Cooperative Robotics

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

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 task 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 give 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 in order to prevent dangerous arm position.

Horizontal Attitude [R, I, S], it is fundamental for being sure the robot it is parallel with the seafloor.

Vehicle Position [R, I, AD], what the system really do. It is an action-defining task, so, it has lower hierarchy. This is the first task we implemented, the goal is to compute velocities to enable the robot to reach the desired position.

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

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

where ${}^w\boldsymbol{R}_v$ is the Rotation matrix from the vehicle frame to the world frame.
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 use the Cartesian error between the vehicle frame and the goal frame, both projected on the world frame, to compute the linear error, and then, the misalignment between the two is computed by the Versor Lemma. We multiply the final vector by a gain (k) factor.

Code for Jacobian definition and computation:
`ComputeJacobian`
Code for computing distance and misalignment:
`simulation script -> CartError`
`simulation script -> UnitVersorLemma`

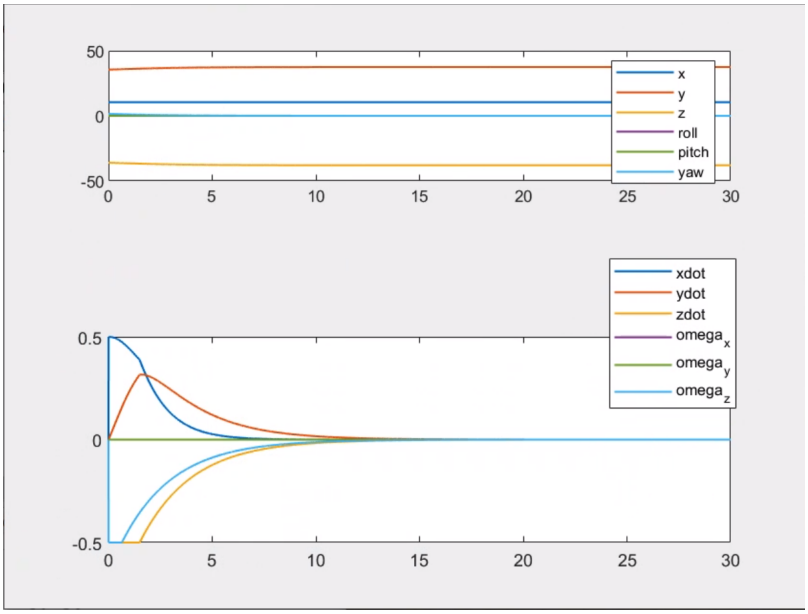

Figure 1: Resulted Graph for the first task

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

By default the *Horizontal Attitude* is enable and it has higher priority than the *Vehicle Position* task. This makes in the UVMS trying to keep an horizontal orientation with respect to the world frame(Figure 2): if the target position has pitch and roll different from zero, the robot doesn't perform this vertical position, as shown during the simulation(Figure 3). When the *Horizontal Attitude* is not enable, the UVMS try to find the alignment between its own frame and the goal one, without taking (of course) into account to stay in an horizontal position. If the target position has pitch or roll, the robot will try to perform them.

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. This time with the priorities switched, if the target, again, as roll or pitch, the robot will try to achieve this orientation. So, if we change the hierarchy, we will have the same performance as if we disable the *Horizontal Attitude* task.
In the end, it is not useful to switch the position and it is wrong because the *Horizontal Attitude* is a safety task, so it is, every time, more important (so high priority) than an action-defining task.

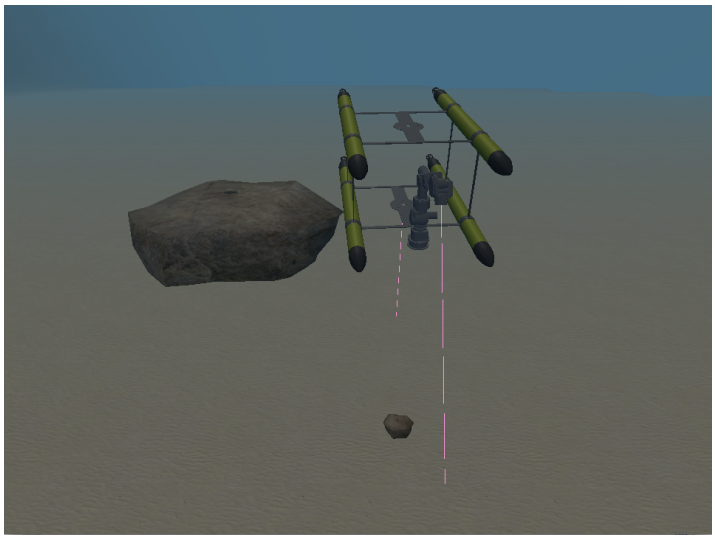

Figure 2: Images from the simulator, *Horizontal Attitude* enable, the robot is parallel with the seafloor

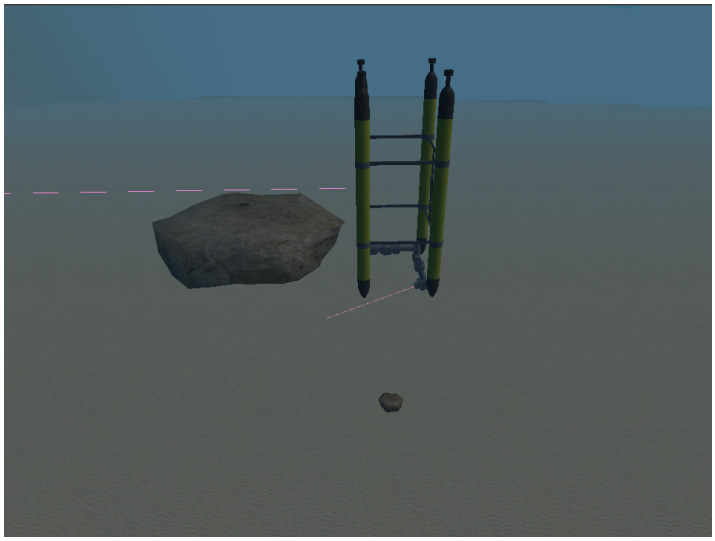

Figure 3: Images from the simulator, *Horizontal Attitude* not enable, the robot is no more parallel with the seafloor

1.1.4 Q4: What is the behavior 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 the *Tool Position* control task is enable, the vehicle could not reach the desired position. If it is enable and it has higher priority than the *Vehicle Position* control task, the robot try to achieve the goal position for the arm, not the ones referred to the vehicle it self.
For this reason we split the goal position in two different, the first for the body of the vehicle and another one for the tool. For this particular exercise we didn't use a arm goal position.
However, we continue to use the *Manipulability* task, in order to maintain a safe position for the arm, during the navigation.

