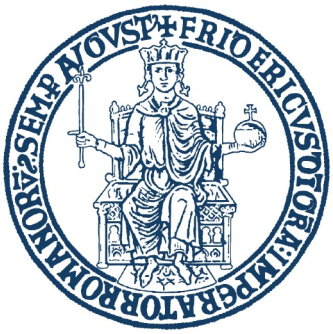

UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

FIELD AND SERVICE ROBOTICS

a.y. 2021/2022

Final technical project:

Cooperative object transport task with multiple UAVs

Ch.mo prof. Fabio Ruggiero

Students:

Michael Mirzaagha

P38000094

Marco Trinchese

P38000111

Contents

List of Figures

1 Introduction to the project

1.1 Idea

This work aims to design two drones able to pick up objects and take them to a new location through a hierarchical control. For each of the two drones, a route search algorithm has been implemented that allows it to avoid obstacles of the environment. The two UAVs must be synchronized because the second UAV takes the object from the place where the first UAV has brought it. The mass of the objects to be picked is not known and it is estimated using an estimator in run time.

This project was chosen due to the interest of both candidates in Aerial Robotics. More and more companies are investing in this sector, understanding the great development margins that this technology can bring. The fields of application of this type of robotics are various: transport, defense, cinema, etc., etc. Both candidates think that the in a close future a great part of the delivery work will be made by drones and also there will be a high request from various companies for unmanned systems capable of moving packages in safe conditions. The presence of many UAVs in the space implies the necessity of coordination between them, and this project tries to solve this issue. The project has set itself the objective not only to apply the topics studied during the Field and Service Robotics course but also to allow us to deepen and acquire skills useful for the working world.

1.2 Tools and packages

For the project we used the software MATLAB and Simulink. In the MATLAB code has been made the trajectory planning for the UAVs, while in the Simulink has been developed a scheme to simulate the control of the drone, its performances, and the estimation of the mass. The UAV toolbox library has been used for the visualization of the UAVs and the development of an environment full of obstacles. Various results will be presented by varying the following parameters

- delta of the RRT
- mass
- order of the estimator
- time of execution
- simulation without estimator

2 Trajectory planning with MATLAB

MATLAB was used for the part of trajectory planning. The code computes the desired position, velocity and acceleration over time for the two UAVs . The two trajectories aren't independent: the takeoff of the second UAV is coincident with the instant when the first UAV is unloaded and the point where the first UAV releases the object is the same where the second UAV takes the object, simulating a cooperative work. The procedure starts with the choice of the extreme points of the travel of each UAV: takeoff, take (the point where the UAV takes the object), put (the point where the UAV is unloaded), and again takeoff.The "put" point of the first UAV coincides with the "take" point of the second UAV. It is simulated a situation where the same load is first moved by the first UAV and then moved again by the second UAV. This explains why the two time laws must be synchronized. The trajectory of the second UAV starts when the first UAV places the mass. So, for each UAV travel, the code executed three times the RTT function to find a path between start and goal: the first time for the "takeoff-take" path, the second time for the "take-put" path, and the third time for the "put-takeoff" part.

The RRT function gives a set of coordinates that are the points of the RRT tree that go from the start point to the goal point avoiding collisions. After uniting the three paths of each UAV, it is assigned a time law to the path, making a trajectory. It is simulated a brief pause at each landing point, so the UAVs can wait some seconds before they continue with their task.

The time law is a polynomial one of the 7th order, realized in such a way that makes the UAV stop at each point of the RRT path.
At the end of the MATLAB code, the velocity and the acceleration are computed, in order to have in the Simulink model the desired position, velocity, and acceleration in x, y and z. The desired yaw angle is computed in Simulink.

2.1 RRT and search in a tree

Given the starting point, the goal point, and the environment (the obstacles), RRT function generates a set of points (their coordinates) that goes from the start to the goal. RTT is a search algorithm designed to build a space-filling tree randomly, in addition to RRT, a search algorithm has been implemented to obtain the route from the start to the goal.

The first step to implement this algorithm is the generation of a random configuration `q_rand` with a uniform probability distribution in the 3-dimensional space. The algorithm ends when the goal has been reached or if the goal has not been reached in a maximum number of attempts. x, y and z are normalized between 0 and 22, this ensures that the generated `q_rand` is inside the domain of the map.

`q_rand` must be connected to the nearest node belonging to the already existing graph (`q_near`), if the distance between this point and the random point is bigger that a a predetermined distance `delta`, `q_rand` is replaced with a point belonging to the segment, which connects the 2 points, but with distance `delta` from `q_near`. For moving on this segment, it is necessary a unit vector that points in the interesting direction, so it is built a unit vector v by dividing the vectors that united `q_rand` and `q_near` by its module, making it unitary but keeping the interesting direction.

With this unit vector v I have a direction pointing from `q_near` to `q_rand`, so it is possible to multiply this unit vector by the chosen length `leng` (delta or the distance between `q_near` and `q_rand`) and obtain a candidate `q_new` that the algorithm is going to check. In fact, before `q_new` is inserted in the graph, it is necessary to check two things: if the `q_new` is in collision with the obstacles; and if the segments that united `q_new` with `q_near` is in collision with the obstacles. These conditions are certified using collision algorithms, in particular the second condition is checked by applying a collision check on a given number of points `check_line` on that segment, and if everything is good, `q_new` is added to the graph.
The code for the collision check is

```
function collision=controlloCollisione(point,meshes_vector,dim)
    test_sphere_collision=collisionSphere(dim);
    test_sphere_collision.Pose=trvec2tform(point);
    collision=0;
    for i=1:size(meshes_vector,2)
        coll = checkCollision(meshes_vector(i),test_sphere_collision);
```

```
        if coll==1
            collision=1;
        end
    end
end
```

This is a collision check made in a 3D space. We didn't check a single point, but we checked if a sphere of a given radius centered in the point that we want to check, so the obtained tree is more strictly distant from the obstacles. This is made with a check collision between the mesh of the sphere and the mesh of the obstacles of the environment. The `dim` variable is the one that indicated the radius of the control sphere. The function `trvec2tform` is a function that converts a translation vector of three elements in a transformation matrix 4x4.
The graph is made by an adjacency matrix `ADJ`, that is enlarged every time we add a new node to our graph. A variable `NODELIST` contains the coordinates of each node of the graph. If the new node is inside a certain sphere of radius `range_goal` that is displaced 5 meters higher from the goal, they are connected and the algorithm stops. This is made because we want a vertical landing of the UAVs. The number of trials for choosing random points `a` is very high because the RRT in 3D needs way more iterations than the 2D one.

After that, the next step is to search for the right path inside the tree that brings from the starting point (the root of the tree) to the goal point (the last leaf attached to the tree). In a tree upside-down, each node has only one father but can have many sons. So, starting from the goal node and looking for its father and iterating the procedure surely will be reached the root node that is the starting node. By saving all the fathers the path is obtained. The code is built in a way that each son has an index greater than its father. So, starting from the goal node, we look up in the adjacency matrix for the only node connected to it, its father. This process is iterated until when the father node found has index 1, which makes it the root node, the only one that can have that index. After that, the RRT function ends and this list of coordinates is given as output.

Many changes can be made in the program to see the effect obtained by changing some values. Can be chosen a bigger delta in the function for more rapid convergence to a solution but it is important to keep an eye on the number of points that we check on each segment of the RRT tree. Some examples of execution will be presented.

2.2 Complete path

Once the RRT function gives as output the set of points between a given start and goal points, it's quite easy to build a complete path that starts from `takeoff`, goes to `take`, goes to `put` and then goes back to `takeoff`. We only need to concatenate the solution of the RRT function obtained by changing properly the starting point ad the goal point: takeoff-take, take-put, put-takeoff. We have a data structure that contains the complete travel of a drone, the `travel` (or `travel2`) matrix:

```
takeoff = [0,0,0.5];
take = [17,10,14.5];
put = [10,15,16.5];
takeoff2 = [0 20 0.5];
take2 = put;
put2 = [10 3 12.5];
travel=[takeoff take; take put; put takeoff];
travel2=[takeoff2 take2; take2 put2;put2 takeoff2];
```

The RRT function is called 3 times, one for each row of `travel` matrix. Between every iteration, there is a small pause to make the UAV stop in the place where it takes/puts the mass, so the work is more realistic.

Now we have all the points of the path. A trajectory is made of a path that is the set of all the geometric points that the point occupied in the time, and of a time law which is the way the drone moves along the path over the time.

2.3 Time law

It has been chosen to move from one point to another following a polynomial of the seventh order over time, but not for all the paths but for every segment between two adjacent points of the path. This choice has been taken because the RRT generates a tree, from the tree we take the path, and from the points of the path we build the geometric path by connecting those points with segments. At the end, we will have a set of segments connected in the points of the path, and in those connections, we will have sharp angular points. We know that in those points we can't have a non-zero velocity because that would mean a peak in the acceleration for change instantaneously the direction of the UAV in the space. We looked for something that guarantees a smooth transition between the segments of the paths and that will guarantee continuity of the time derivative of the acceleration, the jerk. This can be made with a polynomial of the seventh order by imposing a null velocity, acceleration, and jerk ad the starting time and finishing time of the polynomial and imposing the desired position ad the beginning and at the end. We started by making one polynomial of the seventh order with the code

```
    %construction of the seventh order polynomial
numpts_for_segment=1000; %number of points for each segment
tf=1; %final time
lin=linspace(0,1,numpts_for_segment); %time vector
des_vel_0=0; %desired velocity at time 0
des_vel_1=0; %desired velocity at time 1
des_acc_0=0; %desired acceleration at time 1
des_acc_1=0; %desired acceleration at time 1
des_jerk_0=0; %desired jerk at time 1
des_jerk_1=0; %desired jerk at time 1
A=[ 1 0 0 0 0 0 0 0;
    1 1 1 1 1 1 1 1;
    0 1 0 0 0 0 0 0;
    0 1 2 3 4 5 6 7;
    0 0 2 0 0 0 0 0;
    0 0 2 6 12 20 30 42;
    0 0 0 6 0 0 0 0;
    0 0 0 9 24 60 120 210];
b=[0 1 des_vel_0 des_vel_1 des_acc_0 des_acc_1 des_jerk_0 des_jerk_1]';
ai=A\b;
a0=ai(1);
    ...
a7=ai(8);
s_t=a7*lin.^7+a6*lin.^6+a5*lin.^5+a4*lin.^4+a3*lin.^3+a2*lin.^2+a1*lin+a0;
```

