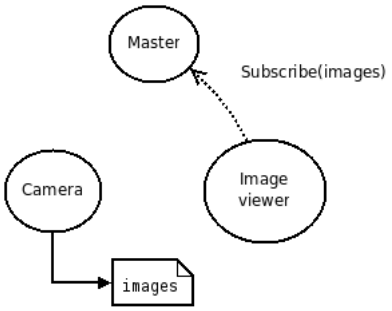

Figure 6.2: Image viewer advertise Master the will to subscribe on topic Images, [**20**]

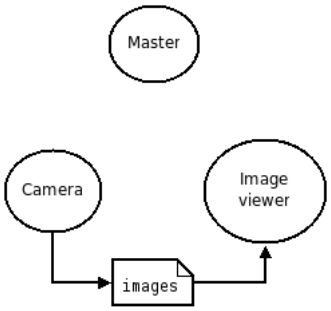

Figure 6.3: Camera and Image can now start exchanging messages on Images topic, [**20**]

- Topic: Topics are named unidirectional buses over which nodes exchange data. Subscribe and Publish methods have an anonymous semantic. When publishing on a topic it's not needed to specify whom is addressed the message. Doing so the production of information is decoupled from its consumption and the resultant code is loosely coupled. Message transport is implemented through TCPROS and UDPROS a TCP/UDP-based transport that uses pre-existing TCP/UDP connections. When a node is willing to use UDPROS if the interested node doesn't support it, the connection can fallback to TCPROS, this happens dynamically. Topics are strongly tied by the message type they can transport. When defining a topic one must specify the message type it will carry. Subscribers must perform a consistency check with the topic type before connecting, instead the master does not enforce a type consistency among the publishers, [**21**].

6.2.3 Community

As mentioned before, code reuse and knowledge sharing among users play a fundamental role in developing a system in ROS. The main concepts related to the community level are the following [**18**]:

- Distributions: ROS distributions are collection of versioned stack available for install. In this work of thesis the Kinetic Kame distribution has been used. When implementing a new package from the community in a project the user must ensure compatibility between the distribution in use and the package.
- Repositories: ROS repositories are collection of code available to download coming from the same institution.
- ROS Wiki: The ROS community Wiki is the main forum for documenting information about ROS. Upon registration every user can contribute to the wiki writing new articles, updating or correcting old ones.
- Mailing List: It is the main channel to be updated about new software releases about ROS.
- Ros Answer: A Q&A forum to support users from the community.

Chapter 7

Appendix B: Robotics

Among the various types of robot that have been designed we are interested in investigating the manipulator category.
Manipulator robots are, generally speaking, a chain of rigid bodies, called links. The connections between links are called joints, they can be revolute or prismatic, whether the constraint is on an axis of revolution or a direction of linear motion.
In manipulators characterized by an open-shaped chain one end is constrained to a base and the other one serves as tool (or end-effector).
The motion of the tool in space is obtained composing those of the joints. The manipulator has number of DOF (degree of freedom), depending on the number of joints and their disposition, which define the robot posture.

7.1 Rotation Matrix

If we consider two reference frames, like those in Fig. 7.1, and a point P in space we can write that:

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{7.1}$$

$$\mathbf{p}' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} \tag{7.2}$$

Since we are representing the same point we can equal (7.1) and (7.2) :

$$p = p'_x \cdot \mathbf{x}' + p'_y \cdot \mathbf{y}' + p'_z \cdot \mathbf{z}' = \begin{bmatrix} \mathbf{x}' & \mathbf{y}' & \mathbf{z}' \end{bmatrix} \mathbf{p}' \tag{7.3}$$

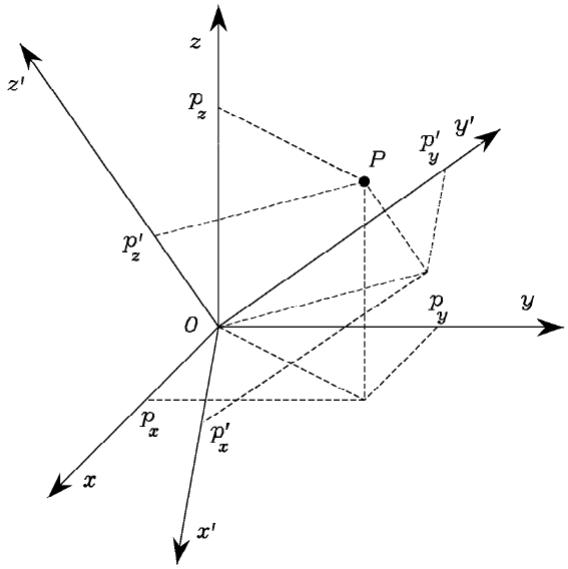

Figure 7.1: Point P expressed in two different reference frames, [**7**]

Where $\begin{bmatrix}\mathbf{x}' & \mathbf{y}' & \mathbf{z}'\end{bmatrix}$ is a matrix whose columns are the versors of O-$x'y'z'$ represented in reference frame O-xyz, we call this matrix the rotation matrix between the two reference frames, and it is found as follows:

$$R = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} = \begin{bmatrix} \mathbf{x}'^T\mathbf{x} & \mathbf{y}'^T\mathbf{x} & \mathbf{z}'^T\mathbf{x} \\ \mathbf{x}'^T\mathbf{y} & \mathbf{y}'^T\mathbf{y} & \mathbf{z}'^T\mathbf{y} \\ \mathbf{x}'^T\mathbf{z} & \mathbf{y}'^T\mathbf{z} & \mathbf{z}'^T\mathbf{z} \end{bmatrix} \tag{7.4}$$

We now can write:

$$\mathbf{p} = R \cdot \mathbf{p}' \tag{7.5}$$

We will illustrate two major properties of matrix R:

- Inverse-Transpose: Since columns of R represents orthonormal vectors we can write:

  $$R \cdot R^T = \mathbf{I} \tag{7.6}$$

  Pre-multiplying by R^{-1} we get:

  $$R^T = R^{-1} \tag{7.7}$$

- Composition: Suppose we have three reference frames O-xyz, O-$x'y'z'$, O-$x''y''z''$ and a point in space P. We know the following rotation matrix R_1^0 and R_2^1 , which represent the transformation from reference frames

