Cooperative Robotics course

University of Genoa

Cooperative robotics final report

Report

Students: Authors: Aurora Bertino, Chiara Saporetti, Sara Romano
EMAILs: bertino.aurora16@gmail.com,
chiara.saporetti@gmail.com,
sara.romano.15@gmail.com

Year: 2020 - 2021

Contents

Abstract

This project implements a Task Priority Control on an Underwater Floating Manipulator System. The code is written in MATLAB and simulated on a provided Unity-based simulation. Results are tested on two different UVMs: Robust (exercises 1-4) and Dexrov (exercises 5-6).

For the Robust UVM we implemented a control based on 4 consecutive actions: Safe Waypoint Navigation, Vehicle/rock Alignment, Basic Landing Action, the Fixed-Base Manipulation.

For the DexROV we implemented 2 of the aforementioned actions (Safe Waypoint Navigation and Fixed-Base Manipulation) and we adopted a parallel coordination scheme to split the control of the vehicle and of the arm.

The code can be found at https://github.com/sararom15/Cooperative_Robotics

1 Exercise 1: Implement a "Safe Waypoint Navigation" Action.

1.1 Adding a vehicle position control objective

Initialize the vehicle far away from the seafloor. An example position could be

$$\begin{bmatrix}10.5 & 35.5 & -36 & 0 & 0 & \pi/2\end{bmatrix}^\top$$

Give a target position that is also sufficiently away from the seafloor, e.g.,

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

Goal: Implement a vehicle position control task, and test that the vehicle reaches the required position and orientation.

The underwater floating manipulator system (UVMS) is a robotic system composed of:

- a vehicle (6 DOFs)
- a manipulator (7 DOFs)

The overall system can be expressed by a configuration system:

$$c = \begin{bmatrix} q \\ \eta \end{bmatrix} \tag{1}$$

where $q = [q_1....q_7]'$ is the arm joint positions vector $\subset \mathbb{R}^l$, with l=7 and

$$\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \in \mathbb{R}^6 \tag{2}$$

refers to the the position and orientation of the vehicle with reference to the world frame. In particular:

$$\eta_1 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \eta_2 = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \tag{3}$$

where η_1 is the vehicle frame position and η_2 is the vehicle frame orientation expressed through its roll-pitch-yaw (RPY) representation which allows to define the rotation matrix ${}^w R_v$.

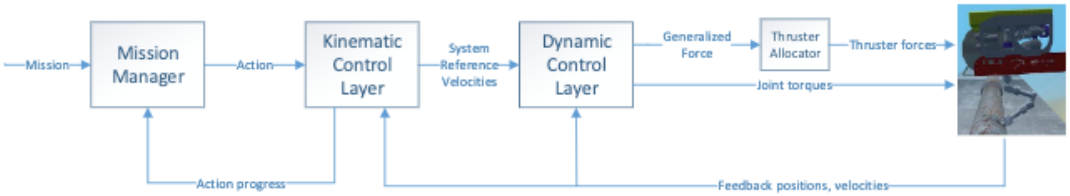

Figure 1: Architecture

In order to control the UVM we consider the overall architecture to be the one shown in 1. What we implemented in our project is the Mission Manager (in this exercise we have one action only, but others are implemented in the next exercises) and the Kinematic Control Layer (KCL), which is the one implementing the task priority approach and executing the current action scheduled by a Mission Manager. The outputs of the Kinematic Control Layer are the system velocities, tracked by the Dynamic Control Layer.

The vehicle is controlled to reach a target position defined into the world frame w. To tackle this scenario, two (equality) control tasks are taken into account: one to control the linear position and one to control the angular attitude of the vehicle. We divided the task in a linear and an angular part because we wanted to have the possibility to control both parts separately in case we only needed to activate one.

In this scenario, other objectives must be satisfied too, such as the the vehicle horizontal attitude, implemented to always maintain the vehicle aligned with respect to the sea floor. Keeping the horizontal attitude requires a lot of effort (in terms of energy consumption). For this reason we implemented this task as an inequality one, in such a way that it is activated only when the misalignment is above a threshold.

1.1.1 Q1: What is the Jacobian relationship for the Vehicle Position control task? How was the task reference computed?

The Jacobian matrix is a tool that links the controlled vector $\dot{y}$ to the reference vector $\dot{x}$.
The Jacobian relationship is the following:

$$\dot{x} = J\dot{y} \tag{4}$$

Where the controlled vector $\dot{y}$ includes the derivative of the arm joint velocities vector and the vehicle velocity vector projected on the v frame, and is therefore defined as:

$$\dot{y} = \begin{bmatrix} \dot{q}_1 .... & \dot{q}_7, & v_x, & v_y, & v_z, & \omega_x, & \omega_y, & \omega_z \end{bmatrix}^T \tag{5}$$

While the desired velocity vector $\dot{x}$ of each task is the velocity vector that the system should have to reach the goal w.r.t the world frame w.
For the angular velocity control task, the **reference vector** is:

$$\dot{x}\_ang = \begin{bmatrix} \omega_x, & \omega_y, & \omega_z \end{bmatrix}' \tag{6}$$

While for the linear velocity control task, the **reference vector** is:

$$\dot{x}\_lin = \begin{bmatrix} \dot{x}, & \dot{y}, & \dot{z} \end{bmatrix}' \tag{7}$$

To build the two [3x13] Jacobians we only considered the part that is multiplied for the vehicle, and kept to zero the arm part. To obtain the linear and angular control tasks, we also divided the **Jacobian** into a linear and an angular part, as previously stated.
The two resulting matrices are:

$$J\_v\_lin = \begin{bmatrix} 0 & ... & 0 & 0 & ... & 0 & \\ 0 & ... & 0 & 0 & ... & 0 & {}^wR_v \\ 0 & ... & 0 & 0 & ... & 0 & \end{bmatrix} \tag{8}$$

$$J\_v\_ang = \begin{bmatrix} 0 & ... & 0 & & 0 & ... & 0 \\ 0 & ... & 0 & {}^wR_v & 0 & ... & 0 \\ 0 & ... & 0 & & 0 & ... & 0 \end{bmatrix} \tag{9}$$

In this case, the **activation function** is an identity matrix for each task.

$$a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

To find the reference vector used to accomplish the task, we computed the *Cartesian error* that considers the difference between the vehicle initial and goal position. This is implemented by using the given function *CartError* that takes as input two transformation matrices in the world frame (related to the linear and to the angular part) and computes the error between the two frames: difference / misalignment. The linear part is computed as difference between the frames while the angular part is calculated via the *Versor Lemma*.
Since the robot should not move too fast, for each task we decided to saturate these reference velocities to 0.2.

Table 1: Hierarchy Table Vehicle Control Objective

Task	Type	$\mathcal{A}_1$
Vehicle Attitude Control Task	E	1
Vehicle Position Control Task	E	2

1.1.2 Q2: What is the behaviour if the Horizontal Attitude is enabled or not? Try changing the initial or target orientation in terms of roll and pitch angles. Discuss the behaviour.

The Horizontal Attitude control task goal is to align the z-axis of the vehicle with the z-axis of the world frame to set the vehicle parallel to the sea floor.

The horizontal attitude may be considered an operational prerequisite for the execution of tasks, but in our case we considered it as a safety task since we want to be sure that, when the robot needs to go very near to the sea floor, it doesn't get stuck in the sand. For this reason we give this control task a high priority.

To implement this action we first computed the projection of the z-axis of the world frame on the vehicle frame. Then, to compute the misalignment between the z-axis of the world and the z-axis of the vehicle, we used the Vector Lemma (implemented by the given function *ReducedVersorLemma* that outputs the misalignment ρ).

Since the task depends on one variable only (the angle that measures the misalignment) the **Jacobian** is a [1 x 13] matrix where only the velocity omega_x of the vehicle frame is responsible to drive the misalignment to zero. In formulas:

$$J = \begin{bmatrix} 0 & 0 & 0 & [...] & 0 & \rho' \end{bmatrix} \tag{11}$$

The **activation function** is an increasing bell shaped function that varies with the norm of ρ, and the **task reference** is given by:

$$\dot{\tilde{x}}_{horiz\_att} = k(0 - |\rho|) \tag{12}$$

Now we can go back to the comparison between the having and not having the horizontal attitude task enabled.

Firstly, we tried disabling the horizontal attitude task and tested results with three goals having different angles.

- Case 1 (Fig. 2): R-P-Y= [0 0 0]:
  Having the horizontal attitude disabled, the vehicle reaches the goal, as we

can notice in Figure 3. The first three figures show the evolution of the x-y-z coordinates of the robot (**expressed in m**) over time (**secs**). Instead, the last tree plots show the behavior of the angles (**expressed in rad**) over time (**secs**), and we can notice that the target is properly reached.

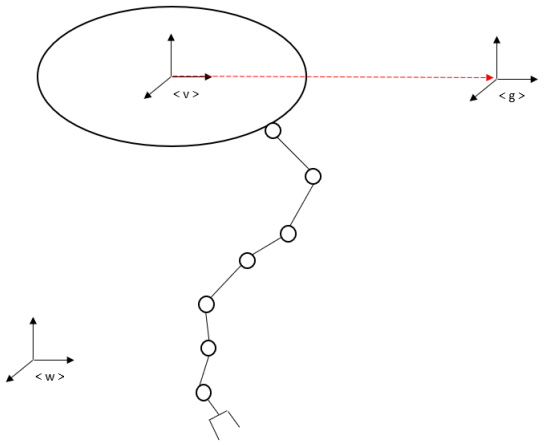

Figure 2: Goal frame with R-P-Y goal=[0 0 0] wrt vehicle frame

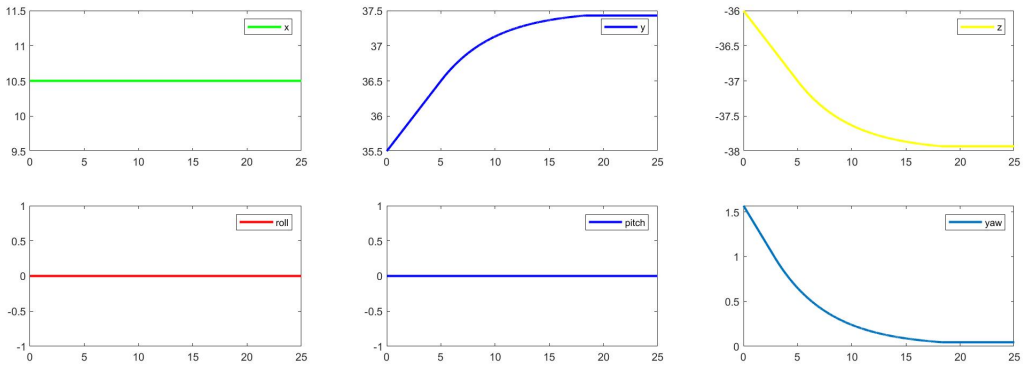

Figure 3: HA disabled: Target position R-P-Y= [0 0 0]

- Case 2 (Fig. 4: R-P-Y= [pi/4 0 0]:
  The results are shown in Figure 5. As in the previous case, the vehicle is able to properly reach its goal position.

