Cooperative Robotics

Authors: Andrea Tiranti, Roberto Albanese
EMAILs: andrea.tiranti97@gmail.com, ralbanese18@gmail.com
Date: 10/09/21

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 control objective

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

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

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

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

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

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

The Jacobian relationship for the vehicle position control task (**Task B1**) is:

$$J_{vpos} = \begin{bmatrix}\mathbb{0}^{3x6} & | & {}^{w}_{v}R & \mathbb{0}^{3x3}\end{bmatrix} \tag{1}$$

where:

- $\mathbb{0}^{nxm}$ is a matrix of zero values of dimension nxm;
- ${}^{w}_{v}R$ is the orientation matrix between the world frame and the vehicle frame.

While the Jacobian relationship for the vehicle attitude control task (**Task B2**) is:

$$J_{vatt} = \begin{bmatrix}\mathbb{0}^{3x6} & | & \mathbb{0}^{3x3} & {}^{w}_{v}R\end{bmatrix} \tag{2}$$

The final Jacobian for vehicle control is:

$$J_v = \begin{bmatrix}J_{vpos} & | & J_{vatt}\end{bmatrix} \tag{3}$$

And the task reference is compute as follows:

$$\dot{x}_v = -K_v * ({}^{w}goal - {}^{w}v_{pos}) = -K_v * \varepsilon \tag{4}$$

where:

- K_v is the control gain applied to the relative task reference;
- ${}^{w}goal$ is the desired goal position of the task projected on the world frame;
- ${}^{w}v_{pos}$ is the vehicle position projected on the world frame;
- ε is the Cartesian Error between goal frame $<g>$ and vehicle frame $<v>$ both computed w.r.t. the world frame $<w>$.

Below we can see in the graphs in Figure 1 the time evolution of the vehicle position and orientation and we can check that the vehicle actually reaches the required pose.

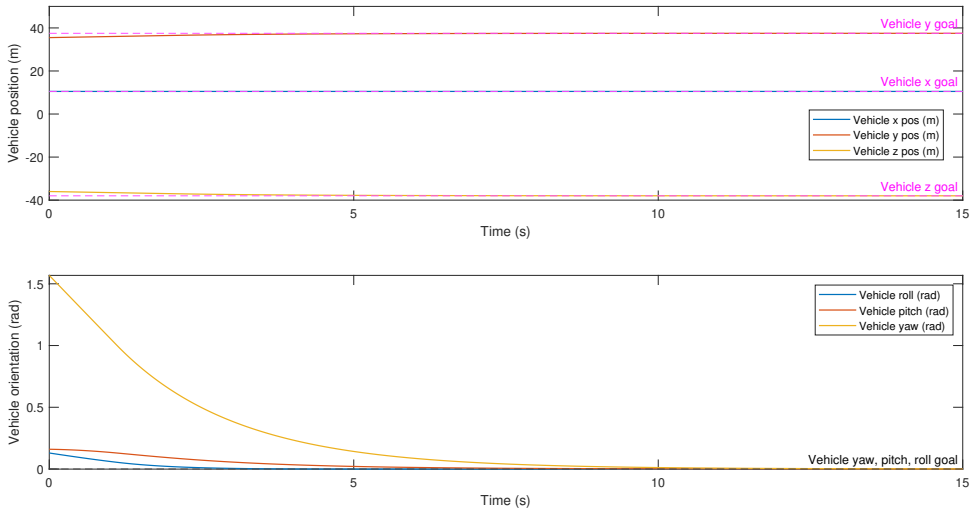


Figure 1: Vehicle Position and Orientation time evolution

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

The Horizontal Attitude Control task (**Task A**) is an inequality task used to control the misalignment between the z axis of the vehicle and the z axis of the world frame in order to keep the vehicle in a position suited to navigation. For sake of clarity, it is useful to check how the reference task is computed:

$$\bar{\rho}(\theta, \bar{n}) = ReducedVersorLemma({}^{v}\hat{k_w}, {}^{v}\hat{k_v})$$

$$\hat{n} = \frac{\bar{\rho}}{\|\bar{\rho}\|} \tag{5}$$

$$\dot{x}_h a = K_{ha} * (0 - \|\bar{\rho}\|) \tag{6}$$

where:

- $\bar{\rho}$ is the displacement vector between two frames computed with the *ReducedVersorLemma* function;
- ${}^{v}\hat{k_w}$ is the z versor of the world frame projected on the vehicle frame;
- ${}^{v}\hat{k_v}$ is the z versor of the vehicle frame projected on the vehicle frame;
- $\hat{n}$ is the displacement versor;
- K_{ha} is the control gain applied to the relative task reference.

Once the computation of the task reference has been discussed, the Jacobian relationship is computed as follows:

$$J_{ha} = \begin{bmatrix} \mathbb{0}^{1x6} & | & \mathbb{0}^{1x3} & \hat{n}' \end{bmatrix} \tag{7}$$

The horizontal attitude task is defined as a safety task and it is implemented with the highest priority. It means that, if the goal position of the vehicle has either a pitch or roll value over the boundaries it will never reach the exact goal orientation. The effect of the Horizontal Attitude Control task is depicted in Figure 2. We set the goal pitch value to *pi/3 (rad)*, which is more than the one allowed values by the safety task. We can clearly see that the vehicle can not reach the desired pitch value as shown in Figure 3.

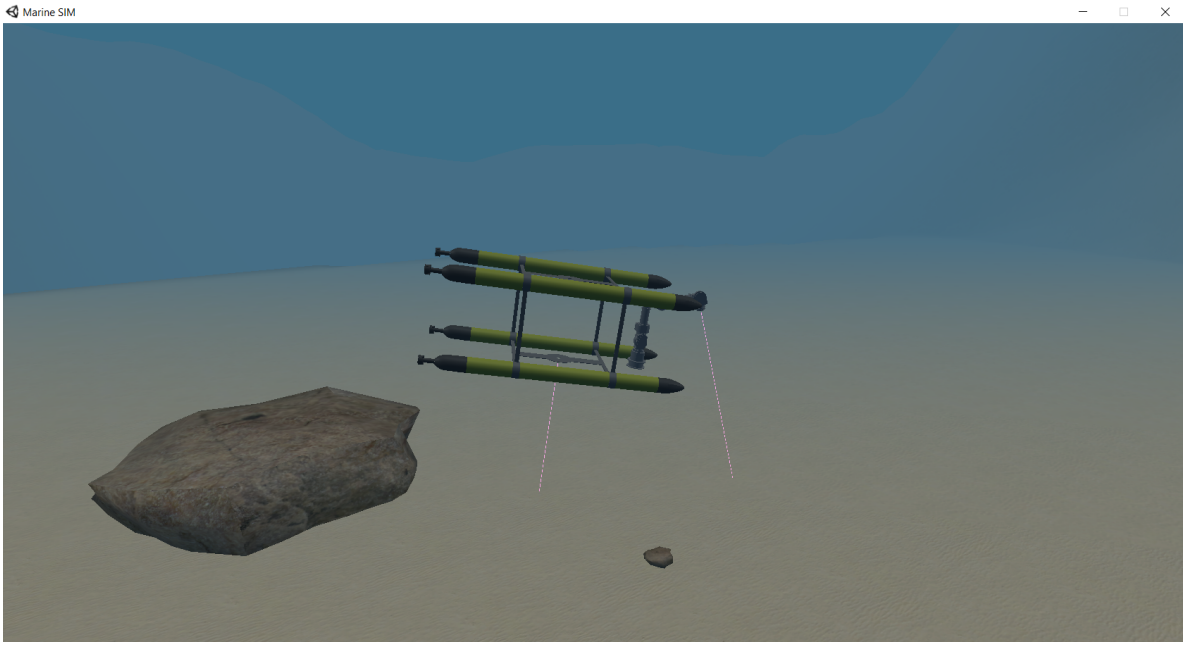


Figure 2: Final vehicle position with Horizontal Attitude Control at highest priority.

