Cooperative Robotics

Authors: Piccinini Davide, Porta Francesco and Gotelli Andrea
EMAILs: 4404040@studenti.unige.it, 4376330@studenti.unige.it, 4343879@studenti.unige.it
Date: 21/12/2020

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

1.1 Adding a vehicle position and attitude 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 and attitude control task, and test that the vehicle reaches the required position and orientation.

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

The Vehicle Position and Attitude control task control respectively the position and the attitude of the vehicle in order to make them converge to a desired goal position. Since there are in principle one goal for the position and another for the attitude, there are two Jacobian matrices, one for each control problem.
Since the goal position and goal attitude are expressed with respect to the world frame, our task reference vector $\dot{x} = J\dot{y}$ needs to be projected on the world frame too.
For this reason, since the control vector $\dot{y} = \begin{bmatrix}\dot{q} & \dot{v} & \dot{\omega}\end{bmatrix}^\top$ is projected in the vehicle frame, the two Jacobian matrices will be:

```
uvms.Jv_pos = [zeros(3, 7), uvms.wTv(1:3, 1:3), zeros(3,3)];
uvms.Jv_att = [zeros(3, 7), zeros(3, 3), uvms.wTv(1:3, 1:3)];
```

The task reference vector must be in the form $\dot{\bar{x}} = \lambda(\bar{x} - x), \lambda > 0$, where the reference value $\bar{x}$ represents the desired value and x is the current value.
The two task reference vectors are computed in this way:

```
[ang, lin] = CartError(uvms.wTg_v , uvms.wTv);
uvms.xdot.v_pos = Saturate(0.5 * lin, 0.5);
uvms.xdot.v_att = Saturate(0.5 * ang, 0.5);
```

Where *uvms.wTg_v* and *uvms.wTv* are respectively the transformation matrix of the goal frame with respect to the world frame and the transformation matrix of the vehicle frame with respect to the world frame.

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.

In order to observe the robot behaviour we considered three cases:

- Case 1: Horizontal attitude disabled and original initial and target configurations as stated in the exercise description;
- Case 2: Horizontal attitude enabled, original initial configuration but different target one, which is the following

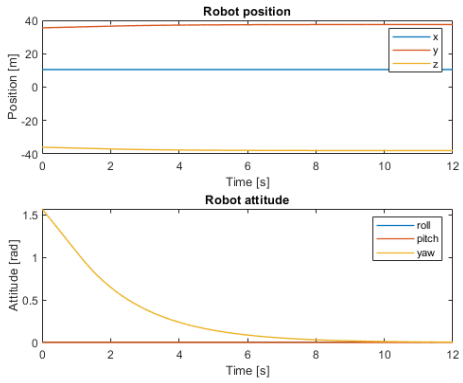
$$\begin{bmatrix}10.5 & 37.5 & -38 & \pi/3 & \pi/3 & 0\end{bmatrix}^\top$$

- Case 3: Horizontal attitude enabled and different initial and target configurations, which are the following

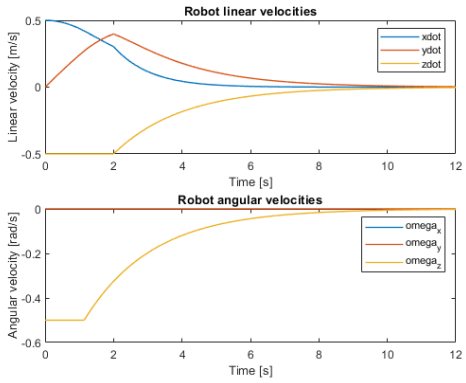
$$\begin{bmatrix}10.5 & 35.5 & -36 & \pi/3 & \pi/3 & \pi/2\end{bmatrix}^\top$$
$$\begin{bmatrix}10.5 & 37.5 & -38 & \pi/3 & \pi/3 & 0\end{bmatrix}^\top$$

Case 1

In this case the behavior is normal, the robot reaches the target position and attitude without problems.



(a) The robot moves in Y and Z and the yaw angle converges to zero.



(b) After an initial transient phase, all the velocities converge to zero.

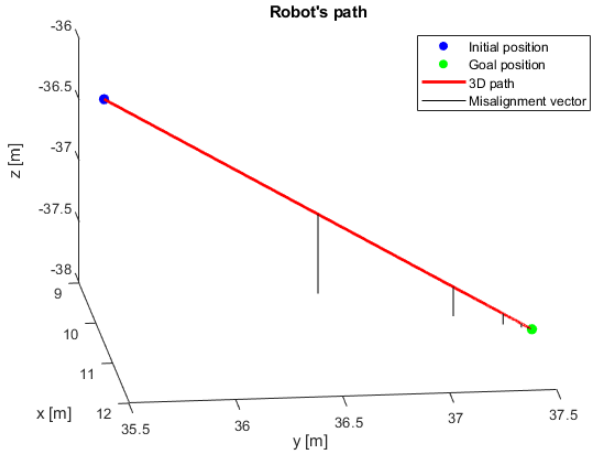
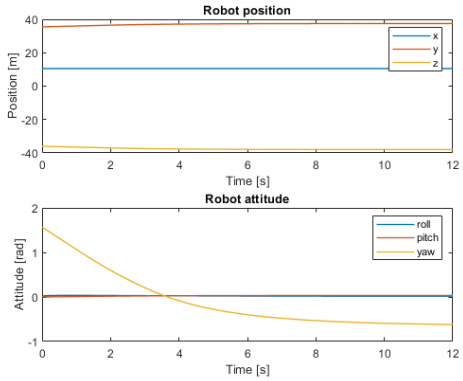


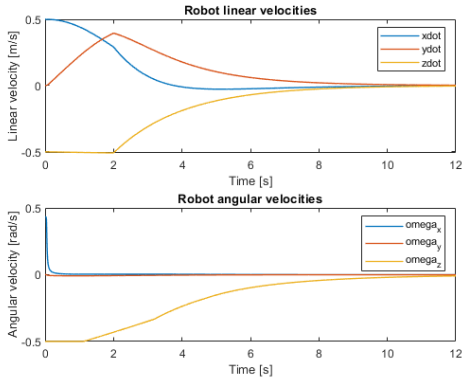
Figure 1: Here we can see that the robot reaches the goal position and that the misalignment vector shrinks towards zero.

Case 2

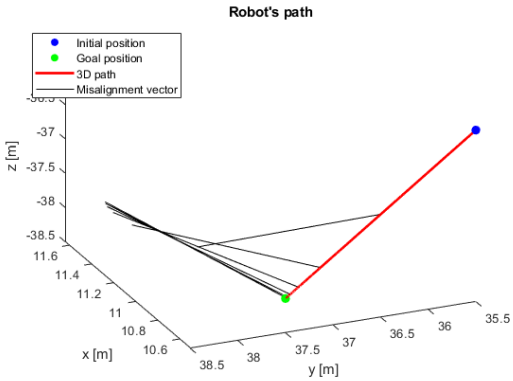
In this case the target configuration has roll and pitch angles both equal to $\pi/3$, so the Vehicle Position and Attitude task tries to rotate the robot in order to fulfill its objective, but, since the Horizontal Attitude task is enabled with higher priority, the robot can't reach the final configuration because it will try to stay as horizontal as possible. Nevertheless, the robot will remain a bit tilted since this task is of inequality type (it becomes inactive when the norm of the misalignment vector is less than about half a degree) and so it does not constrain the robot in a certain interval of tilting angle.



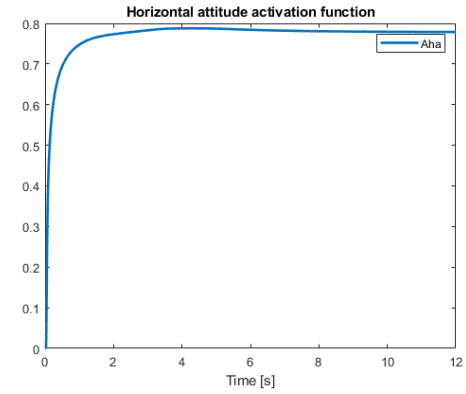
(a) The robot moves in the Y and Z directions, but the attitude doesn't converge to the desired one.



(b) The robot velocities converge to zero after a while.



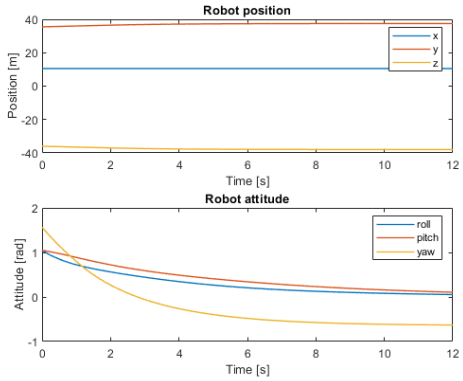
(c) The robot can reach the goal position, but the misalignment vector keeps growing.



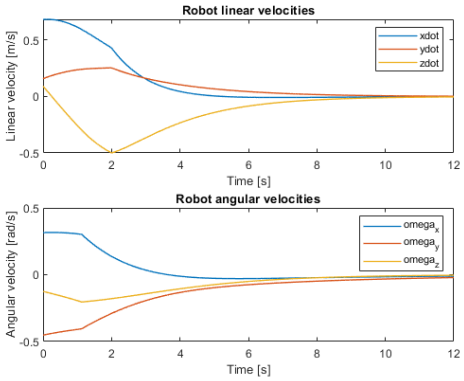
(d) The activation function of the Horizontal Attitude task almost immediately reaches an high value, but it can't go back to zero since the robot has conflicting goals.

Case 3

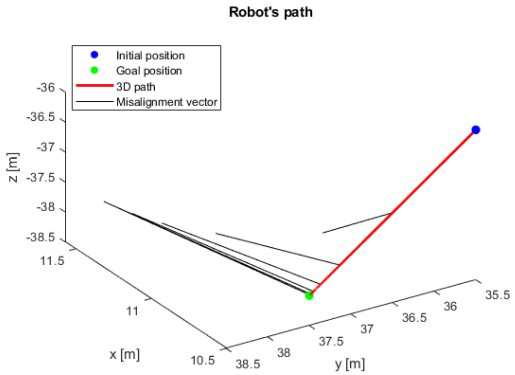
Now both the initial configuration and the target one have pitch and roll angles equal to $\pi/3$: if the Horizontal Attitude wasn't enabled then the robot would only change its yaw angle and move linearly towards the goal, since the initial and the target pitch and roll angles are the same. In this case Horizontal Attitude is enabled, so this task tries to rotate the robot, whereas the Vehicle Attitude one tries to keep the robot from rotating. The robot will reach the target position, but since the Horizontal Attitude task has higher priority, it will prevent the robot from reaching the target attitude.



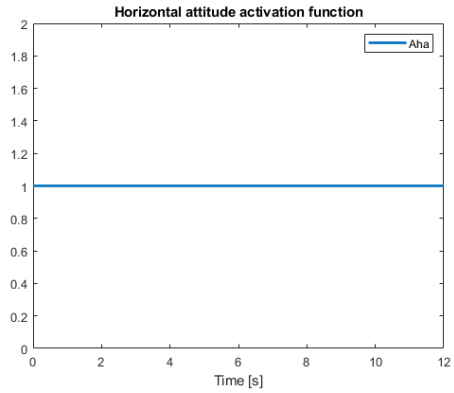
(e) We can see that the pitch and roll angles start from $\pi/3$ and converge near to 0: the robot moves in Y and Z, reaching the goal position.



(f) The robot rotates quite fast in the first 2-3 seconds, but then the angular velocities converge to zero. We can see a different $\dot{z}$ with respect to other cases since the robot is initially tilted.



(g) The robot reaches the goal position, but the misalignment vector continues to grow.



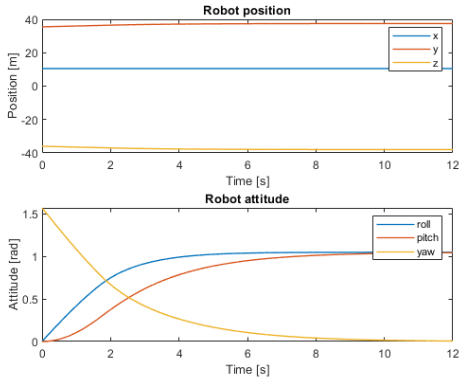
(h) The activation function is always 1 since the norm of the misalignment vector is always bigger than the maximum allowed value.