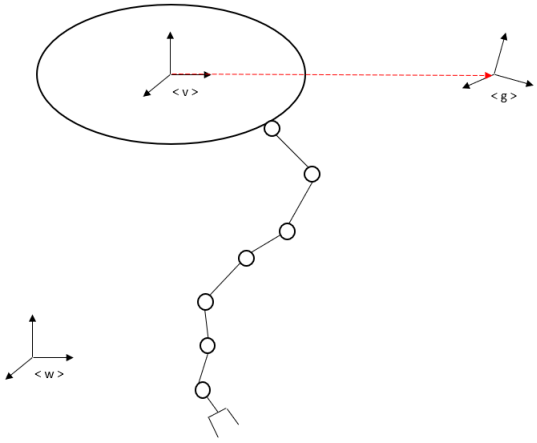

Figure 4: Goal frame rotated wrt vehicle frame

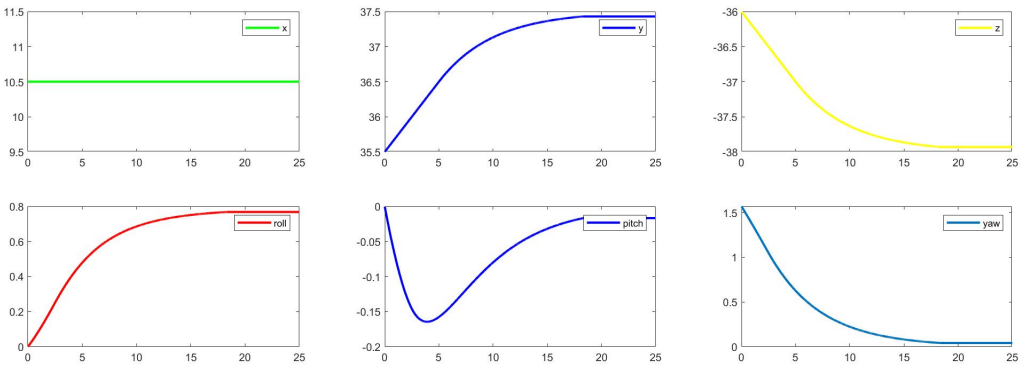

Figure 5: HA disabled. Target: R-P-Y= [pi/4 0 0]

- Case 3: R-P-Y= [0 pi/4 0]:
  The results are shown in Figure 6. The vehicle is capable of reaching its goal as in the previous cases. Therefore, we can conclude by stating that, having HA task disabled and changing the R-P-Y angles for the target, the vehicle always achieves its target goal.

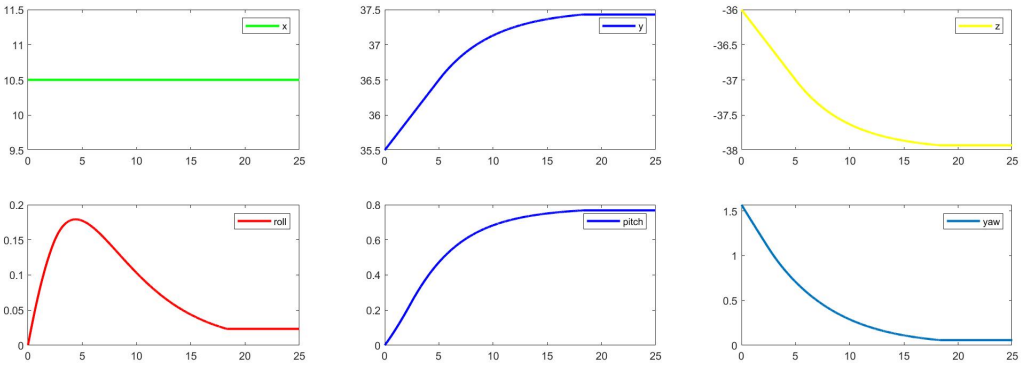

Figure 6: HA disabled. Target position R-P-Y= [0 pi/4 0]

In the following we instead simulate the case in which the three target goals are the same as before, but the HA task is enabled and has higher priority than both the vehicle angular attitude task and vehicle linear position task.

- Case 1: R-P-Y= [0 0 0]:
  In this simulation, the action is concluded successfully since the action defining objectives are satisfied (the target is reached by always keeping an horizontal attitude of the vehicle). The results are shown in Figure 7.

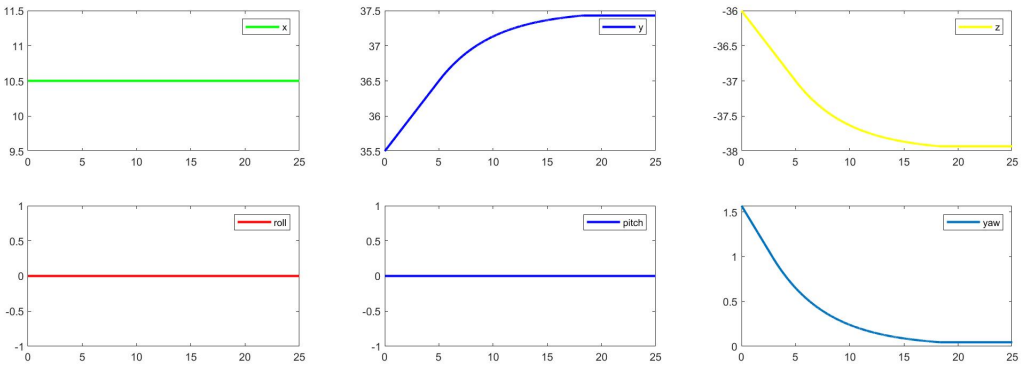

Figure 7: HA enabled. Target position R-P-Y= [0 0 0]

- Case 2: R-P-Y = [pi/4 0 0]:
  In this simulation, the action has not been completed since the roll angle of the vehicle does not tend to 0.8 (as we can see in Figure 8). This is due to the fact that the two tasks have *conflicting objectives* and, since the HA is a safety task, it has higher priority, and thus the Vehicle Angular Attitude objective has not been satisfied.

  Furthermore, the necessary condition to guarantee that an action can be completed is the existence of a non-void set of configurations where all high priority and action defining objectives are simultaneously satisfied. In this case, as stated before, it is not possible.

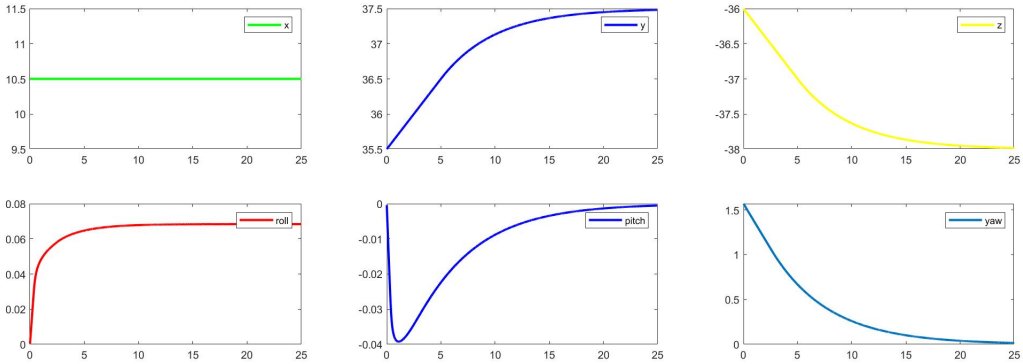

Figure 8: HA enabled. Target position R-P-Y= [pi/4 0 0]

- Case 2: R-P-Y = [0 pi/4 0]:
  In this simulation we have the same results as in the previous one. The necessary condition for completing the action successfully, stated in the previous case, is not met, therefore the pitch angle does not achieve the target position as we can see in the Figure 9.

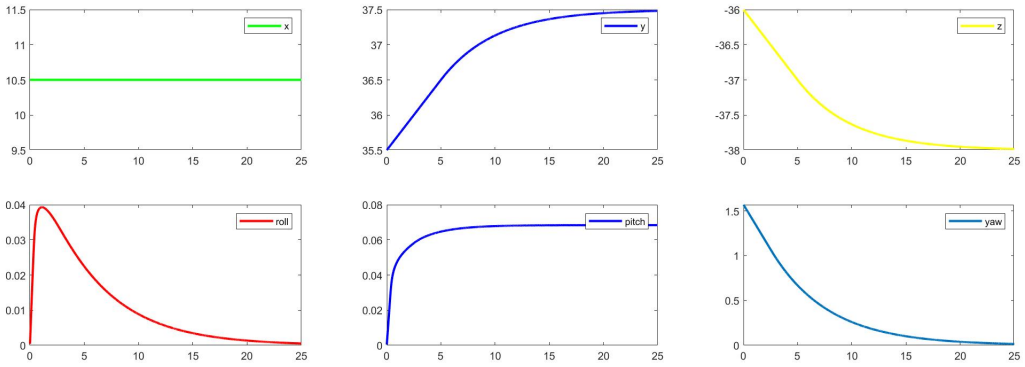

Figure 9: HA enabled. Target Position R-P-Y= [0 pi/4 0].

1.1.3 Q3: Swap the priorities between Horizontal Attitude and the Vehicle Position control task. Discuss the behaviour.

We consider the behaviour of our system when swapping priorities. The goal we set is:

$$\begin{bmatrix} 10.5 & 37.5 & -38 & 0 & \pi/4 & 0 \end{bmatrix}^\top$$

And the initial position is the same as in the exercise 1.1.1.

As we can see in Fig. 10 and Fig. 11, if we set the vehicle position control task with the highest priority, the robot achieves the goal position but doesn't keep the horizontal attitude since the latter has a lower priority and no degrees of freedom to act on. (The roll and pitch angles are already controlled by the attitude position control task).

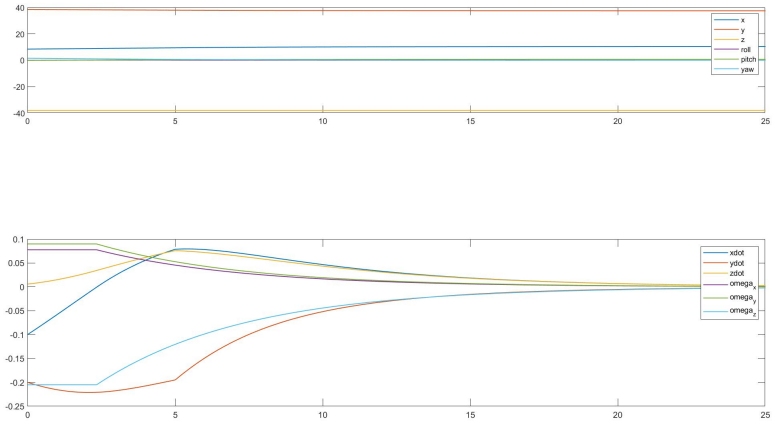

Figure 10: Vehicle position (above): x-y-z are expressed in meters while R-P-Y are expressed in radians, both with respect to the time, expressed in secs.
Vehicle velocities (below): linear velocities are expressed in m/s while angular velocities in rad/s.

