Cooperative Robotics

Authors: Alberto Grillo, Filippo Gandolfi
EMAILs: — albogrillo@gmail.com, filippo.gandolfi95@gmail.com —
Date: 3/4/2020

General notes

- Exercises 1-4 are done with the ROBUST MATLAB main and unity visualization tools. Exercises 5-6 are done with the DexROV MATLAB main and unity visualization tools.
- Comment and discuss the simulations, in a concise scientific manner. Further comments, other than the questions, can be added, although there is no need to write 10 pages for each exercise.
- Aid the discussion with screenshots of the simulated environment (compress them to maintain a small overall file size), and graphs of the relevant variables (i.e. activation functions of inequality tasks, task variables, and so on). Graphs should always report the units of measure in both axes, and legends whenever relevant.
- Report the thresholds whenever relevant.
- Report the mathematical formula employed to derive the task jacobians and the control laws when asked, including where they are projected.
- If needed, part of the code can be inserted as a discussion reference.

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.

1.1.1 Q1: Report the hierarchy of tasks used and their priorities. What is the Jacobian relationship for the Vehicle Position control task? How was the task reference computed?

We use the following notation to characterize each task:

- R/NR, reactive or non-reactive.
- I/E, inequality or equality.
- C/S/P/AD/O, constraint, safety, prerequisite, action-defining, optimization. This characterization gives the priority to each task, constraint task will have the highest priority, optimization the lowest.

The **hierarchy of the tasks**, in order of priority:

1. Manipulability
2. Horizontal Attitude
3. Vehicle Position

Manipulability [R, I, P]: it is used to maintain arm dexterity above a certain threshold.

Horizontal Attitude [R, I, S]: it is fundamental to keep the vehicle horizontal with respect to the absolute world frame.

Vehicle Position [R, I, AD]: it is an action-defining task, therefore it has lower priority. This is the first task we implemented, the goal of this task is to compute velocities to align the vehicle frame with the desired goal frame, within a certain bound.

The **Jacobian** relationship for the *Vehicle Position* control task is the following one:

$$\boldsymbol{J}_v = \begin{bmatrix} & \underset{3\times3}{\mathbf{0}} & \underset{3\times3}{{}^w\boldsymbol{R}_v} \\ \underset{6\times7}{\mathbf{0}} & & \\ & \underset{3\times3}{{}^w\boldsymbol{R}_v} & \underset{3\times3}{\mathbf{0}} \end{bmatrix} \tag{1}$$

where ${}^w\boldsymbol{R}_v$ is the rotation matrix from the vehicle frame to the world frame. Notice that, in this task, we use the control variable with this convention [Roll Pitch Yaw X Y Z].
We compute the **task reference** as:

$${}^w\dot{\boldsymbol{x}}_{v\_g} = k\left({}^w\boldsymbol{x}_g - {}^w\boldsymbol{x}_v\right) \tag{2}$$

where:

- We compute the Cartesian error between the vehicle and the goal frame, both projected on the world frame:

$$^{w}\boldsymbol{x}_g - {}^{w}\boldsymbol{x}_v = [\boldsymbol{\rho}, e_x, e_y, e_z]^\top \tag{3}$$

  $\boldsymbol{\rho}$ is the misalignment vector obtained with the Versor Lemma, while e_x, e_y, e_z are the components of the linear error.

- k is the control gain.

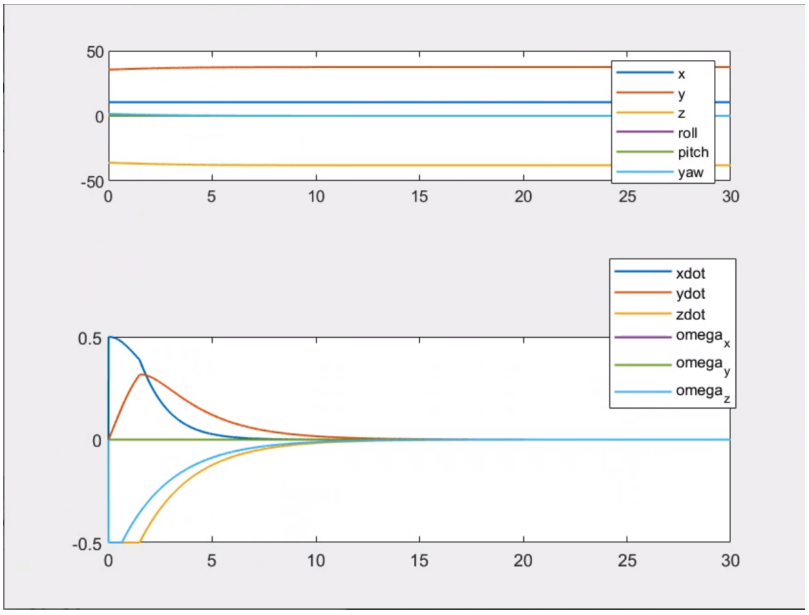

Figure 1: Position and velocity plot for the first task

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.

By default, the *Horizontal Attitude* is enabled and it has a higher priority than the *Vehicle Position* task. This results in the UVMS prioritizing the maintaining of a horizontal orientation attitude over the pitch and roll alignment (Figure 2).
When the *Horizontal Attitude* is not enabled, the UVMS tries to achieve the alignment between its frame and the goal one, without taking (of course) into account to maintain the horizontal orientation attitude (Figure 3).

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

The problem is the same as before. With the priorities swapped, if the target has roll or pitch different from zero, the vehicle will try to achieve such orientation.
Therefore, when we change the priorities, the behaviour observed during the simulation is almost the same as when we disable the *Horizontal Attitude* task (Figure 3).
Moreover, swapping the priority is wrong because the *Horizontal Attitude* is a safety task, which should have a higher priority than an action-defining task.

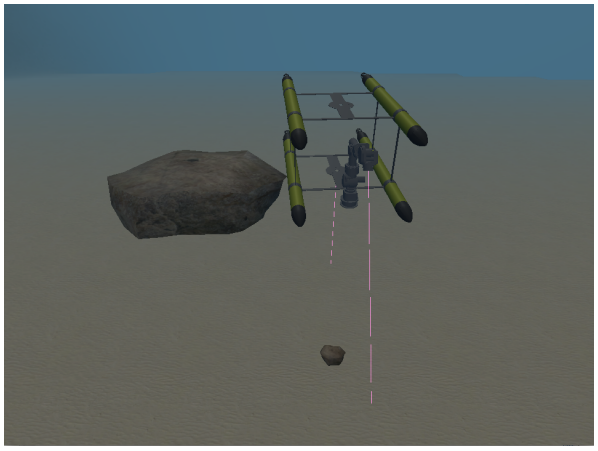

Figure 2: *Horizontal Attitude* enabled, the UVMS is parallel with the seafloor.

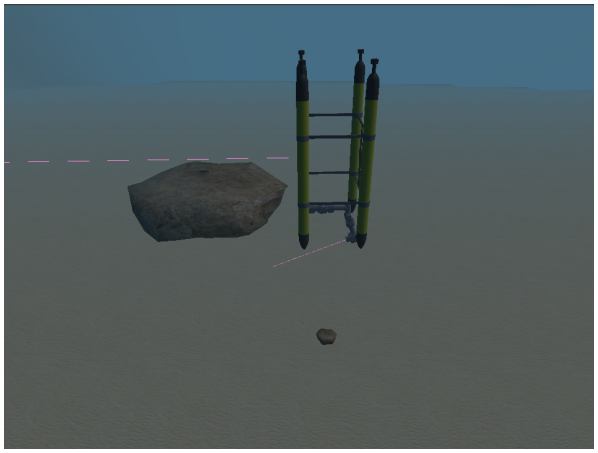

Figure 3: *Horizontal Attitude* not enabled, the UVMS is not parallel with the seafloor.

