Cooperative Robotics

Authors: Annika Delucchi, Valentina Condorelli, Ramona Ferrari, Daniele Rialdi
Student id: 4975984, 4945679, 4944096, 4964038

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 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.

Use the following template when you need to discuss the hierarchy of tasks of a given action or set of actions:

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 A	I	1		1
Task B	I	2	1	
Task 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.

To perform the mission, the vehicle must take into account many concurrent objectives. The hierarchy of tasks used in this exercise is the following:

Task	Type	Safe Navigation	Vehicle Aligning	Landing	Grasping
MA	I, S	1	1		
HA	I, OP	2		1	
VP	E, AD	3	3	3	
VO	E, AD	4	4	4	
LAN	E, AD			2	
HAL	I, OP		2		
G	E, AD				2
ZVC	E, C				1

Table 2: Hierarchy of tasks of UVMS exercise.

In all the actions, we give higher priority to safety and constraint tasks. This is done to make sure that the vehicle successfully reaches the waypoint without any faults or damages, and that the arm can then grab the rock safely.
The description of each task listed in Table 2 is reported below:

- **Minimum Altitude**: the vehicle must stay in a safe region, with a distance from the seafloor $>$ 1m, to avoid damages;
- **Horizontal Attitude**: the vehicle horizontal attitude should be maintained within reasonable bounds, to avoid situations where the vehicle is upside-down. "*Reasonable*" means that the boundaries should be big enough to avoid excessive energy consumption to always keep the vehicle perfectly horizontal but, at the same time, small enough to assure the effective operation of the task;
- **Vehicle Position**: the vehicle frame should be aligned correctly with the goal frame, in terms of position;
- **Vehicle Orientation**: the vehicle frame should be aligned correctly with the goal frame, in terms of orientation;
- **Landing**: once aligned with the waypoint, so the goal frame, the vehicle should land, by lowering its position until a 0m altitude is reached;
- **Horizontal Alignment**: the longitudinal axis of the manipulator's end-effector should be aligned with the frame of the rock, which is the grasping goal. As a consequence, the vehicle rotates along its z-axis until the misalignment is reduced below the threshold;

- **Grasping**: end-effector/tool position control to have the tool frame converging to the tock frame, which is the goal;
- **Zero Velocity Constraint**: the vehicle must maintain its position, so keep its velocity null.

It is important to specify that all **equality tasks** are controlled in terms of small thresholds, in order to have a faster simulation.

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

For all the plots reported below, the simulation time is set to 22s, in order to show the stability of the system.

The first relevant plot to discuss, shown in Figure 1, illustrates the trend of the **Activation Functions** for all the tasks.

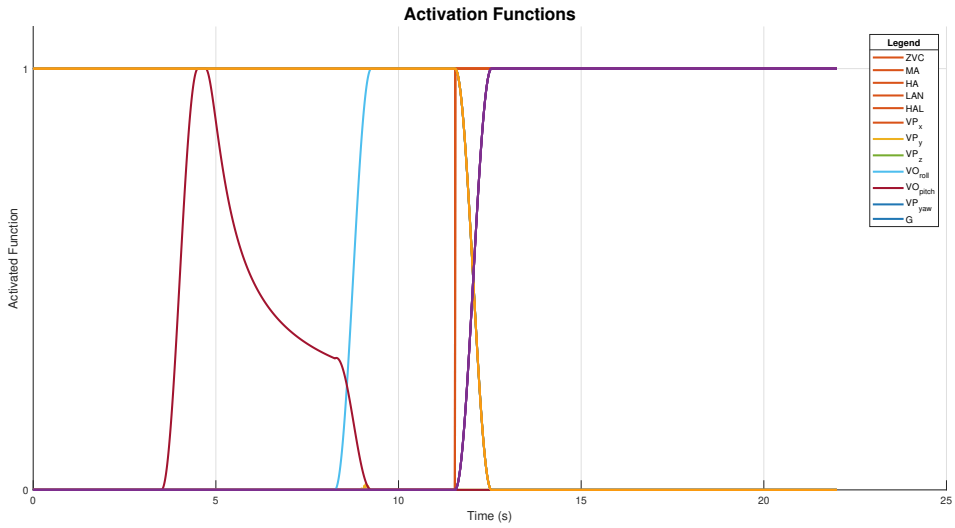

Figure 1: Activation Functions of UVMS tasks

As expected, when each task is activated or deactivated, its behaviour is smooth, as a consequence of the bell function added in the action transition. This is necessary to avoid discontinuity in the vehicle behaviour. The only exception are constraint tasks, which must have an immediate transition due to its binary existing nature. In this case, the only constraint task is the Zero Velocity Constraint one.

Another useful plot is about the **joint positions** and the **joint velocities** of the manipulator, during the *Grasping* action, as shown in Figure 2.
As expected, both joints positions and velocities keep their initial value until the *Grasping* action is reached. Then, the joints positions change accordingly to the wanted final configuration, in which the manipulator has grabbed the rock.
In order to reach it, reference velocities are passed to the joints, with the rock frame as the goal. However, as a safety measure in place of the joint limits task, these velocities are saturated to a limit of 0.2 (m/s or rad/s). Therefore, the oscillatory behaviour shown in Figure 2 for the joints velocities is expected, as a result of the velocity saturation.

The last presented plot, shown in Figure 3, reports the **vehicle positions** and the **vehicle velocities**.

Complementary to what was discussed for the manipulator, the vehicle moves for about 12s, after which it keeps the same landing position and its linear and angular velocities converges to 0. This behaviour shows the correct operation of the **ZVC** task.

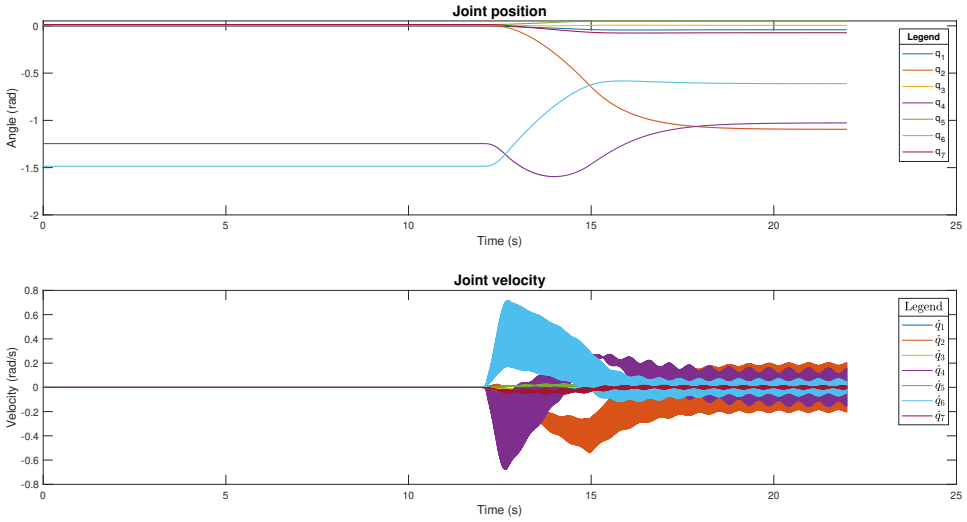

Figure 2: Joint position and velocities of UVMS

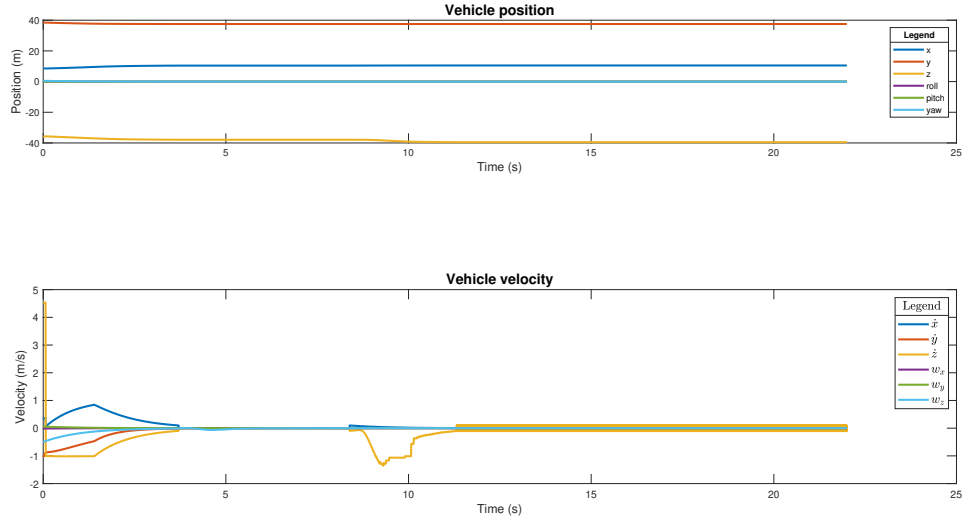

Figure 3: Vehicle position and velocities of UVMS

The comparison between Figure 2 and Figure 3 confirms what was expected: for the first 12s, the vehicle executes the first three actions, thus safely navigating towards the goal, aligning with the rock frame and landing. In the meanwhile, the manipulator keeps its position.
On the other hand, in the remaining 10s of simulation, the vehicle keeps its position, while the manipulator moves in order for the end-effector to reach the rock frame, thus achieving the grasping action. In the last 3-4s of simulation, it is evident how the whole system has reached its stability, has all the action have been successfully performed.

