Università degli studi di Napoli Federico II

FACOLTA' DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria dell'Automazione e Robotica

ROBOTICS LAB & FIELD AND SERVICE ROBOTICS

Simulation and control of a quadrotor in an unknown environment

Studente:
Francesco Lella
P38/15

Docenti:
Prof. Fabio Ruggiero
Prof. Jonathan Cacace

Index

Introduction

The goal of this project is to simulate the behaviour of a quadcopter inside an unknown environment using ROS and Gazebo.
Initially, the general context will be analyzed by evaluating the model of the drone used and the different sensors, the environment and the purpose of the simulation. Subsequently, the choices made for the control and for the planning phase will be indicated following the notions learned during the FSR (Field and Service Robotics) lessons implemented through the use of ROS, learned in the RL (Robotics Lab) lessons.

1. Context

1.1 Drone model

The drone chosen for this project is called Iris, taken from the RotorS library. The drone is a quadcopter with "X" configuration with the following parameters:

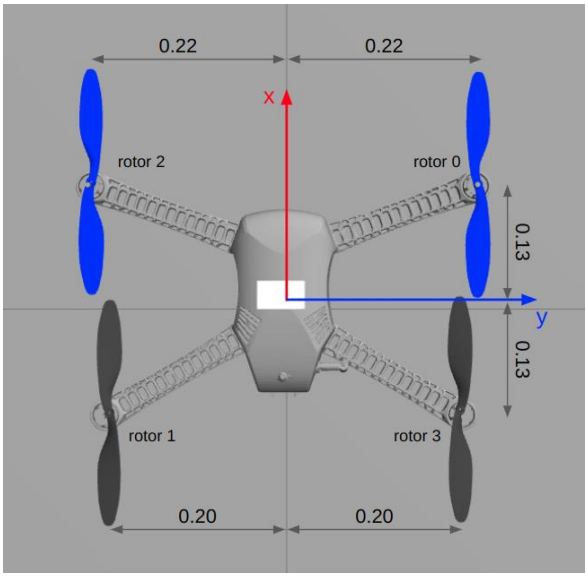


Fig. 1.1 Drone model

Parameter	Description	Value	Unit
acc_g	Gravity	9.81	m/s^2
$mass$	Mass	1.515	kg
I_{xx}	Roll Inertia	0.02941625	$kg \cdot m^2$
I_{yy}	Pitch Inertia	0.02941625	$kg \cdot m^2$
I_{zz}	Yaw Inertia	0.05577725	$kg \cdot m^2$
c_T	Thrust coefficient	$8.55 \cdot 10^{-6}$	$kg \cdot m$
c_a	Drag coefficient	$1.75 \cdot 10^{-6}$	$kg \cdot m^2$

The drone is equipped with the following sensors:

- Camera: it is used to read AR-markers. It has a resolution of 800x600 px, with a refresh rate of 30Hz and a field of view of about 80°

- LiDAR: it is used to estimate the distance to obstacles. It is able to detect obstacles within a radius of 180° with a resolution of 0.1m with ranges from 0.2m to 30m with 720 samples and a frequency of 40 Hz.

It was assumed that the problem of calculating the position could be solved by taking the data from Gazebo and for this reason no additional sensors were used for odometry.
As requested, a constant force of 1N is applied along the X-axis of the world's reference system and the mass of the drone underestimated by 10%. The control chosen is the "hierarchical control" and there is also an estimator of external forces, useful to make up for the incorrect estimate of the weight of the drone and the constant external force applied.

1.2 Environment

The environment is shown in the figure and was created for simulation and to show high-level and low-level control in different conditions. It was built by importing different models taken from Gazebo and then modified through *.model* files to modify the size and colour. There are two different types of obstacles, in addition to walls. There are the obstacles with cylindrical shape and others to simulate windows. The cylindrical obstacles are inserted with two different radii for the base circumference and the window obstacles are of two different heights. The window obstacles have an AR marker in the lower left corner as per the design request and are used to be identified by the drone that must cross it.
There are two platforms, one for take-off and one for landing. The goal of the simulation is to get from the take-off platform to the landing platform by passing through the corridors, avoiding obstacles, passing through the window ones and passing through some waypoints positioned at the end of the corridors delimited by walls.

