Cooperative Robotics

Authors: "Andrea Chiappe, Alberto Didonna, Fabio Guelfi, Samuele Viola"
ID: "4673275, 4944656, 5004782, 6466077"
Date: "8/1/2025"

General notes

- Exercise 1 is done with the ROBUST matlab main and unity visualization tools. Exercises 2-3 are done with the Franka simulation tools.
- Comment and discuss the simulations, in a concise scientific manner. Further comments, other than the questions, can be added, although there is no need to write 10 pages for each exercise.
- Aid the discussion with screenshots of the simulated environment (compress them to maintain a small overall file size), and graphs of the relevant variables (i.e. activation functions of inequality tasks, task variables, and so on). Graphs should always report the units of measure in both axes, and legends whenever relevant.
- Report the thresholds whenever relevant.
- Report the mathematical formula employed to derive the task jacobians and the control laws when asked, including where they are projected.
- If needed, part of the code can be inserted as a discussion reference.
- Provide your answers in the subsection where the question is made.

Use the following template when you need to discuss the hierarchy of tasks of a given action or set of actions. Please report the tasks in the same order as in the unified hierarchy implemented in your code.

Table 1: Example of actions/hierarchy table: a number in a given cell represents the priority of the control task (row) in the hierarchy of the control action (column). The type column indicates whether the objective is an equality (E) or inequality (I) one.

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
Task name A	I	1		1
Task name B	I	2	1	
Task name C	E		2	2

1 Exercise 1: Implement a Complete Mission

Implement several actions to reach a point, land, and then perform the manipulation of a target object.

Initialize the vehicle at the position:

$$\begin{bmatrix} 8.5 & 38.5 & -36 & 0 & -0.06 & 0.5 \end{bmatrix}^\top$$

Use a "safe waypoint navigation action" to reach the following position:

$$\begin{bmatrix} 10.5 & 37.5 & -38 & 0 & -0.06 & 0.5 \end{bmatrix}^\top$$

Then land, aligning to the nodule. In particular, the x axis of the vehicle should align to the projection, on the inertial horizontal plane, of the unit vector joining the vehicle frame to the nodule frame. Once landed, implement a "fixed-based manipulation action" to reach the target nodule (mimicking the scanning of the nodule). During this manipulation phase, the vehicle should not move for any reason.

1.1 Q1: Report the unified hierarchy of tasks used and their priorities in each action.

MA = Minimum Altitude Task (SAFETY, INEQUALITY)
HA = Horizontal Attitude Task (SAFETY, INEQUALITY)
VP = Vehicle Position Task (CONTROL, EQUALITY)
VH = Vehicle Heading Task (CONTROL, EQUALITY)
AC = Attitude Control Task (CONTROL, EQUALITY)
ACL = Altitude Control for Landing Task (CONTROL, EQUALITY)
RN = Reaching Nodule Task (CONTROL, EQUALITY)
JL = Joint Limits Task (SAFETY, INEQUALITY)

The priorities are established by the order in which tasks are written:

Task	Type	Safety Navigation	Landing	Ground Manipulation
Joint Limits	I			2
Minimum altitude	I	1		
Horizontal attitude	I	2	1	1
Vehicle Heading	E	3		3
Vehicle Heading 2	E		2	
Altitude to 0	E		3	
Vehicle Position	E	4		
Vehicle Position 2	E		4	4
Attitude Control	E	5		
Reaching nodule	E			5

Table 2: Table to show the relation between the tasks (columns) and the actions of each specific task (rows) showing the priorities

1.2 Q2: Comment the behaviour of the robots, supported by relevant plots.

The robot's behaviour is splitted in 3 phases:

- **Phase 1**
  In this phase, the robot operates in Safety Waypoint Navigation, and because of this, it moves in order to reach the vehicle goal position and orientation $\begin{bmatrix} 10.5 & 37.5 & -38 & 0 & -0.06 & 0.5 \end{bmatrix}$, being sure it doesn't go too low in altitude (MA), and that it maintains an horizontal alignment (HA).

  For doing this, the quantities controlled are: (x,y,z) by the VP task, yaw by the VH task, and (pitch, roll) controlled by AC task. Note that we decided to separate the control of yaw and

(pitch, roll) in two different tasks because of the fact that, in an underwater robot, is more important to control the rotation around z axis, in comparison to the other rotations. So we separated them in order to give more priority to the yaw control. When the robot reaches the goal (or a value approximating the goal), the system moves to phase 2.

- **Phase 2**
  In this phase the robot aligns (always maintaining an horizontal attitude HA) to the object that it has to inspect, and, at the same time, lands in the sea floor. This is done by two different tasks: VH (which is different from the previous one VP) and ACL. They respectively control yaw and z values.

  Task's scope are:

  – for VH to make the angle between the distance from robot and rock and the robot x-axis go to zero;

  – for ACL to make the altitude ridden from the distance sensor going to 0.

  At the same time task VP is still active but operates only for (x,y), and it's goal is to compensate sea currents that can move the robot away from the target. In 1 is possible to see the Cartesian velocities generated and applied to the robot during this phase.

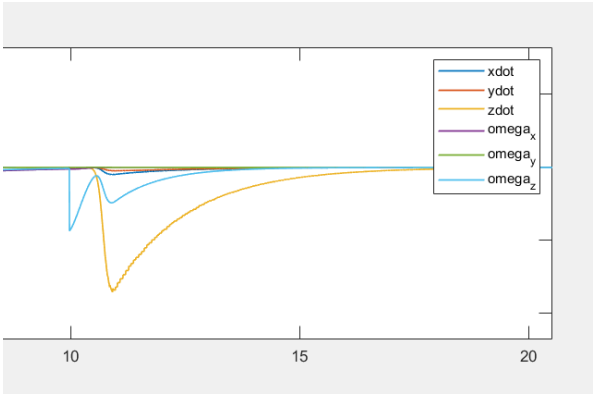

Figure 1: Cartesian velocities during phase 2

Note that zdot has a "stairs" behaviour. This is probably due to the fact that $\dot{x}_{ACL}$ depends on the distance sensor's value, that reads value at a certain frequence. The omega z behaviour will be explained in Q4.

- **Phase 3** In this last phase, robot has to analyze the nodule. For doing this, arm's tool should be positioned in rock position. This is done by task RN, which gives velocities to the arm's joint, which, without going outside the joint limit (safety task JL), extends and reaches the goal. Looking at 2 it's possible to see the joints' movements. In this task, robot is maintained horizontal (HA still active), and in the same position and orientation (VP and VH).

After these 3 phases, robot stops.

