Cooperative Robotics

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

General notes

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

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

Table 1: Example of actions/hierarchy table: a number in a given cell represents the priority of the control task (row) in the hierarchy of the control action (column). The type column indicates whether the objective is an equality (E) or inequality (I) one.

Task	Type	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
Task A	I	1		1
Task B	I	2	1	
Task C	E		2	2

1 Exercise 1: Implement a "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 controls both 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:

$$uvms.Jv\_pos = \begin{bmatrix}zeros(3,7) & uvms.wTv(1:3,1:3) & zeros(3,3)\end{bmatrix}^\top$$
$$uvms.Jv\_att = \begin{bmatrix}zeros(3,7) & zeros(3,3) & uvms.wTv(1:3,1:3)\end{bmatrix}^\top$$

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. In our case the desired value is the goal position (or attitude) obtained with the *CartError()* function and the current value is the current position (or attitude).
The two task reference vectors are computed in this way:

$$\begin{aligned}[ang, lin] &= CartError(uvms.wTg\_v, uvms.wTv);\\ uvms.xdot.v\_pos &= Saturate(0.5 * lin, 0.5);\\ uvms.xdot.v\_att &= Saturate(0.5 * ang, 0.5);\end{aligned}$$

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.

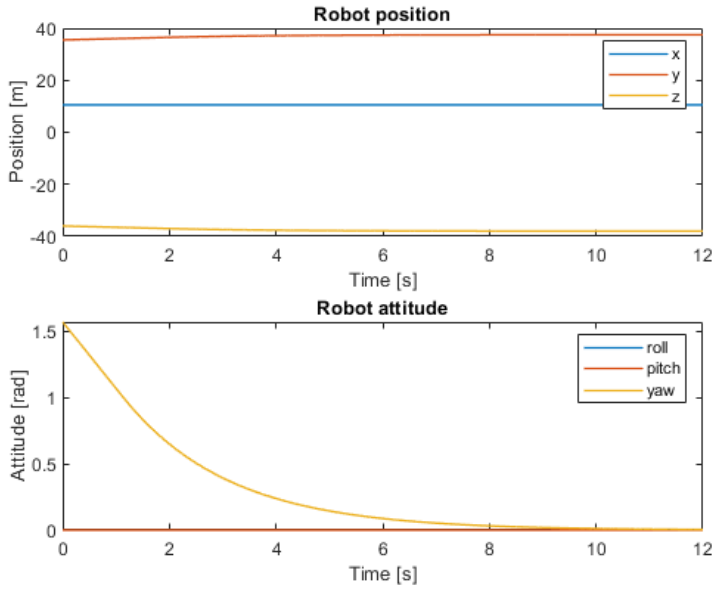


Figure 1: The robot moves in the Y and Z directions and the yaw angle converges to zero.

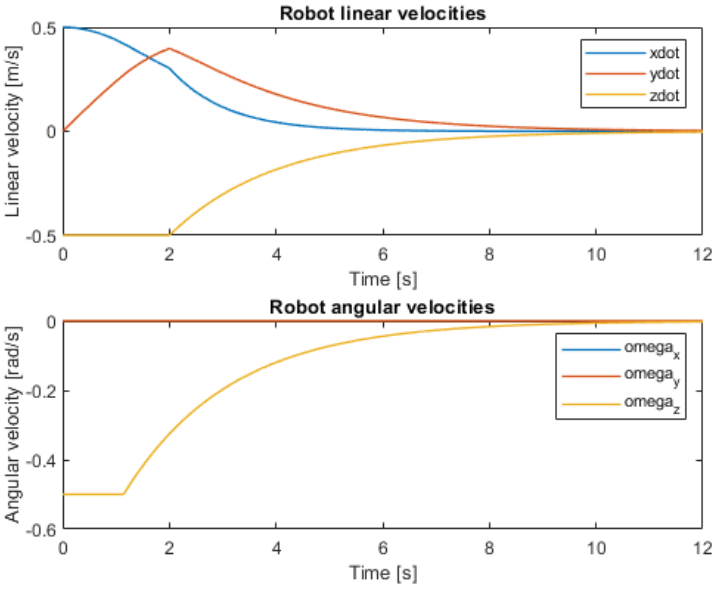


Figure 2: After an initial transient phase, all the velocities converge to zero.

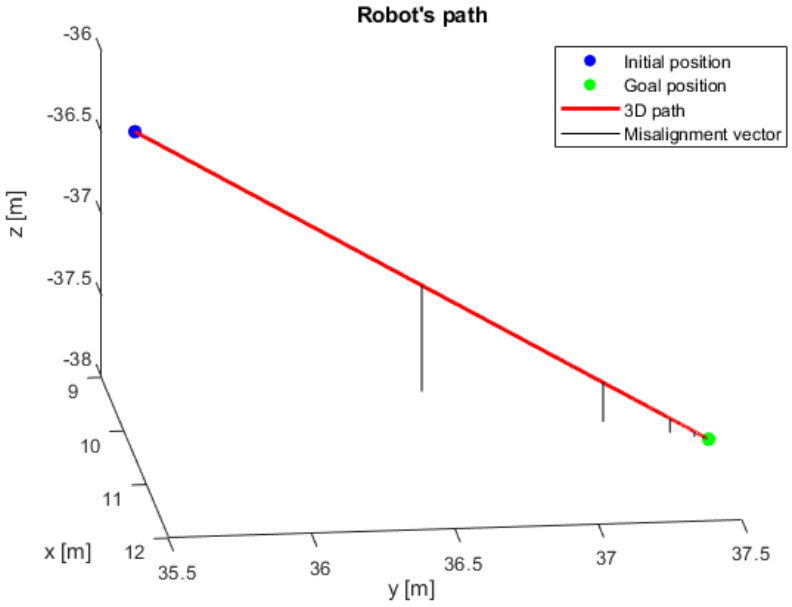


Figure 3: Here we can see that the robot reaches the goal position and that the misalignment vector shrinks towards zero.

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. In this case we have two conflicting tasks, but since the Horizontal Attitude one has higher priority the robot 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).

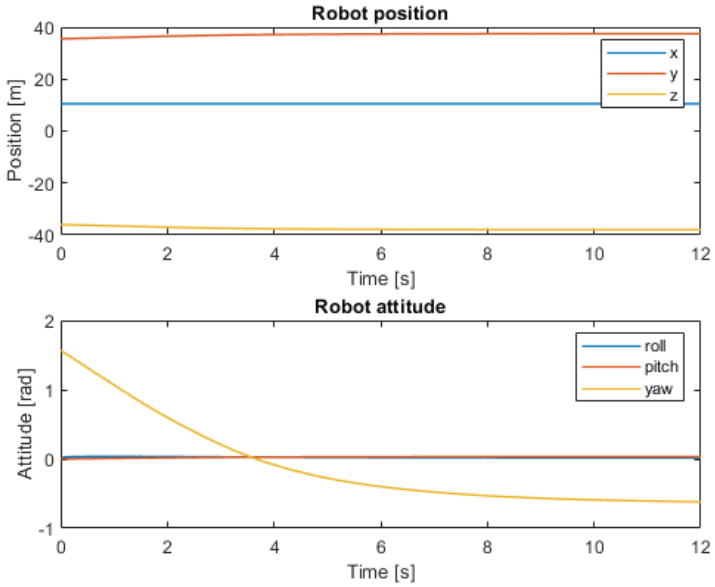


Figure 4: The robot moves in the Y and Z directions, but the attitude doesn't converge to the desired one.