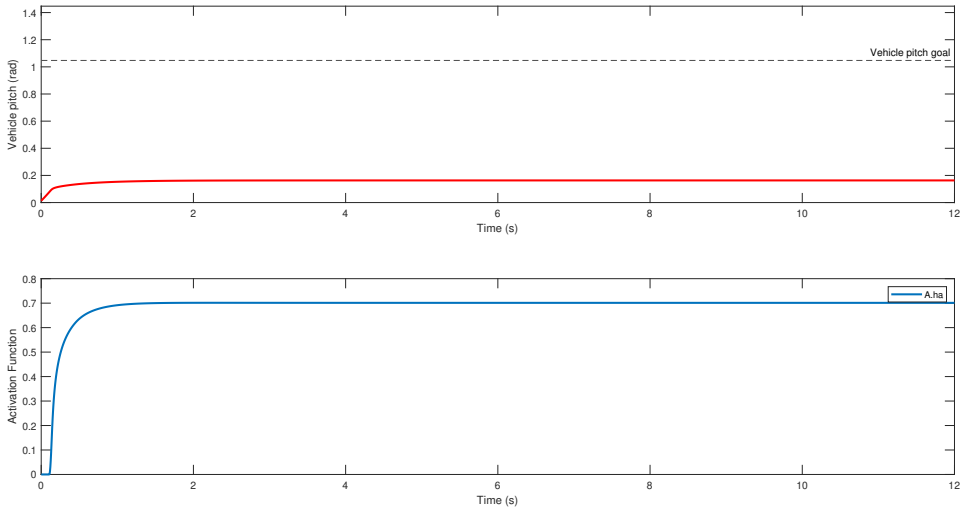


Figure 3: Time evolution of the vehicle pitch when horizontal attitude control task highest priority - Activation function for horizontal attitude control task

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

If we switch the priorities between Horizontal Attitude control task and Vehicle Attitude Control task the final vehicle pitch angle is the desired one, but this can be dangerous since the vehicle is in a unsafe position for navigation. In Figure 4 and Figure 5 it is possible to see the consequences of *swapped-priority* case.

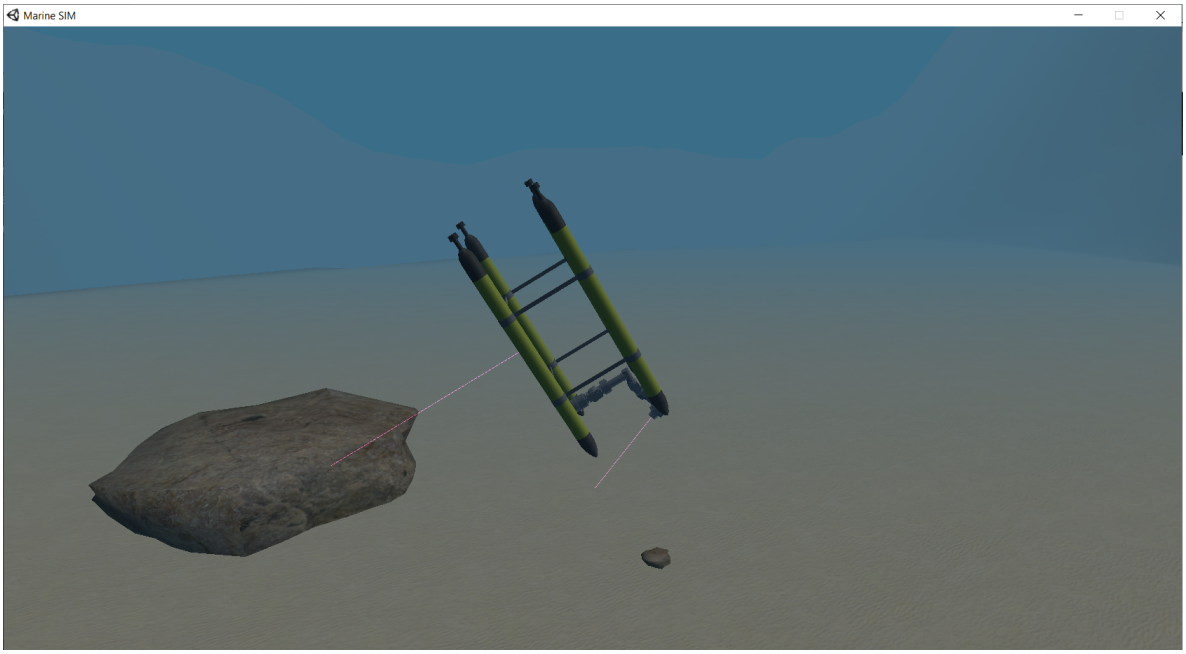


Figure 4: Final vehicle position with Horizontal Attitude Control at lower priority than vehicle control task.

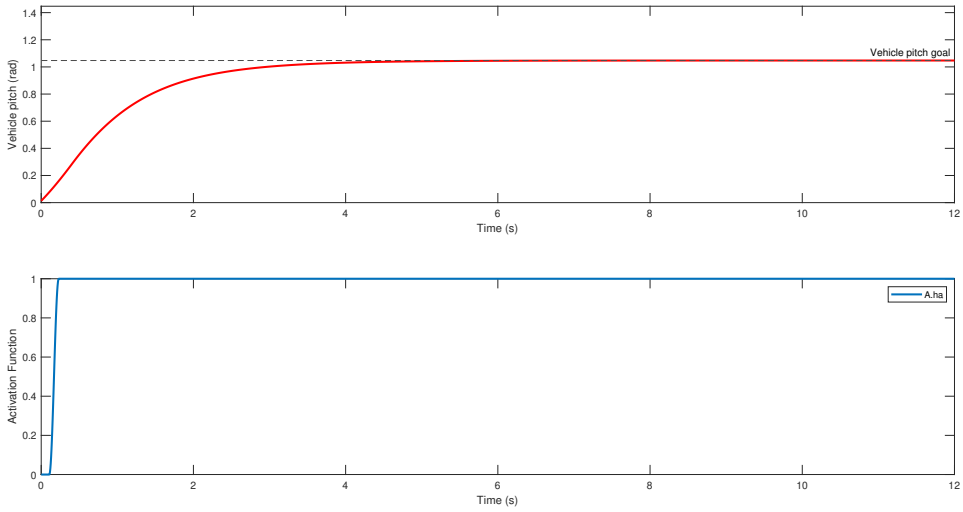


Figure 5: Time evolution of the vehicle pitch when vehicle attitude control task highest priority - Activation function for horizontal attitude control task

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.

The tool position control task (**Task C**) is used to control the manipulator and to rigidly move the vehicle to let the end-effector (from now on we will refer to is as *ee*) to reach the tool goal position. If the task in enabled, we would see the manipulator moving in order to reach the tool goal position. We have to assign to this task the lower priority, otherwise the vehicle control task would result ineffective in the execution of the simulation: if the **Task C** had an higher priority, the vehicle would rigidly follow the motion of the manipulator ignoring the vehicle goal pose imposed by the vehicle control

task, with the risk of having the vehicle in unsafe positions as depicted in Figure . So a good tasks priority order for a *Safe Waypoint Navigatrion* is:

Tasks	Type	Action 1
Task A	I	Active
Task B_1	E	Active
Task B_2	E	Active
Task C	E	Active

The priority levels are:

$$\mathbf{A_1} : \mathbf{A} \prec \mathbf{B_1}, \mathbf{B_2} \prec \mathbf{C}$$

The priority levels above mentioned will guarantee a *Safe Waypoint Navigation* but has the drawback that the vehicle goal position is too far from the tool goal position and the *ee* will never reach it.

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.

In this case we're adding to the previous hierarchy of tasks an inequality task (**Task D**) for safety navigation. As before, the priority of this task must be higher than the vehicle control task and the tool position control task (**Task C**) has been deactivated since now is required to perform *Safe Waypoint Navigation*) only.

Tasks	Type	Action1
Task A	I	Active
Task B_1	E	Active
Task B_2	E	Active
Task C	E	Inactive
Task D	I	Active

The priority levels are:

$$\mathbf{A_1} : \mathbf{A}, \mathbf{D} \prec \mathbf{B_1}, \mathbf{B_2}$$

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 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? NB: Questions 2 and 4 are dealt with together

The Jacobian relationship is obtained considering the sensor distance from the seafloor. **N.B.** We have to point out that the sensor measures the distance of the seafloor ${}^v\bar{d} = [0; 0; d]$ along the vehicle z axis. To obtain the vehicle altitude projected on the world frame we have to compute the jacobian as follow:

$$J_{valt} = -{}^{w}\hat{k_w}' * J_{vpos} \tag{8}$$

where:

- ${}^{w}\hat{k_w}$ is the z versor of the world frame projected on the world frame;
- J_{vpos} is the one computed in equation (1).

The task reference is computed using the information coming from the sensor:

$$ {}^{w}a_v = {}^{w}\hat{k_w} * {}^{w}_{v}T * {}^{v}\bar{d} \tag{9}$$

$$\dot{x}_{alt} = -K_{alt}({}^{w}a_v) \tag{10}$$

where:

- ${}^{w}a_v$ is the vehicle altitude projected on the world frame;
- ${}^{v}_{w}T$ is the transformation matrix between the vehicle and the world frame;
- K_{alt} is the control gain applied to the relative task reference.