The matrix equation `A*ai=b` is the set of eight equations to impose the desired starting and final values for position, velocity, acceleration and jerk. the `ai` is the set of coefficients to find to have the desired polynomial. The result can be seen in figure 1.
With the polynomial, it must be built a curve between two consecutive points of our path. Given two consecutive points i and i+1, the polynomial is scaled and shifted in a way that makes it starts at point i and ends at point i+1. This procedure is iterated for all the points of our path. If the path is made of N points, we will end up with N-1 concatenated polynomial. If each polynomial is made of 1000 points, we will end with a curve of (N-1)*1000 points where the value at i*1000 with i=1...N will be the same value of the element i of the path with i=1...N but between the point i*1000 and (i+1)*1000 there will be a scaled and shifted polynomial curve. This algorithm is implemented in the following code:

```
PATH_TL=[];
for i=1:size(TOTAL_COORD_PATH,1)-1
    new_piece=s_t'*(TOTAL_COORD_PATH(i+1,:)-TOTAL_COORD_PATH(i,:))+...
        TOTAL_COORD_PATH(i,:);
    PATH_TL=[PATH_TL;new_piece];
end
```

The `new_piece` variable is the polynomial scaled and shifted in order to fit between two adjacent points.
After that, it is necessary only to compute the time derivative to have the reference in velocity and acceleration. It has been done it with the command `diff` and a proper scale. The code is

```
tempo=180;
PATH_TL_dot=diff(PATH_TL)/tempo*size(PATH_TL,1);
PATH_TL_dot=[0 0 0;PATH_TL_dot];
PATH_TL_dot_dot=diff(PATH_TL_dot)/tempo*size(PATH_TL,1);
PATH_TL_dot_dot=[0 0 0;PATH_TL_dot_dot];
```

The variable `tempo` is a variable that indicated the total time to make all the three paths of each UAV. NOTE: we do the exact same things for the trajectory planning of the UAV2. The only particular thing is that point `take2` is equal to the put point of the UAV1. The other important difference is the synchronization. The UAV2 starts at the moment when the UAV1 puts the mass in the point `put1`, so it is important to know this time. The time vectors is built with this code:

```
timevec=linspace(0,tempo,size(PATH_TL,1));
RowIdx = find(ismember(PATH_TL,put,'rows'));
timeput1 = timevec(RowIdx(1));
timevec2=linspace(timeput1,tempo+timeput1,size(PATH_TL2,1));
```

The time vector for the trajectory of the UAV2 is shifted forward of `timeput1` seconds.
At the end, the reference values in MATLAB are converted in NED configuration. There are still some values in ENU but their only purpose is to be used in the 3D plots where is needed to give the coordinates values in ENU configuration.

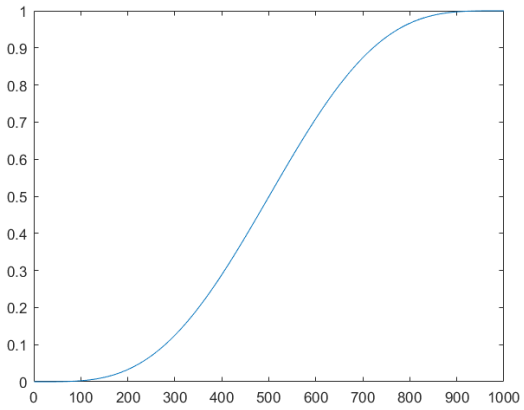

Figure 1: Seventh order polynomial

2.4 Construction of the environment

The environment consists of a space with some obstacles in shape of parallelepipeds in it. These obstacles are described by a mesh file that is also used to the collision check. The part of code for the creation of the environment is

```
Scenario = uavScenario("UpdateRate",100,"ReferenceLocation",[0 0 0]);
ObstaclePositions = [2 16;5 5;6 2;15 16; 10 10;10 3; 17 10; 10 15;17 4];
% Locations of the obstacles
ObstacleHeight = 4;                      % Height of the obstacles
ObstaclesWidth = 1.5;                       % Width of the obstacles
for i = 1:size(ObstaclePositions,1)
    addMesh(Scenario,"polygon", ...
        {[ObstaclePositions(i,1)-ObstaclesWidth*i/7 ...
```

```
        ObstaclePositions(i,2)-ObstaclesWidth*i/7; ...
        ObstaclePositions(i,1)+ObstaclesWidth*i/7 ...
        ObstaclePositions(i,2)-ObstaclesWidth*i/7; ...
        ObstaclePositions(i,1)+ObstaclesWidth*i/7 ...
        ObstaclePositions(i,2)+ObstaclesWidth*i/7; ...
        ObstaclePositions(i,1)-ObstaclesWidth*i/7 ...
        ObstaclePositions(i,2)+ObstaclesWidth*i/7], ...
        [1.3*i ObstacleHeight*i/2]},0.651*ones(1,3));
end
meshes_vector=[];
for i=1:size(Scenario.Meshes,2)
    meshes_vector=[meshes_vector ...
        collisionMesh(Scenario.Meshes{1,i}.Vertices)];
end
```

There are 9 solids in the environment of different sizes and at different heights. It can be see in figure 2.

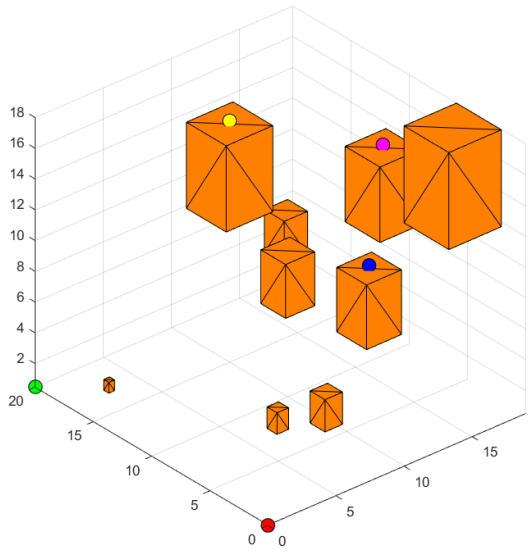

Figure 2: Environment and obstacles. green is takeoff of UAV2, red is takeoff of UAV1, magenta is take of UAV1, blue is put of UAV2, yellow is take of UAV2 and put of UAV1.

2.5 Trajectory result

After this explanation, it will be interesting to see some results. We started with delta=5, range_goal=3, check_line=5.

The results are represented in the figures 3, 4, 5 and 6. From figures 3 and 5 it is possible to notice that the starting time of the two trajectories is not the same: the second one is shifted forward to the moment when the first one reaches the point put for the UAV1, at around 110 seconds. From the two images, it is possible to notice the two drones remain stationary for a few moments when they take and place the mass and, at last, it is possible to see the polynomial shape between two adjacent points.

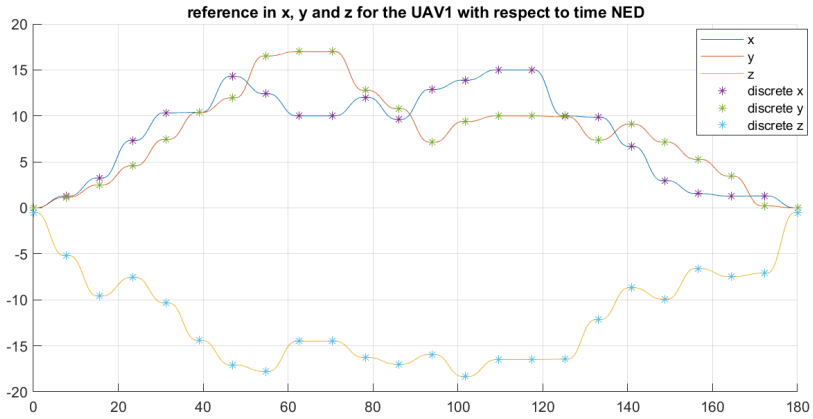

Figure 3: (test 1) Desired position with respect to time for the UAV1

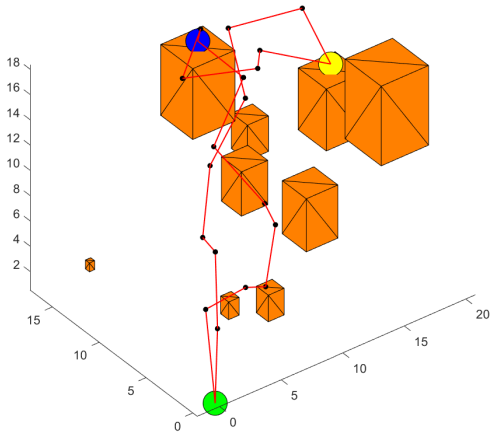

Figure 4: (test 1) The complete path in 3D for the UAV1. green is takeoff, yellow is take, blue is put

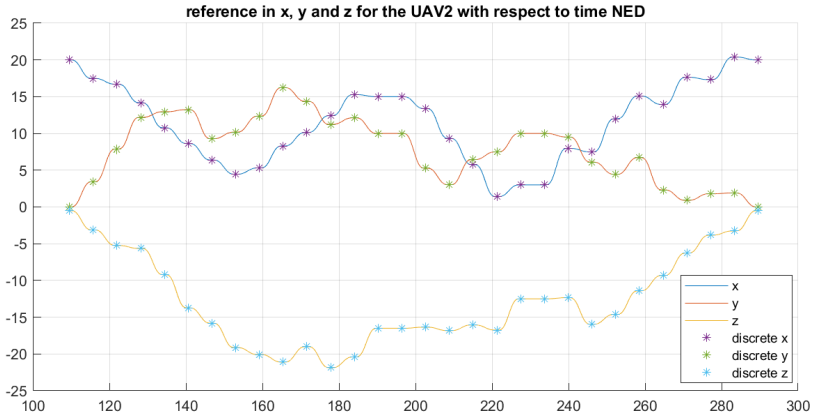

Figure 5: (test 1) Desired position with respect to time for the UAV2

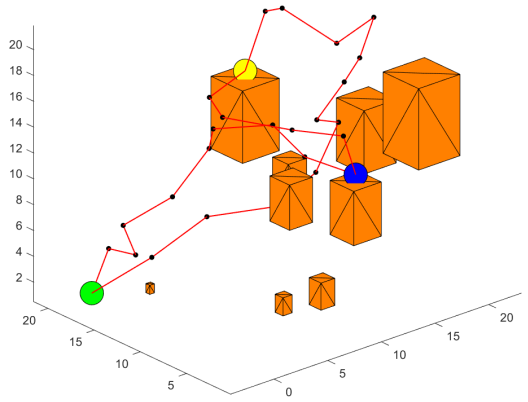

Figure 6: (test 1) The complete path in 3D for the UAV2. green is takeoff, yellow is take, blue is put

It's also interesting to observe the reference in acceleration and velocity that will be used in Simulink. They can be seen in figure 7, 8, 9 and 10. It is possible to see that both velocity and acceleration pass for zero at each step. We can see that they are continuous, as we would expect with a seventh order polynomial.