1.3 Q3: What is the Jacobian relationship for the Alignment to Target control task? How was the task reference computed?

Before of explaining the Jacobian relationship, I think is better to explain how the task reference is computed. For computing task reference, has been calculated the distance **d** between the robot's frame origin, and the rock origin. Then, in order to align, has been made explicit the angle between the vector obtained and **i** vector (x-axis) of the underwater robot, using the following formula:

$$\theta = atan2(\frac{dx}{||\mathbf{d}||}, \frac{dy}{||\mathbf{d}||}) + atan2({}^{w}i_{vx}, {}^{w}i_{vy}),$$

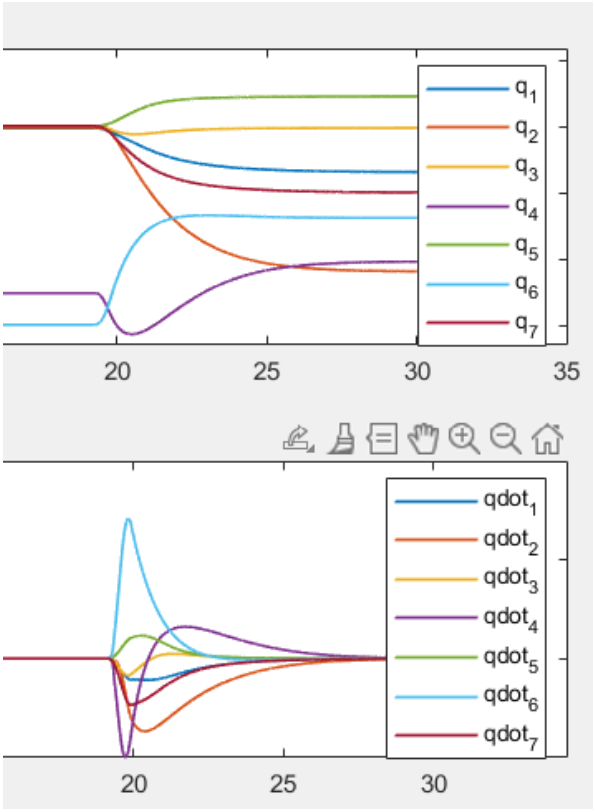

Figure 2: Joints' positions and velocities

$$\text{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}), & \text{se } x > 0, \\ \arctan(\frac{y}{x}) + \pi, & \text{se } x < 0 \wedge y \geq 0, \\ \arctan(\frac{y}{x}) - \pi, & \text{se } x < 0 \wedge y < 0, \\ +\frac{\pi}{2}, & \text{se } x = 0 \wedge y > 0, \\ -\frac{\pi}{2}, & \text{se } x = 0 \wedge y < 0, \\ \text{non definita}, & \text{se } x = 0 \wedge y = 0. \end{cases}$$

where atan2 is defined as following

Then $\dot{x}$ can be computed as the error between the desired angle (0°) and the real one (θ):

$$\dot{x} = \gamma(0 - \theta)$$

So, $\dot{x}$ is a scalar which represent an angular velocity around the z-axis, and, because of this, the Jacobian is a row vector, where each element is 0 and the last is 1.

1.4 Q4: Try changing the gain of the alignment task. Try three different values: 0.001, 0.1, 1.0. What is the observed behaviour? Plot the misalignment error and the altitude error over time, then compare and comment them. Implement a solution that, regardless of the gains chosen for the tasks and their initial conditions, guarantees that the landing is accomplished aligned to the target.

First of all, it's important to say that, how it can be seen in the following graph, altitude control is independent from align task, so because of this, the graph is always the same for every gain value.

For the misalignment error, before analizing it with different gain values, it's important to say that, in our implementation, has been made the control for $\dot{x}_{VH}$, and if it is under a certain threshold $\dot{x}_{VH} < 0.01 rad/s$ and $\theta > 5$, then $\dot{x}_{VH} = sign(\dot{x}_{VH}) \times 0.2 rad/s$. This is to make sure that it aligns

even if it has a low vehicle heading task velocity computed by default (so this answers the question). Summing everything up, how it is possible to see in below graph, the gain's value in this test did not influenced the behaviour of the robot, showing very similar error characteristics.

- $\gamma = 0.001$

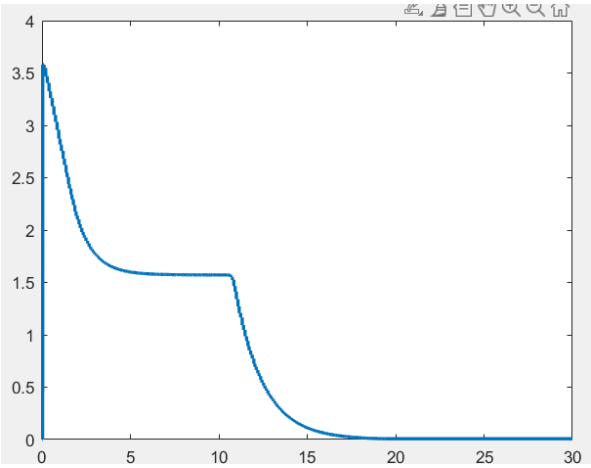

Figure 3: altitude error with gain 0.001

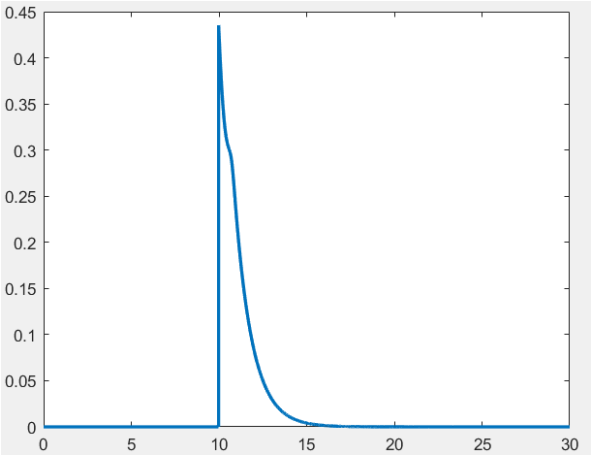

Figure 4: Misalignment error with gain 0.001

- $\gamma = 0.1$

- $\gamma = 1$

1.5 Q5: After the landing is accomplished, what happens if you try to move the end-effector? Is the distance to the nodule sufficient to reach it with the end-effector? Plot the distance of the end-effector to the goal, and the velocities of the vehicle and the manipulator. Comment the observed behaviour.

If you try moving the end effector, it moves, because now it is in the Fixed Based Manipulation action, and reaches the nodule in order to analyze it. This is because, in this case, the goal position given is

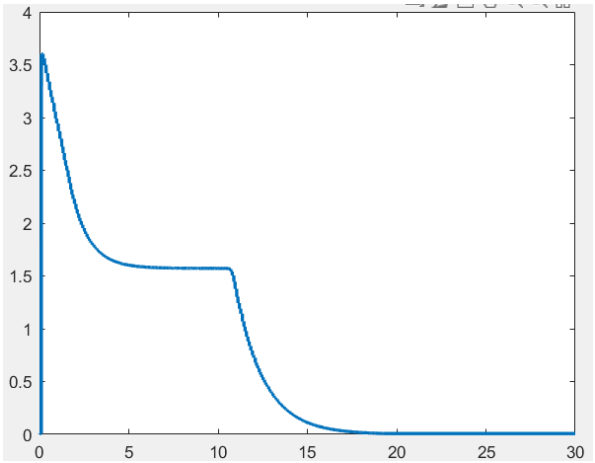

Figure 5: altitude error with gain 0.1

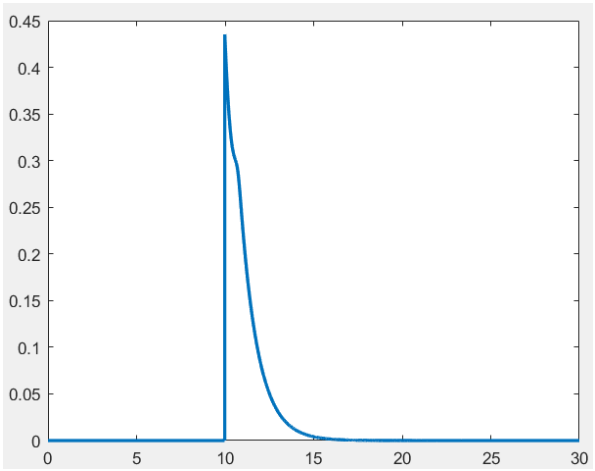

Figure 6: misalignment error with gain 0.1

close enough to the nodule.