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 task 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, so it has high priority. This task improve to the UVMS the ability to avoid crashes 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, so we use the same Jacobian of the vehicle position, but selecting only the component related to the z axis.

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

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

$${}^{w}\dot{\boldsymbol{x}}_{mav} = k((d_{limit} + \Delta) - {}^{w}\, d_{sensor})) \tag{4}$$

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 triggers.
- ${}^{w}d_{sensor}$ is the distance vector measured by the sensor and projected on the world frame.

We compute the difference between the minimum distance given from the operator and the measured distance by the sensor projected on the world frame.

Code for Exercise:

```
Compute Task Reference -> MAV Control
```

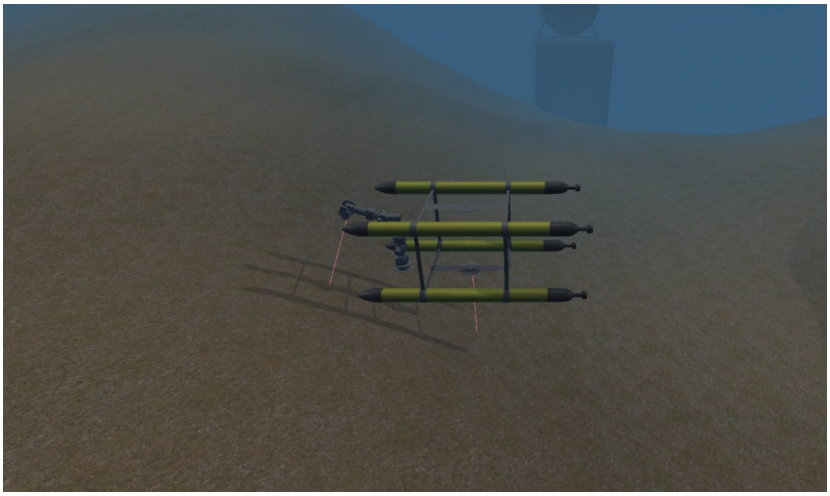

Figure 4: Image from the simulator, the minimum altitude is set to $0.5m$, the robot is following the seafloor

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

To achieve this task we perform the different minimum altitude, all with the same k gain (equal to 0.2), and with the same Δ for the inequality nature of the problem (equal to $0.5m$).

- **10 m:** It seems no a reasonable value. The UVMS has initial position below this threshold, so it start floating to the surface very quickly (it is possible to notice this huge vertical velocity $\dot{z}$, in the graph (Figure 5)). Of course, the task is not accomplished because the target position is far from the threshold.
- **5 m:** The same as before, the activation is active at the first instant because again the initial position is below the minimum altitude. This control task is a safety one, so every time the UVMS is under the minimum altitude, the activation is active and it has the highest priority. Again, it seems no reasonable to perform such big gap with the seafloor (Reported graph, Figure 7).
- **1 m:** This time the problem is the opposite. The control task is no more active at the starting instant. The UVMS starts to accomplish the *Vehicle Position* task, when the seafloor present an hill the UVMS start to follow the pattern (Figure 9). However, there are some troubles during the navigation because the "noise" of the robot, with only 1 meter of distance (and no Δ that could prevent this), could touch the soil because the sensor it is on the center of the vehicle.

1.2.4 Q4: How was the sensor distance processed?

${}^{v}d_{sensor}$ is the distance vector measured by the sensor, so we need to projected it on the world frame, to do this we apply The Rotation matrix as:

$$ {}^{w}\boldsymbol{d}_{sensor} = {}^{w}\boldsymbol{R}_{v}\,{}^{v}\boldsymbol{d}_{sensor} \tag{5} $$

Again we are only interested on the z axis. The $\boldsymbol{d}_{sensor}$ are the vectors build as: $\begin{bmatrix}0 & 0 & d_{seafloor}\end{bmatrix}^{\top}$

Code for Exercise:

```
TO DO
TO DO
```

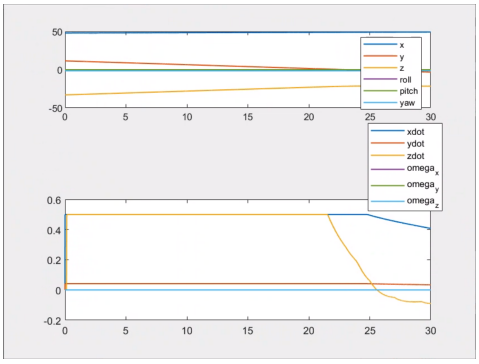

Figure 5: Position and velocities during the Minimum Altitude Vehicle task imposed 10m threshold

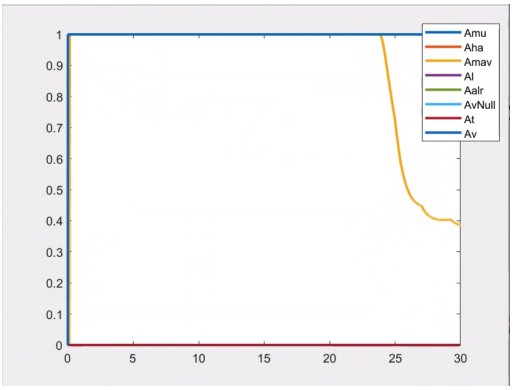

Figure 6: Positions and Velocity with medium

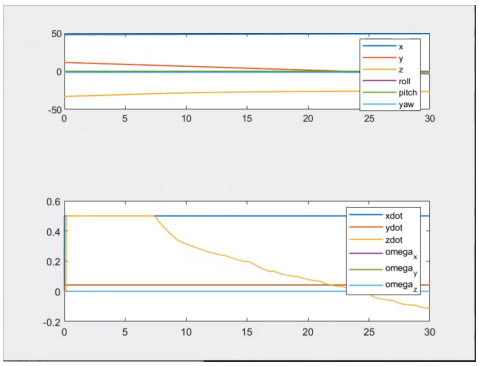

Figure 7: Position and velocities during the Minimum Altitude Vehicle task imposed 10m threshold