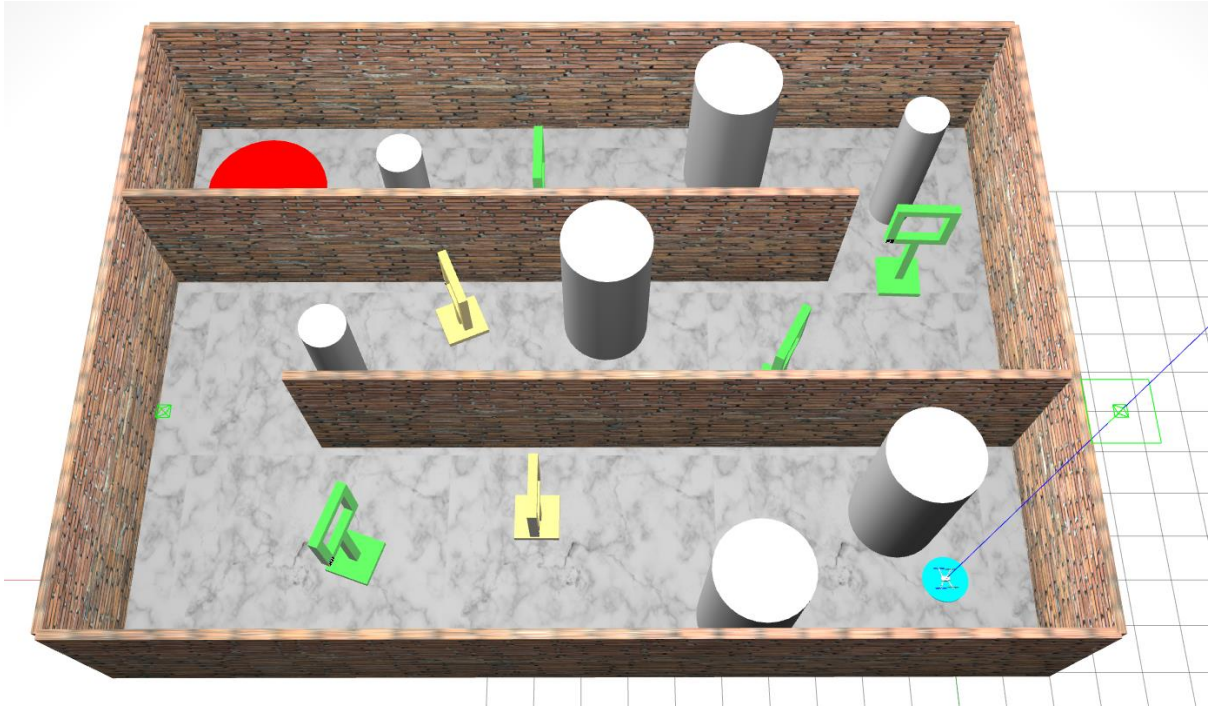


Fig. 1.2 Simulation environment

In the figures are showing the two types of obstacles:

- solid

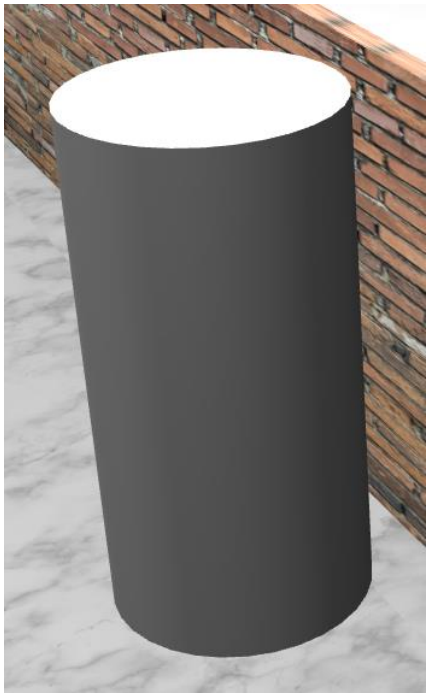


Fig 1.3 Solid obstacle

- empty (window)