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?

If *Tool Position* control task is enabled and it has a higher priority than the *Vehicle Position* control task, the UVMS tries to achieve the goal position with the arm.
For this reason, we defined two different goal position, the first for the vehicle and the other one for the end-effector.
The settings we used for the Safe Waypoint Navigation are:

- enable *Manipulability* to avoid loss of dexterity,
- enable *Horizontal Attitude*,
- use the *Vehicle Position* to reach the vehicle desired position,
- enable the *Tool Position* only when the safe position is reached, to avoid that this task drags the vehicle in a not safe way.

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 hierarchy of tasks used and their priorities. Comment how you choose the priority level for the minimum altitude.

The new **hierarchy** of tasks is:

1. Minimum Altitude
2. Manipulability
3. Horizontal Attitude
4. Vehicle Position

The new task implemented:

Minimum Altitude [R, I, S]: it is a safety task, thus it has high priority. This task improves to the UVMS the ability to avoid collisions with the seafloor.

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

We need to control the movement along the z axis, therefore we use the same Jacobian of the *Vehicle Position*, but selecting only the component related to the z axis.

$$\boldsymbol{J}_{mav} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} & \underset{3\times3}{\mathbf{0}} & {}^{w}\underset{3\times3}{\boldsymbol{R}_v} \\ \underset{6\times7}{\mathbf{0}} & & \\ & {}^{w}\underset{3\times3}{\boldsymbol{R}_v} & \underset{3\times3}{\mathbf{0}} \end{bmatrix} \tag{4}$$

Again, notice that to select the desired component we used the vector [0 0 0 0 0 1] because the control variable uses the convention [Roll Pitch Yaw X Y Z].
We compute the **task reference** as:

$${}^{w}\dot{\bar{\boldsymbol{x}}}_{mav} = k((d_{limit} + \Delta) - {}^{w}d_{sensor})) \tag{5}$$

where:

- k is the control gain.
- d_{limit} is the desired minimum distance from the seafloor.
- Δ is the safety distance at which the activation of the task starts to trigger.
- ${}^{w}d_{sensor}$ is the third component of the distance vector, measured by the sensor and projected on the world frame.

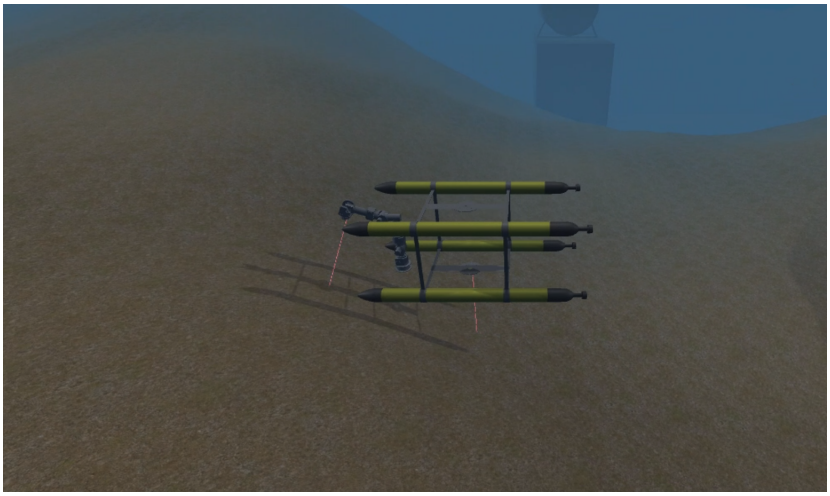

Figure 4: Minimum altitude is enabled, the UVMS is following the seafloor

1.2.3 Q3: Try imposing a minimum altitude of 1, 5, 10 m respectively. What is the behaviour?

In order to test this task, we simulated with different values for the minimum altitude, all with the same k gain (equal to 0.2).

- **10 m:** The UVMS has an initial position below this threshold, thus it starts floating towards the surface very quickly (it is possible to notice this vertical velocity $\dot{z}$, in (Figure 5)). Of course, the task is not accomplished because the target position is far from the threshold, therefore, this value does not seem a reasonable one.
- **5 m:** Again, the task is triggered immediately, because the initial position is below the minimum altitude value. This control task is a safety one, therefore every time the UVMS is under the minimum altitude, the activation is enabled and it has the highest priority. Again, it seems not a reasonable value. (Figure 7).
- **1 m:** The control task is no more active at the starting instant. The UVMS starts to accomplish the *Vehicle Position* task and, when the seafloor presents a hill, the vehicle starts to follow its shape (Figure 9).

However, there could be some troubles during the navigation if the seafloor has a steep slope: the front of the vehicle could collide with the soil because the sensor is placed in the center of the vehicle.

1.2.4 Q4: How was the sensor distance processed?

Since we are interested only in the component along the z axis, the ${}^{v}\boldsymbol{d}_{sensor}$ vector is build as:

$$\begin{bmatrix} 0 & 0 & d_{seafloor} \end{bmatrix}^{\top} \tag{6}$$

However, ${}^{v}\boldsymbol{d}_{sensor}$ is the distance vector measured by the sensor, projected on the vehicle frame. Since we need to project it on the world frame, we apply the following rotation matrix:

$${}^{w}\boldsymbol{d}_{sensor} = {}^{w}\boldsymbol{R}_{v}\,{}^{v}\boldsymbol{d}_{sensor} \tag{7}$$

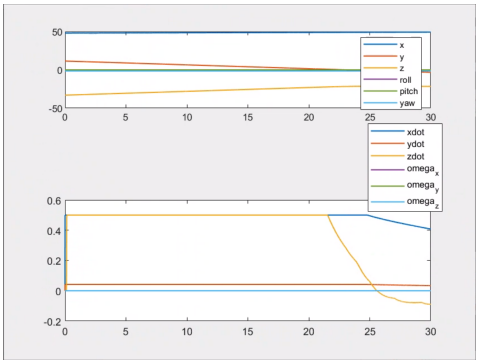

Figure 5: Position and velocity plot with Minimum Altitude threshold of 10m

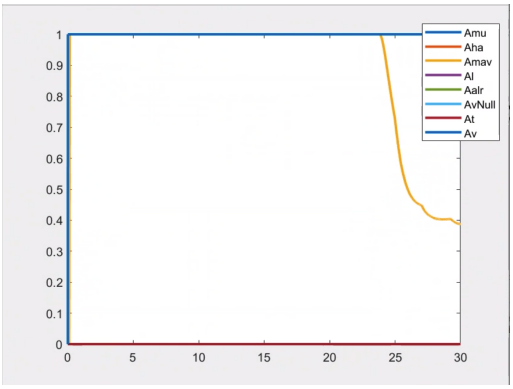

Figure 6: Activation plot (threshold 10m)

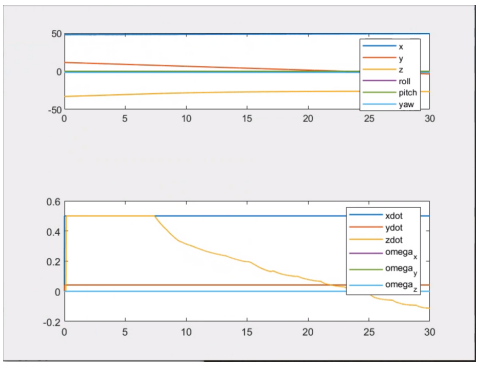

Figure 7: Position and velocity plot with Minimum Altitude threshold of 5m

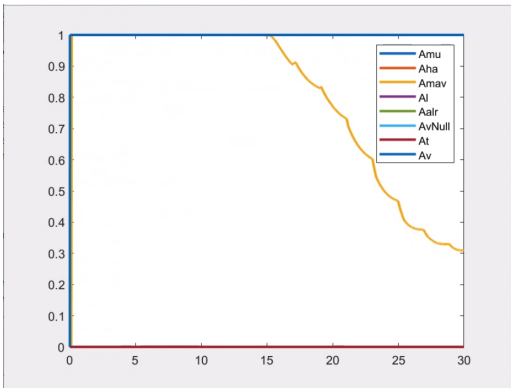

Figure 8: Activation plot (threshold 5m)