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

Let's assume to have the origin of the vehicle frame O_v and the target origin P, it is possible to compute the distance vector $\mathbf{d} \triangleq P - O_v$ and the misalignment vector ρ:

$$\rho = \theta \mathbf{n} = \frac{\theta}{sin\theta} \mathbf{i}_v \wedge \mathbf{n}_d. \tag{1}$$

where $\mathbf{n}_d \triangleq \frac{\mathbf{d}}{||\mathbf{d}||}$. The misalignment is computed between $\mathbf{i}_v$ (x-axis of the vehicle) and $\mathbf{i}_v$ (the unit vector of the distance vector).
The alignment of the vehicle with the target is ensured by ρ equal to 0, for example, requiring the derivative to be proportional to the misalignment:

$$\dot{\rho} = -\lambda \rho = -\lambda \theta \mathbf{n}. \tag{2}$$

If true derivative $D\rho$ follows the wanted value, $\rho \to 0$. We should find the relationship between ρ and $\dot{\mathbf{y}}$ and the result is:

$$\mathbf{w}_{n_d/i_v} = \mathbf{w}_{n_d/w} - \mathbf{w}_{i_v/w}. \tag{3}$$

The tip of the vehicle moves with this velocity $\mathbf{v}_{d/w} = \mathbf{w}_{d/w} \wedge \mathbf{d}$. Next, for $\mathbf{w}_{n_d/w}$:

$$\mathbf{w}_{n_d/w} \triangleq \mathbf{w}_{d/w} = \frac{1}{||\mathbf{d}||^2} (\mathbf{d} \wedge \mathbf{v}_{d/w}). \tag{4}$$

and from definition $\mathbf{v}_{d/w} = \mathbf{v}_{P/w} - \mathbf{v}_{v/w}$. By assuming that the target, specifically the rock to which the vehicle must be aligned, remains fixed, the computation of the problem can be simplified and we end up with the final Jacobian relationship:

$$D_v \rho = (\mathbf{n}\mathbf{n}^T - \frac{\theta}{sin\theta}[a\wedge][b\wedge](I_{3\times 3} - \mathbf{n}\mathbf{n}^T)) \begin{bmatrix} \mathbf{0}_{3\times l} & \frac{-1}{||\mathbf{d}||^2}[\mathbf{d}\wedge] & -\mathbf{I}_{3\times 3} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{v}_{v/w} \\ \mathbf{w}_{v/w} \end{bmatrix} \triangleq \mathbf{J}_{at} \dot{\mathbf{y}}, \tag{5}$$

where $\mathbf{w}_{v/w}$ and $\mathbf{v}_{v/w}$ are our control variable. This expression is general, but we are interested in the derivative of θ, because we assume the target is immobile, thus $\mathbf{v}_{t/w}$:

$$\dot{\theta} = \mathbf{n}^T \begin{bmatrix} \mathbf{0}_{3\times l} & \frac{-1}{||\mathbf{d}||^2}[\mathbf{d}\wedge] & -\mathbf{I}_{3\times 3} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{v}_{v/w} \\ \mathbf{w}_{v/w} \end{bmatrix} \triangleq \mathbf{J}_{at} \dot{\mathbf{y}}. \tag{6}$$

The task reference is computed with respect to $\rho_{at/w}$; since when the two vectors are aligned (consequently, the vehicle is aligned to the target), the previous quantity tends to 0 value.
The computation is the following:

$$\dot{\bar{\mathbf{x}}}_{at/w} = \lambda(0 - ||^w \rho_{at}||). \tag{7}$$

λ is a strictly positive real value: in our case we set it to 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.

The value of the gain of the alignment task influences the speed at which the vehicle aligns with the goal.

In Figure 4, the whole behaviour of the robot was tested with a general gain of 1.0, used also for

the **Horizontal Alignment** task. On the other hand, in Figure 5, you can see the whole robot behaviour still with a general gain on 1.0, but with a specific gain for the **Horizontal Alignment** task of 0.1. As expected, in the latter test, the robot takes more time to execute the **Horizontal Alignment** task, due to the lower gain.
The same test was repeated with a specific gain for the **Horizontal Alignment** of 0.001, resulting in an even higher time required for the robot to complete the task (more than 15 min) and the plots are not interesting to describe the behaviour of the vehicle.

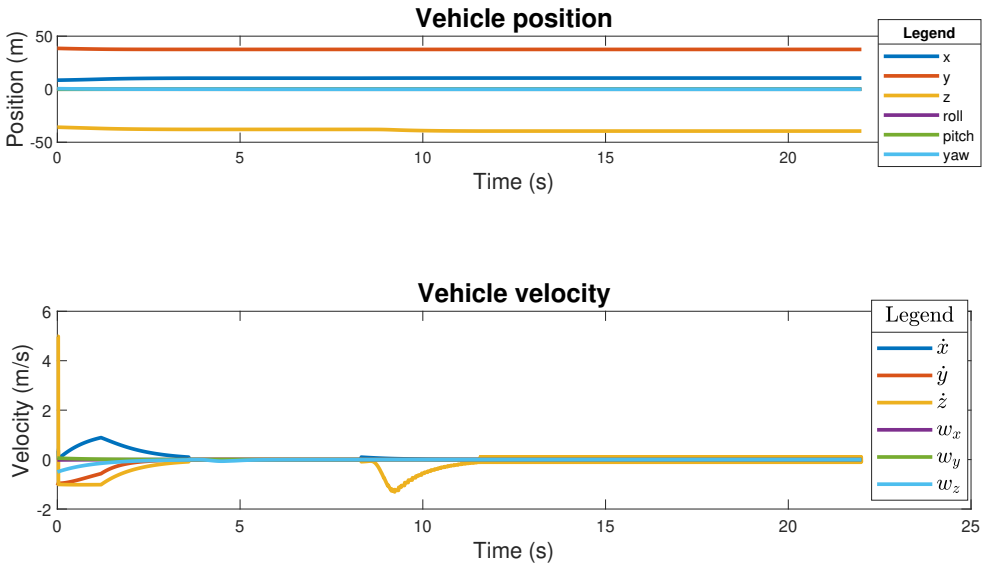

Figure 4: Vehicle position and velocity of UVMS with gain = 1

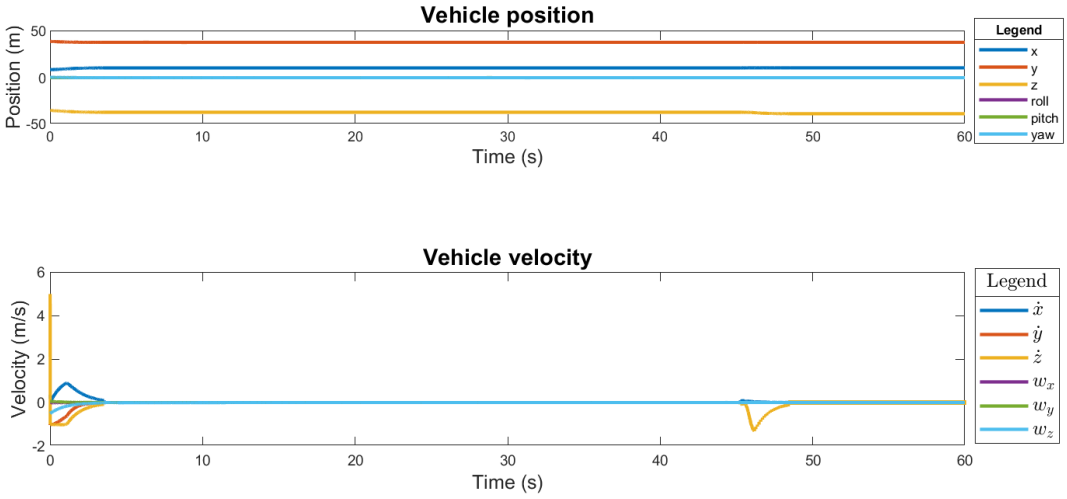

Figure 5: Vehicle position and velocity of UVMS with gain = 0.1

To further validate this thesis, the misalignment and altitude errors with the two tested gain can be plotted and compared.

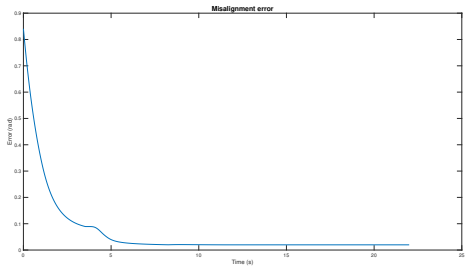

Figure 6: Misalingment error UVMS, with gain = 1.0.

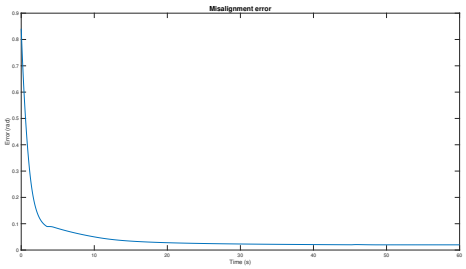

Figure 7: Misalingment error UVMS, with gain = 0.1.

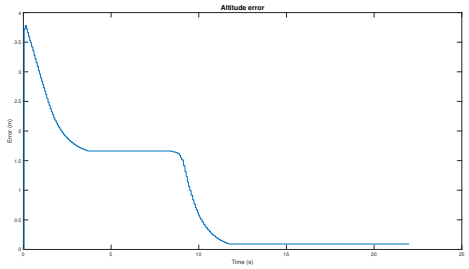

Figure 8: Altitude error UVMS, with gain = 1.0.

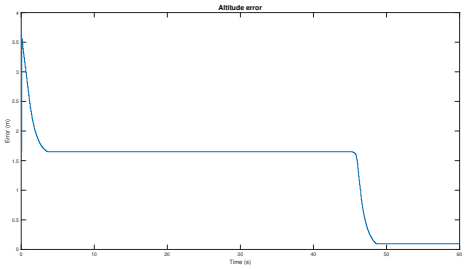

Figure 9: Altitude error UVMS, with gain = 0.1.

It can be seen, especially in the altitude error comparison in Figure 8 and Figure 9, how the alignment task takes much longer to complete with a 0.1 gain, causing the robot to stay at the same altitude for a longer period of time. In the mentioned figures, it can also be appreciated how the different actions are easily distinguishable. Namely, the first action is *Safe Navigation*, where the vehicle lowers its altitude while safely approaching the goal; the second action is *Vehicle Aligning*, where the vehicle keeps its altitude while rotating on its z-axis to align the tool with the rock frame; the third action is *Landing*, where the vehicle lands on the seafloor; the fourth and final action is *Grasping*, where the vehicle keeps its position on the seafloor, not moving anymore.

As for the misalignment error, whose comparison with the two gains can be seen in Figure 6 and Figure 7, it is evident how it approaches the threshold of 0.02 rad slower with a gain of 0.1 for the **Horizontal Alignment** task.

To have a **gain-independent** solution which also guarantees that the landing is accomplished aligned to the target, a bias term can be introduced to the task reference such that:

$$\dot{\tilde{x}}_{at/w} = \lambda \cdot (0 - ||^{w}\rho_{at}||) - \mu, \quad \mu,\ \lambda \in \mathbb{R}^{+} \tag{8}$$

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.

After the landing is accomplished, the end-effector moves, trying to reach the rock and align with its frame. Given that the **ZVC** task was implemented exactly for the vehicle to keep its position still and its velocities null after landing, the movement of the end-effector does not influence the vehicle, as it can be seen if Figure 2 and Figure 3. However, without the **ZVC** task, the vehicle would follow the manipulator movement, which could cause damages in real-life scenarios.