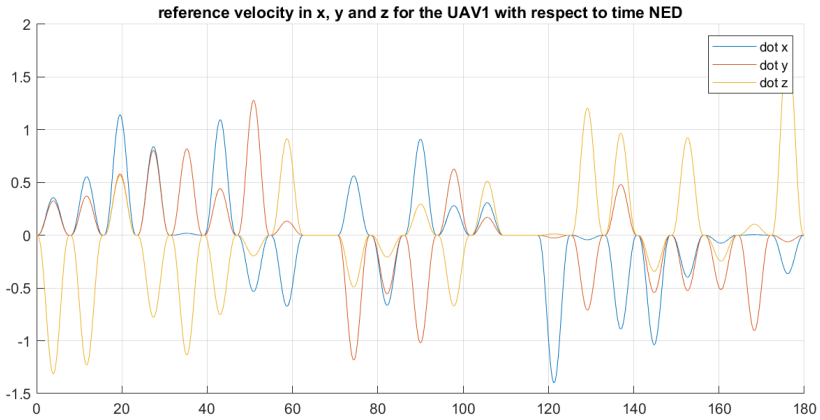

Figure 7: (test 1) Desired velocity with respect to time for the UAV1

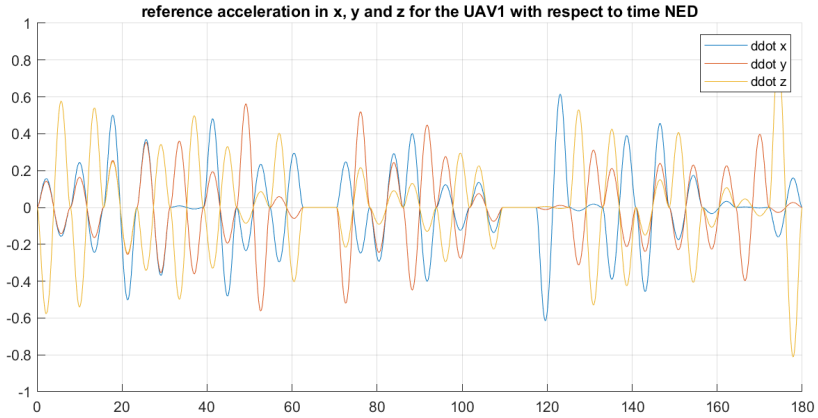

Figure 8: (test 1) Desired acceleration with respect to time for the UAV1

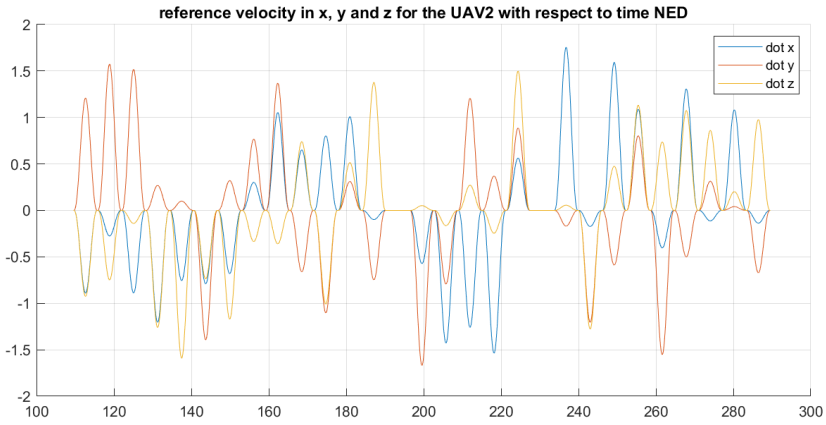

Figure 9: (test 1) Desired velocity with respect to time for the UAV2

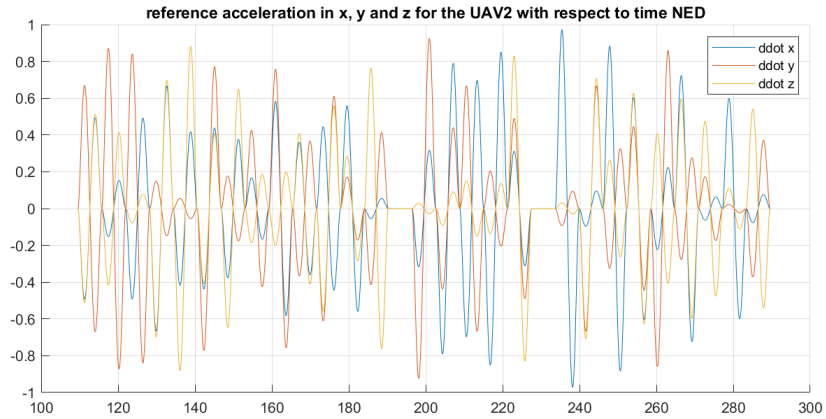

Figure 10: (test 1) Desired acceleration with respect to time for the UAV2

We can try to change `delta` and in order to see the results. We use delta=3, range_goal=3, check_line=5. The results can be seen in figures 11, 12, 13 and 14. It is possible to see that more points are needed to reach the same places because the value of the delta is lower than before.

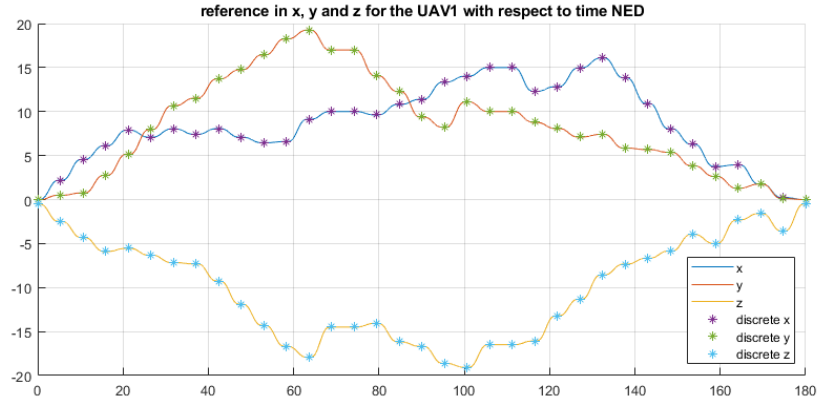

Figure 11: (test 2) Desired position with respect to time for the UAV1

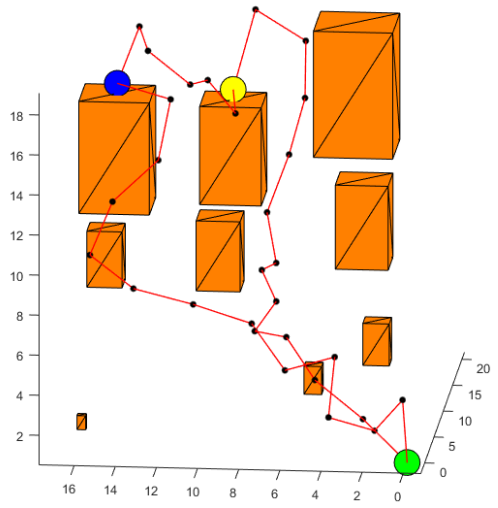

Figure 12: (test 2) The complete path in 3D for the UAV1. green is takeoff, yellow is take, blue is put

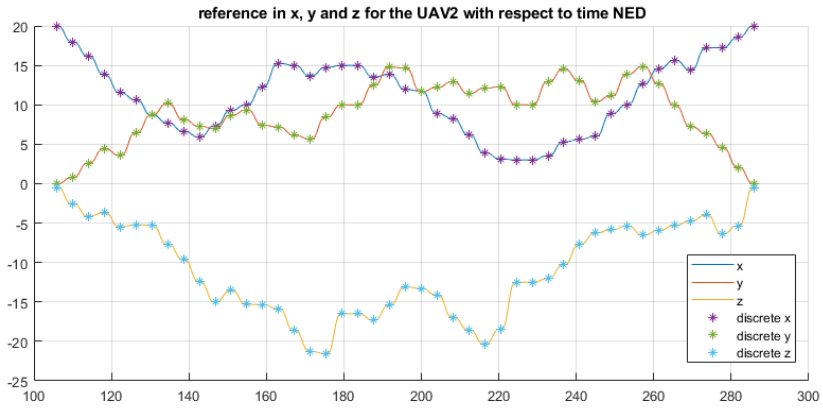

Figure 13: (test 2) Desired position with respect to time for the UAV1

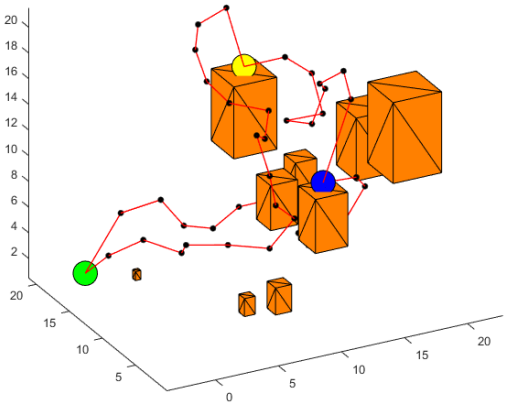

Figure 14: (test 2) The complete path in 3D for the UAV2. green is takeoff, yellow is take, blue is put

3 Simulink scheme

In figure 15 is shown the general Simulink scheme of the project. The scheme can be divided into various parts:

1. **Hierarchical control and Planners**: Hierarchical control is composed by 2 systems, one for the linear part and one for the angular part that is feedback linearizable. The output of the hierarchical control are used as inputs in the plants. The planners give reference values in NED configuration.
2. **Mass switches and mass sums**: There are two mass switches for when the UAVs get close enough to the place where they take or they place the mass. There are also two sum blocks to add the estimated mass of the object to the standard mass of the UAV to obtain a total estimated mass that will be used in the hierarchical control.
3. **UAV**: Here it is implemented the RPY dynamic model of the quadcopter
4. **Estimators**: The output of the the estimators estimates the mass of the object when it is taken. The estimator can be of first or third order.
5. **3D animation**: All the necessary blocks to see a 3D animation of the UAVs moving in the environment with the obstacles.
6. **Plots**: Here it is possible to see the plots of of different signals.

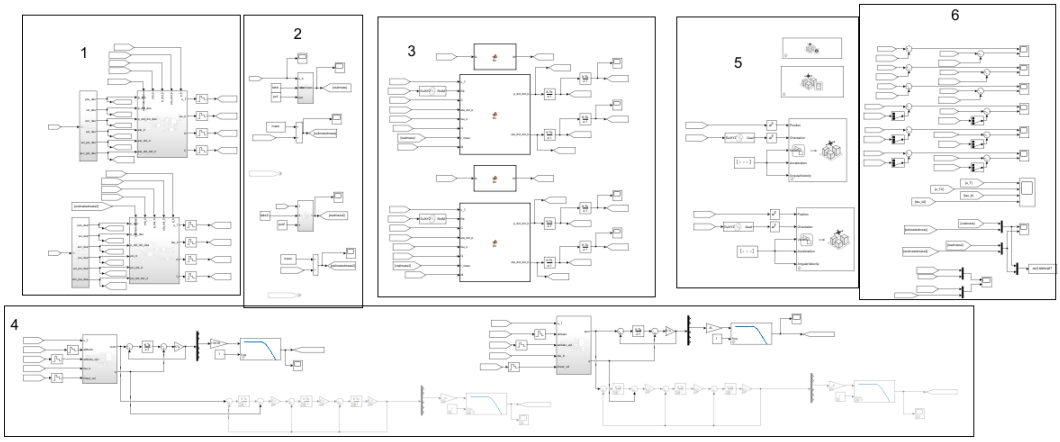

Figure 15: Simulink Scheme

3.1 Hierarchical control

The hierarchical controller is based on the RPY dynamic model of the quadcopter.

$$\begin{cases} m\ddot{p}_b = mge_3 - u_T R_b e_3 \\ M(\eta_b)\ddot{\eta}_b = -C(\eta_b, \dot{\eta}_b)\dot{\eta}_b + Q^T(\eta_b)\tau^b \end{cases}$$

This drone has 4 coplanar propellers and 6 degrees of freedom: position and attitude.In this case the world frame and body frame are expressed in NED configuration. Hierarchical control is composed by 2 systems, one for the linear part and one for the angular part that is feedback linearizable.