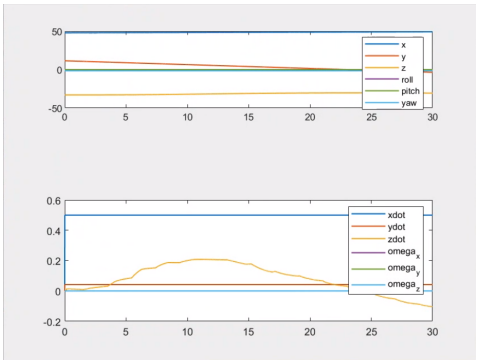

Figure 9: Position and velocity plot with Minimum Altitude threshold of 1m

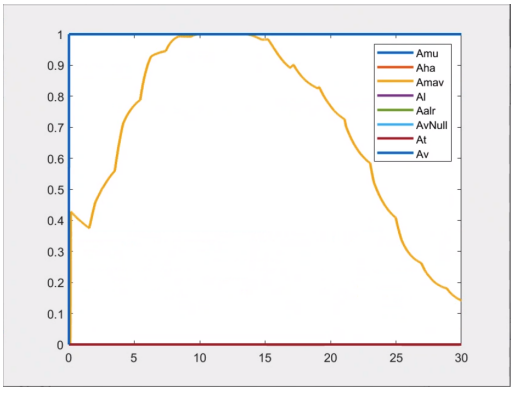

Figure 10: Activation plot (threshold 1m)

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. Comment how you choose the priority level for the altitude control task.

The new **hierarchy** of tasks is:

1. Manipulability
2. Horizontal attitude
3. Landing

Landing [R, E, AD]: it is the new task. It is an Action Defining task, thus it has the same level of priority of the *Vehicle Position*.

The task *Minimum Altitude* is not enabled now, because we are required to land, therefore the UVMS needs to go below the fixed minimum altitude threshold.

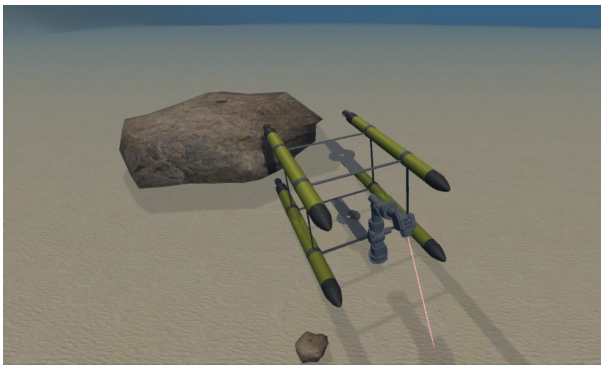

Figure 11: Implemented basic *Landing* action

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

The Jacobian relationship for the *Landing* is:

$$\boldsymbol{J}_{land} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} & \underset{3\times3}{\mathbf{0}} & {}^{w}\underset{3\times3}{\boldsymbol{R}_v} \\ \underset{6\times7}{\mathbf{0}} & & \\ & {}^{w}\underset{3\times3}{\boldsymbol{R}_v} & \underset{3\times3}{\mathbf{0}} \end{bmatrix} \tag{8}$$

As before, the Jacobian is required to control only the component along the z axis.
We compute the **task reference** as:

$${}^{w}\dot{\overline{\boldsymbol{x}}}_{land} = k((d_{landing} + \Delta_{safeguard}) - {}^{w}d_{sensor}) \tag{9}$$

where:

- k is the control gain.
- $d_{landing}$ is the distance from the seafloor, in this case is 0.
- $\Delta_{safeguard}$ is set to $0.17m$ to avoid interpenetration between UVMS and the seafloor.
- ${}^{w}d_{sensor}$ is the component along the z axis of the distance vector measured by the sensor and projected on the world frame.

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

There are two major differences:

- The first one is that *Landing* is not a safety task, but an action-defining one. While the *Minimum Altitude* task is used to avoid collisions with the seafloor, the *Landing* task defines an action (as the *Vehicle Position* task), therefore it has lower priority.

- The second difference is that *Minimum Altitude* is an inequality task rather than *Landing* which is an equality one. As visible on the plots below (Figure 12), MAV is active when we are under the defined threshold while *Landing* is triggered only when a prerequisite is satisfied and it is not deactivated until we land on the seafloor.

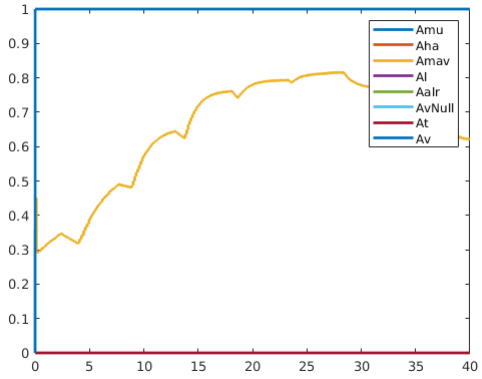

Figure 12: MAV activation plot

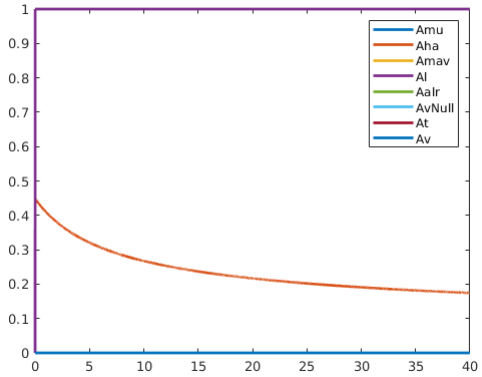

Figure 13: Landing activation plot

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.

We have two different actions, with this **hierarchy** and the following activations:

Control Task	`Code name`	Action A	Action B
Minimum Altitude Vehicle	`mav`	Active	Inactive
Manipulabity	`mu`	Active	Active
Horizontal Attitude	`ha`	Active	Active
Landing	`l`	Inactive	Active
Vehicle Position	`v`	Active	Inactive

Action A: the UVMS performs the safe waypoint navigation. All the safety tasks are enabled. This Action finishes when the position error is below a fixed threshold.

Action B: the UVMS simply lands on the seafloor, performing the previous exercise task. Of course, the *Minimum Altitude* task is disabled.

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

The transition between Action A and Action B is triggered by the achievement of the vehicle target position. We compute the cartesian error between the goal frame and the vehicle frame, the mission phase changes when the error is below the given threshold (in this case $0.1m$).

When we want to disable a running task, we use a Decreasing Bell Shape function to perform a smooth transition. In this case, the *Minimum Altitude* and *Vehicle Position* tasks are both disabled at the start of the second phase. Similarly, when we want to activate a task, we use an Increasing Bell Shape function to perform a smooth transition, as we do for the *Landing* task. We compute the slope of such functions based on the mission phase time: this helps us to fix the time interval (in our case, 0.2 seconds) in which the vehicle switches from an Action to the other and let us achieve a smoother shape, avoiding discontinuities in the actuation of the motors.