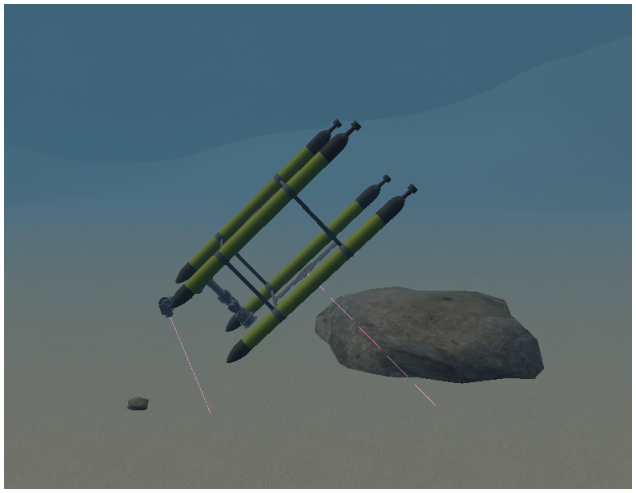

Figure 11: Final Configuration of the UVM with *horizontal attitude* with a lower priority

Therefore, we have to set the Horizontal Attitude Control Task with the highest priority to avoid this situation and to keep the UVM always parallel to the sea floor. The list of tasks of this action are ordered according to the priorities in Table 2:

Table 2: Hierarchy Table for the "Safe Waypoint Navigation"

Task	Type	$\mathcal{A}_1$
Horizontal Attitude Control Task	I	1
Vehicle Attitude Control Task	E	2
Vehicle Position Control Task	E	3

1.1.4 Q4: What is the behaviour if the Tool Position control task is active and what if it is disabled? Which of the settings should be used for a Safe Waypoint Navigation action? Report the final hierarchy of tasks (and their priorities, see the template table in the introduction) which makes up the Safe Waypoint Navigation action.

In order to control the tool of the UVM we added another task. If the Tool Position control task is activated and the Vehicle Position control Task is disabled, it is only the tool that tries to reach the goal (provided that this is within its range of action), while the vehicle stays still.
Instead, if both the Tool position control task and the Vehicle Position Control Task are activated, then the goal position is reached by both the tool and vehicle. Therefore the end effector's final position coincides with the center of the vehicle (as we can see in the Fig. 12). Of course this behavior is completely inappropriate.
Hence, the hierarchy table for "Safe Waypoint Navigation" remains the same as the one shown in Table 8.

Figure 12: Final Configuration UVM with Tool and Vehicle Position control Task

1.2 Adding a safety minimum altitude control objective

Initialize the vehicle at the position:

$$\begin{bmatrix} 48.5 & 11.5 & -33 & 0 & 0 & -\pi/2 \end{bmatrix}^\top$$

Choose as target point for the vehicle position the following one:

$$\begin{bmatrix} 50 & -12.5 & -33 & 0 & 0 & -\pi/2 \end{bmatrix}^\top$$

Goal: Implement a task to control the altitude from the seafloor. Check that at all times the minimum distance from the seafloor is guaranteed.

1.2.1 Q1: Report the new hierarchy of tasks of the Safe Waypoint Navigation and their priorities. Comment how you choose the priority level for the minimum altitude.

The list of tasks for each action are ordered according to the priorities in table 3: The Minimum altitude task should be put first since it's a safety task. The vehicle

Table 3: Hierarchy Table with a safety minimum altitude control objective

Task	Type	$\mathcal{A}_1$
Minimum Altitude control Task	I	1
Horizontal Attitude Control Task	I	2
Vehicle Attitude Control Task	E	3
Vehicle Position Control Task	E	4

horizontal attitude and the vehicle minimum altitude tasks do not interfere with each other (since the first controls the angular velocity only, while the second one controls the linear part only). However, we prefer to keep the minimum altitude task first since we think that it is more important, in terms of safety, with respect to the horizontal attitude.

1.2.2 Q2: What is the Jacobian relationship for the Minimum Altitude control task? Report the formula for the desired task reference generation, and the activation thresholds.

The **Jacobian** regards one variable only, which is the linear velocity on z:

$$J_{min\_alt} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} * J_v \tag{13}$$

Where J_v indicates the complete vehicle Jacobian matrix. The **activation function** is a decreasing bell shaped one that starts decreasing at 1 and becomes zero after a delta of 0.5.
In order to implement the safety minimum altitude control task (velocity along the z direction), we compute the altitude according to the sensor measurement along the k versor. The **reference rate** for the task is calculated as:

$$\dot{\tilde{x}}_{min\_alt} = k(max\_dist - altitude) \tag{14}$$

Where we set the *max_dist* as $hmin + \Delta h$ (minimum altitude + a delta quantity) and *hmin* as the minimum distance to maintain from the seafloor ($hmin = 0.5$). The value of the gain is due to the fact that it should be chosen higher than the positioning task since we want to be sure that the UVM reacts quickly in case of emergency. Moreover, the altitude exact computation and a detailed explanation can be found in 15.

1.2.3 Q3: Try imposing a minimum altitude of 1, 5, 10 m respectively. What is the behaviour? Does the vehicle reach its final goal in all cases?

The task has the highest priority, so the vehicle cannot move under the minimum altitude imposed. By doing some trials while imposing different minimum altitude values (such as 1m in Figure 13, 5m in Figure 14, 10m in Figure 15), we can deduce that the vehicle cannot always reach the goal position, especially when the minimum altitude is very high (as 10 mt for instance). In particular, by looking at the third plot in each image, i.e. the plot of the z position, it is evident that the vehicle can reach the goal only if it has a minimum altitude value of 1.

Furthermore, setting a too much high value for the minimum altitude does not make much sense, since the aim of our vehicle is to be close to the seafloor to allow the end-effector to grasp something.

1.2.4 Q4: How was the sensor distance processed to obtain the altitude measurement? Does it work in all cases or some underlying assumptions are implicitly made?

To compute the altitude, the first step has been to take into account the sensor measurement in a proper way. If we just consider the raw value (i.e. the distance vehicle-seafloor) and the vehicle is inclined, the measurement looses meaning because what we want is the normal component of the distance from the seafloor (see Figure 16).

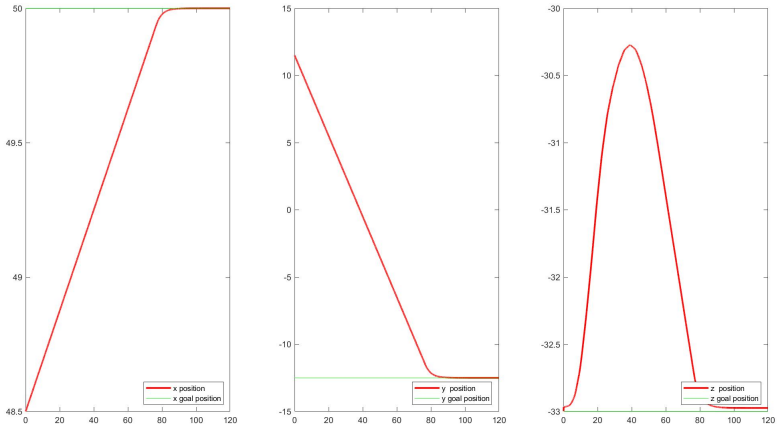

Figure 13: Vehicle actual and goal position (expressed in meters) wrt time (secs), min_alt=1

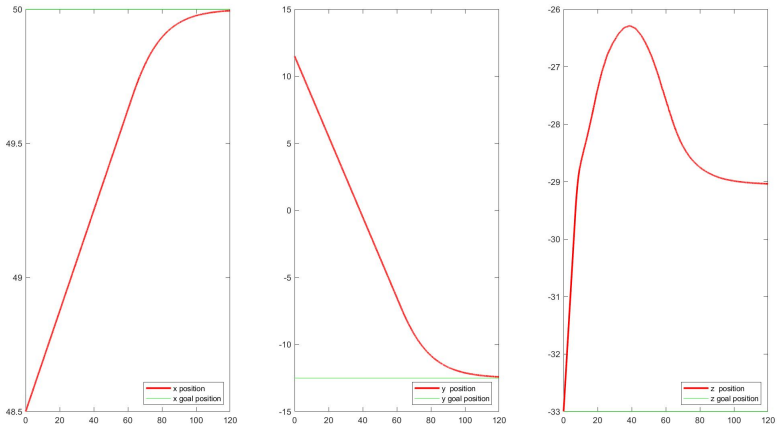

Figure 14: Vehicle actual and goal position(expressed in meters) wrt time (secs), min_alt=5

So we need to consider the distance along the z-axis in the world frame. To do so we project the distance that we find with the sensor and take the z component only:

$$altitude = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} *^{v} R_{w} * \begin{bmatrix} 0 \\ 0 \\ MeasuredDist \end{bmatrix} \tag{15}$$

The problem is that, if the vehicle is inclined and the seafloor is not flat and smooth

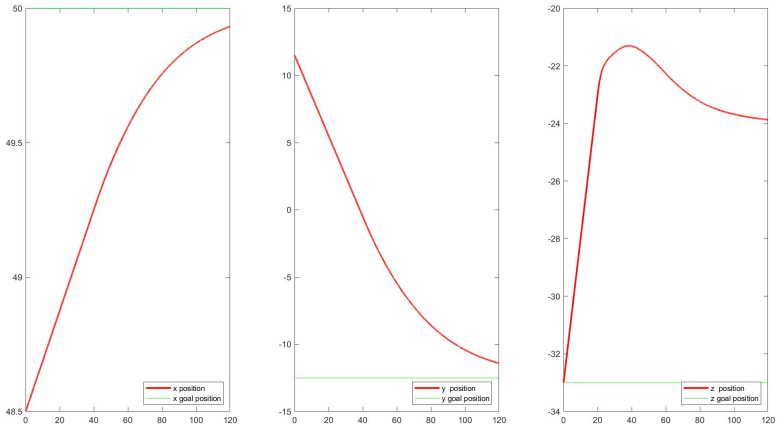

Figure 15: Vehicle actual and goal position(expressed in meters) wrt time (secs), min_alt=10

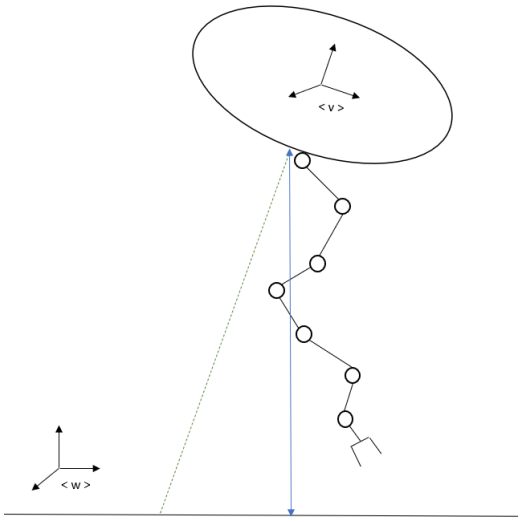

Figure 16: uvminclined, flat seafloor

this looses meaning again (see Figure 17).

This problem will always be true, and the simplest solution is to keep the horizontal attitude task active at all times. In this way we can be sure to measure the actual distance from the seafloor.