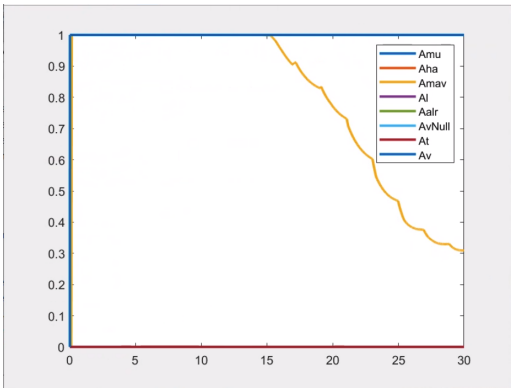

Figure 8: Positions and Velocity with medium

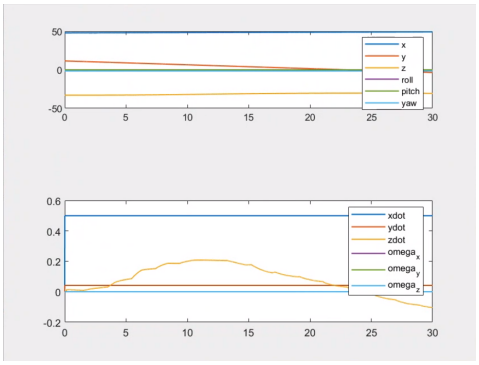

Figure 9: Position and velocities during the Minimum Altitude Vehicle task imposed 1m threshold

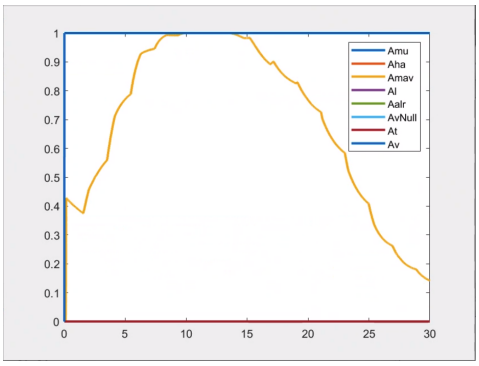

Figure 10: Positions and Velocity with medium

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 task 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 a Action Defining task, so it has the same priority as the *Vehicle Position* task.

The task *Minimum Altitude* is not enable now, because (of course) we need, to land and the UVMS needs to go below the fixed minimum altitude threshold.

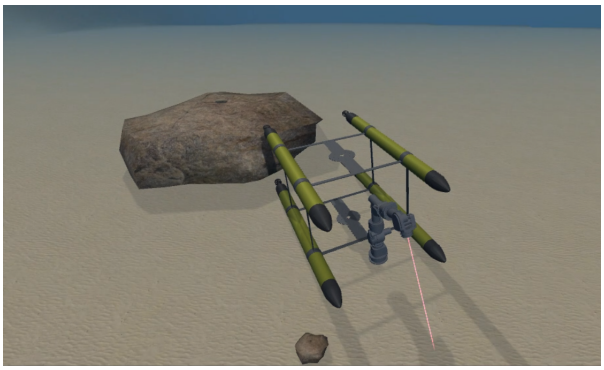

Figure 11: Implemented basic "Landing" action

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:

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

As before the Jacobian take into account only the z component (using the pre-multiplication with the [0 0 0 0 0 1] matrix).

We compute the **task reference** as as the difference between the seafloor and the measured distance projected on the world frame:

$$^{w}\dot{\boldsymbol{x}}_{land} = k((seafloor + safeguard) - ^{w} d_{sensor})) \tag{7}$$

where:

- k is the control gain.
- $seafloor$ is simply 0.
- $safeguard$ is set to $0.15m$ to avoid interpenetration between UVMS and the seafloor.
- $^{w}d_{sensor}$ is the distance vector measured by the sensor and projected on the world frame.

Code for Exercise:

```
TO DO
TO DO
```

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

There are two major difference, the first one is that this task it is not a safety task, it is a action-defining one. The *Minimum Altitude* it is used to avoid crashes with the seafloor, the *Landing* task it is like the *Vehicle Position* task, and, as this last one, it has low priority.
The second big difference is that the *Minimum Altitude* is an inequality task rather than the *Landing* that is and equality. For this reason, as visible on the graphs below (Figure 12), MAV is active when we are under the imposed threshold, *Landing* is active only when is called and it stay like this until we touch the seafloor.

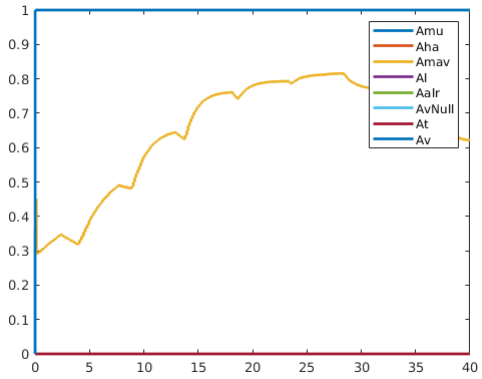

Figure 12: Graph MAV

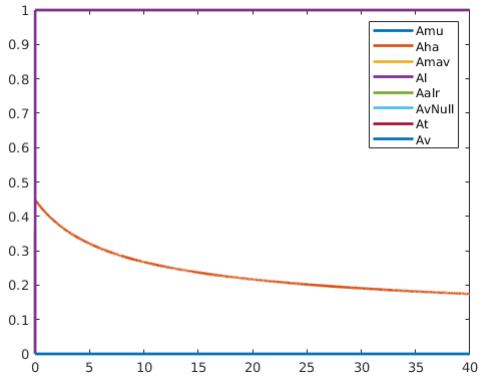

Figure 13: Graph Landing

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 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, perform the safe waypoint navigation. All the safety task are enable.