How it's possible to see in the picture above, the EE reaches the nodule correctly, making the distance converge to zero.

In figure 10 there are the six Cartesian velocities of the End Effector: the angular velocity are null, in fact it doesn't rotate, while the linear component of Cartesian velocity start with an high value, and then converge to zero (this is due to the fact that task velocity depends on the Cartesian error between EE and nodule).

1.6 Q6: Considering that the goal position for the vehicle could be not so precise with respect to the position of the nodule, implement a solution that guarantees that, once landed, the nodule is certainly within the manipulator's workspace.

A solution for the distance problem between the tool and the rock, is to put, in the Landing action, an inequality constraint task (with more priority than every other task), that moves the robot closest to the target, through the direction of $\mathbf{d}$ calculated in Q3, at a distance at least as the maximum arm length.

The $\dot{x}$ of this task would be $\dot{x} = \gamma(\frac{\text{MAX ARM LENGTH}}{||\mathbf{d}||}\mathbf{d} - \mathbf{d})$, the Jacobian would have been the same of VP task (the one of phase 2, so without the z component). About the activation function, it would have been an Increasing bell shape function that actives when the robot is outside the circle

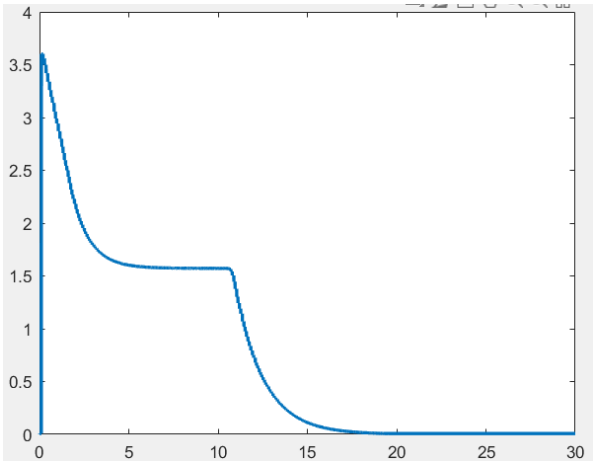

Figure 7: altitude error with gain 1

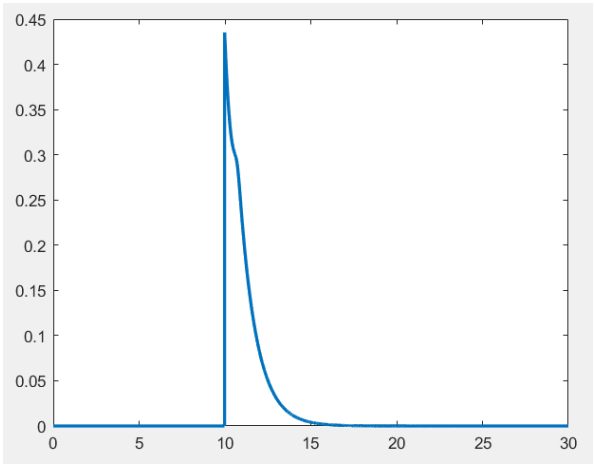

Figure 8: misalignment error with gain 1

with radius equal to MAX ARM LENGTH.

By implementing this task, the case in which the goal position is too far from the nodule, the robot would be pushed anyway closer.

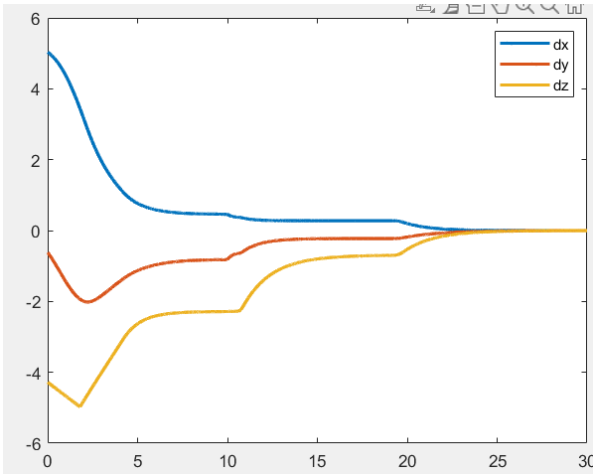

Figure 9: The three components of the distance between nodule and end effector $\mathbf{d_t ool} = (dt_x, dt_y, dt_z)$