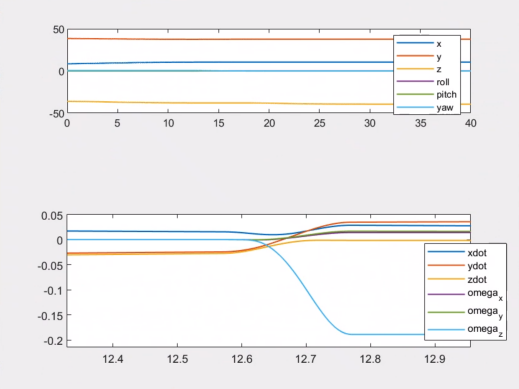

Figure 14: Zoom of the smooth change

We found this more adequate than the previous attempt, in which we performed the shape based on the error distance.

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.

3.1.1 Q1: Report the unified hierarchy of tasks used and their priorities. Comment the behaviour.

We have three different actions, with the following **hierarchy** and activations:

Control Task	`Code name`	Action A	Action B	Action C
Minimum Altitude Vehicle	`mav`	Active	Active	Inactive
Manipulabity	`mu`	Active	Active	Active
Horizontal Attitude	`ha`	Active	Active	Active
Alignment to the Rock	`alr`	Inactive	Active	Active
Landing	`l`	Inactive	Inactive	Active
Vehicle Position	`v`	Active	Inactive	Inactive

Alignment to the Rock [R, I, P]: it is a prerequisite task, therefore it has a higher priority than the action-defining tasks. Inequality, because we admit a range of error equal to 0.07. !

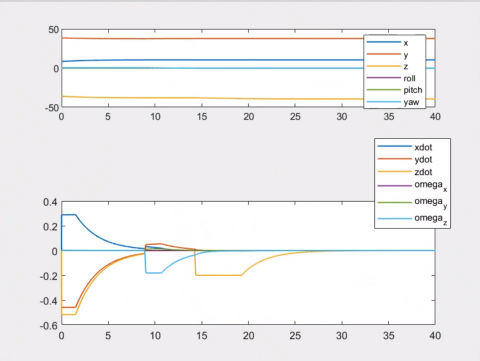

Figure 15: Positions and velocities during the three actions.

Action A, as before, is safe waypoint navigation with all the safety tasks enabled (Figure 16). This Action finishes when the position error is below a fixed threshold.

Action B is the action that performs the alignment to the rock, with all the safety task enabled (Figure 17). his Action finishes when the misalignment error is below a fixed threshold.

Action C performs the safe landing on the seafloor. *Alignment to the Rock* is gradually disabled to optimize the alignment while the vehicle is performing the landing (Figure 18).

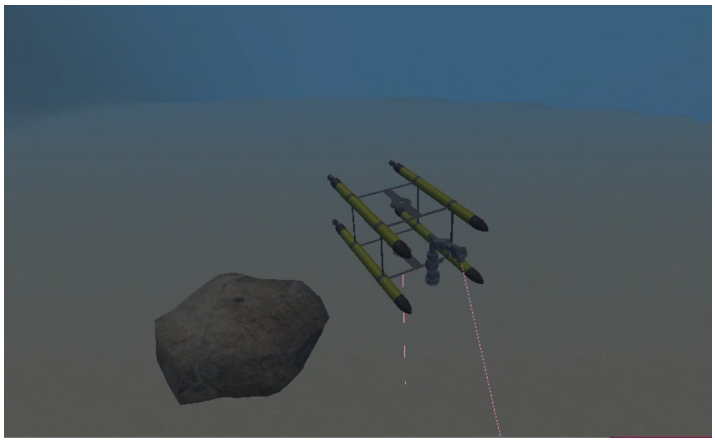

Figure 16: Action A: Navigation till the target position

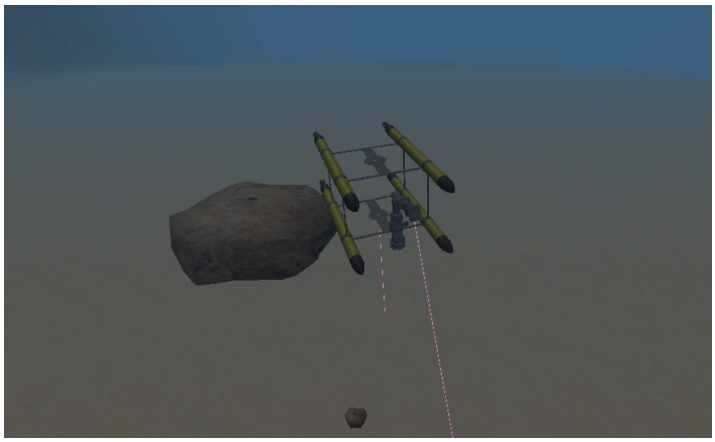

Figure 17: Action B: Alignment with the rock center

Figure 18: Action C: Landing, final action

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

The Jacobian relationship for the Alignment is derived from the following formula:

$$D_w(\boldsymbol{\rho}) = \boldsymbol{J}_{alr}\dot{y} \tag{10}$$

where:

- $\boldsymbol{\rho}$ is the misalignment vector.
- $D_w(\boldsymbol{\rho})$ is the derivative of the misalignment vector.
- $\boldsymbol{J}_{alr}$ is the Jacobian we want to compute.

We know that:

$$D_w(\boldsymbol{\rho}) = {}^w\boldsymbol{n}_\rho\dot{\theta} + \underbrace{\theta D_w{}^w\boldsymbol{n}_\rho}_{orthogonal} \tag{11}$$

Since we are not interested in the orthogonal components, we can ignore them.
By looking at the first term of the sum, we know that ${}^w\boldsymbol{n}_\rho$ is the misalignment versor projected on the world and $\dot{\theta}$ can be written as :

$$\dot{\theta} = {}^w\boldsymbol{\omega}_v - {}^w\boldsymbol{\omega}_{rock} \tag{12}$$

where:

- ${}^w\boldsymbol{\omega}_v$ is the angular velocity of the vehicle, referred to the world.
- ${}^w\boldsymbol{\omega}_{rock}$ is the angular velocity of the projected distance between the rock and the vehicle, referred to the world.

By describing ${}^w\boldsymbol{\omega}_v$ in terms of $\dot{\boldsymbol{y}}$, we can deduce the following Jacobian

$$\boldsymbol{J}_{vehicle} = \begin{bmatrix} \underset{3\times10}{\mathbf{0}} & \underset{3\times3}{{}^w\boldsymbol{R}_v} \end{bmatrix} \tag{13}$$

In order to describe ${}^w\boldsymbol{\omega}_{rock}$ in terms of $\dot{\boldsymbol{y}}$, we use the following relationship

$${}^w\boldsymbol{v}_v = {}^w\boldsymbol{\omega}_{rock} \wedge {}^w\boldsymbol{d} \tag{14}$$

where:

- ${}^w\boldsymbol{v}_v$ is the linear velocity of the vehicle projected on the world frame.
- ${}^w\boldsymbol{d}$ is the projected distance on the horizontal inertial frame, between the rock and the z axis of the vehicle.

$${}^w\boldsymbol{w}_{rock} = \frac{1}{\|{}^w\boldsymbol{d}^2\|}\left[{}^w\boldsymbol{d}\wedge\right]{}^w\boldsymbol{v}_v \tag{15}$$

Since we are interested only in the x and y component of ${}^w\boldsymbol{v}_v$, we select such components by premultiplying like this:

$${}^w\boldsymbol{w}_{rock} = \frac{1}{\|{}^w\boldsymbol{d}^2\|}\left[{}^w\boldsymbol{d}\wedge\right]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}{}^w\boldsymbol{v}_v \tag{16}$$

From the last equation, we deduce the following Jacobian

$$\boldsymbol{J}_{rock} = \frac{1}{\|{}^{w}\boldsymbol{d}^2\|} \left[{}^{w}\boldsymbol{d}\wedge\right] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{0}_{3\times7} & {}^{w}\boldsymbol{R}_{v}{}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{17}$$

The resulting Jacobian is obtained by substitution and it is equal to

$$\boldsymbol{J}_{\text{alr}} = {}^{w}\boldsymbol{n}_{\rho}^{\top}[\boldsymbol{J}_{vehicle} - \boldsymbol{J}_{rock}] \tag{18}$$

We compute the **task reference** as:

$$\dot{\bar{\boldsymbol{x}}}_{alr} = k(0 - \|{}^{w}\boldsymbol{\rho}\|) \tag{19}$$

where:

- k is the control gain.
- $\|{}^{w}\boldsymbol{\rho}\|$ is the norm of the misalignment vector, in this case we want it to be 0.