Action B, the robot need simply to land, it just performs the previous exercise task. Of course, we need to inactivate the Minimum Altitude.

Code for Exercise:
`ComputeMissionPhase`

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

From Action A to Action B, we compute the cartesian error between the goal position and the vehicle. When the error is below our threshold ($0.15m$) the mission phase changes.

When we want to inactivate a running task, we perform a Decreasing Bell Shape function, referred to mission phase time, in order to perform a smooth transition (like *Minimum Altitude* and *Vehicle Position* task, both inactivated in the same method). In the same way, we also perform an Increasing Bell Shape function to perform a smooth activation of the *Landing* task. We perform all this functions based on the mission phase time, this help us to fix the time when we switch from an Action to the other one, and perform shape from 0 to 0.2 seconds, to avoid spike to the motors.

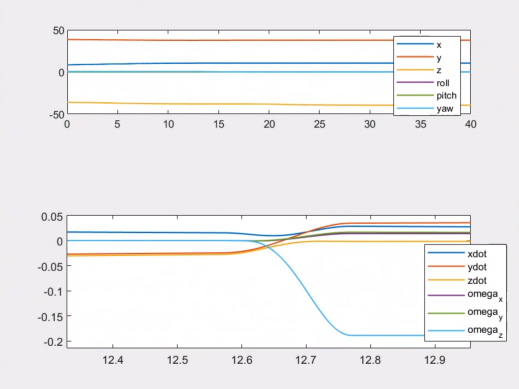

Figure 14: Zoom of the smooth change

We found this more adeguate that 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 behavior.

We have three different actions, with this 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

The new task is:

Alignment to the Rock [R, I, P], it is a prerequisite task, so it has higher priority than the action-defining tasks. Inequality, because we admit a Δ of errors equal to *value*. !

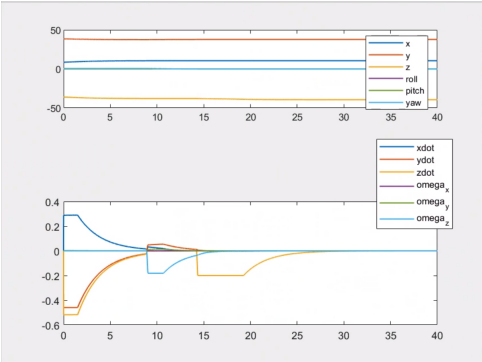

Figure 15: Positions and velocities during the three actions.

Action A, again, a safe waypoint navigation with all the safety tasks enable (Figure 16).

Action B, it is the action that performs the alignment to the rock, with all the safety task enable (Figure 17).

Action C, *Alignment to the Rock* is not disable because, while the UVMS starts landing, we disable smoothly the alignment. In this case, while the robot is performing the landing, it continue to rotate to optimize the orientation (Figure 18).

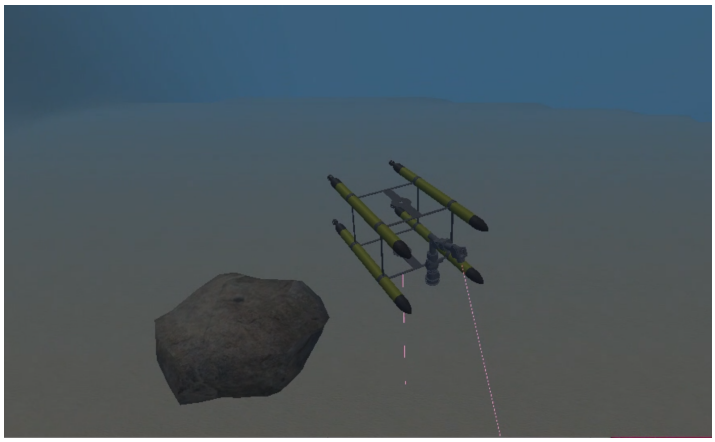

Figure 16: Action A: Navigation till the target position

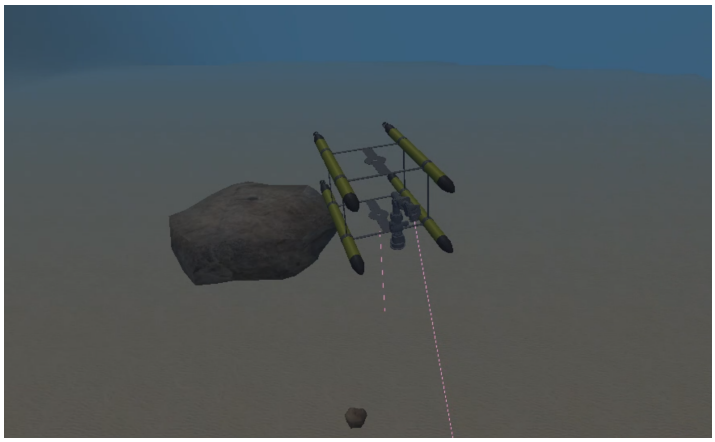

Figure 17: Action B: Alignment with the rock center

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?

$$\boldsymbol{J}_{alr} = {}^{w}\boldsymbol{n}_{\varphi}{}' \left[\underset{3x7}{\mathbf{0}}, \boldsymbol{\Phi} \cdot {}^{w}\boldsymbol{R}_{v}, {}^{w}\boldsymbol{R}_{v} \right]$$
$$\boldsymbol{\Phi} = -\frac{1}{\left\| {}^{w}\boldsymbol{d}_{v,r}^{2} \right\|} \left({}^{w}\boldsymbol{d}_{v,r} \wedge \right) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{8}$$

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
- Big = 1.2

First, looking at positions and velocities graphs, it is clear that between the medium (Figure 20) and the big gain (Figure 21) there are no such important differences, with the big gain that transition between the two phases are not so smooth as for the medium one. Of course, the saturation helps the case with the big gain to not perform a too high velocity. For the very small gain, the resulted graph is very different (Figure 19), becausethe very low velocity performed during the alignment doesn't help to reach the desired position. In our 30 seconds simulation is impossible for the UVMS to align correctly. However, also improving the simulation time, it is normal that it is too slow to accomplish our goal.