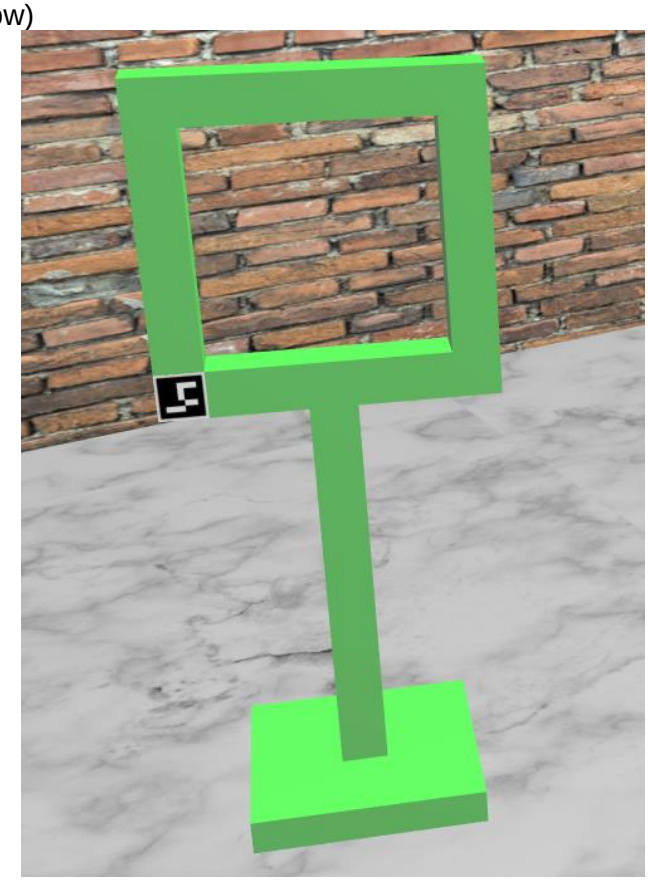


Fig. 1.4 Window obstacle

2. Control - FSR

This chapter will analyze the type of control chosen to command the drone, analyzing the problems that led to the insertion of an estimator for external forces, necessary because the design specifications require correct operation even if there is a constant force along the X axis and the mass of the drone is underestimated by 10%.

2.1 Hierarchical control

The control scheme is shown in the figure:

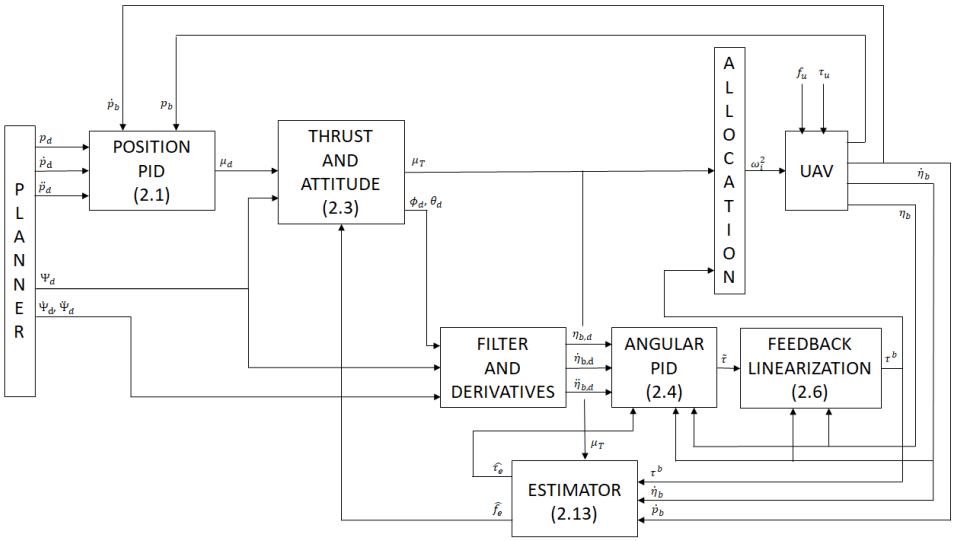


Fig. 2.1 Control scheme

It is possible to identify two cycles, one internal and one external. The external loop is a position control, which takes the desired position, speed and acceleration together with the yaw coming from the planner and calculates the virtual control input μ_d:

$$\mu_d = -K_p \begin{bmatrix} e_p \\ \dot{e} \end{bmatrix} + \ddot{p}_{b,d} - K_I \int_0^t e(\sigma) d\sigma - \frac{f_e}{m} = \begin{bmatrix} \mu_x \\ \mu_y \\ \mu_z \end{bmatrix} \tag{2.1}$$