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

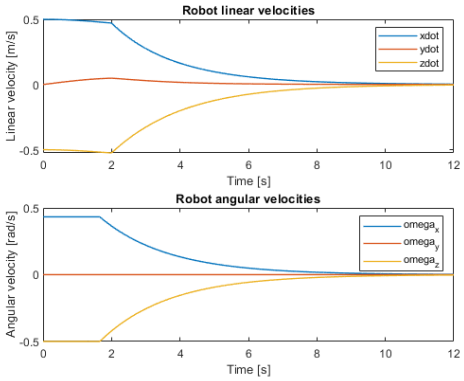
Here we consider Case 2 and Case 3 which have been mentioned in the previous question, but with the Horizontal Attitude task at the bottom of the hierarchy.

Case 2

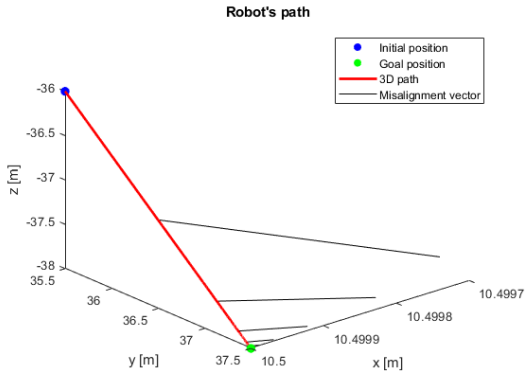
In the previous question, the robot had conflicting goals and the safety one was considered the most important: now we still have conflicting goals, but the most important one is that the robot should reach the target configuration. Since the Horizontal Attitude task won't have much effect on the robot, it will reach the target configuration without any problem.



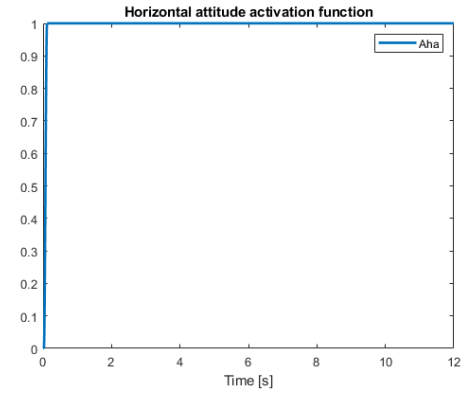
(i) The robot configuration converges to the target one.



(j) The robot velocities all converge to zero after a brief transient period.



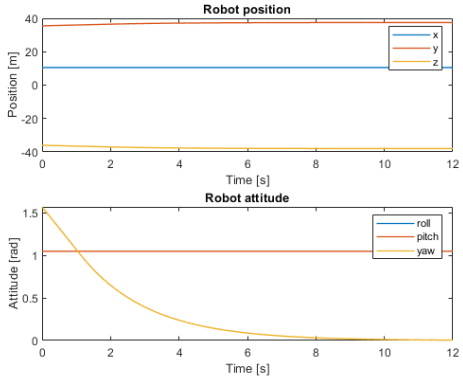
(k) The robot reaches the target position and the misalignment vector shrinks towards zero.



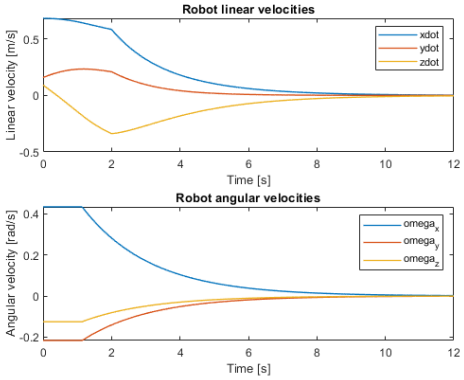
(l) The value of the activation value is almost always 1 since the task doesn't have effect on the robot.

Case 3

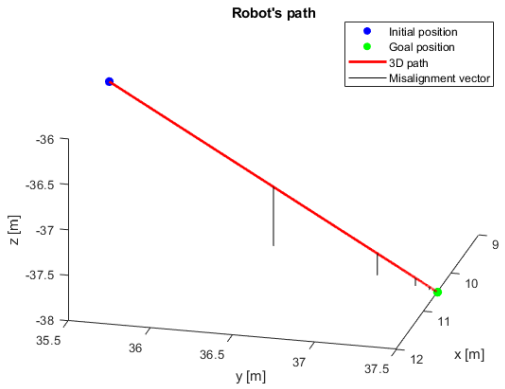
In the previous question, the Horizontal Attitude task prevented the robot to reach the target attitude: now this task won't have effect, allowing the robot to reach the target configuration.



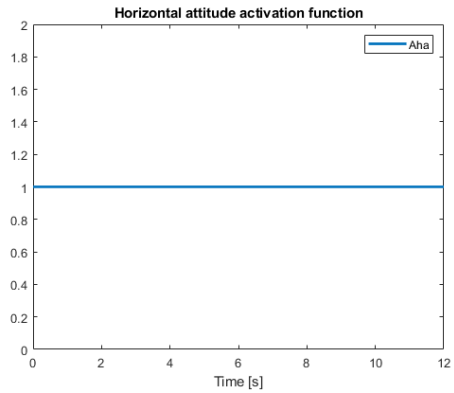
(m) The robot moves in Y and Z and the yaw angle correctly converges to zero.



(n) The robot velocities converge to zero after a brief transient period.



(o) The robot reaches the target position and the misalignment vector shrinks towards zero.



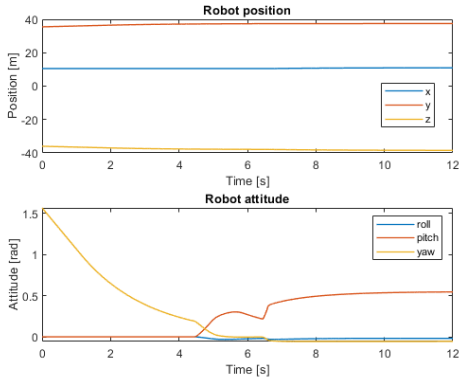
(p) The value of the activation function is always 1 since its corresponding task has the lowest priority.

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

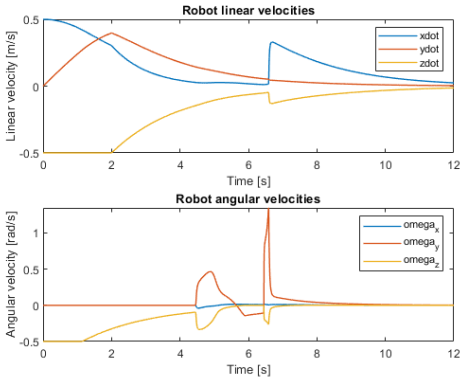
In the previous exercises we observed the robot's behaviour when the Tool Position control task was disabled: the end-effector doesn't move since it doesn't have a target configuration to reach, so the robot reaches its defined target configuration as expected.
When we try to enable with highest priority the Tool Position task having tool's target configuration

$$\begin{bmatrix} 12.2025 & 37.3748 & -39.8860 & 0 & \pi & \pi/2 \end{bmatrix}^\top$$

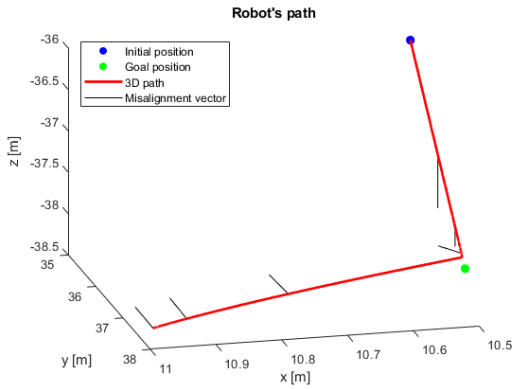
in the previously considered Case 1, it interferes with the Vehicle Position task by making the robot not reach its desired target configuration, as shown in the following figures.



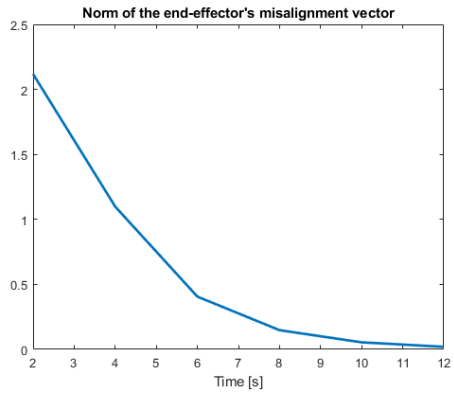
(q) The robot's configuration doesn't reach the desired one: the position is quite close, but the attitude is not.



(r) The robot's velocities spike at around 5 and 7 seconds due to the end-effector's motion.



(s) Here we can see better that the robot doesn't reach the target position and the misalignment vector doesn't shrink towards zero.



(t) The norm of the misalignment vector between the tool frame and the tool's target frame reaches zero.

When implementing a Safe Waypoint Navigation action the best setting should be to disable the Tool Position task since it may interfere with other goals or worse, may present as a risk to the robot's safety: in fact, the end-effector motion may result in accidental collisions between either the vehicle or the manipulator arm and the environment, which can in turn break the system.
For this reason, the final hierarchy of tasks which make up the Safe Waypoint Navigation action is the following:

Table 1: Hierarchy table for the Safe Waypoint Navigation action $\mathcal{A}_1$.

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

1.2 Adding a safety minimum altitude control objective

Initialize the vehicle at the position:

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

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

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

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

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

We added the Minimum Altitude control task to the previous hierarchy, obtaining the following updated table. Since this new task is a safety one, it must have the highest priority: we put it at priority level 1 but it could also be put at priority level 2 just under the Horizontal Attitude task. In this case, the priority of safety tasks is not important as long as they have higher priority than the other tasks (prerequisite, action defining and optimization ones). This is because, as we've seen in question 1.1.3, if a safety task has a lower priority than the action-defining task(s) then it won't have a huge effect on the robot's behaviour or any effect at all.

Table 2: Updated hierarchy table for the Safe Waypoint Navigation action $\mathcal{A}_1$.

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

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

Our variable of interest a is the altitude of the vehicle, with respect to the sea floor. So we want to find the jacobian relationship between the control vector $\dot{y}$ and the time derivative of our variable of interest $\dot{a}$.
Let's define the following vector

$$^w\boldsymbol{d} = a \cdot^w \boldsymbol{k} =^w O_v - Q$$

as the one representing the altitude of the robot with respect to the world frame:$^w\boldsymbol{k}$ is the versor along the z-axis of the world frame, ^wO_v is the origin of the vehicle frame and Q is a point on the sea floor for which the previous equality holds.
We have that the derivative of the altitude will be

$$\begin{gathered}\dot{a} =^w \boldsymbol{k} \cdot D_w(^w\boldsymbol{d}) \\ D_w(^w\boldsymbol{d}) = D_w(^wO_v) - D_wQ = D_w(^wO_v) =^w \boldsymbol{v}_{v/w} \\ \dot{a} =^w \boldsymbol{k} \cdot^w \boldsymbol{v}_{v/w}\end{gathered}$$

where D_w is the derivative operator with respect to the world frame and $^w\boldsymbol{v}_{v/w}$ is the velocity of the vehicle with respect to the world frame projected on the world frame. The equality $D_wQ = 0$ comes from the fact that the point Q doesn't move with respect to the world frame.
Thus, the jacobian matrix will be

```
uvms.JminAlt = k' * [zeros(3, 7), uvms.wTv(1:3, 1:3), zeros(3, 3)];
```

where k' is $^w\boldsymbol{k}$ transposed and $uvms.wTv(1:3, 1:3)$ is the rotation matrix of the vehicle frame with respect to the world frame.
Since the task is of inequality type, we want the robot to reach an altitude belonging to the interval for which the value of the activation function is 0: we established that the activation function is 0 for altitudes greater than 1.5 m and is 1 for altitudes lower than 1 m.
The formula for the task reference is the following

```
uvms.xdot.minAlt =
Saturate(0.5*((uvms.min_offset + uvms.range_offset) - uvms.w_a), 0.5);
```

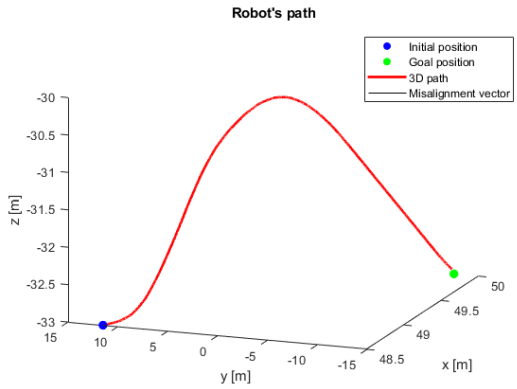
where $uvms.min\_offset = 1m$, $uvms.range\_offset = 0.5m$ and $uvms.w\_a$ is the altitude of the vehicle.

