UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN CONTROL SYSTEM ENGINEERIGN

"Features for non-collaborative robots"

Relatore: Prof. / Dott Ruggero Carli

Laureando: Federico Gessato

Correlatore: Prof Artur Pereira e Dot Eurico Pedrosa

ANNO ACCADEMICO 2021 – 2022

Data di laurea 17/10/2022

Acknowledgement

Prima di addentrarsi nella discussione della tesi vorrei ringraziare le persone che mi hanno seguito e supportato in questo lavoro.
Nello specifico vorrei ringraziare il Dottor Eurico Pedrosa, il Professor Artur Pereira dell'università di Aveiro ed il Professor Ruggero Carli dell'università di Padova per avermi seguito nel progetto e per la loro continua disponibilità.
Volevo inoltre ringraziare la mia fidanzata per avermi supportato moralmente e per essere stata presente anche nei momenti più difficili.
Infine un grazie anche ai miei genitori i quali mi hanno permesso di arrivare fino a qui e a tutti gli amici e alle persone che mi hanno accompagnato durante il mio corso di studi.

Abstract

Over the past few years, the idea of human-robot cooperation has been evolving, and current research in the robotics sector is demonstrating the promise of human-robot collaboration. From these premises also industrial manipulator benefited by outmatching their older version, focused on executing the planned trajectory with the maximum efficiency and without caring about the operator safety, in favour of the new available technologies.
Implementing this new concept, they have the ability to be hand driven by regular users, simplifying this task for those who operate or configure them. Results achieved by the reaction of the manipulator to the external forces followed by a proper control of the robots dynamics in order to move or not according with the given input.
Despite the several advantages of the collaborative robots the complete transition from their older counterparts is not easy as it seems. Given that the replacement involves excessive costs ,sometimes unbearable by the companies, and a sensible production of wastes the main objective of this dissertation arises: implementing a series of feature to the standard robotic manipulators that could be useful in an industrial environment considering as final goal to have a human interaction with the machine even if the robot is not suited for that.
A simulated environment was built to validate all research and test the solutions on a virtual Panda Arm manipulator. To corroborate the implemented methods,several experiences have been performed to compare the designed solution to the more sophisticated impedance control approaches seen in collaborative robots and the results appeared as successful.

Contents

List of Figures

Chapter 1

Introduction

In the 1996 J. Edward Colgate and Michael Peshkin, professors at the Northwestern University, invented the cobots: robots conceived in order to physically interact with a human in a workspace [1]. Since then the cooperative robots achieved great results in several fields due the great potential behind their concept and, still now, a lot of researches are working for expand their purpose and optimize them.

Their older counterparts (common manipulators) instead are though to work in order to achieve their goal in the less time possible and in the most efficient way for a better productivity and get those results thanks only to an earlier proper setting and planning which usually done prior by trained stuff. Those characteristics and the fact that often they are projected in order to work in a specific workspace, free from obstacles and ignoring any external collision make really hard the physical cooperation with a human operator.

Cobot instead, and more specifically the case of cobotics arms, since are developed in a different way, have the capability to be hand driven by the users reacting to external forces and moves in the direction that it's being pushed or pulled. In that way is possible for the operator to move the robot according to the trajectories allowing the realization of jobs too hard to be properly teached to a machine and too difficult or dangerous to be handled by a human operator by itself.

If in the beginning they were thought for automotive purposes, the first realized cobots were employed in the industries but with no internal source of driving force instead provided by the human being. In fact in that moment the function of these cobots was to allow computerized motion control, guiding a payload in cooperation with the worker [3].
Only in the 2004 the German company KUKA made LBR3, the first light cobot with self supply of the driving force and in the 2008 the Danish industry Universal Robots made UR5, the first cobots operating without barriers able to fully cooperate with a human safely.
Furthermore the progress made in this sector inspired the application in several other fields coming up with the built of Da Vinci, the first surgical cobot and many other implementations [2].

Nevertheless replacing part of the standard manipulator with cooperative ones is not that simple.

1.1 Motivations

The majority of the robots used in the assembly lines and in the industrial fields are for the majority robotic arms and manipulator. Actually a large portion of industries still rely on non-cooperative industrial manipulator models to perform their tasks and often, reprogramming these models with specific path plans for every kind of task, would result extremely time expensive, extremely hard to implement and could risk the companies to incur in a decrease of the productivity.

On the other hand the mayor replacement of the industrial manipulators with cobots imply not indifferent costs that not often are affordable by the corporations and more importantly would lend to huge production of wastes. Wastes that with a careless managing, considering that even nowadays is still unfortunately possible, could sensibly pollute the environment.

Here below the graph in figure 1.1, from a note website that collect info about the automation world [4], shows the fields in which non-cooperative robotics have been applied while the graph in

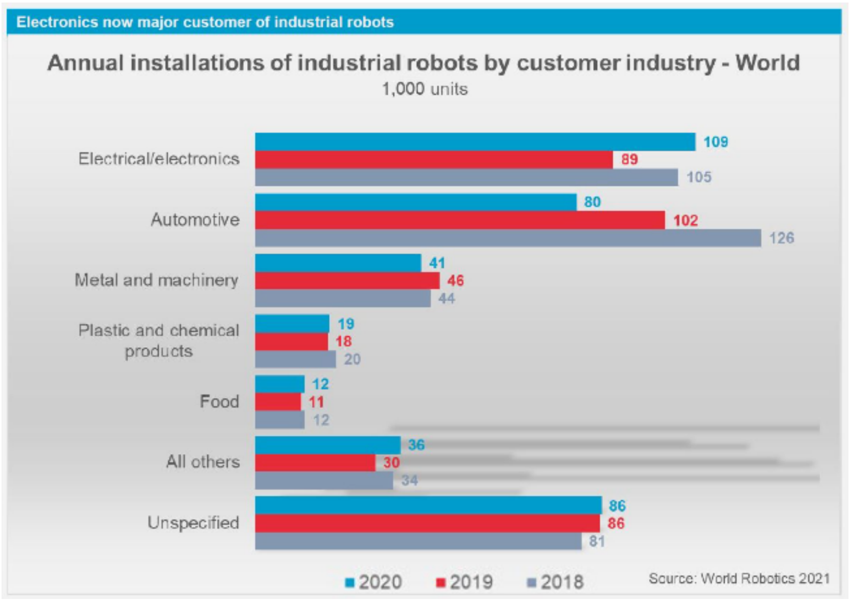


Figure 1.1:

figure 1.2 shows their annual installations.

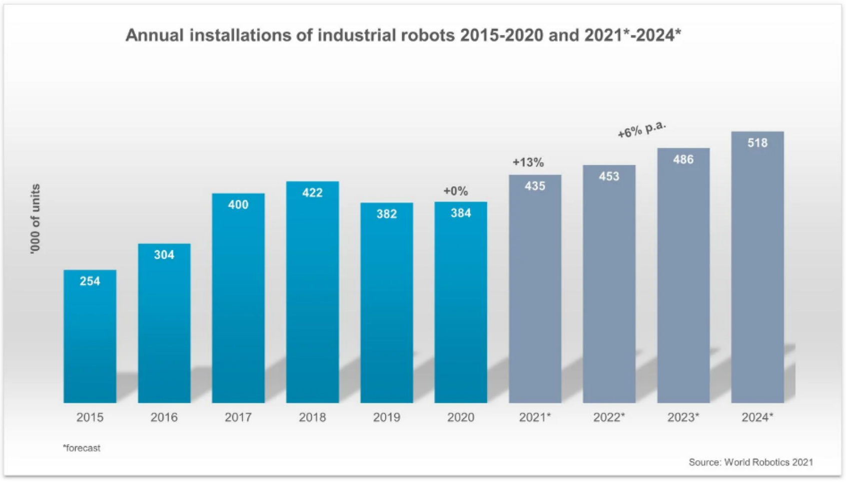


Figure 1.2:

1.2 Aim of the thesis

The aim of this thesis is to develop a series of features for the standard robotic manipulators, available in an industrial environment finalized to make the robot suited for a human interaction despite it is not initially provided for that.

The first topic that will be discussed regard "workspace limitations" that, as the name suggests, aims to limit the 3D workspace of the manipulator letting it avoid collisions with particulars environment conditions or simply to avoid to make undesired contacts with fragile or delicate section of its surrounding. A feature useful for narrow work-spaces, that could helps for sake of human safety considering the eventual workstation of the operator as the threshold taken into account and in cases where the objects to which the manipulator works require specific extra cares;

The second proposed feature concerns the "planning of sinusoidal trajectories" for the end-effector of the manipulator.
An implementation that could be useful and directly applicable for really simple tasks like the interaction with regular and geometric surfaces in order to achieve painting, gluing,welding and so on by following the designed trajectories. At the same time and more concretely it could also be used in order to have reference paths from which the operator should interact with the end-effector according to the needs of his goal;

In the end will be exposed the main feature that is the "force compliance control" for a manipulator not predisposed for it, core idea of this thesis. In order to apply this feature the force sensors of the robot will be exploited for the detect of the external forces, hypothetically applied by the operator, and decide after a proper filtering to follow them or not.
Application that actually is the basic interaction for cobots but not so foregone by the standard ones. Note that this task could be implemented also exploiting velocity sensors, with proper modification in the process, but it arbitrary has been decided to use forces and torques in order to have a more direct and aware communication between perceived signals and actuation.

1.3 Structure of the thesis

In the next chapters you will find the following structure:

- Chapter 2 start by exposing the theoretical knowledge behind robotic arm dynamics and the physics about them as mechanical systems. There are also exposed some theoretical examples presented in the context of robotics to better understand the formulations. All concept necessary for the comprehension of the dissertation and that later will be widely used in the implementation of the features.
- Chapter 3 will expose the software environment adopted in order to program and simulate the features and it also will point out the reason behind their choices.
- Chapter 4 explain specifically the realization of the feature explaining their concepts and idea, their algorithms and the problems found along the design process
- Chapter 5 will be the ending one showing the actual results achieved with the feature implementation through several plots and graphics. Furthermore it will state final statements about the projects and eventual consideration about the exposed features

Chapter 2

Theoretical introduction

For the implementation of the features explained earlier is necessary to first introduce the theory behind a robotic manipulator, its physics and its dynamics.
In order to do that mathematic models help to syntetize those behaviour with simple conventions and in this chapter they will be briefly exposed.
The book that came in help for summarizing all of the knowledge reported in this chapter is "Robotics: Modelling, Planning and Control" [5].

2.1 Base Concepts

Beginning with the topic of the dissertation, a manipulator is a robot which present a fixed base and a characterized by an arm that ensures mobility, a wrist that confers dexterity, and an end-effector that performs the task required by the robot([11]).

It can also be schematically represented from a mechanical viewpoint as a kinematic chain of rigid bodies, called links, connected by means of junctions capable of rotating, called revolute joints, or junctions capable of translating, called prismatic joints.

The end of the chain, constrained to the manipulator base and its links, with attached the part able to interact with the environment is called end-effector.
This is the mechanical piece of the manipulator developed and suited ad hoc for executing the assigned task. For this reason it is the main focus for the planning of the path to execute.

Moving to the surroundings, the area reachable by the end-effector is called workspace. It is often customary to distinguish between reachable workspace and dexterous workspace. The latter is the region that the origin of the end-effector frame can describe while attaining different orientations, while the former is the region that the origin of the end-effector frame can reach with at least one orientation.

Here in 2.1 is shown an example of a planar manipulator composed by three revolute joints.

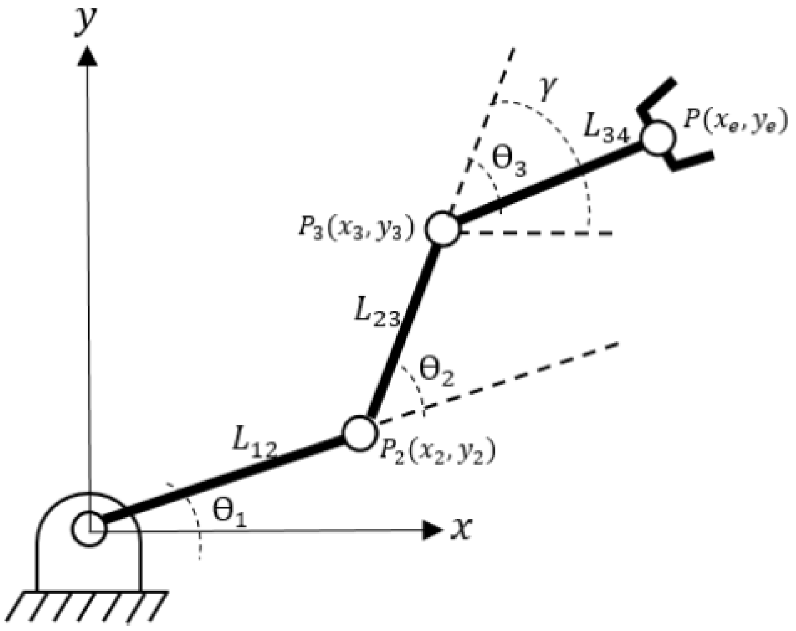


Figure 2.1: example of a planar manipulator and its variables

2.2 Direct Kinematics

As anticipated in order to manipulate an object in space, it is necessary to describe the end-effector position and orientation. The determination of those coordinate components given a reference frame can be expressed as a function of the mechanical structure and of the joint variables (angles or elongation of the joints) is called "Direct kinematics".

The first way for achieving this is to exploit the geometry of the manipulator.

For example in the case of the 3 links planar manipulator shown earlier(2.1) we would have, as final orientation of the end-effector, the sum of the angles and as for the position

$$\begin{cases} x = L_{12}cos(\theta_1) + L_{23}cos(\theta_1 + \theta_2) + L_{34}cos(\theta_1 + \theta_2 + \theta_3) \\ y = L_{12}sin(\theta_1) + L_{23}sin(\theta_1 + \theta_2) + L_{34}sin(\theta_1 + \theta_2 + \theta_3) \end{cases} \tag{2.1}$$

considering θ_3 with negative sign.

The second more generic way is the systematic method that instead allows to transform the base frame into the end-effector one through an homogeneous transformation matrix

$$T_e^b = \begin{bmatrix} R_e^b & P_e^b \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

where R_e^b is a 3 dimensional rotation matrix and P_e^b is a 3x1 translation vector both from the base frame towards the end-effector one.
With this method, in order to achieve the final result, is reasonable to consider first the relation between consecutive frames $T_i^{i-1}(q_i)$ and then obtain the overall description of the manipulator kinematics in a recursive way.
The equation

$$T_e^b = T_0^b * T_1^0(q_0)...T_i^{i-1}(q_i)...T_n^{n-1}(q_n) * T_e^n \tag{2.3}$$

is the obtained result considering a generic manipulator with n joints and $n+1$ links.
Note also that T_0^b and T_e^n are constant transformation matrices since allow the transformation between reference frame that are fixed regardless of the joints moving.

In order to do that some conventions are needed and the adopted are going to be the Denavit Hattemberg ones.

2.2.1 Denavit Hattemberg convention

The advantage of the used convention is that only 1 reference frame for each joint will be used and, instead of considering all of the 6 elementary transformation, it will be taken into account only 4. The procedure that is going to be exposed allow the transformation from the frame of a joint i-1 towards the one of joint i lets start assigning them correctly. To begin with

- the origin of the frame i-1 (O_{i-1}) have to be put on the intersection between the kinematic axes and the common normal.
- its z axis (z_{i-1}) have to be assigned along the kinematic axis (there are 2 possibilities)
- its x axis (x_{i-1}) have to be assigned along the common normal toward the next kinematic axis
- its y axis (y_{i-1}) have to be assigned according to the right hand rule

Here below in the figure (2.2) is shown an example.

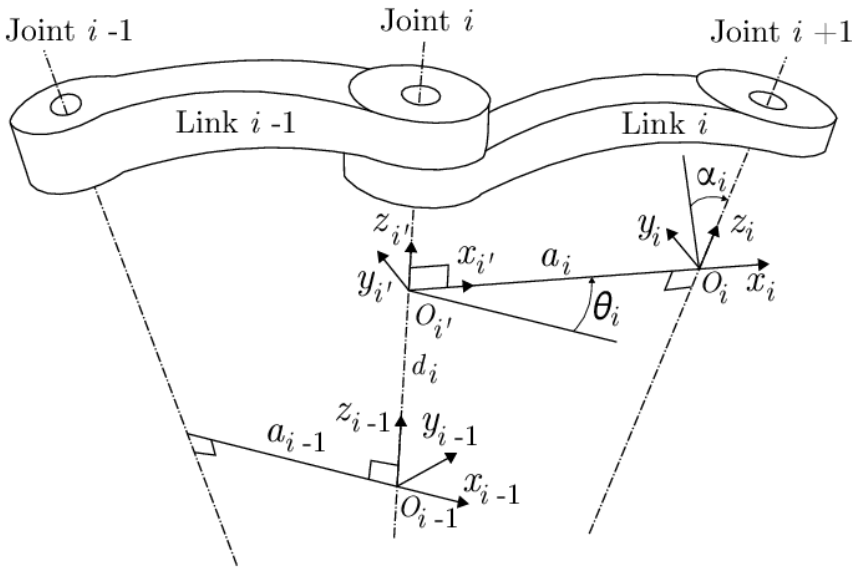


Figure 2.2: positioning of the frames according to Denavit Hattemberg convention

Once done all of the assignments it is possible to proceed with the transformation matrices.
In the following order:

1. compute the 3x1 translation vector $T_z(d_i)$ in order to reach the origin of the next reference frame along z_{i-1}

$$T_z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

2. compute the 3x3 rotation matrix $R_z(\theta_i)$ in order to rotate about z_{i-1} until x and y are aligned with the new frame

$$R_z(\theta_i) = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

3. compute the 3x1 translation vector $T_x(a_i)$ in order to reach the next kinematic axis translating

along x_i

$$T_x(a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

4. compute the 3x3 rotation matrix $R_x(\alpha_i)$ in order to rotate about x_{i-1} until z_{i-1} is parallel to z_i

$$R_x(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sin(\alpha_i) & 0 \\ 0 & sin(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Obtained all of the matrix is sufficient to concatenate them in order to get

$$T_i^{i-1} = T_z(d_i) * R_z(\theta_i) * T_x(a_i) * R_x(\alpha_i) \tag{2.8}$$

When this convention is used for the synthetic description of the manipulator, usually is given also a table with the four described parameters of each link(a_i, α_i, d_i, and θ_i) knowing the scheme of its axis disposition and is called the Denavit Hattemberg table.