In the provided world configuration, the distance between the landing point and the rock position is sufficient for the end-effector to reach it after the vehicle lands. However, without the **HAL** task, it may happen that, once landed, the vehicle is not in a proper orientation for the manipulator to reach the target. To validate this, the plot shown in Figure **??**an be analyzed.

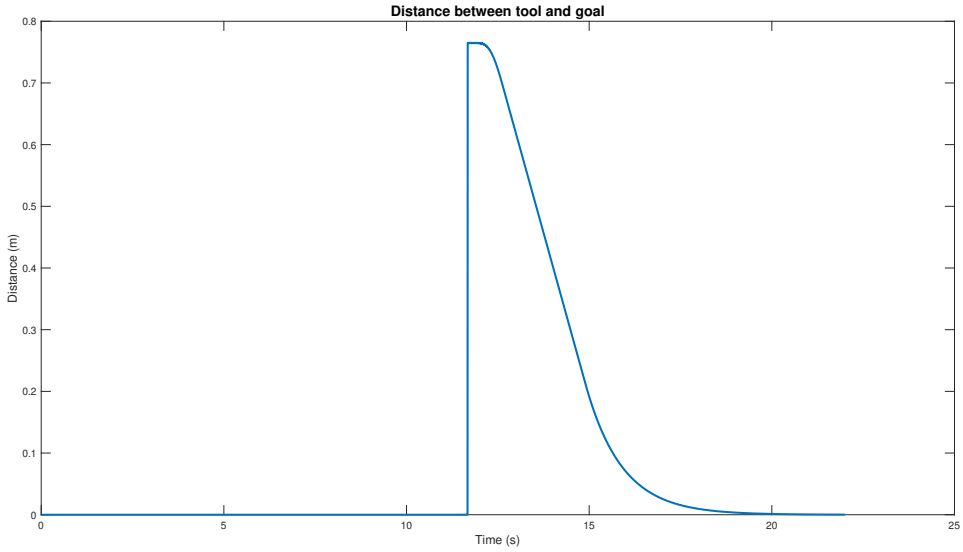

Figure 10: Distance between end-effector and goal.

As expected, the correct value for the initial distance between the end-effector and the goal is initialized once the *Grasping* action begins, around 12s after the simulation starts. Then, the end-effector starts approaching the goal and reaches it successfully after about 8s.
It has to be noted, however, that this assumption is true only for the ad-hoc world provided; in different or real-life scenarios, an additional task checking the distance between the manipulator and the target would be needed.

As for the plot of the velocities of the vehicle and the manipulator, it is possible to see the observed behaviour in Figure 3 and 2. In particular, you should observe the second subplot of each figure starting from 12s, which is the time when the *Landing* phase starts.

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.

In order to ensure that, once the vehicle landed, the nodule is within the manipulator's workspace, the **HAL** task was implemented, along with a separate action *Vehicle Aligning*.
As explained in Section 1.1, the **HAL** task aligns the longitudinal axis of the manipulator's end-effector with the rock frame, thus making the vehicle rotate until the misalignment is reduced below a certain threshold. This ensures that, once the robot landed, the manipulator is already in the correct orientation and, consequently, can reach and grasp the rock. The last statement relies on the assumption that, in the provided work for the simulation, the distance between the landing point and the rock position guarantees that, with a proper landing as described, the nodule is in the manipulator's workspace.

Once grasping is achieved, the robot is in the configuration shown in Figure 11.

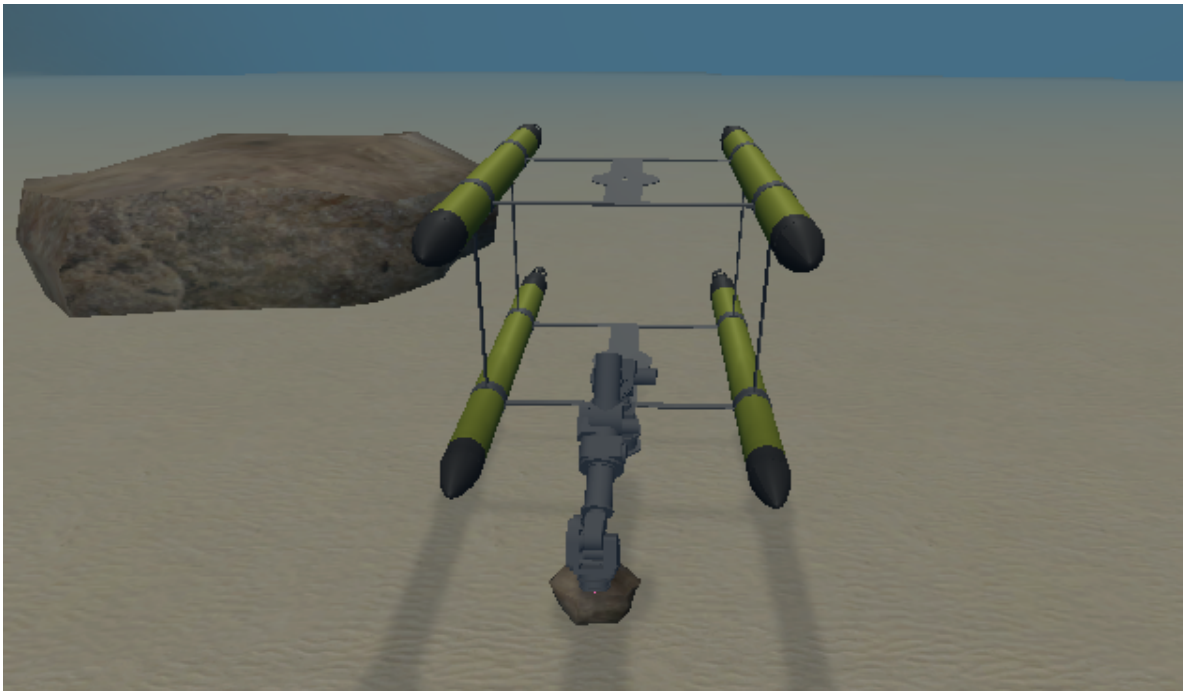

Figure 11: Vehicle at the final state.

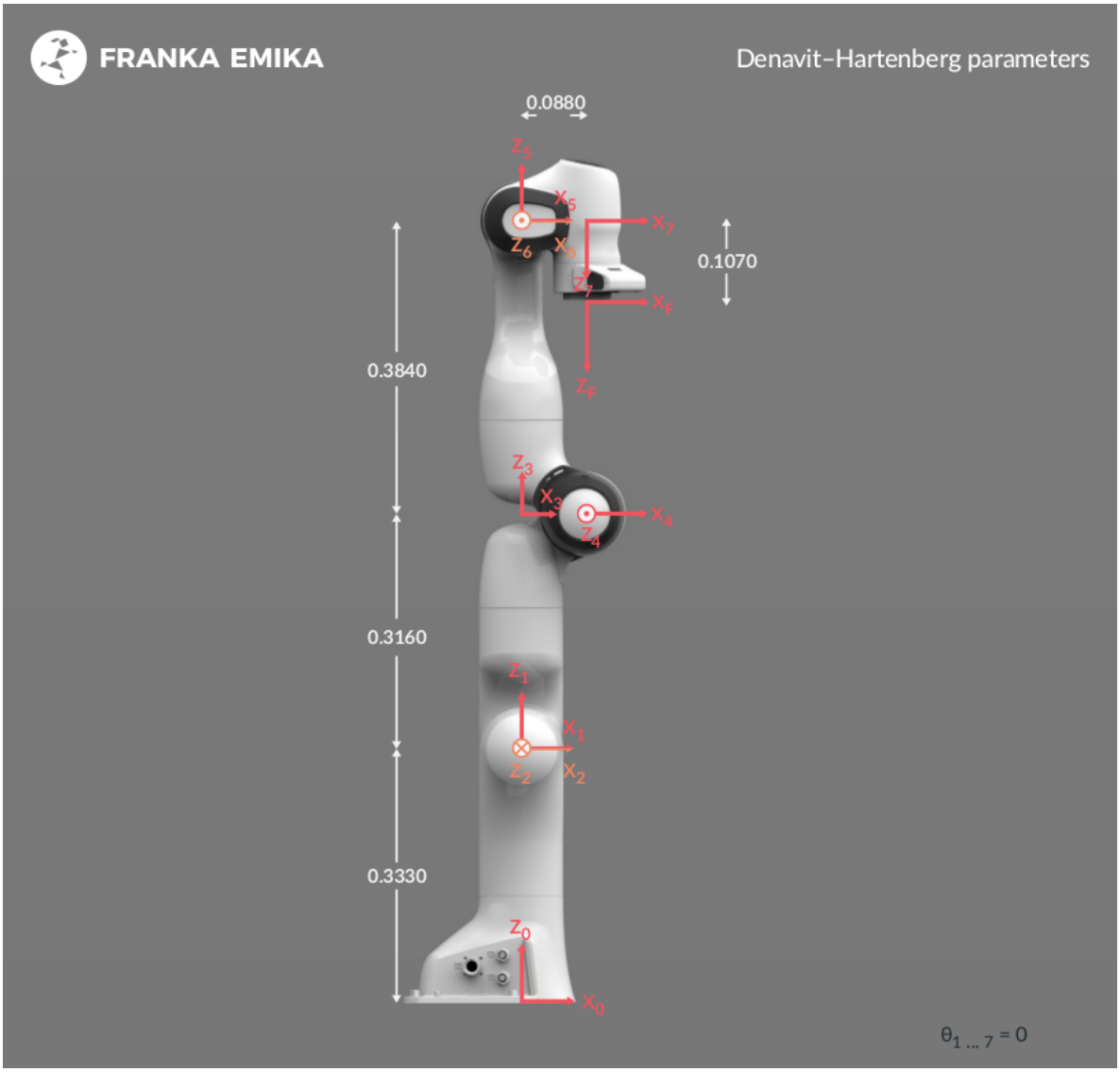

Figure 12: Robot Specifications

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 12)). 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.