Below in Figure 6 we can see the result of this test: the activation threshold are set at $x_{min} = 1.0$ and $x_{max} = 1.5$ meters. We can clearly see that the vehicle remains in the admissible interval for all the duration of the simulation and never decrease under the minimum altitude threshold, so the safety of the system is guaranteed.

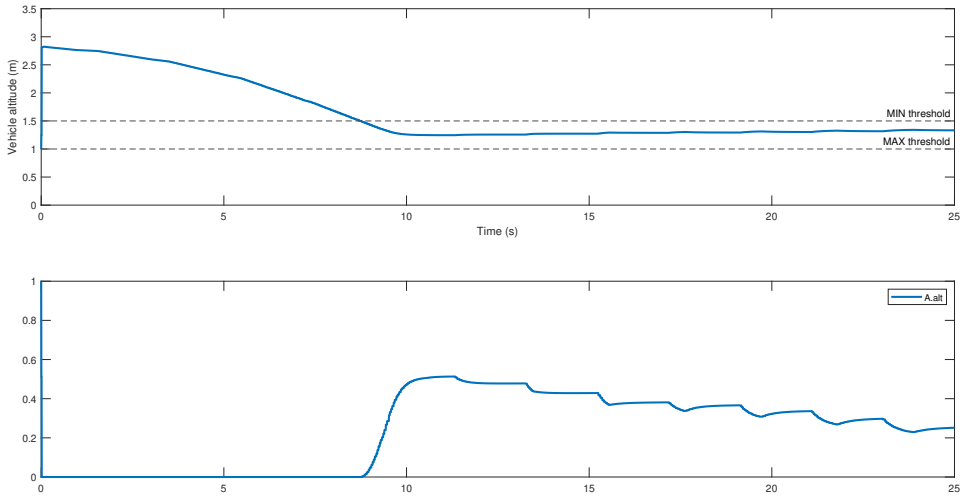


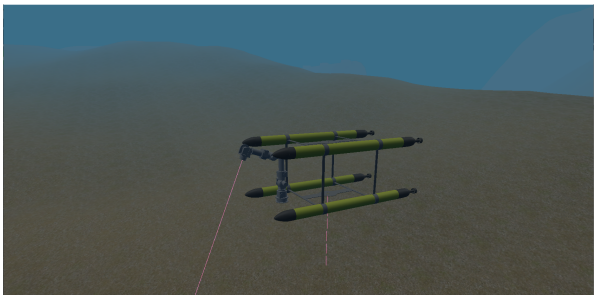
Figure 6: Vehicle altitude and Activation function for keep a minimum distance from the seafloor.

WARNING: The side effect of using a sensor of this type is that the implementation could be less effective with sudden changes in the slope of the seabed and it is therefore necessary to make the assumption of mostly flat seafloor for this particular case.

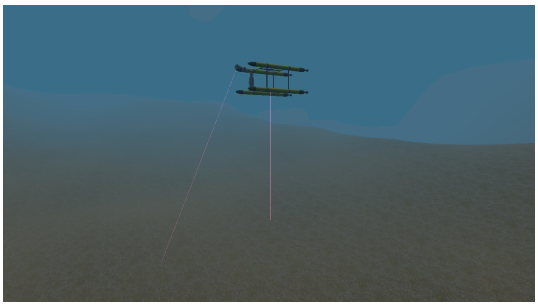
1.2.4 Q3: Try imposing a minimum altitude of 1, 5, 10 m respectively. What is the behaviour? Does the vehicle reach its final goal in all cases?

The choice of the minimum distance allowed from the seafloor is fundamental for the robot behaviour. In fact, if it is set to 1 meter, for example, given the previous goal coordinates, the robot reaches the goal staying away from the seafloor during the shift (Figure 7a). If we increase the minimum altitude threshold (*MinimumAltitude* $>$ 2 m) the **task D** starts to conflict with the Vehicle control position task and since it's priority is higher, the vehicle cannot reach the vehicle goal (the final vehicle position

is above the goal, since the vehicle cannot move along z axis lower than a certain value imposed by the threshold of the activation function of the safety task for altitude control, Figure 7b). A demonstration of this behavior can be seen in Figure 8, where the altitude of the vehicle, the activation function and the final robot positions are compared: we can see that the robot doesn't reach the required goal along *z-axis*, staying at least 5 meters away from the seafloor. While in Figure 9 it's possible to observe the best behavior. With a minimum altitude of 1 meter we can conclude that the vehicle respect the safety altitude and reach the final goal.



(a) Final vehicle position with minimum altitude control set to 1 meter
GOAL REACHED



(b) Final vehicle position with minimum altitude control set to 5 meter
GOAL NOT REACHED

Figure 7: Final vehicle position with different minimum altitude control

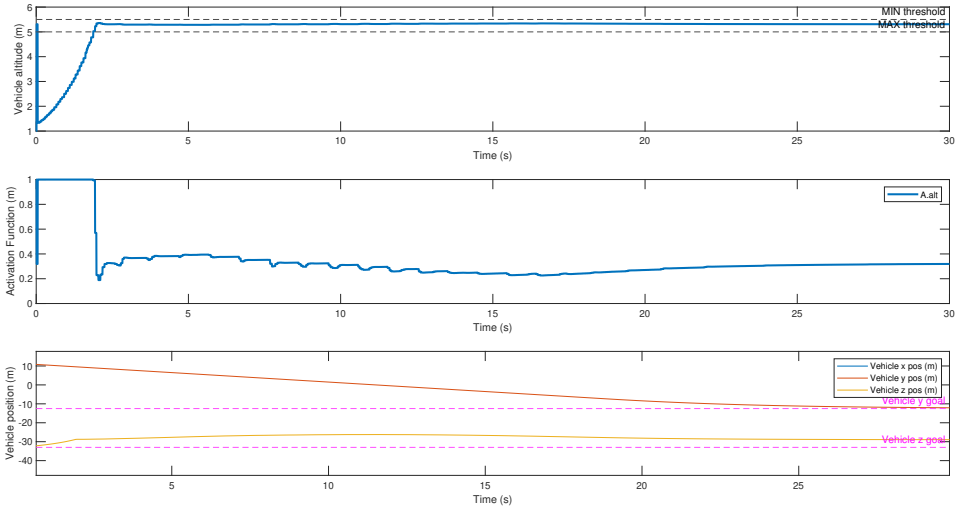


Figure 8: Altitude, Activation Function, Vehicle position - NOTE THAT THE Z GOAL IS NOT REACHED

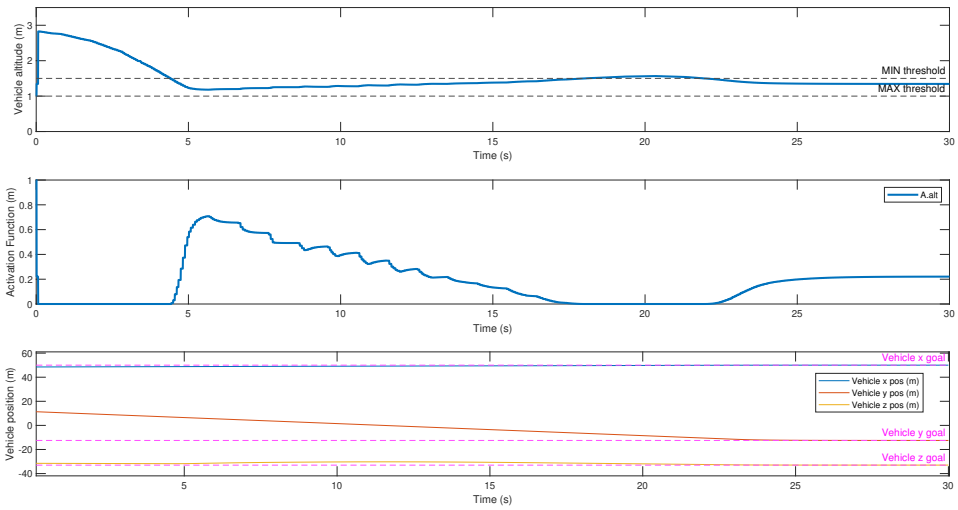


Figure 9: Altitude, Activation Function, Vehicle position - NOTE THAT THE Z GOAL IS REACHED