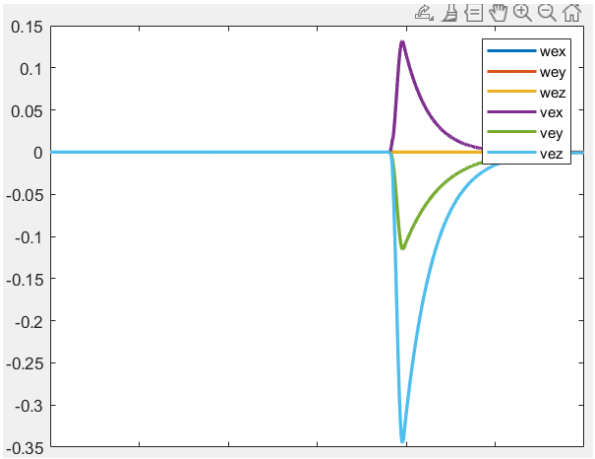

Figure 10: The six Cartesian velocities of the End Effector

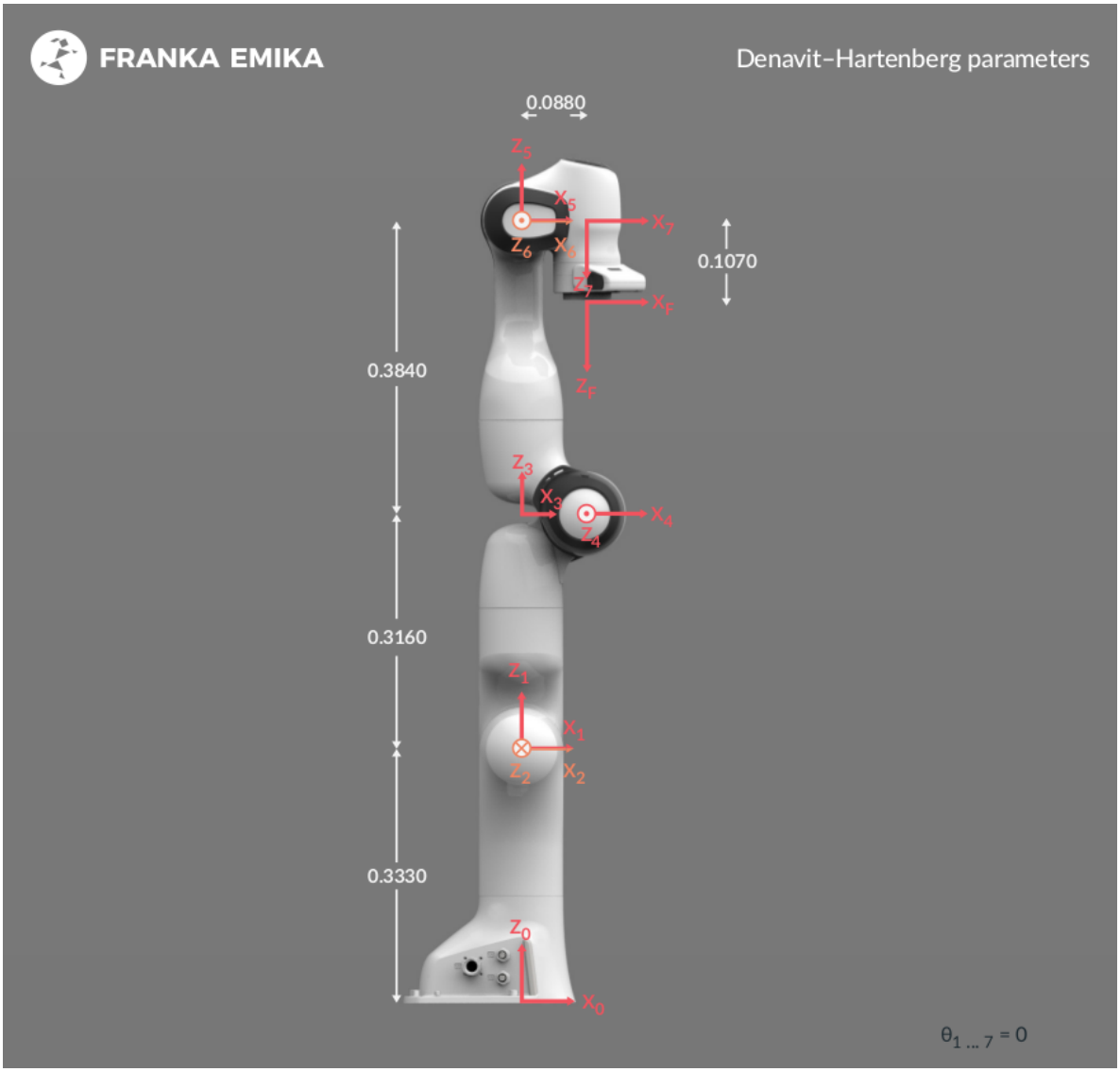

Figure 11: Robot Specifications

2 Exercise 2: Bimanual manipulation

In this exercise it is required to perform a bimanual manipulation by implementing the task priority algorithm considering two manipulators as a single robot. The manipulator adopted for this assignment is the Franka Panda by Emika (see Figure 11)). A simulation in python is provided, in order to visualize the two robots and test your Matlab implementation.

1. The transformations between the world and the base of each robot must be computed knowing that:
   (a) The left arm base coincides with the world frame.
   (b) The right arm base is rotated of π w.r.t. z-axis and is positioned 1.06 m along the x-axis and -0.01 m along the y-axis.
2. The transformation from each base to the respective end-effector is given in the code and is retrieved from the URDF model thanks to the *Robotic System Toolbox* of Matlab.
3. Then, define the tool frame for both manipulators; the tool frames must be placed at 21.04 cm along the z-axis of the end-effector frame and rotated with an angle of -44.9949 deg around the z-axis of the end-effector.

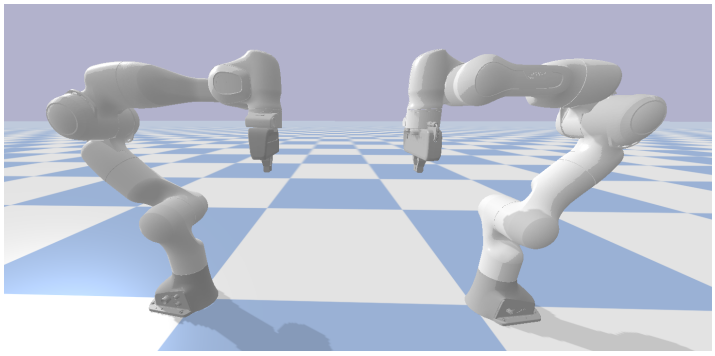

Figure 12: Initial position of the robots

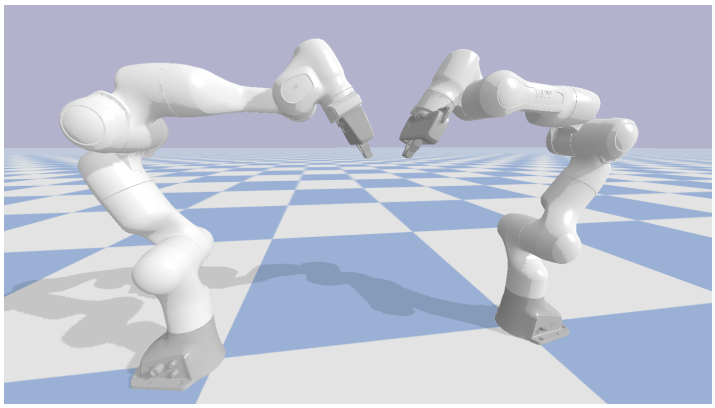

Figure 13: Relative orientation at the grasping position

4. Implement a "move to" action that includes the joint limits task.

5. The first phase foresees to move the tool frame of both manipulators to the grasping points, by implementing a "move to" action and its corresponding tasks. Define the goal frames for each manipulator such that their origin corresponds to the grasping points. **HINT**: the position of the grasping points can be computed by knowing the origin of the object frame ${}^{w}\mathrm{O}_{o}$ and the object length l_{obj}. The goal orientation of the tool frames is obtained by rotating the tool frames 30 deg around their y-axis; the manipulators should reach the configuration depicted in Figure 13.

$$ {}^{w}\mathrm{O}_{o} = [0.5, 0, 0.59]; \tag{1} $$

$$ l_{obj} = 10\,(\mathrm{cm}). \tag{2} $$

6. During this phase of the mission, the following safety tasks must be implemented: *end-effector minimum altitude* and *joint limits*. The joint limits specifications can be found in the datasheet of the robot. The minimum altitude from the table should be 15 cm.

Once the manipulators reach the grasping points, the second phase of the mission should start. Now, implement the Bimanual Rigid Grasping Constraint task to carry the object as a rigid body.

1. Define the object frame as a rigid body attached to the tool frame of each manipulator. **HINT:** Compute this quantity after reaching the grasping point.