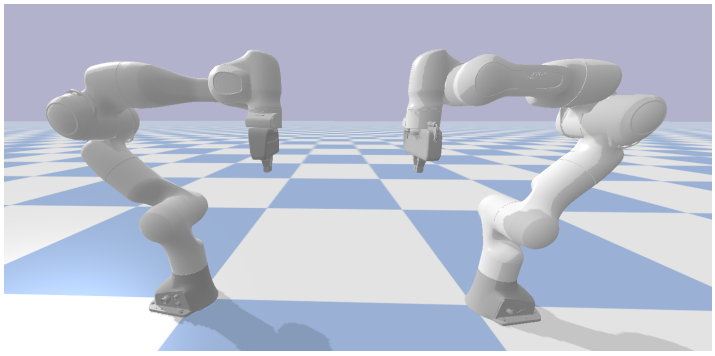

Figure 13: Initial position of the robots

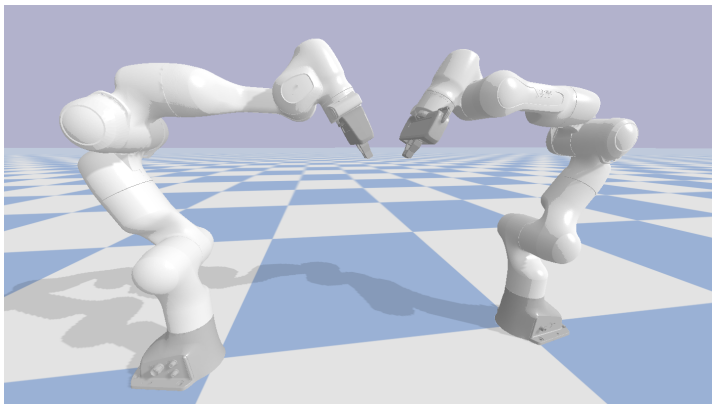

Figure 14: Relative orientation at the grasping position

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 14.

$$ {}^{w}\mathrm{O}_{o} = [0.5, 0, 0.59]; \tag{9} $$

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

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.

The following Table 3 outlines the task hierarchy for Bimanual Manipulation.

Task	Type	Moving towards the object	Grabbing and moving the object towards the goal	Stop the motion
EE minimum altitude	I,S	1	2	1
Joint limits	I,S	2	3	
Rigid constraint	E,C		1	
Tool/EE pose	E,AD	3	4	

Table 3: Hierarchy table for Bimanual Manipulation

Please, note the meaning of "*type*" of actions:

- I, S: inequality, safety
- E, C: equality, constraint
- E, AD: equality, action constraint

In all the actions, we give higher priority to safety and constraint tasks. This is done to make sure that the arms do not fall into any fault position, due to both collisions with the base table and joint limits.
This is the meaning of each task:

- **EE minimum altitude**: to avoid the arms going too far down
- **Joint limits**: to avoid the joint reaching their limits
- **Rigid constraint**: to avoid the arms break the grasped object
- **Tool/EE position**: to control the tool position and move it where wanted

On the other hand, three actions are used for the Bimanual Manipulation, in sequence:

- A_1**: Moving towards the object:** the two arms move to the target point independently, always respecting the minimum altitude and the joint limits tasks
- A_2**: Grabbing and moving the object towards the goal:** firstly, the tools grab the object at the two extremities. Once the grasping is secured, the two arms move the object towards the final target position, always respecting the rigid constraint along with the other tasks.
- A_3**: Stop the motion:** once the final target position has been reached, all motions are stopped, so that the two arms can keep the final configuration while always respecting the minimum altitude safety.

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

In general, all the Jacobians for the Bimanual Manipulation should have a fixed dimension of $m \times 14$, where m corresponds to the row dimension of the task and its reference rate. Given that, in this application, the control for the arm joints is unified for both manipulators and each one has 7 DOF, the Joint Limits Jacobian matrix J_{jl} has dimensions 14×14. In particular:

$$J_{jl} = \begin{bmatrix} I_{14\times 14} \end{bmatrix} \tag{11}$$

The identity matrix was chosen because, in the task reference computation, we directly control the velocity of the joints, which is saturated based on the joint velocity limits reported in the datasheet, both for linear and angular velocity.

As for the task joint limits task reference, the aim was to generate a velocity which moved the joints towards the midpoint between the minimum and the maximum joint limits, starting from the current position of the joints. Therefore, the task reference velocity was computed as:

$$\dot{\bar{x}}_{jl} = \lambda \cdot (\frac{jlmax + jlmin}{2} - q) \tag{12}$$

where:

- λ is a gain
- q is the vector of current joint positions of the arm
- *jlmax* is the vector of maximum joint limits
- *jlmin* is the vector of minimum joint limits

This formula is generally written, but actually it was computed for both left and right arm. Only λ was kept unique for both arms.
In the simulation, we set λ to 1 because it is slower than real scenario; during the tests with real robots the gain value was approximately 0.1/0.2 for safety.

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

For Bimanual Rigid Constraint, we need the tool frame $< t >$ of each manipulator to be aligned with the object frame $< o >$. For this reason, the velocities of both end-effectors have to be equal, in order to achieve a synchronous movement.
Therefore:

$$\dot{x}_{o,left} = \dot{x}_{o,right} \tag{13}$$

In terms of Jacobians and joint velocities:

$$\begin{aligned} {}^{w}_{o}J_{left} \cdot \dot{q}_{left} &= {}^{w}_{o}J_{right} \cdot \dot{q}_{right} \\ {}^{w}_{o}J_{left} \cdot \dot{q}_{left} - {}^{w}_{o}J_{right} \cdot \dot{q}_{right} &= 0 \end{aligned} \tag{14}$$

As a result, the Jacobian for the rigid constraint was defined as the following 6×14 matrix:

$$J_{rc} = \begin{bmatrix} {}^{w}_{o}J_{left} & - & {}^{w}_{o}J_{right} \end{bmatrix} \tag{15}$$

The obtained matrix ensures that, while the Bimanual Rigid Constraint task is active, the two arms satisfy the kinematic constraints necessary to obtain and follow a synchronous movement while grasping and moving the object towards the final target point.

The task reference is computed in this way:

$$\dot{x}_{rc} = 0_{6\times 1} \tag{16}$$

Indeed, for the Bimanual Rigid Constraint task, the common velocity vector $\dot{x}_{rc}$ must be set to zero, to reflect the fact that there should be no relative motion between the object and the end effectors, both in position and orientation. This condition secures a solid grasping, while not damaging the object or executing any other unwanted behaviour.

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)

2.4.1 Feasible configuration

To prove the correctness of the code in the first case, we tested it in both simulation and on the real robot. This could be done for this case as only safe target points were given to the robots.

Simulation

The firsts test have been done in simulation to ensure the correct functioning of the system.

Real robot

Once the code was double-checked, it was tried on the real system, with some arrangements to guarantee the safety of the working environment, such as changing the λ gain from its simulation value of 1 to 0,2.

The following analysis of activations functions and joint velocities is driven on the plots coming from the real use of the robots.

Activation functions

Figures (15) and (16) show the behaviour of the activation functions for both the left and the right robot.

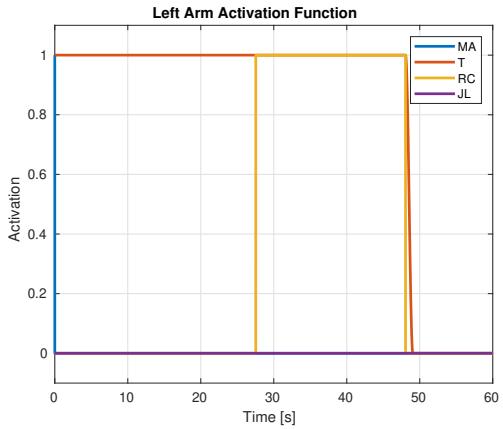

Figure 15: Left Arm: activation functions

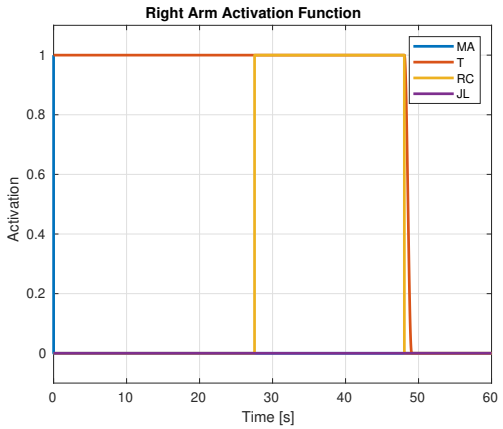

Figure 16: Right Arm: activation functions

The tasks are activated at the proper moment, according to their functionalities:

- **Minimum Altitude**: this task is active at the beginning for both arms, but immediately goes to 0 and keeps this value for the whole test. This was expected, since both the target points that the end-effectors needed to reach had a z-axis value greater than 0.15 m;

- **Joint Limits**: this task remains deactivated throughout the whole test because the joints of each arm never got too close to their limits, which would have caused the Joint Limits task to activate;
- **Rigid Constraint**: this task requires a binary transition for its activation, without the smoothness provided by the bell function. This was needed to reflect the binary nature of the task itself, constraining the two robots to move the object rigidly, together and the results obtained in Figure 15 and Figure 16 show that it was respected.
- **Tool/EE pose**: this task has a smooth deactivation and is correctly executed when the switch to the third action, so *Stop the motion*, happened.