The final reference velocity is given by the following formula (which is equivalent

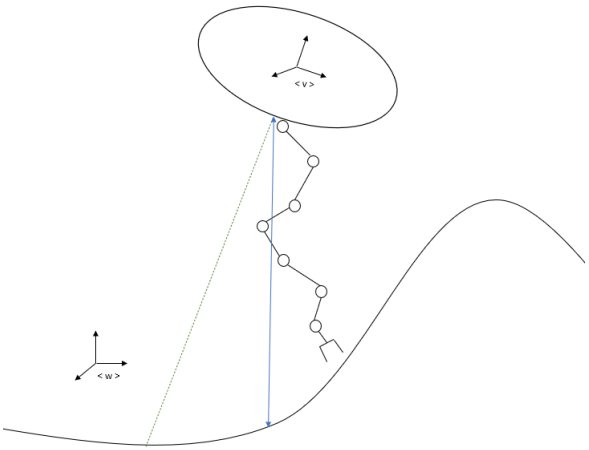

Figure 17: UVM inclined, hill on seafloor

to 14):

$$\dot{\tilde{x}}_{min-alt} = k((min\_altitude + \delta) - dist\_sensor) \tag{16}$$

where *min_altitude* is 1 (function starts decreasing) and δ is 0.5 (delta after which the function is zero).

2 Exercise 2: Implement a Basic Landing Action.

2.1 Adding an altitude control objective

Initialize the vehicle at the position:

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

Goal: add a control task to regulate the altitude to zero.

2.1.1 Q1: Report the hierarchy of tasks used and their priorities to implement the Landing Action. Comment how you choose the priority level for the altitude control task.

The new action allows the UVM to land on the seafloor. The total task hierarchy table is the one described in Table 4:

Table 4: Hierarchy Table for Landing Action

Task	Type	$\mathcal{A}_1$
Horizontal Attitude Control Task	I	1
Vehicle Landing Control Task	E	2

For this action we chose to set the horizontal attitude at first position since it is a safety task that prevents the vehicle from landing in a wrong position that could damage it. The landing task is the only other task in the action but has a lower priority with respect to horizontal attitude. Of course, since we need to perform the landing action, the minimum altitude task does not appear in the task hierarchy.

2.1.2 Q2: What is the Jacobian relationship for the Altitude control task? How was the task reference computed?

The aim of this action is to allow the robot to land on the seafloor, so the motion is just a linear velocity along z axis. Upon this consideration, the **Jacobian** relationship used to implement the new task is the same as the one used for the minimum altitude control task (See Formula 13). However, note that this task differs from the minimum altitude one since it is an equality task.
Since the goal is to regulate the altitude to zero, the **reference task** has been implemented as follows:

$$\dot{\bar{x}}_{alt} = k(0.1 - altitude) \tag{17}$$

Where k=0.2.

The **activation function** is just an identity, since this task is activated upon request.

After activation, the velocity of the vehicle becomes zero when the distance measured by the sensor becomes 0.1 (which means that the robot landed). The choice of 0.1 as threshold to indicate arrival (instead of 0), is due to the fact that the sensor is fixed slightly above the entire structure. If we let the distance to become equal to zero the back part of UVM gets stuck in the sand. In a real scenario we could turn off the engines when very near to the sea floor and let the UVM land by gravity, provided that the currents allow it. Figure 18 shows the result with threshold = 0, while 19 shows the result with threshold = 0.1.

In Figure 20 we can see that the only vehicle variable that changes is the linear z position that, starting from -38 m, arrives at -39.5 m approximately (i.e. seafloor level).

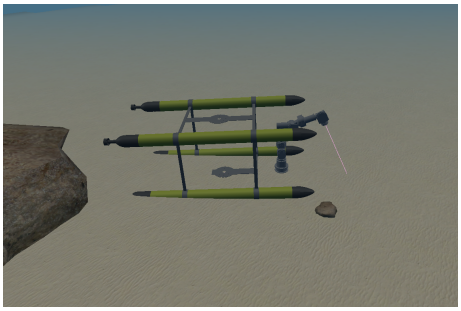

Figure 18: UVM final pose, coeff = 0

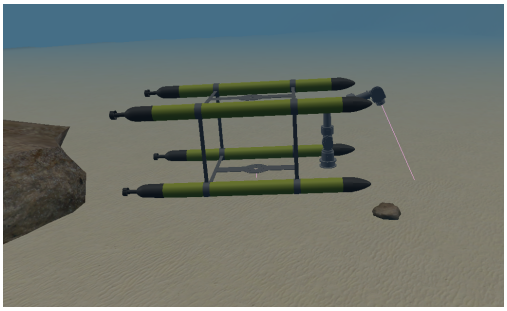

Figure 19: UVM final pose, coeff = 0.1

2.1.3 Q3: how does this task differ from a minimum altitude control task?

The Jacobian matrix of the two tasks is the same since both of them regulate the motion along the z axis of the vehicle. However, the two tasks are quite different.
First of all, the minimum altitude control task is a safety task: this means that it is an inequality task and thus it is activated only under some conditions (i.e. its activation function is a decreasing bell shaped function depending on the altitude of the vehicle). The Landing Control Task is instead an equality task and its objective is an action-defining: it is activated when it is required (i.e. its activation function is an identity matrix) and only its deactivation depends on the altitude.

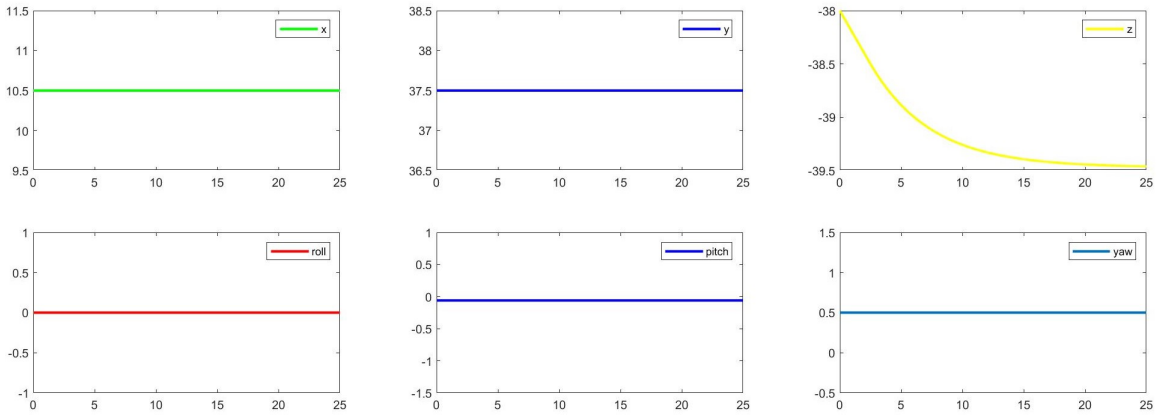

Figure 20: Vehicle Positions (meters for x-y-z, rad for R-P-Y) over time (secs)

2.2 Adding mission phases and change of action

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:

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

When the position has been reached, land on the seafloor using the basic "landing" action.

2.2.1 Q1: Report the unified hierarchy of tasks used and their priorities.

The list of tasks for each action are ordered according to the priorities in table 5:

Table 5: Overall Hierarchy Table adding the Landing Action

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$
Minimum Altitude control Task	I	1	
Horizontal Attitude Control Task	I	2	1
Vehicle Attitude Control Task	E	3	
Vehicle Position Control Task	E	4	
Vehicle Landing Control Task	E		2

Where A1 corresponds to the Safe Waypoint Navigation Action of table 3, and A2 corresponds to the Basic Landing Action of table 4.

2.2.2 Q2: How did you implement the transition from one action to the other?

The transition from an action to another is implemented (in the *UpdateMissionPhase(uvms, mission)*) by imposing the following condition: we compute the cartesian error between the vehicle actual position and the vehicle goal position (wrt world frame) in both linear and angular terms. When these values are both below a threshold (which means that the goal has been achieved), the action switches from A1 to A2.

3 Exercise 3: Improve the Landing Action

3.1 Adding an alignment to target control objective

If we use the landing action, there is no guarantee that we land in from of the nodule/rock. We need to add additional constraints to make the vehicle face the nodule. The position of the rock is contained in the variable `rock_center`*. 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.
Goal: Add an alignment task between the longitudinal axis of the vehicle (x axis) and the nodule target. 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.

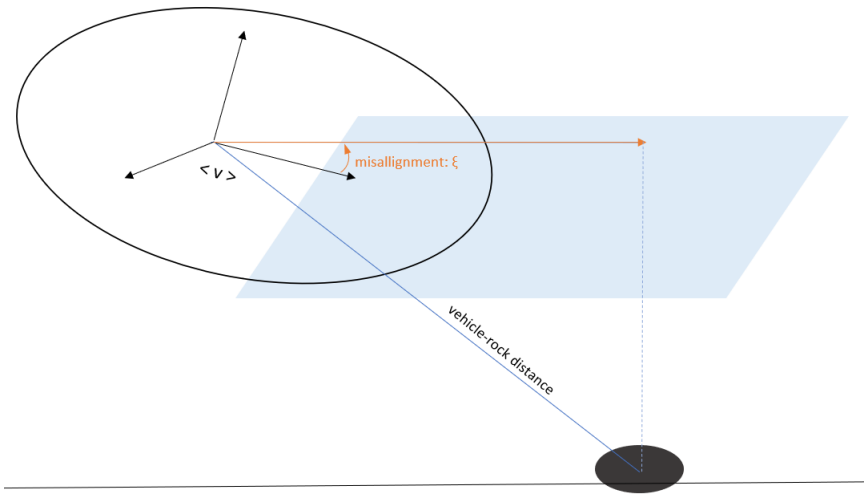

Figure 21: Axis alignment

3.1.1 Q1: Report the hierarchy of tasks used and their priorities in each action. Comment the behaviour.

In order to guarantee that the vehicle lands facing the rock, an additional constraint is required. To do that, we decided to implement another action to align the vehicle

before performing the landing and we execute the two actions in two different phases to avoid conflicts and to avoid the case in which, having an alignment velocity slower than the landing velocity, the robot rotates while it is on the seafloor (a very dangerous situation that should be avoided).
The hierarchy table is the following:

Table 6: Overall Hierarchy Table adding the Vehicle/Rock Alignment Action

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
Minimum altitude Control Task	I	1	1	
Horizontal Attitude Control Task	I	2	2	1
Vehicle Attitude Control Task	E	3		
Vehicle Position Control Task	E	4	3	
Vehicle x-axis alignment	E		4	2
Vehicle Landing Control Task	E			3

Where A1 corresponds to the Safe Waypoint Navigation Action of table 3, A2 corresponds to the newly implemented Vehicle/rock Alignment Action and A3 corresponds to the Basic Landing Action of table 4.

3.1.2 Q2: What is the Jacobian relationship for the Alignment to Target control task? How was the task reference computed?

The implementation of this task is very similar to the one of the horizontal attitude task, since in both cases the goal is to align one axis of the vehicle frame to a given direction. Since the task consists in zeroing the misalignment vector ξ, it is an *Equality Reactive Control Task*. The misalignment is between the projection, on the x-y plane of the vehicle, of the vehicle-rock distance. In formulas, the misalignment between:

$$ {}^{v}rock\_center = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} *^{v} T_w * \begin{bmatrix} rock\_center \\ 1 \end{bmatrix} \tag{18} $$

where

$$ {}^{v}T_w = \begin{bmatrix} {}^{v}R_w & {}^{v}O_w \\ zeros(1,3) & 1 \end{bmatrix} \tag{19} $$

which is the *Transformation Matrix* of the frame w with respect to frame v.

and:

$$^{v}i\_v = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{20}$$

The misalignment ξ is computed with the reduced versor lemma. Now we can compute the **Jacobian** matrix (1x13):

$$J = \begin{bmatrix} 0 & 0 & 0 & [...] & 0 & [\xi/||\xi||]^{\top} \end{bmatrix} \tag{21}$$

While the **task reference** will be:

$$\dot{\tilde{x}}_{x\_alignment} = k(0 - ||\xi||) \tag{22}$$

The **activation function** is an increasing bell shaped function that depends on the value of ξ.