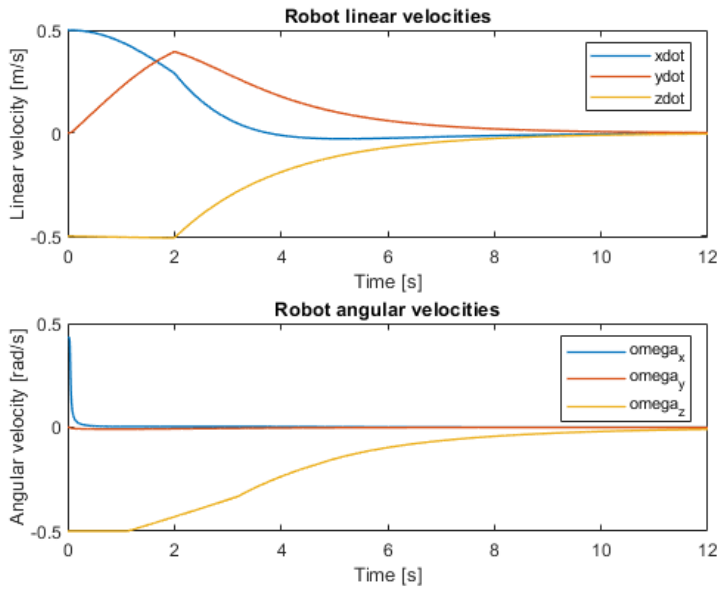


Figure 5: The robot velocities converge to zero after a while.

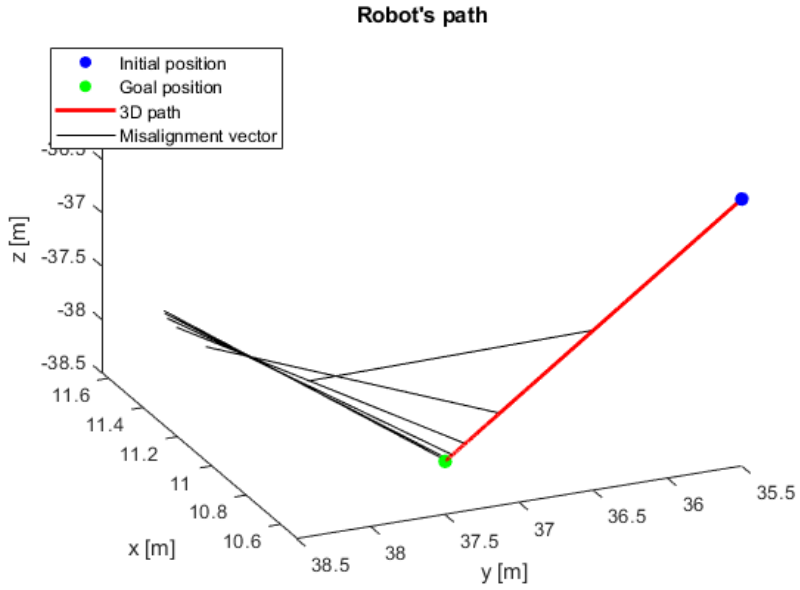


Figure 6: The robot can reach the goal position, but the misalignment vector keeps growing.

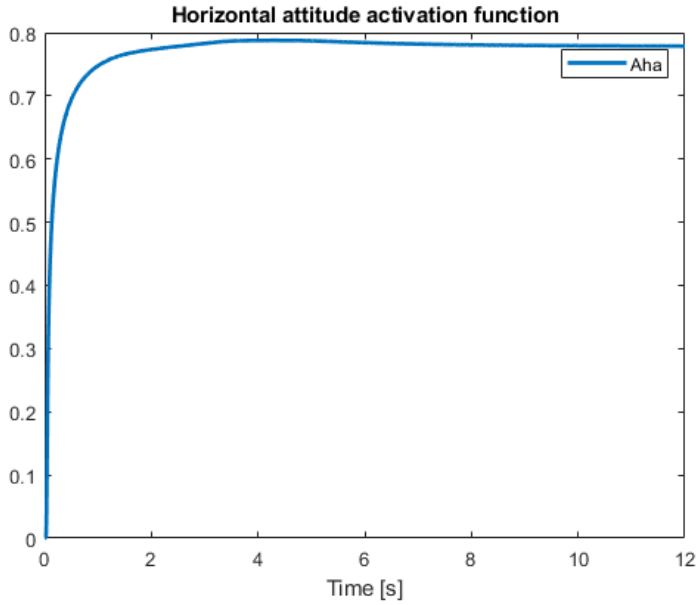


Figure 7: The activation function of the Horizontal Attitude task almost immediately reaches an high value, but it can't go back to zero since the robot has conflicting goals.

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 Position and 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.

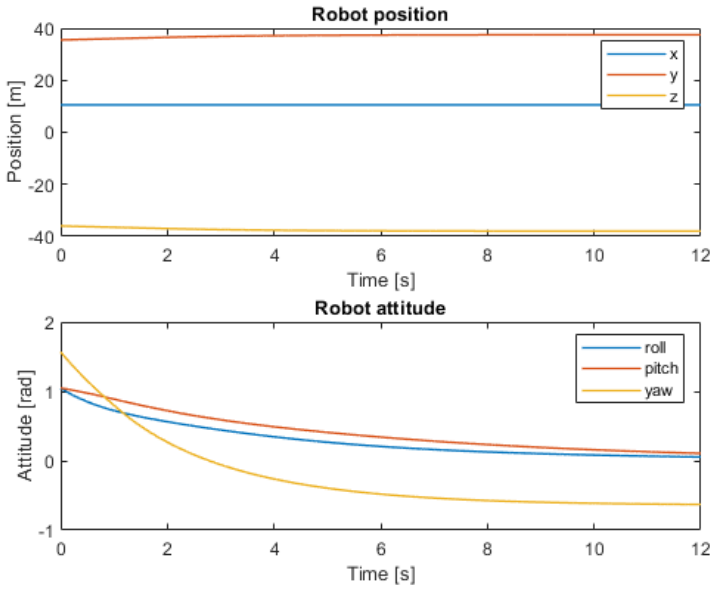


Figure 8: We can see that the pitch and roll angles start from $\pi/3$ and converge near to 0: the robot moves in Y and Z, reaching the goal position.

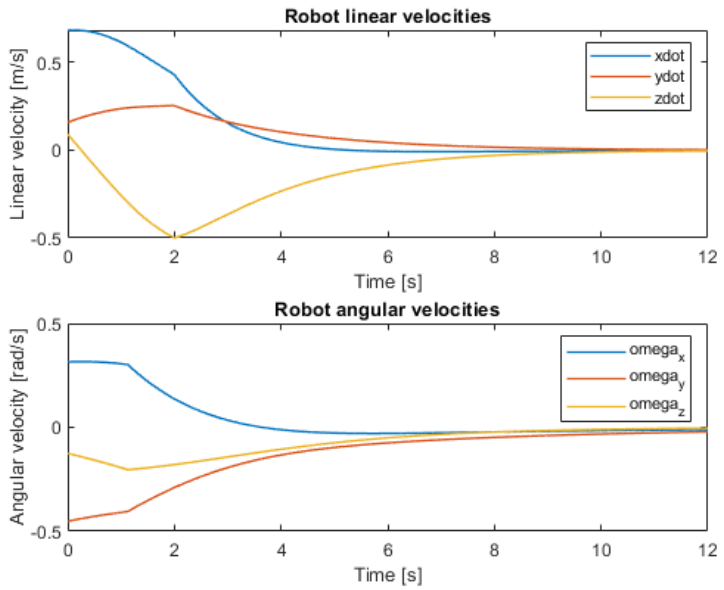


Figure 9: The robot rotates quite fast in the first 2-3 seconds, but then the angular velocities converge to zero. We can see a different $\dot{z}$ with respect to other cases since the robot is initially tilted.