Finally below is attached the main parts of the Matlab code used for the first exercise (Fig.10 & 11).

```
%% Ex 1 - Implement a Safe Waypoint Navigation Action
% 1.1 - Adding a vehicle position control objective
uvms.Jvpos = [zeros(3,7) uvms.wTv(1:3,1:3) zeros(3,3)]; %jacobian Matrix for vehicle position
uvms.Jvatt = [zeros(3,7) zeros(3,3) uvms.wTv(1:3,1:3)]; %jacobian Matrix for vehicle orientation
% 1.2 - Adding a safety minimum altitude control objective
uvms.w_kw = [0 0 1]'; %Versor of z-axis of word frame
v_kv = [0 0 1]'; %Versor of z-axis of vehicle frame
uvms.v_iv = [1 0 0]';
v_dist = [0 0 uvms.sensorDistance]';
w_dist = uvms.wTv(1:3,1:3)*v_dist;
uvms.w_altitude = uvms.w_kw' * w_dist;  % the altitude is the projection of the sensor distance vector on k versor of <w>
uvms.Jalt =  uvms.w_kw' * [ zeros(3,7) uvms.wTv(1:3,1:3) zeros(3,3)];
% 1.3 - Adding a horizzontal attitude control
v_kw = uvms.vTw(1:3,1:3) * uvms.w_kw; %project versor of word frame pre-multiplying for vRw
uvms.v_rho = ReducedVersorLemma(v_kw,v_kv);
v_n = uvms.v_rho/(norm(uvms.v_rho));
uvms.Jha = [zeros(1,7) zeros(1,3) v_n'];
```

Figure 10: Implementation ex 1 - part 1

```
%% Ex 1
% Compute the vehicle position task reference
[w_vang, w_vlin] = CartError(uvms.wTgv , uvms.wTv);
uvms.xdot.vpos(1:3,:) = Saturate(0.5*w_vlin, 0.5); %increase 0.5 -> HIGH GAIN more fast !
uvms.xdot.vatt(1:3,:) = Saturate(0.5*w_vang, 0.5);
% Compute minimum altitude task reference
uvms.xdot.alt = Saturate(0.2 * (2.0 - uvms.w_altitude), 0.2);
```

Figure 11: Implementation ex 1 - part 2

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.

For what concern the priority hierarchy, this new landing task **Task E** is an equality task, and, since it is an action oriented objective, it must have a lower priority than the *safety tasks* **A** and **D**. Below the chosen hierarchy for the landing action.

Tasks	Type	Action1
Task A	I	Active
Task B1	E	Inactive
Task B2	E	Inactive
Task C	E	Inactive
Task D	I	Inactive
Task E	E	Active

The priority levels are:

$$\textbf{A1: A} \prec \textbf{E}$$

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

The jacobian relationship is computed as follows:

$$J_{land} = {}^w\hat{k}'_w * J_{vpos} \tag{11}$$

While the task reference is:

$$\dot{x}_{land} = -K_{land}({}^w a_v) \tag{12}$$

where:

- ${}^w a_v$ is the vehicle altitude projected on the world frame;
- K_{land} is the control gain applied to the relative task reference.

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

The main difference between this new task and the Minimum Altitude Task is that the former is not an inequality safety task since it is an *action* oriented objective, and it has a lower priority then the Task D. Another difference is found in the computation of the task reference vectors since in the Task D the objective is to go away from the seabed possibly endlessly (the task is activated only when necessary by the activation function) while the Task E has the opposite objective of bringing the vehicle towards the zero altitude position.

2.2 Adding mission phases and change of action

Initialize the vehicle at the position:

$$\begin{bmatrix} 8.5 & 38.5 & -36 & 0 & -0.06 & 0.5 \end{bmatrix}^\top$$

Use a "safe waypoint navigation action" to reach the following position:

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

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

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

The updated list of tasks is the following:

Tasks	Type	Action1	Action2
Task A	I	Active	Active
Task B1	E	Active	Inactive
Task B2	E	Active	Inactive
Task C	E	Inactive	Inactive
Task D	I	Active	Inactive
Task E	E	Inactive	Active

The priority levels for each action are:

$$\textbf{A1: A, D} \prec \textbf{B1, B2}$$

$$\textbf{A2: A} \prec \textbf{E}$$

While the unified hierarchy of tasks is:

$$\textbf{U: A, D} \prec \textbf{B1, B2, E}$$

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

The transition between the two actions is implemented measuring the linear part of the cartesian error between goal frame and vehicle frame: if is lower than a certain threshold the mission phase is updated and and the program switch to Action 2. Below the implementation of the code (Figure 12):

```
1   function [uvms, mission] = UpdateMissionPhase(uvms, mission)
2       switch mission.phase
3           case 1
4
5               [uvms.ang_v, uvms.lin_v] = CartError(uvms.wTgv , uvms.wTv);
6               if (norm(uvms.lin_v)<0.1)
7                   mission.phase = 2;
8                   mission.phase_time = 0;
9               end
10          case 2
11      end
12  end
```

Figure 12: Implementation for switch between action 1 and action 2

The presence of multiple actions made necessary to improve the computation of the activation function as follows:

$$a(x, p) = a^i(x)a^p(p) \tag{13}$$

where:

- $a^i(x)$ represents the activation functions computed up to now for each task i;
- $a^p(p)$ is a continuous sigmoidal function which activate each task in the respective action.

Finally below the main parts of the code used for the Exercise 2 (Figure 13,14).

```
%% Ex 2 - Implement a Basic Landing Action.
% 2.1 - Adding an altitude control objective
uvms.v_altitude = v_kv'*[0; 0; uvms.sensorDistance];
uvms.Jcalt = v_kv' * [zeros(3,7) uvms.wTv(1:3,1:3) zeros(3,3)];
```

Figure 13: Implementation ex 2 - part 1

```
%% Ex 2
vel_landing = 1.0;
uvms.xdot.calt = - vel_landing * (uvms.v_altitude - 0); % GAIN INDIPENDENTLY SOLUTION
% Task reference for control horizzontal attitude
uvms.xdot.ha = 0.2 * (0 - norm(uvms.v_rho));
```

Figure 14: Implementation ex 2- part 2

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.

In the previous case, during the landing we don't have any control on the vehicle orientation or position, so if we want to grasp something on the seafloor, it can be a problem because we aren't sure that the *ee* can reach the object after the landing due to the vehicle orientation. To overcome this possible issue, we can improve the second action in order to align the robot to the object during the landing, using an objective task that is a prerequisite for the execution of the current action (**Task F**), such that if the misalignment between the vehicle's longitudinal axis and the goal's longitudinal axis is higher than a predefined threshold, the task is activated.
The updated list of tasks is the following:

Tasks	Type	Action1	Action2
Task A	I	Active	Active
Task B1	E	Active	Inactive
Task B2	E	Active	Inactive
Task C	E	Inactive	Inactive
Task D	I	Active	Inactive
Task E	E	Inactive	Active
Task F	I	Inactive	Active

The priority levels are:

A1: A, D $\prec$ B1, B2

A2: A $\prec$ F $\prec$ E

The strategy adopted requires a first action (**A1**) equal to the **A1** of Exercise 2.2.1. While the robot is reaching the goal, the Cartesian error between $<v>$ and $<g>$ (vehicle goal position frame) decreases and, under a given threshold, the logic of the program switches the mission to Action 2. The new task **Task F** used to align the vehicle to the rock becomes active and drives the robot to orient its x axis towards the goal and is execute with an higher priority than the landing task since it is defined as a prerequisite. The execution of the simulation shows how the robot initially moves towards the vehicle position goal and after it reaches it, the vehicle begins its landing procedure successfully facing the rock.

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

Given that $\mathbf{g}_w = [x_g, y_g, z_g]$ is the *end-effector* goal position projected on the word frame, we have to evaluate it w.r.t. the $<v>$ frame:
${}^v g_w = {}^v_w T * [\mathbf{g} \mid 1]'$
We project the so found goal vector ${}^v g_w$ to the xy plane of $<v>$ using the projection matrix as follow:

$$ {}^v g_{projected} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * {}^v g_w \tag{14} $$

Now we can compute the displacement rotation vector between the vehicle x axis and the unit vector joining the vehicle frame to the rock frame ${}^v g_{projected}$:

$$ \bar{\rho}(\theta, \mathbf{n}) = ReducedVersorLemma({}^v g_{projected}(1:3), {}^v\hat{i}_v) $$
$$ \hat{n} = \frac{\bar{\rho}}{\|\bar{\rho}\|} \tag{15} $$

Finally the jacobian relationship for the alignment is computed:

$$ J_{valign} = \begin{bmatrix} \mathbb{0}^{1x6} & \mid & \mathbb{0}^{1x3} & \hat{n}' \end{bmatrix} \tag{16} $$

While the task reference is computed as:

$$ \dot{x}_{align} = -K_{align} * \|\bar{\rho}\| \tag{17} $$

Similarly to horizontal attitude task, the objective is to minimize the displacement rotation vector in order to perfectly face the rock with the vehicle.