3.1.3 Q3: Try changing the gain of the alignment task. Try at least three different values, where one is very small. What is the observed behaviour? Could you devise a solution that is gain-independent guaranteeing that the landing is accomplished aligned to the target?

Since the alignment of the vehicle/rock is an independent action respect to the landing action, even if we use different gains of the alignment task, the landing will always start when the vehicle is already aligned with the rock. Splitting these two actions avoids the situation in which, using a very small gain for the alignment (the velocity is very low), the vehicle is landed on the seafloor but it is not aligned yet.

Nevertheless, since we have considered a simulation time of 60 secs, if the gain is very low (i.e. it is 0.05), the phase switch from 2 to 3 occurs after a great amount of time (37 seconds) and the landing action does not finish within the available time (As we can see in Fig. 23).

Instead, when using an high gain (i.e. 1), the alignment is achieved quickly (5 secs) (see fig. 35). This behavior is not completely appropriate since we cannot properly control the motion.
Finally, with a gain value of 0.5 we have a good behaviour (Figures 24 and 25).

3.1.4 Q4: 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? Comment the observed behaviour. If, after landing, the nodule is not in the manipulator's workspace, how would you solve this problem to guarantee it?

If we try to move the end effector after the landing has occurred, the vehicle helps the accomplishment of the task by moving too. This is an undesired behaviour, since

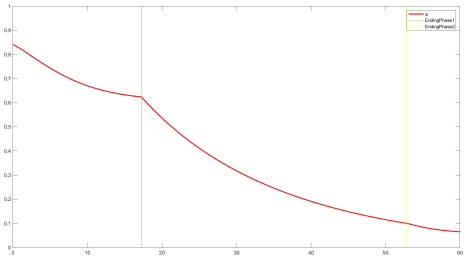

Figure 22: Misalignment (meters), Gain = 0.05

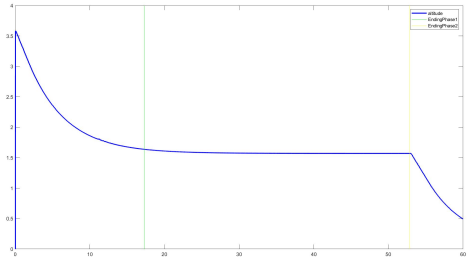

Figure 23: Altitude (meters), Gain = 0.05

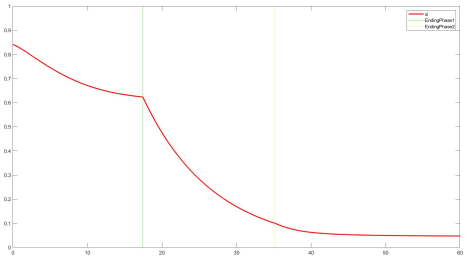

Figure 24: Misalignment (meters), Gain = 0.5

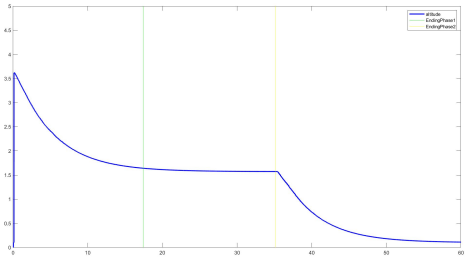

Figure 25: Altitude (meters), Gain = 0.5

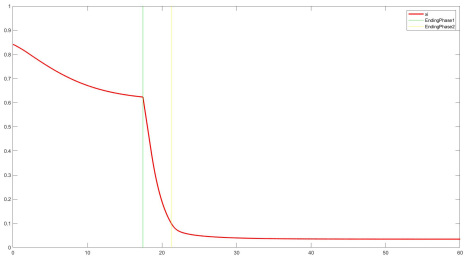

Figure 26: Misalignment (meters), Gain = 1

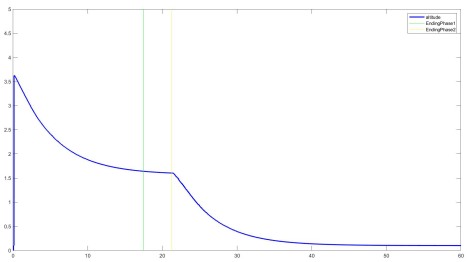

Figure 27: Altitude (meters), Gain = 1

the vehicle had already reached its own goal position.

We should therefore implement an action that blocks it and moves the end effector only. Moreover, in this case the nodule is in the manipulator's workspace, so it should be reachable by blocking the vehicle. If it weren't, we would have to either move the vehicle after landing (but we said we don't want this type of behaviour) or act before the landing itself by imposing a constraint.

The idea would be to check that the landing happens only if the nodule is reach-

able. For example we could check if the projection of the rock on the vehicle horizontal plane (see equation 18) respects a certain value $\rho < extended\_arm\_length$. This should be implemented because of two reasons:

- the arm may be constrained by the seafloor and not be able to reach the desired position;
- if the arm could move properly and set to a fully extended configuration we would have a singularity, which isn't desirable

4 Exercise 4: Implementing a Fixed-base Manipulation Action

4.1 Adding non-reactive tasks

To manipulate as a fixed based manipulator, we need to constrain the vehicle to not move, otherwise the tool frame position task will make the vehicle move.

Goal: Add a constraint task that fixes the vehicle velocity to zero. Land on the seafloor. Try reaching the rock position with the end-effector, and observe that the vehicle does not move.

4.1.1 Q1: Report the hierarchy of tasks used and their priorities in each action. At which priority level did you add the constraint task?

Table 7: Overall Hierarchy Table adding the Fixed-base Manipulation Action

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
Minimum altitude Control Task	I	1	1		
Horizontal Attitude Control Task	I	2	2	1	1
Vehicle Attitude Control Task	E	3			
Vehicle Position Control Task	E	4	3		
Vehicle x-axis Alignment Control Task	E		4	2	
Vehicle Landing Control Task	E			3	
Vehicle Null Velocities Control Task	E				2
Tool Positioning Control Task	E				3

Where A1 corresponds to the Safe Waypoint Navigation Action of table 3, A2 corresponds to the Vehicle/rock Alignment action of table 6, A3 corresponds to the Basic Landing Action of table 4 and A4 corresponds to the Fixed-Base Manipulation Action.

4.1.2 Q2: What is the Jacobian relationship for the Vehicle Null Velocity task? How was the task reference computed?

Since we are considering a task which involves all the DOFs related to the vehicle, the **Jacobian** is a [6x13] matrix, expressed as follows:

$$J\_null = \begin{bmatrix} J\_v\_lin \\ J\_v\_ang \end{bmatrix} \tag{23}$$

where the two matrices correspond to 8 and 9.
Since the task is non-reactive and we want all vehicle velocities to be zero, the **task reference** is constant and equal to :

$$\dot{\bar{x}}_{null} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top \tag{24}$$

Finally, the **activation function** is of course an identity matrix.

4.1.3 Q3: Suppose that the vehicle is floating, i.e. not landed on the seafloor. What would happen if, due to currents, the vehicle moves?

The task we implemented simply sets the vehicle velocities to zero. Therefore it does not compensate for a possible disturbance due to the currents.
To test what would happen with currents we added a disturbance on x, y, z axes then transformed the disturbance on the vehicle frame to have it on p_dot. In this case, even though the vehicle is able to change actions (since we are considering a relatively low disturbance), the Vehicle Null Velocities Control Task does not cancel it.

As a matter of fact, in the next exercises we decided to change the activation function of this control task in such a way that, if the vehicle moves too far away from the goal position due to currents, the null velocity task is deactivated and the uvmis given the possibility to get closer to the goal again.

Figures 28 and 29 represent the case without disturbances.

Figures 30 and 31 represent the case with disturbances. The tool cannot keep the goal position and oscillates about it.

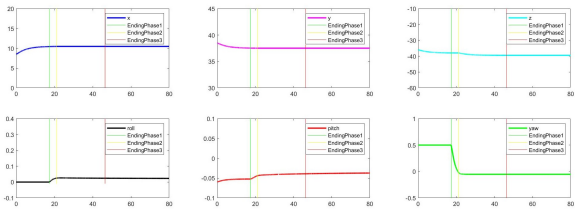

Figure 28: Vehicle coordinates (meters for x-y-z, rad for R-P-Y over time in secs) without disturbance

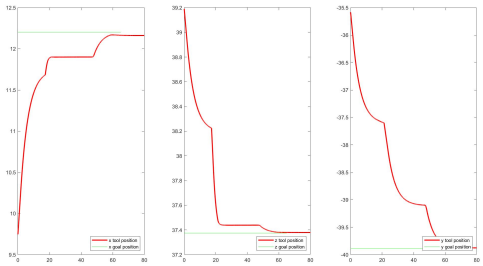

Figure 29: Tool actual positions (in meters) vs goal position over time (secs) without disturbances

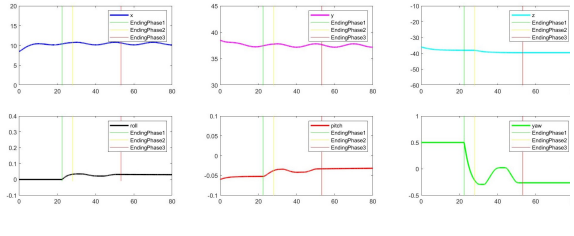

Figure 30: Vehicle coordinates (meters for x-y-z, rad for R-P-Y over time in secs) with disturbance

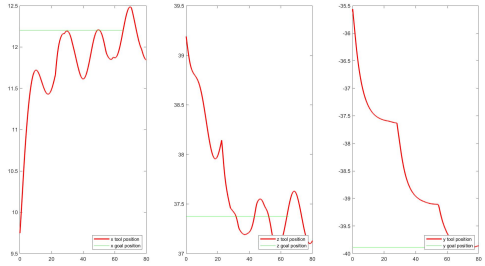

Figure 31: Tool actual positions (in meters) vs goal position over time (secs) with disturbances

4.2 Adding a joint limit task

Let us now constrain the arm with the actual joint limits. The vector variables `uvms.jlmin` *and* `uvms.jlmax` *contain the maximum and minimum values respectively.*

Goal: Add a joint limits avoidance task. Land on the seafloor. Try reaching the rock position with the end-effector, and observe that the vehicle does not move and that all the joints are within their limits.

4.2.1 Q1: Report the hierarchy of tasks used and their priorities in each action. At which priority level did you add the joint limits task?