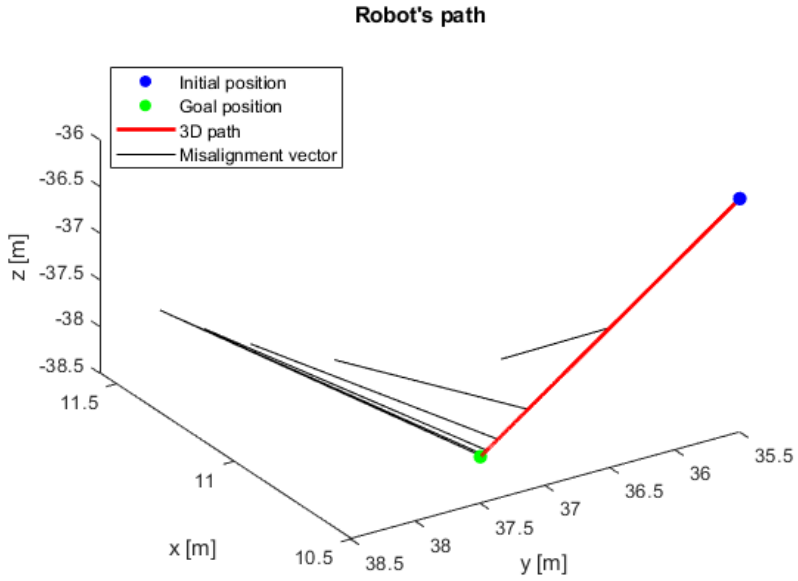


Figure 10: The robot reaches the goal position, but the misalignment vector continues to grow.

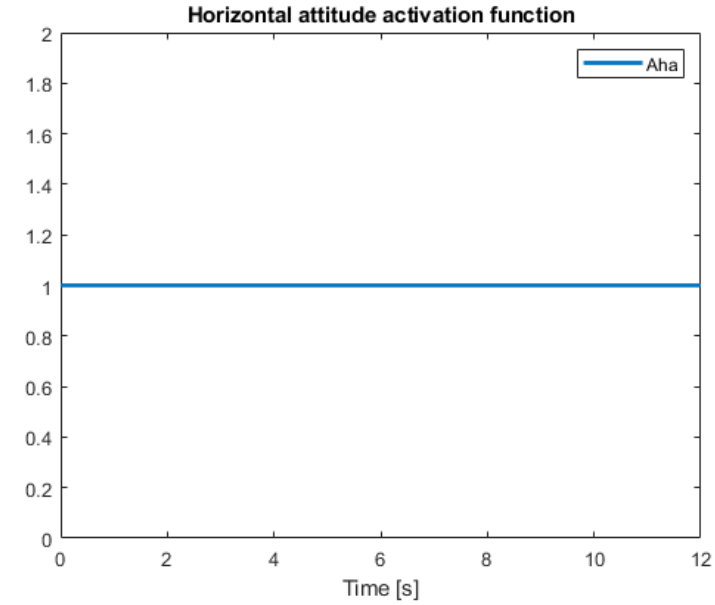


Figure 11: The activation function is always 1 since the norm of the misalignment vector is always bigger than the maximum allowed value.

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

Here I 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.

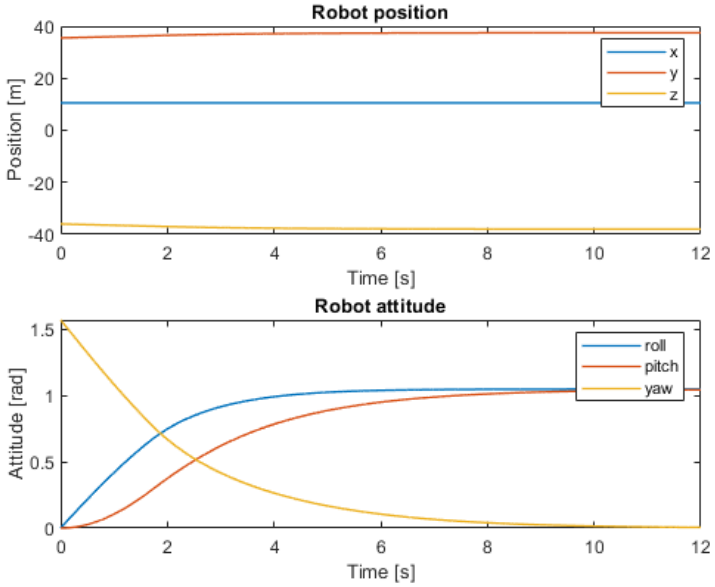


Figure 12: The robot configuration converges to the target one.

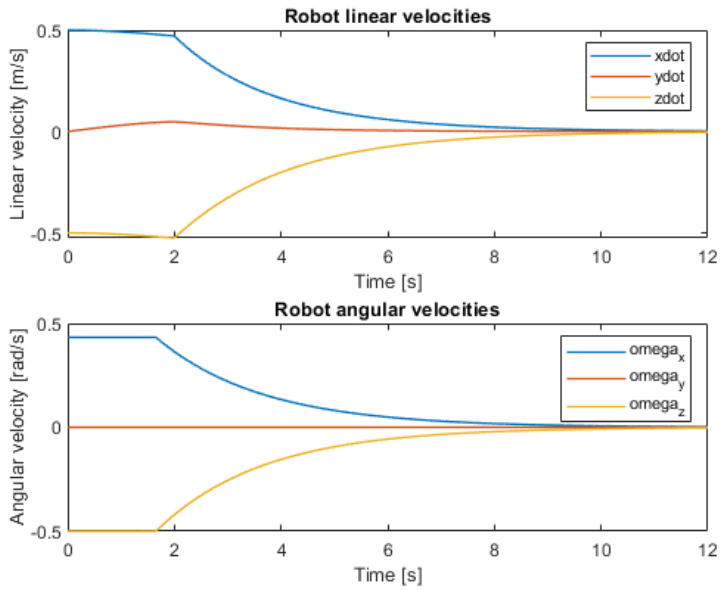


Figure 13: The robot velocities converge to zero after a brief transient period.

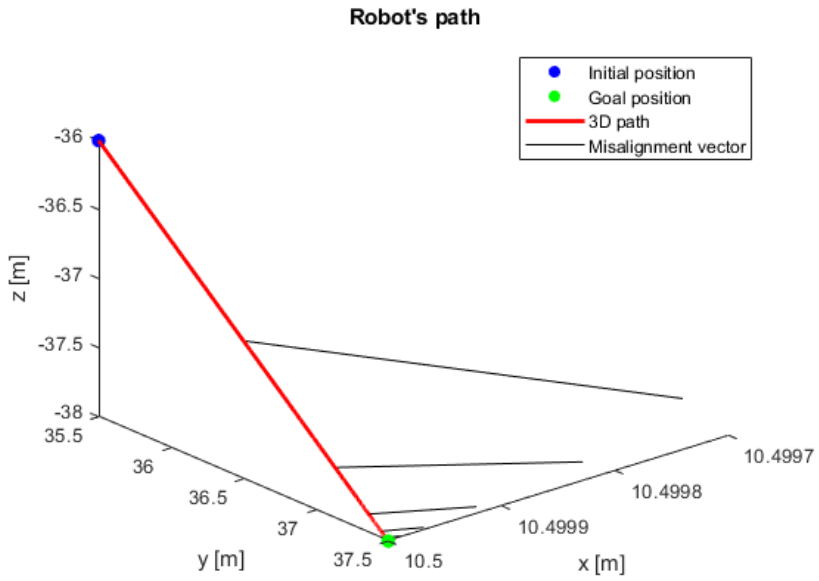


Figure 14: The robot reaches the target position and the misalignment vector shrinks towards zero.

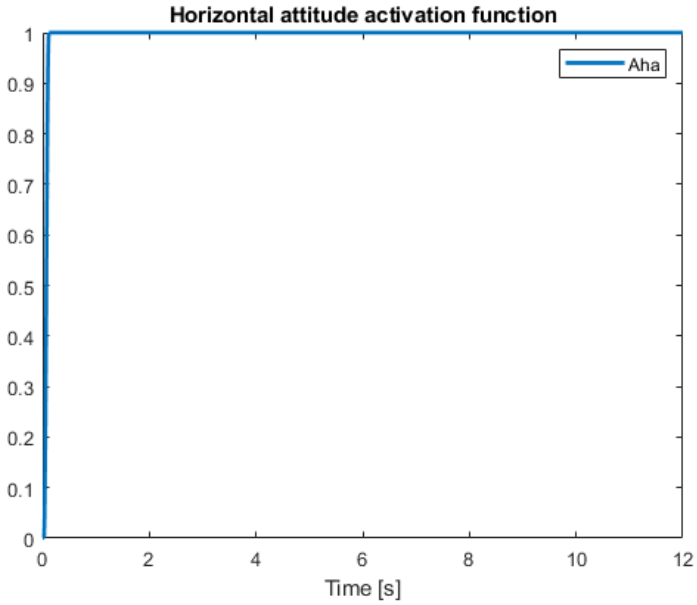


Figure 15: The value of the activation value is almost always 1 since the task doesn't have effect on the robot.