Joint Velocities

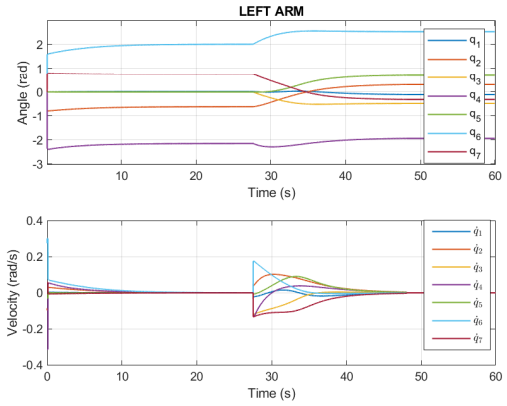

Figure 17: Left Arm Joint Positions and Velocities

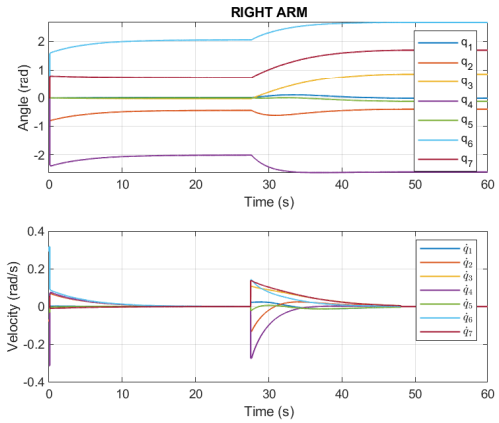

Figure 18: Right Arm Joint Positions and Velocities

Figures for desired angular velocities vs actual ones:

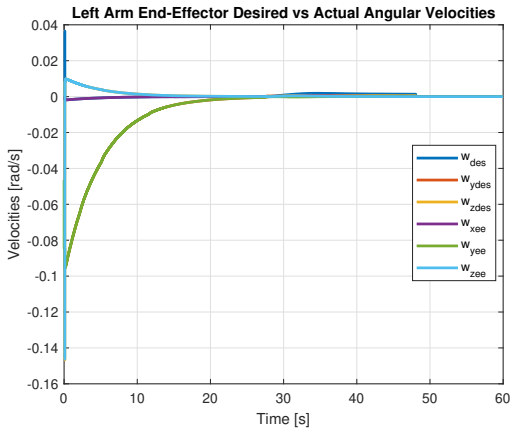

Figure 19: Left Arm Desired Angular Velocity

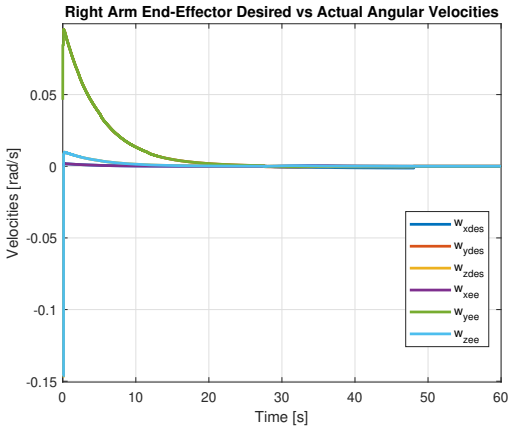

Figure 20: Right Arm Desired Angular Velocity

Figures for desired linear velocities vs actual ones:

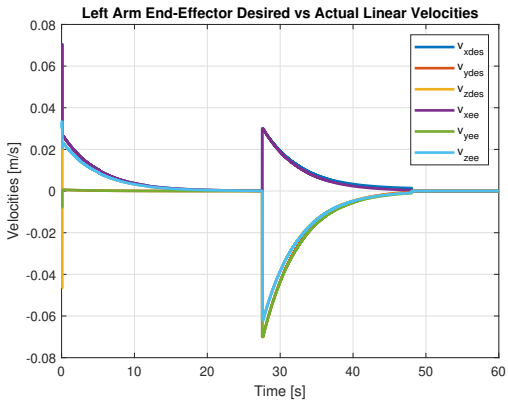

Figure 21: Left Arm Desired Linear Velocity

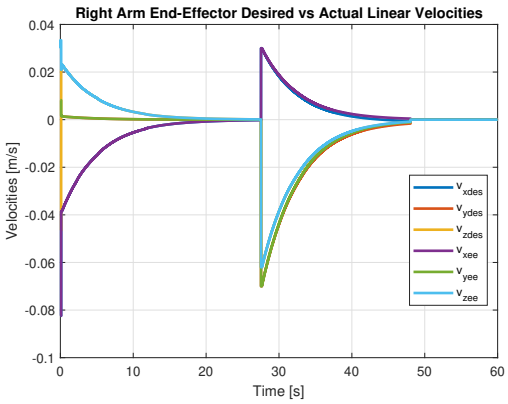

Figure 22: Right Arm Desired Linear Velocity

Images taken during the laboratory session, both at the initial and final state:

Figure 23: Initial Position

Figure 24: Final Position

Particular importance was given to ensuring that the distance between the tools remained constant. The Rigid Constraint requirement was successfully met as shown in Figure(25).

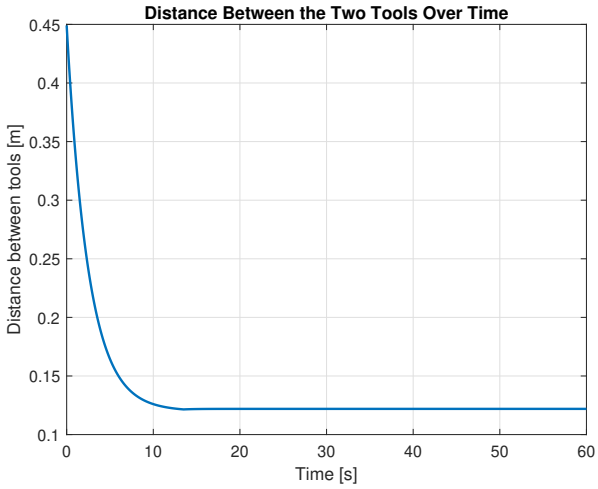

Figure 25: Distance between the two tools during the whole simulation

2.4.2 Unfeasible configuration

This section shows two different unfeasible configurations implemented to trigger the activation of safety tasks. Of course the following plots

Minimum altitude

To guarantee the safety of the two robots, the Minimum Altitude they can achieve is $20cm$ from the floor. To test the correct activation of this safety task, the following goal was set:

$$O_o^w = [0.5, 0, 0.01]^T. \tag{17}$$

As it is possible to see in figures (26) and (27), as the robots are about to reach the object position placed below the minimum altitude threshold, the minimum altitude safety task activates to limit the motion.

Furthermore, the Rigid Constraint task is never active, since the robots cannot grasp the object and reach the following mission phase.

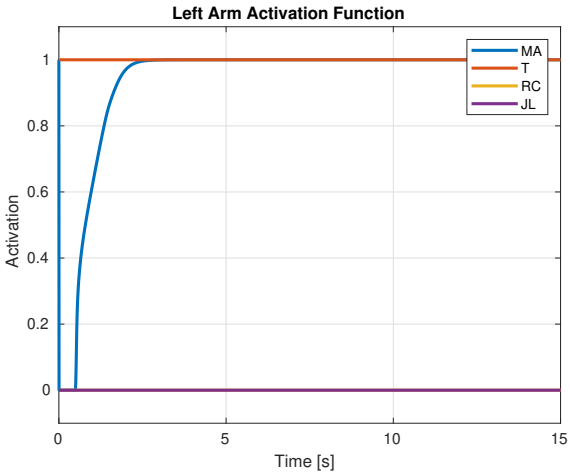

Figure 26: Left Arm: activation functions with Minimum Altitude task running

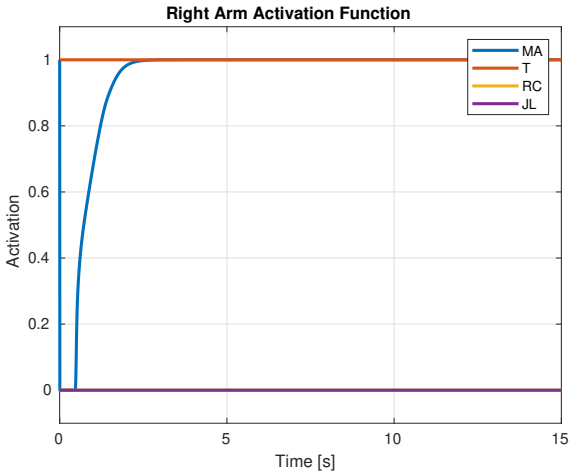

Figure 27: Right Arm: activation functions with Minimum Altitude task running

The following plots also show the actual velocities of the two robots do not match the desired one, as the goal can not be reached.

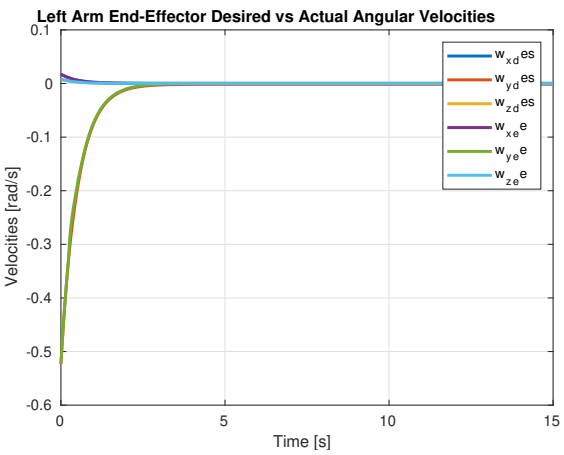

Figure 28: Left Arm Desired VS Actual Angular Velocity

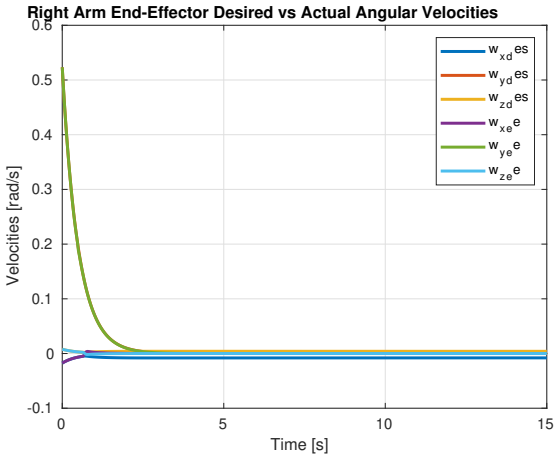

Figure 29: Right Arm Desired VS Actual Angular Velocity

Figures for desired linear velocities vs actual ones:

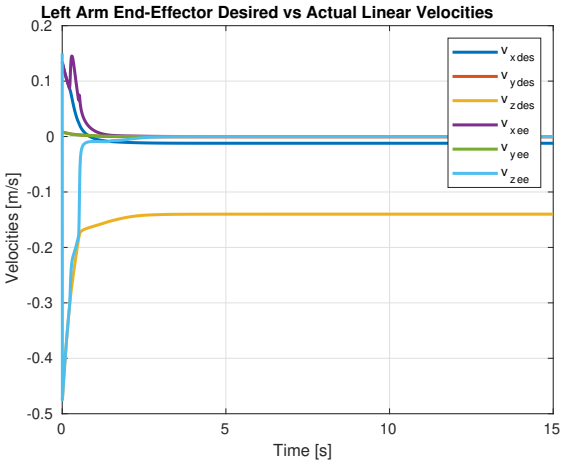

Figure 30: Left Arm Desired VS Actual Linear Velocity

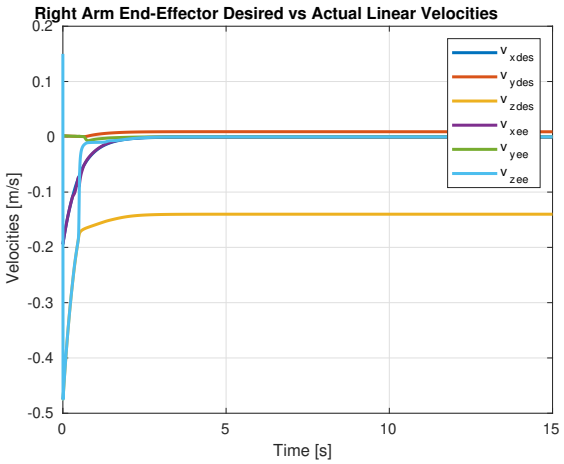

Figure 31: Right Arm Desired VS Actual Linear Velocity

Joint Limits

A safety Joint Limits task was implemented to guarantee that the robots never reach singular configurations throughout the different mission phases. Given the number of degrees of freedom of the robotics arm, the target points belonging to the workspace of both robots can be achieved without triggering the activation of this task; on the other hand, target points out of both workspaces, cannot be reached, causing the saturation of the velocities, but do not necessarily lead to singular configurations of the arms.