From the point of view of the activations and the errors, there are not different results than before. Medium and big gain graphs are very similar (almost identical) and the very small gain graph confirm the previous sentences.

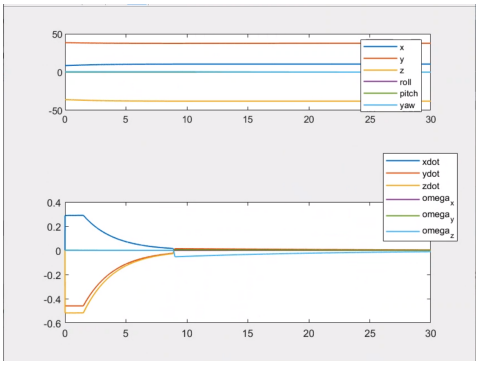

Figure 19: Positions and Velocity with small gain

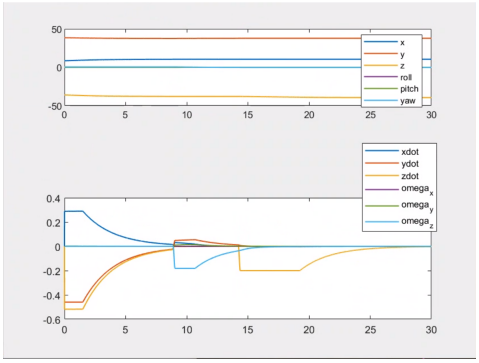

Figure 20: Positions and Velocity with medium gain

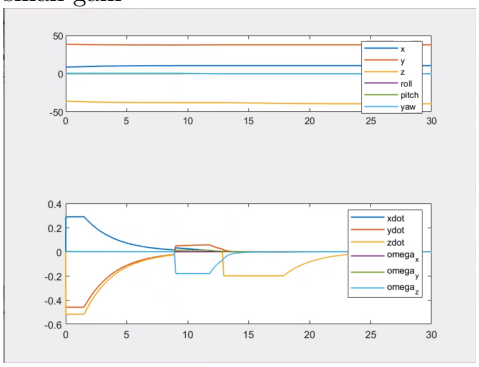

Figure 21: Positions and Velocity with big gain

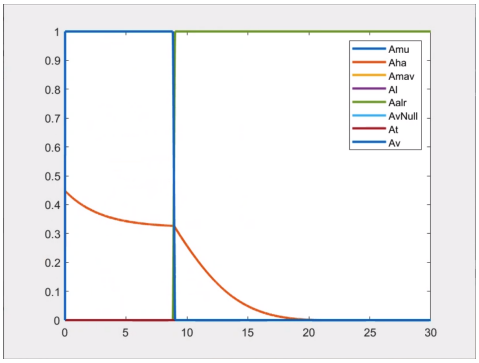

Figure 22: Activations with very small gain

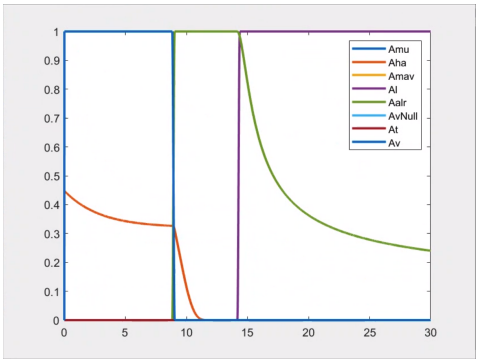

Figure 23: Activations with medium gain

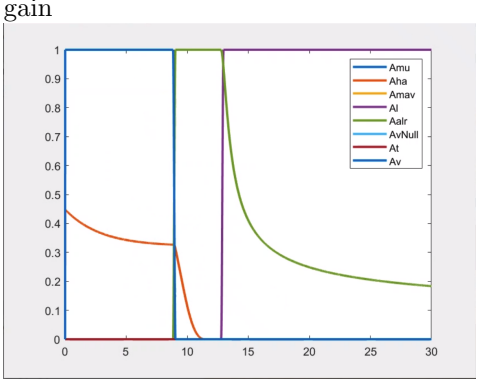

Figure 24: Activations with big gain

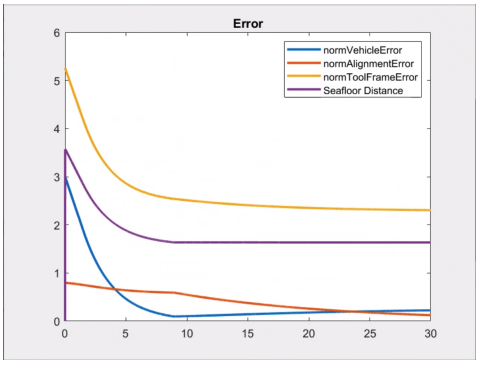

Figure 25: Errors with small

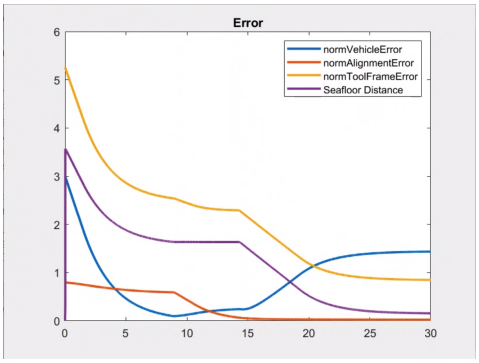

Figure 26: Errors with medium

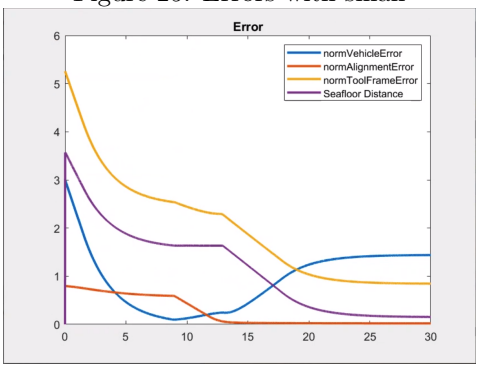

Figure 27: Errors with big

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 behavior.