The input is composed by the references in position and the angular references for ψ and their velocities and accelerations. The position, speed and acceleration are given by `PATH_TL` and by the appropriate derivatives `PATH_TL_dot` and `PATH_TL_dot_dot`, developed in the MATLAB code. Those are the position, the velocity and the acceleration for the UAV1, but we have also those for the UAV2, `PATH_TL2`, `PATH_TL2_dot` and `PATH_TL2_dot_dot`. The angular reference in position is given by a sinusoidal signal of amplitude 0.4 and frequency $\frac{1}{2\pi}$. The angular velocity and the angular acceleration are

obtained imposing the equation of the derivative of the angular reference signal. When the drones are not in flight, the angular references are set to 0. This is possible through a switch that commutates when the drone is close enough to the position `takeoff` (or `takeoff2`). The planned subsystem can be seen in figure 16.

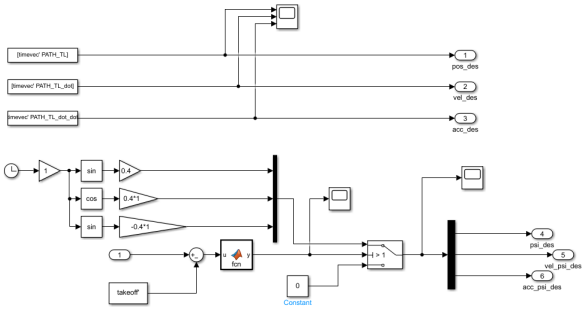

Figure 16: Input of hierarchical control

The structure of the hierarchical control scheme is shown in figure 17.

Here we have a brief description of the various blocks of the hierarchical control scheme:

- **The outer loop** takes in input the desired and the real position, speed and acceleration and gives in output the total thrust u_T and reference for the roll and the pitch angles(θ_d and φ_d). The outer loop is composed by 2 blocks: The position controller and Thrust and attitude. In these blocks are computed the following equations:
  - **Position controller:** $\mu_d = -K_p \begin{bmatrix} e_p \\ \dot{e}_p \end{bmatrix} + \ddot{p}_{b,d}$
  - **Thrust and attitude:** $u_T = m\sqrt{\mu_x^2 + \mu_y^2 + (\mu_z - g)^2}$;
    $\varphi_d = \arcsin(\frac{m}{u_T}(\mu_y \cos\psi_d - \mu_x \sin\psi_d))$;
    $\theta_d = \arctan(\frac{\mu_x \cos\psi_d + \mu_y sin\psi_d}{\mu_z - g})$
- **The inner loop** takes in input the roll and pitch angles and gives in output the torques τ^b. This block is composed by: Filter and derivatives; Angular PID, Feedback Linearization.In these blocks are computed the following equations:
  - **Angular controller:** $\tilde{\tau} = -K_e \begin{bmatrix} e_\eta \\ \dot{e}_\eta \end{bmatrix} + \ddot{\eta}_{b,d}$
  - **Feedback Linearization:** $\tau^b = I_b Q(\eta_b)\tilde{\tau} + Q(\eta_b)^{-T} C(\eta_b, \dot{\eta}_b)\dot{\eta}_b$
  - **Filtering and derivatives:** $\eta_{b,d} = \begin{bmatrix} \psi_d \\ \theta_d \\ \phi_d \end{bmatrix}$; $\dot{\eta}_{b,d} = \begin{bmatrix} \dot{\psi}_d \\ \dot{\theta}_d \\ \dot{\phi}_d \end{bmatrix}$; $\ddot{\eta}_{b,d} = \begin{bmatrix} \ddot{\psi}_d \\ \ddot{\theta}_d \\ \ddot{\phi}_d \end{bmatrix}$

Having the total thrust and τ^b, it is then possible to compute the propellers' velocities. To compute $\eta_{b,d}$, $\dot{\eta}_{b,d}$, $\ddot{\eta}_{b,d}$ has been implemented a second order low pass digital filter with $w_n = 800$ and $\zeta = 0.7$.

3.2 Change of the mass

The mass changes when the UAV gets close enough to some particular points. When the UAV gets close to the `take` point, its mass is instantaneously increased of a certain Δ_m that is the mass of the object taken by the UAV. When the UAV gets close to its `put` point, its mass is instantaneously decreased of the same value Δ_m. The Simulink scheme, which can be seen in figure 18, is based on switches, relays and a MATLAB function that calculates the norm of the input. It is possible to see that the inputs are the position of the drone, the position of the `take` point and the position of the `put` point.

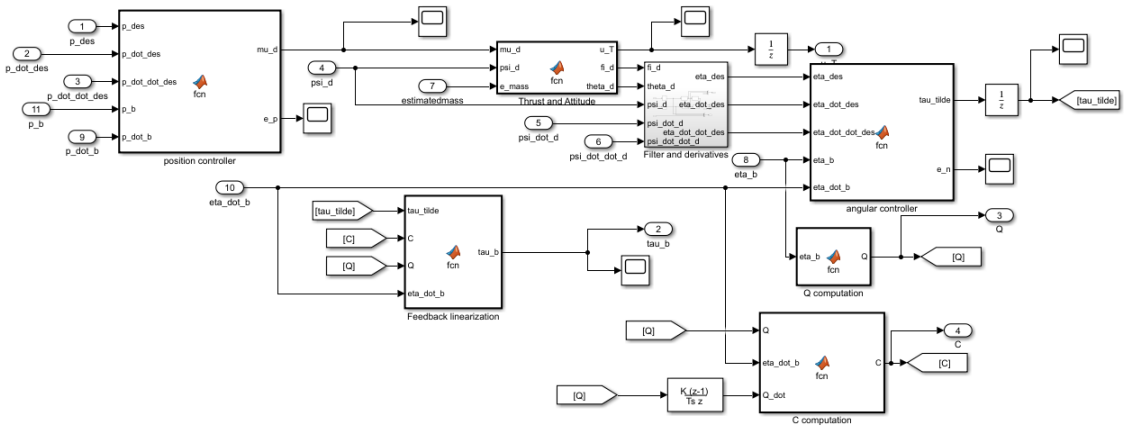

Figure 17: Hierarchical controller

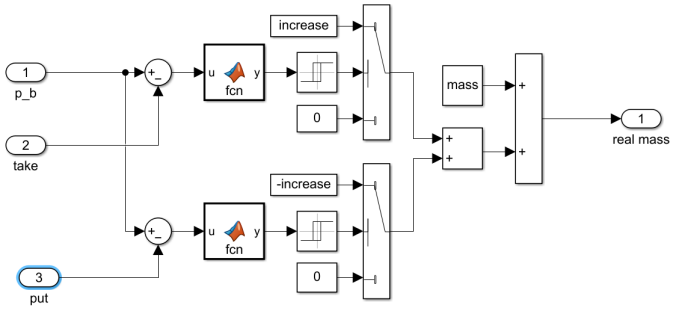

Figure 18: Mass switch scheme

3.3 Changing mass estimation

An estimator was implemented to estimate the mass of the taken object and so the total mass of the drone and the object.

The equations that regulate the estimator are:

$$\gamma_1(t) = K_1 \left(q - \int_0^t \begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix} + \begin{bmatrix} mge_3 + u_T R_b e_3 \\ C^T(\eta_b, \dot{\eta}_b) + Q^T(\eta_b)\tau^b \end{bmatrix} dt \right)$$

$$\gamma_i(t) = K_i \int_0^t - \begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix} + \gamma_{i-1} dt \qquad i = 2, ..., r$$

$$Q = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & c_\theta s_\phi \\ 0 & -s_\phi & c_\theta c_\phi \end{bmatrix}$$

$$M(\eta_b) = Q^T(\eta_b) I_b Q(\eta_b)$$

$$C(\eta_b, \dot{\eta}_b) = Q^T(\eta_b) S(Q(\eta_b)\dot{\eta}_b) I_b Q(\eta_b) + Q^T(\eta_b) I_b \dot{Q}(\eta_b)$$

$$R_b(\eta_b) = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & c_\phi c_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - c_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

$$q = \begin{bmatrix} mI_3 & O_3 \\ O_3 & M(\eta_b) \end{bmatrix} \begin{bmatrix} \dot{p}_b \\ \dot{\eta}_b \end{bmatrix}$$

The estimated mass gain $\Delta_{m,estimated}$ is equal to the ratio between the disturbance component along the z axis divided by the gravitational acceleration. The gains have been chosen in order to respect the equations that govern the estimator, but also to minimize the error. To do this, Itae method has been implemented which minimizes the quantity $\int_0^\infty t|e(t)|dt$ through the choice of the characteristic

polynomial.

```
wn1=15;%wn are chosen to guarantee a bandwidth almost equal to 15 for G
wn3=13;%of the 2 estimators of first and third order.
G1 = wn1/(s+wn1);
bandwidth(G1)
k11=wn1;
G3 = wn3^3/(s^3+1.75*wn3*s^2+2.15*wn3^2*s+wn3^3);
bandwidth(G3)
k33=1.75*wn3;
k23=2.15*wn3^2/k33;
k13=wn3^3/k33/k23;
```

Those frequencies are been selected with a trial and error criterion. The best performance of the estimators where given by those values of frequency.
A low pass filter was implemented to eliminate high frequency noise. The total estimates mass is used like input of the hierarchical control. The choice of the threshold value for the low pass filter is a compromise between the quantity of noise that we can accept in our mass estimation and the speed of convergence for our estimation. A very low frequency threshold for the filter means a very low level of noise inside the estimation, but also means that the output of the filter will converge to the final value slowly. After that, we only need to add the estimated $\Delta_{m,estimated}$ to the standard mass of the UAV and this will be the estimated mass value that we will use in the control, while the plant will use the real mass that instantaneously switches between the standard mass and the standard mass + mass of the object.
The Ts of the digital components is 0.05 seconds.