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

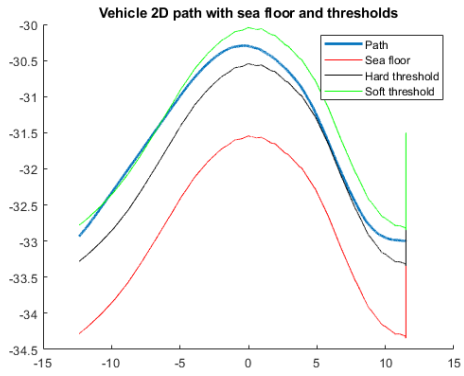
We tried setting the minimum altitude to the three requested values, obtaining the following results:

Minimum altitude equal to 1 m

The vehicle's initial and target positions are at an altitude greater than 1 m, so it reaches its goal by moving over the small hill on the sea floor as we can clearly see from the second image.



(u) The vehicle reaches the target position while respecting the minimum altitude requirement.



(v) We can see that the vehicle's path doesn't violate the hard threshold. Note that the vehicle's initial position is on the right side of the plot.

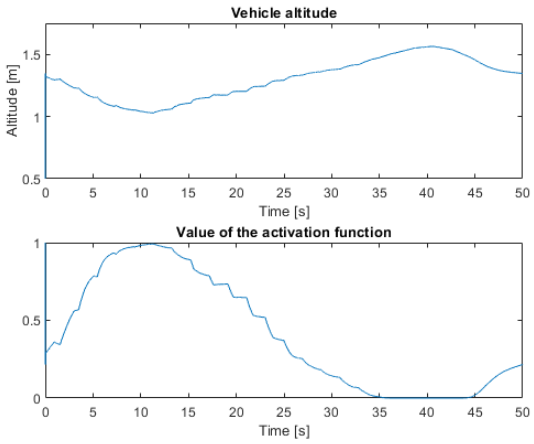
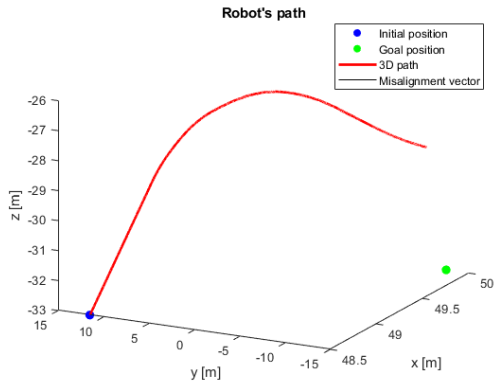


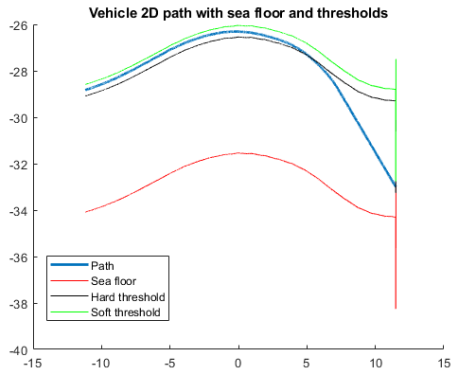
Figure 2: The altitude of the vehicle goes really close to 1 m at around 10 seconds: correspondingly, the value of the activation function becomes almost 1.

Minimum altitude equal to 5 m

Since the initial position of the vehicle is at about 1.3 m of altitude, in the first 10 seconds it moves upwards to respect the minimum altitude constraint. Once it reaches the quote of 5 m, the vehicle keeps on moving to reach the target position, which is also at about 1.3 m of altitude: for this reason, the vehicle will reach the x and y coordinates of the target position, but it will remain at around 5 m of altitude.



(a) The vehicle doesn't reach the target position.



(b) We can see that the vehicle moves upwards to respect the minimum altitude constraint.

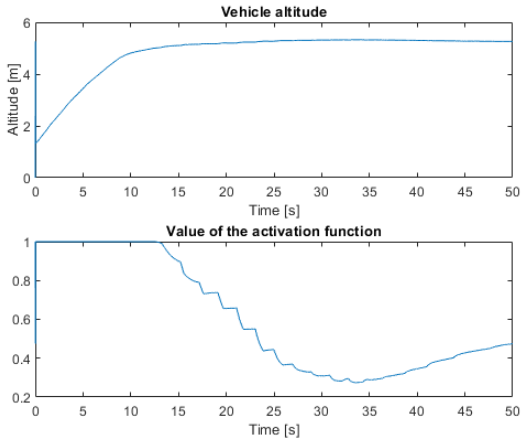
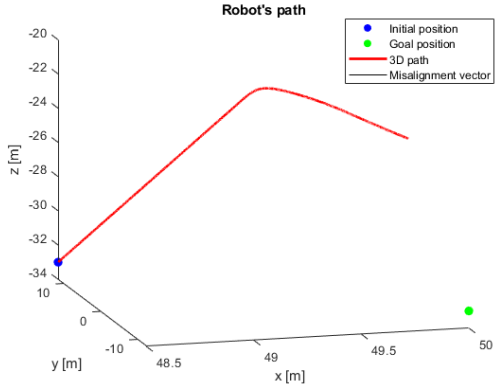


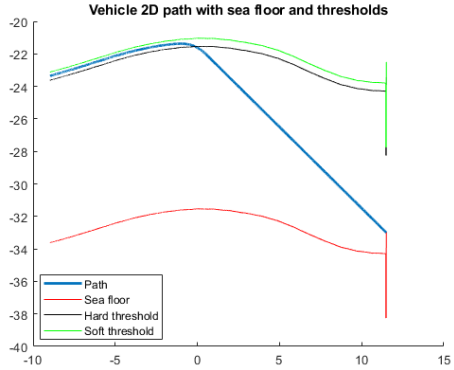
Figure 3: The altitude reaches 5 m in about 10 seconds, after which the value of the activation function starts getting down.

Minimum altitude equal to 10 m

We can make the same considerations as the previous case, the only thing that changes is that the vehicle will take more time to reach the requested minimum altitude. Again, the vehicle will not reach its exact target position.



(a) The vehicle doesn't reach its target position.



(b) The vehicle takes more time to reach the altitude of 10 m.

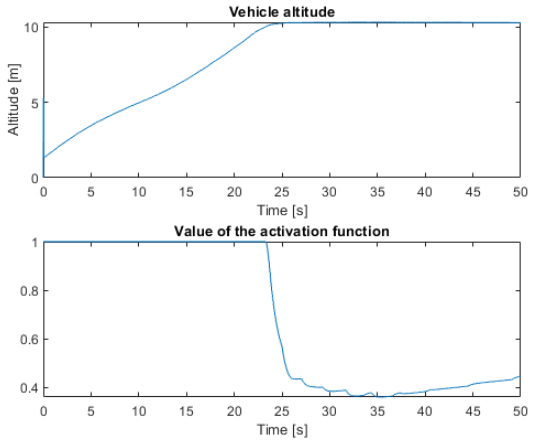


Figure 4: As soon as the vehicle reaches the minimum altitude, the value of the activation function starts getting down.

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

Our Matlab code used to compute the altitude measurement is the following

```
k = [0, 0, 1]';
v_d = [0, 0, uvms.sensorDistance]';
w_d = uvms.wTv(1:3, 1:3) * v_d;
uvms.w_a = k' * w_d;
```

To obtain the vehicle altitude *uvms.w_a* we took the vector representing the sensor distance in the vehicle frame, project it into the world frame and then perform the dot product to get the projection of *w_d* onto the z-axis of the world frame.
The altitude is not simply the sensor distance measurement since the vehicle can be in a tilted position, which results in an sensor distance value greater than the actual altitude: but, by computing the altitude via dot product we make the implicit assumption that the sea floor is flat. This results from the definition of the dot product itself. Obviously, this assumption won't always hold: there might be areas in which the sea floor is almost flat, but in most cases this is not true.
Moreover, this assumption is pessimistic: in the ideal case the computed altitude will be equal to the actual one, but often we're overestimating how close we are to the sea floor; this isn't bad per se since this is a safety task, but it might happen that the vehicle could get closer to the sea floor without actually violating the minimum altitude requirement but it isn't allowed due to the altitude overestimation. Finally, we also assume of being able to know the robot orientation with respect to the reference frame at any time instant. Even if this is often the case, it is still an underlying assumption of this task.

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 to implement the Landing Action. Comment how you choose the priority level for the altitude control task.

Table 3: Task hierarchy for the action landing.

- Action ($\mathcal{A}_1$) : Landing (Vehicle Altitude control)

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

The basic Landing Action is only composed by two tasks: one safety task which is the Horizontal Attitude, and one action defining task, the Vehicle Altitude Control.
The priority level of the horizontal attitude task has to be higher, because it is a safety task, while the other one is an action defining task, so it must have a lower priority.

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

The Jacobian relationship is exactly the same as the one for minimum altitude control, because in both case we want to control the altitude of the vehicle from the sea floor, even if the objective is different. For this reason we will not explain again the details, which can be found in section **1.2.2**.
The Jacobian matrix obtained is:

```
uvms.Jlanding = k' * [zeros(3, 7), uvms.wTv(1:3, 1:3), zeros(3, 3)];
```

The task reference, instead, is different, because this time the objective is to land on the sea floor, so the vehicle needs to reach the altitude zero.
The variable controlled is the altitude from the sea floor ($uvms.w\_a$), and the desired value will be obviously 0.
The task reference obtained is the following:

```
uvms.xdot.landing = Saturate(0.5 * (0 - uvms.w_a), 0.5)
```

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

This task is different with respect to the minimum altitude control task because they belong to two different categories of tasks: the minimum altitude control is a safety task, that is necessary, for example, during safe navigation to ensure that the robot stays at a safe altitude from the sea floor; instead, this task belongs to the action defining category, because it represents the final scope of the action.
Moreover, they are different also because the minimum altitude control task objective is of inequality type, while the landing task one is of equality type, which means that their activation functions will be different.

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.

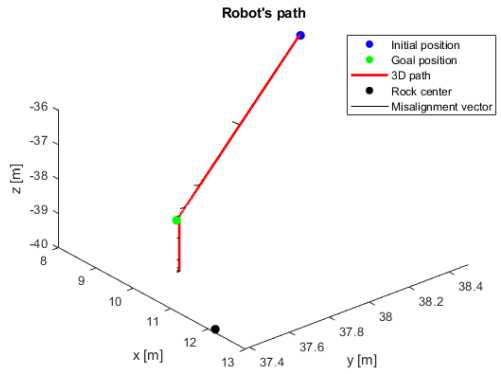
2.2.1 Q1: Report the unified hierarchy of tasks used and their priorities.

Table 4: Task hierarchy for the set of actions used to move the vehicle to a goal and then land.

- Action ($\mathcal{A}_1$) : Landing (Vehicle Altitude control)
- Action ($\mathcal{A}_2$) : Safe Waypoint Navigation

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$
Horizontal attitude	I	2	1
Vehicle position	E	3	
Vehicle attitude	E	4	
Vehicle minimum altitude	I	1	
Vehicle altitude control	E		2

(a) Position of the vehicle when landed on the seafloor.



(b) Path of the vehicle and misalignment vector

Figure 5: Behaviour of the robot performing the safe navigation and landing.