2.3 Inverse Kinematics

If instead the goal is to find the joints components from the Cartesian coordinates of the end-effector the task that is dealt with is called "Inverse Kinematics".
The solution to this problem is of fundamental importance in order to transform the motion specifications, assigned to the end-effector in the operational space, into the corresponding joint space motions that allow execution of the desired movement.

It have to be stated that solve this problem it is not as simple as the forward case.
In fact dealing with inverse kinematics, comports multiple problems to occur since the equations to solve are in general non-linear and it could cause multiple solutions to exist or even infinite in the case of a redundant manipulator and even to present not admissible ones in view of the manipulator kinematic structure. Only if the given end-effector position and orientation belong to the manipulator dexterous workspace the existence of solutions is guaranteed.

The methodologies available in order to compute the results are the ones in closed form and the numerical ones.
In the first case it is possible to use the geometrical approach which as the name suggests, like in the direct kinematics, exploit the geometry of the manipulator, or it is possible to adopt the algebraic solution trough a transformation matrix that can be computed from the forward kinematic one.
Those closed form solutions usually are preferred if they can be found since they find the exact result and just generates in a systematic ways a reduced set of equations to be solved. Moreover the first one is a method that is easier to write but doesn't work with all of the configurations while the second one does even if it is usually harder to implement.

The numerical methods are instead an approach that through iteration find the best approximate values that solves the inverse kinematic problem and must be used for redundant manipulators. Despite they usually are slower approaches and the results do not say in which configuration we are they are easier to set up.

An example is the Newton's method.

2.3.1 Newton's method

In the uni-dimensional case of the Newton's method the aim is to find the root of a real function $f(x)$ and in order to achieve that few steps need to be iterated. The idea is to start with an initial guess x_0, then to approximate the function by its tangent line, and finally to compute the x-intercept of this tangent line.

To be more precise, by the assumption $f(x_i + \Delta x) \approx 0$,

$$x_{i+1} = x_i + \Delta x \tag{2.9}$$

$$\Delta x = -f(x_i)/f^{'}(x_i) \tag{2.10}$$

Through this technique after several iteration the algorithm converges in a quadratic time.

In the multidimensional case it is instead needed the computation of the Jacobian matrix. For what concern the field of mathematics and calculus this concept is related to vector function derivation and broadly states that it is formed by the first order partial derivatives considering $f(x) = (f_1(x), f_2(x), ..., f_n(x))^T$ and $x = (x_1, x_2, ..., x_n)^T$ since in this specific case we are considering the dimension of the variables equal to the function one obtaining

$$J = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{2.11}$$

and since $f(x_i + \Delta x) \approx f(x_i) + J(x_i)\Delta x_i$ the new delta will be

$$\Delta x = -J^{-1}(x_i)f(x_i) \tag{2.12}$$

Returning in the robotic field, in order to solve the inverse kinematics problem, it will be used a Jacobian calculated deriving the equation 2.1 getting its algebraic version

$$x_e^d = k(q) = k(q_i) + J_i(q_i)(q - q_i) \tag{2.13}$$

$$q_{i+1} = q_i - +J_i^{-1}(q_i)(x_e^d - q_i) \tag{2.14}$$

if the dimension of the operational space is the same as the dimension of the joint space and calling k the vector of the end-effector Cartesian coordinate concatenated to its orientation.

In case of redundancy the inverse of the Jacobian is not available so it should be used its pseudo-inverse defined by

$$J^+ = (J^T J)^{-1} J^T \tag{2.15}$$

2.4 Dynamic equations of a manipulator

Until now have been shown some of the static equations of a robotic manipulator, if it is needed to take into account also velocities and forces a dynamic model have to be derived.

Its creation in fact plays an important role for simulation of motion, analysis of manipulator structures, and design of control algorithms. Moreover the computation of the forces and torques required for the execution of typical motions provides useful information for designing joints, transmissions and actuators.

2.4.1 Lagrange formulation

The method that is going to be used in order to formulate those equations of motion of a manipulator in the joint space is the Lagrangian one. Simpler and systematic.

The Lagrangian formulation provides a description of the relationship between the joint actuator torques and the motion of the structure independently of the reference coordinate frame.

Considering a manipulator with n joint and so n variables that describe them(q_i) the Lagrangian of the mechanical system can be defined as a function of the generalized coordinates:

$$L = T - U \tag{2.16}$$

where T denote the total kinetic energy of the system and U the potential one. Moreover if ϵ_i is the generalized forces associated with the i-th joint then

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \epsilon_i \tag{2.17}$$

where $\dot{q}_i$ is the derivative of q_i.

The contributions to the generalized forces are given by the non-conservative forces like, the joint actuator torques, the joint friction torques, as well as the joint torques induced by end-effector forces at the contact with the environment.

2.4.2 Computation of the kinetic energy

In order to compute the Lagrangian formulation lets start with the kinetic component. This is given by the sum of the contributions relative to the motion of each link and the contributions relative to the motion of each joint actuator.

$$T = \sum_{i=1}^{n}(T_{l_i} + T_{m_i}) \tag{2.18}$$

where T_{l_i} is the kinetic energy of Link i and T_{m_i} is the kinetic energy of the motor actuating Joint i i.

The first component of the sum is given by

$$T_{l_i} = \frac{1}{2}\int_{V_{l_i}} \dot{p}_i^{*T} p_i^{*} \rho dV \tag{2.19}$$

where p_i^* denotes the linear velocity vector, ρ is the density of the elementary particle of volume dV and v_i is the volume of Link i.

Consider the position vector p_i^* of the elementary particle and the position vector p_{C_i} of the link centre of mass, both expressed in the base frame. One has

$$r_i = [r_{ix}, r_{iy}, r_{iz}]^T = p_i^* - p_{l_i} \tag{2.20}$$

with

$$p_{l_i} = \frac{1}{m_{l_i}}\int_{V_{l_i}} p_i^* \rho dV \tag{2.21}$$

where m_i is the link mass.

As a consequence, with the skew symmetric matricial property the link point velocity can be expressed as

$$\dot{p}_i^* = \dot{p}_{l_i} + w_i \times r_i = \dot{p}_{l_i} + S(w_i)r_i \tag{2.22}$$

where $\dot{p}_{l_i}$ is the linear velocity of the centre of mass and w_i is the angular velocity of the link.

Now if we substitute the last equation with the 2.18 it can be recognized that the kinetic energy of each link is composed by the sum of the following contributions:

- Translational as
$$\frac{1}{2}\int_{V_{l_i}} \dot{p}_{l_i}^T \dot{p}_{l_i} \rho dV = \frac{1}{2} m_{l_i} \dot{p}_{l_i}^T \dot{p}_{l_i} \tag{2.23}$$
- Mutual as
$$2(\frac{1}{2}\int_{V_{l_i}} \dot{p}_{l_i}^T S(w_i) r_i \rho dV) = 2(\frac{1}{2}\dot{p}_{l_i}^T S(w_i) \int_{V_{l_i}} (p_i^* - p_{l_i})\rho dV) = 0 \tag{2.24}$$
- Rotational as
$$\frac{1}{2}\int_{V_{l_i}} r_i^T S^T(w_i) S(w_i) r_i \rho dV = \frac{1}{2} w_i^T (\int_{V_{l_i}} S^T(r_i) S(r_i) \rho dV) w_i \tag{2.25}$$

  where exploiting the properties of the skew symmetric matrices we can write
$$\frac{1}{2}\int_{V_{l_i}} r_i^T S^T(w_i) S(w_i) r_i \rho dV = \frac{1}{2} w_i^T I_{l_i} w_i \tag{2.26}$$

The matrix I_{l_i} defined in 2.25 is called Inertia tensor and is relative to the centre of mass of Link i when expressed in the base frame

$$I_{l_i} = \begin{bmatrix} \int_{V_{l_i}} (r_{iy}^2 + r_{iz}^2)\rho dV & -\int_{V_{l_i}} r_{ix} r_{iy} \rho dV & -\int_{V_{l_i}} r_{ix} r_{iz} \rho dV \\ * & \int_{V_{l_i}} (r_{ix}^2 + r_{iz}^2)\rho dV & -\int_{V_{l_i}} r_{iy} r_{iz} \rho dV \\ * & * & \int_{V_{l_i}} (r_{iy}^2 + r_{ix}^2)\rho dV \end{bmatrix} = \begin{bmatrix} I_{l_i xx} & -I_{l_i xy} & -I_{l_i xx} \\ * & I_{l_i yy} & -I_{l_i yz} \\ * & * & I_{l_i zz} \end{bmatrix} \tag{2.27}$$

Moreover if it is wanted to express it with respect to the link i frame, considering R_i the transformation matrix from to the link i frame to the base one it will be achieved

$$I_{l_i} = R_i I_{l_i}^i R_i^T \tag{2.28}$$

To sum up with the found formulas and new definitions the result is

$$T_{l_i} = \frac{1}{2} m_{l_i} \dot{p}_{l_i}^T \dot{p}_{l_i} + \frac{1}{2} w_i^T R_i I_{l_i}^i R_i^T w_i \tag{2.29}$$

but this equation is not in function of the joints variables. In order to update the equation is possible to refer to the properties of the geometric Jacobian applied to the intermediate link other than the end-effector, yielding

$$\dot{p}_{l_i} = J_{P1}^{(l_i)} \dot{q}_1 + ... + J_{Pi}^{(l_i)} \dot{q}_i = J_P^{(l_i)} \dot{q} \tag{2.30}$$

$$w_i = J_{O1}^{(l_i)} \dot{q}_1 + ... + J_{Oi}^{(l_i)} \dot{q}_i = J_O^{(l_i)} \dot{q} \tag{2.31}$$

where

$$J_P^{l_i} = [J_{P1}^{l_i}, ..., J_{P_i}^{l_i}, 0, ..., 0] \tag{2.32}$$

$$J_O^{l_i} = [J_{O1}^{l_i}, ..., J_{O_i}^{l_i}, 0, ..., 0] \tag{2.33}$$

with columns computed as

$$J_{P_j}^{(l_i)} = \begin{cases} z_{j-1} for\, a\, prismatic\, joint \\ z_{j-1} \times (p_{l_i} - p_{j-1}) for\, a\, revolute\, joint \end{cases} \tag{2.34}$$

$$J_{O_j}^{(l_i)} = \begin{cases} 0 for\, a\, prismatic\, joint \\ z_{j-1} for\, a\, revolute\, joint \end{cases} \tag{2.35}$$

considering p_{j1} as the position vector of the origin of Frame $j1$ and z_{j1} as the unit vector of axis z of Frame $j-1$.

It follows that the kinetic energy of Link i can be written as

$$T_{l_i} = \frac{1}{2} m_{l_i} \dot{q}^T J_P^{(l_i)T} J_P^{(l_i)} \dot{q} + \frac{1}{2} \dot{q}^T J_P^{(l_i)T} R_i I_{l_i}^i R_i^T J_P^{(l_i)} \dot{q} \tag{2.36}$$

If now it is assumed that the motor that makes the link i move is located on the link $i-1$ and if it is assumed that no induced motion occurs so that the motion of Joint i does not actuate the motion of other joints then

$$T_{m_i} = \frac{1}{2} m_{m_i} \dot{p}_{m_i}^T \dot{p}_{m_i} \frac{1}{2} w_{m_i}^T I_{m_i}^i w_{m_i} \tag{2.37}$$

where m_{m_i} is the mass of the rotor,$\dot{p}^{m_i}$ denotes the linear velocity of the centre of mass of the rotor, $I_{m_i}^i$ is the inertia tensor of the rotor relative to its centre of mass, and w_{m_i} denotes the angular velocity of the rotor. Consider now θ_{mi} to denote the angular position of the rotor and k_{ri} the gear reduction ratio. On the assumption of a rigid transmission, one has

$$k_{ri} \dot{q}_i = \dot{\theta}_{m_i} \tag{2.38}$$

From this, exploiting the angular composition rule, it is possible to write

$$w_{m_i} = w_{i1} + k_{ri} \dot{q}_i z_{m_i} \tag{2.39}$$

where w_{i1} is the angular velocity of Link $i1$ on which the motor is located, and z_{mi} denotes the unit vector along the rotor axis.

As before in order to express the rotor kinetic energy as a function of the joint variables it is possible to express the linear velocity of the rotor centre of mass

$$\dot{p}_{m_i} = J_P^{m_i} \dot{q} \tag{2.40}$$

and the angular velocity as

$$w_{m_i} = J_O^{m_i} \dot{q} \tag{2.41}$$

It comes that the Jacobians are

$$J_P^{m_i} = [J_{P1}^{(m_i)}, ..., J_{P_{i-1}}^{(m_i)}, 0, ..., 0] \tag{2.42}$$

$$J_O^{m_i} = [J_{O1}^{(m_i)}, ..., J_{O_i}^{(m_i)}, 0, ..., 0] \tag{2.43}$$

where the columns are

$$J_{P_j}^{(m_i)} = \begin{cases} z_{j-1} for\, a\, prismatic\, joint \\ z_{j-1} \times (p_{m_i} - p_{j-1}) for\, a\, revolute\, joint \end{cases} \tag{2.44}$$

$$J_{O_j}^{(m_i)} = \begin{cases} J_{O_j}^{l_i} for j = 1, .., i-1 \\ k_{r_i} z_{m_i} for j = 1 \end{cases} \tag{2.45}$$

After those calculations is possible to write the kinetic of the rotor i as

$$T_{m_i} = \frac{1}{2} m_{m_i} \dot{q}^T J_P^{m_i T} J_P^{m_i} \dot{q} + \frac{1}{2} \dot{q}^T J_O^{m_i T} R_{m_i} I_{m_i}^{m_i} R_{m_i}^T J_O^{m_i} \dot{q} \tag{2.46}$$

Finally, by summing the various contributions relative to the single links and single rotors, the total kinetic energy of the manipulator with actuators is given by the quadratic form

$$T = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij}(q) \dot{q}_i \dot{q}_j = \frac{1}{2} \dot{q}^T B(q) \dot{q} \tag{2.47}$$

where $B(q)$ is called Inertia matrix and is defined as

$$B(q) = \sum_{i=1}^{n} (m_{l_i} \dot{q}^T J_P^{(l_i)T} J_P^{(l_i)} \dot{q} + \dot{q}^T J_P^{(l_i)T} R_i I_{l_i}^i R_i^T J_P^{(l_i)} \dot{q} + m_{m_i} \dot{q}^T J_P^{m_i T} J_P^{m_i} \dot{q} + \dot{q}^T J_O^{m_i T} R_{m_i} I_{m_i}^{m_i} R_{m_i}^T J_O^{m_i} \dot{q}) \tag{2.48}$$

2.4.3 Computation of the potential energy

Lets now proceed to the computation of the potential energy of the manipulator. Similarly to the kinetic case it is possible to express the potential energy as a sum of the contributions relative to each link as well as to each rotor

$$U = \sum_{i=1}^{n} (U_{l_i} + U_{m_i}) \tag{2.49}$$

Exploiting the assumption of having rigid links the contribution due to the gravitational force is given by

$$U_{l_i} = -\int_{V_{l_i}} g_0^T P_i^* \rho dV = -m_{l_i} g_0^T p_{l_i} \tag{2.50}$$

naming $g_0 = [0, 0, -g]^T$ the gravitational acceleration vector.

As for the rotor is similarly obtained

$$U_{l_i} = -m_{m_i} g_0^T p_{m_i} \tag{2.51}$$

gaining as potential energy

$$U = -\sum_{i=1}^{n} (m_{l_i} g_0^T p_{l_i} + m_{m_i} g_0^T p_{m_i}) \tag{2.52}$$

It is also possible to note that the final equation is a function of variables q and not of their velocities $\dot{q}$.

2.4.4 Equation of motion

Now that both of the components have been computed it is possible to write the Lagrangian as

$$L(q, \dot{q}) = T(q, \dot{q}) - U(q) \tag{2.53}$$

and taking the derivatives required by Lagrangian equation yields

$$B(q)\ddot{q} + n(q, \dot{q}) = \epsilon \tag{2.54}$$

where

$$n(q, \dot{q}) = \dot{B}(q)\dot{q} - \frac{1}{2}(\frac{\partial}{\partial q}(^T B(q)))^T + (\partial U(q) \partial q)^T \tag{2.55}$$

then

$$ddt(\frac{\partial T}{\partial q_i}) = \sum_{j=1}^{n} b_{ij}(q)\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} \partial b_{ij}(q)\partial q_k \dot{q}_k \dot{q}_j \tag{2.56}$$

$$\frac{\partial U}{\partial q_i} = -\sum_{j=1}^{n}((m_{l_j} g_0^T J_{P_i}^{l_j}(q) + m_{m_j} g_0^T J_{P_i}^{m_j}(q))) = g_i(q) \tag{2.57}$$

Finally with all the needed formulas the equation of motion becomes

$$\sum_{j=1}^{n} b_{ij}(q)\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}(q)\dot{q}_k\dot{q}_j + g_i(q) = \epsilon_i \, with \, i = 1, ..., n \tag{2.58}$$

having

$$h_{ijk} = \partial b_{ij} \partial q_k - 12 \partial b_{jk} \partial q_i \tag{2.59}$$

From this final equation of motion it is possible to note that the first term is strictly related to the Inertia of the manipulator, the second to the Coriolis effect induced on joint i by joints j and k and that the last one is a configuration dependent factor.