O-xyz to O-$x'y'z'$ and from O-$x'y'z'$ to O-$x'y''z''$ respectively.
We know that:

$$\mathbf{p} = R_1^0 \cdot \mathbf{p}' \tag{7.8}$$

And that:

$$\mathbf{p}' = R_2^1 \cdot \mathbf{p}'' \tag{7.9}$$

Substituting (7.9) in (7.8) we obtain:

$$\mathbf{p} = R_1^0 \cdot \mathbf{p}' = R_1^0 \cdot R_2^1 \cdot \mathbf{p}'' \tag{7.10}$$

which is the property of composition of rotations matrix.
We will call further on:

$$R_2^0 = R_1^0 \cdot R_2^1 \tag{7.11}$$

7.2 Homogeneous Transformation

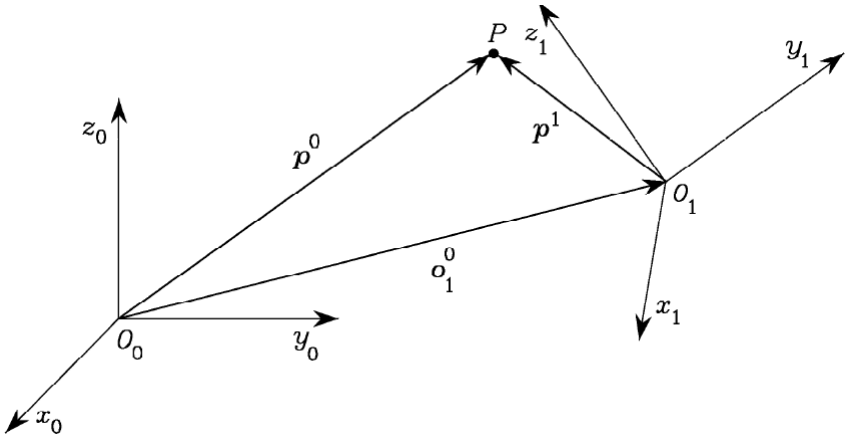

Figure 7.2: Point P expressed in two different coordinates frames,[**7**]

Consider the Fig. 7.2, P is a point in space and R_0 and R_1 two reference frame. $\mathbf{p^1}$ is the vector expressing the position of P with respect to R_1 and , similarly, $\mathbf{p^0}$ is the vector expressing the position of P with respect to R_0. Let $\mathbf{o_1^0}$ be the vector who express the position of O_1, origin of R_1, with respect to R_0.
We can say that :

$$\mathbf{p^0} = \mathbf{o_1^0} + R_1^0 \cdot \mathbf{p^1} \tag{7.12}$$

To achieve a compact representation of this expression we must introduce the homogeneous representation of a vector, we simply add a fourth row equal

to one and obtain:

$$\tilde{\mathbf{p}} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{7.13}$$

Then the transformation expressed in (7.12), composed of a rotation and a translation, can be written in a single matrix named homogeneous transformation:

$$T_1^0 = \begin{bmatrix} R_1^0 & \mathbf{o_1^0} \\ \mathbf{0^T} & 1 \end{bmatrix} \tag{7.14}$$

Using (7.14) in (7.12) we get:

$$\begin{aligned} \tilde{\mathbf{p^0}} &= T_1^0 \cdot \mathbf{p^1} \\ \tilde{\mathbf{p^1}} &= T_0^1 \cdot \mathbf{p^0} \end{aligned} \tag{7.15}$$

$$T_0^1 = (T_1^0)^{-1} = \begin{bmatrix} R_0^1 & -R_0^1\mathbf{o_1^0} \\ \mathbf{0^T} & 1 \end{bmatrix}$$

7.3 Joints

In robotics "a joint is the mechanical connection between two consecutive links" [**7**].
Joints can be active, if they are actuated by a motor, or else passive. Among the various types of joints the most common are revolute joints and prismatic joints, Fig. 7.3. In revolute joints motion happens on an axis of revolution, the joint variable associated is the angle around that axis.
In prismatic joints instead motion happens as a linear sliding motion, the joint variable associated is then the displacement along that direction.

7.4 Direct Kinematics

Dealing with direct kinematics means to compute the pose of the end-effector as a function of the joint variables, whose values correspond to the positions of each joint. As shown in Fig. 7.4 the pose of the end-effector with respect to the reference frame O_b-$x_by_bz_b$ is defined by the position and orientation of the end-effector reference frame. The direct kinematic function is then the

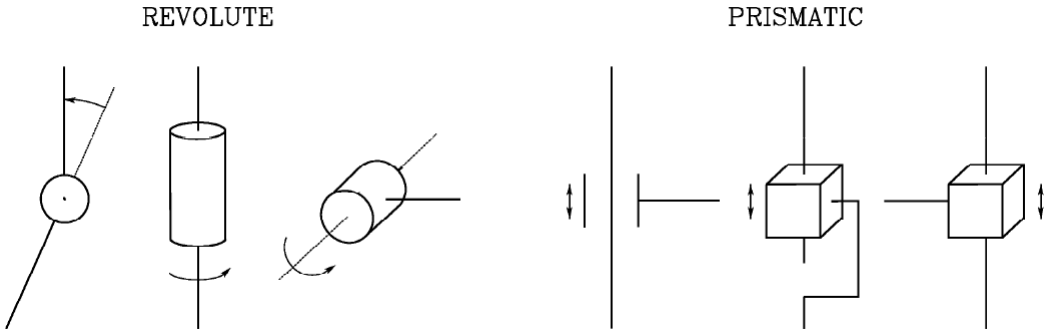

Figure 7.3: Revolute (left) and prismatic (right) joint type, [**7**]