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.

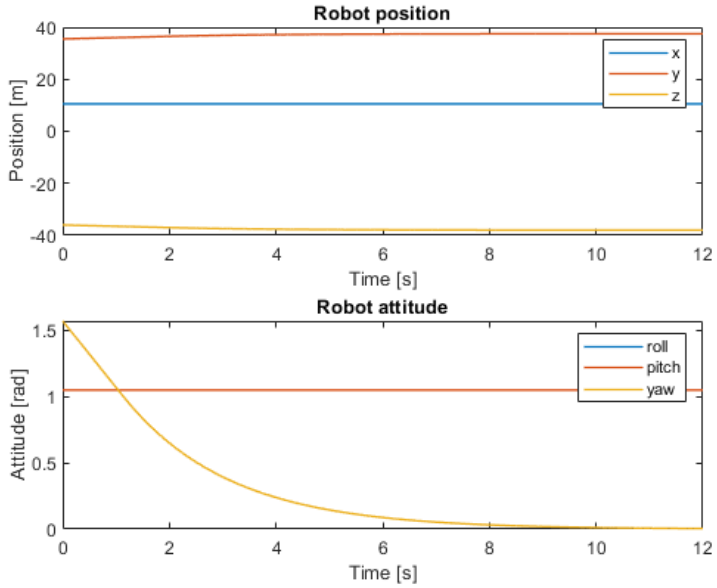


Figure 16:

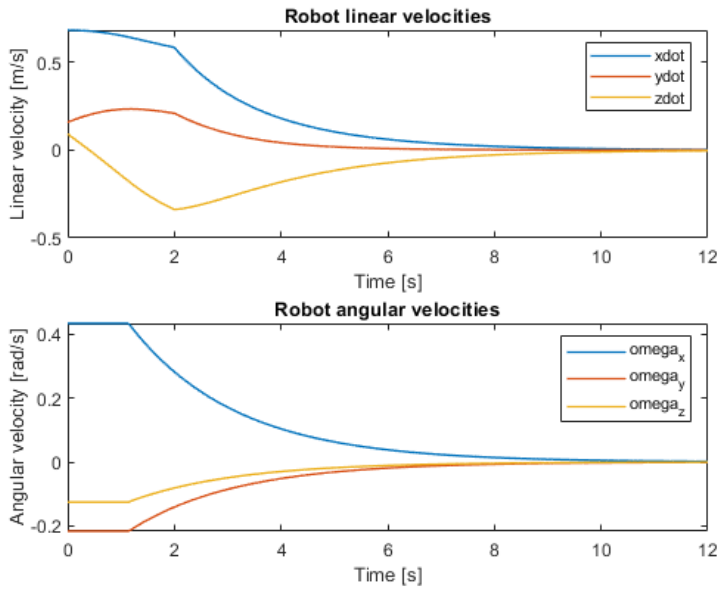


Figure 17:

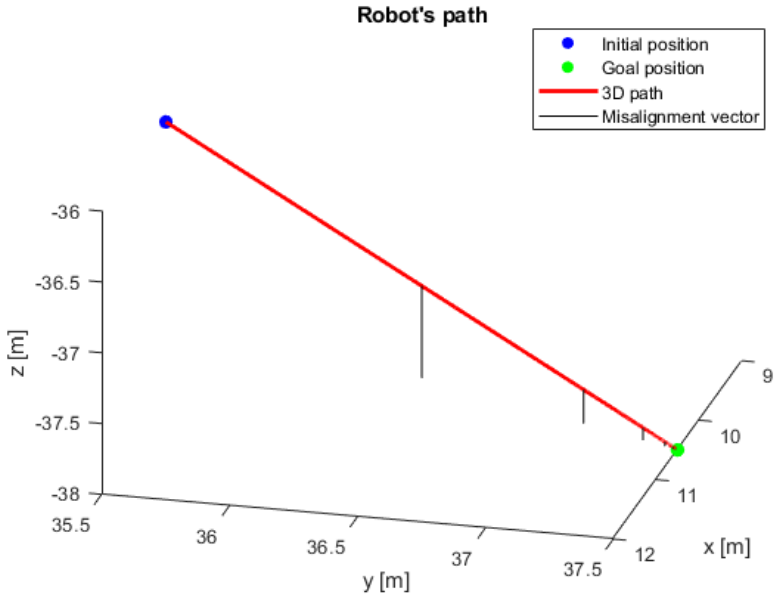


Figure 18:

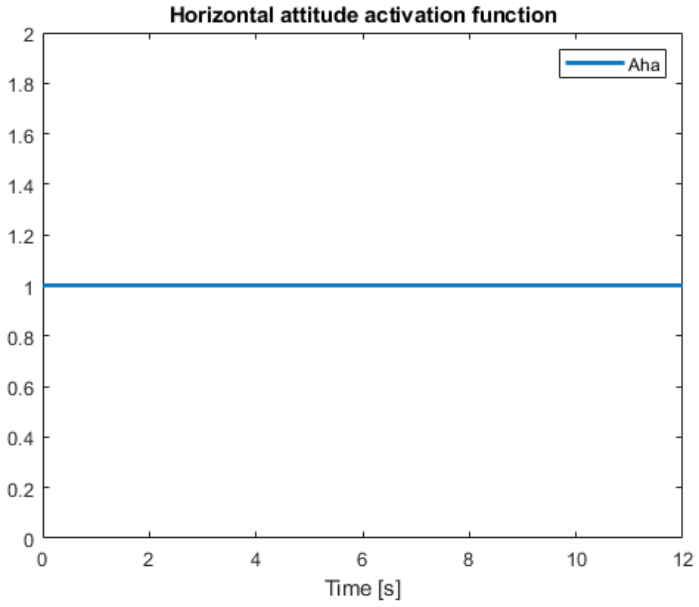


Figure 19:

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.

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.

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.

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?

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?

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.

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

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

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.

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

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 2: 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 Minimum Altitude 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 20.

Figure 20: 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.

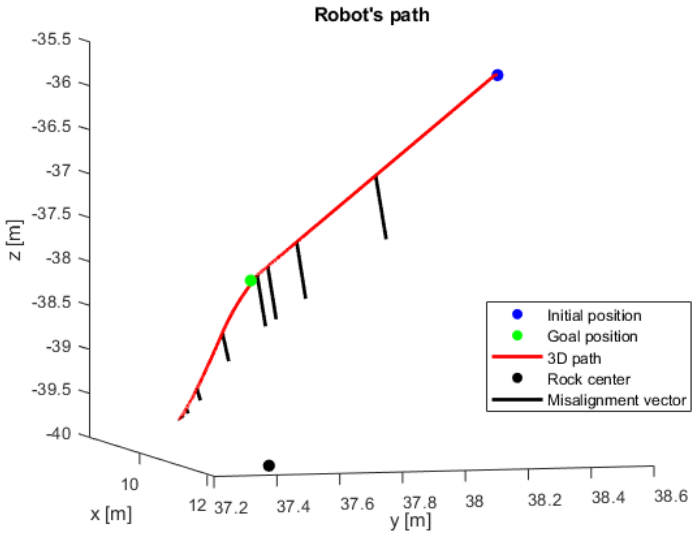


Figure 21: Plot of the Vehicle path.

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.

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

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

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.