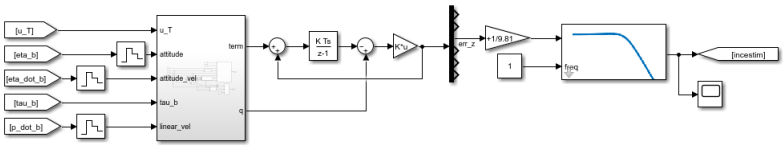

Figure 19: First order estimator

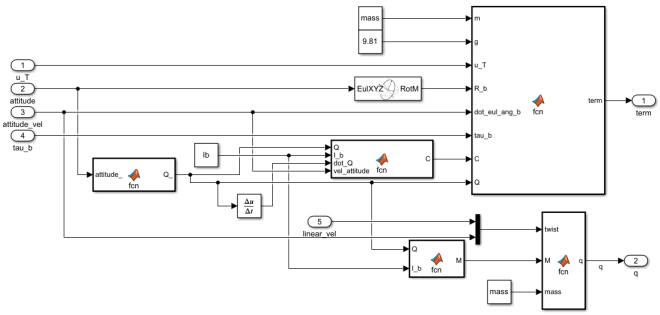

Figure 20: subsystem of the estimator

4 Results

It could be interesting now to see some simulations of the scheme.

4.1 test 1: an overview

For the planning of the trajectory, we generated a trajectory with

- `delta`=5;
- `range_goal`=3;
- `check_line`=5;
- time of execution of each trajectory of 180 seconds.

In the Simulink scheme, the parameters used are

- Kp = [5 0 0 2 0 0;0 4 0 0 3 0;0 0 3 0 0 4] in position PID;
- Ke = [5 0 0 2 0 0;0 4 0 0 3 0;0 0 3 0 0 4] in angular PID;
- mass = 0.3 kg standard mass of the UAV;
- increase=0.1 kg mass of the object;
- Ib = diag([0.2416 0.2416 2*0.2416]) Nm inertia tensor of the UAV;
- first order estimator with K11=15;
- cutoff frequency for the filter = 1 rad/s.

The results can be seen in figures 21, 22, 23, 24, 25, 26, 27, 28 and 29.

4.2 test 2: change the order of the estimator

The first change that we can made is the order of the estimator:

- third order estimator with: $wn_3 = 13; k33 = 1.75{*}wn_3; k23 = 2.15{*}wn_3^2/k33; k13 = wn_3^3/k33/k23$. The gains are chosen with the ITAE method.

We can see the result in figure 30.

4.3 test 3: changing the mass of the object

It may be useful to see how different masses of the object that the UAVs have to take can change their behavior. Up to now, we used an object of 0.1 kg, 33% of the mass of the UAV. Now we are going to see what an improvement of the mass can make:

- increase=0.15 kg mass of the object (50% of the mass of the UAV);
- the other configuration variables are the same of test 1.

The results can be seen in figures 31, 32, 33, 34, 35, 36, 37, 38 and 39. The behavior suggests that an higher mass of the object increases the error peaks in the instants when those objects are taken or released. The estimator continues to work properly and the systems reaches the various points of the path without going in instability.

4.4 test 4: changing the time of execution

Now we try to see what happens if we ask our UAVs to track a faster trajectory.

- time of execution of each trajectory of 120 seconds.
- the other configuration variables are the same of test 1.

The results can be seen in figures 40, 41, 42, 43, 44 and 45. The error components in x and y are much greater than in test 1. We can also see that the mass estimation has become much noisier than in test 1.

4.5 test 5: without estimator

Now we try to see what happens if we neglect the estimator.

- no object mass estimation.

The results can be seen in figure 46. We only need to see what happens to the position error to understand the problem: There is a steady error in the z due to the fact that the object is always heavier than what the controller knows. There is no compensation. There is also another problem. Due to this constant error, the UAVs never get close enough to the point where they shoud place their mass, so they have the mass attached until the end. The second drone starts the flight, only because the 2 drones trajectories are synchronized, but in reality the control is not working satisfactorily.

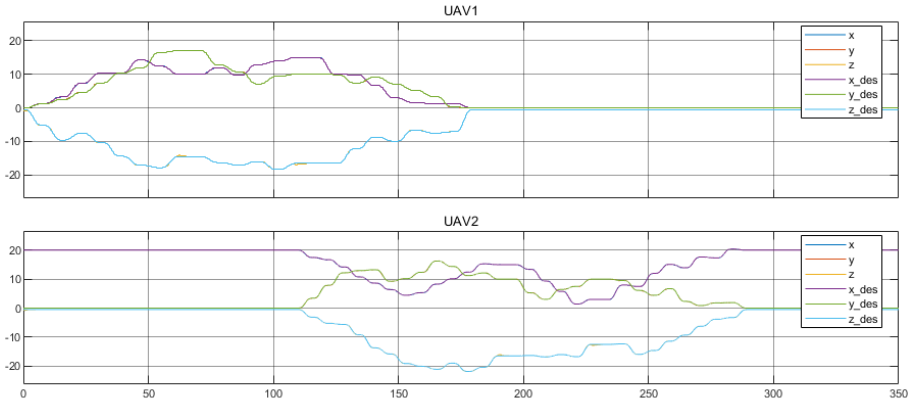

Figure 21: (test 1) Real position VS desired position over time for the UAV1 (top) and UAV2 (down).

Looking closely, we can see two small peaks in both plots in the moments when they take or put the mass.
Below are shown the errors in position (22), speed (23) and acceleration (24), there are spikes when the drones take and put the object. At the beginning there is a peak on the third component of the acceleration in both the graphs equal to about 30, it was preferred to zoom in to better show the acceleration trend during the simulation. Those values of position errors are acceptable. Remember that the take/put points are 0.5m higher than the actual obstacles where the mass is placed on. The error peaks are around 0.4 m. Moreover during the RRT execution we used a collision check based on a sphere on not on the check on a given point, so we have a security range around our desired trajectory where the UAV can move safely.

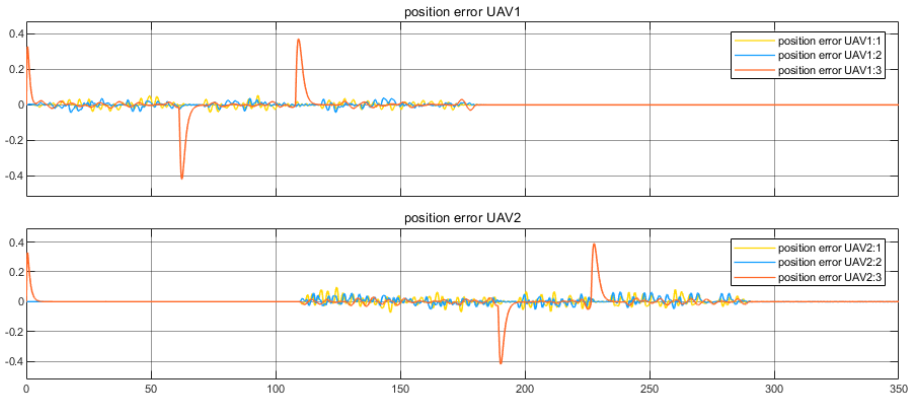

Figure 22: (test 1) Position error over time for the two UAVs.

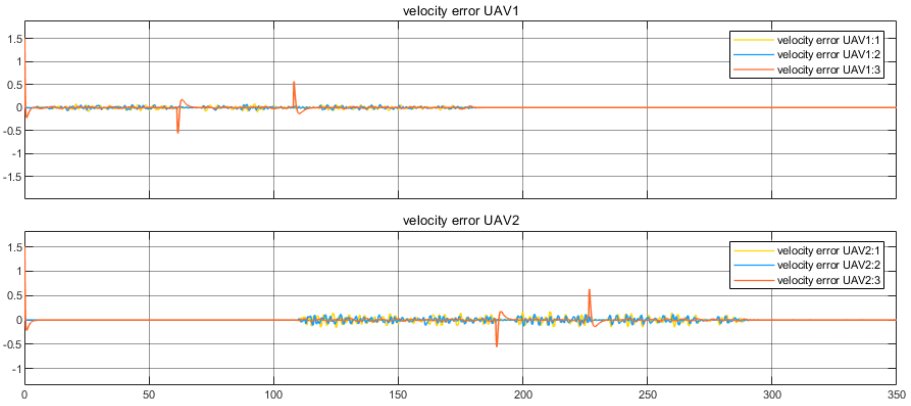

Figure 23: (test 1) Velocity error over time for the two UAVs.

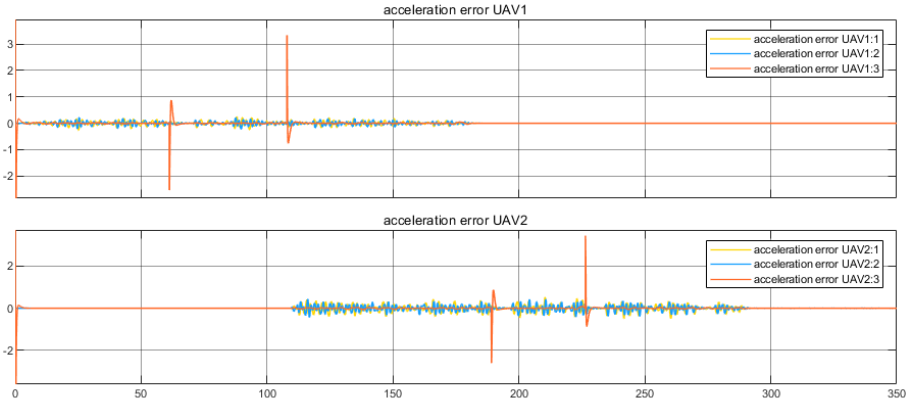

Figure 24: (test 1) Acceleration error over time for the two UAVs.

Below are shown the angular errors in position (25), speed (26) and acceleration (27) of the 2 UAV, there are spikes at the beginning and at the end of the trajectory.

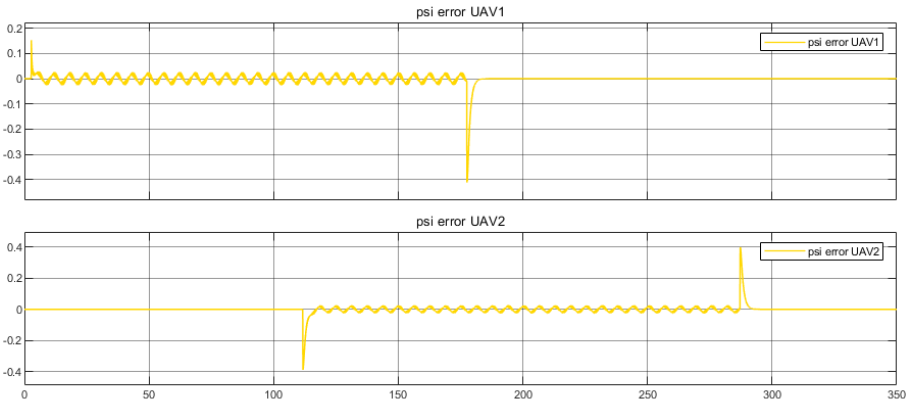

Figure 25: (test 1) Psi angle error over time for the two UAVs.