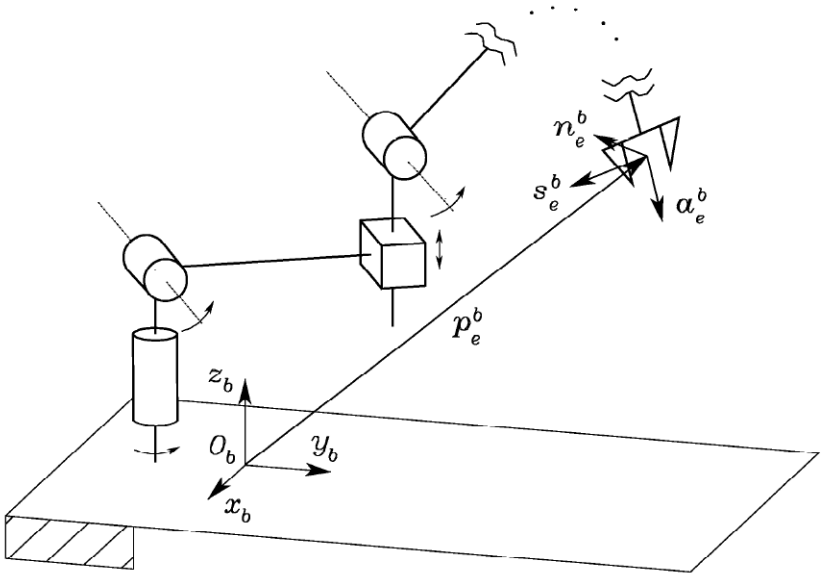

Figure 7.4: Description of the end-effector reference frame position,[**7**]

homogeneous transformation between the tool frame and the base frame:

$$T_e^b = \begin{bmatrix} \mathbf{n_e^b}(\mathbf{q}) & \mathbf{s_e^b}(\mathbf{q}) & \mathbf{a_e^b}(\mathbf{q}) & \mathbf{p_e^b}(\mathbf{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.16}$$

In an open chain manipulator with n joints and $n+1$ link we can fix one reference frame to each one of the links. Being the links connected by joints, the relation between two consecutive links is given by a homogenous transformation with the joint position as variable. Composing for each joint we obtain the direct kinematic function, Fig. 7.5.

$$T_n^0(\mathbf{q}) = T_1^0(q_1) \cdot T_2^1(q_2)...T_n^{n-1}(q_n) \tag{7.17}$$

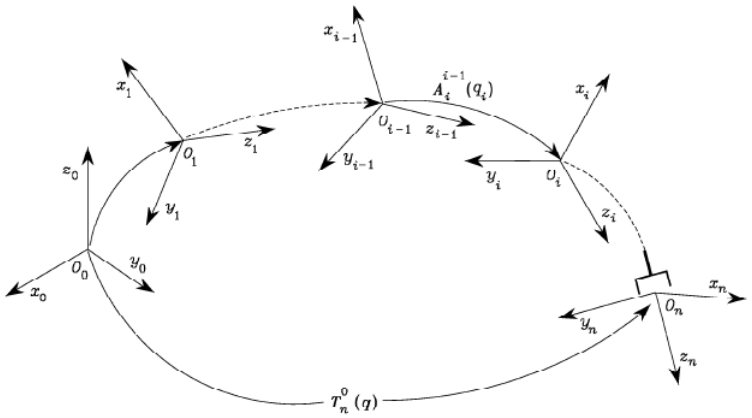

Figure 7.5: Composition of transformation between links in open chain manipulator,[**7**]

7.5 Denevit Hartenberg convention

Denavit Hartenberg (DH) convention provides a standard way for choosing how to attach the reference frame on each link.
Using this convention allows to easily represent the mechanical structure of the robot and, as a consequence, find its direct kinematics.

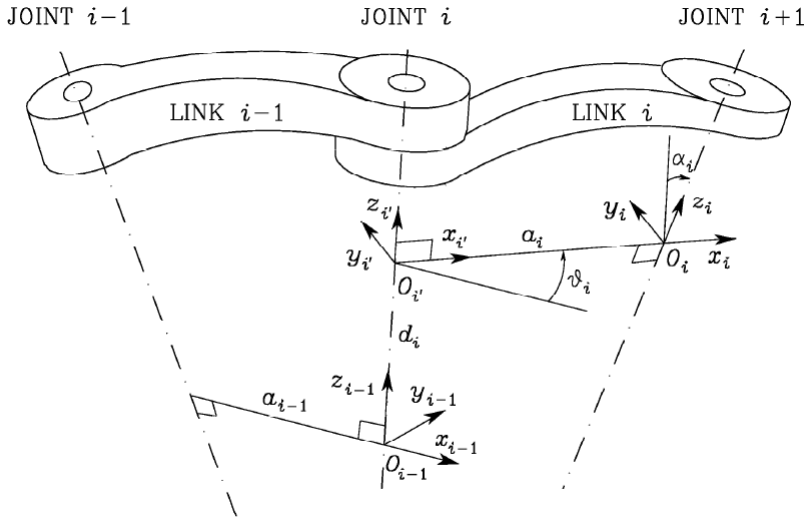

Figure 7.6: DH parameters in between two consecutive links,[**7**]

In Fig. 7.6 the axis i refers to the axis of the joint connection between link $i-1$ and link i. We use the DH convention to define the reference frame

of the link i, the rules are the following, [**7**]:

- Put z_i along the axis of joint $i+1$.
- Locate O_i at the intersection of the axis z_i with the common normal to axes z_{i-1} and z_i, and O_i' at the intersection with the common normal with axis z_{i-1}.
- Choose x_i on the common normal to axes z_{i-1} and z_i, with direction from joint i to joint $i+1$.
- Pick y_i adopting the rule of the right hand, in order to obtain an orthogonal base.

The position and orientation between two consecutive frames is then specified by four parameter:

- a_i: distance between O_i and O_i'.
- d_i: coordinate of O_i' along z_{i-1}.
- α_i: angle between axes z_i and z_{i-1} about axis x_i.
- θ_i: angle between axis x_i and x_{i-1} about axis z_i.