2. Define the rigid grasp task.

3. Then, move the object to another position while both manipulators hold it firmly, e.g. ${}^{w}\mathrm{O}_{g} = [0.65, -0.35, 0.28]^{\top}\,(m)$

4. Note that the transition for the *Bimanual Rigid Constraint* should be a binary one, i.e., without smoothness. This is the nature of the constraint, i.e., either it exists or not. Modify the *Action-Transition* script seen during the class to take into account the different nature of this task (a constraint one).

Once the object has been moved to the required position you have to implement a final phase of the mission in which the joint velocities are set to zero, and every action is deactivated except for the minimum altitude task.

Repeat the simulation, using a goal position or by changing the grasping in such a way that one of the two manipulators activates multiple safety tasks, to stress the constraint task.

2.1 Q1: Report the unified hierarchy of tasks used and their priorities in each action.

In this assignment we have implemented three main actions:

1. Go to (go_to)
2. Cooperative manipulation (coop_manip)
3. End motions (end_motions)

Each action is composed by several tasks, following the previous action's order we will elencate them:

1. Joints Limit (JL), Minimum Altitude (MA), Move Tool Task(T)
2. Joints Limit (JL), Minimum Altitude (MA), Rigid constraint (RC), Move Tool Task (T)
3. Minimum Altitude (MA), Move Tool Task (T)

Inside the action the sequences of ICAT_task calls defines the priority.
In the action one (**go_to**) the priorities we decided to adopt are:

1. Joints limit task (JL), (Constraint task, Inequality task)
2. Minimum altitude task (MA), (Safety task, Inequality task)
3. Move to tool task (T), (Action oriented task, Equality task)

In the second action (coop_manip)

1. Rigid constraint task (RC), (Constraint task, Equality task)
2. Joints limit task (JL), (Safety task, Inequality task)
3. Minimum altitude task (MA), (Safety task, Inequality task)
4. Move to tool task (T), (Action oriented task, Equality task)

In the last action (end_motions), only two tasks are active:

1. Minimum altitude task (MA), (Safety task, Inequality task)
2. Move to tool task (T), (Action oriented task, Equality task)

Task	Type	go_to	coop_manip	end_motions
Rigid Constraint (RC)	E		1	
Joints limit (JL)	I	1	2	
Minimum altitude (MA)	I	2	3	1
Move To Tool (T)	E	3	4	2

2.2 Q2: What is he Jacobian relationship for the Joint Limits task? How was the task reference computed?

Joint Limit Tasks

The Jacobian relationship for the joint limit tasks is an identity matrix of dimension 14 because both $\dot{x}$ and $\dot{y}$ are in the joint space. Therefore, working in this space eliminates the need for the Jacobian to map between Cartesian space and joint space.

Task References

The task references for both arms are computed as follows:

$$pandaArm.Arm.xdot.jl = gain\_jl \cdot \left(\frac{pandaArm.jlmax - pandaArm.jlmin}{2} - pandaArm.ArmL.q \right);$$

Here, the error is calculated by subtracting the middle position (the average between the maximum and minimum allowed joint positions) from the actual joint positions.

Activation Function

For the activation function, we used a diagonal matrix $\mathbf{A} \in \mathbb{R}^{14\times14}$, where the diagonal elements are the sum of two sigmoids—one for the maximum limit and one for the minimum limit.

2.3 Q3: What is the Jacobian relationship for the Bimanual Rigid Grasping Constraint task? How was the task reference computed?

In the constraint task we have to fix the distance between the two tools. The jacobian matrix we use in this task is wJo , so we consider the object as a rigid object attached to the tool frame.
From theory the condition to be resepcted is:

$$pandaArm.xdot.rc = pandaArm.ArmL.wJo * ydotbar = pandaArm.ArmL.wJo * ydotbar$$

SO in our case we want that pandaArm.xdot.rc = 0, because we want they don't move but maneain same distance between them.
On software point of view we implement the jacobian in this way:

$$pandaArm.Jrc = [pandaArm.ArmL.wJo, -pandaArm.ArmR.wJo];$$

pandaArm.Jrc $\in \mathbb{R}^{6\times7}$

So using this jacobian for the previous equations from theory we assign the task references = 0

$$pandaArm.xdot.rc = zeros(6, 1);$$

2.4 Q4: Comment the behaviour of the robots, using relevant plots. In particular, show the difference (if any) between the desired object velocity, and the velocities of the two end-effectors in the two cases. Add plots to show that the bimanual manipulation unfolds effectively (e.g., the distance between the tool frames is constant to show that the kinematic constraint is satisfied)

The most important configuration of our robot is when it arrives in the two different main point: The grasping point and the final points. So now we will link the two images of the two different configurations fig 14, 15. The two point into images are represented by teo red points.

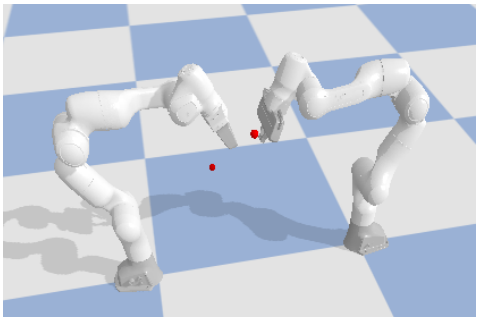

Figure 14: Grasping point

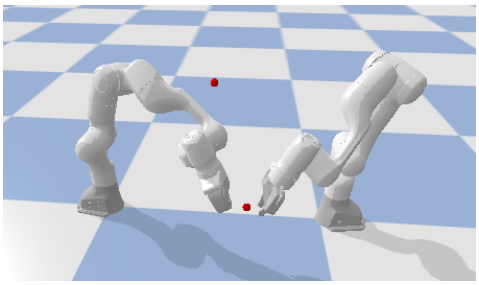

Figure 15: Final goal

The relative speed of the joint of the two robots are showed in figure 16, 17.

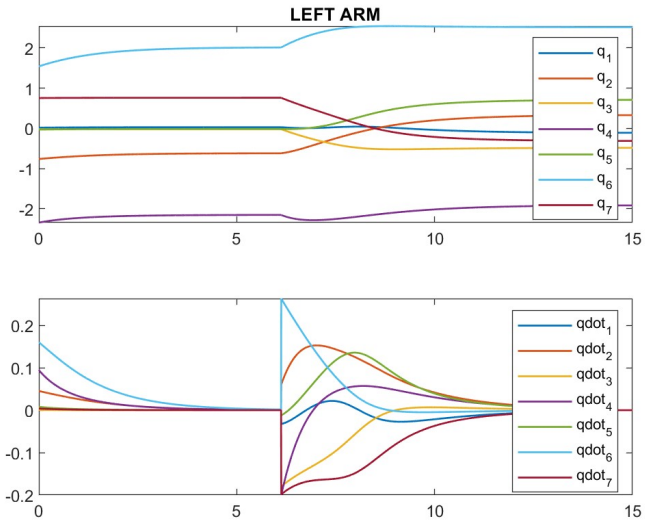

Figure 16: q, q_dot left arm

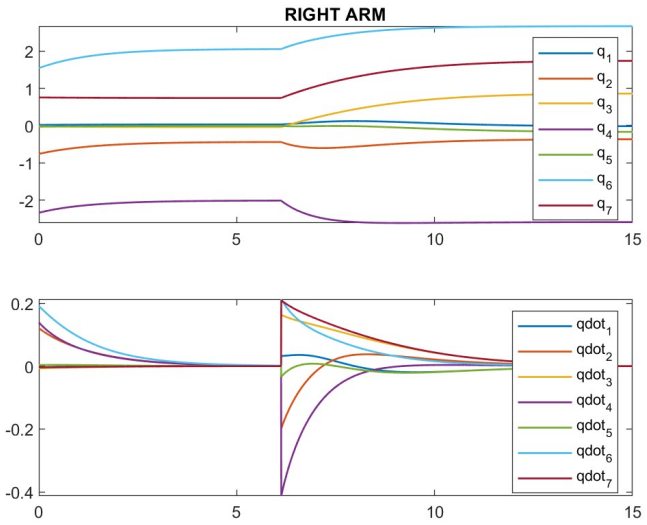

Figure 17: q, q_dot right arm

In this this two images is possible to distinguish the two different phasis of the motion. So in