Where e_p are the position errors, $\dot{e}$ velocity errors, K_p and K_I gain matrices and external forces are f_e.
The matrices k_P and K_I were chosen as:

$$K_p = diag(3,3,3,4,4,4), \qquad K_I = diag(0.2, 0.2, 0.2) \tag{2.2}$$

Right after that, it is possible to calculate the total thrust and the angle that you want to give to the drone (attitude).

$$\mu_t = m \sqrt{\mu_x^2 + \mu_y^2 + (\mu_z - g)^2} \tag{2.3}$$

$$\phi_d = \sin^{-1}\left(\frac{m}{\mu_t}\left(\mu_y \cos(\psi_d) - \mu_x \sin(\psi_d)\right)\right)$$
$$\theta_d = \tan^{-1}\left(\frac{\mu_x \cos(\psi_d) + \mu_y \sin(\psi_d)}{\mu_z - g}\right)$$

The angles calculated with the previous equations were filtered through a low-pass filter with a cut-off pulsation of 50rad/s. The angles were also derived and again filtered twice to get their speed and acceleration.
The desired torque was calculated as follows:

$$\tilde{\tau} = -K_e \begin{bmatrix} e_\eta \\ \dot{e}_\eta \end{bmatrix} + \ddot{\eta}_{b,d} - \tau_e \tag{2.4}$$

where orientation and angular velocity errors are e_η and $\dot{e}_\eta$, K_e is the gain matrix and τ_e the external torque.
The matrix was chosen as:

$$K_e = diag(6,6,8,3,3,3) \tag{2.5}$$

Using the feedback linearization:

$$\tau^b = I_b Q \tilde{\tau} + Q^{-T} C(\eta_b, \dot{\eta}_b)\dot{\eta}_b = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{2.6}$$

where the matrices Q and C are derived from the dynamic RPY model of the quadcopter and are shown below:

$$Q = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix} \tag{2.7}$$

$$C(\eta_b, \dot{\eta}_b) = Q^T S(Q\dot{\eta}_b) I_b Q + Q^T I_b \dot{Q} \tag{2.8}$$

The inputs for the UAV are the velocities of the propellers that can be calculated by multiplying the desired thrust and torque by the inverse of the allocation matrix. And right after, making the square root.

$$\begin{bmatrix} \omega_0^2 \\ \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \end{bmatrix} = G_q^{-1} \begin{bmatrix} \mu_t \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{2.9}$$

Where:

$$G_q^{-1} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ -l_{0y}c_T & l_{1y}c_T & l_{2y}c_T & -l_{3y}c_T \\ l_{0x}c_T & -l_{1x}c_T & l_{2x}c_T & -l_{3x}c_T \\ c_a & c_a & -c_a & -c_a \end{bmatrix} \tag{2.10}$$

2.2 Momentum-based estimator

The estimator is important to overcome the uncertainties present in the control and therefore estimate external forces and pairs that in this case are to be imposed as per the design request.

The estimator is designed to have a linear relationship in the Laplace domain between the estimated external forces and the real ones:

$$\widehat{F_e}(s) = G(s)F_e(s) \tag{2.11}$$

Six-element vector F_e refers to external linear forces and torques.

$G(s)$ is a 6x6 diagonal matrix of first-order transfer functions:

$$G_i(s) = \frac{k_0}{s + c_0} \tag{2.12}$$

With k_0 and $c_0 > 0$.
In particular, the values used are: $k_0 = c_0 = 2$.

By performing the Laplace antitransform and integrating with respect to time it is possible to obtain the following equation:

$$\widehat{F_e}(t) = K_0 q(t) - \int_0^t \left(K_0 C_\xi^T \dot{\xi} + K_0 \Lambda u + K_0 G_\xi + C_0 \widehat{F_e}(\sigma) \right) d\sigma \tag{2.13}$$

where:

$$\widehat{F_e} = \begin{bmatrix} \hat{f}_e \\ \hat{\tau}_e \end{bmatrix}, \qquad \xi = \begin{bmatrix} p_b \\ \eta_b \end{bmatrix},$$

$$q = M_\xi \dot{\xi}, \qquad M_\xi = \begin{bmatrix} mI_3 & 0_3 \\ 0_3 & M \end{bmatrix}$$