The error oscillates very little respect to the real desired oscillation on the psi angle.

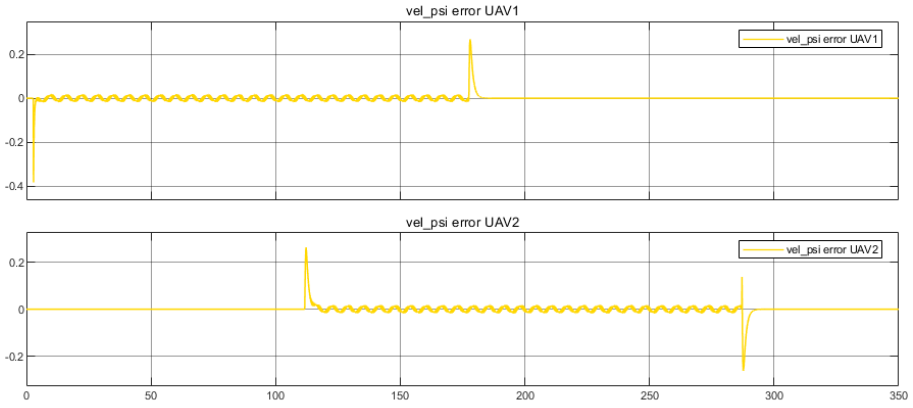

Figure 26: (test 1) Psi angular velocity error over time for the two UAVs.

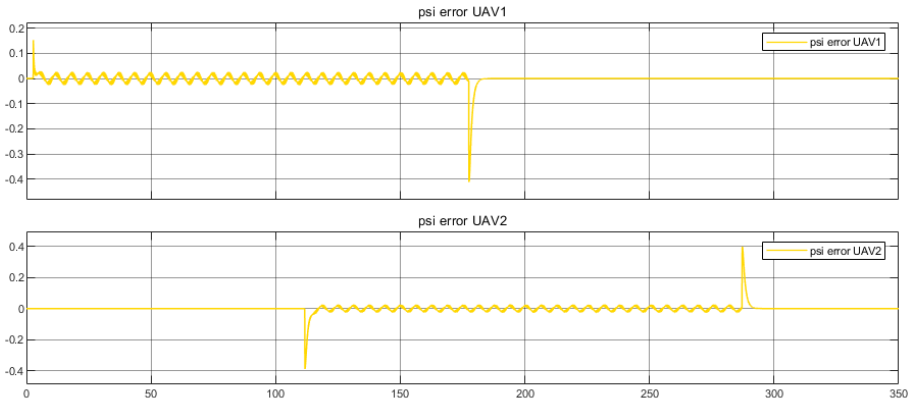

Figure 27: (test 1) Psi angular acceleration error over time for the two UAVs.
Below is shown the comparison between the real and estimated mass .

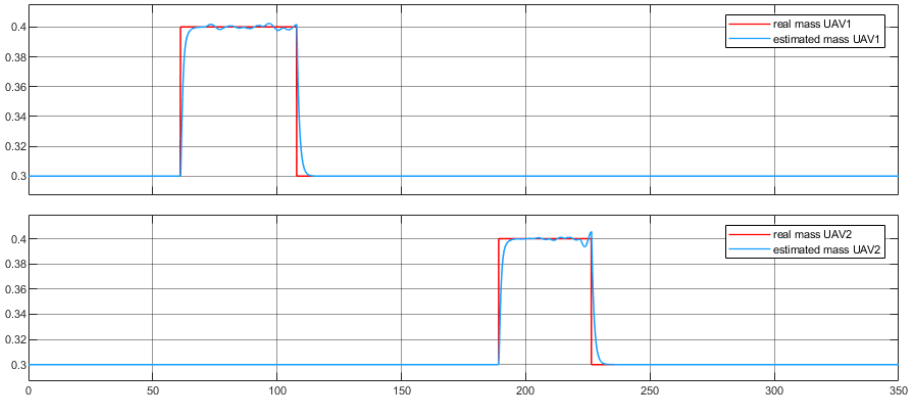

Figure 28: (test 1) Real mass vs estimated mass for the two UAVs.

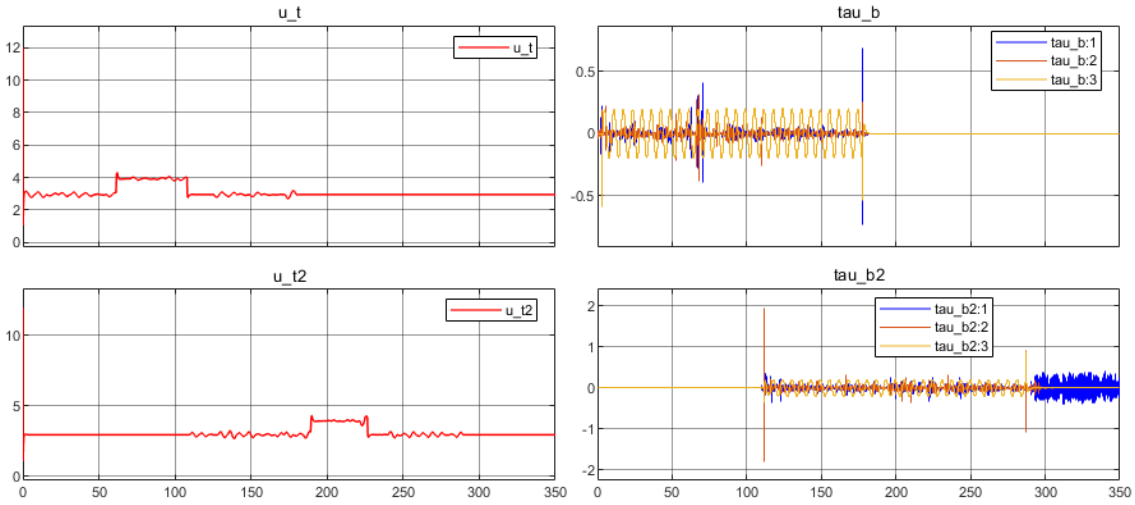

Figure 29: (test 1) the values of u_T and tau_b over time.
We can see the effect of the added mass on u_T and u_T2

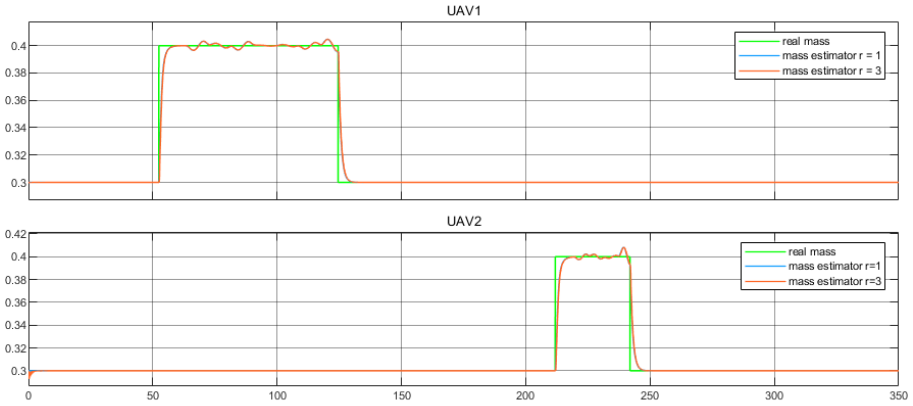

Figure 30: (test 2) Real mass vs estimated mass for the two UAVs with the estimators of third order.

There isn't almost any improvement, and this also means that there isn't an noticeable improvement in the general behavior of the system. The results of the two estimators are perfectly overlapped.

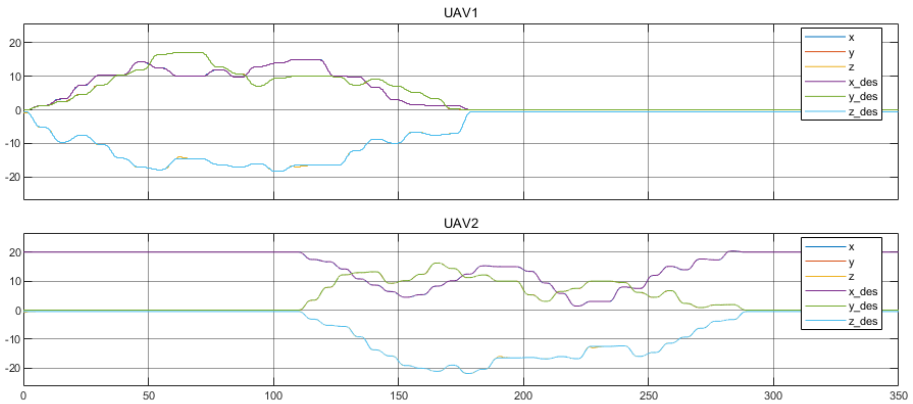

Figure 31: (test 3) Real position VS desired position over time for the UAV1 (top) and UAV2 (down).

Looking closely, it is possible to see two small peaks in both plots in the moments when they take or put the mass. Below are shown the errors in position (32), speed (33) and acceleration (34), there are spikes when the drones take and put the object. At the beginning there is a peak on the third component of the acceleration in both the graphs equal to about 30, it was preferred to zoom in to better show the acceleration trend during the simulation.

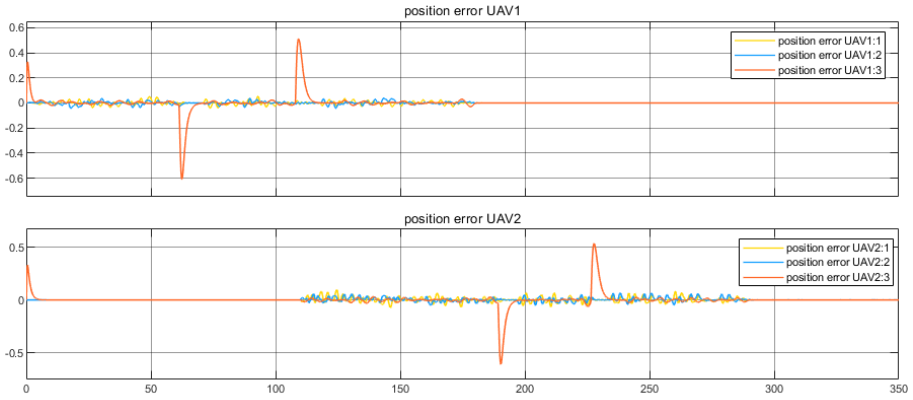

Figure 32: (test 3) Position error over time for the two UAVs.