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

In order to create a system capable of transitioning from one action to the other, we decided to think about a procedure that is not specific for this problem, but that solves the transition between actions once and for all.
The first thing to do is to make a list of all the activation functions of the tasks that will be used, and enumerate them such that each task is identified by a univocal number, corresponding to the position of the task in the list. The priority level of the tasks will not be influenced by this ordering.

```
mission.activationFunctions = {
uvms.A.t,...
```

```
uvms.A.ha,...
uvms.A.v_pos,...
uvms.A.v_att,...
uvms.A.minAlt,...
uvms.A.landing }
```

The phase time is initialized and is be updated at each iteration, until the action is concluded. The first action to be performed is stored in *mission.currentAction*.
This is the definition of the actions "Safe Navigation" and "Landing":

```
mission.actionLanding = [2,6];
mission.actionSafeNavigation = [2, 3, 4, 5];
mission.currentAction = mission.actionSafeNavigation;
```

The phase changes after certain conditions are achieved. For the safe navigation action this happens when the linear distance between the vehicle and the goal is below a certain threshold.
At this point the phase time is reset to zero, the current action is saved in a variable called *previousAction*, and the next action (*mission.actionLanding*) becomes the current action.

```
mission.previousAction = mission.currentAction;
mission.currentAction = mission.actionLanding;
```

The activation function of the tasks is managed in this way:

- If a task is present in the list of tasks of the current action and it is not present in the previous action, its activation function is multiplied by an increasing bell shaped function.
- If a task is present in the list of tasks of the current action and it is also present in the previous action, its activation function is multiplied by an identity matrix of the corresponding dimension.
- If the task is not present in the list of tasks of the current action and it is present in the previous action, its activation function is multiplied by an decreasing bell shaped function.
- If the task is not present in the list of tasks of the current action and it is not present in the previous action, its activation function is multiplied by a matrix of zeros of the corresponding dimension.

The bell shaped functions are designed to reach their maximum (or minimum) value after two seconds from the beginning of each phase, so that a smooth transition between the actions is ensured.

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 hierarchy of tasks used and their priorities in each action. Comment the behaviour.

Table 5: Task hierarchy for the set of actions used to move the vehicle near the target and then align it.

- Action 1 ($\mathcal{A}_1$) : Safe Navigation
- Action 2 ($\mathcal{A}_2$) : Aligned Landing

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

The vehicle initially starts from a point quite high with respect to the ground and some meters away from the target ($\begin{bmatrix}8.5 & 38.5 & -36 & 0 & -0.06 & 0.5\end{bmatrix}^\top$)
The first action performed is the Safe Navigation, which leads the vehicle to move to a specified point using two safety tasks, which have the top priority, the Minimum Altitude Task and the Horizontal Attitude Task, and two tasks to move the vehicle to the goal, the Vehicle Position and Vehicle Attitude Tasks.
When the linear distance between the vehicle and the goal position ($\begin{bmatrix}10.5 & 37.5 & -38 & 0 & -0.06 & 0.5\end{bmatrix}^\top$) falls under $0.1m$, the action changes and the vehicle begins the landing. This second action is called Aligned Landing and it is composed by three tasks, one safety task which is the Horizontal Attitude, and the two tasks for landing in front of the rock: Vehicle Altitude Control that makes the vehicle land on the sea floor, and the Alignment to Target, which makes the vehicle x axis align to the projection on the inertial horizontal plane of the unit vector joining the vehicle frame to the nodule frame.
The vehicle successfully manages to land right in front of the rock and with the expected alignment, as shown in Figure 6.

Figure 6: Simulation showing the vehicle after landing in front of the rock.

Here the path that the robot performs is shown in a 3D plot, joint with the misalignment vector between the x axis of the robot and the projection on the inertial horizontal plane of the unit vector joining the vehicle frame to the rock frame.

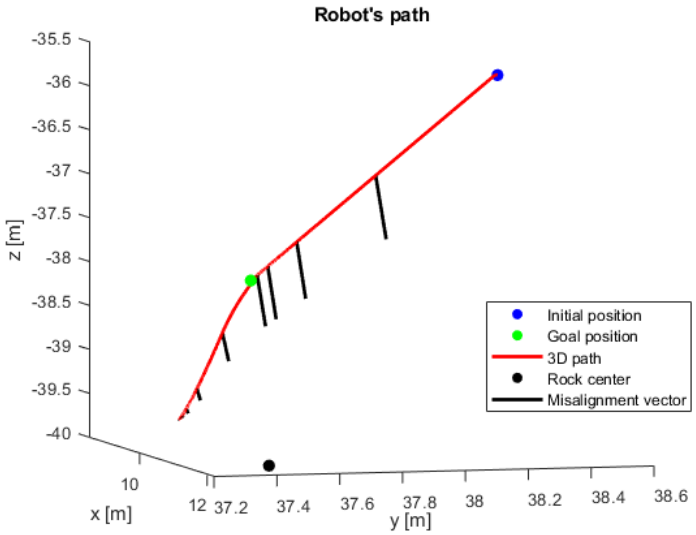


Figure 7: Plot of the Vehicle path.

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

The first thing to do is to calculate the vector to which the vehicle axis have to align, that is the distance between the vehicle frame origin and the target (in the world frame), projected on the horizontal plane of the inertial frame.

$$^{w}d = ^{w}P - ^{w}O_v$$

$$^{w}d_{proj} = ^{w}d - ((^{w}d \cdot ^{w}k_w) \cdot ^{w}k_w)$$

Then the misalignment vector (ρ) between the x-axis of the vehicle and the d_{proj} versor ($n_{d_{proj}} = \frac{d_{proj}}{\|d_{proj}\|}$) is calculated using the Reduced Versor Lemma, after having projected both versors on the vehicle frame.

$$^{v}\boldsymbol{\rho} = \theta \boldsymbol{n}$$

Once obtained the misalignment vector ${}^v\boldsymbol{\rho}$ we need to evaluate its derivative. Thus, we refer to the derivative of a rotation vector:

$$D_v({}^v\boldsymbol{\rho}) = \boldsymbol{nn} \cdot \boldsymbol{\omega}_{n_{d_{proj}}/i_v} + \theta N(\theta)\boldsymbol{\omega}_{n_{d_{proj}}/i_v}$$

Expressing $\dot{\rho}$ with respect to the control variables, we get the following relationship:

$$D_v({}^v\boldsymbol{\rho}) = \left(\boldsymbol{nn}^T - \frac{\theta}{\sin(\theta)}\left[\boldsymbol{a}\wedge\right]\left[\boldsymbol{b}\wedge\right]\left(\boldsymbol{I}_{3x3} - \boldsymbol{nn}^T\right)\right)\begin{bmatrix}\boldsymbol{0}_{[3x7]} & -\frac{1}{\|{}^v d_{proj}\|^2}\left[{}^v d_{proj}\wedge\right] & -\boldsymbol{I}_{[3x3]}\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{q}}\\ \boldsymbol{v}\\ \boldsymbol{\omega}\end{bmatrix}$$

Since we're only interested in the derivative of θ, we obtain the Jacobian matrix:

$$\dot{\theta} = J_{ha}\dot{\boldsymbol{y}}$$

$$\dot{\theta} = n \cdot \begin{bmatrix}\boldsymbol{0}_{[3x7]} & -\frac{1}{\|{}^v d_{proj}\|^2}\left[{}^v d_{proj}\wedge\right] & -\boldsymbol{I}_{[3x3]}\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{q}}\\ \boldsymbol{v}\\ \boldsymbol{\omega}\end{bmatrix}$$

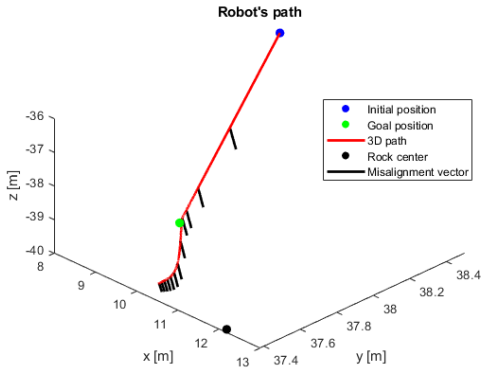
The task reference is computed using the norm of the misalignment vector, θ. The desired value in this case is 0, because we want the two vectors to be aligned.

```
uvms.xdot.horAlign = Saturate(0.5 * (0 - uvms.theta), 0.5);
```

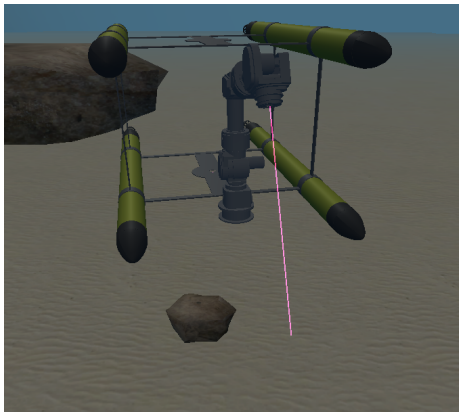
3.1.3 Q3: Try changing the gain of the alignment task. Try at least three different values, where one is very small. What is the observed behaviour? Could you devise a solution that is gain-independent guaranteeing that the landing is accomplished aligned to the target?

The initial alignment task gain is saturated at 0.5, then we tried to change its value with four different ones: 0.05, 0.2, 0.4 ,0.7.

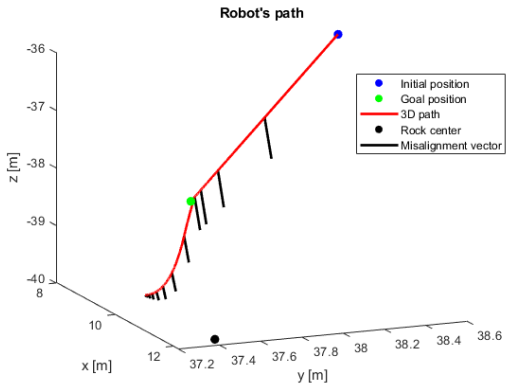
(a) Position of the vehicle when it approches the ground with gain = 0.05



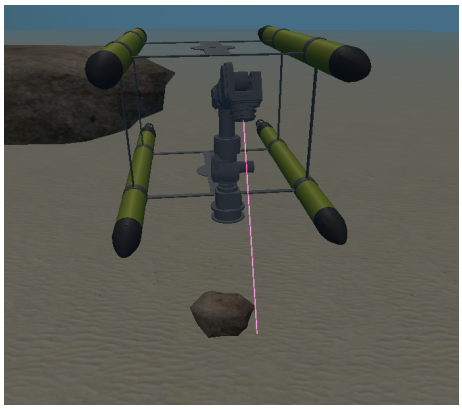
(b) Path of the vehicle and misalignment vector with gain = 0.05



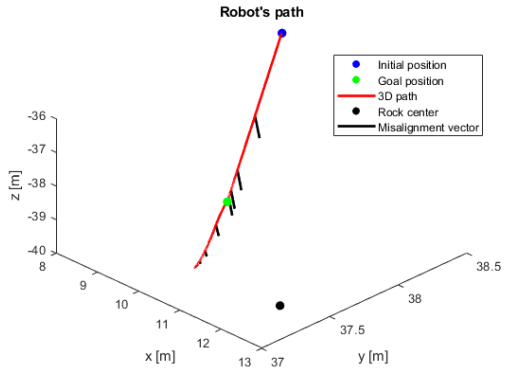
(c) Position of the vehicle when it approches the ground with gain = 0.2



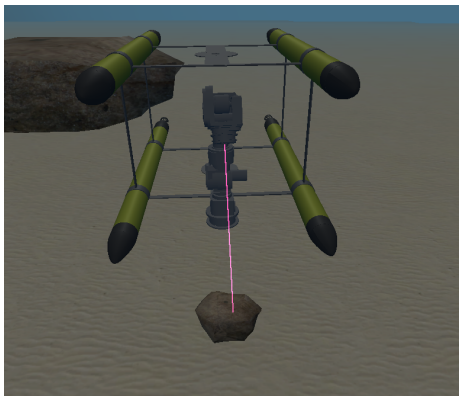
(d) Path of the vehicle and misalignment vector with gain = 0.2



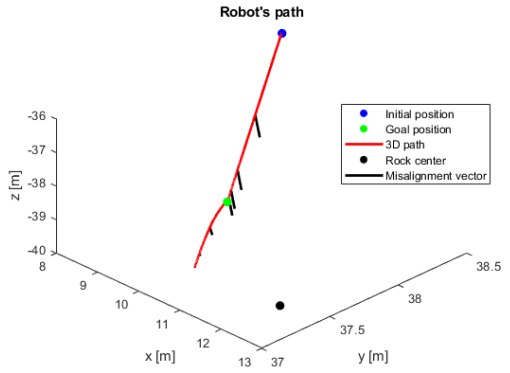
(e) Position of the vehicle when it approches the ground with gain = 0.4



(f) Path of the vehicle and misalignment vector with gain = 0.4



(g) Position of the vehicle when it approches the ground with gain = 0.7



(h) Path of the vehicle and misalignment vector with gain = 0.7

The behaviour of the vehicle changes when the gain value is modified: the smaller the gain, the harder it is for the vehicle to reach the correct alignment with d_{proj}. As it can be seen in the images above, while the time used by the robot to reach the sea floor remains the same, increasing the gain makes it reach different configurations when approaching the sea floor.
For gain values smaller than about 0.2, the misalignment vector fails to become a null vector, as can be seen in images (b) and (d), where the vehicle is not able to align to the target in time. In the other two cases instead the vehicle manages to align in time to the target, even if, for gain = 0.4, some

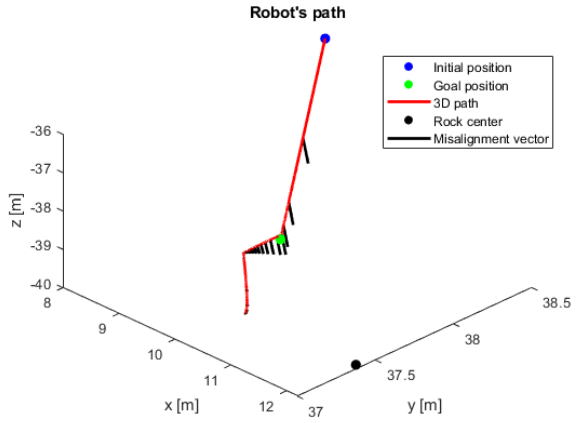
alignment error is still present.
This shows that this solution is heavily gain-dependent, so we need to think about a solution to ensure that the alignment is always achieved. What we thought was to add a new Action in between the two actual ones that is activated when the vehicle reaches the goal position for the Safe Navigation action. This new Action is composed by two tasks, horizontal attitude for safety and the horizontal aligning to the target. The vehicle doesn't land until the module of the misalignment vector, θ, becomes lower than 0.02 *rad*. The table with the new action is detailed below.