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

The Jacobian relationship obtained is the following:

$$\begin{bmatrix} \boldsymbol{\omega_x} \\ \boldsymbol{\omega_y} \\ \boldsymbol{\omega_z} \end{bmatrix} = J_{ha}\dot{\boldsymbol{y}}$$

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

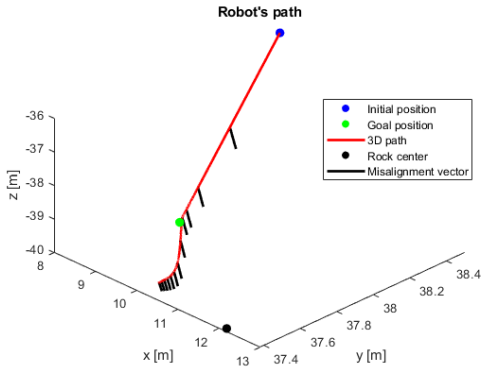
The task reference is computed using the norm of the misalignment vector, θ. The desired value in this case is 0, because we want the two vectors to be aligned.

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

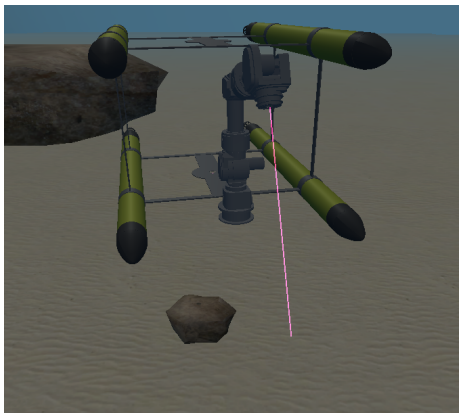
3.1.3 Q3: Try changing the gain of the alignment task. Try at least three different values, where one is very small. What is the observed behaviour? Could you devise a solution that is gain-independent guaranteeing that the landing is accomplished aligned to the target?

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

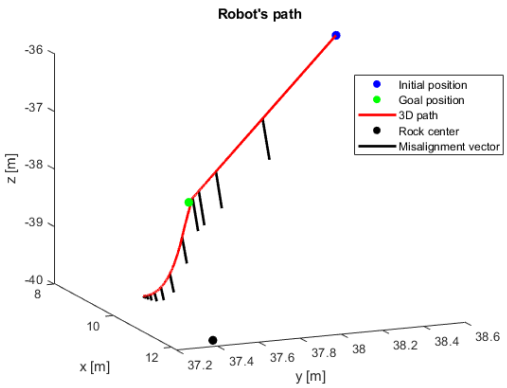
(a) Heading of the vehicle when it approches the ground with gain = 0.05



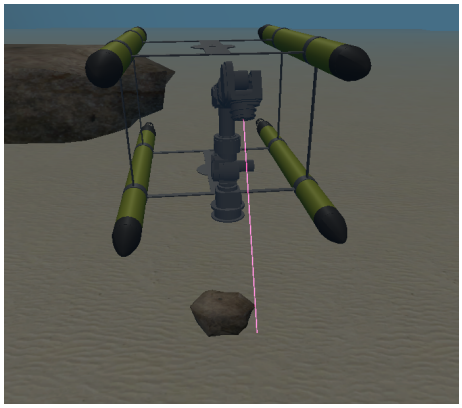
(b) Path of the vehicle and misalignment vector with gain = 0.05



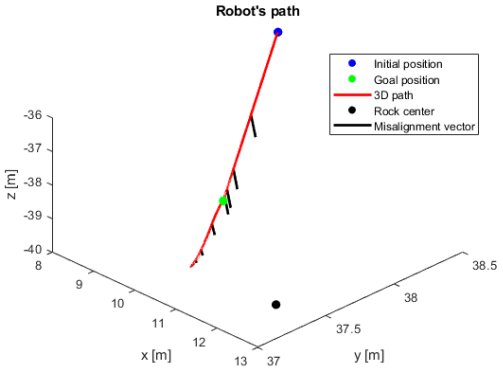
(c) Heading of the vehicle when it approches the ground with gain = 0.2



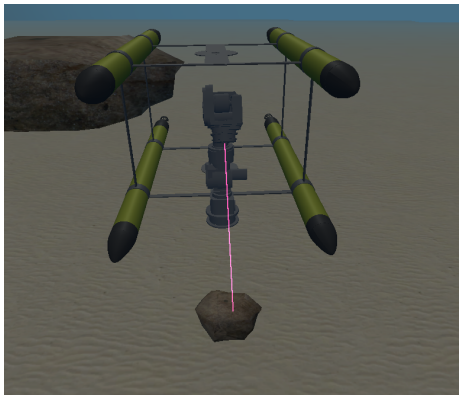
(d) Path of the vehicle and misalignment vector with gain = 0.2



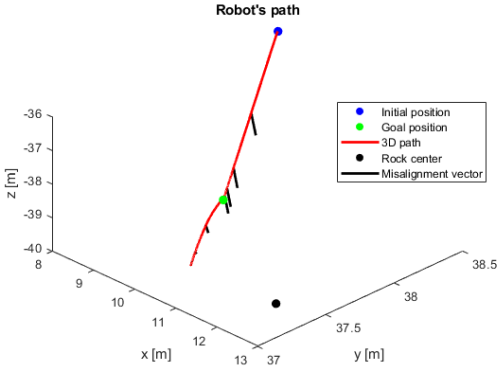
(e) Heading of the vehicle when it approches the ground with gain = 0.4



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

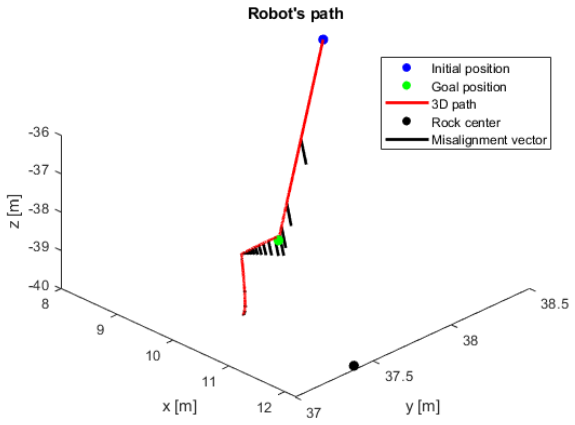


(g) Heading of the vehicle when it approches the ground with gain = 0.7



(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 you can clearly see in images (b) and (d), so 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. 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. This action is performed until the module of the misalignment vector, θ, becomes lower than 0.02 rad.
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?

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

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

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?

4.2 Adding a joint limit task

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

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

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

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

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?

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

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

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?

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

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.wTv(1:3,1:3)*disturb;
uvms.p_dot(4:6) = uvms.wTv(1:3,1:3)*disturb_ang;