correspondence with the peack there is a change of phase. A possible optimization could be to use a strategies for attenuate the peak on joint velocities, this peak is due to the fact we change the velocity reference at the task move_tool.

During the second phase, the cooperative phase, the rigid constraint is active. Therefore, it is important to analyze the relative position of the two tools throughout the movement, especially during this phase. To evaluate whether the rigid constraint works as intended, we will plot the Cartesian errors during the entire movement by comparing the transformation matrices of the two tools.

Thus, we will plot the Cartesian error of Arm Left wTt and Arm Right wTt.

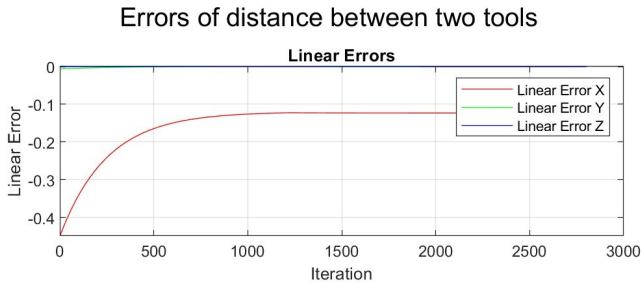

Figure 18: Cartesian error between the tools during the movement. (for test rigid constraint)

As shown in Figure 18, focusing on the linear error, we can clearly observe the transition between mission phases. In the first part, the two tools move relative to each other, and around iteration 1500, the distance becomes constant at 0.12 meters along the x-axis (red line in the figure). This confirms that the rigid constraint is correctly enforced.

Another important aspect is to compare the actual object velocity with the desired velocity of the two end-effectors. The object's Cartesian velocity is shown in Fig. 19, while the desired tool velocity is presented in Fig. 20.

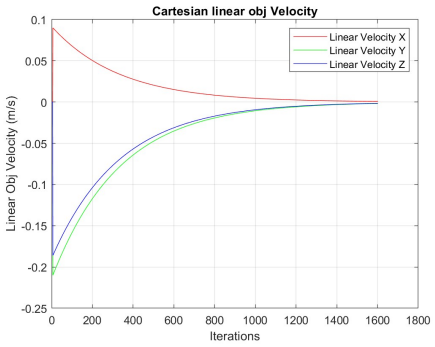

Figure 19: Object Cartesian velocity

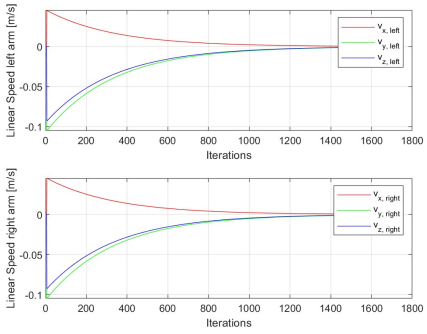

Figure 20: Tool desired velocities

As we can see, all three images are very similar; they have the same shape, and the magnitude is also very similar. The very small difference in magnitude could be attributed to the constraint task, as it is the only active task during this action.

Another important test would be to evaluate the safety tasks, starting with the joint limits. To test this safety task, I set $\dot{y}_{\text{bar}}$ to zero and apply a constant input to the first joint of the robot on the right side. At this point, the joint starts moving until it reaches the maximum limit, and then the activation function gradually starts activating until it reaches 1. The activation of the action Transition is on figure 21

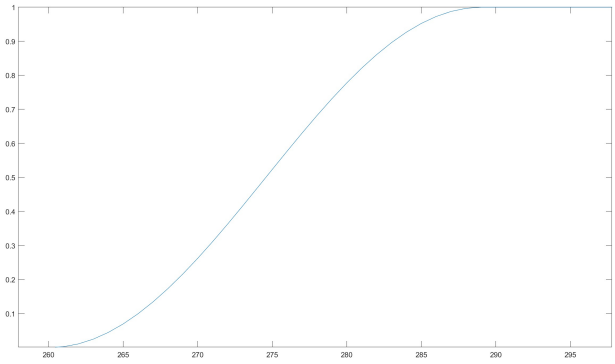

Figure 21: First Joint left Arm joint limit activation function

After testing the joint limit task, We also verified that the minimum altitude safety task worked correctly. The goal of phase 2 was set to a point below the activation threshold of the minimum altitude task. The chosen goal point was $[0.65, -0.35, -20]'$.

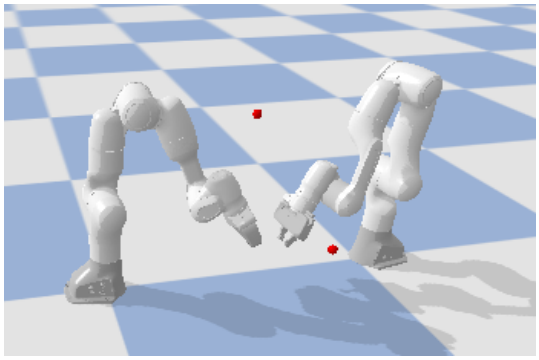

Figure 22: Min Altitude simulation

In this case, we can observe that the two arms never touch the ground and remain within the safety threshold set using the δ parameter in the activation function. In Fig. 22, the two robots are shown in the environment, while Fig. 24 presents the activation of the activation function. Additionally, in Fig. 23, the values of $\dot{q}$ for the left arm are displayed, clearly showing that the velocities approach zero as the robot nears the ground. This behavior occurs because the minimum altitude task, with a correctly defined reference, prevents further motion in that direction.

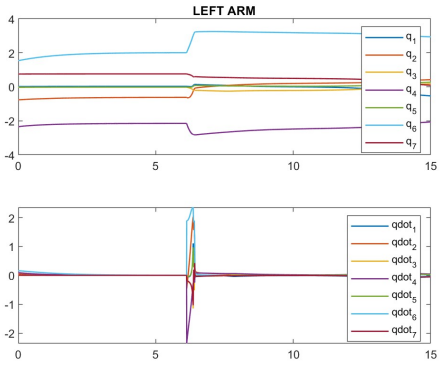

Figure 23: q, q_dot with min alt task active

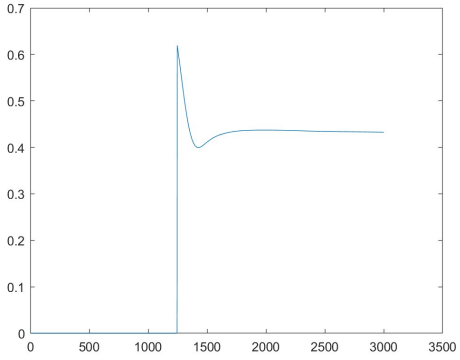

Figure 24: Minimum altitude activation function

3 Exercise 3: Cooperative manipulation

In this exercise, it is required to perform cooperative manipulation by implementing the task priority algorithm considering the two Franka Panda manipulators as two distinct robots.