What can be said about these four parameter is that if the joint is revolute, then the joint variable is θ_i, else, if the joint is prismatic, the joint variable is d_i. Also α_i and a_i are always constant no matter the type of joint.
In both type of joints we then have one parameter which is the joint variable and the other three that are constant , determined by the mechanical structure of the robot.
If the convention is respected and the four parameters are known for each of the manipulator joints, the procedure to find the coordinate transformation between each of the consecutive links becomes systematic.
In fact we know that chosen a frame aligned with Frame $i-1$ to get to frame i we must perform the following transofrmations:

- Translate the frame aligned with frame $i-1$ by d_i along the axis z_i and rotate it by θ_i about axis z_{i-1}, in such a way we align the current frame with the frame i'. We obtain the following transformation:

$$T_{i'}^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.18}$$

- Translate the frame aligned with i' by a_i along x'_i and rotate it by α_i about axis x_i, in such a way we have aligned the current frame with frame i. We obtain the following transformation :

$$T_i^{i'} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & c_{\alpha_i} & s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.19}$$

- Multiplicating these two transformations we compose them and obtain a single transformation with the following expression:

$$T_i^{i-1} = T_{i'}^{i-1} \cdot T_i^{i'} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.20}$$

Once the transformation matrix is obtained for each one of the joints, it is possible to compose them and obtain the expression of the direct kinematic function.

7.6 Workspace

The set of point in space that the end-effector can reach is called workspace. The workspace is a closed, connected and finite volume limited by surfaces [**7**].
The boundaries surfaces of the workspace depend on the physical structure of the robot, the links length, and the mechanical limits of joints.
For a manipulator with n joints the workspace is the geometric locus of all the points obtained by the sole part of position of the direct kinematics varying the joint variables in their limits.

$$\mathbf{p_e} = \mathbf{p_e}(\mathbf{q}) \quad q_{im} \leq q_i \leq q_{iM} \tag{7.21}$$

7.7 Differential Kinematics

Differential kinematics studies the relationships between the end-effector velocity and the joints velocities. These connections are expressed by the Jacobian matrix.
When describing velocity of the end-effector we must distinguish between

angular velocity and linear velocity.
The concept of angular velocity of a rigid body comes with two different approaches. A first one in which angular velocity is a vector whose components are the elements of vector ω, physical angular velocity vector. A second approach consists in considering the rate of change of the angular variables chosen for representing the end effector orientation. When it comes to describe linear speed of the end-effector instead the representation is unique and obtained by the rate of change of the x, y and z coordinates of the tool reference frame origin.
Accordingly to the type of velocity, among those explained above, that we want to relate with joints velocities we obtain two different types of Jacobian matrix. A first one is called Analytical Jacobian and relates linear velocities and angular velocities, accordingly to the second approach mentioned before, with joints velocities. A second one, the Geometric Jacobian, that relates linear and angular velocities, expressed by the vector ω, to joints velocities.

7.7.1 Geometric Jacobian: Angular Speed

$$\omega_{\mathbf{e}} = \mathbf{J_o}(\mathbf{q}) \cdot \dot{\mathbf{q}} \tag{7.22}$$

Where $\omega_{\mathbf{e}}$ is the angular velocity vector of the end effector reference frame. Geometric jacobian, depending on the type of joint, whether is revolute or prismatic, assume the following expression:

- Prismatic: $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
- Revolute: $\mathbf{z_{i-1}}$ expressed in base reference frame

7.7.2 Geometric Jacobian: Linear Speed

$$\dot{\mathbf{p}}_{\mathbf{e}} = \sum_{i=1}^{n} \frac{\partial \mathbf{p_e}}{\partial q_i} \dot{q}_i = \sum_{i=1}^{n} \mathbf{J}_{Pi} \dot{q}_i \tag{7.23}$$

Where $\dot{\mathbf{p}}_{\mathbf{e}}$ is the linear velocity of the end effector reference frame.

- Prismatic: $\mathbf{J}_{Pi} = \mathbf{z_{i-1}}$
- Revolute: $\mathbf{J}_{Pi} = \mathbf{z_{i-1}} \times (\mathbf{p_e} - \mathbf{p_{i-1}})$

7.8 Singolarity

Generally the Jacobian is function of the joint variables. Those configurations that makes the Jacobian rank deficient are called singular configuration, $\mathbf{q}_{sing}$. Those configuration are then such that:

$$det(\mathbf{J}(\mathbf{q}_{sing})) = 0 \tag{7.24}$$

When in a singular configuration the following phenomena occur,[**7**]:

- There is a loss of mobility of the manipulator, it's not possible to impose any arbitrary trajectory to the end effector.
- Small velocities of the end effector can result in large velocities at the joints.
- The solutions of the inverse kinematic problem can be infinite

For an anthropomorphic arm with spherical wrist it is possible to study separately the arm singularities and the wrist singularities under the hypothesis that the center of the tool frame coincides with the intersection point of the wrist axes. This procedure is called singolarity decoupling, [**7**].

7.8.1 Wrist Singolarity

Wrist singularity happens when $q_5 = 0$, Fig. 7.7. In this configuration the

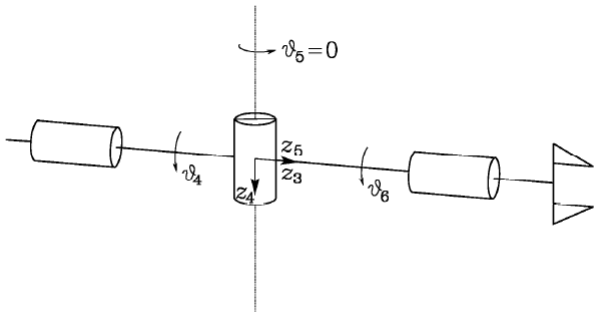

Figure 7.7: Wrist singolarity for the spherical wrist, [**7**]

tool frame can't rotate about the axis perpendicular both to z_5 and z_4.

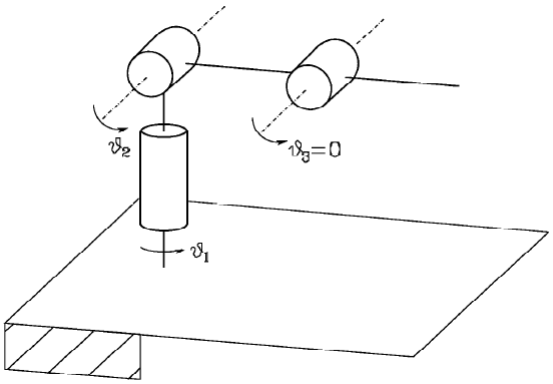

Figure 7.8: Elbow singolarity for the antropomorphic arm, [**7**]

7.8.2 Arm Singolarities

For an antropomorphic robot, arm singularities are of two types. A first one happens when $q_3 = 0$, it is called elbow singularity, Fig. 7.8. In this configuration the arm is outstretch and can't extend anymore. A second singularity, named shoulder singularity, Fig. 7.9 happens when the wrist center is along the z_0 axis. In this configuration the tool frame can't translate along the z_3 direction. This specific situation does not happen for a specific value of a joint, like the previous ones, but for a set of combination of q_2 and q_3 depending on the link length.