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

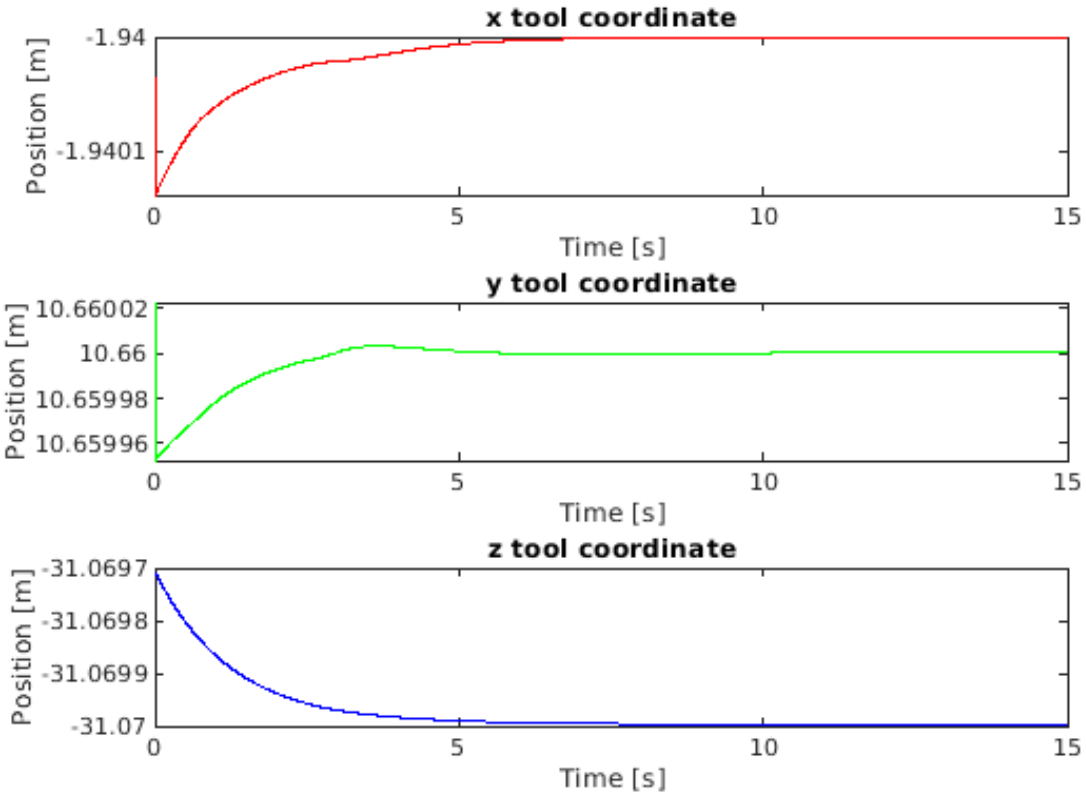
This task objective is to keep the end effector position stable while the vehicle moves. As a result, the arm and the vehicle result as two different systems; specifically, the first could cope with incugruences in the second, but only in a limited way. In order to cope with this disturbances we defined a new task called: armVehiCoord. This task is placed with the highest hierarchy, limiting the manifold of solution to the ones accounting for the vehicle velocities. 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 becames

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

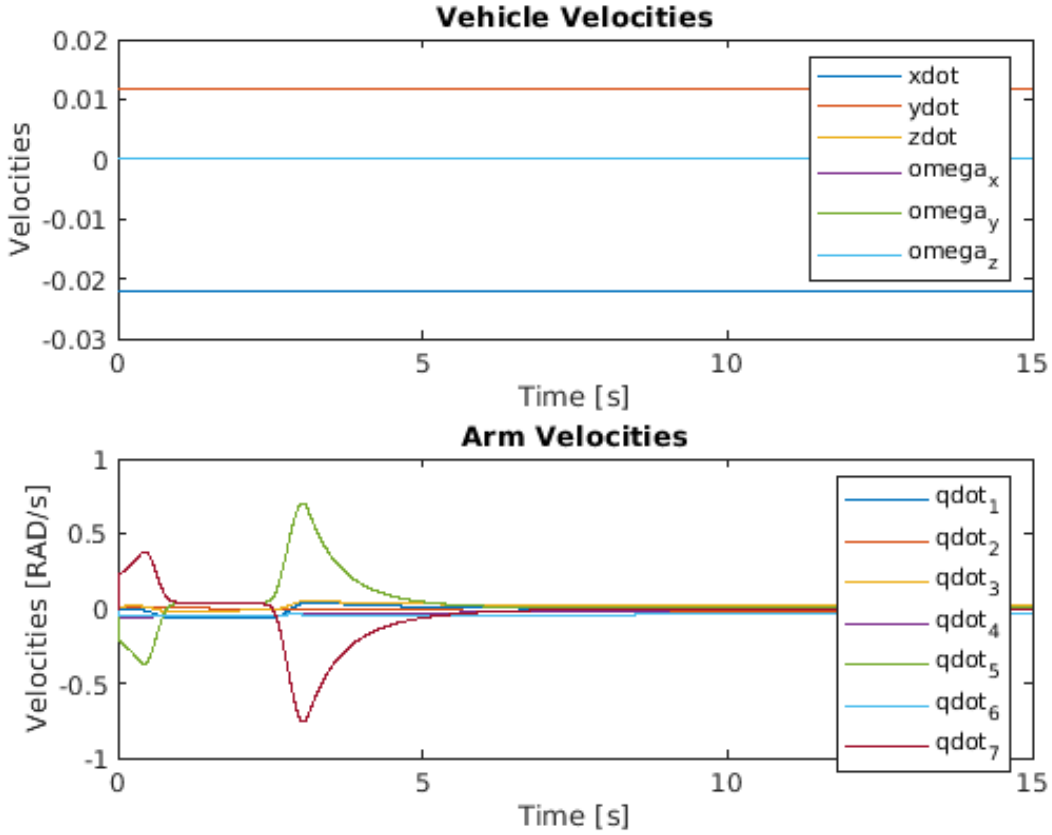
The task takes as a reference the velocities of the vehicle, as this is the input the arm has to compensate. Concerning the activation funtion, this is defined as an identity matrix, as the task is an equality one.

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.

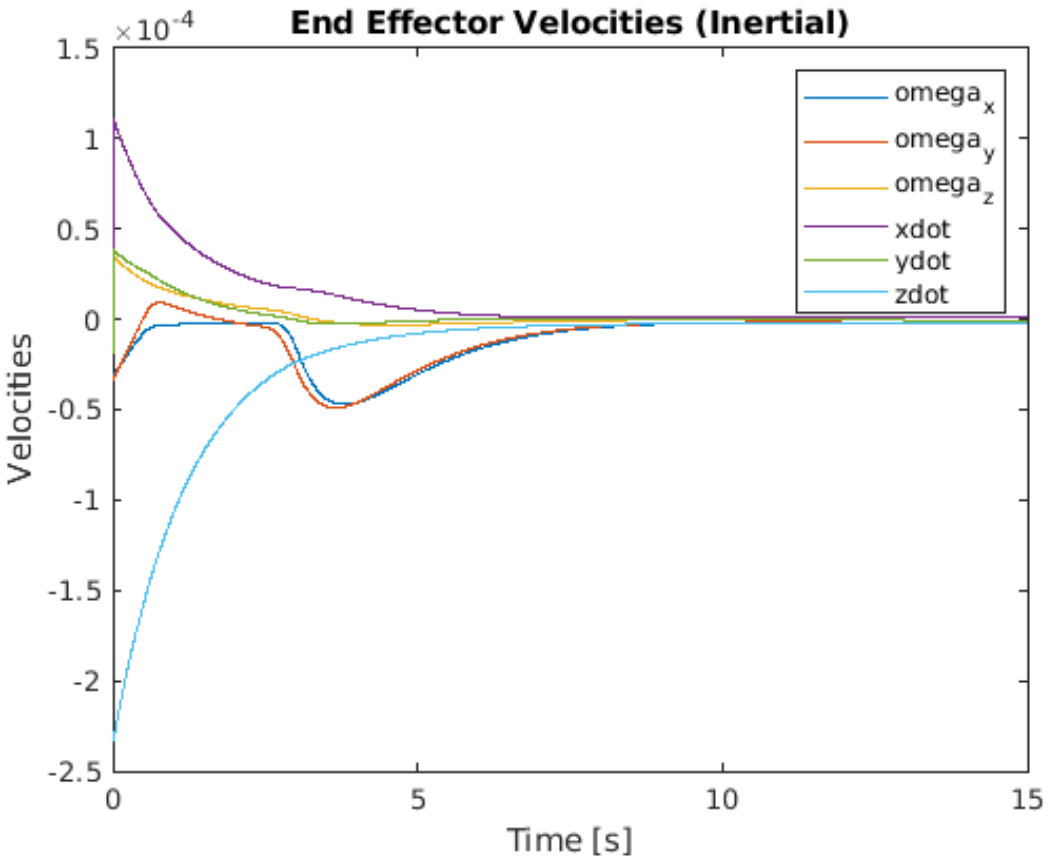
In the following images, the end effector position is plotted, showing the three cartesian components. The scale of the plots clearly states that the end effector position could be assumed as constant, as it only suffers from oscillation smaller than a millimiter.