To summarize it in a more compact way and keeping into account also non conservative factors it is possible to write

$$B(q)\ddot{q} + C(q, \dot{q})\dot{q} + F_v\dot{q} + F_s sgn(\dot{q}) + g(q) = \tau - J^T(q) f_e \tag{2.60}$$

where F_v represent the viscous friction, F_s the static one, τ the actuation torques, f_e the vector of force and moment exerted by the end-effector on the environment and C is the Coriolis component such that

$$\sum_{j=1}^{n} c_{ij}\dot{q}_j = \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}(q)\dot{q}_k\dot{q}_j \tag{2.61}$$

Chapter 3

System environment

All of the implemented feature that are developed in a virtual environment so in this chapter all of the tools and supporting programs that have been used are going to be exposed.

3.1 Ros

The aim of the dissertation is to develop a generalized software able to work with any kind of manipulator not provided with any manual drive features.

This implies that the frameworks and underlying system must be totally independent of the internal processes employed by the various robotic arm types.
The best solution in order to achieve that is Ros [6], the robotic meta-operating system. An open-source framework that allows the development of robot systems and application through its vast set of software libraries. It also provides tools for obtaining, building, writing and running code across multiple computers and furthermore it provides services expected from an operating system including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes and package management.

With this environment the problem of generalizing the software is solved with its node-based structure. A representative image is shown in figure 3.2. Each node controls a different functions of the

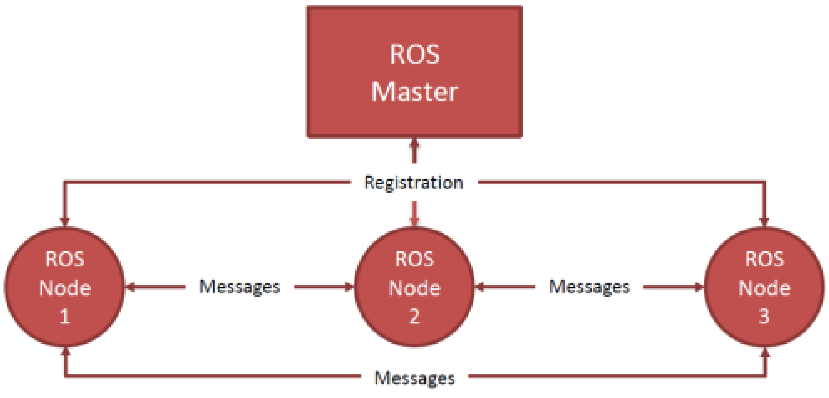


Figure 3.1: Ros node-based structure

system and can interact with the others thanks to the ROS network managed by the ROS Master. The ROS master stores information about the network because it sets up p2p connections between the nodes.

Those nodes communicate with the others through the message passing interface, a second important

feature of Ros that allows communication between multiple processes that may or may not live on the same computer.
A scheme of the process is reported in figure 3.2. In fact since the publish/subscribe system is

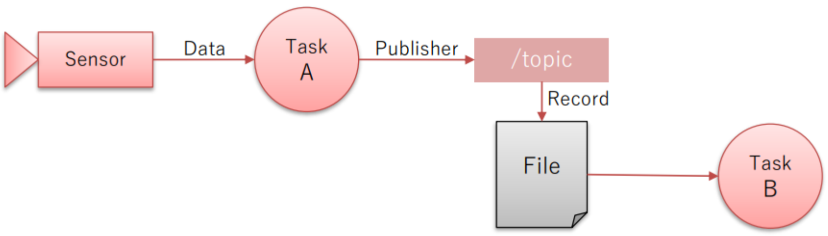


Figure 3.2: Ros messages interface

anonymous and asynchronous, the data can be easily captured and replayed without any changes to code. At the same time if a synchronous communication is desired the system's middleware offers also services that allows it.
Moreover, an another feature provided by the middleware, is the way for tasks to share configuration information through a global key-value store allowing to easily modifying task settings and to change the configuration of other tasks.

One last advantage of Ros is the high re-usability that its libraries offer. Its packages provide stable, debugged implementations of many robotics algorithms that since the message passing interface is becoming a standard for robot's software interoperability are often available.

3.1.1 Franka emika

Between those packages the one that has been used for the feature construction belong to a notorious company with a lot of experience and famous in the robotics field: Franka emika [7].
The Franka Control Interface (FCI),connected to Ros through franka_ros, allows a fast and direct low-level bidirectional connection to the Arm and Hand. It provides the current status of the robot and enables its direct control with an external workstation PC connected via Ethernet. By using libfranka, an open source C++ and Python interface, is possible to send real-time control values at 1 kHz like gravity and friction compensated joint level torque commands, joint position or velocity commands and Cartesian pose or velocity commands.
Moreover at the same time, is possible to get access the measurements at 1kHz of measured joint data, such as the position, velocity and link side torque sensor signals and various collision and contact information.
An another important feature is that is the the library provides informations concerning forward kinematics of all robot joints, the jacobian matrix of all robot joints and the dynamics components like the inertia matrix, Coriolis and centrifugal vector and gravity vector.

It has to be stated that for the purposes of this dissertation other libraries work as well but Franka emika have been chosen after a comparison with the other available ones(like the PandaArm python library) since its features resulted to be more complete and results to be preciser [9].

3.1.2 Rviz

For the visualization and the test planning part it is used a specific tool offered by Ros itself: Rviz [10]. Rviz (short for "ROS visualization") is a 3D visualization software tool for robots, sensors, and algorithms that enables to see the robot's perception of its world (real or simulated). Since it is already part of the adopted environment it also allow an easy communication between himself and the coding part without time consuming calibrations or specific support packages. Between the feature there is the one that permitting the use of markers and of visual interfaces to plan and set the goals of the manipulator in a really user friendly way simplifying the testing part and the simulations. An illustrative picture is shown in image 3.3

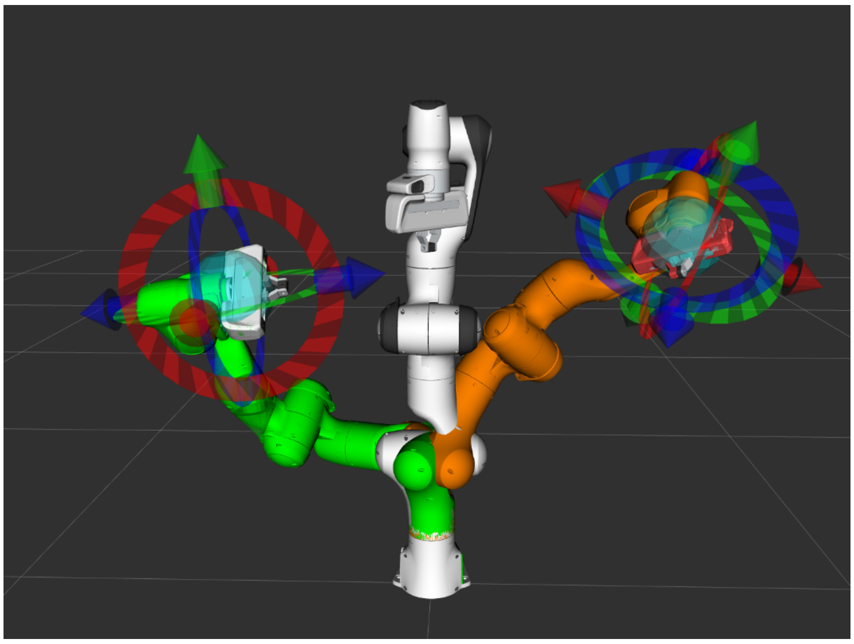


Figure 3.3: Example of the Rviz markers on a PandaArm

3.2 Gazebo

If rviz describe what the robot perceive and help in the planning the actual "real world" simulator is gazebo. This software allows a 3D simulation of the environment and of the robotic manipulator and is able to create a complete and realistic scenario for it having the possibility of adding obstacles and object. Furthermore it uses a physical engine for illumination, gravity and inertia that allows to execute simulations really close to the reality. Despite many program more user-friendly exists, like V-rep and Darts, gazebo is the most complete disposing of a plenty of features and settings to help the simulation and make its design more tecnical. For this reason is widely used across the robotics community resulting as a universal and relatable alternative.

3.3 Model of the robotic manipulator

Once it have been described all of the used simulation tools it is time to move on the actual manipulator. For the simulations between the several choices it have been decided to adopt the PandaArm manipulator from Franka emika. Its model is exposed in figure 3.4 One of the reason of its choice, apart being part of the standard collaborative manipulator for simulations, is its capability of detecting the torque acting on the joint that, as will be exposed in detail later, is one of the keys for a correct detection of the external forces. Despite it is already provided by algorithm for the human collaboration it has been used without those implementation resorting only to the torque sensors that it present. Here below in the figure 3.5 is reported the Denavit-Hartenberg table of its parameters

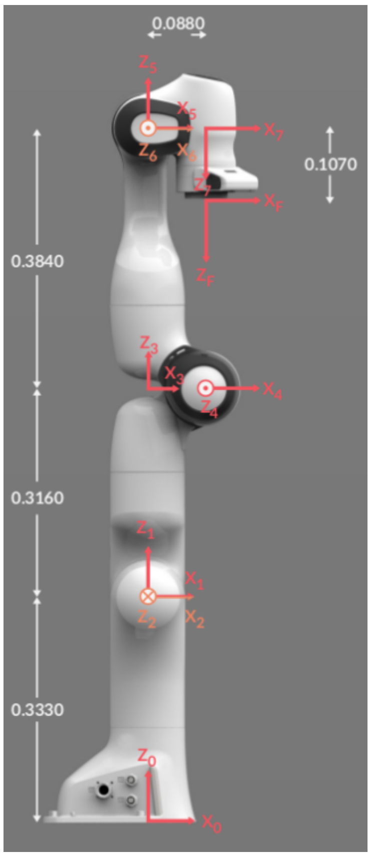
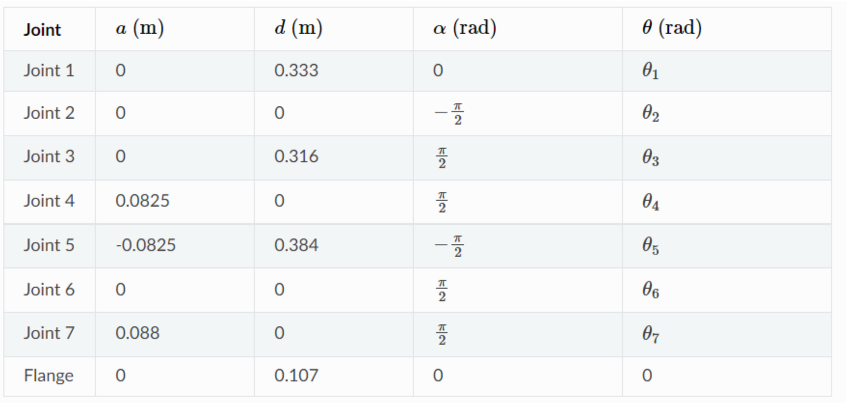


Figure 3.4: PandaArm model



Joint	a (m)	d (m)	α (rad)	θ (rad)
Joint 1	0	0.333	0	θ_1
Joint 2	0	0	$-\frac{\pi}{2}$	θ_2
Joint 3	0	0.316	$\frac{\pi}{2}$	θ_3
Joint 4	0.0825	0	$\frac{\pi}{2}$	θ_4
Joint 5	-0.0825	0.384	$-\frac{\pi}{2}$	θ_5
Joint 6	0	0	$\frac{\pi}{2}$	θ_6
Joint 7	0.088	0	$\frac{\pi}{2}$	θ_7
Flange	0	0.107	0	0

Figure 3.5: PandaArm DH table

Chapter 4

Features implementation

Having the information about the needed theoretical knowledge and the information about the used settings it is possible to move on the effective features.
Starting from the beginning, since the following developed algorithms are built from a base common script, it is going to be exposed now. This script aim to control the position of the manipulator's end-effector given the r-viz markers and achieve the result using a simple PD control.
Having this fundamental feature allows to move the manipulator according the desired goals with high precision, a standard feature for industrial robotics, and is obtained finding the needed torque values in a way similar to the inverse kinematics.
Its code can be entirely found in the appendix A.1 and, given a rate of work, it assume as goal position and orientation the one received by the marker and then constantly iterate the control thread aiming to minimize the errors. In order to achieve that it exploit the Jacobian for relationship between force on the end-effector and torque on the joints and a controller with refined proportional and derivative gains.

$$\begin{cases} F = \begin{bmatrix} P_{Position}(position_{goal} - position_{current}) \\ P_{Orientation}(orientation_{goal} - orientation_{current}) \end{bmatrix} - \begin{bmatrix} D_{Position}(LinearVelocity_{current}) \\ D_{Orientation}(AngularVelocity_{current}) \end{bmatrix} \\ \tau = J^T F \end{cases} \tag{4.1}$$

4.1 Workspace limitation

In order to achieve a correct workspace limitation aiming to limit the end-effector space of movement there are two kind of available approaches.

The first one is to act on the position of the end-effector frequently checking its coordinates, comparing them with the desired boundaries and, if it is out of the desired limitation, force it to the closest available position. In that way the end-effector is constantly checked and controlled to the right position resulting sturdy even to external interactions. On the other hand this kind of approach imply a really high computational cost, since the system has to constantly check with an high frequency the parameters of the coordinates and force or not the manipulator, and an average precision since the end-effector is constantly trying to reach its goal moving towards it and this could lend to slight border crossing.

The second approach, that it the adopted one, focus instead on targeting directly the goal of the manipulator. In fact if it is out of the desired thresholds it force it on the closest point of the boundaries and then let the system reach it with the standard process. In this way the global computational cost become significantly lighter with a higher precision than the previous case since the manipulator won't need to cross any border.

4.1.1 Spherical coordinates

The basic assumption needed in order to have a useful workspace limitation is that the intersection between the manipulator reachable workspace and the desired boundaries of the forced section of space has to be not empty. In order to respect this condition the example adopted is the one of a sphere: a simple and convex area which center is an arbitrary point inside the manipulator reachable workspace.

The equations that describe it are

$$\begin{cases} \mathsf{x} = \mathsf{x}_0 + \rho\cos\varphi\sin\theta \\ y = y_0 + \rho\sin\varphi\sin\theta \\ z = z_0 + \rho\cos\theta \end{cases} \tag{4.2}$$

and an example is reported in figure 4.1

4.1.2 Implementation

Since the aim of the feature is to force the goal on the border of the sphere if it is outside the desired space, due the geometrical circumstances of the example, is sufficient to only adapt the ρ to the desired one. In that way is possible to avoid the use of the inverse equations and pass from Cartesian coordinates to the spherical ones just for adjusting the radius, letting in that way φ and θ untouched. One method available for this radius adjustment is to first shift the coordinate of the goal in order to have the center of the sphere(O') as reference origin, get its normalized vector(v), multiply it by the ρ of the desired sphere(getting O'B) and then shift back again the result in the original reference system.
An explicative scheme is reported in figure 4.2.

In the Appendix A.2 is shown the code of this feature.

4.1.3 Drawbacks

At the same time from this strategy disadvantages we can find the fact that it is not robust against external forces since it only act on the end-effector's target, negligible aspect if we are considering a safe or closed environment or if the robots is driven with compliance by a human operator and is exploiting the workspace limitation in order to keep the arm on a specific surface in order to force by himself the end-effector tools for the given task. An another disadvantage arise in the cases of

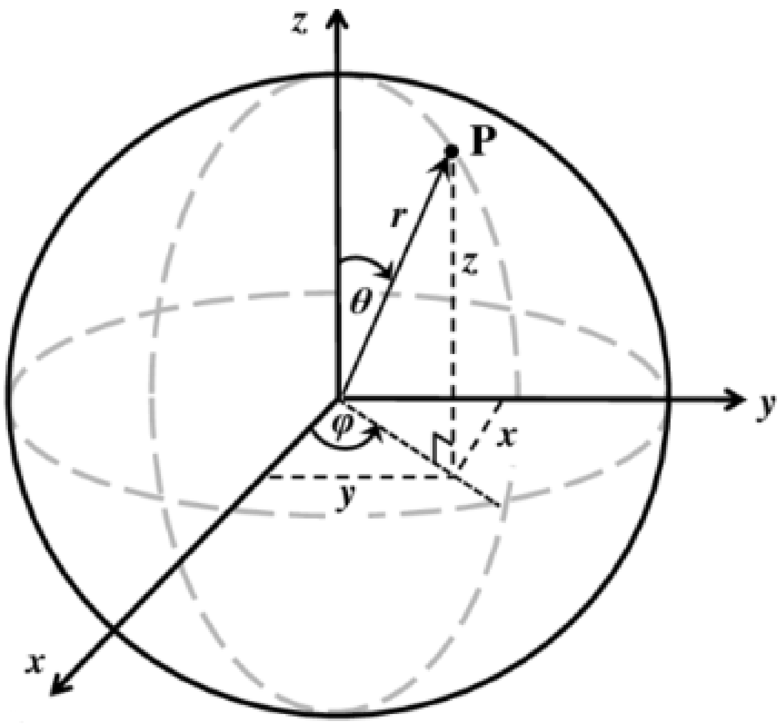


Figure 4.1: image of a sphere with its spherical coordinates

really complex workspace limitations. To be more specific in concave spaces, since the manipulator is programmed for taking the shorter path for the goal it could happen the eventuality of a collision for consecutive movements on the border(see figure 4.3). Eventuality not considered in this work but easily avoidable with a prior proper check on the points in between the end effector and its goal and for example subdividing the path in several smaller translation towards to points on the border.