Based on these considerations, the following goal (out of the workspace of the robots) was implemented to verify the correct activation of the task, starting with an initial configuration of both arms near to a singular position.

$$O_o^w = [1.5, 0.1, 0.5]^T. \tag{18}$$

From figures (32) and (33), the Joint Limits task is active since the very beginning and moves towards deactivation as the robots get away from the singular configuration.

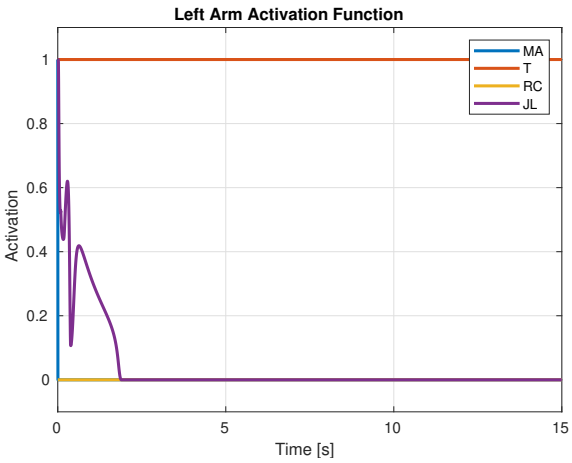

Figure 32: Left Arm: activation functions with Joint Limits task running

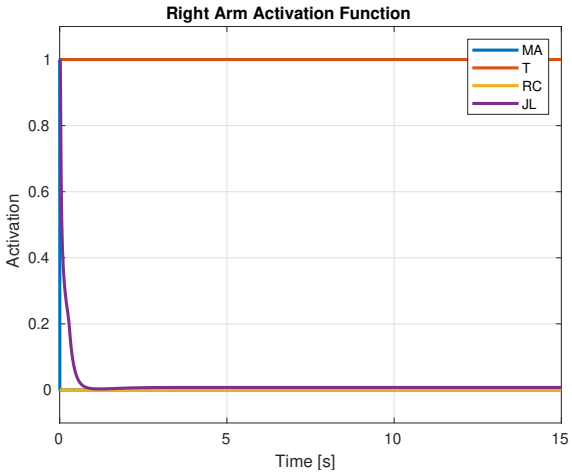

Figure 33: Right Arm: activation functions with Joint Limits task running

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.

$$ {}^{w}\mathrm{O}_{o} = [0.5, 0, 0.59]^{\top}(m); \tag{19} $$

$$ l_{obj} = 6\,(\mathrm{cm}). \tag{20} $$

   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

$$ {}^{w}\mathrm{O}_{g} = [0.60, 0.40, 0.48]^{\top}(m) \tag{21} $$

   (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.

The following table contains the unified hierarchy of tasks used and their priorities in each action.

As in the previous section, the type refers to:

Task	Type	Moving towards the object	Grabbing and moving the object towards the goal	Stop the motion
EE minimum altitude	I,S	1	2	1
Joint limits	I,S	2	3	
Tool/EE pose	E,AD	3	1	

Table 4: Hierarchy table for Cooperative Manipulation

- I, S: inequality, safety;
- E, C: equality, constraint;
- E, AD, C: equality, action constraint.

The tasks can be described as:

- **EE Minimum Altitude**: employed to avoid the arms going too far down;
- **Joint Limits**: used to avoid the joint reaching their limits;
- **Tool/EE pose**: deployed to control the tool position and move it where wanted before and during the cooperation.

EE Minimum Altitude and Joints Limits are two inequality safety tasks, as they are activatived below a certain threshold, respectively the altitude or the joint limit.

On the other hand, the Tool/EE pose is an equality task, as a specific position must be satisfied reached, and its can be either:

- action constraint: during the first mission phase, as each robot is considered separately;
- constraint: during the cooperation phase, as the coordination policy must be respected.

As in the previous exercise, three actions are used for the Cooperative Manipulation:

- A_1**: Moving towards the object:** the two arms move to the target point independently, always respecting the minimum altitude and the joint limits tasks:
- A_2**: Grabbing and moving the object towards the goal:** firstly, the tools grab the object at the two extremities. Once the grasping is secured, the two arms move the object towards the final target position, always respecting the common tool frame velocity constraint along with the other tasks.
- A_3**: Stop the motion:** once the final target position has been reached, all motions are stopped, so that the two arms can keep the final configuration while always respecting the minimum altitude safety.

Note that the crucial difference with respect to Bimanual Manipulation is that the arms are considered as two separate collaborating agents. This means that each arm, called a and b for simplicity:

- runs the TPIK procedure as if it was the only system acting on the object;
- evaluates the Cartesian velocity that it would impose on the object if it was alone and its corresponding $\mathbf{H}_i$ matrix, to get the admissible tool-frame velocity space:

$$\dot{\mathbf{x}}_{t,i} = \mathbf{J}_{t,i}\dot{\mathbf{y}}_i, \tag{22}$$

$$\mathbf{H}_i = \mathbf{J}_{t,i}\mathbf{J}_{t,i}^{\#}, \tag{23}$$

  where $i = a, b$;
- transfers its computed quantities to the other arm;

- evaluates, in a distributed way, the cooperative-tool frame velocity vector as the weighted average of the non-cooperative velocities, to find a compromise between the two:

$$\dot{\hat{\mathbf{x}}}_t = \frac{1}{\mu_a + \mu_b}(\mu_a \dot{\mathbf{x}}_{t,a} + \mu_b \dot{\mathbf{x}}_{t,b}) \tag{24}$$

  with $\mu_a, \mu_b > 0$;

- evaluates the Cartesian constraint matrix $\mathbf{C}$, defined as:

$$\mathbf{C} = [\mathbf{H}_a - \mathbf{H}_b]; \tag{25}$$

- considers the cooperative tool-frame velocity vector $\dot{\hat{\mathbf{x}}}_t$ and projects it on the constrained end-effector motion subspace $\text{Span}(\mathbf{H_{ab}}(I - C^{\#}C)$ with $\mathbf{H_{ab}}$ being the diagonal matrix having $\mathbf{H_a}, \mathbf{H_b}$ on its diagonal, to retrieve the feasible cooperative velocity vector $\dot{\tilde{\mathbf{x}}}_t$:

$$\begin{bmatrix} \dot{\tilde{\mathbf{x}}}_t \\ \dot{\tilde{\mathbf{x}}}_t \end{bmatrix} = \mathbf{H_{ab}}(I - C^{\#}C) \begin{bmatrix} \dot{\hat{\mathbf{x}}}_t \\ \dot{\hat{\mathbf{x}}}_t \end{bmatrix} \tag{26}$$

  ;

- runs a new TPIK procedure separately, with the **constrained tool velocity at the top of the hierarchy**, coordinated with the other agent.

Imposing this new hierarchy, both arms can independently optimize their safety tasks under the common tool frame velocity constraint. As before, safety and constraint tasks have the highest priority in all the actions, to prevent the arms from falling in any fault position.

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.

This section is fully dedicated to the analysis of the data collected throughout different simulations, to compare different scenarios and having a better understanding of the system behaviour.

3.2.1 Feasible configuration

The first plots reported are collected running a simulation with feasible object and goals position. The plots are divided into separate sets to facilitate a more detailed discussion of the observed behaviour.

Activation functions

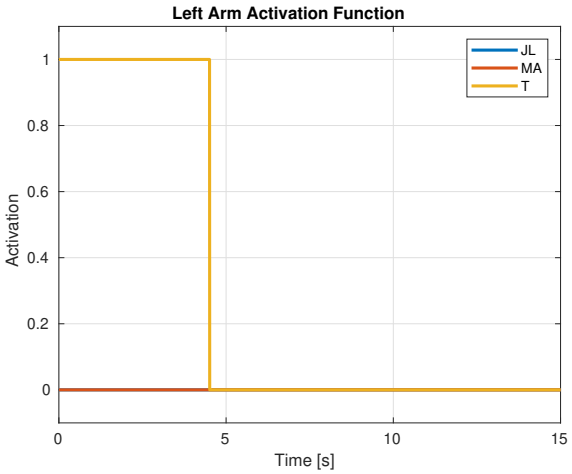

Figure 34: Left Arm: activation functions

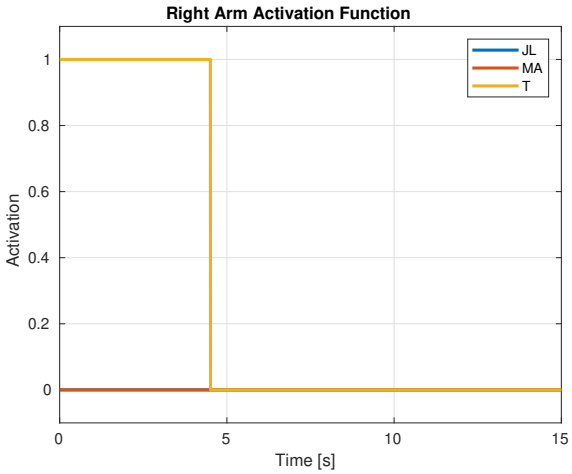

Figure 35: Right Arm: activation functions

Figure 36: Legend: Joint Limits, Minimum Altitude and Tool/EE pose

Figures (34) and (35) represent the activation functions of the left and right arm respectively.