1. The first phase foresees to move the tool frames to the grasping points, by implementing the "move to" action for both manipulators. **Please note: each manipulator has his own Task Priority Inverse Kinematic Algorithm**. Define the goal frames for each manipulator such that their origin corresponds to the grasping points. **HINT**: the position of the grasping points can be computed by knowing the origin of the object frame ${}^{w}\mathrm{O}_{o}$ and the object length l_{obj}. The goal orientation of the tool frames is obtained by rotating the tool frames 20 deg around their y-axis.

$$ {}^{w}\mathrm{O}_{o} = [0.5, 0, 0.59]^{\top}(m); \tag{3} $$

$$ l_{obj} = 6\,(\mathrm{cm}). \tag{4} $$

   During this phase of the mission, the following safety tasks must be implemented: *end-effector minimum altitude* and *joint limits*. The joint limits specifications can be found in the datasheet of the robot. The minimum altitude from the table should be 15 cm.

2. Once the manipulators reach the grasping points the second phase of the mission should start. Implement the *Cooperative Rigid Constraint* task to carry the object as a rigid body.

   (a) Define the object frame as a rigid body attached to the tool frame of each manipulator.

   (b) Define the rigid grasp task.

   (c) You have to move the object to another position while both manipulators hold it firmly. The desired object goal position is

$$ {}^{w}\mathrm{O}_{g} = [0.60, 0.40, 0.48]^{\top}(m) \tag{5} $$

   (d) Compute the *non-cooperative* object frame velocities. **HINT:** Suppose manipulators communicate ideally and can exchange the respective end-effector velocities

   (e) Apply the coordination policy to the *non-cooperative* object frame velocities.

   (f) Compute the *cooperative* object frame velocities.

   Note that the transition for the *Cooperative Rigid Constraint* should be a binary one, i.e., without smoothness. This is the nature of the constraint, i.e., either it exists or not. Modify the *ActionTransition* script seen during the class to take into account the different nature of this task (a constraint one).

3. Once the object has been moved to the required position you have to implement a final phase of the mission in which the joint velocities are set to zero, and every action is deactivated except for the minimum altitude task.

Again, test it twice, once with the provided (reachable) goal, and then with a goal or with a different grasp configuration that triggers the activation of multiple joint limits or a joint limit and minimum altitude by one manipulator. The idea is that this second position should trigger the cooperation policy (i.e., the cooperative velocity should be different than the original desired object velocity).

3.1 Q1: Report the unified hierarchy of tasks used and their priorities in each action, and report clearly the actions used in the two phases of the cooperation algorithm.

Task	Type	Go To $\mathcal{A}_1$	CoopMan $\mathcal{A}_2$	$->$ CoopMan $\mathcal{A}_2$
Joint Limits	I	1	1	2
Minimum altitude	I	2	2	3
Move Tool	E	3	3	
"Cooperative reference"	E			1

Table 3: Table to show the relation between the tasks (columns) and the actions of each specific task (rows) showing the priorities

In this scenario, there are two agents that share the same goal: reaching an object and collaboratively moving it to a target point.
During the *go to object* task, the two agents operate independently. Each computes the joint limit safety task to ensure feasible joint velocities while also considering the minimum altitude safety task to avoid collisions with the base. Once these constraints are addressed, the motion required to reach the object's position can be calculated.

Conversely, when the two agents need to cooperate to follow a shared policy (implemented in the *main.m* file of the code), they engage in a fast exchange of information. This allows each agent to compute the best feasible motion for moving the object toward the new goal position and then share this information with the other agent.
So, becomes possible

1. Each agent computes the desired velocity to move the object following the priority hierarchy (this case: joint limits, minimum altitude and tool constraint)

2. Each agent evaluates the Cartesian velocity to impose to the abject through the tool.
   So the Non cooperative velocity is : $\dot{x} = J_{t,i} \cdot \dot{y}$

3. Compute the cooperative tool frame velocity : $\dot{x}_{\hat{t}} = \frac{1}{\mu_a + \mu_b} \cdot (\mu_a \cdot \dot{x}_{t,a} + \mu_b \cdot \dot{x}_{t,b})$

   The $\mu_i = \mu_0 + ||\bar{\dot{x}}_{t,i} - \dot{x}_{t,i}||^2$, where μ_0 corresponds to a small value (our case = 0.2) to avoid irregularities and always accomplish the motion.
   μ_i is a parameter to allow a compromise between the agents to improve the performance of the robot with the biggest error to the detriment of the other one

4. Compute the motion space single agent $H_i = J_{t,i} * pseudoinverse(J_{t,i})$ for i =a,b

5. Compute the Cartesian constraint matrix: C = $[H_a - H_b]$

6. Compute the Cartesian feasible velocities, for both agents (if a good cooperation they should be the same)
   Cartesian feasible velocities:

$$\begin{bmatrix} \tilde{\dot{x}}_t \\ \tilde{\dot{x}}_t \end{bmatrix} = \begin{bmatrix} H_a & 0_{6x6} \\ 0_{6x6} & H_b \end{bmatrix} * (I_{12x12} - pseudoinverse(C) * C) \begin{bmatrix} \hat{\dot{x}}_t \\ \hat{\dot{x}}_t \end{bmatrix}$$

7. Each agent starts the computation of the new *TPIK* from the feasible cooperative velocities and then considering the other task of the mission phase to perform the goal

We want to highlights that haven't considered the internal wrenches for the object manipulation.

3.2 Q2: Comment the behaviour of the robots, using relevant plots. In particular, make sure there are differences between the desired object velocity and the non-cooperative Cartesian velocities at least in one simulation. Show also how the cooperative velocities of the two end-effectors behave. Add plots to show that the cooperation unfolds effectively (e.g., the distance between the tool frames is constant to show that the kinematic constraint is satisfied)

In this first two plots we are showing the desired object velocity (in blue), computed as follow:

$$\dot{x}_d = gain \cdot e$$

where e is the error, angular and linear, between the goal position and orientation and the actual position and orientation of the object (that in this second phase is rigidly attached to the tool of the manipulators), while the gain represents a proportionality factor that regulates the intensity of the system's response to the error.
Then to the same plot we add the actual xdot, that are computed in this way:

$$\dot{x}_a = wJo \cdot ydotbar$$

so, by computing the product between the jacobian matrix (wJo) and the vector of joints velocities ($ydotbar$), obtained by applying the iCAT_task, that computes the solution which satisfy at best, in a minimum norm fashion, the need of tracking $\dot{x}_d$.
To plot this two velocities together can be interesting since we know that they can differ. Infact the key mechanism in our task priority algorithm is exploiting any residual arbitrariness that is allowed by the manifold of solutions: if the projector in the kernel has a rank greater than 0, it spans a subspace of $\mathbb{R}^n$ which can be further exploited by subsequent, lower priority, tasks.
In other terms, the lower priorities problems needs to minimize "what remains to be done" after a part of the solution has been set by the higher priorities tasks.
Furthermore we want to activate and deactivate the task smoothly, and it is done thanks to the activation function that can activate or deactivate some rows of the jacobian matrix without discontinuity. For better visualization of the problem, we decided to separate the linear velocities and angular velocities into two distinct plots.