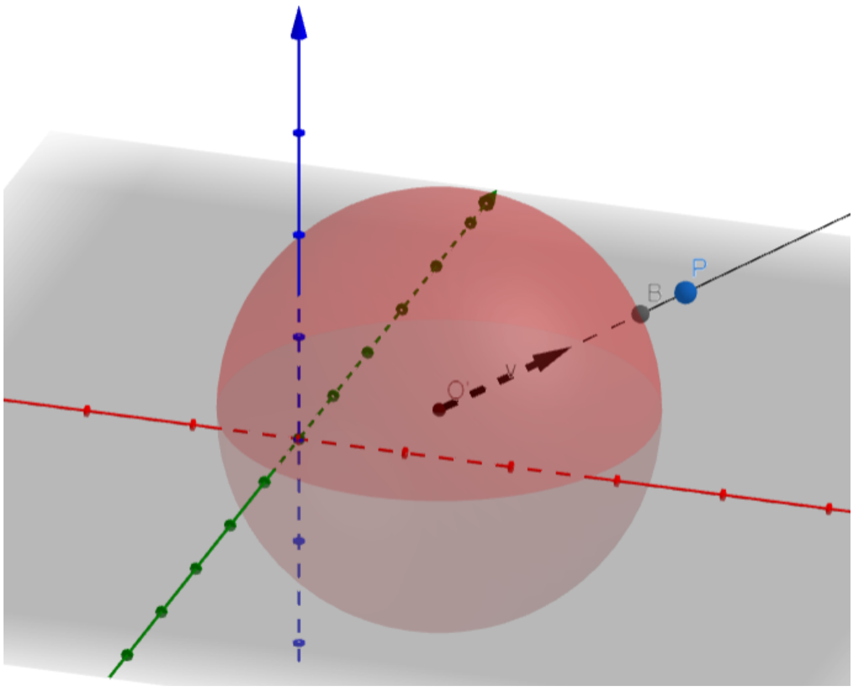


Figure 4.2: image of the downscaling of the ρ

4.2 Sinusoidal path planning

For the planning of a sinusoidal path instead of using just a goal position or segmenting the trajectory with too many predefined points the polynomial method have been used. Thanks to that it have been possible to find the coordinate of the necessary points for the following of a sinusoidal path given the time and according to the rate of the system. The obtained path resulted way smoother than the former approach and take into account also acceleration and velocities aiming to optimize them.

4.2.1 Polynomial equations

As said earlier the current feature exploits the polynomial equations and, if the desired trajectory is not just linear, for every dimension at least two equations needs to be computed.
For explanatory purposes now consider the fundamental case: an iteration across a segment executing a round trip between its extremes. In order to realize it a third grade polynomial law is sufficient and state that the position is

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{4.3}$$

where $x(t)$ is the spatial coordinate given the time and a_0, a_1, a_2, a_3 are constants belonging to R.
Stated the starting equation, for an actually adoptable law, the constant values need to be found. This is easily achievable considering the initial and final conditions desired for the path: If in fact the initial velocity(v_0) and the final one(v_f) are imposed as null and considering the starting coordinate(x_0) and the ending one(x_f) it is possible to proceed toward the actual computations. Deriving the equations 4.3 and imposing those condition is possible to compute

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ v_0 \\ x_f \\ v_f \end{bmatrix} \tag{4.4}$$

where t_0 and t_f are the desired beginning time and ending time, fundamental parameters since the equation doesn't take into account the manipulator's capabilities and those two values should be

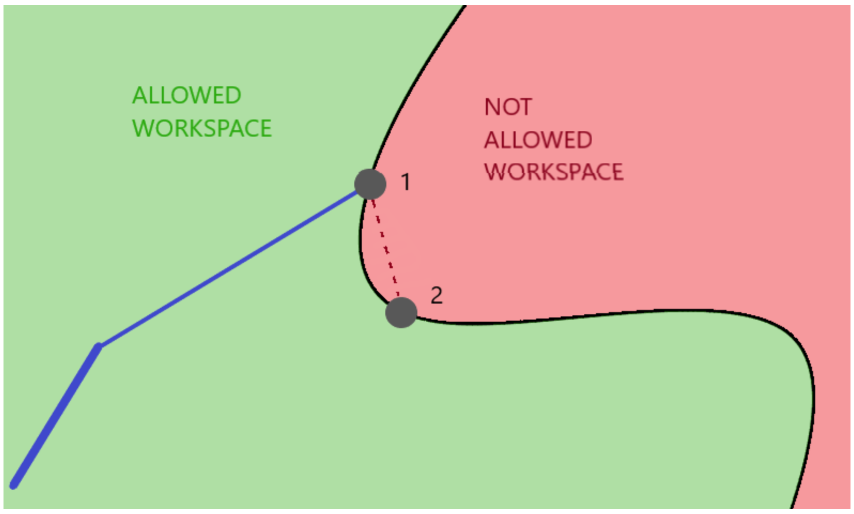


Figure 4.3: example of a concave workspace

aware of the feasible performances of the joint's motors.
Thanks to this equation is possible to find the desired constants and so have a full law available for the system to follow based on the time that from a point start accelerating towards the goal and in the middle point begin to decrease the acceleration until it stops in the target position.
It have to be noted that, in order to have a round trip, a second polynomial equation is needed for the way back imposing new initial and final times and switching beginning and ending coordinates.

4.2.2 Circular example

Passing to the case of an increase of the dimensions in the desired path lets for instance discuss the case of a circular trajectory. In order to realize it 5 polynomial equation are needed: 2 for both dimension plus one for the beginning of the path actuation as shown in the image 4.4 considering a clockwise spinning.

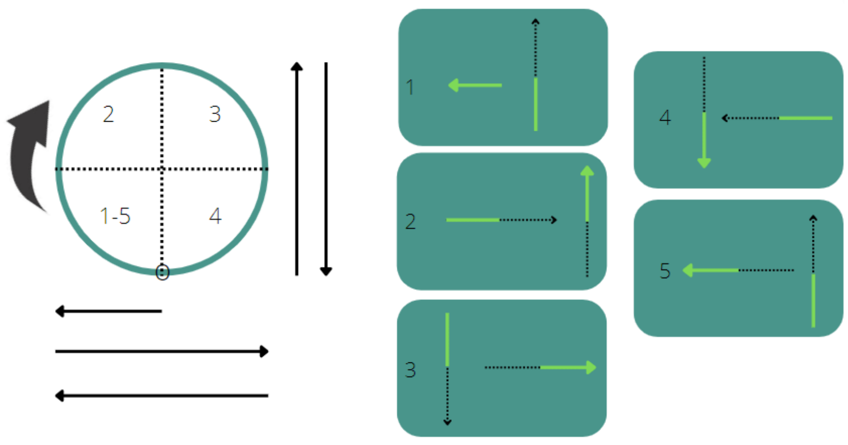


Figure 4.4: Polynomial equations for a circular path planning

As the image explain in order to achieve the desired result is sufficient to assign the right polynomial law to the correspondent coordinates according to the current time. To be more specific if we assume a travel time of 10 seconds spit equally in the circle quarters:

- for the beginning section 1, the one that after the first execution will be permanently substitute by the number 5, is used the short polynomial law and so developed to go from the center towards the left extreme for the horizontal component while for the vertical one will be adopted and the equation that lead the manipulator from the bottom point toward the upper one;

- for the section 2 the vertical component follows the previous vertical law while the horizontal one this time follows an polynomial equation that aims to start from the leftmost point and go towards the rightmost one;
- for the section 3 the horizontal component have to follow the horizontal equation of the section 2 while the vertical one this time need to go downwards and so to adopt the vertical law that this time starts from the upper point and move towards the bottom one
- for the section 4 the vertical component continues to follow the equation used by the vertical coordinate in section 3 while the horizontal one starts to move to the leftmost point with the polynomial equation left to use
- finally the section 5 is the same as the section 1 exception made for the horizontal component that this time have to follow the horizontal law of the section 4;

In the Appendix A.3 is possible to see both of the implementation under "sinusoidal control thread" and "circular control thread".

4.2.3 Possible implementations

The previous examples were extremely basic and represent a fundamental path for the development of more complex trajectories.
Despite that even with the simple use of the linear and circular case, with the implementation of the force compliance that will be shown later, some interesting results could be achieved.
For instance in the case of a long surface to glue or pain, the manipulator could follow a linear and sinusoidal path letting a human operator to push the end-effector up and down or to press it on the surface as needed by the task.
Same case for a compact surface which could present various irregularities that could be managed by a simple circular path from the manipulator with the consequent eventual forcing by the human operator.
Aiming instead to a full automated circuit working for regular shaped surfaces some interesting implementations could be used.
One example could be the use of the linear case but every time adding to the ending horizontal coordinate a fixed value in order to have an horizontal translation(see figure 4.5). Result that could as well be obtained directly implementing an horizontal polynomial law towards left or right. An

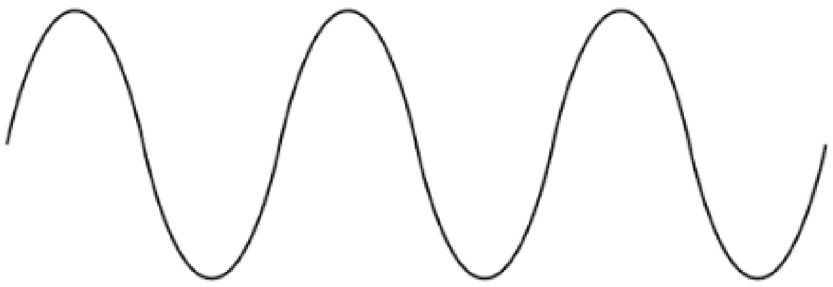


Figure 4.5: example of a sinusoidal path obtained by the linear one

another example, if the task require a wider area to cover, could be the modification of the circular path setting in the last quarter to execute a slightly shifted coordinate toward the desired direction obtaining a sort of projection of a helicoidal trajectory(see figure 4.6).
Moreover the possible applications could also be implemented in the three dimension by the addition of further polynomial laws suited for the desired cases. For instance the implementation of a shifted plane or the one of a sphere section or even more complicated shapes with the right mathematical implementations.

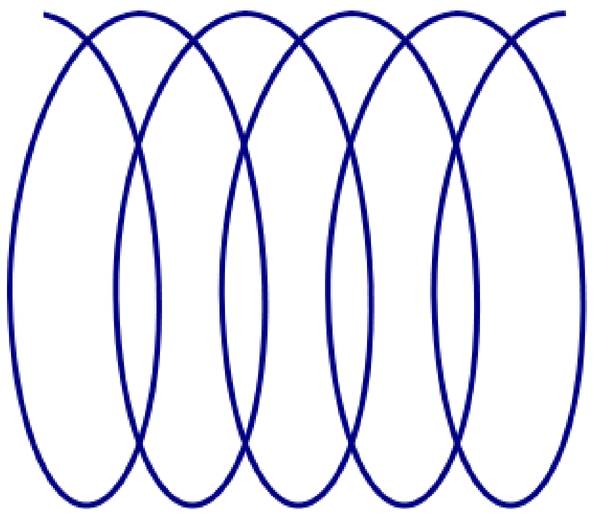


Figure 4.6: example of a helicoidal 2D path obtained by the circular one

4.3 Force compliance control

The compliance control aims to allow a human operator to freely move a manipulator according to his movements in order to perform tasks. For this reason a properly developed system for allowing that should continuously taking care of external stimulus and consequently actuate the proper actions.

Keeping this in mind the proposed solution that will be exposed suggests an algorithm that, given a reference pose that the manipulator pursue by default, will indulge on the detected external forces and counteract to the undesired ones. The distinction between desired and undesired forces is made on few conventions that depends on the task that the manipulator will achieve and are going to be exposed later.

As told earlier, in this feature the used input for achieving this cooperation is going to be the detected external force applied on the manipulator from the reference frame of the end-effector. In this way even if the human operator's inputs are applied on the others links of the manipulator the focus of the compliance will remain the end-effector itself since they will be finalized for moving the tool in charge of interacting with the environment in order to compute the desired task.
Furthermore it have arbitrarily have been decided to detect forces instead of velocities for a matter of precision and accuracy, since generally the robot's sensor are usually more sensitive to the forces because they are provided with the possibilities of recognizing their own joints exerted torques, and for a matter of practicality. For what concern the latter aspect it has to be considered that when they deals with manual works, due to their muscular composition and to their behaviour, humans tend to exert short and intense forces rather than constant and prolonged velocities obtained by the repetition of small instantaneous efforts.

For the implementation of this feature two big sub-goal have to be managed:

- the force detection, followed by a proper filtering of the inputs
- the managing between force control and stiffness control

Both steps that will require a conscious and aware calibration.

4.3.1 Force detection

The force detection part as the name suggests aims to correctly identify the external forces acting on the manipulator, to minimize the eventual errors and if needed to properly filter the receive signal.
In order to find the external forces acting on the end-effector the formula 2.60 comes to help.

In fact the equation lend to

$$\tau_{ext} = \tau_{total} - (B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s sgn(\dot{q}) + g(q) + \tau_{control}) \tag{4.5}$$

where the $\tau_{control}$ taken into account is the control vector obtained from the manipulator given a certain move command. So considering a system with null frictions or a system where the friction models were observed and described computationally and taking into account an end-effector Jacobian for the inverse mapping between Cartesian and joint space, it would be possible to have

$$\tau_{ext} = J(q)^T f_{ext} \tag{4.6}$$

with a 6x1 f_{ext} having its first three lines as components of the external forces acting on the end-effector.
It can be noted that a lot of prior information are needed for those computations but the modern software that interact with the manipulators easily allow to have access to them.

For the realization of this part in the beginning it was used the PandaArm python library since it lends all of the equation's components for then calculating the external forces but the obtained results were not as precise as wanted.
Probably since the components of the dynamic equations already took into account values and conventions different from the considered ones.
The graphs in figure 4.7 reports some results about that.

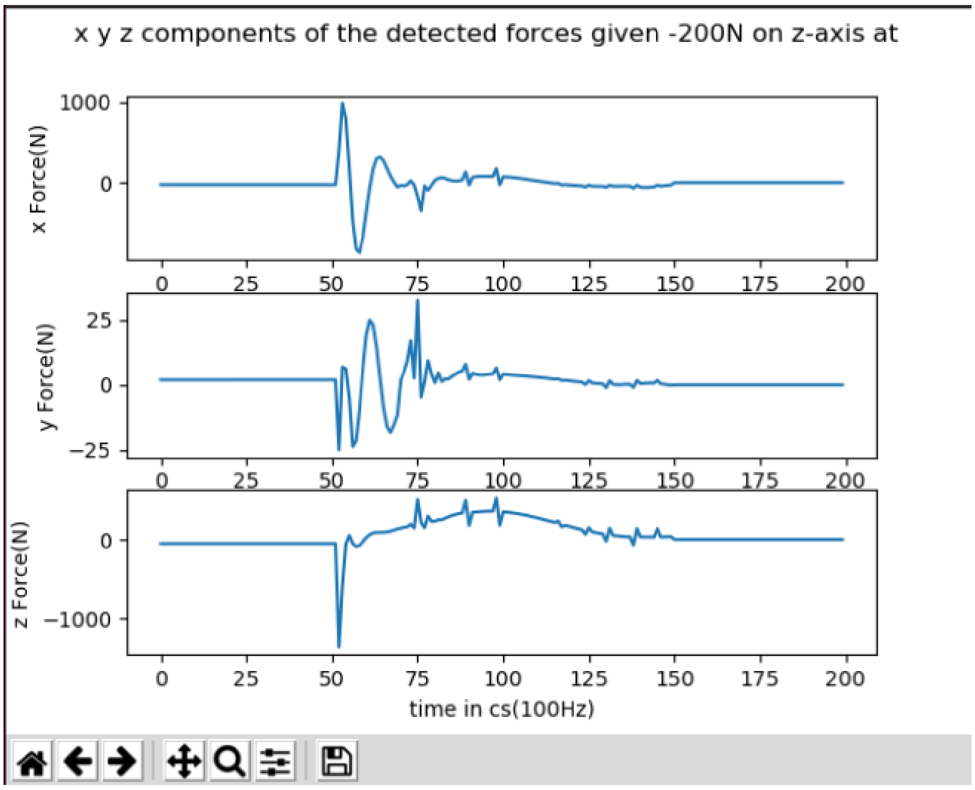


Figure 4.7: example of the results gotten with the PandaArm library

For this reason it have been substituted by the franka emika one that made all of this calculations following automatically the explained equation giving as result more clear and relatable data to work with.

Despite the accuracy of the obtained detected forces and even if the detection trials where made in a simulated environments applying instantaneous and constant forces to the end-effector, the obtained measures were of course not precise as the imposed one. Expected consequence of working with a good and realistic simulator. For this reason a lot of trials have been made in order to understand the behaviour of the system and the results shown a pretty clear pattern.