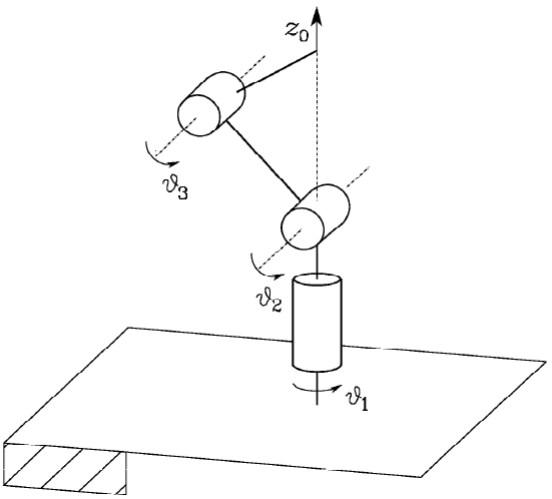

Figure 7.9: Shoulder singolarity for the antropomorphic arm, [**7**]

Chapter 8

Appendix C: Numerical Results

8.1 Direct Kinematic Function

8.1.1 Expression of T_3^0

$$T_3^0(:,1)= \begin{pmatrix} \cos(q_1)\cos(q_2)\cos(q_3) - \cos(q_1)\sin(q_2)\sin(q_3) \\ \cos(q_2)\cos(q_3)\sin(q_1) - \sin(q_1)\sin(q_2)\sin(q_3) \\ -\cos(q_2)\sin(q_3) - \cos(q_3)\sin(q_2) \\ 0 \end{pmatrix}$$

$$T_3^0(:,2)= \begin{pmatrix} \sin(q_1) \\ -\cos(q_1) \\ 0 \\ 0 \end{pmatrix}$$

$$T_3^0(:,3)= \begin{pmatrix} -\cos(q_1)\cos(q_2)\sin(q_3) - \cos(q_1)\cos(q_3)\sin(q_2) \\ -\cos(q_2)\sin(q_1)\sin(q_3) - \cos(q_3)\sin(q_1)\sin(q_2) \\ \sin(q_2)\sin(q_3) - \cos(q_2)\cos(q_3) \\ 0 \end{pmatrix}$$

$$T_3^0(:,4)= \begin{pmatrix} \frac{421\cos(q_1)\cos(q_2)}{2000} \\ \frac{421\cos(q_2)\sin(q_1)}{2000} \\ \frac{337}{1000} - \frac{421\sin(q_2)}{2000} \\ 1 \end{pmatrix}$$

8.1.2 Expression of T_6^3

$$T_6^3(:,1)= \begin{pmatrix} \cos(q_4)\ \cos(q_5)\ \cos(q_6) - \sin(q_4)\ \sin(q_6) \\ \cos(q_4)\ \sin(q_6) + \cos(q_5)\ \cos(q_6)\ \sin(q_4) \\ \cos(q_6)\ \sin(q_5) \\ 0 \end{pmatrix}$$

$$T_6^3(:,2)= \begin{pmatrix} -\cos(q_6)\ \sin(q_4) - \cos(q_4)\ \cos(q_5)\ \sin(q_6) \\ \cos(q_4)\ \cos(q_6) - \cos(q_5)\ \sin(q_4)\ \sin(q_6) \\ -\sin(q_5)\ \sin(q_6) \\ 0 \end{pmatrix}$$

$$T_6^3(:,3)= \begin{pmatrix} -\cos(q_4)\ \sin(q_5) \\ -\sin(q_4)\ \sin(q_5) \\ \cos(q_5) \\ 0 \end{pmatrix}$$

$$T_6^3(:,4)= \begin{pmatrix} -\frac{349\ \cos(q_4)\ \sin(q_5)}{2000} \\ -\frac{349\ \sin(q_4)\ \sin(q_5)}{2000} \\ \frac{349\ \cos(q_5)}{2000} + \frac{67}{250} \\ 1 \end{pmatrix}$$

8.1.3 Expression of T_6^0

$$\mathbf{n}^0 = \begin{pmatrix} \sin(q_6)\,(\cos(q_4)\,\sin(q_1) - \sin(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3))) + \\ + \cos(q_6)\,(\cos(q_5)\,(\sin(q_1)\,\sin(q_4) + \cos(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3))) - \\ - \sin(q_5)\,(\cos(q_1)\,\cos(q_2)\,\sin(q_3) + \cos(q_1)\,\cos(q_3)\,\sin(q_2))) \\ \\ -\sin(q_6)\,(\cos(q_1)\,\cos(q_4) - \sin(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \cos(q_2)\,\cos(q_3)\,\sin(q_1))) - \\ - \cos(q_6)\,(\cos(q_5)\,(\cos(q_1)\,\sin(q_4) + \cos(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \\ - \cos(q_2)\,\cos(q_3)\,\sin(q_1))) + \sin(q_5)\,(\cos(q_2)\,\sin(q_1)\,\sin(q_3) + \cos(q_3)\,\sin(q_1)\,\sin(q_2))) \\ \\ \sin(q_4)\,\sin(q_6)\,(\cos(q_2)\,\sin(q_3) + \cos(q_3)\,\sin(q_2)) - \cos(q_6)\,(\sin(q_5)\,(\cos(q_2)\,\cos(q_3) - \sin(q_2)\,\sin(q_3)) + \\ + \cos(q_4)\,\cos(q_5)\,(\cos(q_2)\,\sin(q_3) + \cos(q_3)\,\sin(q_2))) \\ \\ 0 \end{pmatrix}$$