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 experiments were carried out using the following gains:

- $K_1 = 0.06$
- $K_2 = 0.6$
- $K_3 = 6$

The results for each test are shown in the Figure 16, 15, 17) . We can clearly notice that, if the gain is too small, the alignment task is satisfied only after the landing procedure (Fig. 15. Such a behaviour is dangerous for the vehicle integrity since it would scratch with the seabed to reach the desired orientation. In the other two cases the results are satisfactory as shown in Figure 16 (we show only the plot of the case with K_2 since the case with K_3 has a similar behaviour): with K_2 the alignment is accomplished together with the landing, while with K_3 it is achieved before. A high gain such K_3 could improve the alignment procedure but is preferable to avoid higher gain values to guarantee the system integrity.

Despite the results just presented, we prefer to implement a gain-independent approach to avoid any issues regarding gain's tuning. The idea is to relate the gain of the task to the altitude of the vehicle, since we're sure that the robot is descending during the execution of the action. So we implemented the task reference as follows:

$$ \dot{x}_{align} = -K_{align} * (1/({}^w a_v + \varepsilon)) * \|\bar{\rho}\| \tag{18} $$

In fact, if the altitude tends to decrease to zero the gain of the alignment task increase and allows to complete the aligning task together with the landing task. To test this aspect we reproduced the first

experiment with K_1 and as we can see in the graph in Figure 17 the two tasks are accomplished together unlike the previous result. Figure 20 shows a final screenshot of the simulation of this experiment.

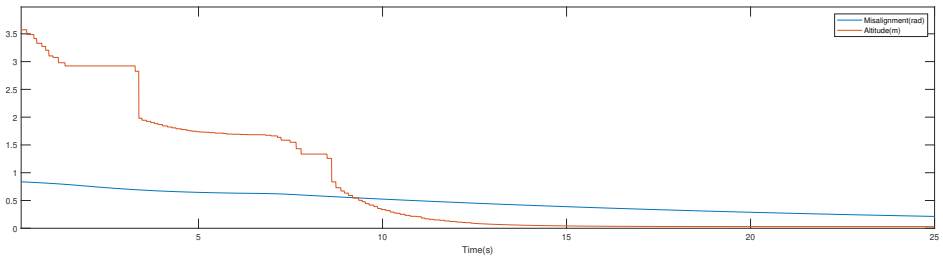


Figure 15: Rock alignment task with gain $K_1 = 0.06$

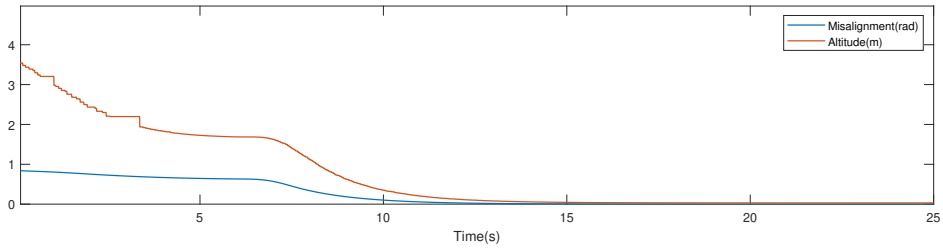


Figure 16: Rock alignment task with gain $K_2 = 0.6$

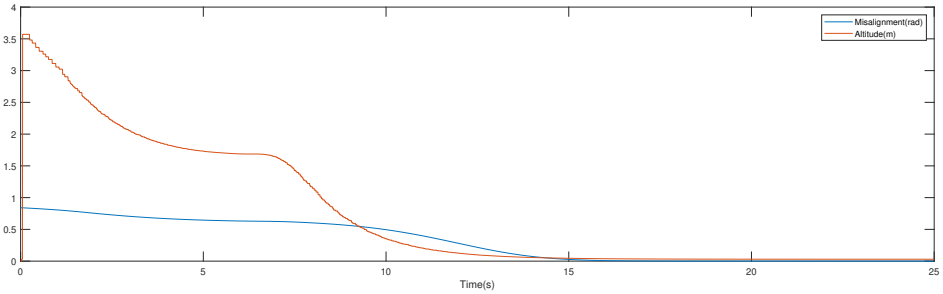


Figure 17: Rock alignment task after gain-independent solution with $K_1 = 0.06$

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?

After the landing and the alignment tasks, the *ee* reaches the rock as show in Figure 20. This happens because we started the landing in an acceptable position, so we needed simply to rotate the vehicle until the orientation matched the goal orientation. If we want to guarantee that the *ee* reaches the rock even if the vehicle start landing in a point that is too far away from the goal, we need to implement a different approach: it is necessary to introduce another prerequisite objective task to reduce the distance to the goal. This new task must have the same priority of **Task F**, since both of them are defined as prerequisite for the action. The idea is to measure the distance between the vehicle and the goal and if it is over a predetermined threshold the new introduced task controls the vehicle toward the goal. The threshold value should be lower than the manipulator work-space boundaries, and the activation function is aimed to activate the task only if the vehicle is too far from the goal and deactivate it when the distance from the goal is lower than the threshold. The task reference finally can be implemented to minimize the Cartesian error between the vehicle position and the rock position (as we have done similarly with the vehicle control task) .

Finally below the code used for exercise 3:

```
%% Ex 3 - Improve the Landing Action
% 3.1 - Adding an alignment to target control objective (rock position)
v_rock_center = uvms.vTw * [uvms.goalPosition; 1];
v_rock_center_projected = [1 0 0 0; 0 1 0 0; 0 0 0 0] * v_rock_center; % project on the x-y plane of <v> frame
uvms.v_rock_vehicle = ReducedVersorLemma(v_rock_center_projected, uvms.v_iv);
if (norm(uvms.v_rho) > 0) % avoid division by zero
    uvms.nv_rock_vehicle = uvms.v_rock_vehicle/norm(uvms.v_rock_vehicle);
else
    uvms.nv_rock_vehicle = [0 0 0]';
end
uvms.Jalg = [zeros(1,7) zeros(1,3) uvms.nv_rock_vehicle'];
```

Figure 18: Implementation ex 3 - Jacobian relationship

```
30      % rock alignment control task
31 -    uvms.xdot.ra = (1/(uvms.altitude+0.01)) * 0.06 * (norm(uvms.v_rock_vehicle));
```

Figure 19: Implementation ex 3- Task reference

Figure 20: Final vehicle and tool position with gain indipendently solution - GOAL REACHED

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?

For now, the vehicle after a Safe Waypoint Navigation tries to reach the rock landing on the seafloor and aligning to the rock but this is still not sufficient for a safe manipulation since the vehicle position is not controlled on the seabed so any type of perturbations might abort the mission. To improve the manipulation, we create a new action, so that now we have a total of 3 action for a more robust, efficient and safe system:

- Action 1: Safe Waypoint Navigation
- Action 2: Landing
- Action 3: Manipulation

The idea is that after reaching the vehicle goal (mission 1) the vehicle starts landing and aligning to the target (mission2) and then the new mission comes in with the intent to secure the position of the vehicle for safety (**Task G**) and start the manipulation task. In the Figure 21 we can see that at the end of the execution the vehicle landed on the seabed aligned with the rock and the manipulator successfully reached the rock even with null vehicle velocities.

Figure 21: Final vehicle and tool position with Null Velocity task - GOAL REACHED

The updated list of tasks is the following:

Tasks	Type	Action1	Action2	Action3
Task A	I	Active	Active	Active
Task B1	E	Active	Inactive	Inactive
Task B2	E	Active	Inactive	Inactive
Task C	E	Inactive	Inactive	Active
Task D	I	Active	Inactive	Inactive
Task E	E	Inactive	Active	Inactive
Task F	I	Inactive	Active	Inactive
Task G	E	Inactive	Inactive	Active

The priority levels are:

A1: A, D $\prec$ B1,B2

A2: A $\prec$ E $\prec$ F

A3: G $\prec$ A $\prec$ C

As we can see in the table the tasks involved in the last action are two: the tool task (**Task C**) for reaching the object with the *ee* and the new task added in this exercise that keeps the vehicle velocity to zero **Task G**. The new task is added with highest priority because it is an objectives related to physical constraints since it has to set to zero the vehicle velocities so that only the *ee* can move. The priority of **Task G** has to be higher than **Task C** and **Task A**.