In a concrete scenario the expected results from a force measure would be in a form expressed by 4.8

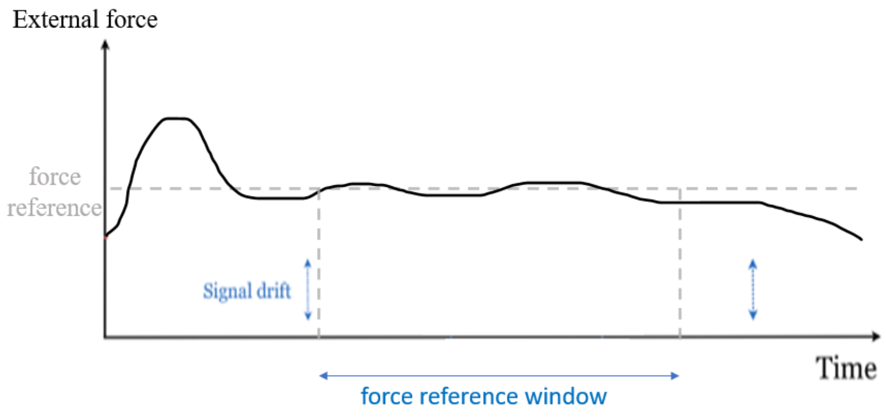


Figure 4.8: hypothetical example of an external force applied in a concrete scenario

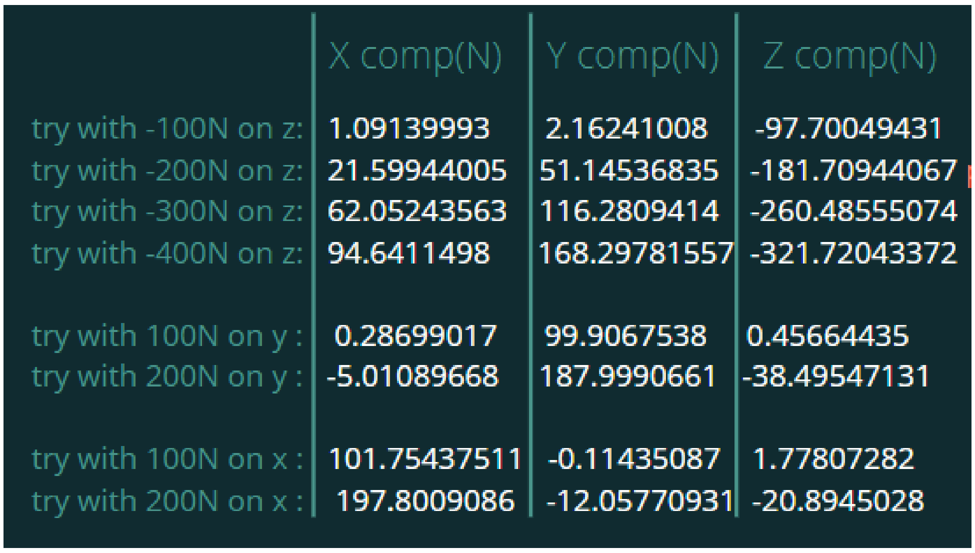
As it is possible to notice from the image 4.8 when the force is applied,it usually present an initial consistent overshoot and then it stabilize over the desired force reference but without totally converging.
For this reason the part of the signal taken into consideration doesn't start with the initial variation of force but after a certain delay.

A further observation that can be made consist in the presence of an always present and consistent factor that does not allow the signal to be null in the stationary case which generates a signal drift, this factor is the gravity itself.
In facts joints that are currently being compensated for gravity are typically holding the link's weight for a long time and, therefore, variables like temperature will make joint torques deviate from the computed theoretic values that don't account for this kind of variation. For this reason the expected external torques can't be exactly zero when the system is still.

In the simulation, considering that the simulated external force is a constant signal exerted in a specific and defined time window, the detected values report different behaviour.
In order to get relatable and valuable data, several measurements have been made exerting the desired forces on the end-effector. Considering a force applied on the system at the second 1(time delay) and for a duration of 1 second, 100 samples of the signal variation have been recorded and those trails have been repeated 5 times for each kind of force.
The chosen rate of measurement is sufficient to evaluate the signal performances.
Considering that without external forces applied the detected one(in Newton) are $\begin{bmatrix} 0.18 \\ 0.32 \\ -1.80 \end{bmatrix}$ with as highest parameter the one most affected by the gravity, here the mean of the obtained values in the different components is shown in figure 4.9 It has to be taken into account that averagely a human arm can exert a force between the 50 and the 150-200N

, the measures made with higher forces where performed in order to stress the behaviour of the system and for sake of completeness.

Moving onto the numerical data, it is possible to note that the detected forces present some inter-



	X comp(N)	Y comp(N)	Z comp(N)
try with -100N on z:	1.09139993	2.16241008	-97.70049431
try with -200N on z:	21.59944005	51.14536835	-181.70944067
try with -300N on z:	62.05243563	116.2809414	-260.48555074
try with -400N on z:	94.6411498	168.29781557	-321.72043372
try with 100N on y :	0.28699017	99.9067538	0.45664435
try with 200N on y :	-5.01089668	187.9990661	-38.49547131
try with 100N on x :	101.75437511	-0.11435087	1.77807282
try with 200N on x :	197.8009086	-12.05770931	-20.8945028

Figure 4.9: Measures of the detected external force simulated and applied on the end-effector

esting features different from the real case. To begin with it is immediate to see that the peak of the detected signals never exceed the actual one stopping itself earlier and without converging to the actual force. At the same time as the magnitude of the force increase the error with respect to the applied force grows.
In fact is possible to see that

- in the trial with -100N applied to the Z axis, the fully operational value of the detected force is about the 97% of the real one
- in the trial with -200N applied to the Z axis, the fully operational value of the detected force is about the 90% of the real one
- in the trial with -300N applied to the Z axis, the fully operational value of the detected force is about the 86% of the real one
- in the trial with -200N applied to the Z axis, the fully operational value of the detected force is about the 80% of the real one

An another fact is that while the module of the applied force increase, increase also the noise in the system "spreading" the detected force into the other two components, even if they should be null.
In fact if when in one of the three dimension a force of 100N is applied their value is at most about the 3% of the dominant factor while, rising the magnitude of the applied force, the error grows to the 27% in the case of the 200N and reach its 50% in the extreme case of the 400, situation that won't be taken into account in the next development.

So the resulting image that emerge from this kind of data should have a shape close to the one shown in figure 4.10

Due to the presence of those inaccuracies it have been decided to apply a filtering before taking into account strong forces for the sake of a cleaner control.
This filtering consider all of the previous observation and take care of them through a calibration based on the several numerical values seen.
Here below is shown the pseudo-code applied only in the case of a relevant forces

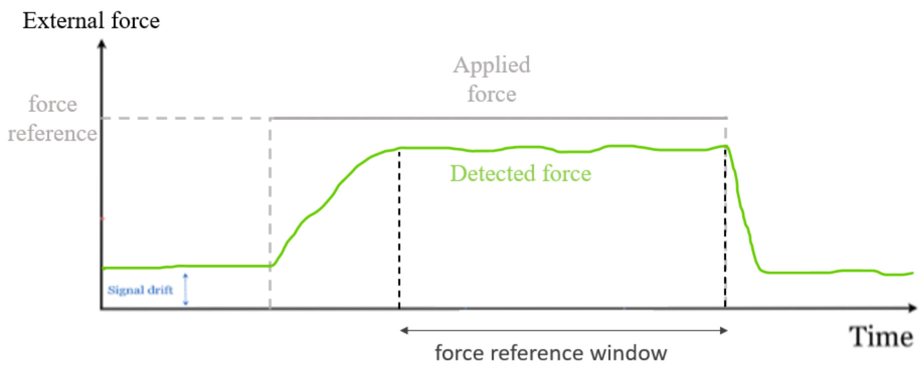


Figure 4.10: hypothetical example of an external force applied in a simulated scenario

Algorithm 1: Pseudo-code of the external force filtering

```
Data: detected_force.
Result: filtered detected force.
filtered_force=zeros(3,1);
receive detected_force;
if (|detected_force|>minimum_force) && (|detected_force|<security_threshold) then
    max = maximum component of the detected_force;
    insert max in filtered_force;
    if filtering_percentage|max|<detected_force[*] then
        insert detected_force[*] in filtered_force;
    end
    if filtering_percentage|max|<detected_force[**] then
        insert detected_force[**] in filtered_force;
    end
end
```

This pseudo-code is supposed to be fed with a detected external force that has been measured after a certain time delay in order to consider only the meaningful part of the signal, time of about 0,3 seconds. Then the absolute value of the detected signal is compared with a minimum force that have been arbitrarily set to 20N, imagining a task that doesn't require extreme delicacy, and at the same time is compared with a maximum force called "security_threshold". As the name suggests this constant was thought for safety purposes in order to avoid the following from the manipulator of inputs too "strong" to be safely handled by a human operator and have arbitrarily put to 200N. Value that as the former one should depend on the goal of the desired job goal and be custom designed.
Once this check is made, is first of all taken into account the maximum parameter of the detected force that promptly inserted into the output vector in the corresponding position and then a control is made on the other two components.
Those are compared with a percentage of the maximum one and if they are bigger it is assumed their authenticity allowing them to be inserted in the corresponding positions of the output array, otherwise they are treated as errors and ignored. The percentage taken into account basing on the trials and their available data is set to be the 20% since the cases of external forces bigger than 200N have been excluded.

With this implementation is possible to filter the detected forces and obtain a relatable value that considering the simulations will be slightly weaker than the effective one but that in a real scenario would be close to the real peak.
It have to be also stated that this approach limits the 360° range of the possible translation discretizing the number of the available direction forcing, in some sporadic cases, the subdivision of a motion in

more steps but do this for sake of a safe control and according to a prior awareness of the eventual human operator.
Moreover a matter that have been ignored due to its negligibility is the signal drift: a value extremely low for the measures taken into account but that depending on the concrete application could become more significant and so could probably need to be taken care of.

4.3.2 Control managing

With the control managing section the software, after having correctly received the detected external force from the previous part, aims to manage the behaviour of the manipulator between a control that go according to the exerted forces, that will be called force control, and a control that aims to contrast any external interaction, that will be called stiffness control.
As said earlier the whole program goal is to follow a predefined position by default and, in case of a significant human input, to work with the operator; for this reason the script have been developed such that in the eventuality of absent or (most likely) weak external forces it should by default contrast any interference controlling the end-effector position to the predefined one with the impedance control, but that with the arise of a significant interaction should put into effect the force control until the end of the collaboration.
In figure 4.11 is shown a diagram of the algorithm.

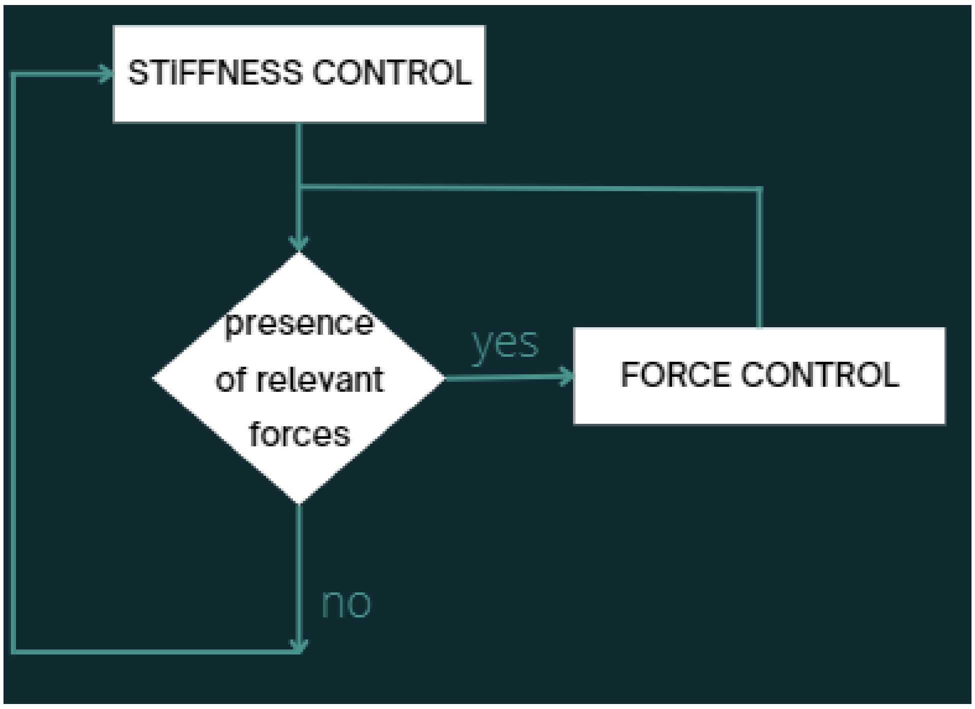


Figure 4.11: diagram of the program algorithm

Beginning with the default approach the stiffness control will be now analyzed.
Inspired to the base scripts exposed in the beginning of the chapter this kind of approach aims to track and follow the goal position but with an extra factor taken into account that is the detected external force.
Since the values that are considered in the impedance control belong to a window of really weak forces, between the 2N and the 20N, and since earlier it has been shown that with low magnitudes, more specifically below the 100N, the detection is really accurate, these values were not filtered with the former algorithm.
This decision has been made suited on the imagined task but it has to be taken into account that in other scenario a proper treatment of the received data could occur. An example for instance could be the case of a task in which the input required forces are extremely high like for heavy weight to

carry or a situation in which the precision of the end-effector position is crucial consequently having a more extended force window for the stiffness control.

The considered force window for the stiffness control starts from the 2N because since even in the simulation in always present the signal drift that in the real environment most likely will be enhanced, it has been decide to have the thread working until the goal position is reached with a certain precision and until the presence of forces at least stronger than the environmental noises disappear.

Said that the pseudo-code of the stiffness control is

Algorithm 2: Pseudo-code of the stiffness control

```
receive first detected_force;
while (position_error>0.5) && (|detected_force|>signal_drift) do
    if (|detected_force|>minimum_force) && (|detected_force|<security_threshold) then
        Force_control(rate);
    end
    receive position, orientation, and velocities of the end-effector;
    $F = \begin{bmatrix} P_{Position}(position_{goal} - position_{current}) \\ P_{Orientation}(orientation_{goal} - orientation_{current}) \end{bmatrix} - \begin{bmatrix} D_{Position}(LinearVelocity_{current}) \\ D_{Orientation}(AngularVelocity_{current}) \end{bmatrix}$
    F$_{tot} = F + k_{smoothen} * detected\_force$
    $\tau = J^T F_{tot}$;
    command($\tau$);
    calculate position_error;
    receive new detected_force;
    wait(rate);
end
```

where the algorithm operate according to the given desired rate basing on the possibilities of the manipulator.
Moreover it has been added a constant $k_{smoothen}$ that decrease the value of the detected external force in order to have a reaction from the manipulator less impulsive and more gentle.
For testing purposes this implementation of the system have been tested with external force higher than 5 Newtons and thanks to the correct use of the rate it was able to contrast the external interference suggesting a high reliability in the considered external force eventuality.

As for the force control the procedure where similar with slight differences. Going straight to the pseudo-code

Algorithm 3: Pseudo-code of the impedance control

```
while (|detected_force|>minimum_force) && (|detected_force|<security_threshold) do
    actual_force=filter(detected_force);
    $\tau = J^T(k_{force} * actual\_force)$;
    command($\tau$);
    receive new detected_force;
    wait(rate);
end
```

As anticipated earlier the script here filter the detected force and apply it to the manipulator directly since when the human operator is interacting with the machine there is no need to come back to the base position.
Moreover a new constant k_{force} is taken into account in order to strengthen the input force.
Its value is 1.2 and it is an empowering one firstly because of the difference in the detected signal with the applied one and so it aim to recover the true value and secondly because the thought

implementation for this software is a generic one and it has been decided to give more importance to the received input in order to make the resulting interaction less tiring on the human operator side. In fact thinking of a welding or painting task a more reactive manipulator could be felt as more helpful than a stiffer one while in the case of an end-effector endowed with pliers or syringe for biomedical purposes, given the delicacy and the precision required by the task, a constant factor included between one and zero and so able to smooth down the applied force could result more comfortable.

In the appendix A.4 is shown the whole code of the implementation.

Chapter 5

Results

In this chapter some numerical results and some considerations about the implemented features will be exposed evaluating the behaviour of the scripts.

5.1 Workspace limitation

For the workspace limitation feature it have been implemented a spherical threshold that has its center located on the end effector default position and that present a radius of 0.5, definitely less that the manipulator reach.
For showing purposes it have been placed the marker on the x axis and in order to have a goal position outside the desired threshold.

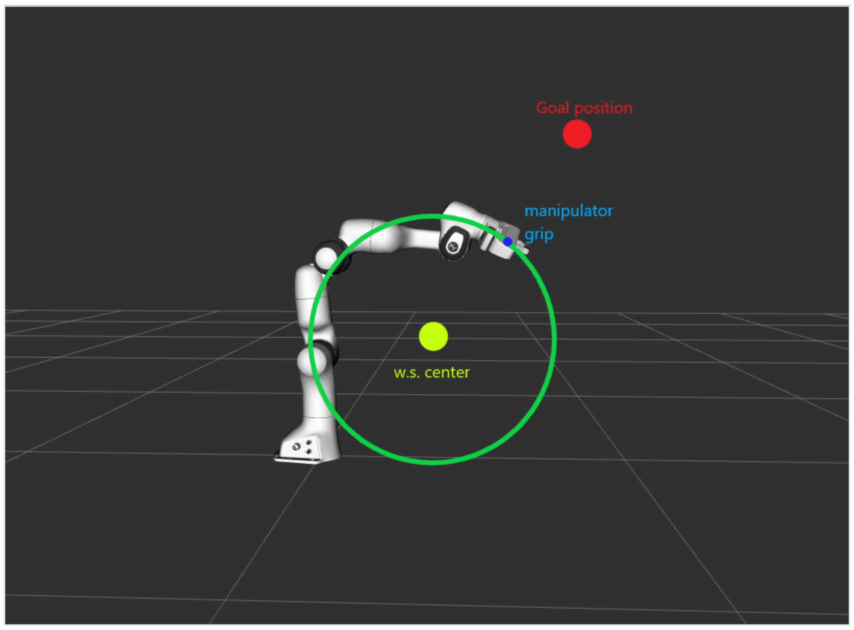


Figure 5.1: Manipulator that tries to reach a goal beyond its limited workspace

As it is possible to see from the image 5.1, the end-effector does not overcome the borders and stay under the spherical surface along the line between the goal position and the center of the spherical workspace.
In figure 5.2 a graph about the steady state error between the end effector position and the threshold witness the behaviour.