The following image compares the vehicle velocities, expressed in the vehicle frame, and the velocites 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.



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.

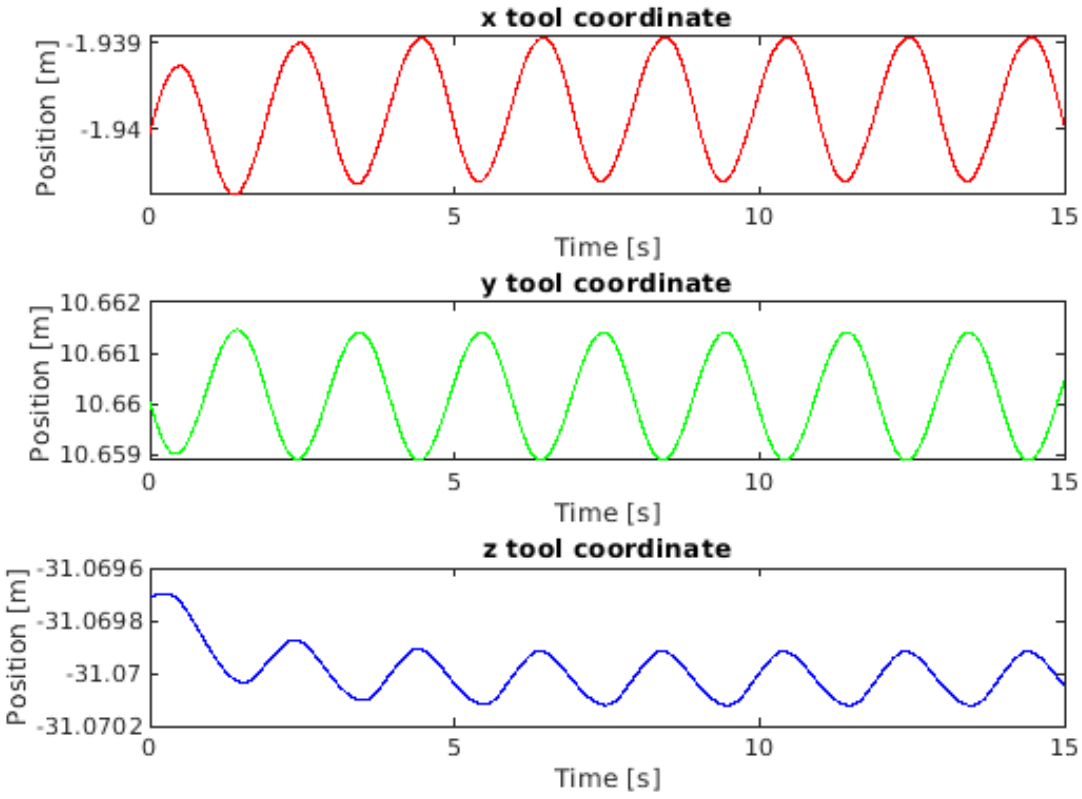


6.1.3 Q3: What happens if the sinusoidal disturbance becomes too big? Try increasing the saturation value of the end-effector task if it is too low.

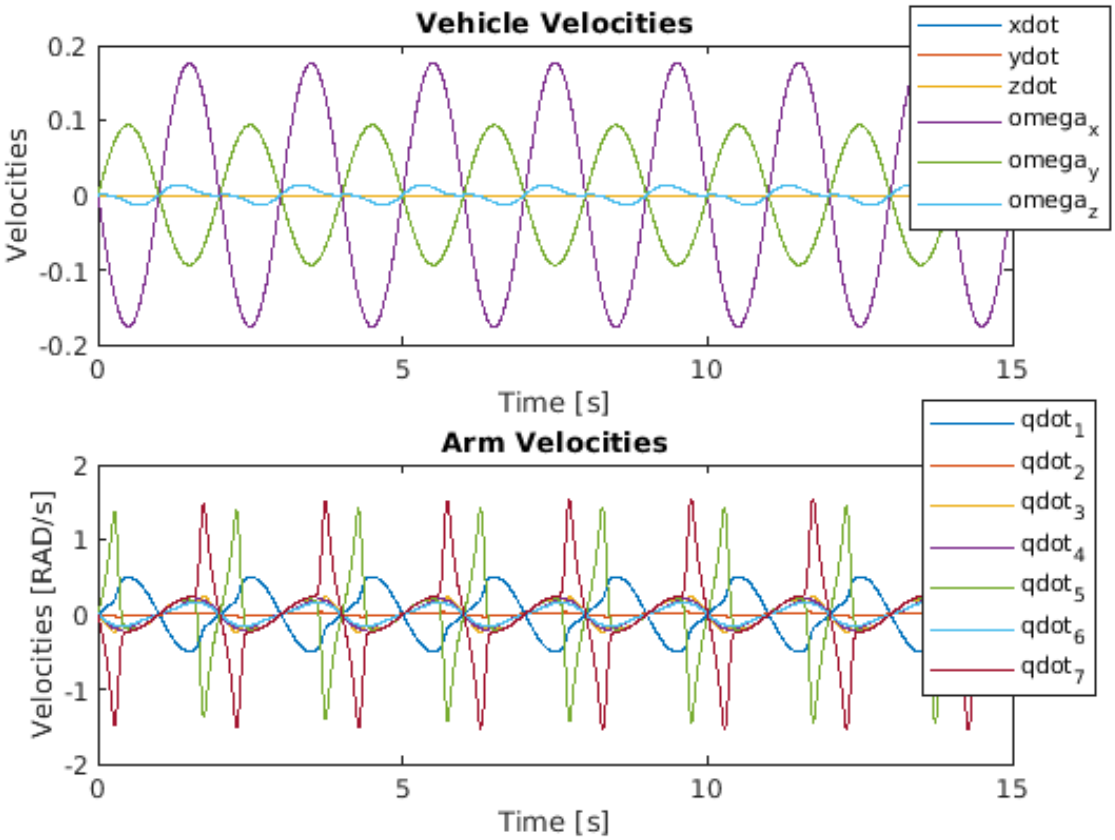
If we add a sinusoidal disturbance, it deteriorates the ability of keeping the end effector still. The disturbance is added as described below.

```
%       Adding the disturbances
disturb = [0  0.0  0]';
disturb_ang = [0  1  0]';
disturb_ang = disturb_ang*0.2*sin(0.5*t*2*pi);
uvms.p_dot(1:3) = uvms.wTv(1:3,1:3)*disturb;
uvms.p_dot(4:6) = uvms.wTv(1:3,1:3)*disturb_ang;
```