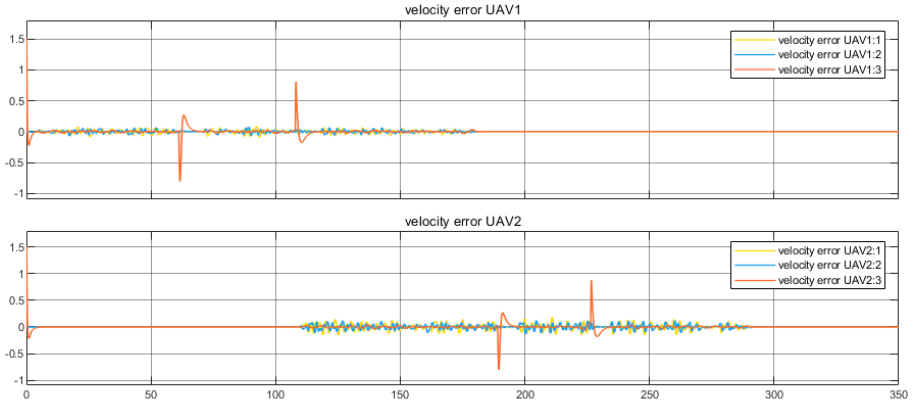

Figure 33: (test 3) Velocity error over time for the two UAVs.

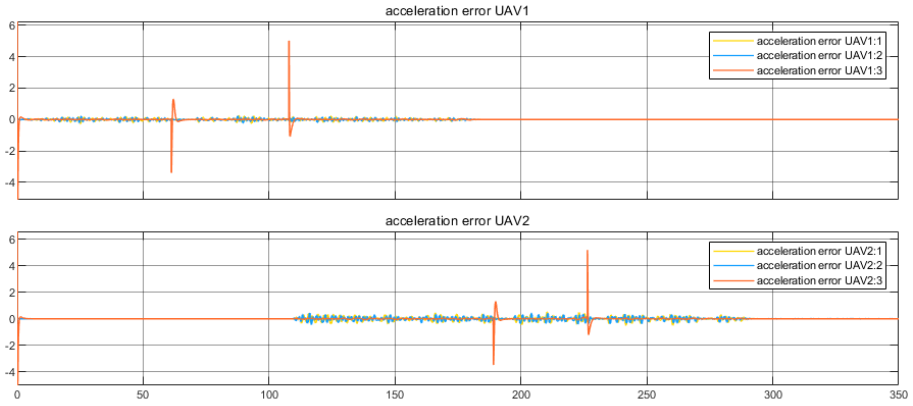

Figure 34: (test 3) Acceleration error over time for the two UAVs.

Below are shown the angular errors in position (35) ,speed (36) and acceleration (37) of the 2 UAV, there are spikes at the beginning and at the end of the trajectory.

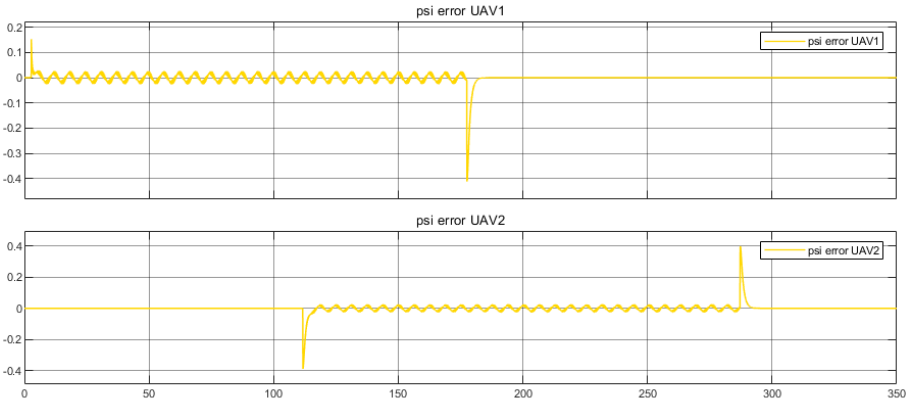

Figure 35: (test 3) Psi angle error over time for the two UAVs.

The error oscillates very little respect to the real desired oscillation on the psi angle.

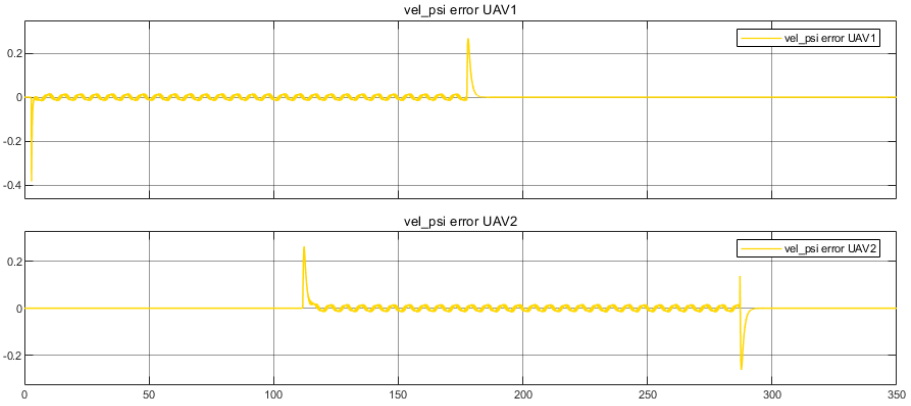

Figure 36: (test 3) Psi angular velocity error over time for the two UAVs.

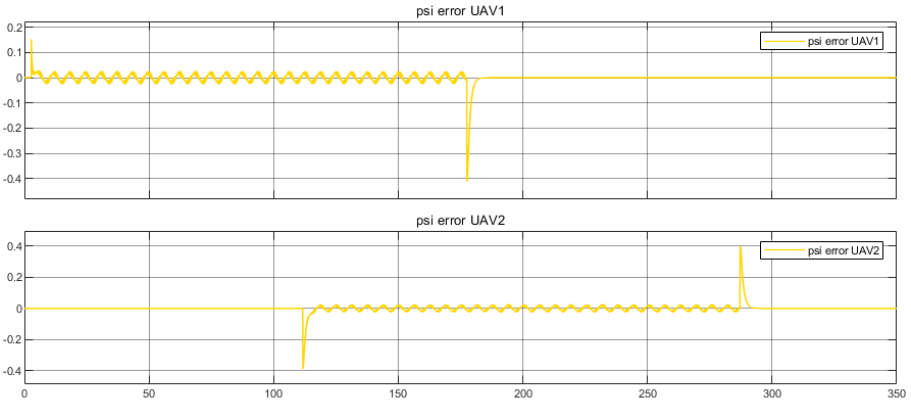

Figure 37: (test 3) Psi angular acceleration error over time for the two UAVs.

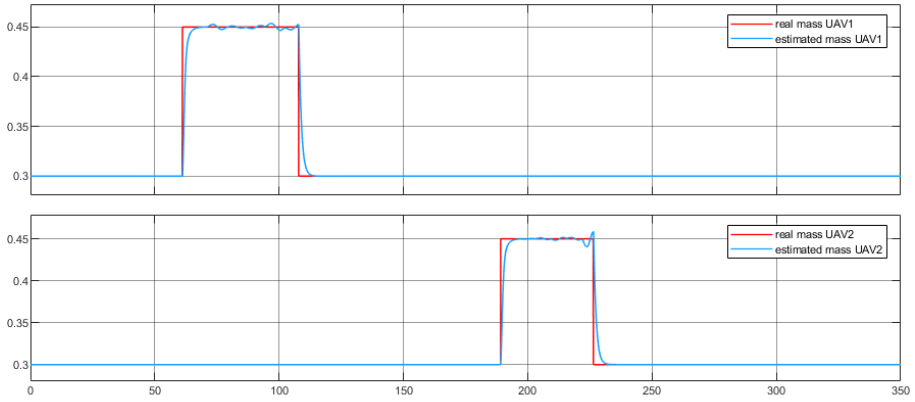

Figure 38: (test 3) Real mass vs estimated mass for the two UAVs.
The real mass changes instantaneously while the estimated mass need some seconds to well track the real mass

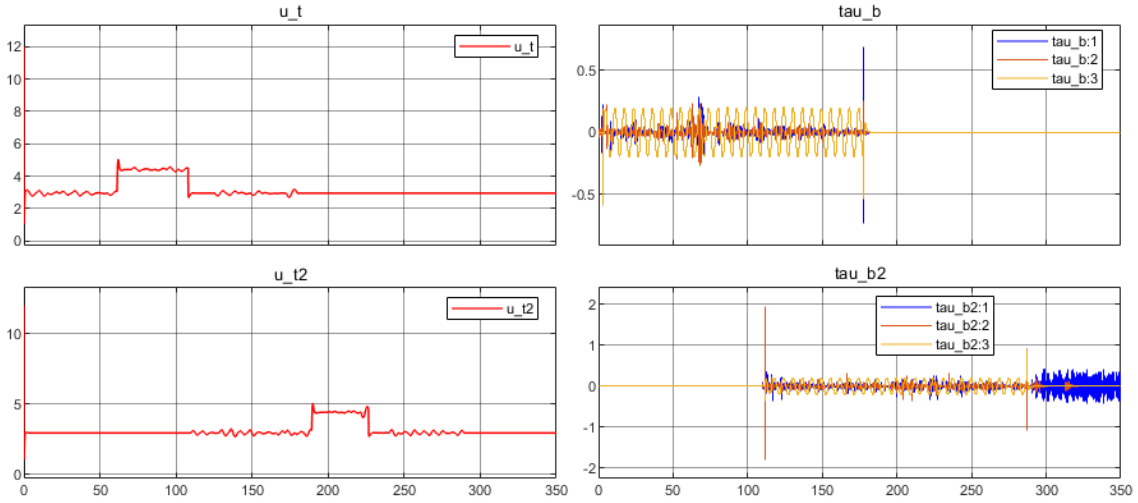

Figure 39: (test 3) the values of u_T and tau_b over time.

We can see a larger step in u_T and u_T2 when UAV1 or UAV2 takes the mass. That's because the mass of the object is bigger now than in test 1

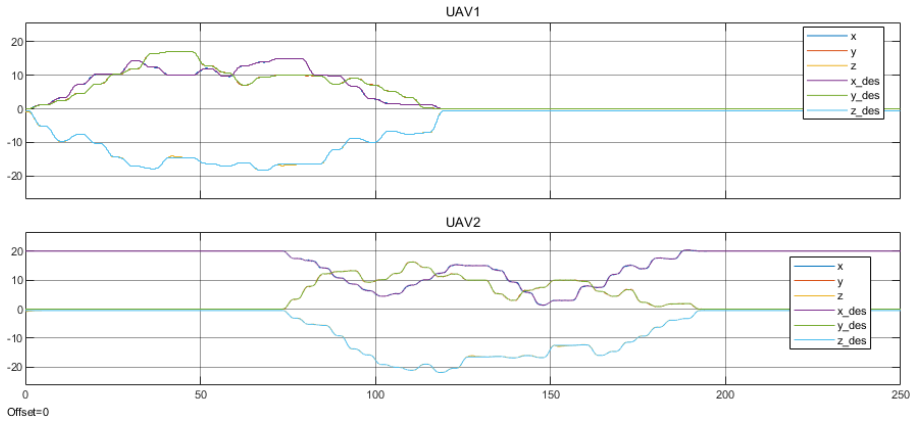

Figure 40: (test 4) Real position VS desired position over time for the UAV1 (top) and UAV2 (down).