Table 6: Task hierarchy for the set of actions used to create a gain-independent alignment action.

- Action 1 ($\mathcal{A}_1$) : Safe Navigation
- Action 2 ($\mathcal{A}_2$) : Aligning
- Action 3 ($\mathcal{A}_2$) : Aligned Landing

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
Horizontal Attitude	I	2	1	1
Vehicle Position	E	3		
Vehicle Attitude	E	4		
Vehicle Minimum Altitude	I	1		
Vehicle Altitude Control	E			3
Alignment to Target	E		2	2

Then, the vehicle start the landing, but the misalignment vector between its x-axis and $n_{d_{proj}}$ is already very close to zero, so it is guaranteed that it will land aligned to the target, independently from the gain of the alignment task.



(i) Robot path and misalignment vector with gain = 0.1 (gain independent)

The behaviour obtained shows that the vehicle manages to land aligned with the target even if the gain value is very low.

3.1.4 Q4: After the landing is accomplished, what happens if you try to move the end-effector? Is the distance to the nodule sufficient to reach it with the end-effector? Comment the observed behaviour. If, after landing, the nodule is not in the manipulator's workspace, how would you solve this problem to guarantee it?

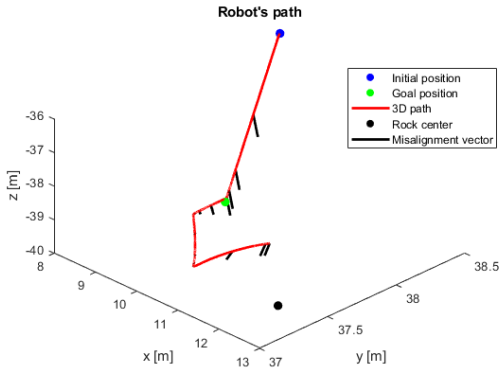
In order to move the end effector after landing is accomplished, we need to introduce a new action, that we called Grasp Object, that will be performed after the previous action is concluded, that is,

when the vehicle altitude from the sea floor is lower than $0.1m$. This action is composed by three tasks:

- the Horizontal Attitude, for safety
- the Vehicle Altitude Control, that is the same task present in the landing action, which is needed to let the vehicle remain at the same altitude from the sea floor
- the Tool Position Control, that is needed to move the end effector of the arm on the desired position

After the landing is accomplished, the robot arm starts moving towards the target and, since there is no constraint task on the vehicle velocities, the body shifts laterally and leans forward, following the arm movement, until the end effector reaches the goal position.

(j) Position of the vehicle after the grasping is completed



(k) Path of the vehicle and misalignment vector

So we observed that in this case the distance to the nodule is sufficient to reach it with the end-effector. Nevertheless it is not guaranteed to happen, so it is a good idea to add a new task that ensures that the target will be in the manipulator space when the landing is accomplished (operational prerequisite). Since we want to maintain the vehicle to a certain distance from the target in order to let it go stay in the manipulator workspace, we need to control the robot position on the inertial horizontal plane. To do so, the calculation to obtain the Jacobian matrix are the following:

```
uvms.xd = [1 0 0] * w_d_proj;
uvms.yd = [0 1 0] * w_d_proj;

if uvms.xd > 0
xd = [1 0 0];
else
xd = -[1 0 0];
end
if uvms.yd > 0
yd = [0 1 0];
else
yd = -[0 1 0];
end

w_Dxy = [xd; yd];
uvms.JdistGoal = [zeros(2, 7) -w_Dxy * uvms.wTv(1:3, 1:3) zeros(2, 3)];
```

where *w_d_proj* is the distance vector between the vehicle frame and the nodule projected on the horizontal inertial frame. The Jacobian will be different depending on the position of the vehicle with respect to the target: in fact a minus sign will be added to the rows of ${}^{w}D_{xy}$ if the corresponding projected distance vector component is negative. If we don't do that, depending on the situation the

vehicle may drift away instead of approaching the nodule.
The task reference is obtained using the x and y components of the distance between the vehicle frame and the target projected on the horizontal inertial frame.
The desired values are set as:

```
uvms.ensured_distance_x = 1;
uvms.ensured_distance_y = 1;
```

The task reference is the following:

```
uvms.xdot.distGoal =
[Saturate(0.5 * (uvms.ensured_distance_x - abs(uvms.xd)), 0.5) ;
 Saturate(0.5 * (uvms.ensured_distance_y - abs(uvms.yd)), 0.5)];
```

We take the absolute value of the variables x_d and y_d because we want to control only the distance from the target and we do not mind the sign of the two coordinates for this task. In order to test this new task we modified the vehicle goal position to $\begin{bmatrix}10.5 & 35.5 & -38 & 0 & -0.06 & 0.5\end{bmatrix}^\top$, so that we can see the task activate. The resulting path is shown in the Figure 8.

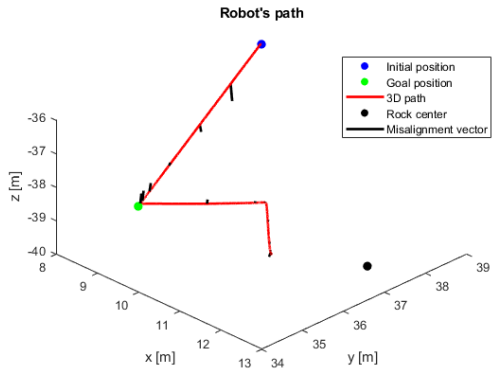


Figure 8: Path of the vehicle with the new task added ensuring the correct distance from the target is achieved before landing.

It is clear that the vehicle, before landing and while aligning to the target, gets closer to the rock position as intended.

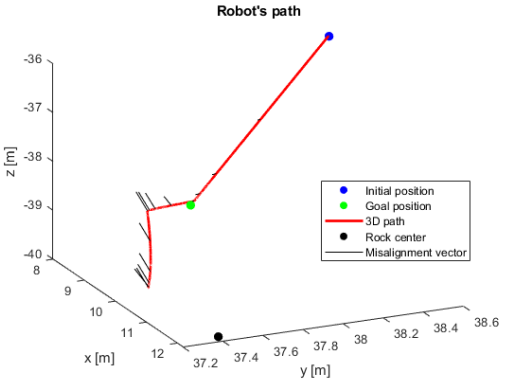
We are aware that there is no task assuring that the vehicle is distant enough from the target, but it can easily be added using the same task with an opposite activation function (decreasing instead of increasing).

4 Exercise 4: Implementing a Fixed-base Manipulation Action

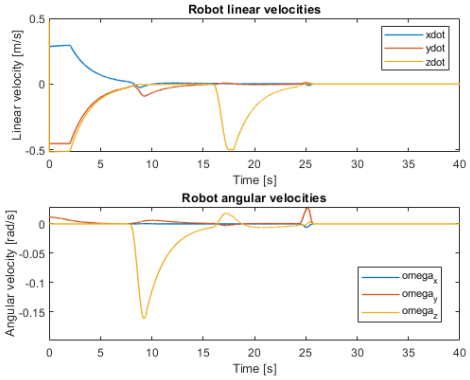
4.1 Adding non-reactive tasks

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.



(a) The vehicle reaches the target position, aligns to the rock, lands in front of it and finally moves the end-effector on top of it while remaining still.



(b) The vehicle conpletes the landing phase at about 24 seconds, then moves a bit due to the smooth transition between actions, and finally stays still until the end of the simulation.

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

Here is the hierarchy table which summarizes the four actions we implemented to fulfill the exercise's request. We can see that the Constraint task was placed in the fourth action (i.e. the last one to be performed) at the highest priority: this is mandatory because in this way we completely fix the vehicle's velocities to be 0 and the other lower priority tasks won't be able to change them. If we were to place this task at a lower priority level, higher priority ones would be able to change the vehicle's velocities.

Table 7: Hierarchy table defined to perform the four actions that make up the fixed-base manipulation objective.

- Action 1 ($\mathcal{A}_1$) : Safe Navigation
- Action 2 ($\mathcal{A}_2$) : Aligning to the target
- Action 3 ($\mathcal{A}_3$) : Landing
- Action 4 ($\mathcal{A}_4$) : Reach the Target with the End-Effector

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
Tool Position	E				5
Horizontal Attitude	I	2	2	1	2
Vehicle Position	E	3			
Vehicle Attitude	E	4			
Vehicle Minimum Altitude	I	1	1		
Vehicle Altitude Control	E			3	3
Alignment to Target	E		3	2	
Ensuring Distance from the Target	I		4	4	4
Constrain Velocities	E				1

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

The Jacobian relationship is really simple. Since the variables of interest are the vehicle's linear and angular velocities projected on the vehicle frame, to extract them from the control vector we need to multiply it by

```
uvms.Jconstraint = [zeros(6,7), eye(6,6)];
```

Moreover, we want the vehicle to be still: this means setting all velocities to 0. Thus, the task reference vector will be

```
uvms.xdot.constraint = zeros(6,1);
```

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

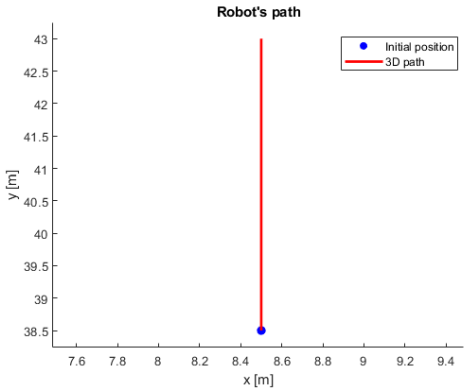
We considered two types of currents, a constant one and a sinusoidal one: in this way we can see what happens respectively near the sea floor and at the sea surface.
The constant current was defined in this way:

```
disturb = [0, 0.3, 0]';
uvms.p_dot(1:3) = uvms.vTw(1:3, 1:3)*disturb;
```

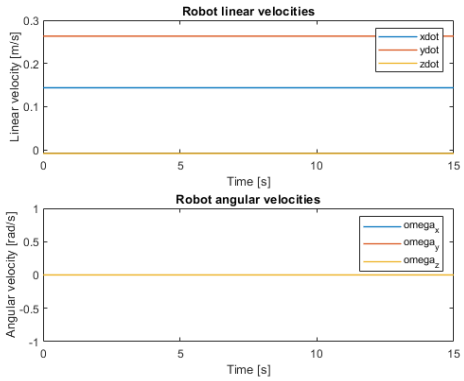
and the sinusoidal one in this way:

```
disturb_ang = [0, 1, 0]'; \\
disturb_ang = disturb_ang*0.35*sin(2*pi*0.5*t);
uvms.p_dot(4:6) = uvms.vTw(1:3,1:3)*disturb_ang;
```

We get the following results:



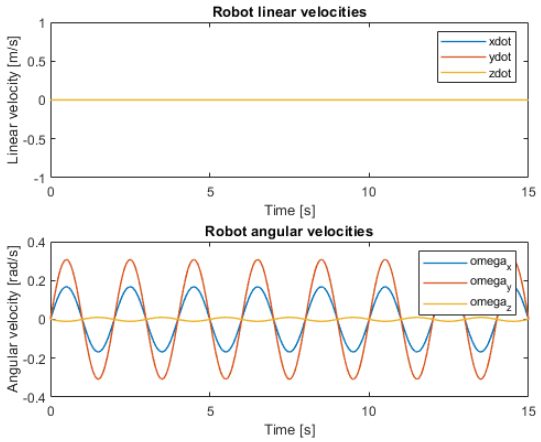
(c) Since the constant current is defined with respect to the world frame, the vehicle moves only in the Y direction.