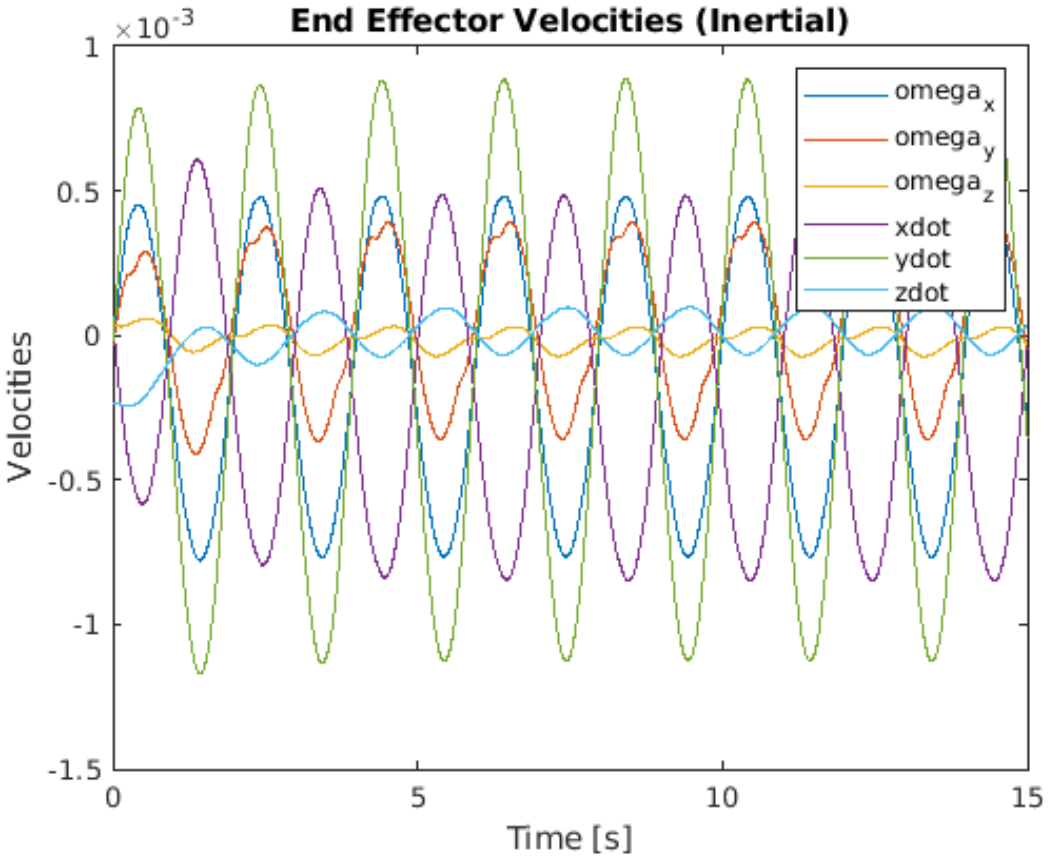
However, as it can be seen in the following images, the end effector does not moves much. The following figure shows that the oscillations of the tool position are of the order of one millimiter, which can be considered as a acceptable result.



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



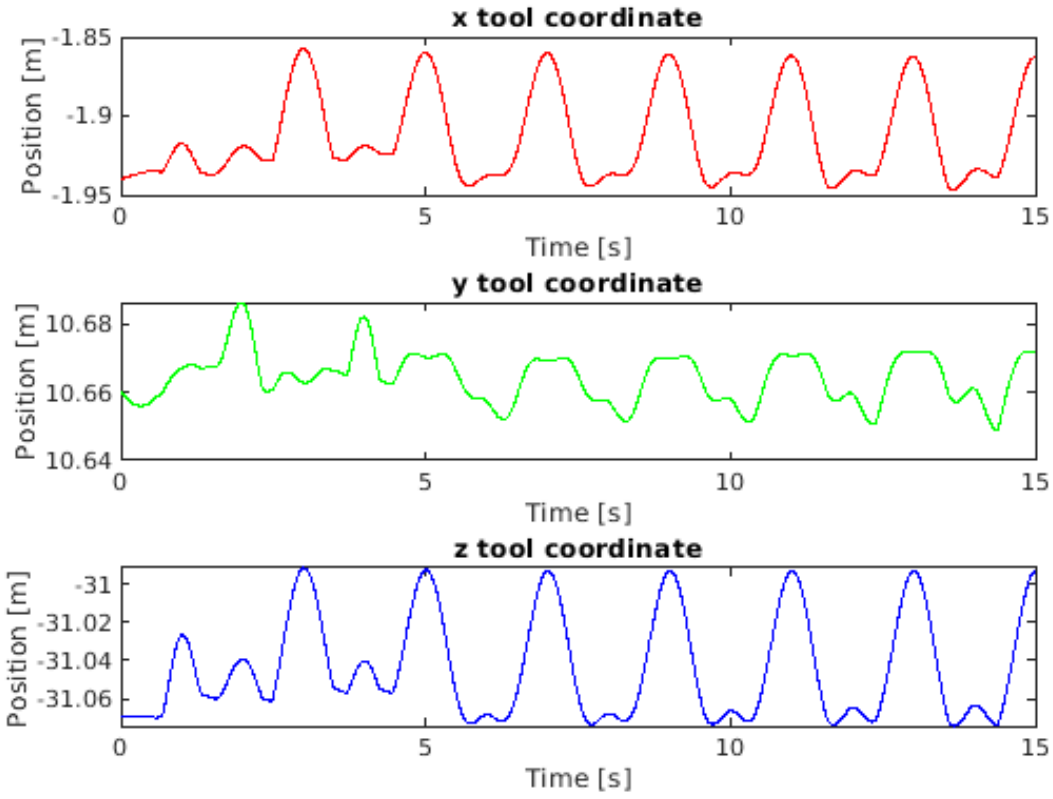
Finally, this last image shows that the end effector velocities remains reasonable small.

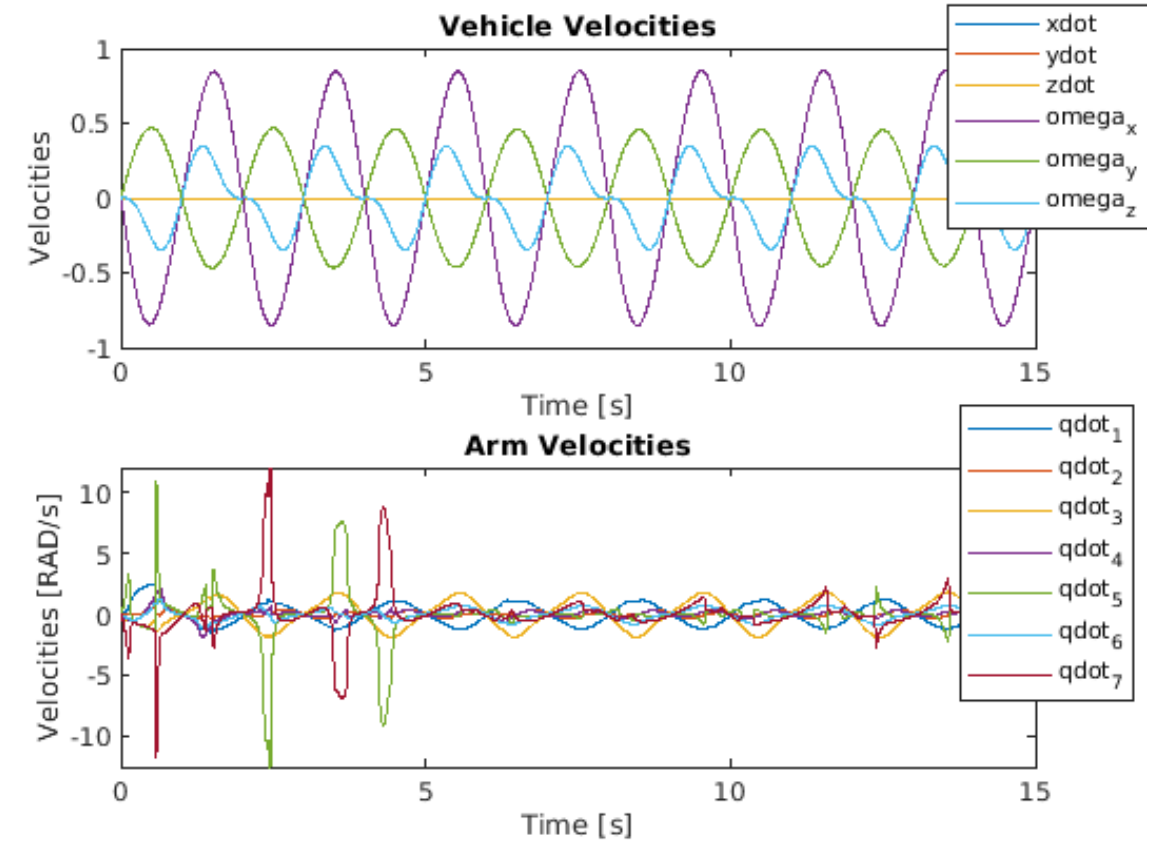


However, if we increase the disturbance too much, the arms cannot cope with the disturbance in the vehicle. As a result, the displacement of the end effer are not negletable. Even if we increase the gain and the saturation of the tool control task, this is not sufficient to cancel the disturbannce. The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

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

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





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.