The tool could reach the center of the nodule, however, we notice that the vehicle moves to "help" the arm to achieve the desired position. It is clear that, to avoid this problem, we need to perform a non-reactive task to constraint the vehicle not move, as we will do in the next exercise 4.1.

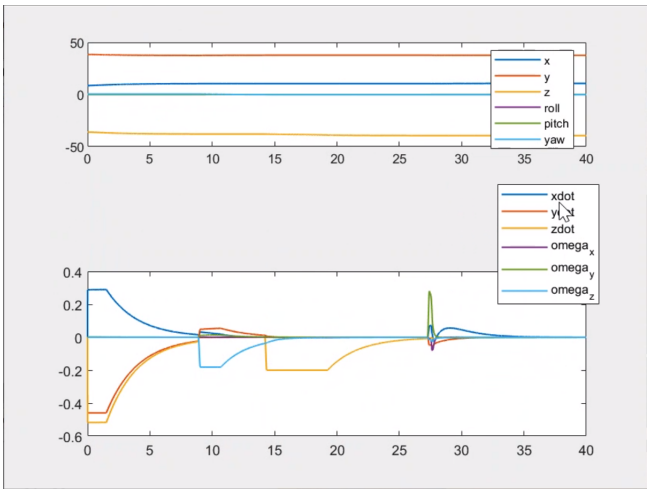

Figure 28: Visible that when tool starts move, vehicle helps it

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 this 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	`t`	Inactive	Inactive	Inactive	Active
Vehicle Position	`v`	Active	Inactive	Inactive	Inactive

Vehicle Null Velocity [NR, E, C], it is a constraint task, so it has higher priority than all the other tasks. The goal of the task is to block the motors.

Tool Position [R, E, AD], it is an Action Definition task, that enable the arm to reach the tool target position.

Action A, safe waypoint navigation with all the safety task activated

Action B, alignment to the nodule with all the safety task activated

Action C, landing and final rotation to be aligned with the rock

Action D, the new one. Here it is enable the new *Vehicle Null Velocity* task, that disable the UVMS' motors, to prevent movement. The only movement will be the extension of the arm to reach the desired target tool position.

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, but this time we need to fix the velocity of the UVMS to zero, so we premultiply it for the zeros vector.

$$\boldsymbol{J}_{vehNull} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} & \underset{3\times3}{\mathbf{0}} & {}^{w}\underset{3\times3}{\boldsymbol{R}_{v}} \\ \underset{6\times7}{\mathbf{0}} & & \\ & {}^{w}\underset{3\times3}{\boldsymbol{R}_{v}} & \underset{3\times3}{\mathbf{0}} \end{bmatrix} \tag{9}$$

The velocity will not change during the activation of this task because it is a Non-Reactive.