Below are shown the errors in position (41) , speed (42) and acceleration (43), there are spikes when the drones take and put the object.

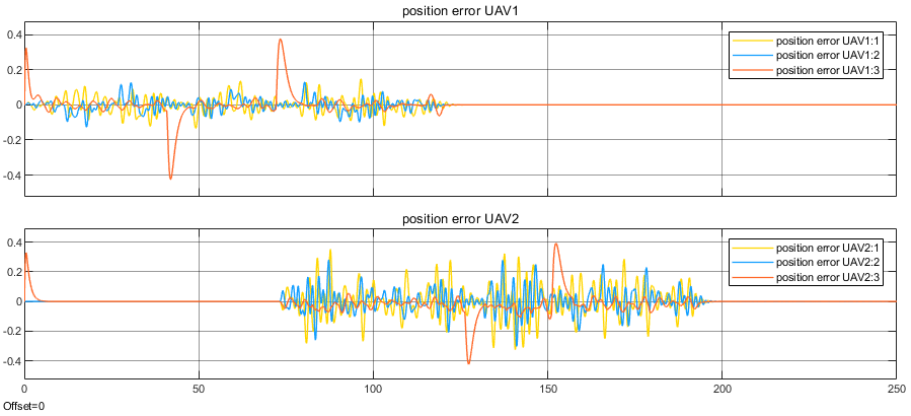

Figure 41: (test 4) Position error over time for the two UAVs.

We can see the peaks in the moments when they take and put the object aren't much greater than in test 1 but the x and y error components are greater. At the beginning there is a peak on the third component of the acceleration in both the graphs equal to about 30, it was preferred to zoom in to better show the acceleration trend during the simulation.

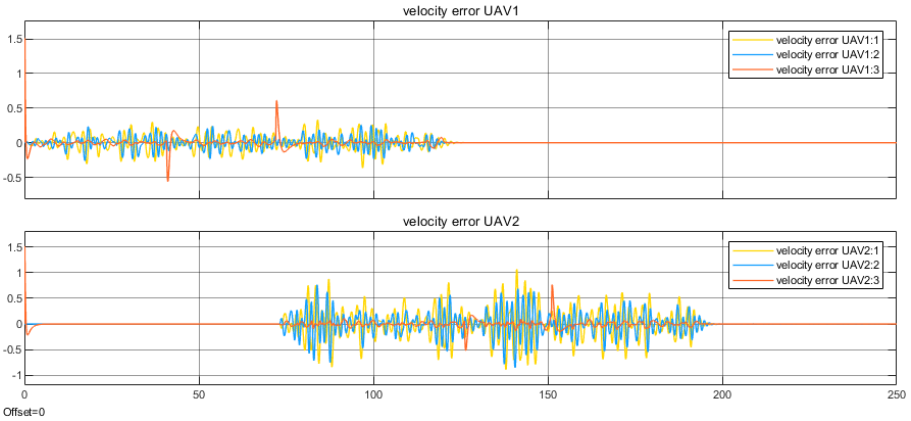

Figure 42: (test 4) Velocity error over time for the two UAVs.

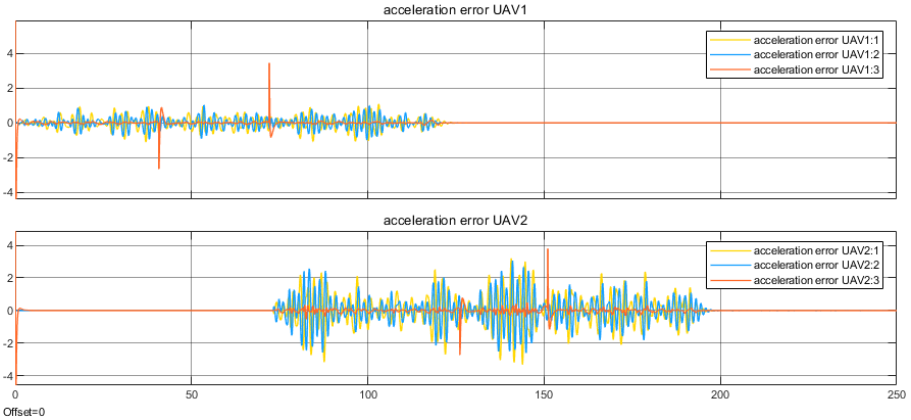

Figure 43: (test 4) Acceleration error over time for the two UAVs.

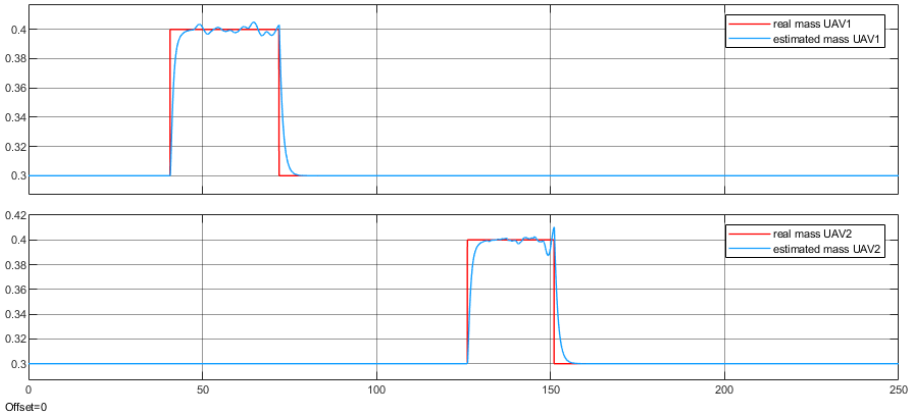

Figure 44: (test 4) Real mass vs estimated mass for the two UAVs.
The real mass changes instantaneously while the estimated mass need some seconds to well track the real mass. The faster trajectory shows its effects with a much noisier estimation of the mass

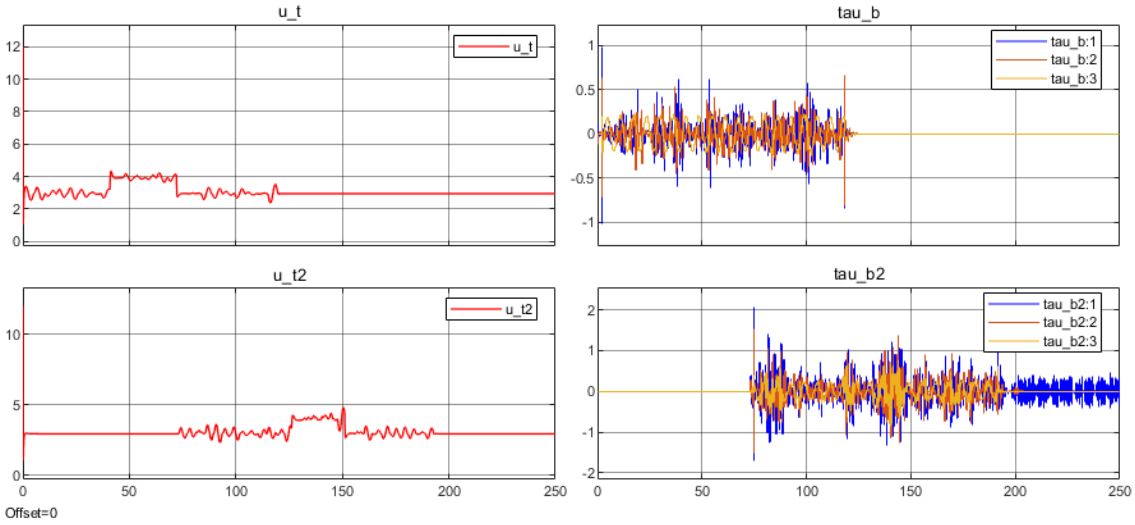

Figure 45: (test 4) the values of u_T and tau_b over time.

We can see that we have higher values in both control inputs for both the UAVs. That's because we are asking for the same task but in a shorter time, so the UAVs need to achieve higher values of velocity and acceleration

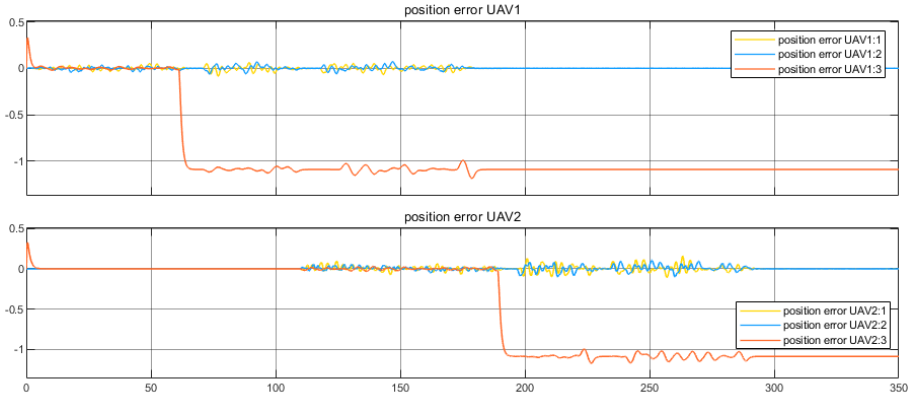

Figure 46: (test 5) position error over time.

The error in z is almost constant and doesn't go to zero.

5 Conclusions

The work presented concerns the design of two drones able to pick up objects and take them to a new location through a hierarchical control. For each of the two drones, RRT algorithm has been implemented to avoid obstacles of the environment and to reach the desired destinations. The mass of the objects to be picked was not known and it was estimated using an estimator in run time. The innovations presented in this project with respect to the topics presented during the homework were:

- RRT in 3D;
- creation of a multi-path task;
- creation of the working environment with obstacles;
- implementation of the Hierarchical control;
- synchronization of the work of the two drones;
- real and estimated mass changes in run time updated in the controller;
- the mass change is based on distance.

The work presented has room for improvements:

- Use of an online search algorithm, so as to verify that the 2 drones do not collide, this would have allowed the implementation of moving obstacles;
- Use of an algorithm that guarantees a shorter and smoother path;
- More elaborate 3D animation using other software.

5.1 contributions from members

- 3D RRT: Marco
- Polynomial time law: Marco
- Construction of the planner values: Marco
- 3D scenario: Michael
- Distance based mass switch: Marco
- Mass estimators: Michael
- Hierarchical control: Michael
- UAV plant: Michael
- Graphics and plots: Michael
- Documentation: Together
- Simulations: Together