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?

We performed tests with different gain values for the task reference:

- Very Small = 0.1
- Medium = 0.6
- High = 1.2

By looking at the position and velocity plots, it is clear that there are no significant differences between the medium (Figure 20) and the high gain (Figure 21) cases. The only difference is that the transition between phases in the high gain case is not as smooth as in the medium one. Of course, the saturation prevents the former to perform a much higher velocity. For the very small gain, the plot is very different (Figure 19), because the velocity achieved during the alignment is not enough to reach the desired position. In our 30 seconds simulation is impossible for the UVMS to align correctly. However, by increasing the simulation time, it is possible to accomplish our goal, even with such slow behaviour.

From the point of view of the activations and the errors, the results are coherent with the ones observed in the position and velocity plots.

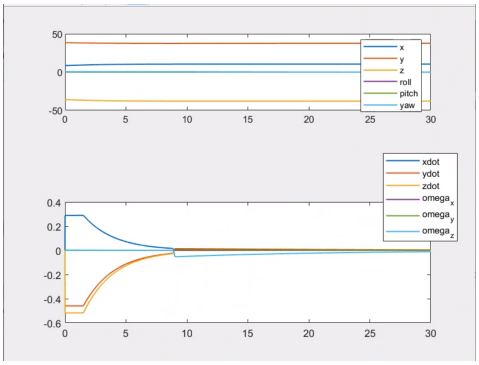

Figure 19: Positions and Velocity with small gain

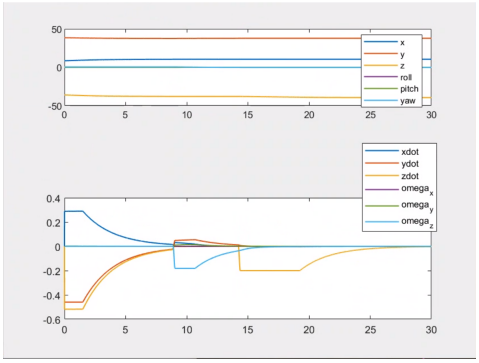

Figure 20: Positions and Velocity with medium gain

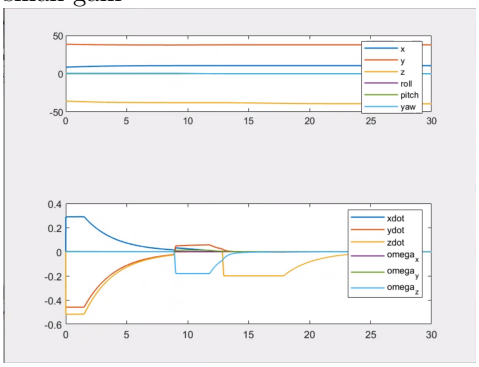

Figure 21: Positions and Velocity with high gain

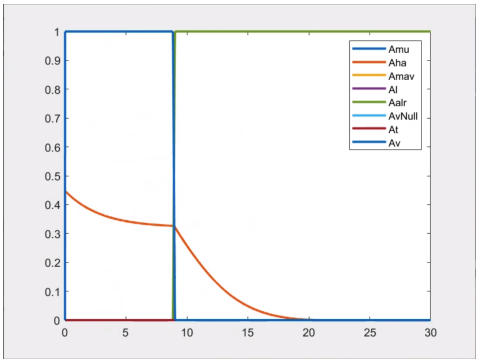

Figure 22: Activations with very small gain

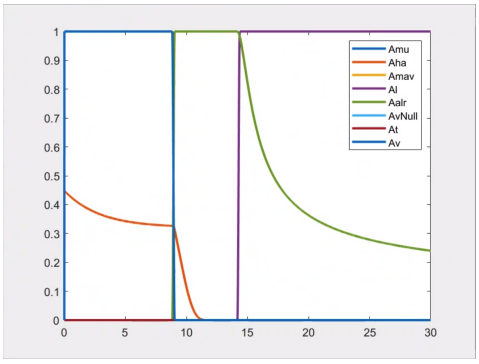

Figure 23: Activations with medium gain

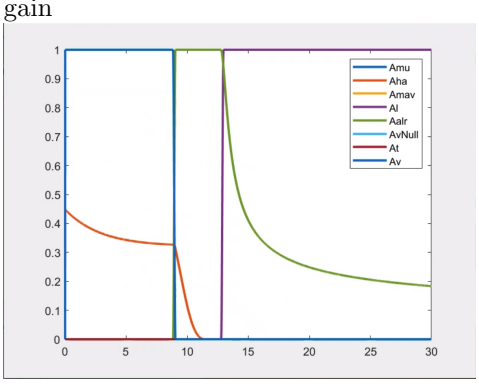

Figure 24: Activations with high gain

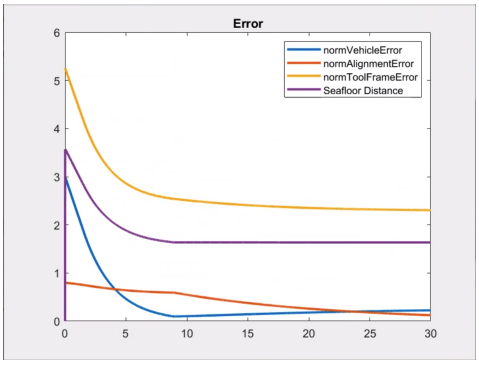

Figure 25: Errors with small gain

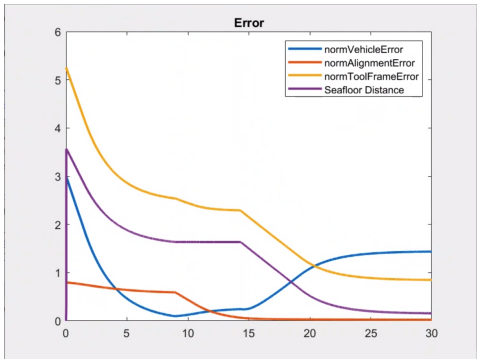

Figure 26: Errors with medium gain

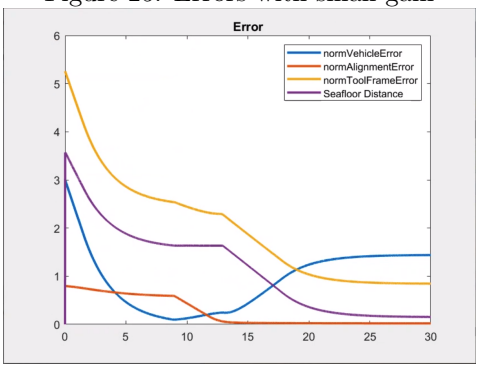

Figure 27: Errors with high gain

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.

The tool is able to reach the center of the nodule, however, we notice that the vehicle "helps" the arm to achieve the desired position by moving itself (coherently with the expected behaviour of the *Tool* task). It is clear that to avoid this problem, we need to perform a non-reactive task to constraint the vehicle movements, as we will do in the next exercise 4.1.