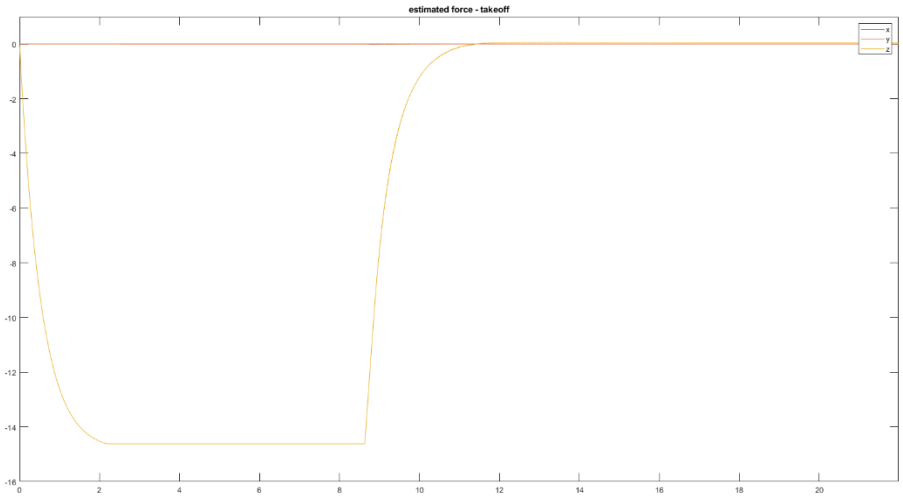
$$C_\xi = \begin{bmatrix} 0_3 & 0_3 \\ 0_3 & C \end{bmatrix}, \qquad \Lambda = \begin{bmatrix} -R_b i_3 & 0_3 \\ 0_{3x1} & Q^T \end{bmatrix} \tag{2.14}$$

$$G_\xi = \begin{bmatrix} -mgi_3 \\ 0_{3x1} \end{bmatrix}$$

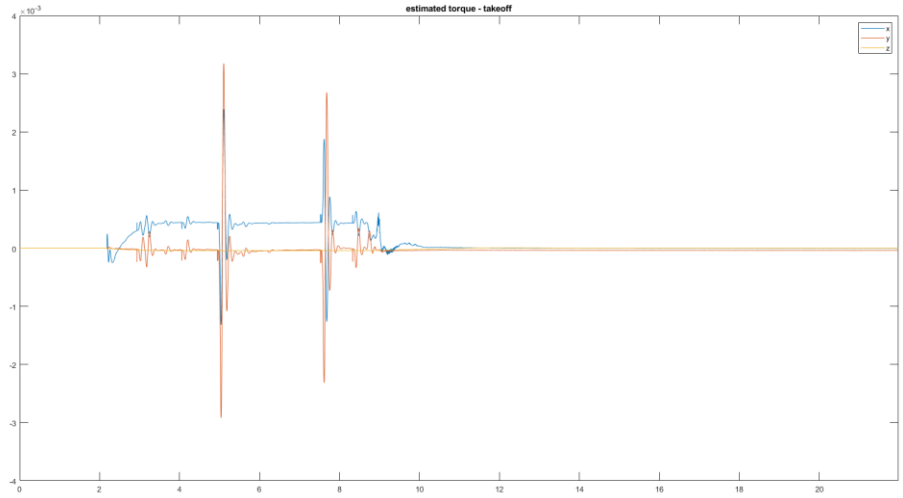
It is necessary that $q(0) = \widehat{F_e}(0) = 0$. The estimator must start before take-off.

2.3 Test

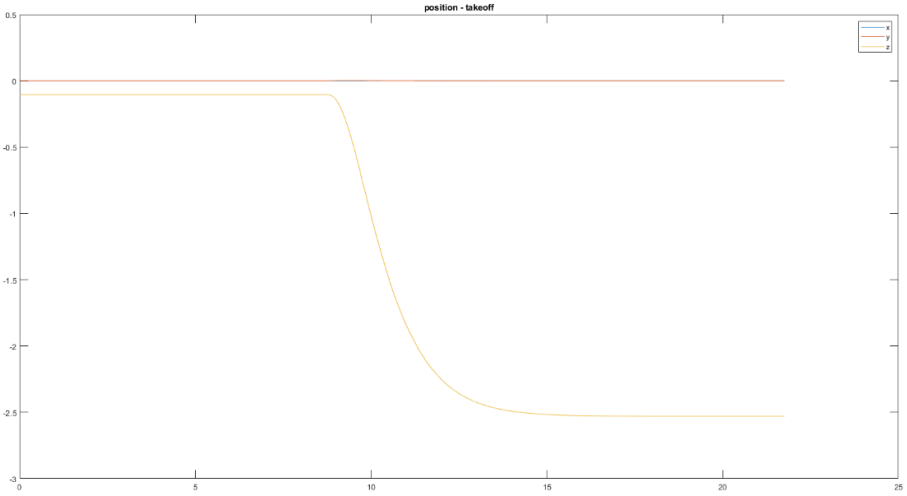
The following figures represent the behavior of the drone following some stresses, in particular the take-off phase is simulated and next it is perturbed by applying forces along the three axes of different intensity to verify that the control manages to bring the drone back to the position where it was before the perturbation in addition. After this, the drone is perturbed with instantaneous rotations around the axes. There are also graphs showing the external forces that are estimated, i.e. the forces that cause the perturbations.
These graphs were obtained by recording the different variables using ROSbag and then converted the files. ".bag" in ".csv" files and then plotted using Matlab.