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

The jacobian relationship for the Null Velocity Task (**Task G**) is the following:

$$J_{zv} = [J_{vpos} \mid J_{vatt}] \tag{19}$$

And the task reference is computed as:

$$\dot{x}_{zv} = \mathbb{0}^{6x1} \tag{20}$$

Finally we can check the results of this test looking at Figure 22: as we expected, when the mission switches to phase 3, the vehicle velocities all converge to zero and remain invariant for the entire duration of the manipulation.

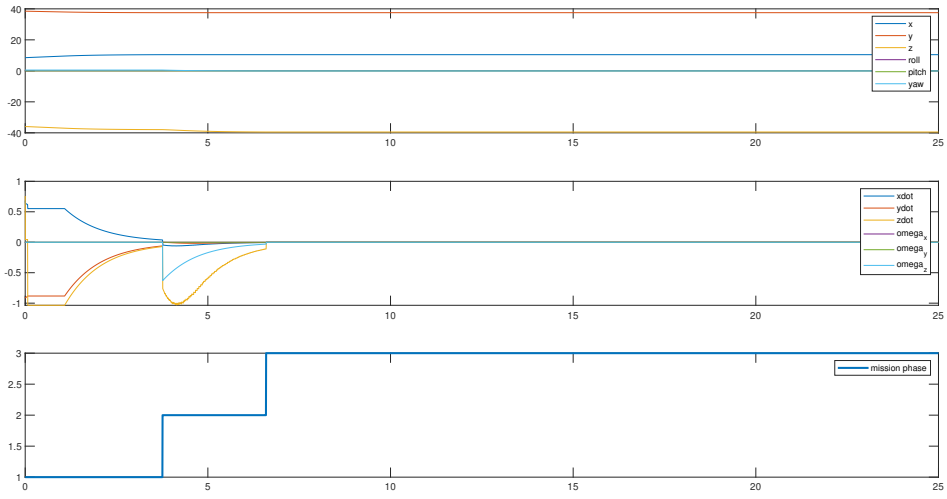


Figure 22: Implementation ex 4 - Activation Functions

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?

If the vehicle floats instead of been anchored to the seabed of course a perturbation, due to the waves, is introduced in the vehicle velocities. This disturbance propagates to the *ee* which in fact cannot reach the goal since is oscillates. If the vehicle moves, the tool must compensate this movements otherwise the disturbances of the floating base would propagate through the coupled kinematics immediately to the *ee* of the manipulator.

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?

The overall tasks are the followings:

Tasks	Type	Action1	Action2	Action3
Task A	I	Active	Active	Inactive
Task B1	E	Active	Inactive	Inactive
Task B2	E	Active	Inactive	Inactive
Task C	E	Inactive	Inactive	Active
Task D	I	Active	Inactive	Inactive
Task E	E	Inactive	Active	Inactive
Task F	I	Inactive	Active	Inactive
Task G	E	Inactive	Inactive	Active
Task H	I	Active	Active	Active

The priority levels are:

A1: A, D, H $\prec$ B1, B2

A2: A, H $\prec$ E, F

A3: G $\prec$ H $\prec$ C

The joint limit task has been implemented with the second highest priority because it is defined as a *safety task* and it has to always be active during the simulation: such task will prevent the manipulator's joints to reach their limit positions since the actuators can be damaged in the motion. The only exception is in **A3** where the **Task H** has priority lower than **Task G** which is an objectives related to physical constraints.

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

The jacobian relationship for the Joint Limit Task is the following (we're acting only on the manipulator configuration):

$$J_{jl} = [\mathbb{I}^{7x7} \mid \mathbb{0}^{7x6}] \tag{21}$$

While the task reference is computed using as follows (q is the joints position vector which are assumed to be measurable):

$$\dot{x}_{jl} = K_{jl} * ((jl_{max} + jl_{min})/2) - q) \tag{22}$$

The idea is to control the joint position toward the average value of the the maximum and minimum threshold positions. Then the activation function would enable the task for each joint only if the joint is reaching any limit position: in order to get this result, the activation function is implemented as the combination of two sigmoid functions, which are at their maximum value toward the joint limits and approach zero away from them. The above mentioned activation function is shown in Figure 23.

```
42    % Joint Limits Task
43    % Ex 4.2
44 -      for i = 1:length(uvms.q)
45 -          uvms.A.jl(i, i) = DecreasingBellShapedFunction( uvms.jlmin(i), uvms.jlmin(i) + 0.1, 0, 1, uvms.q(i) ) +
46 -          IncreasingBellShapedFunction( uvms.jlmax(i) - 0.1,uvms.jlmax(i), 0, 1, uvms.q(i));
47 -      end
```

Figure 23: Implementation ex 4 - Activation Functions

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?

Before moving on, a small summary of all the task involved is dutiful: Task A (horizontal attitude), Task C (tool position), Task D (minimum altitude control task) are taken from the previous exercises. To solve the new request, we're adding the Preferred Arm Shape Task (**Task I**) with lowest priority since is an optimization objective and it must be performed only if the other tasks are accomplished. Below, the hierarchy of task used is presented:

$$\textbf{A1: A, D} \prec \textbf{C} \prec \textbf{I}$$

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

Regarding the jacobian relationship, the idea is to act on the first four joints of the manipulators so we need to extract a sub-matrix from the total jacobian for acting only on the required joints. The final jacobian relationship turns out to be:

$$J_{tool} = [\mathbb{I}^{7x7} | \mathbb{O}^{6x7}] \tag{23}$$

$$J_{tOpt} = J_{tool}(1:4, 1:13) \tag{24}$$

where:

- $\mathbb{I}^{nxn}$ is an identity square matrix of dimension nxn

The reference task is computed using the desired joint position (we assume that the joint positions are always measurable):

$$\dot{x}_{tOpt} = K_{tOtp} * (q_{des} - q(1:4)) \tag{25}$$

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

If the optimization task (**Task I**) is deactivated, all joints of the kinematic chain freely moves to reach the desired tool position: in fact, while the task is active, only the joints 5, 6 and 7 are changing their position over time as we can see in the graphs below where both the cases are presented.

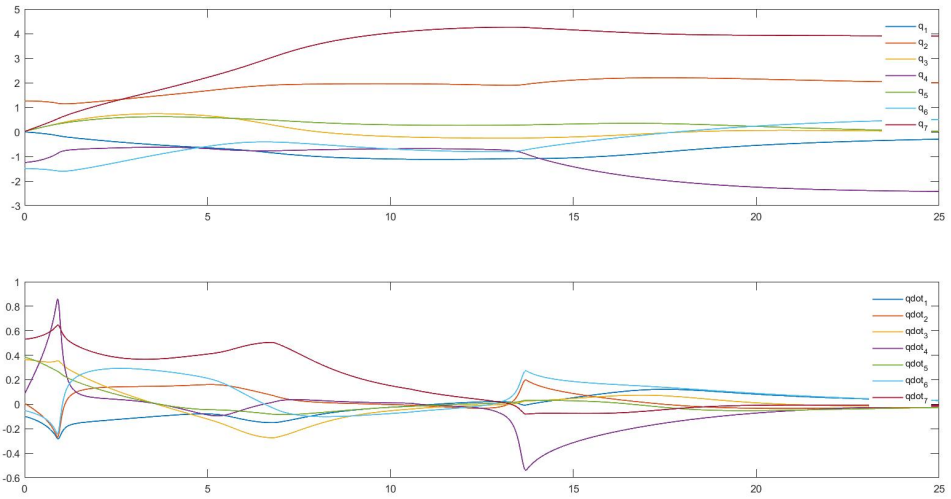


Figure 24: Joint positions and velocities without the optimization task

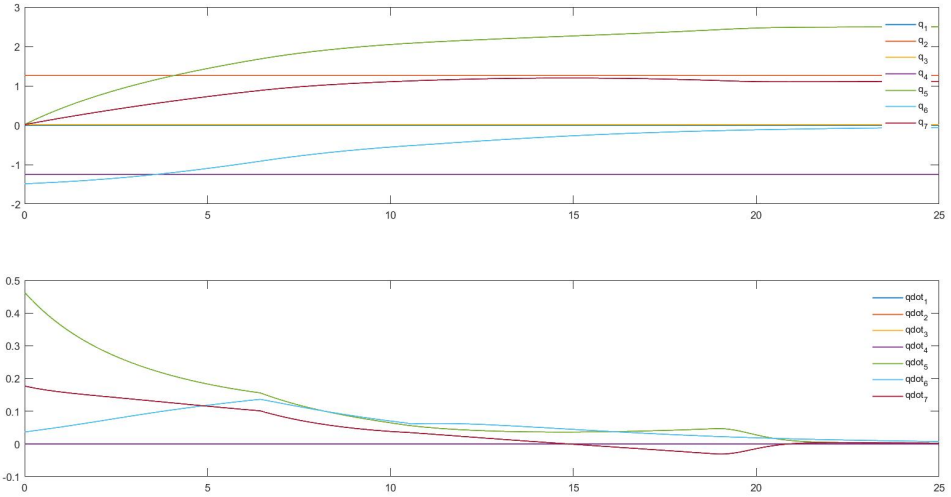


Figure 25: Joint positions and velocities with the optimization task

5.2 Adding mission phases

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

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

Now we're splitting the mission in 2 phases: the first one is represented by the Action 1 (**A1**) for "*Safe Waypoint Navigation*" composed by the tasks of the exercise 1 (**Task A, B1, B2, D**) plus the joint limits task and new optimization task; the second action (**A2**) is the floating manipulation which is performed using the Tool-position control task (**Task C**) and the joint-limit task (**Task H**).