(d) As we can see, the linear velocities of the vehicle are not 0.

The constraint task imposes that the reference for the vehicle velocities must be 0, which results in a control vector made of zeros: this vector is then modified by the constant current contribution. For this reason, the vehicle can't stay still and drifts away.

Figure 9: With a sinusoidal current we can see that the vehicle keeps rotating.

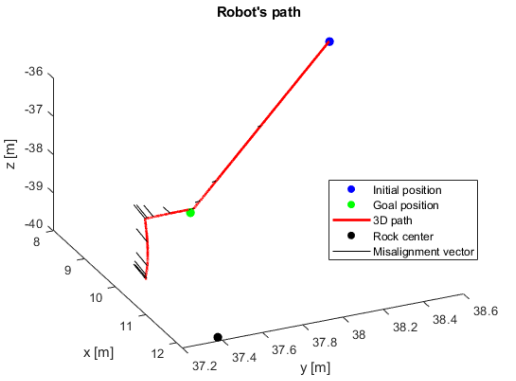


We can make the same considerations in this case: now the vehicle doesn't move linearly, but keeps rotating. Again, the constraint task can't keep the vehicle still.
There are two ways in which we could solve this problem: the first one involves creating position and attitude tasks of inequality type such that the vehicle doesn't move too much; equality tasks would result in a really high energy consumption. The other way would require sensors capable of measuring the contribution of the current in order to compensate it via a task designed for this purpose.

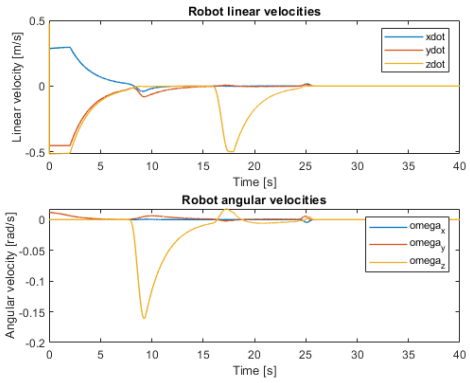
4.2 Adding a joint limit task

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.



(a) The vehicle reaches the goal position, proceeds to align to the target, lands on the sea floor and finally reaches the rock with the end-effector.



(b) The vehicle completes the landing again at about 24 seconds, moves a bit due to the smooth transition between actions and then it remains still.

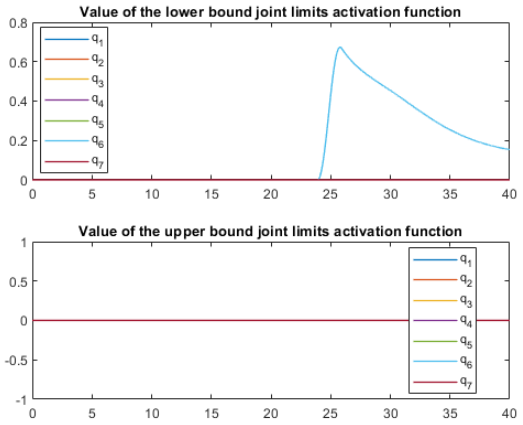


Figure 10: Here we can see that no joint reaches its maximum or minimum limits: all the activation functions' values are always 0 apart from joint q_6, which gets near its minimum limit.

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

Here is the hierarchy table used to implement the new mission. The only thing that changed with respect to the previous table presented in point 4.1.1 is the fourth action: now it also features the two new tasks, one to avoid the minimum joint limits and the other to avoid the maximum joint limits. Since they are both safety tasks, they need to have high priority, so we placed them just below the constraint task, which needs to stay at the top of the hierarchy.

Table 8: Hierarchy table defined to perform the four actions that make up the fixed-base manipulation objective.

- Action 1 ($\mathcal{A}_1$) : Safe Navigation
- Action 2 ($\mathcal{A}_2$) : Aligning to the target
- Action 3 ($\mathcal{A}_3$) : Landing
- Action 4 ($\mathcal{A}_4$) : Reach the Target with the End-Effector

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
Tool Position	E				7
Horizontal Attitude	I	2	2	1	4
Vehicle Position	E	3			
Vehicle Attitude	E	4			
Vehicle Minimum Altitude	I	1	1		
Vehicle Altitude Control	E			3	5
Alignment to Target	E		3	2	
Ensuring Distance from the Target	I		4	4	6
Constrain Velocities	E				1
Avoid Lower Bound Joint Limits	I				2
Avoid Upper Bound Joint Limits	I				3

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

Our variable of interest is the joint position vector so, since the vehicle velocities don't contribute to the manipulator's joints positions or velocities, the jacobian relationship will simply be:

```
uvms.JjointLimits = [eye(7), zeros(7, 6)];
```

The jacobian matrix is the same for both the maximum joint limits task and the minimum one, but the task references are different. The tasks are both of inequality type, so the task reference must drive the variable of interest to the first point where the activation function is equal to zero: for the minimum task this value is equal to the minimum joint limits vector *uvms.jlmin* plus an offset of 0.5 rad, for the maximum task it's the maximum joint limits vector *uvms.jlmax* minus the same offset. Thus:

```
offset = 0.5;
uvms.xdot.lbjointLimits = Saturate(0.5 * ((uvms.jlmin + offset) - uvms.q), 0.5);
uvms.xdot.ubjointLimits = Saturate(0.5 * ((uvms.jlmax - offset) - uvms.q), 0.5);
```

5 Exercise 5: Floating Manipulation

5.1 Adding an optimization control objective

Use the DexROV simulation for this exercise.

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

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

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

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

In order to accomplish the goal, we have chosen to implement an action with the following task. We first want to ensure the safety of the robot so the safety task is added with the highest priority, as shown in the table below. After having ensured the robot to stay consistently parallel to the seafloor

Table 9: Hierarchy table defined to perform the arm optimal configuration control.

- Action 1 ($\mathcal{A}_1$) : Tool Optimized Control

Task	Type	$\mathcal{A}_1$
Tool Position	E	3
Horizontal Attitude	I	1
Alignment to Target	E	2
Optimal Arm Shape	E	4

(or at least parallel to the O_{xy} plane of the inertial frame) we introduced the tasks to control the robot. First we want to ensure that the robot is aligned to the target, in order to enhance the arm effectiveness. Secondly, we want to actually control the robot and its arm together, so the tool control task is implemented. The optimization task is added with the lowest priority. In fact, it should not directly influence the behavior of the arm, but only change the configuration, if possible, in the remaining manifold.

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

For the task of optimization of the arm position, the jacobian accounts for the joints for which a preferred position is specified. Specifically, as the first four joints have a preferred position, the jacobian will map the vehicle state vector to the vector space of these four joints.

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{y}}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{I}_{[4x4]} & \boldsymbol{0}_{[4x9]} \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ v \\ \boldsymbol{\omega} \end{bmatrix}$$

For this task, the task reference was computed comparing the current joints position to their preferred, or optimal, value.

```
uvms.xdot.armPrefPos = Saturate(0.5 * (uvms.armPrefPos - uvms.q(1:4)), 0.5);
```

As equality task, we want to drive the arm as close al possible to the desired optimal configuration, acting on the remaining manifold of solution.

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

We now want to present the influence of this task on the robot behavior. We want to proceed showing the robot behavior without the task and then with the task active. In both cases, we will qualitatively present the result followed by a more rigorous mathematical description. In this last part, we will consider the joints value as well as the arm dexterity. The arm dexterity is given by the formula

$$\kappa = ||\boldsymbol{J}||_2 \left|\left|\boldsymbol{J}^{-1}\right|\right|_2$$

Where $\boldsymbol{J}$ is the arm jacobian and $\boldsymbol{J}^{-1}$ its inverse. In the case the jacobian is not a square matrix, the pseudo inverse is evaluated. The arm dexterity is an indices of the capability of the arm of perform motions around the current arm configuration. For sake of completeness, we want also to point out some lack of optimization of this task. In fact, the main goal is to keep the arm to an optimal configuration but, a part of the dexterity, also the arm manipulability should be of concern. In fact, the arm manipulability decreases when the arm is close to a singular configuration. This may deteriorate the arm performances around the considered configuration. To this aim, the arm manipulability is evaluated for both cases. The definition of the arm manipulability is the following.

$$\mu = \sqrt{|\boldsymbol{J}\boldsymbol{J}^T|}$$

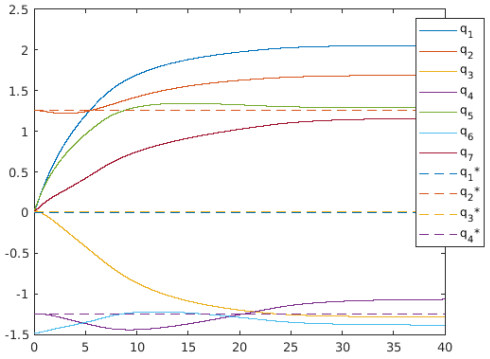
The difference in having or not this task are shown in the following.

Behavior without optimization

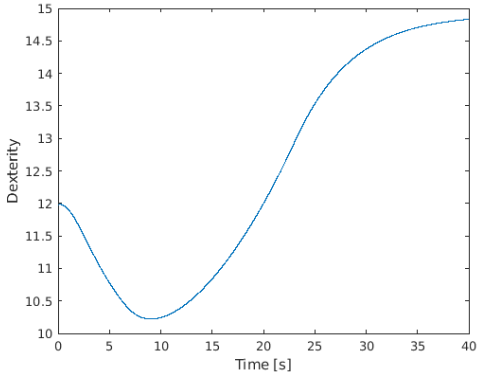
In this first part, we want to describe the robot behavior while trying to bring the arm close to the pipeline. In the following image, the final position where the robot stopped is shown.

Figure 11: Position of the vehicle and arm configuration once reached the pipeline (without optimization task).

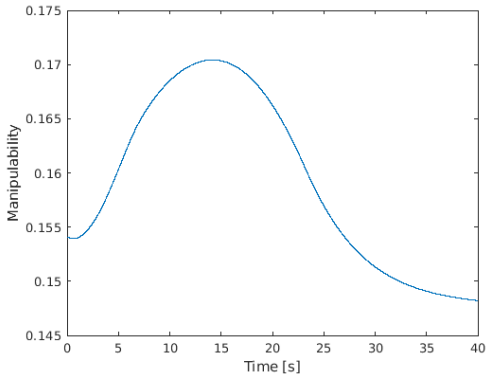
As we can see from the previous figure, the arm is very stretched and the configuration it has reached has a poor dexterity. As a result, a continuous operation on the pipeline is not possible, as it will require the movement of the vehicle. The joints are far from what is prescribed ad the optimal value. The figure below clearly shows that the values are not respected.



(a) Joint values without optimization task



(b) Dexerity of the arm with respect to time



(c) Manipulability of the arm with respect to time

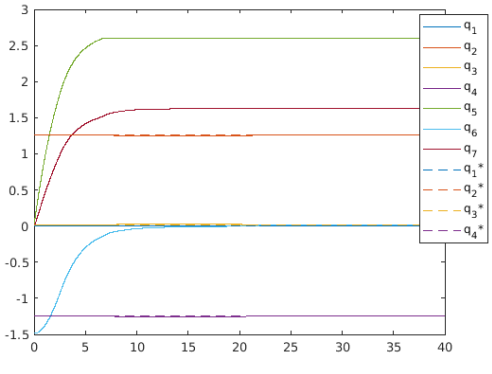
Furthermore, the effect of this lack in maintaining the optimal arm configuration is reflected in the arm dexterity. first it drops in the first time instants. After recovering the reduction, the overall dexterity remains lower than the one achieved when the task is active. Finally, for what concerns the arm manipulability, the following figure shows that the arm stays away from any singularity configuration, as its manipulability never decreases too much. However, this is mostly due to luck, as there no control on that. When the task is active, we can see that the arm is a dexterous position, allowing to perform effectively motions on the spot. This deduction is made by looking at the following figure. It is clear that the arm is in a configuration where no joint is close to one of its limit, so the arm can effectively perform a wide range of motions.