As in the Bimanual Manipulation exercise, it is possible to notice that for feasible object and goal positions, the joint limits and minimum altitude tasks are never activated, since any dangerous movement is performed. For this simulation the following positions have been used:

$$^{w}\mathrm{O}_{o} = [0.5, 0, 0.59]^{\top}(m), \tag{27}$$

$$^{w}\mathrm{O}_{g} = [0.60, 0.40, 0.48]^{\top}(m). \tag{28}$$

The behaviour of the activation functions can be justified by breaking down the simulation into different phases and analyzing each:

- at the beginning of the simulation both robots move to the grasping position as directed by action A_1: during this phase, the only active task involves controlling the tool position to guide both arms to the target point;
- after the grasping the robots cooperatively transport the object to its goal position as defined by action A_2: since the tool task remains continuously active, it is not evident from the plots alone that cooperative velocity is now a factor in the task execution.
- after the goal is reached the last action A_3 stops the motion of both arms: it deactivates all the tasks except for the minimum altitude one, as required.

Joint Velocities

The simulation results provide evidence that the two cooperating Franka robots perform reliably throughout the task execution.

In general, the actual robots' velocities closely match the desired ones, indicating good performances for the whole system; the implemented algorithms appear effective in managing both individual and cooperative tasks.

The first set of plots in figures (37) and (38) show the joint velocities for the left and right arms, without any anomalous behaviour.

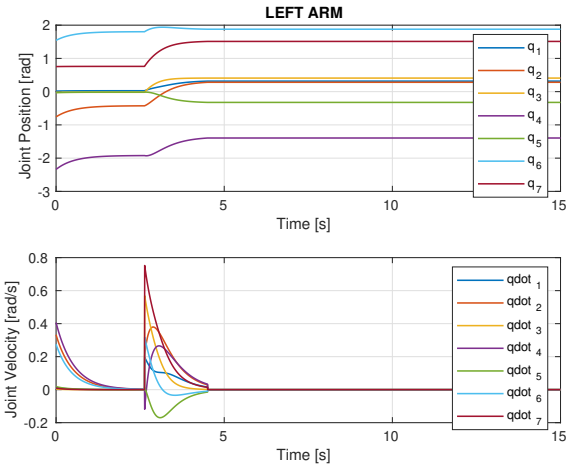

Figure 37: Left Arm Joint Positions and Velocities

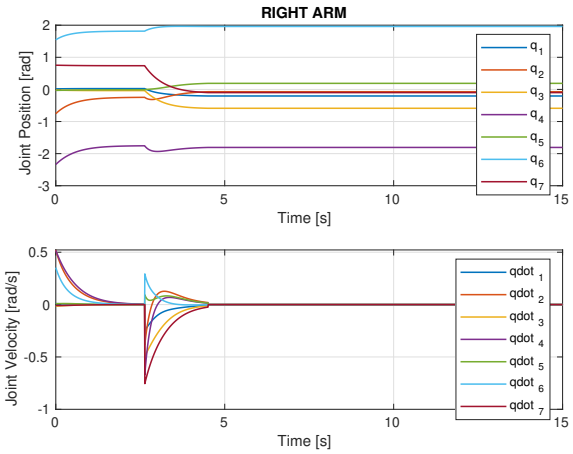

Figure 38: Right Arm Joint Positions and Velocities

Plots (39), (40), (41) and (42) represent the comparison between desired and actual angular and linear velocity for both arms. As it is possible to notice, the robots velocities follow the desired ones with minimal discrepancy.

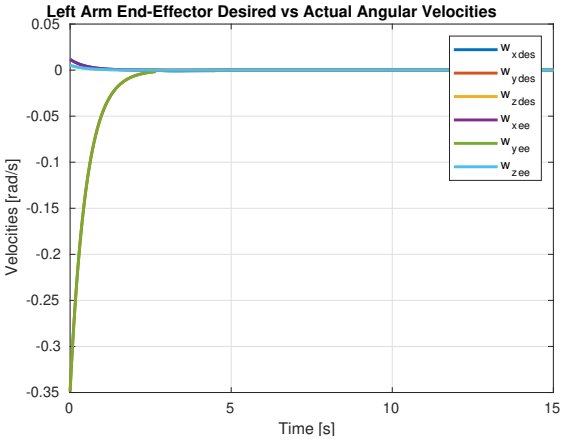

Figure 39: Left Arm Desired Angular Velocity

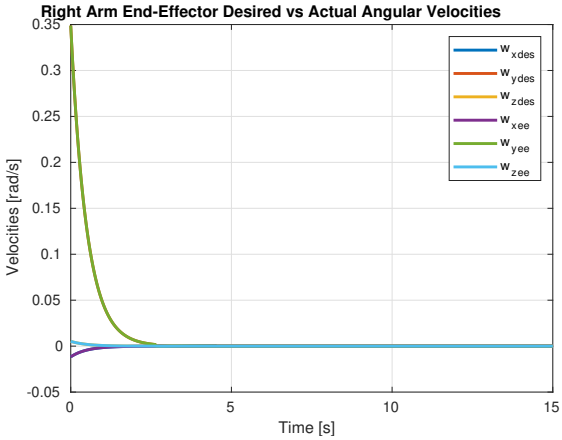

Figure 40: Right Arm Desired Angular Velocity

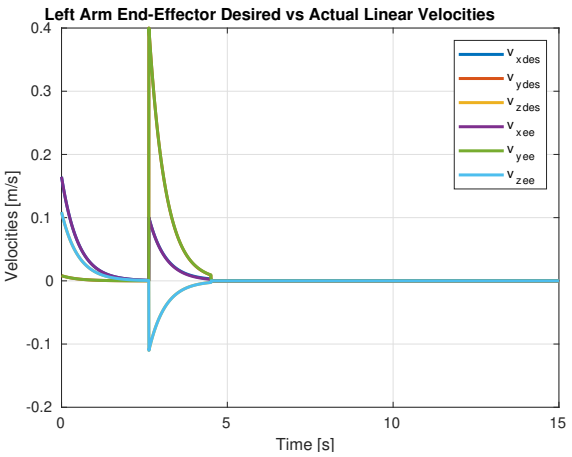

Figure 41: Left Arm Desired Linear Velocity

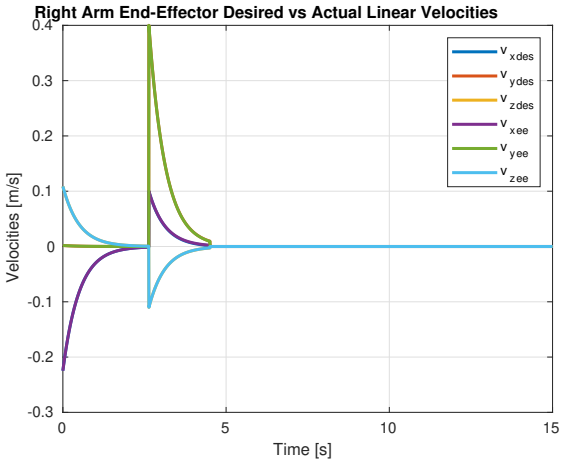

Figure 42: Right Arm Desired Linear Velocity

This can also be seen from figures (43) and (**??**), that show the comparison between the non-cooperative velocities of the two tools and the desired velocity.

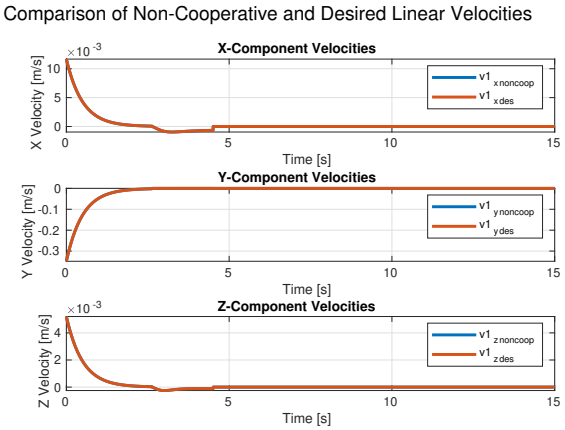

Figure 43: Left Arm: Non-cooperative vs desired linear Cartesian velocities

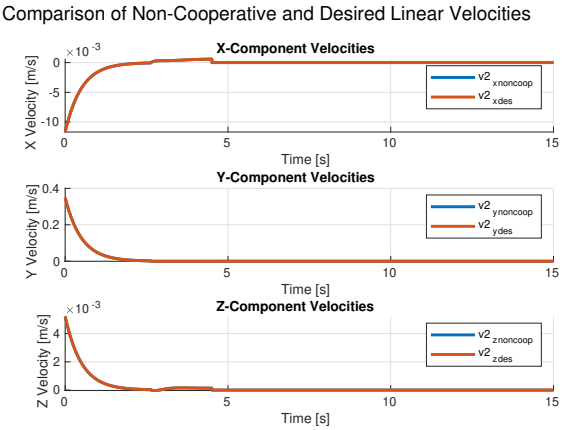

Figure 44: Right Arm: Non-cooperative vs desired linear Cartesian velocities

The last two plots are about the comparison of cooperative and non-cooperative linear and angular velocities.

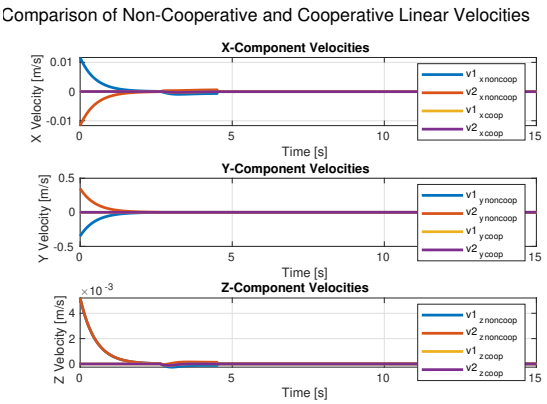

Figure 45: Cooperative Vs Non Cooperative Linear Velocities

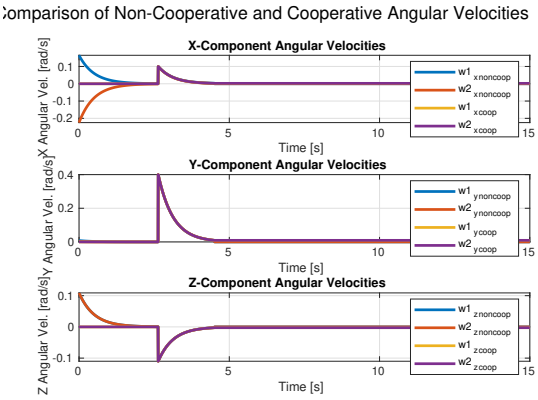

Figure 46: Cooperative Vs Non Cooperative Angular Velocities

During the initial phase of the simulation, the linear non-cooperative velocities exhibit significant deviations from the desired object velocity, since the robots are not cooperating yet. Then, during the second mission phase, which requires cooperation, they align closely to the cooperative one, ensuring synchronization between the end-effectors.