$$\mathbf{s}^0 = \begin{pmatrix} \cos(q_6)\,(\cos(q_4)\,\sin(q_1) - \sin(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3))) - \sin(q_6)\,(\cos(q_5)\,(\sin(q_1)\,\sin(q_4) + \\ + \cos(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3))) - \sin(q_5)\,(\cos(q_1)\,\cos(q_2)\,\sin(q_3) + \\ + \cos(q_1)\,\cos(q_3)\,\sin(q_2))) \\ \\ \sin(q_6)\,(\cos(q_5)\,(\cos(q_1)\,\sin(q_4) + \cos(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \\ - \cos(q_2)\,\cos(q_3)\,\sin(q_1))) + \sin(q_5)\,(\cos(q_2)\,\sin(q_1)\,\sin(q_3) + \cos(q_3)\,\sin(q_1)\,\sin(q_2))) - \\ - \cos(q_6)\,(\cos(q_1)\,\cos(q_4) - \sin(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \cos(q_2)\,\cos(q_3)\,\sin(q_1))) \\ \\ \sin(q_6)\,(\sin(q_5)\,(\cos(q_2)\,\cos(q_3) - \sin(q_2)\,\sin(q_3)) + \cos(q_4)\,\cos(q_5)\,(\cos(q_2)\,\sin(q_3) + \cos(q_3)\,\sin(q_2))) + \\ + \cos(q_6)\,\sin(q_4)\,(\cos(q_2)\,\sin(q_3) + \cos(q_3)\,\sin(q_2)) \\ \\ 0 \end{pmatrix}$$

$$\mathbf{a}^0 = \begin{pmatrix} -\sin(q_5)\,(\sin(q_1)\,\sin(q_4) + \cos(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3))) - \\ - \cos(q_5)\,(\cos(q_1)\,\cos(q_2)\,\sin(q_3) + \cos(q_1)\,\cos(q_3)\,\sin(q_2)) \\ \\ \sin(q_5)\,(\cos(q_1)\,\sin(q_4) + \cos(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \cos(q_2)\,\cos(q_3)\,\sin(q_1))) - \\ - \cos(q_5)\,(\cos(q_2)\,\sin(q_1)\,\sin(q_3) + \cos(q_3)\,\sin(q_1)\,\sin(q_2)) \\ \\ \cos(q_4)\,\sin(q_5)\,(\cos(q_2)\,\sin(q_3) + \cos(q_3)\,\sin(q_2)) - \cos(q_5)\,(\cos(q_2)\,\cos(q_3) - \sin(q_2)\,\sin(q_3)) \\ \\ 0 \end{pmatrix}$$

$$\mathbf{p}^0 = \begin{pmatrix} \frac{421\,\cos(q_1)\,\cos(q_2)}{2000} - \frac{349\,\sin(q_5)\,(\sin(q_1)\,\sin(q_4) + \cos(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3)))}{2000} - \\ - \frac{349\,\cos(q_5)\,(\cos(q_1)\,\cos(q_2)\,\sin(q_3) + \cos(q_1)\,\cos(q_3)\,\sin(q_2))}{2000} - \frac{67\,\cos(q_1)\,\cos(q_2)\,\sin(q_3)}{250} - \frac{67\,\cos(q_1)\,\cos(q_3)\,\sin(q_2)}{250} \\ \\ \frac{421\,\cos(q_2)\,\sin(q_1)}{2000} + \frac{349\,\sin(q_5)\,(\cos(q_1)\,\sin(q_4) + \cos(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \cos(q_2)\,\cos(q_3)\,\sin(q_1)))}{2000} - \\ - \frac{349\,\cos(q_5)\,(\cos(q_2)\,\sin(q_1)\,\sin(q_3) + \cos(q_3)\,\sin(q_1)\,\sin(q_2))}{2000} - \frac{67\,\cos(q_2)\,\sin(q_1)\,\sin(q_3)}{250} - \frac{67\,\cos(q_3)\,\sin(q_1)\,\sin(q_2)}{250} \\ \\ \frac{67\,\sin(q_2)\,\sin(q_3)}{250} - \frac{67\,\cos(q_2)\,\cos(q_3)}{250} - \frac{421\,\sin(q_2)}{2000} - \\ - \frac{349\,\cos(q_5)\,(\cos(q_2)\,\cos(q_3) - \sin(q_2)\,\sin(q_3))}{2000} + \frac{349\,\cos(q_4)\,\sin(q_5)\,(\cos(q_2)\,\sin(q_3) + \cos(q_3)\,\sin(q_2))}{2000} + \frac{337}{1000} \\ \\ 1 \end{pmatrix}$$

8.2 Geometric Jacobian

8.2.1 Orientation Jacobian

$${}^{0}\vec{z}_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$${}^{0}\vec{z}_1 = \begin{pmatrix} -\sin(q_1) \\ \cos(q_1) \\ 0 \\ 1 \end{pmatrix}$$

$${}^{0}\vec{z}_2 = \begin{pmatrix} -\sin(q_1) \\ \cos(q_1) \\ 0 \\ 1 \end{pmatrix}$$

$${}^{0}\vec{z}_3 = \begin{pmatrix} -\cos(q_1)\,\cos(q_2)\,\sin(q_3) - \cos(q_1)\,\cos(q_3)\,\sin(q_2) \\ -\cos(q_2)\,\sin(q_1)\,\sin(q_3) - \cos(q_3)\,\sin(q_1)\,\sin(q_2) \\ \sin(q_2)\,\sin(q_3) - \cos(q_2)\,\cos(q_3) \\ 1 \end{pmatrix}$$

$${}^{0}\vec{z}_4 = \begin{pmatrix} \sin(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3)) - \cos(q_4)\,\sin(q_1) \\ \cos(q_1)\,\cos(q_4) - \sin(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \cos(q_2)\,\cos(q_3)\,\sin(q_1)) \\ -\sin(q_4)\,(\cos(q_2)\,\sin(q_3) + \cos(q_3)\,\sin(q_2)) \\ 1 \end{pmatrix}$$

$$
{}^0\vec{z}_5 = \begin{pmatrix}
-\sin(q_5)\,(\sin(q_1)\,\sin(q_4) + \\
+\cos(q_4)\,(\cos(q_1)\,\cos(q_2)\,\cos(q_3) - \cos(q_1)\,\sin(q_2)\,\sin(q_3))) - \\
-\cos(q_5)\,(\cos(q_1)\,\cos(q_2)\,\sin(q_3) + \\
+\cos(q_1)\,\cos(q_3)\,\sin(q_2)) \\
\\
\sin(q_5)\,(\cos(q_1)\,\sin(q_4) + \\
+\cos(q_4)\,(\sin(q_1)\,\sin(q_2)\,\sin(q_3) - \cos(q_2)\,\cos(q_3)\,\sin(q_1))) - \\
-\cos(q_5)\,(\cos(q_2)\,\sin(q_1)\,\sin(q_3) + \\
+\cos(q_3)\,\sin(q_1)\,\sin(q_2)) \\
\\
\cos(q_4)\,\sin(q_5)\,(\cos(q_2)\,\sin(q_3) + \\
+\cos(q_3)\,\sin(q_2)) - \cos(q_5)\,(\cos(q_2)\,\cos(q_3) - \\
-\sin(q_2)\,\sin(q_3)) \\
\\
1
\end{pmatrix}
$$