Behavior with optimization

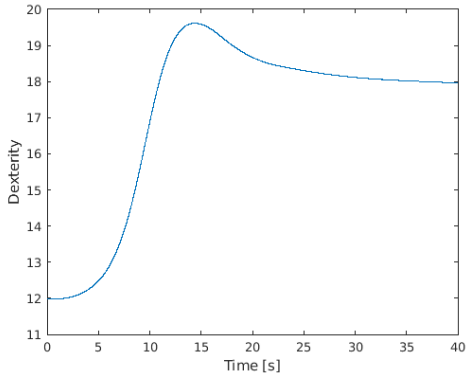
We now want to present the results obtained when the optimization task wa defined and activated. First, the effect can be qualitatively seen from the image below.

Figure 12: Position of the vehicle and arm configuration once reached the pipeline (with optimization task).

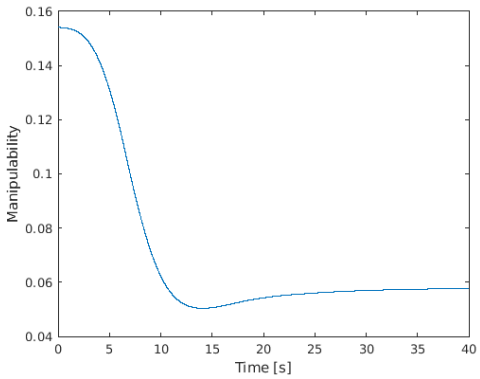
This behavior is desirable as we would like the arm to reach the desired position and also being able to perform tasks. As we can see from the following figures, the joints assume the desired position and the arm dexterity is satisfactory.



(a) Joint values without optimization task



(b) Dexerity of the arm with respect to time



(c) Manipulability of the arm with respect to time

Moreover, the arm dexterity is positively influenced by this task. In fact, the following figure reports that its dexterity doesn't drop as in the case without the task activated. Moreover, it increases

considerably from the initial value reaching an asymptote. However, as the arm manipulability is not considered, it decades to low value during the execution of the task. In other words, the arm is close to a singularity configuration once reached the target position. In order to counteract this lack of manipulability a prerequisite task should be defined to take care of it at an higher priority level.

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?

The table below reports the hierarchy of tasks for both actions. First, we have the safe navigation

Table 10: Hierarchy table defined to perform the arm optimal configuration control.

- Action 1 ($\mathcal{A}_1$) : Safe Navigation
- Action 2 ($\mathcal{A}_2$) : Tool Optimized Control

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$
Tool Position	E		5
Horizontal Attitude	I	1	3
Vehicle Position	E	2	
Vehicle Attitude	E	3	
Alignment to Target	E		4
Avoid Lower Bound Joint Limits	I		1
Avoid Upper Bound Joint Limits	I		2
Optimal Arm Shape	E		6

action, in this task the main goal is to bring the robot close to the tool target: the pipeline. For this action, the common tasks for safe navigation are included: vehicle position and attitude control and the horizontal attitude control. As always, the safety task comes first, so the horizontal attitude control has the highest priority. It is followed by the two tasks for the vehicle control. The safe navigation is needed in order to avoid possible obstacles and ensure that we are moving the arm only in a safe environment. Secondly, once the robot has reached a position slightly above the pipeline, the action bringing the arm on the pipeline is triggered. In this action, we first ensure the safety of the arm and robot. The first by ensuring that the minimum and maximum values for each joint are respected, while the second is ensured controlling the robot to be parallel to the seafloor. We also included a tasks ensuring the alignment of the robot to the target of the tool, in order to facilitate the task of the arm. This last tasks comes after all the safety and practical tasks, and it is followed by the optimization task.

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

The main difference is that in this case the arm is controlled only when the robot is close to the end effector target position. This ensures that the arm is moved only when the robot is in a safe environment. Moreover, we have introduced the alignment to the target, in this way not only the arm maintains a good dexterity around the target position, but the robot itself is able to effectively move and be controlled around the current position.

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 obtain the desired behavior we have introduced two kinds of disturbances, one linear and the other as angular velocity disturbance, expressed in the inertial frame. The vehicle suffers from these velocities, as they are added before integration.

```
%       Adding the disturbances
disturb = [0 -0.025 0]';
disturb_ang = [0 0 0]';
disturb_ang = disturb_ang*0.5*sin(0.5*t*2*pi);
uvms.p_dot(1:3) = uvms.vTw(1:3,1:3)*disturb;
uvms.p_dot(4:6) = uvms.vTw(1:3,1:3)*disturb_ang;

% beware: p_dot should be projected on <v>
uvms.p = integrate_vehicle(uvms.p, uvms.p_dot, deltat);
```

This task objective is to keep the end effector position stable while the vehicle moves. As a result, the arm and the vehicle result as two different systems: specifically, the first could cope with disturbances and the slow dynamic of the second, but only in a limited way. In order to cope with this disturbances we defined a new task called: armVehiCoord. This task is placed with the highest hierarchy, limiting the manifold of solution to the one where the vehicle velocities are imposed. In fact, in order to decouple the two systems, we need to feed the arm with the vehicle velocities, the jacobian matrix accounts only for the vehicle velocities, disregarding the arm velocities. $\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{y}}$ which in this case becomes

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \mathbf{0}_{[6x7]} & \boldsymbol{I}_{[6x6]} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}$$

The task takes as a reference the velocities of the vehicle, as this is the input the arm has to compensate. Concerning the activation function, this is defined as an identity matrix, as the task is an equality one.

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

In the following images, the end effector position is plotted, showing the three cartesian components. The scale of the plots clearly states that the end effector position could be assumed as constant, as it only suffers from oscillation smaller than a millimeter.

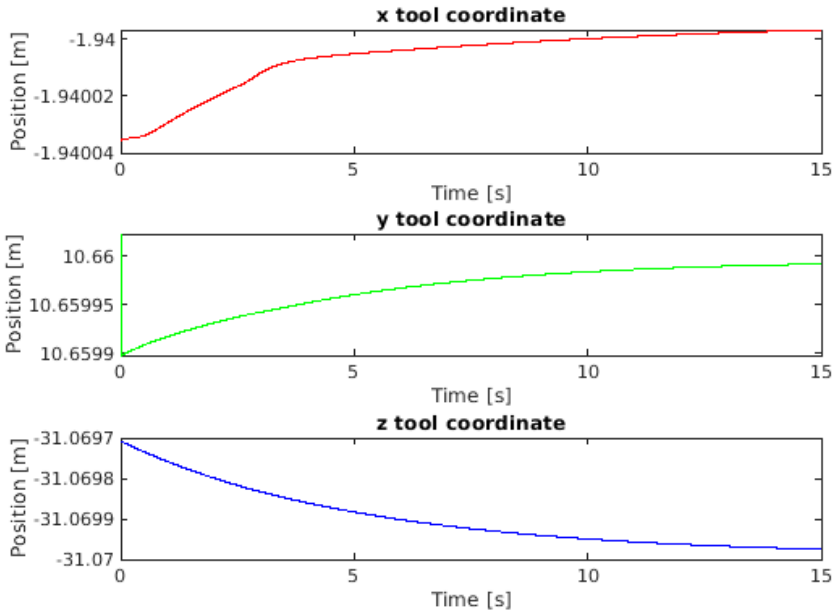


Figure 13: End effector position (x, y, z).

The following image compares the vehicle velocities, expressed in the vehicle frame, and the velocities of the arm joints. The vehicle linear velocities show the effect of the linear disturbance, which brings a change in the vehicle velocities on x and y. The arm velocities are the resultant for the control of the end effector in its original position.

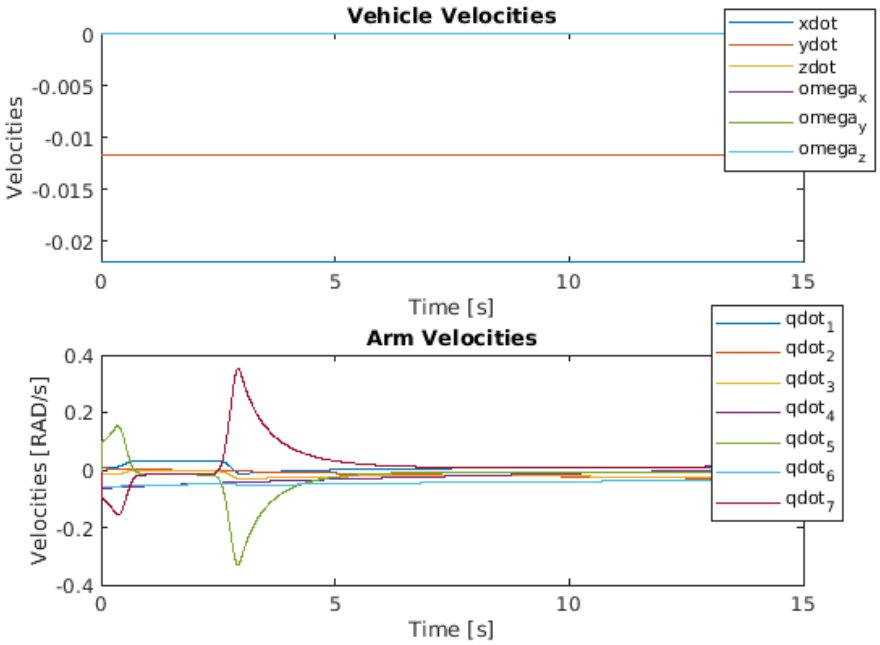


Figure 14: Arm joints velocities.

In this last image, we would like to show the evolution of the end effector velocities expressed in the inertial frame. The velocities start with a nonzero infinitesimal value, but they converge to zero in few seconds.

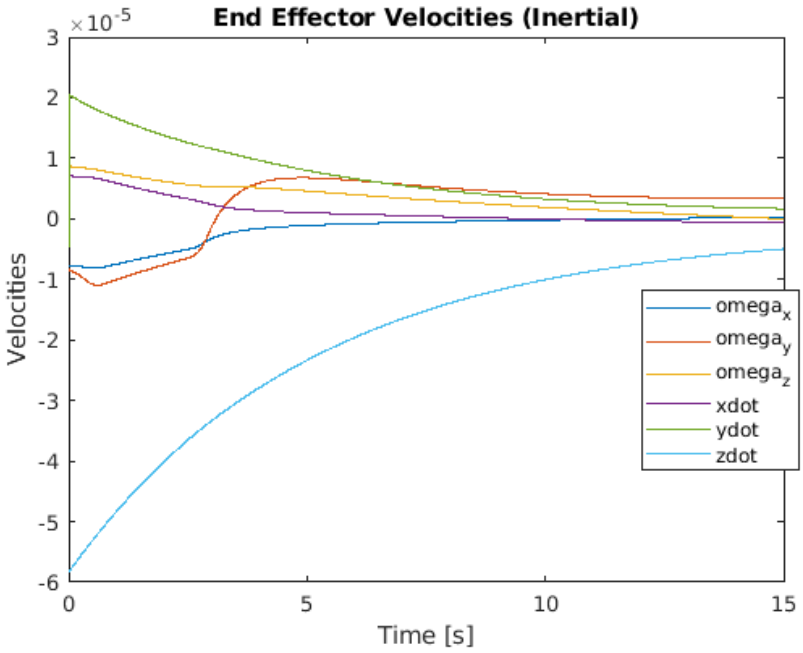


Figure 15: End effector velocities in the inertial frame.

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

If we add a sinusoidal disturbance, it deteriorates the ability of keeping the end effector still. The disturbance is added as described below.

```
%        Adding the disturbances
disturb = [0  0.0  0]';
disturb_ang = [0  1  0]';
disturb_ang = disturb_ang*0.1*sin(0.5*t*2*pi);
uvms.p_dot(1:3) = uvms.vTw(1:3,1:3)*disturb;
uvms.p_dot(4:6) = uvms.vTw(1:3,1:3)*disturb_ang;
```

However, as it can be seen in the following images, the end effector does not moves much. The following figure shows that the oscillations of the tool position are of the order of one millimeter, which can be considered as a acceptable result.