Table 8: Hierarchy Table for the Fixed-base Manipulation Action

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
Joint Limits Control Task	I	1	1	1	1
Minimum altitude Control Task	I	2	2		
Horizontal Attitude Control Task	I	3	3	2	2
Vehicle Attitude Control Task	E	4			
Vehicle Position Control Task	E	5	4		
Vehicle x-axis Alignment Control Task	E		5	3	
Vehicle Landing Control Task	E			4	
Vehicle Null Velocities Control Task	E				3
Tool Positioning Control Task	E				4

Where A1 corresponds to the Safe Waypoint Navigation Action of table 3, A2 corresponds to the Vehicle/rock Alignment action of table 6, A3 corresponds to the

Basic Landing Action of table 4 and A4 corresponds to the Fixed-Base Manipulation Action.
We set the joint limits task within the highest priority task since it is a safety task. Moreover, we precisely set it at highest priority since it even has more importance than the other safety tasks. The robotic arm is probably weaker and more expensive than the vehicle and thus must be checked first. In addition, it is important to define the limits for the joint to avoid the singularity configurations of the arm which can damage the robot.

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

The **Jacobian** matrix is the juxtaposition of an identity matrix related to the arm joints and a zero matrix related to the vehicle. We're acting on the arm joints (7 DOFs), so the Jacobian is a [7x13] matrix.

$$J\_lin = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & ... & 0 \end{bmatrix} \tag{25}$$

While the **task reference** is:

$$\dot{\tilde{x}}_{x\_alignment} = k[(jl\_min + jl\_max)/2 - q)] \tag{26}$$

And the **activation function** that is equal to 1 for $q < q\_min$ and for $q > q\_max$, equal to 0 for $(q\_min + 0.1) < q < (q\_max - 0.1)$.

Results are shown in Figures 32 and 33. In particular, Figure 32 shows the tool position and the tool goal position and Figure 33 shows the q_dots and the joint limits.

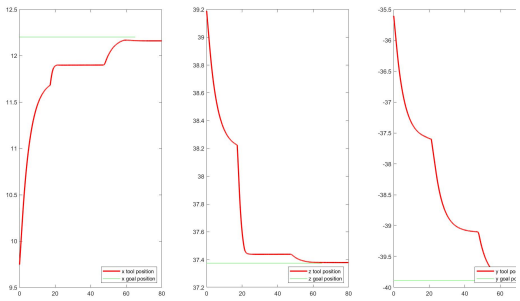

Figure 32: Tool actual positions (meters) vs goal position over time (secs)

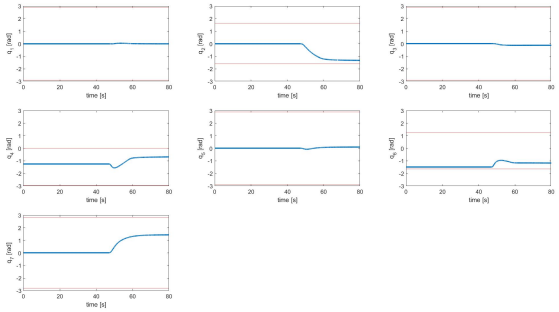

Figure 33: q_i and joint limits (expressed in rad) over time (secs)

4.3 List of control tasks of the final version of the Robust software

- **Joint Limits Control Task** is an *Inequality Reactive Safety* Task
- **Minimum altitude Control Task** is an *Inequality Reactive Safety* Task
- **Horizontal Attitude Control Task** is an *Inequality Reactive Safety* Task
- **Vehicle Attitude Control Task** is an *Equality Reactive Action-Defining* Task
- **Vehicle Position Control Task** is an *Equality Reactive Action-Defining* Task
- **Vehicle x-axis Alignment Control Task** is an *Inequality Reactive Prerequisite* Task
- **Vehicle Landing Control Task** is an *Equality Reactive Action-Defining* Task
- **Vehicle Null Velocities Control Task** is an *Equality Non-Reactive Optimization* Task
- **Tool Positioning Control Task** is an *Equality Reactive Action-Defining* Task

5 Exercise 5: Floating Manipulation

5.1 Adding an optimization control objective

Use the DexROV simulation for this exercise.

The goal is to try to optimize the joint positions, if possible, to keep the first four joints in a "preferred shape", represented by the following vector

$$\begin{bmatrix} -0.0031 & 1.2586 & 0.0128 & -1.2460 \end{bmatrix}^\top$$

Goal: Add an optimization objective to keep the first four joints of the manipulator in the preferred shape. Observe the behaviour with and without the task

5.1.1 Q1: Report the hierarchy of tasks used and their priorities in each action. At which priority level did you add the optimization task?

Table 9: Hierarchy Table with the Preferred Configuration task

Task	Type	$\mathcal{A}_1$
Joint limits	I	1
Horizontal attitude	I	2
Tool positioning	E	3
Preferred configuration	E	4

- **Joint Limits Control Task** is an *Inequality Reactive Safety* Task
- **Minimum altitude Control Task** is an *Inequality Reactive Safety* Task
- **Horizontal Attitude Control Task** is an *Inequality Reactive Safety* Task
- **Tool Positioning Control Task** is an *Equality Reactive Action-Defining* Task
- **Preferred Configuration Control Task** is an *Inequality Reactive Optimization* Task

The "Preferred Configuration" task is added at the end of the hierarchy, with the lowest priority. This because it is only an optimization task: its objective does not influence the mission, but permits to choose between multiple solutions,in case multiple solutions exist. Therefore it is accomplished only if the other tasks have been performed and the goal has reached.

5.1.2 Q2: What is the Jacobian relationship for the Joint Preferred Shape task? How was the task reference computed?

Since we want a preferred configuration only for the first four joints, then the **Jacobian** is a [4x13] matrix and it is structured as follows:

$$J\_lin = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 1 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 1 & 0 & ... & 0 \end{bmatrix} \tag{27}$$

The **task reference** is represented by the following relationship:

$$\dot{\tilde{x}}_{preferred\_config} = -k * [q(1:4) - preferred\_config] \tag{28}$$

where the *preferred_config* has been defined above.
Since the objective is to reach the preferred configuration of the first four joints ($||q\_i|| \leq q\_preferred$), then it is an *Inequality Reactive Control Task*.

5.1.3 Q3: What is the difference between having or not having this objective?

Comparing the behaviour of the UVM in the two cases we can note that in both cases the tool reaches the goal position (since this task has higher priority than the preferred configuration one). Nevertheless, without the objective, the UVM tries to reach the goal position by stretching the arm in a singular configuration (Figure 34). This is not a desirable behaviour, and the objective solves it, as shown in Figure 35.

In figures 36 and 37 there are the plots of the joints angles with the preferred configuration task enabled and disabled respectively. In Figure 37 we can notice that q_1, q_3 and q_4 reached the preferred shape in the second case, while q_2 did not reach it. This is due to the fact that other tasks with higher priority might not allow it (conflicting objectives). However, it is completely admissible and remarks the fact that it is just an optimization task.

5.2 Adding mission phases

Let us now structure the mission in more than one phase. In the first phase, exploit the previous exercises, and implement a safe waypoint navigation. Move the vehicle to a location close to the current defined end effector goal position, just slightly above it. Then, trigger a change of action and perform floating manipulation.

Goal: introduce mission phases in the floating manipulation scenario. Observe the difference.

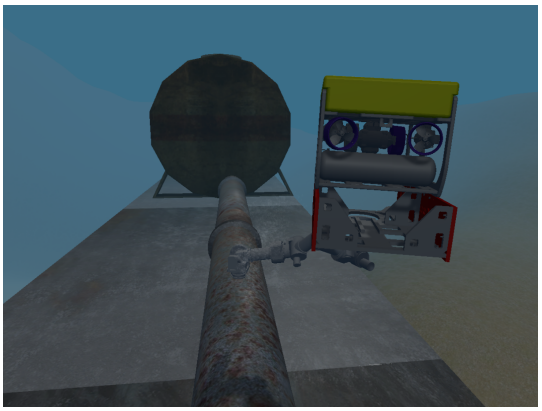

Figure 34: uvms without preferred configuration

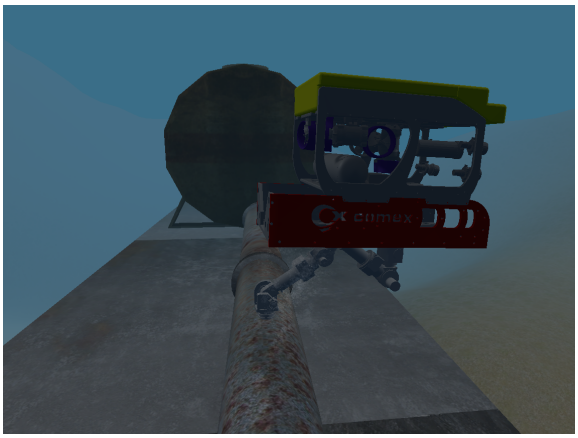

Figure 35: uvms with preferred configuration

s

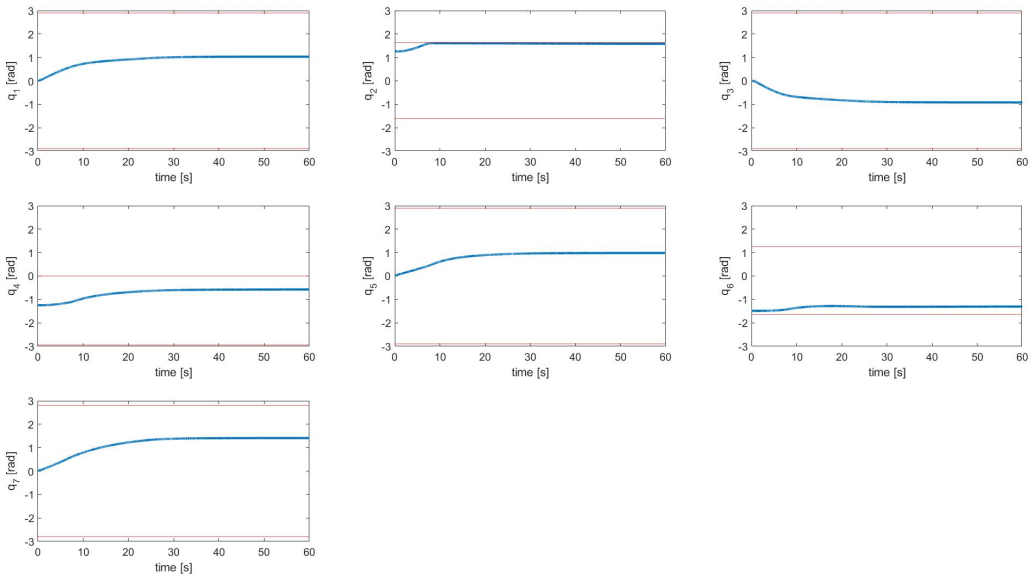

Figure 36: q without preferred configuration

5.2.1 Q1: Report the unified hierarchy of tasks used and their priorities. Which task is active in which phase/action?

The hierarchy table is shown in Table ??.