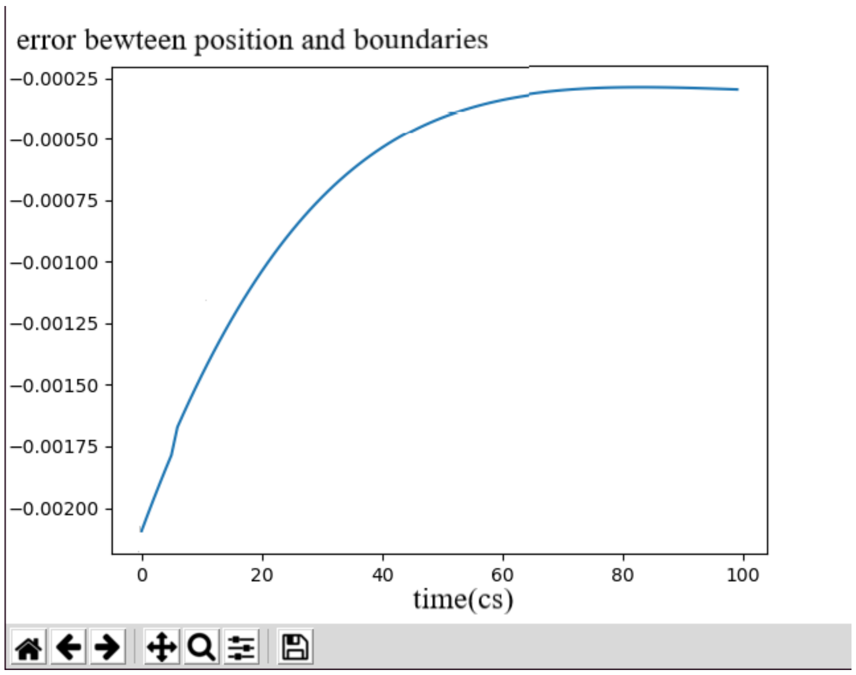


Figure 5.2: Mean error between the end effector position and the threshold

Showing the whole process of threshold reaching would require a huge plot making difficult to appreciate the details of the feature herself but like the graphic shows the end effector, after the initial approach phase where it move towards the goal, as told earlier place himself really close to the border staying below it.
The plotted signal is the mean of the error of the three components and present a negative sign since the manipulator is approaching the goal from below.

Looking at the image 5.1 is also possible to note a crucial aspect: if the end-effector respect the desired thresholds and limitation the rest of the manipulator often does not.
This is because the software implements a control suited only for the tool of the robots so that the other physical part are not considered.
In this dissertation the possible implementation are not treated but there are some possibilities that could be realized in future works. An immediate but not functional method if the aim of the task would become to avoid the interaction of the whole manipulator with the environment could be the restriction of the desired limited workspace considering the maximum extension of others components of the robot. Despite it should work it would extremely limits the end-effector workspace excluding poses that could be needed.
An another method that is widely used nowadays it the one of the collision boxes which assign to all of the mechanical parts of the manipulator a virtual box that will be taken into account during the trajectory planning avoiding intersection between them and the environment. A more sophisticated solution to reach the goal and that doesn't limits consistently the end-effector available positions.

5.2 Path planning

For what concern the path planning for showing purposes it has been implemented a simple circular path with a travel time of 10 seconds.

In the figure 5.3 some screenshots of the motion have been reported.

The only observation worth of notice concern the right choosing of the travel time. In this case it was in fact set in order to be slack and to permit a correct and clear visualization of the motion

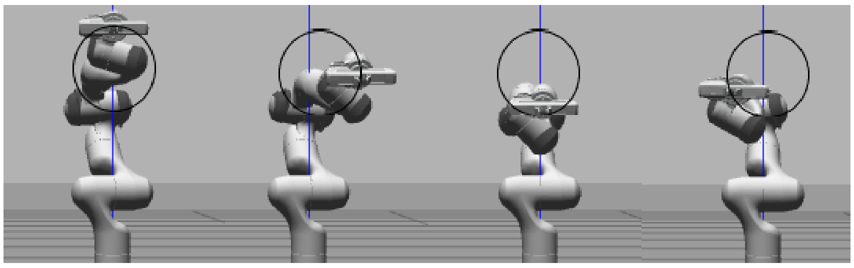


Figure 5.3: Screenshots of the manipulator following a circular trajectory

thought the simulation environment.
Otherwise in the eventuality of a wrong designed time parameter, for what has been observed, the manipulator tent to loose and not reach in time some goal positions during the process resulting in a rhomboidal shaped trajectory. A clearly undesired achievement that in the wrong concrete application(like in a welding task) could bring to important or dangerous consequences like the harming of human operators or the damaging of the materials in the surrounding.
Fortunately this feature has been designed for a human cooperation permitting an interaction when needed therefore implying controllable and manageable motions without reaching the maximum speed the manipulator can achieve(note that each joint can perform a maximum velocity between 150-180 degree per second [12]).

5.3 Compliance control

To show some results concerning the compliance control it has been made a simple test. In order to show the effectiveness of the stiffness control a vertical force of 5N has been applied downwards on the end-effector for 2 seconds considering it already in the default position. The results are reported in the figure 5.4. From the reported data is possible to note that the error is really low but even better

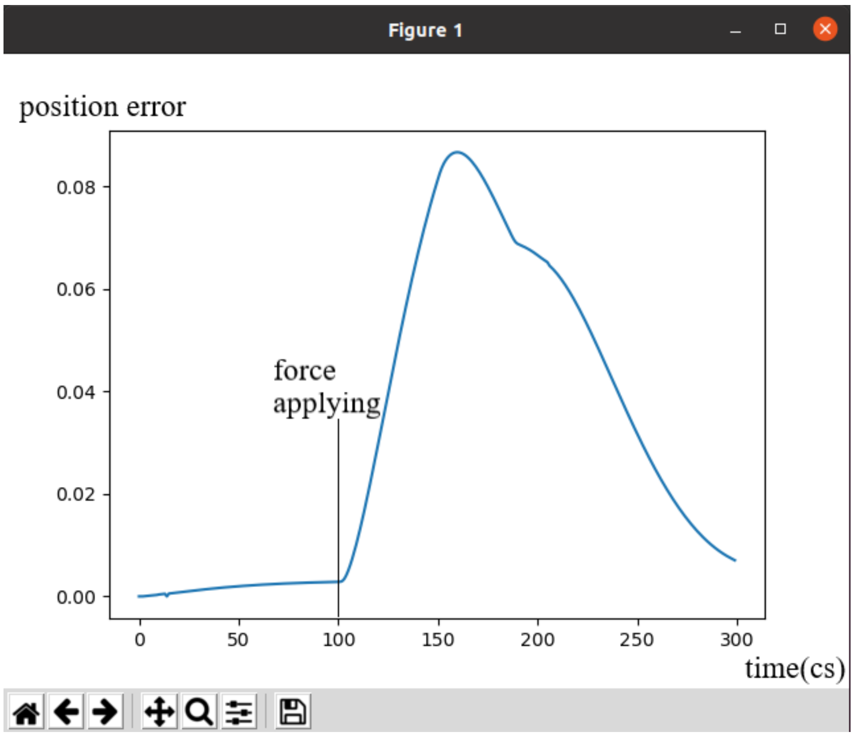


Figure 5.4: results of the stiffness control test

results could be achieved by more detailed calibration or by more sophisticated control techniques. An example could be a PID controller for estimation of the global force but also a global system with higher rate could result helpful for the sake of the goal.

As for the force control the experimental test to perform are a bit harder to expose.
Despite this, for showing purposes, a comparison between the perceived velocities of the end-effector with and without the feature is reported in the following.
In order to realize it a force of 25N have been applied on the end-effector along the z axis for 0.5 seconds and, with the constant variables exposed earlier, the resultant velocities have been plotted in the figure 5.5.

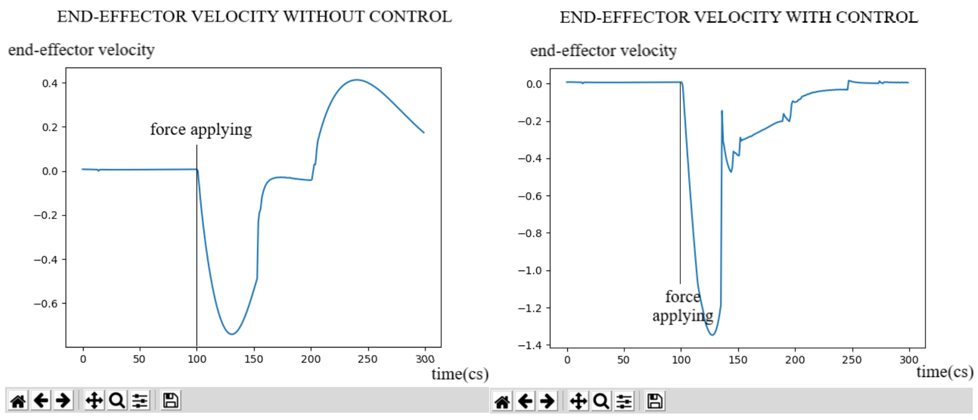


Figure 5.5: comparison between the end-effector velocities

From the image is possible to note that the algorithm specifically enhance only the relevant forces that it detects but due to the physical nature of the system, several kind of noises arise after the application.
Moreover it has also to be noticed that in the second plot no positive velocities, related to end-effector coming back to its default position, are shown. This is due to the fact that they are just postponed, and so not reported in the graph, since the system waits until external forces becomes negligible, keeping the end-effector away from the rest position for more time with respect to the standard case.
Despite those errors can be conceived as negligible, they will most likely appear also in a real scenario and maybe in an even stronger way but with more complete controls one the ending part of a detected force,with a further parameters calibration and with better sensor and system rate, those could be smoothed.

Chapter 6

Conclusions

The purpose of this thesis was to find a way of implementing generic manual guidance solutions into non-collaborative robots. The compliance behavior seen in cobots provides ways of creating a natural interaction between the robotic arm and its human operator.
Standard manipulators lack this feature but are consistently found in multiple industries that currently fulfill a certain task by designing hand driving mechanisms. Providing a human-robot interaction without completely replacing the robots themselves creates opportunities in these industries and provides quality behaviors that certainly help increase overall productivity.

To provide a clear understanding of formal compliance approaches, a thorough investigation into the theory underlying mechanical systems and robotic arm dynamics was conducted. It has to be stated that many others manual guidance solutions exist but that the majority of them rely on more practical mechanisms and sensor [13] while the exposed approach involves a more theoretical strategy in order to generalize this application.

In order to assess the compliance solutions and compare them with various manipulators, a simulation in Gazebo was used during development and testing. Nevertheless this is considered as a safe and stable environment so the developed methods will need to be further analyzed in a real robotic arm.

While the compliance solution itself is generalized and should work with any manipulator, the interface and middleware between the robotic arm and the compliance entity needs to be created for each specific robot.

The results of this thesis involve both a generalized hand guidance solution, a workspace limitation algorithm and a sinusoidal path planning, all features available for any manipulator.

To enhance the compliance features and provide precise control over the hand guidance behavior in particular situations, a conditional compliance mode is also proposed and despite being an uncommon feature, restricting the robot's movement while in manual guidance mode proved to be an elegant solution to fulfill certain trajectory programming tasks. The results of the experiences performed to test this mechanism proved that it can certainly be used in the discussed use-case scenarios.

Although the compliance solution as a whole works considerably well, some issues with this system have been faced. First and foremost, using the current readings to infer the joint torques isn't the most attractive solution. It is undoubtedly an option, but it has issues with high signal noise and temperature-dependent readings, where as more energy is lost as heat and other factors, the currents sent to the joint motors behave differently.

For what concern both the workspace limitation and the sinusoidal path planning it can be stated that the initial purposes have been fulfilled in a realization of working features.
In the case of the planning the actual limits are the mathematical instruments capable of designing the desired trajectories, the dexterity of the manipulator and its physical structure. All parameters

independent from the program.
In the case of the workspace limitation, a working software has been implemented but some difficulties arose treating complex shapes. With a proper computing availability a control on the end-effector could be applied to the feature itself for the sake of a more robust program solving the problem.

Appendices

Appendix A

Python scripts

In this appendix will be exposed all of the scripts used to implements the features

A.1 Starting script

The one below is the reference script used in order to implement all of the other features. It is a program that, through the use of a PD controller with reference to the position, keep the manipulator in the position and with the orientation decided by the rviz markers.

```python
#!/usr/bin/env python3

import copy
import rospy
import threading
import quaternion
import numpy as np
from geometry_msgs.msg import Point
from visualization_msgs.msg import *
from interactive_markers.interactive_marker_server import *
from franka_core_msgs.msg import EndPointState, JointCommand, RobotState

# -- add to pythonpath for finding rviz_markers.py
import sys, os
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
# -------------------------------------------------

from rviz_markers import RvizMarkers

# Task-space controller parameters
# stiffness gains
P_pos = 50.
P_ori = 25.
# damping gains
D_pos = 10.
D_ori = 1.
#my strange gains
P_vel=0.5   #it should include the 1/publis_rate component in order to get acc
    from differences and the mass one
P_omg=0.5
# -----------------------------------------
publish_rate = 100

JACOBIAN = None
CARTESIAN_POSE = None
CARTESIAN_VEL = None

destination_marker = RvizMarkers()

```

```python
def _on_robot_state(msg):
    """
        Callback function for updating jacobian and EE velocity from robot state
    """
    global JACOBIAN, CARTESIAN_VEL,Inertia,grav_comp,coriolis_comp,joint_v
    JACOBIAN = np.asarray(msg.O_Jac_EE).reshape(6,7,order = 'F')

    Inertia=np.asarray(msg.mass_matrix).reshape(7, 7, order='F') #mio
    grav_comp= np.asarray(msg.gravity).reshape(1, 7, order='F') #mio
    coriolis_comp= np.asarray(msg.coriolis).reshape(1, 7, order='F') #mio


    CARTESIAN_VEL = {
                'linear': np.asarray([msg.O_dP_EE[0], msg.O_dP_EE[1], msg.
    O_dP_EE[2]]),
                'angular': np.asarray([msg.O_dP_EE[3], msg.O_dP_EE[4], msg.
    O_dP_EE[5]]) }

def _on_endpoint_state(msg):
    """
        Callback function to get current end-point state
    """
    # pose message received is a vectorised column major transformation matrix
    global CARTESIAN_POSE,ext_f
    cart_pose_trans_mat = np.asarray(msg.O_T_EE).reshape(4,4,order='F')
    ext_f =msg.O_F_ext_hat_K #np.asarray(msg.O_F_ext_hat_K) #mio

    CARTESIAN_POSE = {
        'position': cart_pose_trans_mat[:3,3],
        'orientation': quaternion.from_rotation_matrix(cart_pose_trans_mat
    [:3,:3]) }



def quatdiff_in_euler(quat_curr, quat_des):
    """
        Compute difference between quaternions and return
        Euler angles as difference
    """
    curr_mat = quaternion.as_rotation_matrix(quat_curr)
    des_mat  = quaternion.as_rotation_matrix(quat_des)
    rel_mat = des_mat.T.dot(curr_mat)
    rel_quat = quaternion.from_rotation_matrix(rel_mat)
    vec = quaternion.as_float_array(rel_quat)[1:]
    if rel_quat.w < 0.0:
        vec = -vec

    return -des_mat.dot(vec)

def control_thread(rate):
    """
        Actual control loop. Uses goal pose from the feedback thread
        and current robot states from the subscribed messages to compute
        task-space force, and then the corresponding joint torques.
    """
    while not rospy.is_shutdown():
        error = 100.
        while error > 0.005:
            curr_pose = copy.deepcopy(CARTESIAN_POSE)
            curr_pos, curr_ori = curr_pose['position'],curr_pose['orientation']

            curr_vel = (CARTESIAN_VEL['linear']).reshape([3,1])
            curr_omg = CARTESIAN_VEL['angular'].reshape([3,1])

            delta_pos = (goal_pos - curr_pos).reshape([3,1])
            delta_ori = quatdiff_in_euler(curr_ori, goal_ori).reshape([3,1])

            # Desired task-space force using PD law
            F = np.vstack([P_pos*(delta_pos), P_ori*(delta_ori)]) - \
                np.vstack([D_pos*(curr_vel), D_ori*(curr_omg)])

            error = np.linalg.norm(delta_pos) + np.linalg.norm(delta_ori)

            J = copy.deepcopy(JACOBIAN)

            # joint torques to be commanded
            tau = np.dot(J.T,F)

            # publish joint commands
            command_msg.effort = tau.flatten()
            joint_command_publisher.publish(command_msg)
            #print("goal coords: ",goal_pos)
            rate.sleep()

def process_feedback(feedback):
    """
    InteractiveMarker callback function. Update target pose.
    """
    global goal_pos, goal_ori

    if feedback.event_type == InteractiveMarkerFeedback.MOUSE_UP:
        p = feedback.pose.position
        q = feedback.pose.orientation
        goal_pos = np.array([p.x,p.y,p.z])
        goal_ori = np.quaternion(q.w, q.x,q.y,q.z)

def _on_shutdown():
    """
        Clean shutdown controller thread when rosnode dies.
    """
    global ctrl_thread, cartesian_state_sub, \
        robot_state_sub, joint_command_publisher
    if ctrl_thread.is_alive():
        ctrl_thread.join()

    robot_state_sub.unregister()
    cartesian_state_sub.unregister()
    joint_command_publisher.unregister()

if __name__ == "__main__":
    # global goal_pos, goal_ori, ctrl_thread

    rospy.init_node("ts_control_sim_only")

    cartesian_state_sub = rospy.Subscriber(
        'panda_simulator/custom_franka_state_controller/tip_state',
        EndPointState,
        _on_endpoint_state,
        queue_size=1,
        tcp_nodelay=True)

    robot_state_sub = rospy.Subscriber(
        'panda_simulator/custom_franka_state_controller/robot_state',
        RobotState,
        _on_robot_state,
        queue_size=1,
        tcp_nodelay=True)

    # create joint command message and fix its type to joint torque mode
    command_msg = JointCommand()
    command_msg.names = ['panda_joint1','panda_joint2','panda_joint3','
   panda_joint4','panda_joint5','panda_joint6','panda_joint7']
```

```
    command_msg.mode = JointCommand.TORQUE_MODE

    # Also create a publisher to publish joint commands
    joint_command_publisher = rospy.Publisher(
            'panda_simulator/motion_controller/arm/joint_commands',
            JointCommand,
            tcp_nodelay=True,
            queue_size=1)

    # wait for messages to be populated before proceeding
    rospy.loginfo("Subscribing to robot state topics...")
    while (True):
        if not (JACOBIAN is None or CARTESIAN_POSE is None):
            break
    rospy.loginfo("Recieved messages; Starting Demo.")

    pose = copy.deepcopy(CARTESIAN_POSE)
    start_pos, start_ori = pose['position'],pose['orientation']
    goal_pos, goal_ori = start_pos, start_ori # set goal pose a starting pose in
     the beginning

    # start controller thread
    rospy.on_shutdown(_on_shutdown)
    rate = rospy.Rate(publish_rate)
    ctrl_thread = threading.Thread(target=control_thread, args = [rate])
    ctrl_thread.start()
    #
    ----------------------------------------------------------------------------------

    server = InteractiveMarkerServer("basic_control")

    position = Point( start_pos[0], start_pos[1], start_pos[2])
    marker = destination_marker.makeMarker( False, InteractiveMarkerControl.
    MOVE_ROTATE_3D, \
                                            position, quaternion.as_float_array(
    start_ori), True)
    server.insert(marker, process_feedback)

    server.applyChanges()

    rospy.spin()
    #
    ----------------------------------------------------------------------------------
```