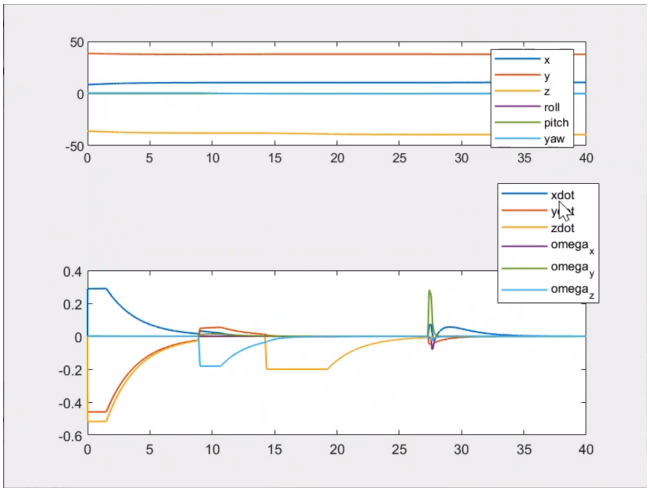

Figure 28: The velocity plot shows that when tool starts moving, the vehicle moves as well

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 constraint 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 unified hierarchy of tasks used and their priorities. At which priority level did you add the constraint task?

We have four different actions, with the following **hierarchy** and activations:

Control Task	`Code name`	Action A	Action B	Action C	Action D
Vehicle Null Velocity	`vNull`	Inactive	Inactive	Inactive	Active
Minimum Altitude Vehicle	`mav`	Active	Active	Inactive	Inactive
Manipulabity	`mu`	Active	Active	Active	Active
Horizontal Attitude	`ha`	Active	Active	Active	Active
Alignment to the rock	`alr`	Inactive	Active	Active	Inactive
Landing	`l`	Inactive	Inactive	Active	Inactive
Tool Position	`t`	Inactive	Inactive	Inactive	Active
Vehicle Position	`v`	Active	Inactive	Inactive	Inactive

Vehicle Null Velocity [NR, E, C]: it is a constraint task, therefore it has higher priority than all the other tasks. The goal of the task is to prevent the vehicle movements.

Tool Position [R, E, AD]: it is an Action Definition task, that enables the arm to reach the tool target position, eventually by moving the vehicle as well.

Action A, safe waypoint navigation with all the safety task enabled. This Action finishes when the position error is below a fixed threshold (in this case $0.1m$).

Action B, alignment to the nodule with all the safety task enabled. This Action finishes when the misalignment error is below a fixed threshold (in this case $0.07m$).

Action C, landing and smooth rotation to be aligned with the rock. This Action finishes when the vehicle touches the seafloor (in the simulation this happens at $\approx 0.17m$).

Action D, manipulator actuation after landing. In this Action *Vehicle Null Velocity* task is enabled, preventing vehicle movements. The only movement will be the extension of the manipulator to reach the desired target tool position.

The transitions between Actions are visibile in the Figure 29.

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

The Jacobian for the Vehicle Null Velocity task is the same of the Vehicle Position:

$$\boldsymbol{J}_{vehNull} = \begin{bmatrix} & \underset{3\times3}{\mathbf{0}} & \underset{3\times3}{{}^{w}\boldsymbol{R}_{v}} \\ \underset{6\times7}{\mathbf{0}} & & \\ & \underset{3\times3}{{}^{w}\boldsymbol{R}_{v}} & \underset{3\times3}{\mathbf{0}} \end{bmatrix} \tag{20}$$

Since we do not want the vehicle to move, the **task reference** is a zero vector defined as:

$$\dot{\bar{\boldsymbol{x}}}_{vNull} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{\top} \tag{21}$$

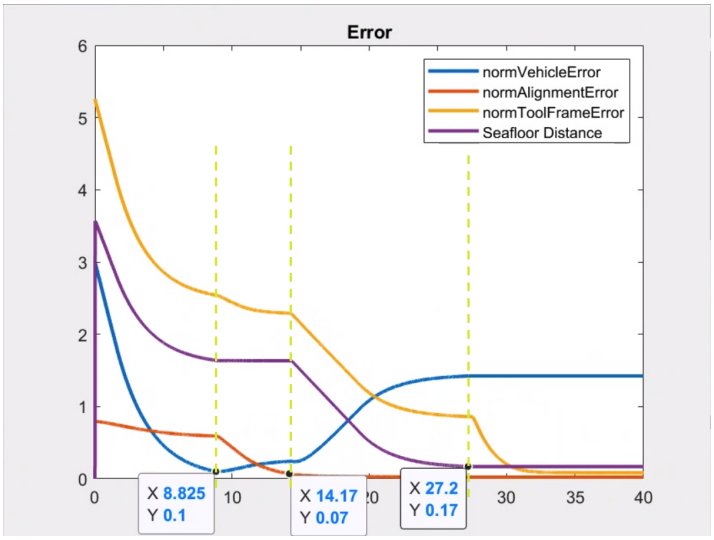

Figure 29: The error plot during the different phases (vertical green lines)

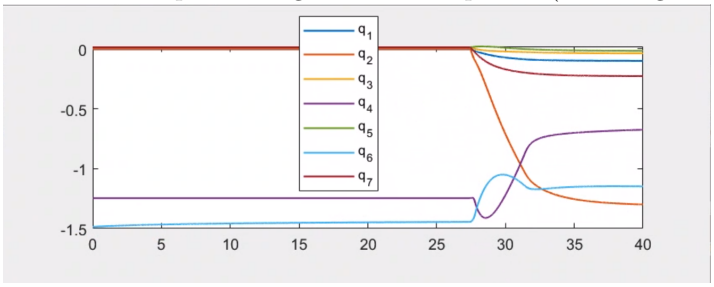

Figure 30: Arm joints start moving when Action D is triggered

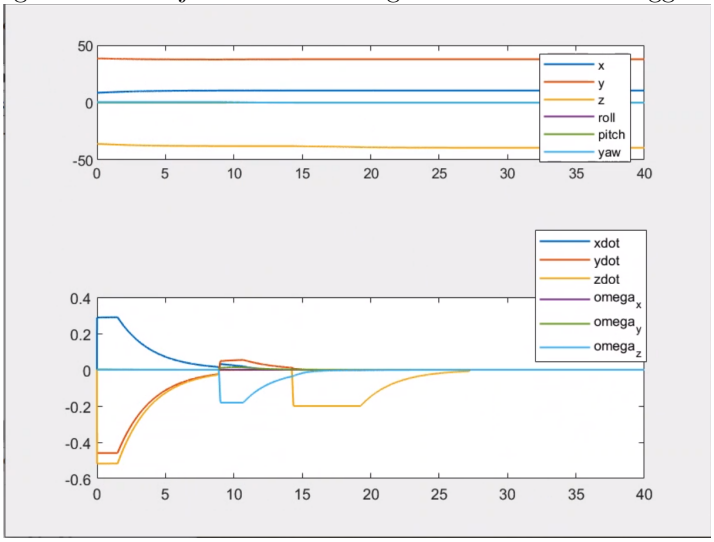

Figure 31: Vehicle has zero velocities when Action C is finished

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

The *Vehicle Null* only puts vehicle velocities to zero, therefore it is not able to compensate disturbances. In a similar scenario, the UVMS will be influenced by the currents: if the disturbance is not too big, the vehicle is able to arrive at the desired position and to land. However, when the *Vehicle Null* task is triggered in Action D, the UVMS will start drifting, eventually losing the target position. The arm will keep trying to reach the goal by stretching as much as possible.

With a check on the position error during the mission phase update, it is possible to return to Action A and achieve again the desired position.

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 unified hierarchy of tasks used and their priorities. At which priority level did you add the constraint task?