The overall task organization is presented in the table below, and the priority are discussed more precisely for each action.

Tasks	Type	Action1	Action2
Task A	I	Active	Active
Task B1	E	Active	Inactive
Task B2	E	Active	Inactive
Task C	E	Inactive	Active
Task D	I	Active	Inactive
Task E	E	Inactive	Inactive
Task F	I	Inactive	Inactive
Task G	E	Inactive	Inactive
Task H	I	Active	Active
Task I	O	Active	Active

The priority levels for Action 1 (*Safe Waypoint navigation*) are:

$$\mathbf{A1: A, D, H \prec B1, B2 \prec I.}$$

The the priority level for Action 2 (*Floating Manipulation*) are:

$$\mathbf{A2: A, H \prec C \prec I.}$$

While the unified hierarchy of tasks is:

$$\mathbf{U: A, H, D \prec B1, B2, C \prec I.}$$

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

The main improvement obtained by splitting the mission in two phases is that some safety tasks are used only to navigate and the others only to manipulate; therefore, rather than using the activation functions to deactivate superfluous tasks, the change of mission allows a better management of the system logic and simplifies the computation. In the previous implementation (one action only) the minimum altitude task is clearly in conflict with the manipulation phase near the pipe, if we split the mission in two phases, we can keep active the minimum altitude task only during navigation and deactivate it after reaching the required goal position while, the tool position task is taken into account only if we are in Action 2. In fact, the transition between phases is done considering the Cartesian error between the vehicle frame and the vehicle goal frame: we want to switch the mission phase to the manipulation action only if we are sure that the vehicle is in the right 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?

To test the system we added a sinusoidal disturbance $D(t)$, on the speeds along the *x-axis* and *y-axis* of the vehicle, of this form:

$$D(t) = A * sin(\omega * t) \tag{26}$$

We consider for now a fixed frequency, since, considering the scenario, we want to reproduce the waves disturbance which has fixed period (excluding borderline cases) which is computed as follows.

$$T_{waves} = (2 * \pi)/\omega \tag{27}$$

In the experimental case $\omega = 1.3$ so we have a resulting period of 5 seconds which is the waves medium period recorded for the Mediterranean Sea (Climatology of the waves - September Average). The resulting velocity vector is computed as follow:

$$\dot{p} = (D(t) * \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix})' + \dot{p} \tag{28}$$

The task (**Task L**) introduced in this section is a *non-reactive control task* which consider the vehicle as non actuated and has the objective of control the manipulator to compensate this disturbances to keep the *ee* in the desired goal position. Since there is a disturbance on the vehicle velocities, if the manipulator and vehicle motions is not kinematically decoupled, then disturbances of the floating base would propagate through the coupled kinematics of the manipulator. To solve this problem, we need two optimization algorithms running in parallel. The first optimization algorithm considers the vehicle together with the manipulator as a fully controllable system (the manipulator joints velocities are discarded from the control vector). The second optimization algorithm considers the vehicle as totally non-controllable. At this stage, it is added the *non-reactive control task* and together with the tool-position control task it is provided the final manipulator velocity vector (the vehicle velocities are discarded from the control vector). Formally, the steps done to achieve this goal are: ($\dot{p}$ stands for final vehicle velocities, $\dot{q}$ stands for final manipulator velocities):

- Compute jacobian relationship for vehicle control: $J_v = [J_{vpos} \mid J_{vatt}]$
- Compute task reference for vehicle control $\dot{x}_v = -K_v * ({}^w goal - {}^w v_{pos})$
- Compute vehicle velocities (the manipulator output velocities are discarded): $\dot{p} = pinv(J_v) * \dot{x}_v$

Then we move to the second optimization process in which the vehicle is considered as non actuated and now we need the already presented *non-reactive control task* which is implemented as follows:

- Compute jacobian relationship: $J_f = [\mathbb{0}^{6x7} \mid \mathbb{I}^{6x6}]$
- The task reference is the vehicle velocities (assuming they are always measurable): $\dot{x}_f = \dot{p}$
- The output velocities for the manipulator are computed using $\dot{x}_f$ and the output velocities of **Task C** (tool-position control task).
- Finally the control vector is composed of two parts, the output of the optimization 1 for the vehicle velocities and the output of optimization 2 for manipulator velocities.

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.

The experiment proposed in Exercise 6 was solved implementing two different actions: one used to approach the pipe (**A1**) and one used for the manipulation phase on the arm (**A2**). In the Figure 26 it is shown the evolution of the coordinates of the absolute position of the *ee* during the simulation (ee_x, ee_y, ee_z) and the magenta-dashed lines show the three components of the *ee* goal position for this task. For a better understanding of what is going on in this simulation it is necessary to introduce the meaning of Figure 27 & 28: in both of them it is possible to see the different behaviour caused by the change of action happening around $t = 8s$ in which the sinusoidal motion resulting from the disturbances remains the only vehicle motion which takes place in the following part of the execution and, more in detail, the former shows the position of the vehicle in the space projected on the world frame and both its linear and angular velocities computed by the control system, while the latter shows the position of each manipulator joint projected on the vehicle frame and the velocities generated by the priority task controller. It is possible to understand from these plots that the vehicle is under the effect of some disturbances and the parallel arm-vehicle coordination scheme successfully compensate the motion of the robot in order to let the *ee* to remain in a fixed spatial position: it is possible to see that after $t = 15s$ the *ee* frame perfectly reaches the goal position while the joint velocities never stops the compensation process.