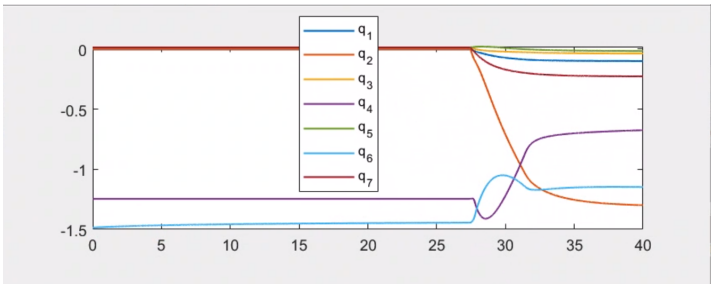

Figure 29: Arm joints start moving when reached task 4

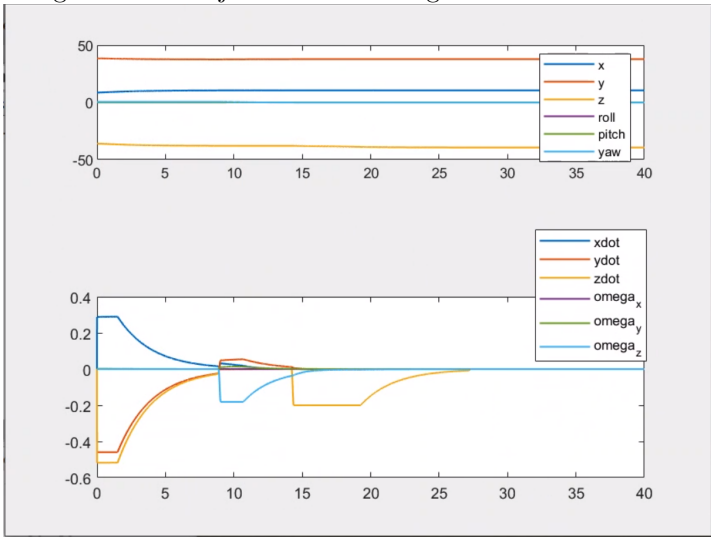

Figure 30: Vehicle has zero velocities when task 3 is finished

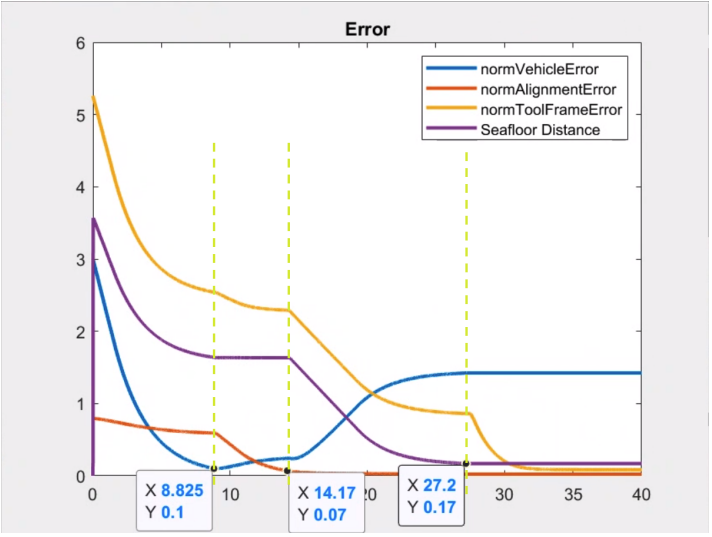

Figure 31: Vehicle has zero velocities when task 3 is finished

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 task cannot take in position the UVMS, it doesn't take into account disturbances, it only put velocities to zero, so it is not able to contrast currents. In this scenario, the robot will start follow the current and it will go to a wrong position. The arm will continue to try to reach the desired position, and because it is not landed, the *Vehicle Null Velocity* is not active, so the Tool Position try to achieve its goal and bring back all the vehicle. If the currents are not too strong it could reach again the position. Different from the previous exercise 3.1.4, in which the vehicle could not help the arm to be able to arrive to the target, now it could "help". !

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 this 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, so it has higher priority than the other action-defining tasks. It is able to control that the joints don't operate over their prefixed 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 robot perform 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:

$$\boldsymbol{J}_{jointLimit} = \begin{bmatrix} \underset{7\times7}{\boldsymbol{I}} & \underset{7\times6}{\boldsymbol{0}} \end{bmatrix} \tag{10}$$

With this Jacobian we are sure that each joint apply the limit constraints only itself and not on the other ones.
We compute the **task reference** as: Seen that each joint influences obviously only itself for the joint limit task.

$$\dot{\bar{x}}_{jointLimit,i} = k \begin{cases} x_{activationMin,i} - x_{q,i} & \text{if } x_{q,i} \leq mid_i \\ x_{activationMax,i} - x_{q,i} & \text{if } x_{q,i} > mid_i \end{cases} \tag{11}$$

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 ones ($ActivationMax$).

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] 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-by-13 matrix:

$$\boldsymbol{J}_{vehNull} = \begin{bmatrix} -0.0031 & 1.2586 & 0.0128 & -1.2460 \end{bmatrix}^\top \begin{bmatrix} \underset{4\times4}{\boldsymbol{I}} & \underset{4\times9}{\boldsymbol{0}} \end{bmatrix} \tag{12}$$

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

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

$$\dot{\bar{\boldsymbol{x}}}_{prefShape} = k(x_{prefShape} - x_q) \tag{13}$$

- k is the control gain.
- $x_{prefShape}$ is the desired configuration of the first four joints.
- 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

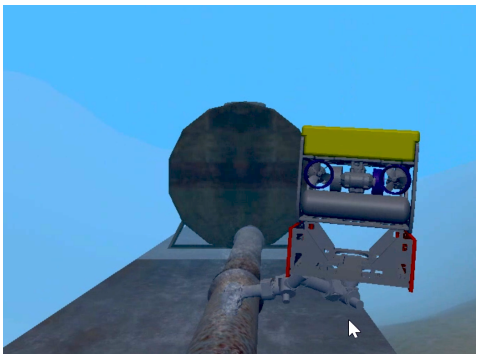

Figure 34: Simulation without Mission Phase

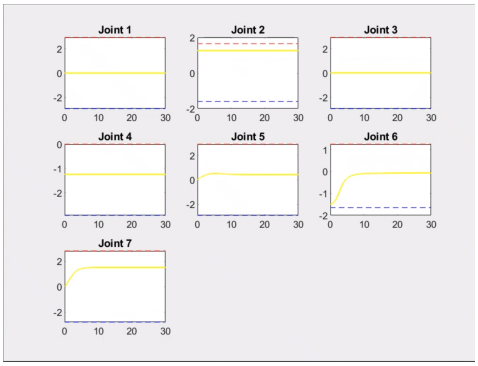

Figure 33: Joints Limits with Mission Phase

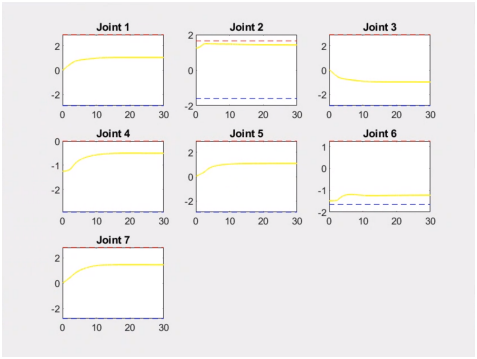

Figure 35: Joints Limits without Mission Phase

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?

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

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 is possible to use the vehicle position task to move the UVMS to a position slightly above the pipe, then constrain vehicle velocities in order to allow the manipulator to reach the goal frame by activating the tool task. Moreover, this results in a smoother behavior as it shown in the figures below (Figure 36 and 37).

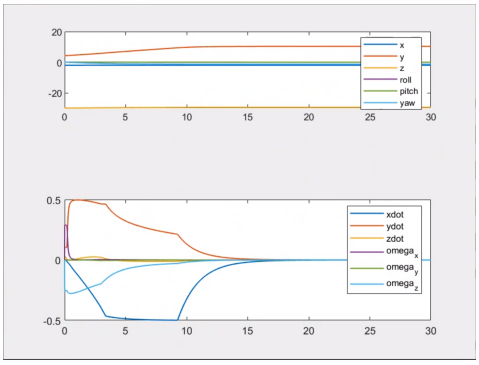

Figure 36: Simulation with Mission Phase

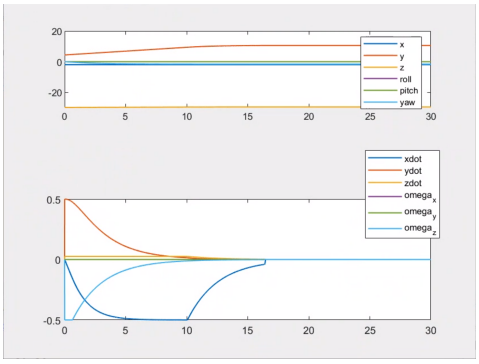

Figure 37: Joints Limits with Mission Phase

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 of the previous exercises. It considers the vehicle and the manipulator as a fully controllable system. Therefore, 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. This task is a non-reactive one and it is used to constrain the vehicle velocities. The Jacobian for this task is:

$$\boldsymbol{J}_{vConstr} = \begin{bmatrix} \underset{6\times7}{\mathbf{0}} & \underset{6\times6}{\boldsymbol{I}} \end{bmatrix} \tag{14}$$

   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.