We have four different actions, with the following **hierarchy** and activations:

Control Task	`Code name`	Action A	Action B	Action C	Action D
Vehicle Null Velocity	`vNull`	Inactive	Inactive	Inactive	Active
Joint Limit	`jl`	Active	Active	Active	Active
Minimum Altitude Vehicle	`mav`	Active	Active	Inactive	Inactive
Manipulabity	`mu`	Active	Active	Active	Active
Horizontal Attitude	`ha`	Active	Active	Active	Active
Alignment to the rock	`alr`	Inactive	Active	Active	Inactive
Landing	`l`	Inactive	Inactive	Active	Inactive
Tool	`t`	Inactive	Inactive	Inactive	Active
Vehicle Position	`v`	Active	Inactive	Inactive	Inactive

Joint Limit [R, I, S]: it is a safety task, thus it has higher priority than the other action-defining tasks. It is able to control that the joints don't operate over their fixed threshold.

The actions are the same as before, the only difference is that Joint Limit is always active because it is a safety task. However, it is important when the manipulator performs the final action.

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

The Jacobian for the Joint Limit task is:

$$\boldsymbol{J}_{jointLimit} = \begin{bmatrix} \underset{7\times7}{\boldsymbol{I}} & \underset{7\times6}{\boldsymbol{0}} \end{bmatrix} \tag{22}$$

With this Jacobian we are sure that each of the seven joint moves itself within defined limits.
We compute the **task reference** as:

$$\dot{\overline{x}}_{jointLimit,i} = k \begin{cases} x_{activationMin,i} - x_{q,i} & \text{if } x_{q,i} \leq mid_i \\ x_{activationMax,i} - x_{q,i} & \text{if } x_{q,i} > mid_i \end{cases} \tag{23}$$

where:

- $i = 1, \ldots, 7$ is the i-th joint.
- $x_{q,i}$ is the i joint position (q)
- $ActivationMin$ and $ActivationMax$ are the 10% of the limit imposed for each joint.
- $mid_i = \dfrac{x_{ActivationMin,i} + x_{ActivationMax,i}}{2}$ is the middle point between the Minimum Activation and the Maximum.
- k is the gain

We use the Min and Max in order to be able to activate the joint limit when the joint arrives near the minimum guard ($ActivationMin$) or near the maximum one ($ActivationMax$).

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 unified hierarchy of tasks used and their priorities. At which priority level did you add the optimization task?

The new hierarchy of tasks is:

- Manipulability
- Horizontal attitude
- Tool
- Preferred Shape

Preferred Shape [NR, I, O]: it is an optimization task. For this reason, it is placed at the end of the hierarchy which means that it has the lowest priority and it is accomplished only if all the other tasks has been already satisfied.

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

We decided to implement this task as a vector one, thus the **Jacobian** is a 4×13 matrix:

$$\boldsymbol{J}_{vehNull} = \begin{bmatrix} \underset{4\times4}{\boldsymbol{I}} & \underset{4\times9}{\boldsymbol{0}} \end{bmatrix} \tag{24}$$

The Jacobian has four rows since it is required to control just the first four joints. Since each joint affects only itself (similarly to the joint limits task), the non-zero part of the Jacobian is a 4×4 identity matrix.

We compute the **task reference** as the difference between the desired joint configuration and the actual one:

$$\dot{\overline{\boldsymbol{x}}}_{\text{prefShape}} = k(\boldsymbol{x}_{\text{prefShape}} - \boldsymbol{x}_q) \tag{25}$$

where:

- k is the control gain.
- $\boldsymbol{x}_{\text{prefShape}}$ is the desired configuration of the first four joints.
- $\boldsymbol{x}_q$ is the actual configuration of the first four joints.

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

In this specific case the UVMS goal is to align the end-effector frame to the goal frame which is located slightly above the pipe. Since no explicit goal for the vehicle was given, we have used just the tool task (which is able to move the vehicle as well) in order reach the goal directly. Nevertheless, since we have just one mission phase the manipulator moves in order to achieve the goal as soon as possible, leading to a "stretched" shape (Figure 32). If we add the preferred shape constraint as defined in the above mentioned task, we can maintain the desired end-effector configuration, even with just one mission phase (Figure 33). As we can see in the figures below (Figure 34 and Figure 35), when the preferred shape task is active the joints are in a more natural shape, which is also further from the joint limits.

Figure 32: Simulation with Mission Phase

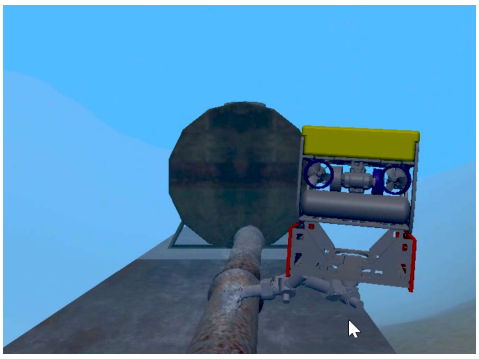

Figure 34: Simulation without Mission Phase

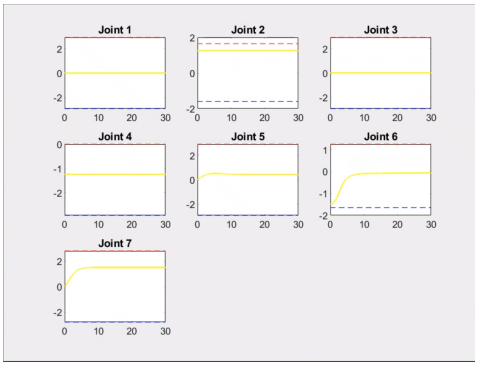

Figure 33: Joints Limits with Mission Phase

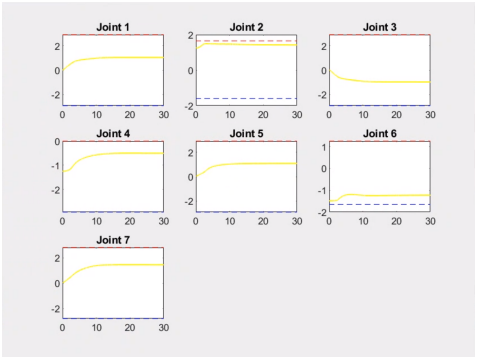

Figure 35: Joints Limits without Mission Phase

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.

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

We have two different actions, with the following **hierarchy** and activations:

Control Task	`Code name`	Action A	Action B
Vehicle Null Velocity	`vNull`	Inactive	Active
Joint Limit	`jl`	Active	Active
Manipulabity	`mu`	Active	Active
Horizontal Attitude	`ha`	Active	Active
Tool	`t`	Inactive	Active
Vehicle Position	`v`	Active	Inactive
Preferred Shape	`ps`	Active	Active

The transition from Action A to Action B is triggered whenever the position error is below a priori fixed threshold.

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, it was not possible to constrain the vehicle velocities to zero, in order to allow a safe manipulation. By adding a safe navigation phase, it possible to move the UVMS by using the *Vehicle Position* instead of using the *Tool* task. In the second phase, the *Tool* task is triggered, but the vehicle velocities are constrained to zero: the manipulator is able to safely reach the goal frame. Moreover, this results in a smoother behaviour as it is shown in the figures below (Figure 36 and 37).