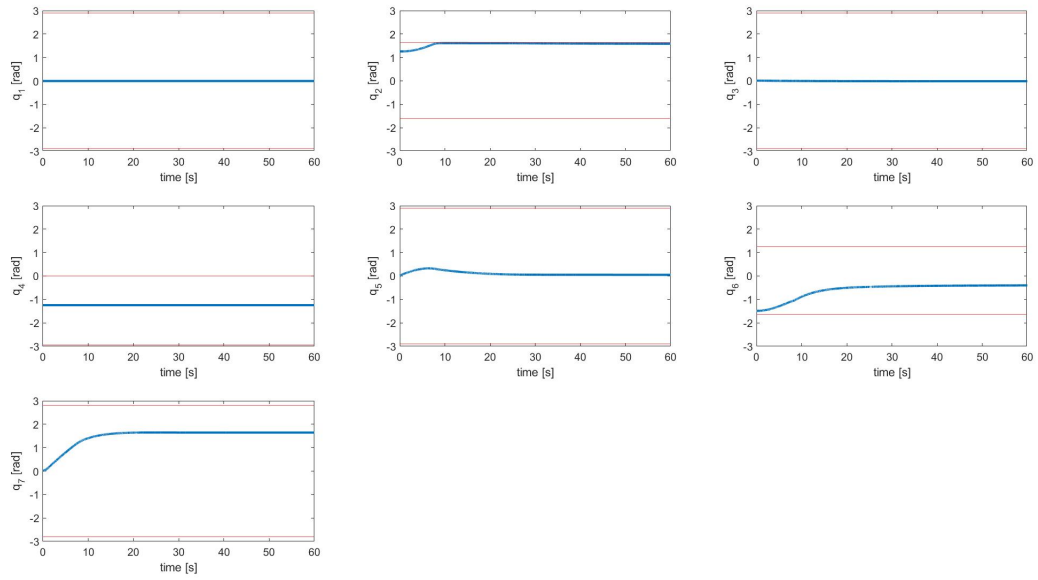

Figure 37: q with preferred configuration

In the first action, the vehicle moves and reaches the following goal position:

$$\begin{bmatrix} -2.3 & 10.1 & -29.8 & 0 & 0.06 & -\pi/2 \end{bmatrix}^\top$$

This value was chosen by *trial and error*: we ran the simulation with only one single phase by deactivating the null vehicle velocities task and keeping the tool position task enabled. We saved the position that the vehicle reached when the tool action was completed successfully and set it as vehicle goal position.
All safety tasks are kept active throughout all the phases.

5.2.2 Q2: What is the difference with the previous simulation (still in exercise 5), where only one action was used?

In the previous simulation we didn't constrain the vehicle to be fixed when moving the tool. In this way we could hypothetically reach any goal, but not in a safe way. With the second approach we can control the vehicle better (with the null velocity task) but we need to properly choose the vehicle goal position so that the tool can reach the position. This is of course a limitation and the project becomes application dependant: If you want to manipulate something *in loco* you can use the second approach. If you want to move along the pipe you should use the first approach instead.

Table 10: Overall Hierarchy Table used with the Dexrov

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$
Joint Limits Control Task	I	1	1
Minimum Altitude Control Task	I	2	2
Horizontal Attitude Control Task	I	3	3
Vehicle Attitude Control Task	E	4	
Vehicle Position Control Task	E	5	
Vehicle Null Velocities Control Task	E		4
Tool Positioning Control Task	E		5
Preferred Configuration Control Task	I	6	6

When running the simulations and trying to reach the same tool goal with the two different approaches we obtained the results of Figures 38 and 39. We can see that with the second approach we cannot reach the exact x goal position, while with the first we could. By checking the q limits we saw that q_2 saturates to try to reach the goal, but it still does not (see Figure 40). By deactivating the joint limits we could maybe reach the position, but we would decrease safety, and thus we opted for the more imprecise but safe solution.

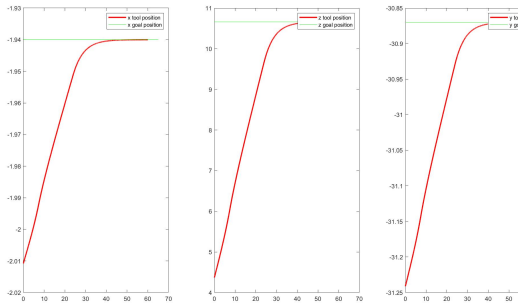

Figure 38: Tool positions (meters) using one mission phase

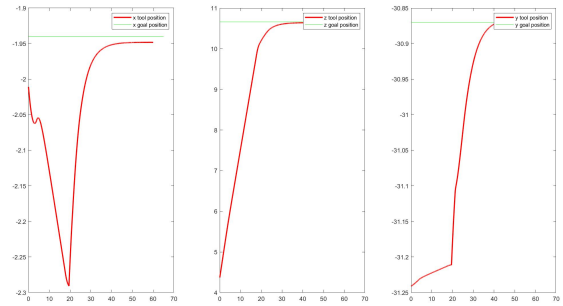

Figure 39: Tool positions (meters) using two mission phases

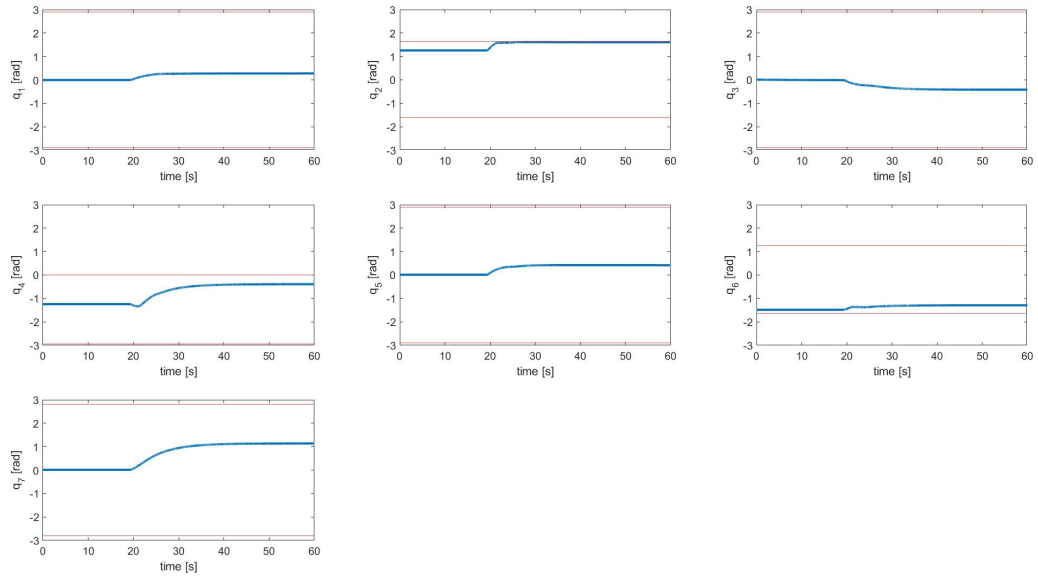

Figure 40: q_i (expressed in rad) over time (secs) using two mission phases

6 Exercise 6: Floating Manipulation with Arm-Vehicle Coordination Scheme

6.1 Adding the parallel arm-vehicle coordination scheme

Let us now see how the two different subsystems (arm and vehicle) can be properly coordinated. Introduce in the simulation a sinusoidal velocity disturbance acting on the vehicle, and assume the actual vehicle velocity measurable. To do so, add a constant (in the inertial frame) velocity vector to the reference vehicle velocity before integrating it in the simulator.

Goal: modify the control part to implement the parallel arm-vehicle coordination scheme. Observe that, even with a disturbance acting on the vehicle, the end-effector can stay in the required constant position.

6.1.1 Q1: Which tasks did you introduce to implement the parallel coordination scheme?

We based our implementation of the vehicle arm coordination on the idea of having two optimizations running in parallel (TPIK1, TPIK2), as depicted in Figure 41.

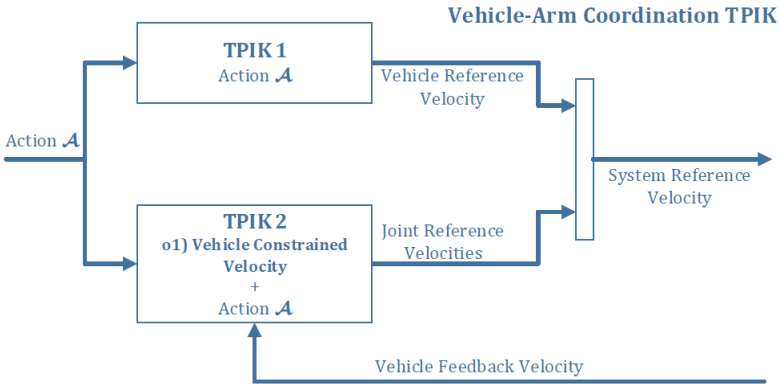

Figure 41: Two optimizations in parallel

We opted for this kind of control since, in floating manipulation operations, an important feature is to split the vehicle and arm into 2 subsystems. This prevents the propagation of errors from the vehicle to the arm when disturbances are taken into account (considering a realistic scenario). The vehicle has in fact much slower dynamics, and may corrupt the arm motion.

Given an action **A**, the subsystems are the following:

- **TPIK1:** arm and vehicle are considered together as a fully controllable system. The hierarchy of tasks corresponding to action **A** is solved but only the vehicle reference velocity is used (discarding the manipulator part).
- **TPIK2:** the vehicle is considered fully uncontrollable: its velocity is initialized with a velocity given by sensor measurements. Here a new task is added and placed at the top of the task hierarchy. It constraints the vehicle velocity to be the velocity feedback given by sensor.

The aforementioned new constraining task has a **jacobian** matrix of:

$$J\_vc = \begin{bmatrix} 0 & 0 & ... & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & ... & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & ... & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & ... & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & ... & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & ... & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{29}$$

And a **reference velocity** of:

$$\dot{\bar{x}}_{vc} = v\_measured \tag{30}$$

While its **activation function** is an identity matrix.

Thanks to this optimization, the arm joint velocities are always optimal in tracking the desired velocity generated by TPIK1.

The complete Task Hierarchy for TPIK1 is the same of the Table ??. The task hierarchy for TPIK2 is the same as the one for TPIK1, with the addiction of the Vehicle Constrained Control Task at the top.

Another difference from the previous case is that, in this implementation, the non reactive control task that sets to zero the vehicle velocity has a different activation function. In the previous exercise it was an identity matrix since we wanted an equality task. Now we consider it as an inequality task with an activation function that becomes unitary as the error between the end-effector and the goal position approaches zero. In fact, we want that, when the tool is sufficiently close to the goal, the vehicle tries to remain still. Instead, when there is a strong disturbance (i.e. sea-current) and the vehicle is shifted away from the goal, the control task is deactivated so that the vehicle is free to move towards the goal position.

To test the code we added a disturbance on the x, y, z axes and then projected

the disturbance on the vehicle frame (since we had to add it to p_dot). Results are shown in the next section.

6.1.2 Q2: Show the plot of the position of thea end-effector, showing that it is constant. Show also a plot of the velocities of the vehicle and of the arm.

We tried several experiments considering sinusoidal disturbances with different values of module, phase and direction.

Experiment1: Sinusoidal Disturbances on x-y

We use a disturbance along x and y (vehicle frame), with amplitude 0.1 and frequency 0.1 Hz. (Fig 42, Fig43, Fig 44, Fig 45).

The results of this experiment are good, since most of the noise is compensated by the correct behavior of the arm joints that move to make tool position as stable as possible. The error is mitigated enough and tends to zero.