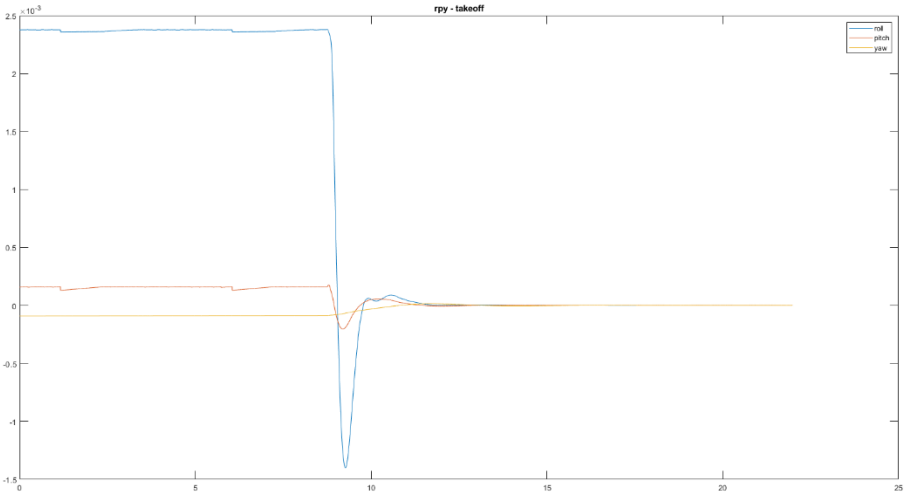
(a)



(b)

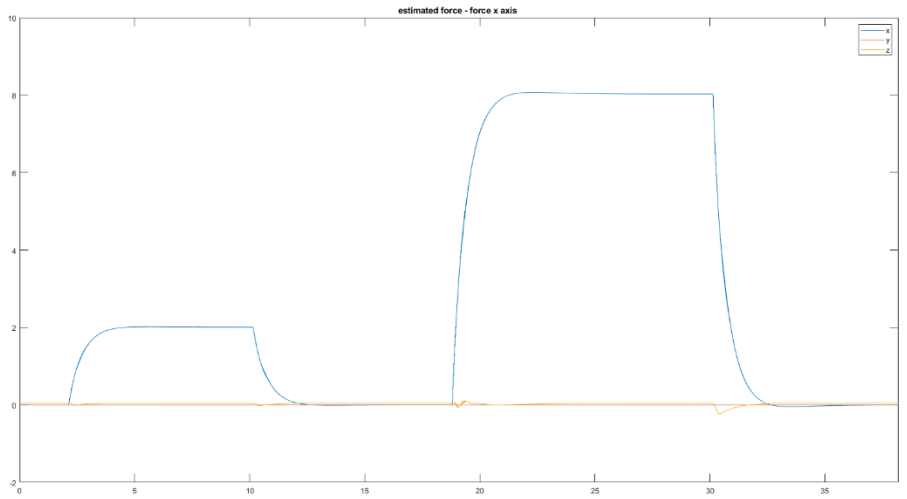


(c)