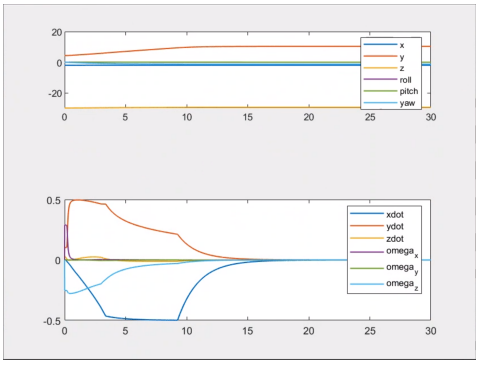

Figure 36: Action performed with save navigation

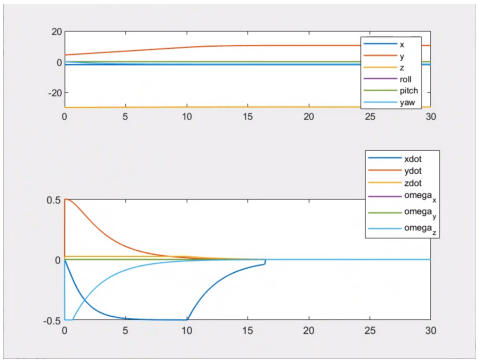

Figure 37: Action performed without save navigation

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 coordinate. 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?

In order to implement an arm-vehicle coordination scheme, we implemented two optimizations (TPIK1 and TPIK2) running in parallel. Given an action A, the scheme works as follow:

1. TPIK1 is the same as the previous exercises. It considers the vehicle and the manipulator as a fully controllable system. When the hierarchy of task corresponding to the action A is solved, we take the reference for the vehicle velocities while we discard the joints velocities one.

2. TPIK2 is equal to TPIK1 plus an additional task at the top of the hierarchy.
   Constrain Vehicle Velocities [NR, E, C], this task is a non-reactive one and it is used to constrain the vehicle velocities to coordinate the arm and vehicle movements.

   The **Jacobian** for this task is:
   $$\boldsymbol{J}_{vConstr} = \begin{bmatrix} \mathbf{0}_{6\times 7} & \boldsymbol{I}_{6\times 6} \end{bmatrix} \tag{26}$$
   Its reference is the vector of the vehicle velocities computed by TPIK1 at the previous step. In a real-world case, the reference would be the vector of the measured vehicle velocities, but, since we are working in a simulation environment, we are allowed to use the ones computed by the algorithm in the previous loop iteration. Therefore, TPIK2 provides the optimal joints velocities by taking into account the "real" movement of the vehicle, which is different from the one computed by the algorithm because of the sinusoidal disturbance (without TPIK2 images: 38, 40, 42, 44; with TPIK2 images: 39, 41, 43, 45).

6.1.2 Q2: What happens if the sinusoidal disturbance becomes too big?

The vehicle and the manipulator try to compensate the disturbances. This means that if the gain is high enough and the amplitude of the disturbance is not too big, they are fast enough to reduce the error at the price of a smooth behaviour. Nevertheless when the amplitude of the sinusoidal disturbance becomes too big, they are not able to follow and to minimize the error, resulting in a sinusoidal behaviour.

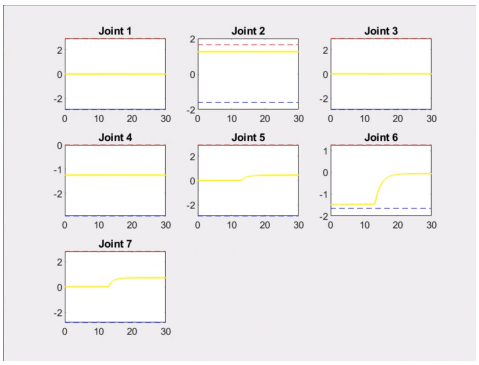

Figure 38: Joint limits: without TPIK2

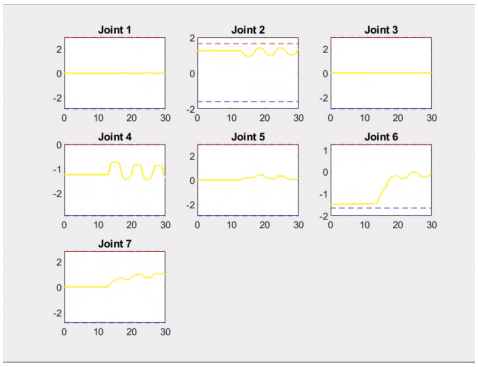

Figure 39: Joint limits: with TPIK2

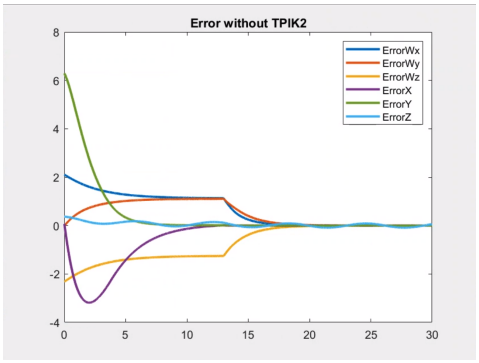

Figure 40: Errors: without TPIK2

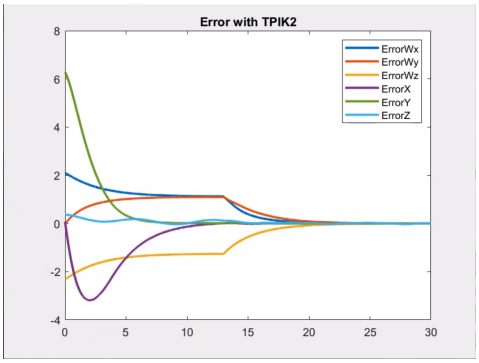

Figure 41: Errors: with TPIK2

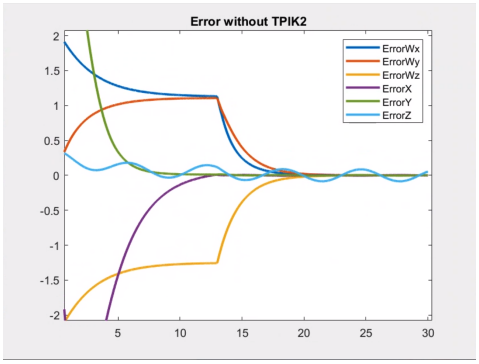

Figure 42: Zoomed Error: without TPIK2

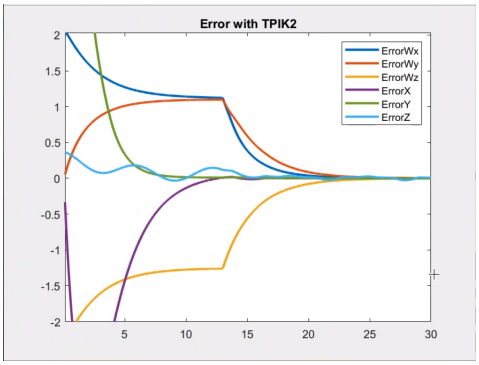

Figure 43: Zoomed Error: with TPIK2

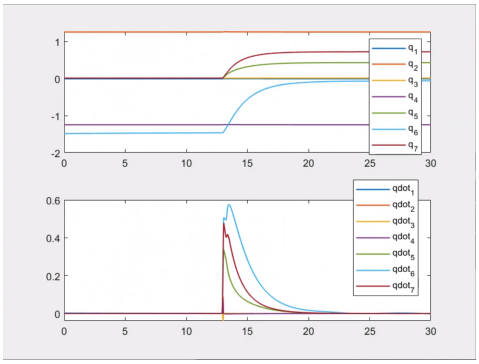

Figure 44: Joints configurations and velocities: without TPIK2

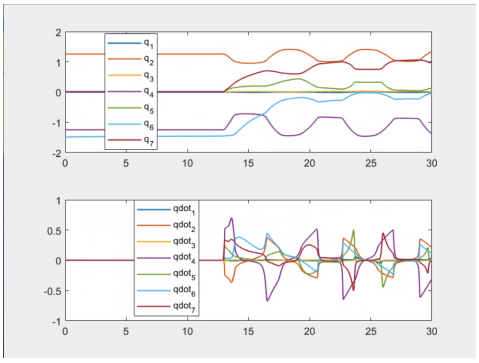

Figure 45: Joints configurations and velocities: with TPIK2