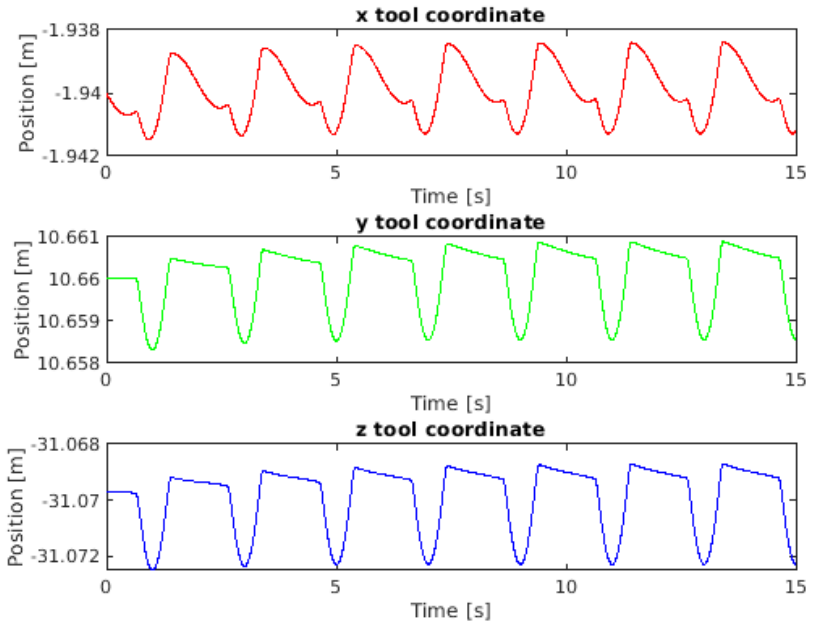


Figure 16: End effector position (x, y, z) with weak disturbance.

The following images compares the vehicle velocities with the joint rotational speed.

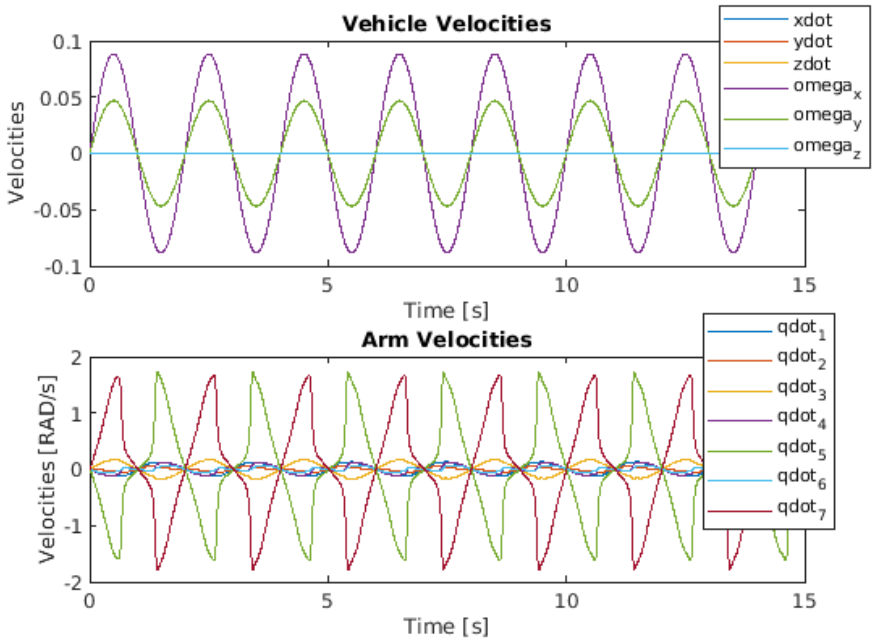


Figure 17: Arm joints velocities with weak disturbance.

Finally, this last image shows that the end effector velocities remains reasonable small.

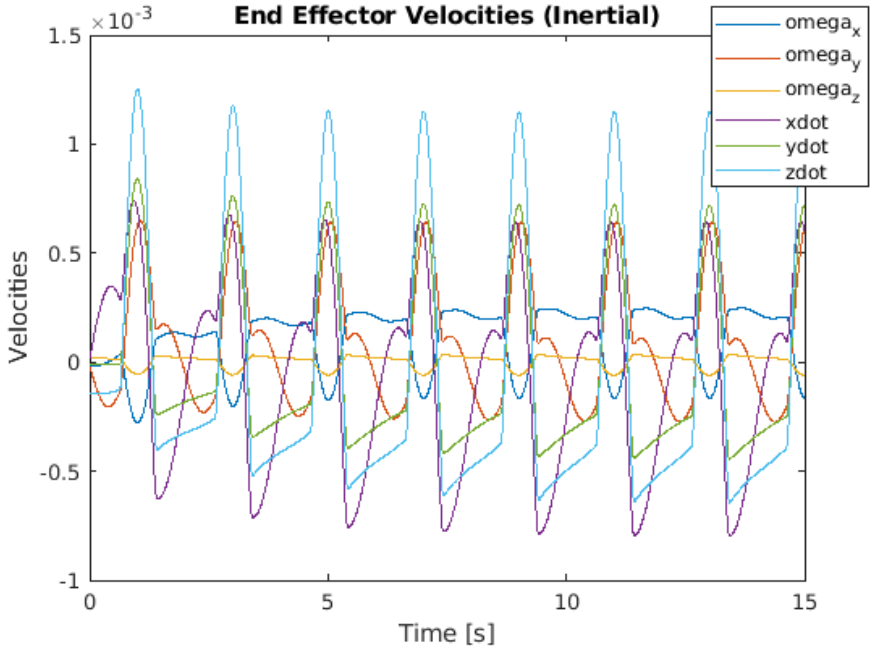


Figure 18: End effector velocities in the inertial frame with weak disturbance.

However, if we increase the disturbance too much, the arms cannot cope with the disturbance in the vehicle. As a result, the displacement of the end effector is not negligible. Even if we increase the gain and the saturation of the tool control task, this is not sufficient to cancel the disturbance. The disturbance is added as described below.

```
%       Adding the disturbances
    disturb = [0  0.0  0]';
    disturb_ang = [0  1  0]';
    disturb_ang = disturb_ang*0.7*sin(0.5*t*2*pi);
    uvms.p_dot(1:3) = uvms.vTw(1:3,1:3)*disturb;
    uvms.p_dot(4:6) = uvms.vTw(1:3,1:3)*disturb_ang;
```

The results of such a disturbance as shown in the images below. From this first image we can see that the end effector is clearly moving from the desired position. The displacements on x reach almost 10 cm.

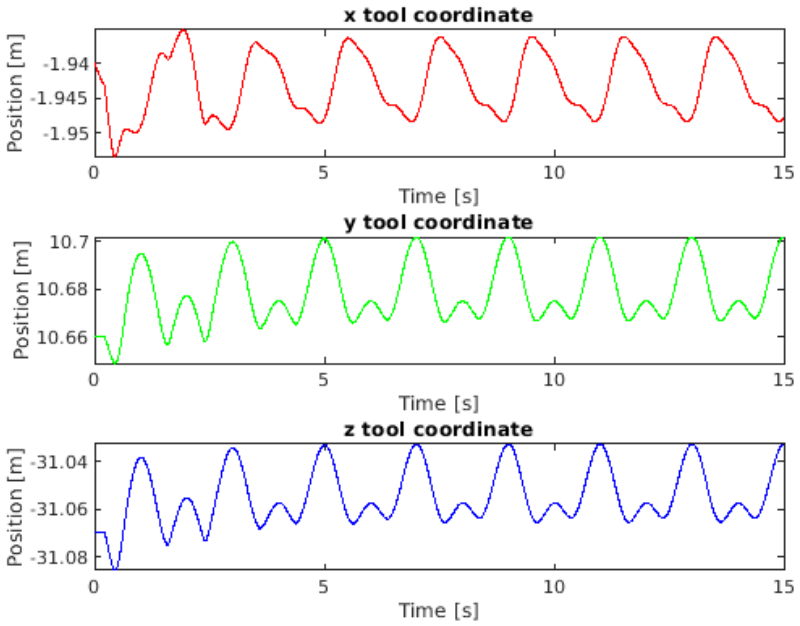


Figure 19: End effector position (x, y, z) with strong disturbance.

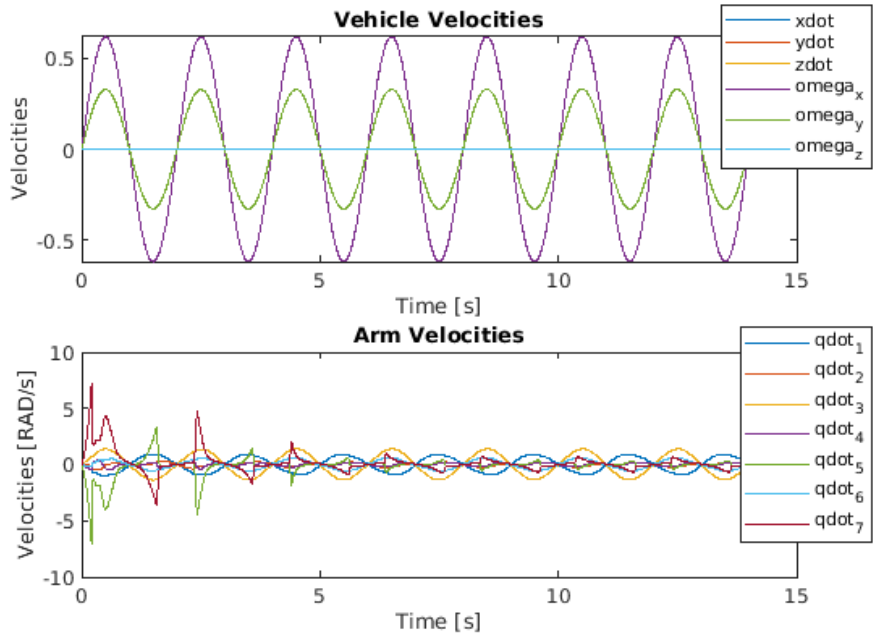


Figure 20: Arm joints velocities with strong disturbance.

Comparing the vehicle velocities with either the arm joints or end effector velocities, it is clear that in this case the task fails in its purpose.

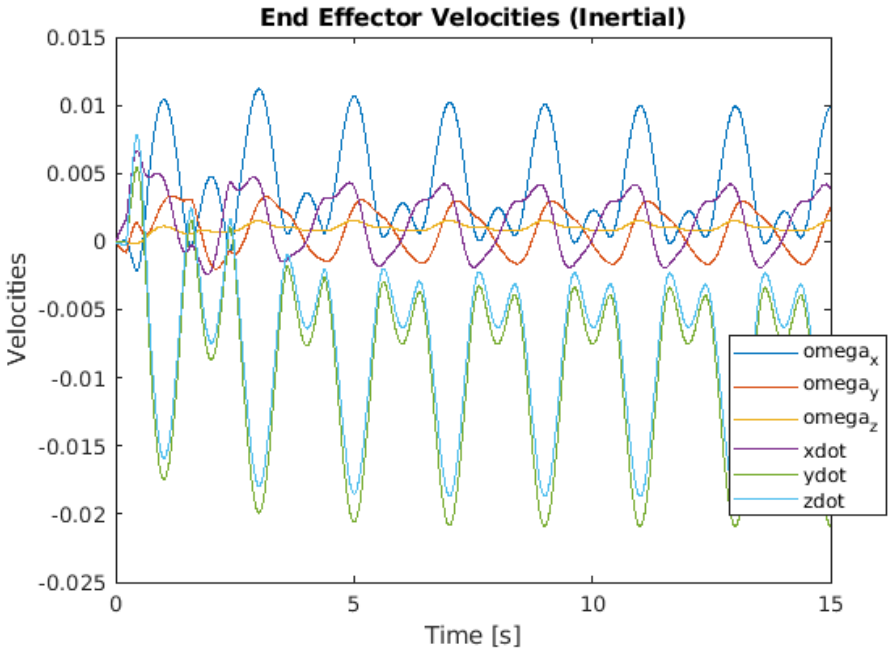


Figure 21: End effector velocities in the inertial frame with strong disturbance.

These results were computed using an high gain on the tool control task, trying to counteract to the effect of the changes in the vehicle position. However, these efforts were not sufficient to correct the end effector position. The reason is that this task can cope with disturbances only if they are quantitatively small. The aim of the task is stabilize the end effector in the scenario where small disturbances act on the vehicle; such as current or the slow vehicle dynamics.