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 behavior visible on the related plot (AGGIUNGERE IMMAGINI)??.

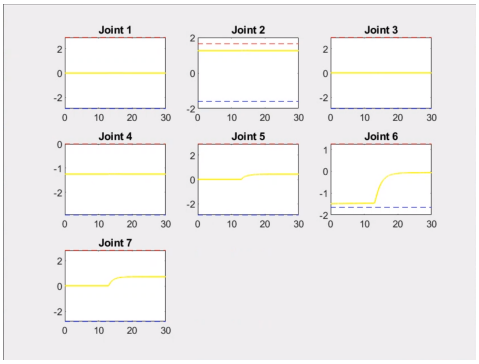

Figure 38: Joint limits: without TPIK2

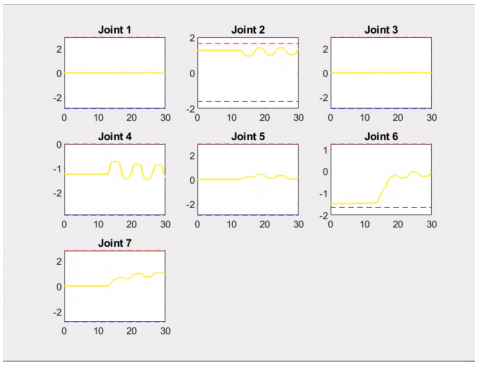

Figure 39: Joint limits: with TPIK2

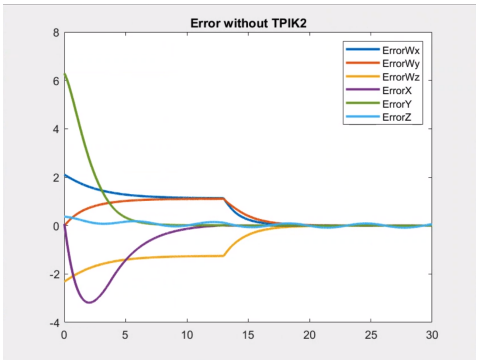

Figure 40: Errors: without TPIK2

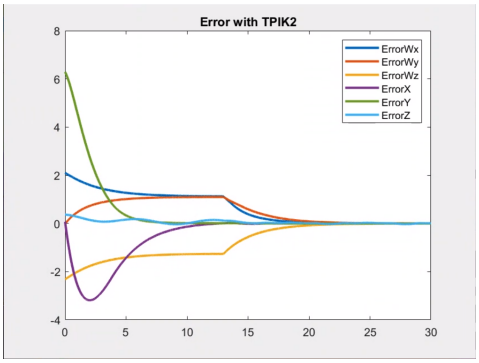

Figure 41: Errors: with TPIK2

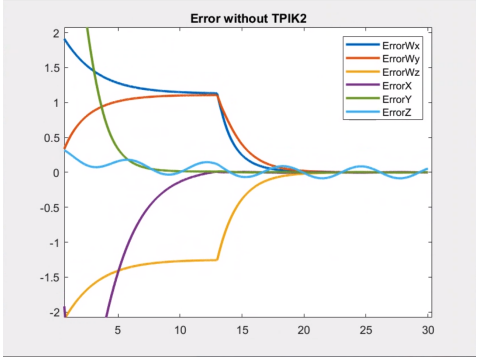

Figure 42: Zoomed Error: without TPIK2

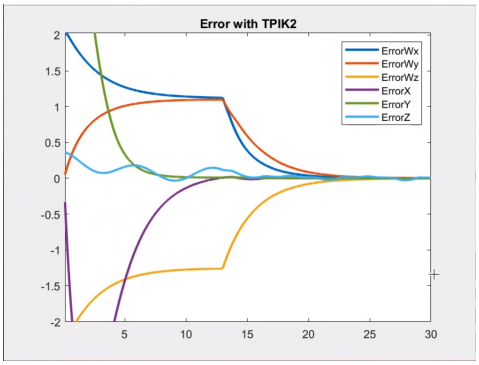

Figure 43: Zoomed Error: with TPIK2

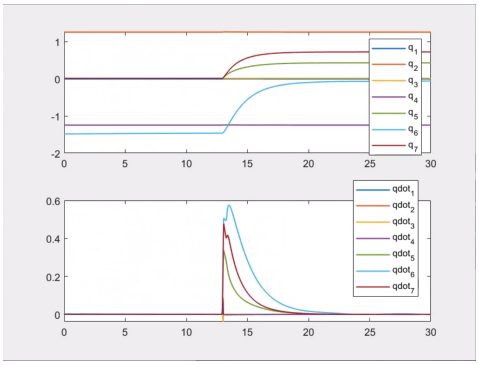

Figure 44: Joints configurations and velocities: without TPIK2

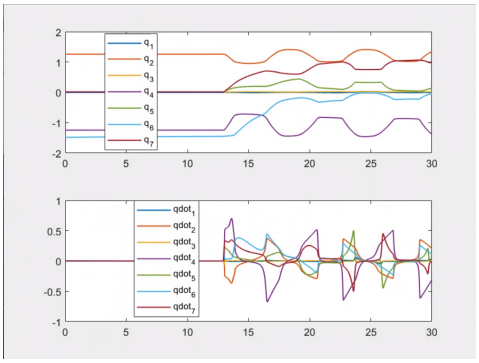

Figure 45: Joints configurations and velocities: with TPIK2