A.2 Workspace limitation script

For the Workspace limitation feature it was enough to just change the process function in order to modify the program's goal with the desired limited one. Here below it is possible to see that method.

```

#limited workspace specifics
spherical_workspace_range=0.5
spherical_workspace_center_x0= 0.17
spherical_workspace_center_y0= -0.21
spherical_workspace_center_z0= 0.5
# ----------------------------------------


def process_feedback(feedback):
    """
    InteractiveMarker callback function that update target pose forcing it
    inside the boundaries
    """
    global goal_pos, goal_ori

    if feedback.event_type == InteractiveMarkerFeedback.MOUSE_UP:
        p = feedback.pose.position
        q = feedback.pose.orientation

        #check range
        x_shift=p.x-spherical_workspace_center_x0
        y_shift=p.y-spherical_workspace_center_y0
        z_shift=p.z-spherical_workspace_center_z0
        rho=np.sqrt(np.power(x_shift,2)+np.power(y_shift,2)+np.power(z_shift,2))
        if(rho>spherical_workspace_range):
            p.x=x_shift*spherical_workspace_range/rho+spherical_workspace_center_x0
            p.y=y_shift*spherical_workspace_range/rho+spherical_workspace_center_y0
            p.z=z_shift*spherical_workspace_range/rho+spherical_workspace_center_z0
            print("out of boundaries")
        goal_pos = np.array([p.x,p.y,p.z])
        goal_ori = np.quaternion(q.w, q.x,q.y,q.z)
```

A.3 Sinusoidal path planning script

Here is reported the sinusoidal path planning code entirely.

```
#!/usr/bin/env python3
"""

    After launching the simulator (panda_world.launch),
    run this demo using the command:

        roslaunch esperimentopacco my_demo_task_space_control.launch

"""

import copy
import rospy
import threading
import quaternion
import numpy as np
from geometry_msgs.msg import Point
from visualization_msgs.msg import *
from interactive_markers.interactive_marker_server import *
from franka_core_msgs.msg import EndPointState, JointCommand, RobotState

# -- add to pythonpath for finding rviz_markers.py
import sys, os
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
# -------------------------------------------------

from rviz_markers import RvizMarkers

from gazebo_msgs.srv import ApplyBodyWrench
from geometry_msgs.msg import Wrench

def applyForce(): rospy.wait_for_service('/gazebo/apply_body_wrench')
force = rospy.ServiceProxy('/gazebo/apply_body_wrench',ApplyBodyWrench)

# --------- Modify as required ------------
# Task-space controller parameters
# stiffness gains
P_pos = 50.
P_ori = 25.
```

```python
# damping gains
D_pos = 10.
D_ori = 1.
#my strange gains
P_vel=0.5   #it should include the 1/publis_rate component in order to get acc
    from differences and the mass one
P_omg=0.5
# -----------------------------------------
publish_rate = 100

JACOBIAN = None
CARTESIAN_POSE = None
CARTESIAN_VEL = None

destination_marker = RvizMarkers()


def _on_robot_state(msg):
    """
        Callback function for updating jacobian and EE velocity from robot state
    """
    global JACOBIAN, CARTESIAN_VEL,Inertia,grav_comp,coriolis_comp,joint_v
    JACOBIAN = np.asarray(msg.O_Jac_EE).reshape(6,7,order = 'F')

    CARTESIAN_VEL = {
                'linear': np.asarray([msg.O_dP_EE[0], msg.O_dP_EE[1], msg.
    O_dP_EE[2]]),
                'angular': np.asarray([msg.O_dP_EE[3], msg.O_dP_EE[4], msg.
    O_dP_EE[5]]) }

def _on_endpoint_state(msg):
    """
        Callback function to get current end-point state
    """
    # pose message received is a vectorised column major transformation matrix
    global CARTESIAN_POSE,ext_f
    cart_pose_trans_mat = np.asarray(msg.O_T_EE).reshape(4,4,order='F')

    CARTESIAN_POSE = {
        'position': cart_pose_trans_mat[:3,3],
        'orientation': quaternion.from_rotation_matrix(cart_pose_trans_mat
    [:3,:3]) }



def quatdiff_in_euler(quat_curr, quat_des):
    """
        Compute difference between quaternions and return
        Euler angles as difference
    """
    curr_mat = quaternion.as_rotation_matrix(quat_curr)
    des_mat  = quaternion.as_rotation_matrix(quat_des)
    rel_mat = des_mat.T.dot(curr_mat)
    rel_quat = quaternion.from_rotation_matrix(rel_mat)
    vec = quaternion.as_float_array(rel_quat)[1:]
    if rel_quat.w < 0.0:
        vec = -vec

    return -des_mat.dot(vec)

def sinusoidal_control_thread(rate):
    t=0
    while not rospy.is_shutdown():
        error = 100.
        while error > 0.005:
            if(t<5):
                z=0.55 + 0.036*t*t - 0.0048*t*t*t #calculated with cubic polynomial
                t=t+0.01  #1/rate
            else:
                z=-0.65 +0.72*t- 0.108*t*t + 0.0048*t*t*t
                t=t+0.01
                if(t>10):
                    t=0
            goal_pos=[0.31091106, 0.00948111, z]

            curr_pose = copy.deepcopy(CARTESIAN_POSE)
            curr_pos, curr_ori = curr_pose['position'],curr_pose['orientation']
            curr_vel = (CARTESIAN_VEL['linear']).reshape([3,1])
            curr_omg = CARTESIAN_VEL['angular'].reshape([3,1])
            delta_pos = (goal_pos - curr_pos).reshape([3,1])
            delta_ori = quatdiff_in_euler(curr_ori, goal_ori).reshape([3,1])
            # Desired task-space force using PD law
            F = np.vstack([P_pos*(delta_pos), P_ori*(delta_ori)]) - \
                np.vstack([D_pos*(curr_vel), D_ori*(curr_omg)])
            error = np.linalg.norm(delta_pos) + np.linalg.norm(delta_ori)
            J = copy.deepcopy(JACOBIAN)
            tau = np.dot(J.T,F)
            command_msg.effort = tau.flatten()
            joint_command_publisher.publish(command_msg)
            #print("t: ", t," z: ", z)
            rate.sleep()

def circular_control_thread(rate):
    t=0
    while not rospy.is_shutdown():
        error = 100.
        first_spin=False
        while error > 0.005:
            if(t<2.5):
                y=9.48111000e-03 + 0.072*t*t - 0.0192*t*t*t
                if(first_spin):
                    y=0.00948111-0.09*t +0.0048 *t*t*t
                z=0.55 + 0.036*t*t - 0.0048*t*t*t #calculated with cubic polynomial
                t=t+0.01  #1/rate
                print("parte uno")
            if(t>=2.5 and t<5):
                y= 0.15948111-0.27*t + 0.072*t*t - 0.0048*t*t*t
                z=0.55 + 0.036*t*t - 0.0048*t*t*t
                t=t+0.01
                print("parte due")
            if(t>=5 and t<7.5):
                y= 0.15948111-0.27*t + 0.072*t*t - 0.0048*t*t*t
                z=-0.65 +0.72*t- 0.108*t*t + 0.0048*t*t*t
                t=t+0.01
                print("parte tre")
            if(t>=7.5):
                y=-3.89051889+1.35*t -0.144 *t*t +  0.0048 *t*t*t
                z=-0.65 +0.72*t- 0.108*t*t + 0.0048*t*t*t
                t=t+0.01
                print("parte quattro")
                if(t>10):
                    t=0
                    first_spin=True

            goal_pos=[0.31091106, y, z]

            curr_pose = copy.deepcopy(CARTESIAN_POSE)
            curr_pos, curr_ori = curr_pose['position'],curr_pose['orientation']
            curr_vel = (CARTESIAN_VEL['linear']).reshape([3,1])
            curr_omg = CARTESIAN_VEL['angular'].reshape([3,1])
            delta_pos = (goal_pos - curr_pos).reshape([3,1])
            delta_ori = quatdiff_in_euler(curr_ori, goal_ori).reshape([3,1])
            # Desired task-space force using PD law
            F = np.vstack([P_pos*(delta_pos), P_ori*(delta_ori)]) - \
                np.vstack([D_pos*(curr_vel), D_ori*(curr_omg)])
            error = np.linalg.norm(delta_pos) + np.linalg.norm(delta_ori)
            J = copy.deepcopy(JACOBIAN)
            tau = np.dot(J.T,F)
            command_msg.effort = tau.flatten()
            joint_command_publisher.publish(command_msg)
            #print("t: ", t,"  z: ", z)
            rate.sleep()

def process_feedback(feedback):
    """
    InteractiveMarker callback function. Update target pose.
    """
    global goal_pos, goal_ori

    if feedback.event_type == InteractiveMarkerFeedback.MOUSE_UP:
        p = feedback.pose.position
        q = feedback.pose.orientation
        goal_pos = np.array([p.x,p.y,p.z])
        goal_ori = np.quaternion(q.w, q.x,q.y,q.z)



def _on_shutdown():
    """
        Clean shutdown controller thread when rosnode dies.
    """
    global ctrl_thread, cartesian_state_sub, \
        robot_state_sub, joint_command_publisher
    if ctrl_thread.is_alive():
        ctrl_thread.join()

    robot_state_sub.unregister()
    cartesian_state_sub.unregister()
    joint_command_publisher.unregister()
################################################################################
if __name__ == "__main__":
    # global goal_pos, goal_ori, ctrl_thread

    rospy.init_node("ts_control_sim_only")

    # if not using franka_ros_interface, you have to subscribe to the right
    topics
    # to obtain the current end-effector state and robot jacobian for computing
    # commands
    cartesian_state_sub = rospy.Subscriber(
        'panda_simulator/custom_franka_state_controller/tip_state',
        EndPointState,
        _on_endpoint_state,
        queue_size=1,
        tcp_nodelay=True)

    robot_state_sub = rospy.Subscriber(
        'panda_simulator/custom_franka_state_controller/robot_state',
        RobotState,
        _on_robot_state,
        queue_size=1,
        tcp_nodelay=True)

    # create joint command message and fix its type to joint torque mode
    command_msg = JointCommand()
    command_msg.names = ['panda_joint1','panda_joint2','panda_joint3','panda_joint4','panda_joint5','panda_joint6','panda_joint7']
    command_msg.mode = JointCommand.TORQUE_MODE

    # Also create a publisher to publish joint commands
    joint_command_publisher = rospy.Publisher(
            'panda_simulator/motion_controller/arm/joint_commands',
            JointCommand,
            tcp_nodelay=True,
            queue_size=1)

    # wait for messages to be populated before proceeding
    rospy.loginfo("Subscribing to robot state topics...")
    while (True):
        if not (JACOBIAN is None or CARTESIAN_POSE is None):
            break
    rospy.loginfo("Recieved messages; Starting Demo.")

    pose = copy.deepcopy(CARTESIAN_POSE)
    start_pos, start_ori = pose['position'],pose['orientation']
    goal_pos, goal_ori = start_pos, start_ori # set goal pose a starting pose in the beginning

    # start controller thread
    rospy.on_shutdown(_on_shutdown)
    rate = rospy.Rate(publish_rate)
    #ctrl_thread = threading.Thread(target=sinusoidal_control_thread, args = [rate]) depending of what I want to use
    ctrl_thread = threading.Thread(target=circular_control_thread, args = [rate])
    ctrl_thread.start()
    # ----------------------------------------------------------------------------------

    server = InteractiveMarkerServer("basic_control")

    position = Point( start_pos[0], start_pos[1], start_pos[2])
    marker = destination_marker.makeMarker( False, InteractiveMarkerControl.MOVE_ROTATE_3D, \
                                    position, quaternion.as_float_array(start_ori), True)
    server.insert(marker, process_feedback)

    server.applyChanges()

    rospy.spin()
    # ----------------------------------------------------------------------------------
```

A.4 Compliance control script

Here is reported the compliance control script.

```python
#!/usr/bin/env python3
"""

    After launching the simulator (panda_world.launch),
    run this demo using the command:
        roslaunch panda_gazebo panda_world.launch

        roslaunch esperimentopacco force_control_1f_using_sim_only.py.launch

"""

```

```python
import copy
import rospy
import threading
import quaternion
import numpy as np
import matplotlib.pyplot as plt
from geometry_msgs.msg import Point
from visualization_msgs.msg import *
from interactive_markers.interactive_marker_server import *
from franka_core_msgs.msg import EndPointState, JointCommand, RobotState

# -- add to pythonpath for finding rviz_markers.py
import sys, os
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
# -------------------------------------------------

from rviz_markers import RvizMarkers

from gazebo_msgs.srv import ApplyBodyWrench
from geometry_msgs.msg import Wrench

def applyForce(): rospy.wait_for_service('/gazebo/apply_body_wrench')
force = rospy.ServiceProxy('/gazebo/apply_body_wrench',ApplyBodyWrench)

# --------- Modify as required ------------
# Task-space controller parameters
# stiffness gains
P_pos = 50.
P_ori = 25.
# damping gains
D_pos = 10.
D_ori = 1.
#my strange gains
P_vel=0.5   #it should include the 1/publis_rate component in order to get acc
    from differences and the mass one
P_omg=0.5
# -----------------------------------------
publish_rate = 100

JACOBIAN = None
CARTESIAN_POSE = None
CARTESIAN_VEL = None

Inertia = None #mio
grav_comp=None #mio
coriolis_comp=None #mio
ext_f= None #mio

joint_v = None #mio
joint_eff = None #mio
#-----------------------------------------------for compliance control
filter_const=0.3 #percentage under which I don't consider the detected value of
    the forces components
k_impedence=20   #threshold(in N) for which I don't consider forces
k_force=1.2      #factor that empower the force I apply on the machine
security_threshold=200 #maximum force

detected_force=None

n=0.5*np.sqrt(2)
Rot45=[[n,n,0],[-n,n,0],[0,0,1]]#da capire come mai il segno

destination_marker = RvizMarkers()


def _on_robot_state(msg):
    """
        Callback function for updating jacobian and EE velocity from robot state
    """
    global JACOBIAN, CARTESIAN_VEL,Inertia,grav_comp,coriolis_comp,joint_v
    JACOBIAN = np.asarray(msg.O_Jac_EE).reshape(6,7,order = 'F')

    Inertia=np.asarray(msg.mass_matrix).reshape(7, 7, order='F') #mio
    grav_comp= np.asarray(msg.gravity).reshape(1, 7, order='F') #mio
    coriolis_comp= np.asarray(msg.coriolis).reshape(1, 7, order='F') #mio


    CARTESIAN_VEL = {
                'linear': np.asarray([msg.O_dP_EE[0], msg.O_dP_EE[1], msg.O_dP_EE[2]]),
                'angular': np.asarray([msg.O_dP_EE[3], msg.O_dP_EE[4], msg.O_dP_EE[5]]) }

def _on_endpoint_state(msg):
    """
        Callback function to get current end-point state
    """
    # pose message received is a vectorised column major transformation matrix
    global CARTESIAN_POSE, ext_f
    cart_pose_trans_mat = np.asarray(msg.O_T_EE).reshape(4,4,order='F')
    ext_f =msg.O_F_ext_hat_K  #mio

    CARTESIAN_POSE = {
        'position': cart_pose_trans_mat[:3,3],
        'orientation': quaternion.from_rotation_matrix(cart_pose_trans_mat[:3,:3]) }



def quatdiff_in_euler(quat_curr, quat_des):
    """
        Compute difference between quaternions and return
        Euler angles as difference
    """
    curr_mat = quaternion.as_rotation_matrix(quat_curr)
    des_mat  = quaternion.as_rotation_matrix(quat_des)
    rel_mat = des_mat.T.dot(curr_mat)
    rel_quat = quaternion.from_rotation_matrix(rel_mat)
    vec = quaternion.as_float_array(rel_quat)[1:]
    if rel_quat.w < 0.0:
        vec = -vec

    return -des_mat.dot(vec)

def control_thread(rate):
    """
        Actual control loop. Uses goal pose from the feedback thread
        and current robot states from the subscribed messages to compute
        task-space force, and then the corresponding joint torques.
    """
    while not rospy.is_shutdown():
        error = 100.
        while error > 0.005:
            curr_pose = copy.deepcopy(CARTESIAN_POSE)
            curr_pos, curr_ori = curr_pose['position'],curr_pose['orientation']

            curr_vel = (CARTESIAN_VEL['linear']).reshape([3,1])
            curr_omg = CARTESIAN_VEL['angular'].reshape([3,1])

            delta_pos = (goal_pos - curr_pos).reshape([3,1])
            delta_ori = quatdiff_in_euler(curr_ori, goal_ori).reshape([3,1])