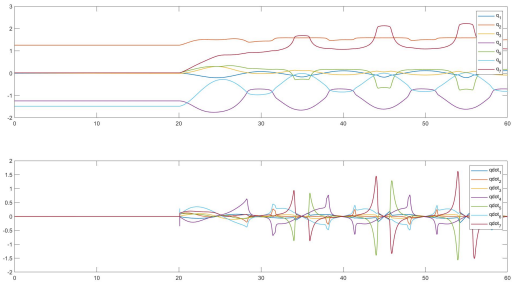

Figure 42: Vehicle positions expressed in meters (above) and velocity expressed in m/s (below), both over time (secs)

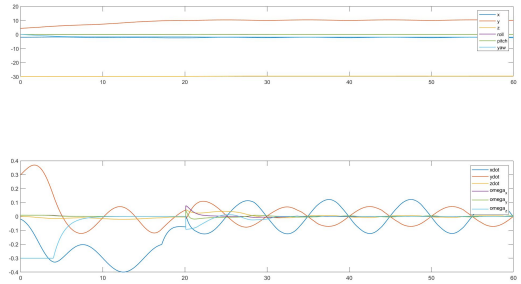

Figure 43: Joint positions expressed in rad (above) and velocity expressed in rad/s (below), both over time (secs)

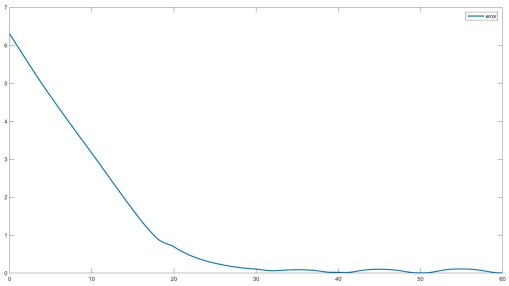

Figure 44: Error (meters) between the end-effector and the goal position in presence of disturbances over time (secs)

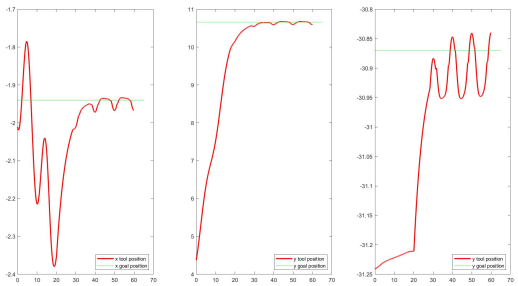

Figure 45: Tool positions and tool goal expressed in meters over time (secs)

Experiment2: Sinusoidal Disturbances on x-y

We use a disturbance along x and y (vehicle frame), with amplitude 0.1 and frequency 1 Hz. (Fig 46, Fig47, Fig 48, Fig 49).

The number of oscillations are very high as expected but the error is acceptably small since the gain remains low as well as in the previous experiment.

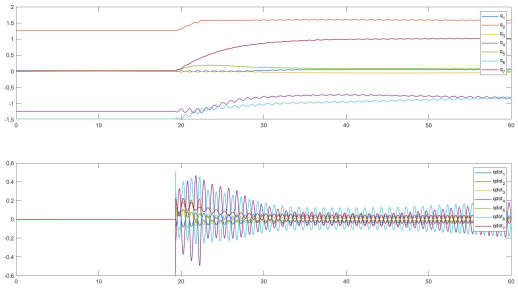

Figure 46: Vehicle positions expressed in meters (above) and velocity expressed in m/s (below), both over time (secs)

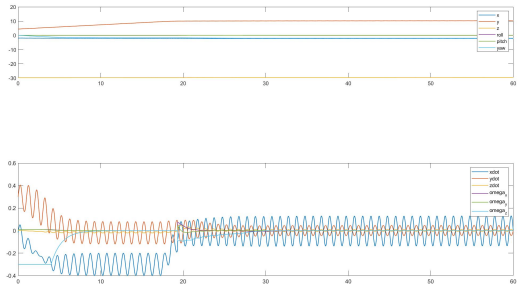

Figure 47: Joint positions expressed in rad (above) and velocity expressed in rad/s (below), both over time (secs)

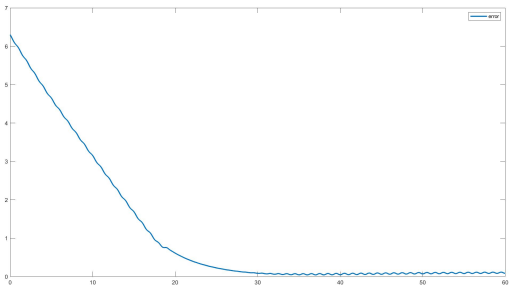

Figure 48: Error (meters) between the end-effector and the goal position in presence of disturbances over time (secs)

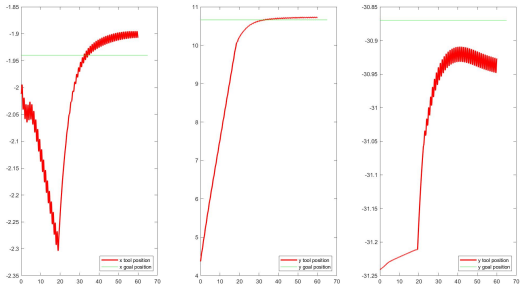

Figure 49: Tool positions and tool goal expressed in meters over time (secs)

Experiment3: Sinusoidal Disturbances on x-y

We use a disturbance along x and y (vehicle frame), with amplitude 0.05 and frequency 0.1 Hz. (Fig50, Fig51, Fig 52, Fig 53).

From the plots, we can notice that the arm joints move less to compensate the disturbance, since we set a lower gain to the disturbance. Therefore we obtained better results than the previous cases.

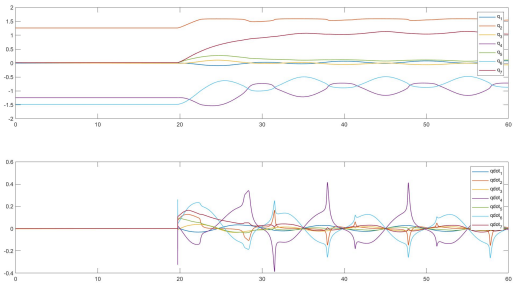

Figure 50: Vehicle positions expressed in meters (above) and velocity expressed in m/s (below), both over time (secs)

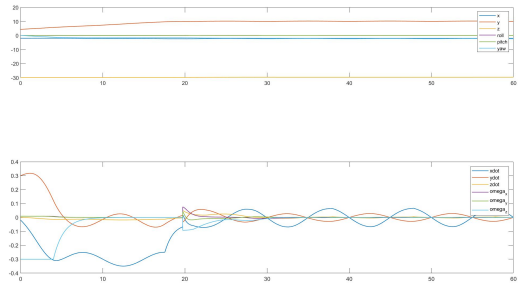

Figure 51: Joint positions expressed in rad (above) and velocity expressed in rad/s (below), both over time (secs)

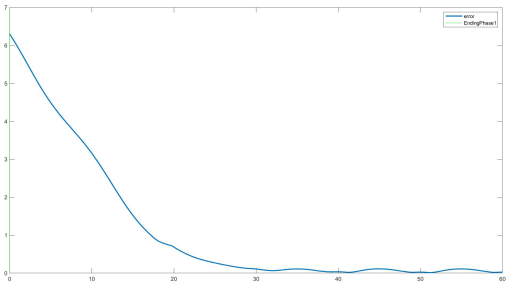

Figure 52: Error (meters) between the end-effector and the goal position in presence of disturbances over time (secs)

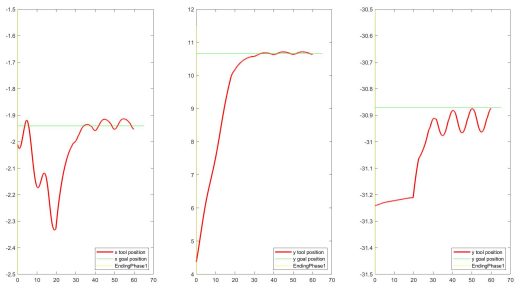

Figure 53: Tool positions and tool goal expressed in meters over time (secs)

6.1.3 Q3: What happens if the sinusoidal disturbance becomes too big? Try increasing the saturation value of the end-effector task if it is too low.

Experiment1: Sinusoidal Disturbances on x-y
We use a disturbance along x and y (vehicle frame), with amplitude 0.3 and frequency 0.1 Hz. (Fig54, Fig55, Fig 56, Fig 57).
As we can see, the error between the end-effector and the pipe is very significant and cannot be ignored. Using a very high value of the gain of the disturbance, the arm goes out of its workspace and consequently the position of the tool does not remain in the target pose.

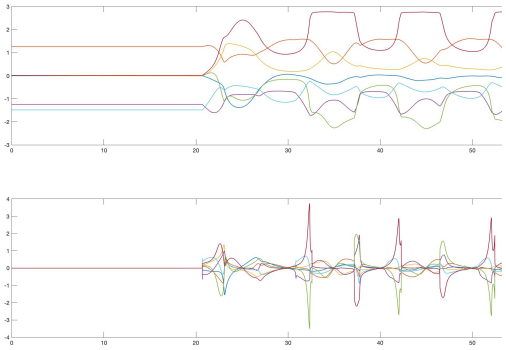

Figure 54: Vehicle positions expressed in meters (above) and velocity expressed in m/s (below), both over time (secs)

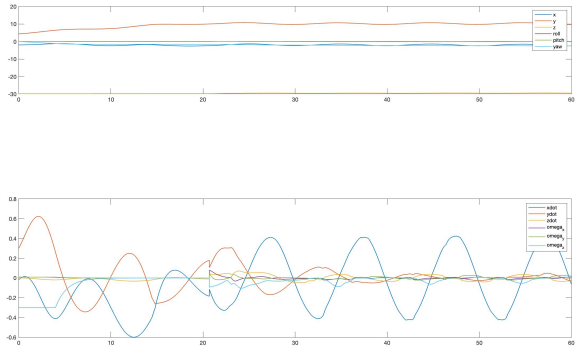

Figure 55: Joint positions expressed in rad (above) and velocity expressed in rad/s (below), both over time (secs)

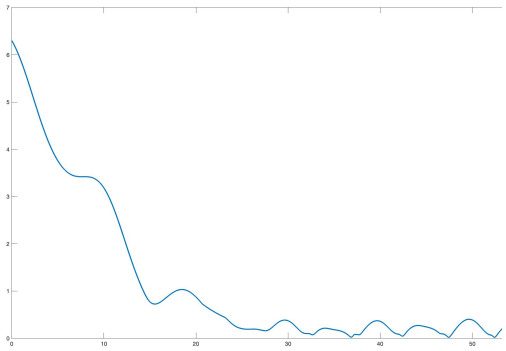

Figure 56: Error (meters) between the end-effector and the goal position in presence of disturbances over time (secs)

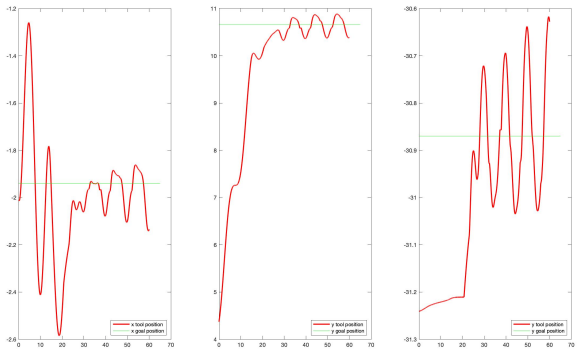

Figure 57: Tool positions and tool goal expressed in meters over time (secs)