8.2.2 Position Jacobian

$$
{}^0\vec{z}_0 \times {}^0\vec{o}_6 = \begin{pmatrix}
\frac{67\,\cos(q_2)\,\sin(q_1)\,\sin(q_3)}{250} - \frac{421\,\cos(q_2)\,\sin(q_1)}{2000} + \frac{67\,\cos(q_3)\,\sin(q_1)\,\sin(q_2)}{250} \\
\frac{421\,\cos(q_1)\,\cos(q_2)}{2000} - \frac{67\,\cos(q_1)\,\cos(q_2)\,\sin(q_3)}{250} - \frac{67\,\cos(q_1)\,\cos(q_3)\,\sin(q_2)}{250} \\
0
\end{pmatrix}
$$

$$
{}^0\vec{z}_1 \times {}^0\vec{o}_6 = \begin{pmatrix}
-\cos(q_1)\left(\frac{421\,\sin(q_2)}{2000} + \frac{67\,\cos(q_2)\,\cos(q_3)}{250} - \frac{67\,\sin(q_2)\,\sin(q_3)}{250}\right) \\
-\sin(q_1)\left(\frac{421\,\sin(q_2)}{2000} + \frac{67\,\cos(q_2)\,\cos(q_3)}{250} - \frac{67\,\sin(q_2)\,\sin(q_3)}{250}\right) \\
\cos(q_1)\left(\frac{67\,\cos(q_1)\,\cos(q_2)\,\sin(q_3)}{250} - \frac{421\,\cos(q_1)\,\cos(q_2)}{2000} + \frac{67\,\cos(q_1)\,\cos(q_3)\,\sin(q_2)}{250}\right) + \\
+\sin(q_1)\left(\frac{67\,\cos(q_2)\,\sin(q_1)\,\sin(q_3)}{250} - \frac{421\,\cos(q_2)\,\sin(q_1)}{2000} + \frac{67\,\cos(q_3)\,\sin(q_1)\,\sin(q_2)}{250}\right)
\end{pmatrix}
$$

$$
{}^0\vec{z}_2 \times {}^0\vec{o}_6 = \begin{pmatrix}
-\cos(q_1)\left(\frac{67\,\cos(q_2)\,\cos(q_3)}{250} - \frac{67\,\sin(q_2)\,\sin(q_3)}{250}\right) \\
-\sin(q_1)\left(\frac{67\,\cos(q_2)\,\cos(q_3)}{250} - \frac{67\,\sin(q_2)\,\sin(q_3)}{250}\right) \\
\cos(q_1)\left(\frac{67\,\cos(q_1)\,\cos(q_2)\,\sin(q_3)}{250} + \frac{67\,\cos(q_1)\,\cos(q_3)\,\sin(q_2)}{250}\right) + \sin(q_1)\left(\frac{67\,\cos(q_2)\,\sin(q_1)\,\sin(q_3)}{250} + \frac{67\,\cos(q_3)\,\sin(q_1)\,\sin(q_2)}{250}\right)
\end{pmatrix}
$$

$$
{}^0\vec{z}_3 \times {}^0\vec{o}_6 = \begin{pmatrix}
(\cos(q_2)\,\sin(q_1)\,\sin(q_3) + \cos(q_3)\,\sin(q_1)\,\sin(q_2))\,(\frac{67\,\cos(q_2)\,\cos(q_3)}{250} - \frac{67\,\sin(q_2)\,\sin(q_3)}{250}) - \\
-(\frac{67\,\cos(q_2)\,\sin(q_1)\,\sin(q_3)}{250} + \frac{67\,\cos(q_3)\,\sin(q_1)\,\sin(q_2)}{250})\,(\cos(q_2)\,\cos(q_3) - \sin(q_2)\,\sin(q_3)) \\
\\
(\frac{67\,\cos(q_1)\,\cos(q_2)\,\sin(q_3)}{250} + \frac{67\,\cos(q_1)\,\cos(q_3)\,\sin(q_2)}{250})\,(\cos(q_2)\,\cos(q_3) - \sin(q_2)\,\sin(q_3)) - \\
-(\cos(q_1)\,\cos(q_2)\,\sin(q_3) + \cos(q_1)\,\cos(q_3)\,\sin(q_2))\,(\frac{67\,\cos(q_2)\,\cos(q_3)}{250} - \frac{67\,\sin(q_2)\,\sin(q_3)}{250}) \\
\\
(\cos(q_1)\,\cos(q_2)\,\sin(q_3) + \cos(q_1)\,\cos(q_3)\,\sin(q_2))\,(\frac{67\,\cos(q_2)\,\sin(q_1)\,\sin(q_3)}{250} + \frac{67\,\cos(q_3)\,\sin(q_1)\,\sin(q_2)}{250}) - \\
-(\frac{67\,\cos(q_1)\,\cos(q_2)\,\sin(q_3)}{250} + \frac{67\,\cos(q_1)\,\cos(q_3)\,\sin(q_2)}{250})\,(\cos(q_2)\,\sin(q_1)\,\sin(q_3) + \cos(q_3)\,\sin(q_1)\,\sin(q_2))
\end{pmatrix}
$$

$$
{}^0\vec{z}_4 \times {}^0\vec{o}_6 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}
$$

$$
{}^0\vec{z}_4 \times {}^0\vec{o}_6 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}
$$