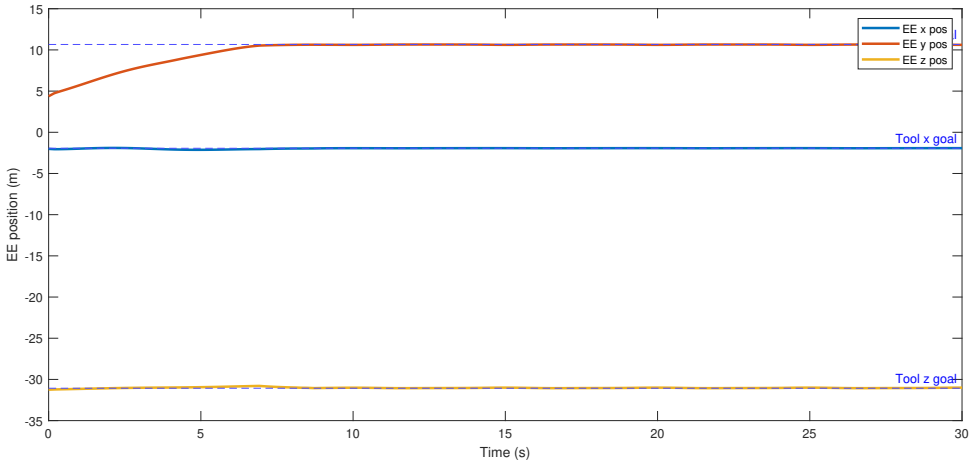


Figure 26: End effector position of floating manipulation task

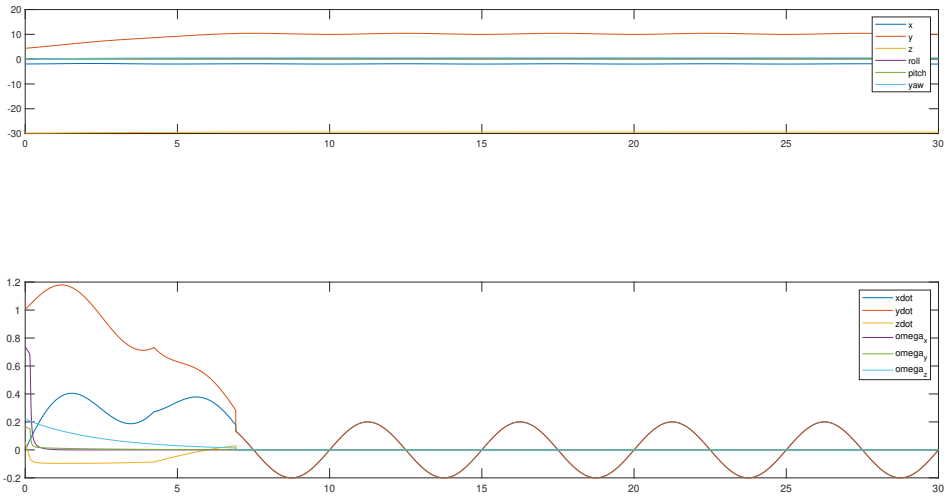


Figure 27: Vehicle position and velocities of floating manipulation task

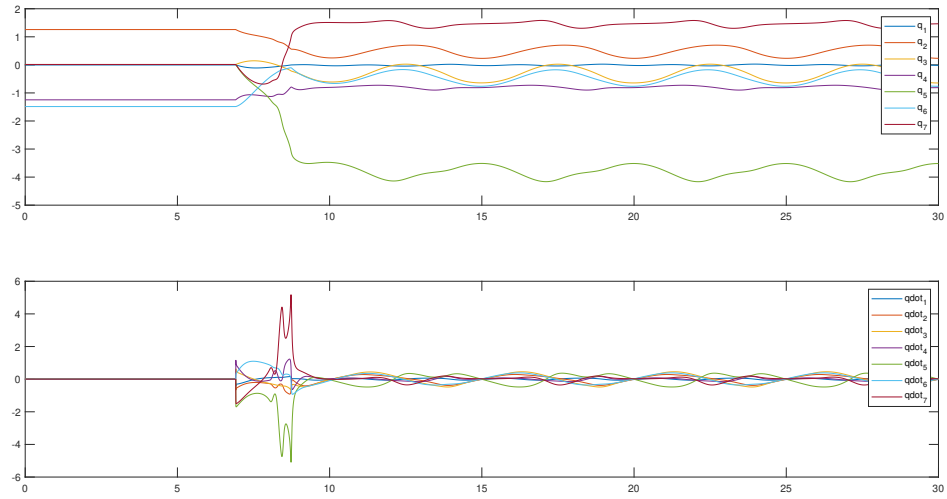


Figure 28: Arm position and velocities of floating manipulation task

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 the system is under the effect of very big disturbances it may happen that the vehicle drifts in space bringing the *ee* away from the goal: in fact, such a behaviour would move the system such as the *ee* goal is outside the workspace and the compensation is not enough to guarantee the fixed position of the *ee*. To test this aspect we increased the amplitude of the sinusoidal disturbance and we noticed that for values of $\mathbf{A} > \mathbf{0.5}$ the compensation cannot keep the tool in the desired position and part of the oscillation of the vehicle is propagated to the *ee* as we can see in Figure 29 where $\mathbf{A} = \mathbf{0.8}$ and the *ee* is clearly oscillating. The manipulator tries to compensate the disturbance as we can see in Figure 30, but the vehicle perturbation (Figure 31) is too big. This result confirms our initial hypothesis presented at the beginning of this section.

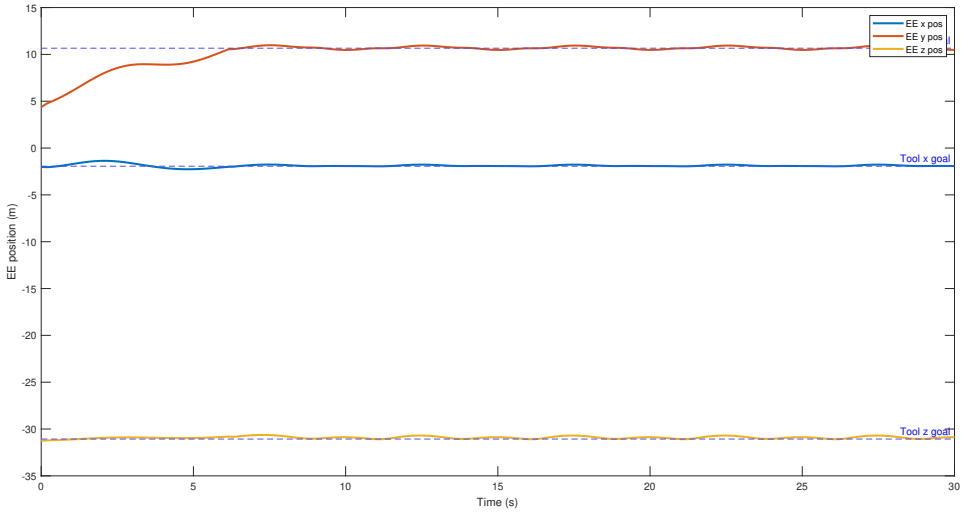


Figure 29: Arm position and velocities of floating manipulation task

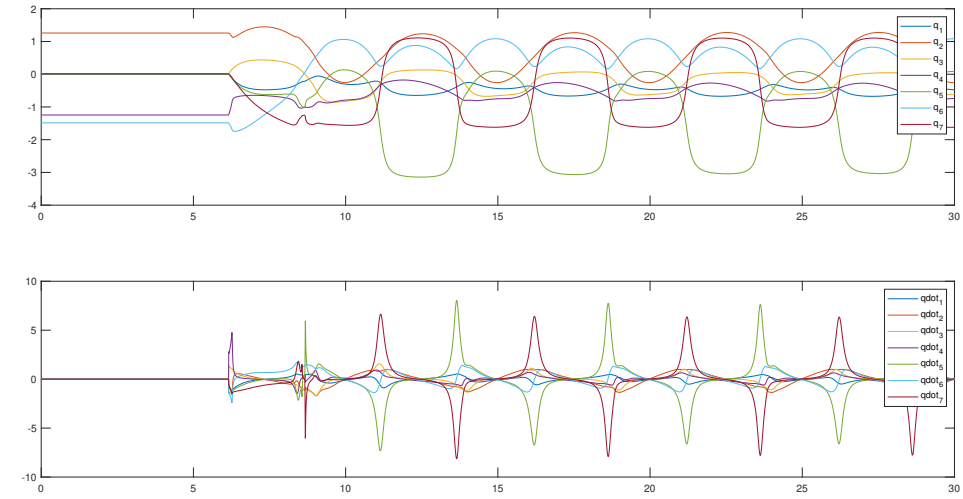


Figure 30: Arm position and velocities of floating manipulation task

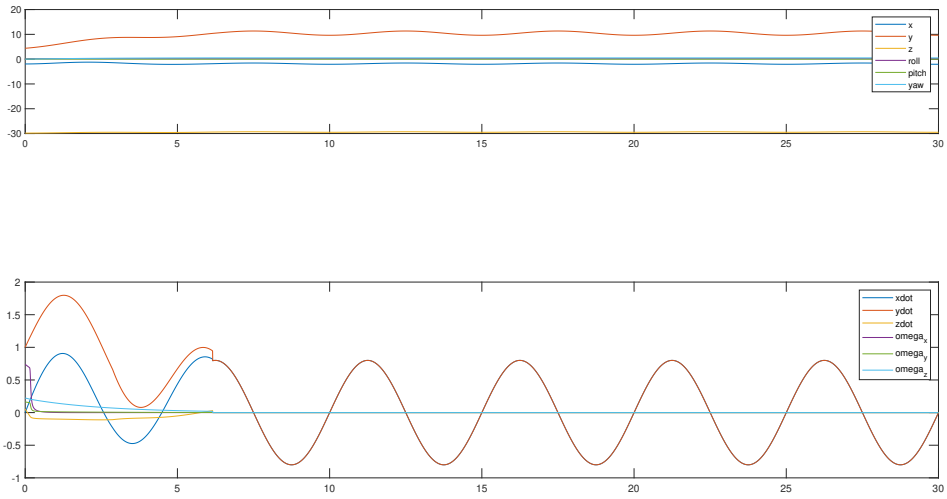


Figure 31: Arm position and velocities of floating manipulation task