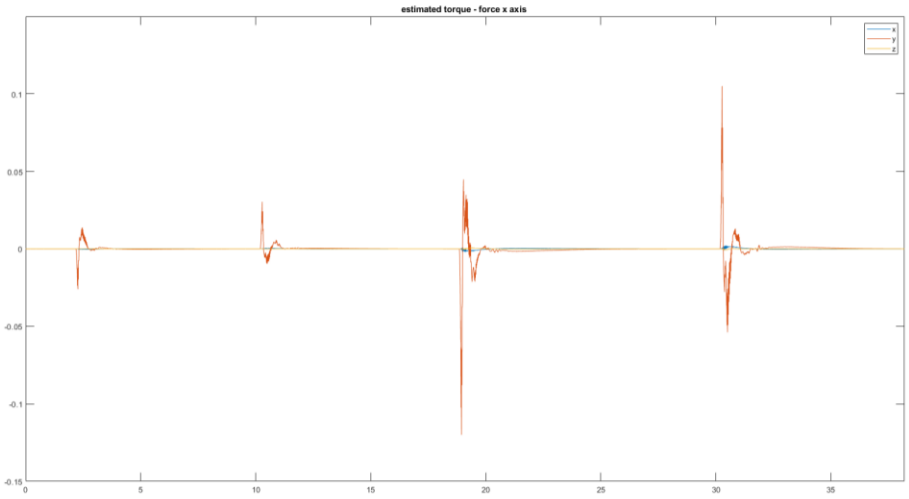


(d)

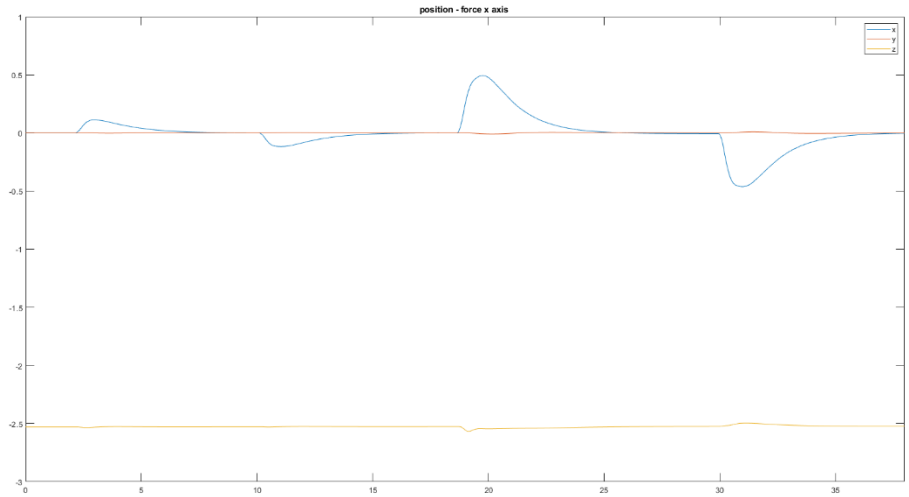
Fig. 2.2 Take-off phase



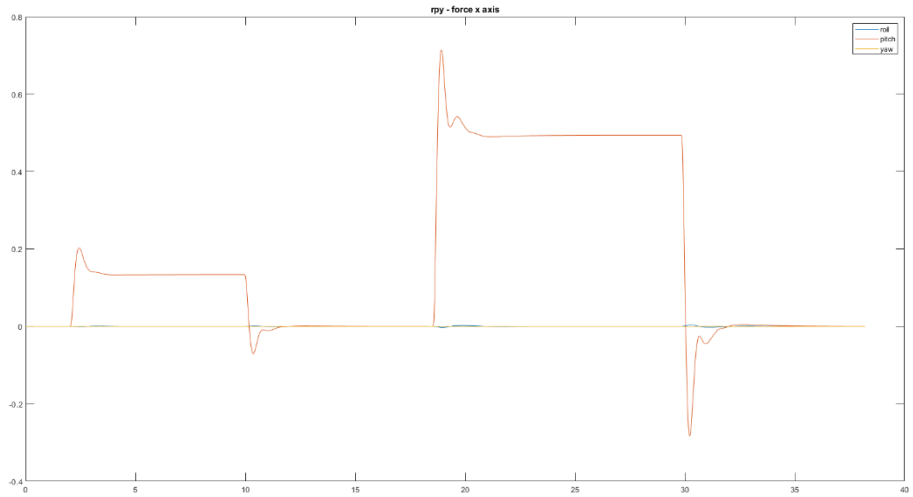
(a)



(b)