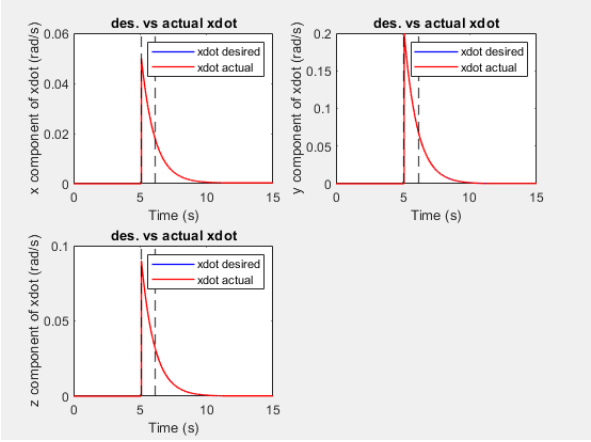

Figure 25: Desired vs Actual Object Angular Velocity, LEFT ARM

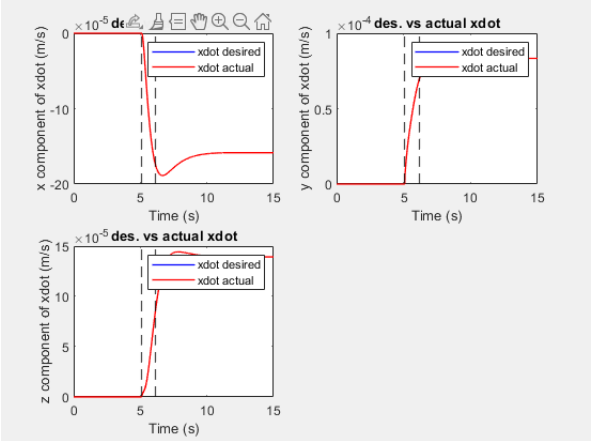

Figure 26: Desired vs Actual Object Linear Velocity, LEFT ARM

Afterward we illustrate the joint position and velocity of the two arms.
The *ydotbar* are now assigned to the *qdot*
We don't see here the "action" of the activation functions that, usually, make the *qdot* increasing or decreasing gently by activating or deactivating some rows of the jacobian matrix without discontinuity, because the task, that makes the tool moving, is always activated during the three phases. To improve the behaviour of the *qdot* we could only try to act on the $\dot{x}_d$.
The q, instead, after an initial transient, reach and stabilize around the desired angle.

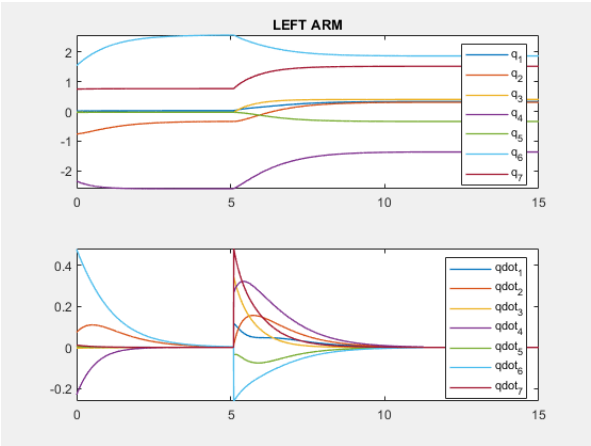

Figure 27: Joint position and velocity Left Arm

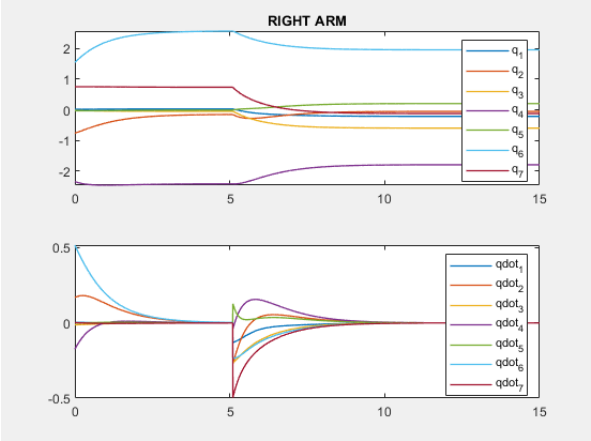

Figure 28: Joint position and velocity Right Arm

Now we can look to the linear distance between the tools of the agents
Obviously at the beginning is very big, until they both reach the object, the first segmented line in the figure 30. From now on, so during the cooperation, the tools' distance is constant and equal to the object length. This proves a correct implementation of the software control.

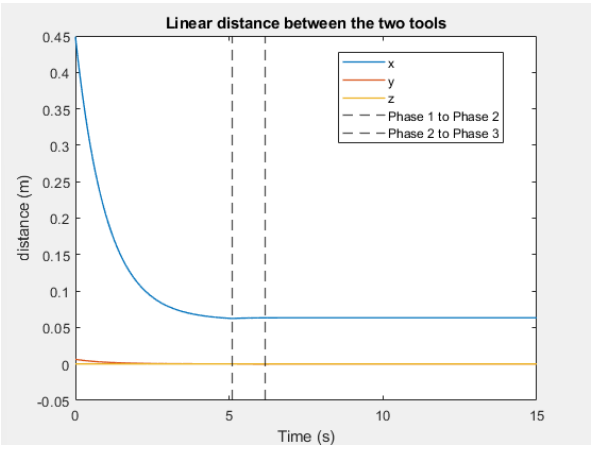

Figure 29: Linear distance between tool of arm1 and tool of arm2

Finally, we can see the cooperative velocities
It is evident from the plots that at the two manipulator are assigned the same velocities.

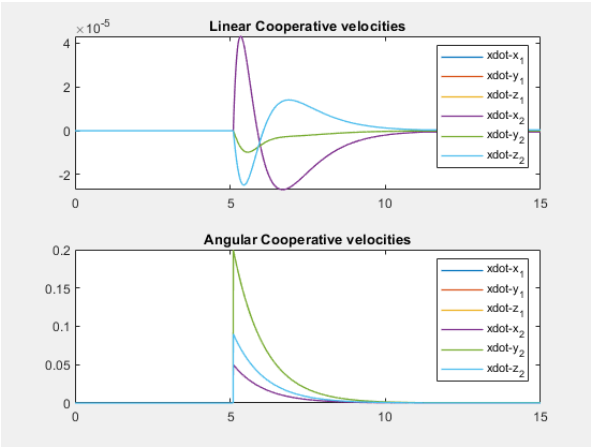

Figure 30: Cooperative velocities of the two end-effectors