8.3 Workspace Surfaces

$$S_1=\left(\begin{array}{c} \frac{783\,\cos(v)\,\cos(u)}{1000} \\ \frac{783\,\cos(u)\,\sin(v)}{1000} \\ \frac{337}{1000}-\frac{783\,\sin(u)}{1000} \end{array}\right)$$

$$S_2=\left(\begin{array}{c} \frac{421\,\cos\left(\frac{23\,\pi}{180}\right)\,\cos(v)}{2000}-\frac{229\,\cos\left(\frac{23\,\pi}{180}\right)\,\cos(v)\,\sin(u)}{400}-\frac{229\,\sin\left(\frac{23\,\pi}{180}\right)\,\cos(v)\,\cos(u)}{400} \\ \frac{421\,\cos\left(\frac{23\,\pi}{180}\right)\,\sin(v)}{2000}-\frac{229\,\cos\left(\frac{23\,\pi}{180}\right)\,\sin(v)\,\sin(u)}{400}-\frac{229\,\sin\left(\frac{23\,\pi}{180}\right)\,\cos(u)\,\sin(v)}{400} \\ \frac{229\,\sin\left(\frac{23\,\pi}{180}\right)\,\sin(u)}{400}-\frac{229\,\cos\left(\frac{23\,\pi}{180}\right)\,\cos(u)}{400}-\frac{421\,\sin\left(\frac{23\,\pi}{180}\right)}{2000}+\frac{337}{1000} \end{array}\right)$$

$$S_3=\left(\begin{array}{c} \frac{421\,\cos(v)\,\cos(u)}{2000}+\frac{609\,\cos(\frac{761\,\pi}{1800})\,(\cos(\frac{23\,\pi}{180})\,\cos(v)\,\sin(u)+\sin(\frac{23\,\pi}{180})\,\cos(v)\,\cos(u))}{2000}- \\ -\frac{609\,\sin(\frac{761\,\pi}{1800})\,(\cos(\frac{23\,\pi}{180})\,\cos(v)\,\cos(u)-\sin(\frac{23\,\pi}{180})\,\cos(v)\,\sin(u))}{2000}-\frac{67\,\cos(\frac{23\,\pi}{180})\,\cos(v)\,\sin(u)}{250}- \\ -\frac{67\,\sin(\frac{23\,\pi}{180})\,\cos(v)\,\cos(u)}{250} \\ \\ \frac{421\,\cos(u)\,\sin(v)}{2000}+ \\ +\frac{609\,\cos(\frac{761\,\pi}{1800})\,(\cos(\frac{23\,\pi}{180})\,\sin(v)\,\sin(u)+\sin(\frac{23\,\pi}{180})\,\cos(u)\,\sin(v))}{2000}- \\ -\frac{609\,\sin(\frac{761\,\pi}{1800})\,(\cos(\frac{23\,\pi}{180})\,\cos(u)\,\sin(v)-\sin(\frac{23\,\pi}{180})\,\sin(v)\,\sin(u))}{2000}-\frac{67\,\cos(\frac{23\,\pi}{180})\,\sin(v)\,\sin(u)}{250}- \\ -\frac{67\,\sin(\frac{23\,\pi}{180})\,\cos(u)\,\sin(v)}{250} \\ \\ \frac{67\,\sin(\frac{23\,\pi}{180})\,\sin(u)}{250}-\frac{67\,\cos(\frac{23\,\pi}{180})\,\cos(u)}{250}-\frac{421\,\sin(u)}{2000}+ \\ +\frac{609\,\cos(\frac{761\,\pi}{1800})\,(\cos(\frac{23\,\pi}{180})\,\cos(u)-\sin(\frac{23\,\pi}{180})\,\sin(u))}{2000}+ \\ +\frac{609\,\sin(\frac{761\,\pi}{1800})\,(\cos(\frac{23\,\pi}{180})\,\sin(u)+\sin(\frac{23\,\pi}{180})\,\cos(u))}{2000}+\frac{337}{1000} \end{array}\right)$$

List of Figures

Bibliography

[1] Steven Wyatt, (2019), *IFR World Robotics Outlook 2019*, 17 Sep 2019.

[2] Catlin, Dave, and John Woollard, (2014), *"Educational robots and computational thinking."* Proceedings of 4th International Workshop Teaching Robotics, Teaching with Robotics, 5th International Conference Robotics in Education.

[3] *https://www.comau.com/en/about-comau/history*, 19 Sep 2019.

[4] *https://www.comau.com/en/about-comau/global-presence*, 19 Sep 2019.

[5] e.Do 6 axes technical sheet, (14 Sep 2019), *https://edo.cloud/documents/*.

[6] *wiki.ros.org/rosserial*, 14 Sep 2019.

[7] B.Siciliano, L. Sciavicco, L.Villani, G. Oriolo, (2009), *Robotics, Modelling planning and control*, Springer.

[8] *https://shop.edo.cloud/robots/37-gripper-pick-up-the-pace.html*, 17 Sep 2019.

[9] P.I. Corke, *"Robotics, Vision & Control"*, Springer 2017, ISBN 978-3-319-54413-7.

[10] Goyal, Khushdeep and Sethi, Davinder, (2010), *An analytical method to find workspace of a robotic manipulator.*

[11] *github.com/mesoSPIM*, 19 Sep 2019.

[12] *http://www.labonthecheap.com/index.php/2019/03/29/mesospim-an-open-source-light-sheet-microscope-for-imaging-in-cleared-tissue/*, 19 Sep 2019.

[13] *https://roslibpy.readthedocs.io/en/stable/*, 17 Sep 2019.

[14] *https://wiki.ros.org/rosbridge_suite*, 18 Sep 2019.

[15] Mark Lutz, (2003), *Learning Python*, O'Reilly & Associates.

[16] *http://wiki.ros.org/ROS/Introduction*, 20 Sep 2019.

[17] *https://www.ros.org/is-ros-for-me/*, 20 Sep 2019.

[18] *http://wiki.ros.org/ROS/Concepts*, 20 Sep 2019.

[19] *http://wiki.ros.org/Nodes*, 20 Sep 2019.

[20] *http://wiki.ros.org/Master*, 20 Sep 2019.

[21] *http://wiki.ros.org/Topics*, 20 Sep 2019.