```

```python
            # Desired task-space force using PD law
            F = np.vstack([P_pos*(delta_pos), P_ori*(delta_ori)]) - np.vstack([D_pos*(curr_vel), D_ori*(curr_omg)])

            error = np.linalg.norm(delta_pos) + np.linalg.norm(delta_ori)

            J = copy.deepcopy(JACOBIAN)

            # joint torques to be commanded
            tau = np.dot(J.T,F)

            # publish joint commands
            command_msg.effort = tau.flatten()
            joint_command_publisher.publish(command_msg)
            #print("goal coords: ",goal_pos)
            rate.sleep()

def full_control(rate):
    while not rospy.is_shutdown():
        t=0 #time part just for force simulation
        error = 100.
        ext_f_i=copy.deepcopy(ext_f)
        ext_f_i=[ext_f_i.wrench.force.x, ext_f_i.wrench.force.y, ext_f_i.wrench.force.z]
        ext_f_i= np.matmul(Rot45,ext_f_i)

        while (error > 0.005) or ((np.abs(ext_f_i[0]>2))or(np.abs(ext_f_i[1]>2))or(np.abs(ext_f_i[2])>2)):
            '''# for simulating a force
            if (t>1 and t<1.01):
                wrench          = Wrench()
                wrench.force.x  = 0
                wrench.force.y  = 0
                wrench.force.z  = -100
                wrench.torque.x = 0
                wrench.torque.y = 0
                wrench.torque.z = 0
                force(body_name = "panda::panda_link7",wrench = wrench, duration = rospy.Duration(3))
                print("force/torque applied")
            '''
            ext_f_i=copy.deepcopy(ext_f)
            ext_f_i=[ext_f_i.wrench.force.x, ext_f_i.wrench.force.y, ext_f_i.wrench.force.z]
            ext_f_i= np.matmul(Rot45,ext_f_i)
            if((np.abs(ext_f_i[0])>k_impedence)or(np.abs(ext_f_i[1])>k_impedence)or(np.abs(ext_f_i[2])>k_impedence)):
                detected_force=[50,50,50]
                while ((np.abs(detected_force[0])>k_impedence)       or (np.abs(detected_force[1])>k_impedence)       or (np.abs(detected_force[2])>k_impedence)) and \
                      ((np.abs(detected_force[0])<security_threshold)and(np.abs(detected_force[1])<security_threshold)and(np.abs(detected_force[2])<security_threshold)):

                    detected_force=[0,0,0,0,0,0]
                    ext_f_c = copy.deepcopy(ext_f)
                    measured_forces=[ext_f_c.wrench.force.x, ext_f_c.wrench.force.y, ext_f_c.wrench.force.z]
                    measured_forces= np.matmul(Rot45,measured_forces)
                    abs_force=[np.abs(measured_forces[0]),np.abs(measured_forces[1]),np.abs(measured_forces[2])]#since the abs function on the array cut some values
                    if(np.max(abs_force)==np.abs(measured_forces[0])):
                        detected_force[0]=measured_forces[0]
                        if(filter_const*np.max(abs_force)<=np.abs(measured_forces[1])):                     #force-error check
                            detected_force[1]=measured_forces[1]
                        if(filter_const*np.max(abs_force)<=np.abs(measured_forces[2])):
                            detected_force[2]=measured_forces[2]
                    if(np.max(abs_force)==np.abs(measured_forces[1])):
                        detected_force[1]=measured_forces[1]
                        if(filter_const*np.max(abs_force)<=np.abs(measured_forces[0])):
                            detected_force[0]=measured_forces[0]
                        if(filter_const*np.max(abs_force)<=np.abs(measured_forces[2])):
                            detected_force[2]=measured_forces[2]
                    if(np.max(abs_force)==np.abs(measured_forces[2])):
                        detected_force[2]=measured_forces[2]
                        if(filter_const*np.max(abs_force)<=np.abs(measured_forces[1])):
                            detected_force[1]=measured_forces[1]
                        if(filter_const*np.max(abs_force)<=np.abs(measured_forces[0])):
                            detected_force[0]=measured_forces[0]
                    print("detected and filtered force ",detected_force)
                    J = copy.deepcopy(JACOBIAN)
                    tau = np.dot(J.T,detected_force)
                    command_msg.effort = tau.flatten()
                    joint_command_publisher.publish(command_msg)
                    t=t+0.01
                    rate.sleep()
            else:
                if ((np.abs(ext_f_i[0])>security_threshold) and (np.abs(ext_f_i[1])>security_threshold)and(np.abs(ext_f_i[2])>security_threshold)):
                    print("force too high, could be dangerous")
                ext_f_i=copy.deepcopy(ext_f)
                ext_f_i=[ext_f_i.wrench.force.x, ext_f_i.wrench.force.y, ext_f_i.wrench.force.z]
                ext_f_i= np.matmul(Rot45,ext_f_i)
                k_decrease=0.1
                ext_f_i=[-k_decrease*ext_f_i[0],-k_decrease*ext_f_i[1],-k_decrease*ext_f_i[2],0,0,0]
                curr_pose = copy.deepcopy(CARTESIAN_POSE)
                curr_pos, curr_ori = curr_pose['position'],curr_pose['orientation']
                curr_vel = (CARTESIAN_VEL['linear']).reshape([3,1])
                curr_omg = CARTESIAN_VEL['angular'].reshape([3,1])
                delta_pos = (goal_pos - curr_pos).reshape([3,1])
                delta_ori = quatdiff_in_euler(curr_ori, goal_ori).reshape([3,1])

                error = np.linalg.norm(delta_pos) + np.linalg.norm(delta_ori)
                F = np.vstack([P_pos*(delta_pos), P_ori*(delta_ori)]) - np.vstack([D_pos*(curr_vel), D_ori*(curr_omg)])
                F_with_ext= F + ext_f_i
                J = copy.deepcopy(JACOBIAN)

                tau = np.dot(J.T,F_with_ext)
                command_msg.effort = tau.flatten()
                joint_command_publisher.publish(command_msg)
                t=t+0.01
                rate.sleep()


def process_feedback(feedback):
    """
    InteractiveMarker callback function. Update target pose.
    """
    global goal_pos, goal_ori
```

```python

    if feedback.event_type == InteractiveMarkerFeedback.MOUSE_UP:
        p = feedback.pose.position
        q = feedback.pose.orientation
        goal_pos = np.array([p.x,p.y,p.z])
        goal_ori = np.quaternion(q.w, q.x,q.y,q.z)



def _on_shutdown():
    """
        Clean shutdown controller thread when rosnode dies.
    """
    global ctrl_thread, cartesian_state_sub, \
        robot_state_sub, joint_command_publisher
    if ctrl_thread.is_alive():
        ctrl_thread.join()

    robot_state_sub.unregister()
    cartesian_state_sub.unregister()
    joint_command_publisher.unregister()
################################################################################
if __name__ == "__main__":
    # global goal_pos, goal_ori, ctrl_thread

    rospy.init_node("ts_control_sim_only")


    # if not using franka_ros_interface, you have to subscribe to the right
    topics
    # to obtain the current end-effector state and robot jacobian for computing
    # commands
    cartesian_state_sub = rospy.Subscriber('panda_simulator/
    custom_franka_state_controller/tip_state',EndPointState,_on_endpoint_state,
    queue_size=1,tcp_nodelay=True)

    robot_state_sub = rospy.Subscriber( 'panda_simulator/
    custom_franka_state_controller/robot_state',RobotState, _on_robot_state,
    queue_size=1,  tcp_nodelay=True)

    # create joint command message and fix its type to joint torque mode
    command_msg = JointCommand()
    command_msg.names = ['panda_joint1','panda_joint2','panda_joint3','
    panda_joint4','panda_joint5','panda_joint6','panda_joint7']
    command_msg.mode = JointCommand.TORQUE_MODE

    # Also create a publisher to publish joint commands
    joint_command_publisher = rospy.Publisher('panda_simulator/motion_controller
    /arm/joint_commands', JointCommand,tcp_nodelay=True, queue_size=1)

    # wait for messages to be populated before proceeding
    rospy.loginfo("Subscribing to robot state topics...")
    while (True):
        if not (JACOBIAN is None or CARTESIAN_POSE is None):
            break
    rospy.loginfo("Recieved messages; Starting Demo.")

    pose = copy.deepcopy(CARTESIAN_POSE)
    start_pos, start_ori = pose['position'],pose['orientation']
    print(start_ori)
    goal_pos, goal_ori = start_pos, start_ori # set goal pose a starting pose in
     the beginning

    #print("velocities: ",(CARTESIAN_VEL['linear']).reshape([1,3]))#mio
    #print("Inertia: ",Inertia) #mio
    #print("grav_comp: ",grav_comp) #mio
    #print("coriolis_comp: ",coriolis_comp) #mio
    #print("ext_f: ",ext_f.wrench.force) #mio

    # start controller thread
    rospy.on_shutdown(_on_shutdown)
    rate = rospy.Rate(publish_rate)
    #ctrl_thread = threading.Thread(target=control_thread, args = [rate]) #
codice vero
    #ctrl_thread = threading.Thread(target=vel_contr, args = [rate]) #mio



    #little delay
    for i in range(5):
        rate_long = rospy.Rate(100)
        rate_long.sleep()

    #
---------------------------------------------------------------------------------------

    ctrl_thread = threading.Thread(target=full_control, args = [rate])
    #ctrl_thread = threading.Thread(target=test_impedance, args = [rate])
    ctrl_thread.start()
    print("end of the thread")
    #
---------------------------------------------------------------------------------

    server = InteractiveMarkerServer("basic_control")

    position = Point( start_pos[0], start_pos[1], start_pos[2])
    marker = destination_marker.makeMarker( False, InteractiveMarkerControl.
MOVE_ROTATE_3D, position, quaternion.as_float_array(start_ori), True)
    server.insert(marker, process_feedback)

    server.applyChanges()
    rospy.spin()
    #
---------------------------------------------------------------------------------
```

Bibliography

[1] Giacomini, Paolo. "Il rischio macchine nell'era Industria 4.0: robot collaborativi, nuove possibilità e nuovi rischi."

[2] Scibilia, Lavinia. "La fabbrica del futuro: uomo e robot al servizio del business in uno spazio di lavoro condiviso." (2018).

[3] https://robot.omitech.it/cobot-storia-applicazioni-robot-collaborativi/

[4] https://ifr.org/ifr-press-releases/news/robot-sales-rise-again

[5] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani(2010), Robotics: Modelling, Planning and Control

[6] https://www.ros.org/

[7] https://www.franka.de/

[8] https://frankaemika.github.io/docs/franka_ros.html

[9] https://frankaemika.github.io/docs/control_parameters.html

[10] http://wiki.ros.org/rviz

[11] Peshkin, M. and Colgate, J.E. (1999), "Cobots", Industrial Robot

[12] https://devicebase.net/en/franka-emika-panda-robot/questions/what-is-the-speed-of-joint-movement/3bw

[13] C. Escobedo, M. Strong, M. West, A. Aramburu and A. Roncone, "Contact Anticipation for Physical Human–Robot Interaction with Robotic Manipulators using Onboard Proximity Sensors," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021, pp. 7255-7262

[14] Denier van der Gon, J J, ter Haar Romeny, B M, van Zuylen, E J, (1985), Behaviour of motor units of human arm muscles: differences between slow isometric contraction and relaxation

[15] Fanny Ficuciello, Villani Luigi, and Bruno Siciliano. Impedance control of redundant manipulators for safe human-robot collaboration. Acta Polytechnica Hungarica , 13:223–238, 01 2016.

[16] Siciliano Bruno, et al. Force control. Springer London, 2009.

[17] Tondu, B. and Lopez, P. (1997), "The McKibben muscle and its use in actuating robot-arms showing similarities with human arm behaviour", Industrial Robot

[18] Siciliano, Bruno, Oussama Khatib, and Torsten Kröger, eds. Springer handbook of robotics. Vol. 200. Berlin: springer, 2008.

[19] C. Escobedo, M. Strong, M. West, A. Aramburu and A. Roncone, "Contact Anticipation for Physical Human–Robot Interaction with Robotic Manipulators using Onboard Proximity Sensors," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021

[20] Nejc Likar and Leon Zlajpah. External joint torque-based estimation of contact information.

International Journal of Advanced Robotic Systems

[21] Frederik Schmatz, Florian Beuß, Jan Sender, and Wilko Fl ugge. Use of human-robot col- laboration to enhance process monitoring of mechanical joining. Procedia Manufacturing , 52:272–276, 2020. System-Integrated Intelligence – Intelligent, Flexible and Connected Systems in Products and ProductionProceedings of the 5th International Conference on System-Integrated Intelligence (SysInt 2020), Bremen, Germany

[22] Kaj Madsen, Hans Nielsen, and O Tingleff. Methods for non-linear least squares problems (2nd ed.). page 60, 01 2004. Technical University of Denmark

[23] Bruno Siciliano. Kinematic control of redundant robot manipulators: A tutorial. Journal of Intelligent and Robotic Systems

[24] Jilai Song, Fang Xu, Fengshan Zou, Shouliang Chen, and Bin Zhao. Joint torque detec- tion based on motor current and singular perturbation control for cleaning room manip- ulator. In 2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER) , pages 1609–1614, 2017

[25] Mosher, Ralph S. "Industrial manipulators." Scientific American 211.4 (1964)

[26] Abderrahim, Mohamed, et al. "Accuracy and calibration issues of industrial manipulators." Industrial robotics: programming, simulation and application (2004): 131-146.

[27] Khalil, Wisama, Maxime Gautier, and Philippe Lemoine. "Identification of the payload inertial parameters of industrial manipulators." Proceedings 2007 IEEE International Conference on Robotics and Automation. IEEE, 2007.

[28] Myers, Donald R., and Diana F. Gordon. "Kinematic equations for industrial manipulators." Industrial Robot: An International Journal (1982).

[29] Omodei, Alberto, Giovanni Legnani, and Riccardo Adamini. "Three methodologies for the calibration of industrial manipulators: Experimental results on a SCARA robot." Journal of Robotic Systems 17.6 (2000): 291-307.

[30] Tobias Gold, Andreas V olz, and Knut Graichen. External torque estimation for an industrial robot arm using joint torsion and motor current measurements. IFAC- PapersOnLine , 52(15):352–357, 2019. 8th IFAC Symposium on Mechatronic Systems MECHATRONICS 2019

[31] Jeongseok Lee, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Sid- dhartha S. Srinivasa, Mike Stilman, and C. Karen Liu. DART: Dynamic animation and robotics toolkit. Journal of Open Source Software , 3(22):500, 2018

[32] P.I. Corke. A robotics toolbox for matlab. IEEE Robotics Automation Magazine , 3(1):24– 32, 1996

[33] Nuño, Emmanuel, et al. "A globally stable PD controller for bilateral teleoperators." IEEE Transactions on Robotics 24.3 (2008): 753-758.

[34] Brian S. Eberman and J. Kenneth Salisbury. Determination of manipulator contact information from joint torque measurements. In Vincent Hayward and Oussama Khatib, editors, Experimental Robotics I , pages 463–473, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg

[35] A. Colom e, D. Pardo, G. Aleny‘a, and C. Torras. External force estimation during compliant robot manipulation. In 2013 IEEE International Conference on Robotics and Automation , pages 3535–3540, 2013

[36] Leahy Jr, Michael B., and George N. Saridis. "Compensation of industrial manipulator dynamics." The International Journal of Robotics Research 8.4 (1989): 73-84.

[37] Hsia, TC Steve, Ty A. Lasky, and Zhengyu Guo. "Robust independent joint controller design for industrial robot manipulators." IEEE transactions on industrial electronics 38.1 (1991): 21-25.

[38] Jin, Jingfu, and Nicholas Gans. "Parameter identification for industrial robots with a fast and robust trajectory design approach." Robotics and Computer-Integrated Manufacturing 31 (2015): 21-29.