The same conclusion can be driven for the angular cooperative and non cooperative velocities.

Tools distance

This last plot (47) represents how the distance between the tools diminishes over time. Starting from an initial significant distance, it decreases as the robots move towards the objects and remain the same during the manipulation task, when they are constrained to move the object rigidly to its goal position.

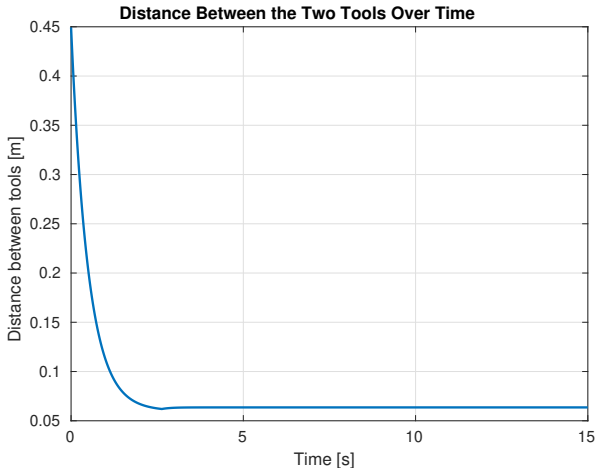

Figure 47: Tools distance over the simulation time

3.2.2 Unfeasible configuration

Minimum altitude

To guarantee the safety of the two robots, the Minimum Altitude they can achieve is $20cm$ from the floor. To test the correct activation of this safety task, the following goal for the object was set:

$$O_o^w = [0.5, -0.35, 0.10]^T. \tag{29}$$

The robots were requested to reach a position below the minimum altitude allowed.

As it is possible to see in figures (48) and (49), as the robots are about to reach the object goal position placed below the minimum altitude threshold, the minimum altitude safety task activates to limit the motion.

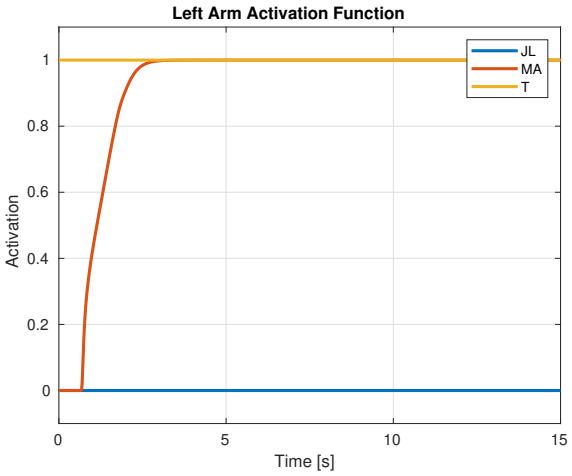

Figure 48: Left Arm: activation functions with Minimum Altitude task running

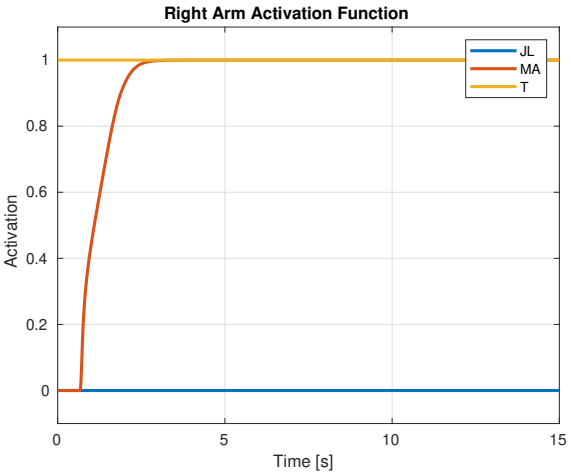

Figure 49: Right Arm: activation functions with Minimum Altitude task running

Remaining in the Minimum Altitude task activation context, a peculiar result is the following. The object position is placed at:

$$O_o^w = [0.5, -0.35, 0.19]^T, \tag{30}$$

so just a little below the $20cm$ altitude threshold.

The expected behaviour is the previous one: the Minimum Altitude task activates preventing the

robot from going under the threshold.

However, as it is possible to notice in figures (50) and (51) the Minimum Altitude task activates with a bell-shaped function, making it slow. Having the target so little below the threshold, the object position is reached before the Minimum Altitude task is fully active, thus the object is grabbed anyway. However, even if the object is grabbed, the minimum altitude task remains active preventing them to perform any other motion.

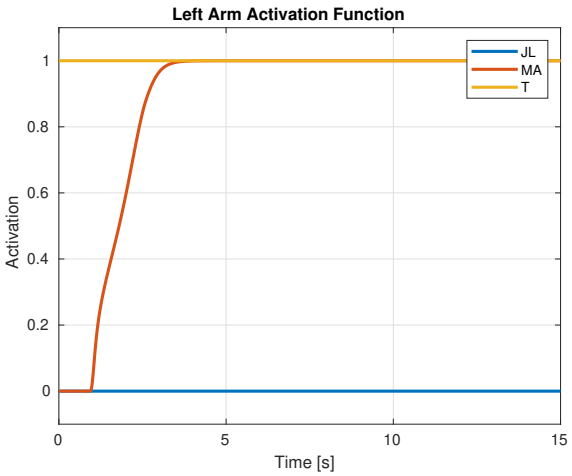

Figure 50: Left Arm: activation functions with Minimum Altitude tasks running

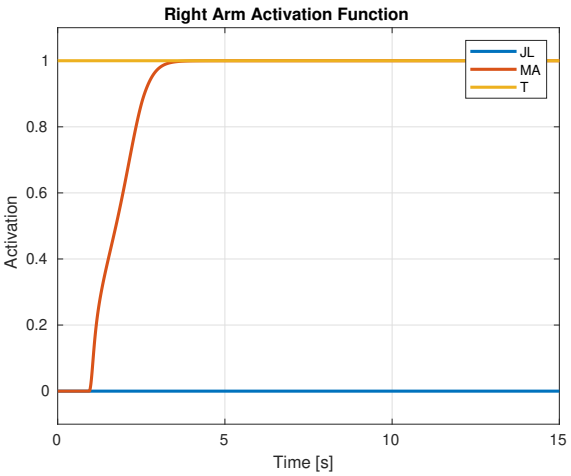

Figure 51: Right Arm: activation functions with Minimum Altitude tasks running

Non-cooperative and desired velocities comparison

To better understand the behaviour of the non-cooperative velocities with respect to the desired ones, different simulations were run.

In a general case, with reachable object and object's goal positions, the non-cooperative velocities adapt very precisely to the desired ones.

However, when the object's goal position lays out of the workspace of the robots, different behaviours may arise.

Assume to place the object's goal in a position reachable by just one of the two robots. In this case, the non-cooperative velocities and the desired ones cannot coincide anymore, as the goal is unfeasible for the two robots while cooperating.
The goal position is:

$$O_g^w = [0.2, 0.8, 0.38]^T. \tag{31}$$

In this situation, the robot that cannot reach the goal does not assume a singular configuration but instead saturates its joint velocities because it cannot reach the target. This behavior is probably a consequence of the redundancy of the robotic arms, which each have seven degrees of freedom.

Starting from this assumption it is possible to notice from the comparison of the Cartesian velocities in figures (52) and (53), the right robot velocities do not adhere to the desired ones as it reaches its maximum extension towards the goal. On the other hand, the goal is feasible for the left robot, but it is constrained by the cooperative task, so it cannot follow the desired velocity as if it was alone either.

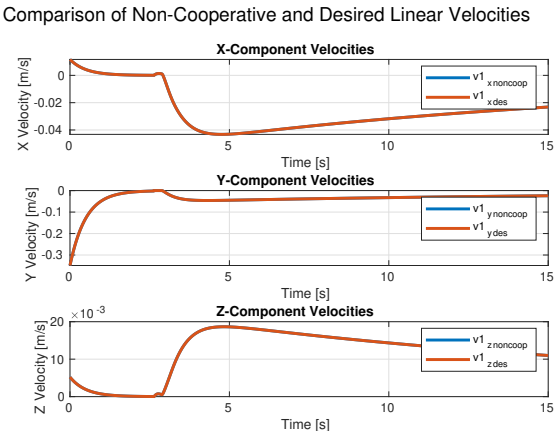

Figure 52: Left Arm: Non-cooperative vs desired linear Cartesian velocities

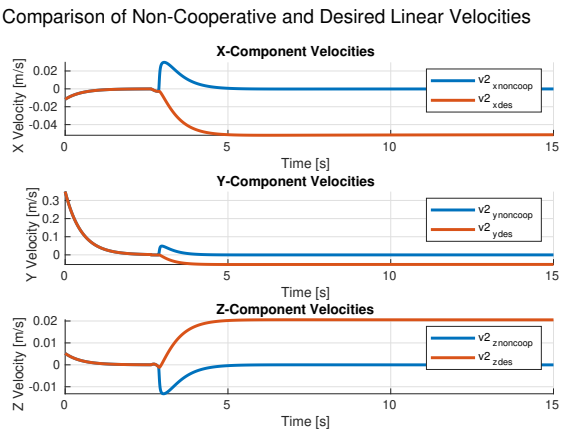

Figure 53: Right Arm: Non-cooperative vs desired linear Cartesian velocities

These figures illustrate how the two robots must adapt to cooperate, highlighting that, during the initial stages of the cooperative movement, the velocities they follow may not necessarily coincide with their non-cooperative counterparts.

As mentioned earlier, the joint velocities do not perfectly match the desired ones, particularly for the right robot, which attempts to reach the target position but fails as it is outside its workspace.

One might wonder why, in Figure (52) the left robot's velocities appear to match the desired ones. This behavior probably arises due to the task priority hierarchy: since the task with the highest priority is the Tool one, that requires to bringing the object to its final position.