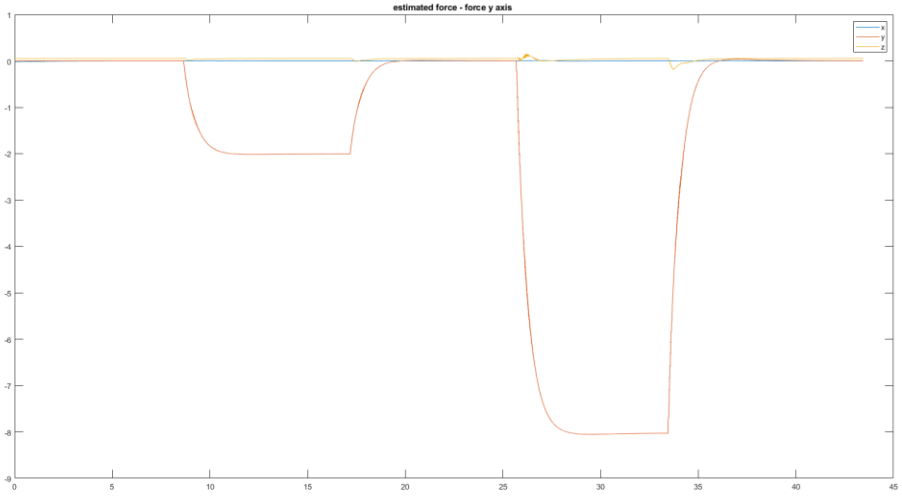


(c)

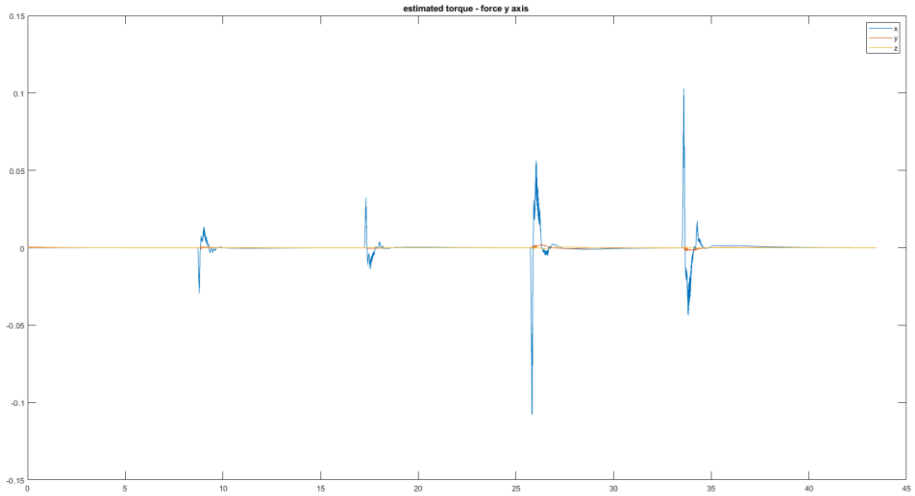


(d)

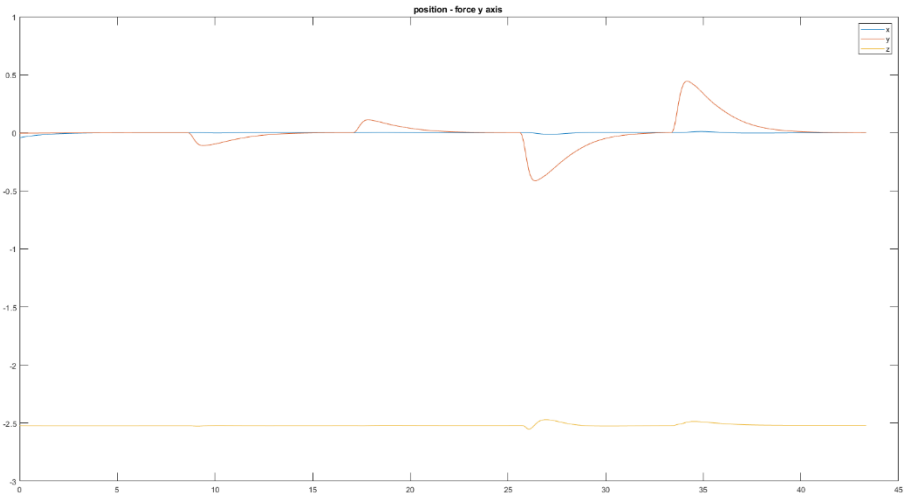
Fig. 2.3 Application and removal of a force along the X axis of 2N and then 8N


estimated force - force y axis
x
y
z

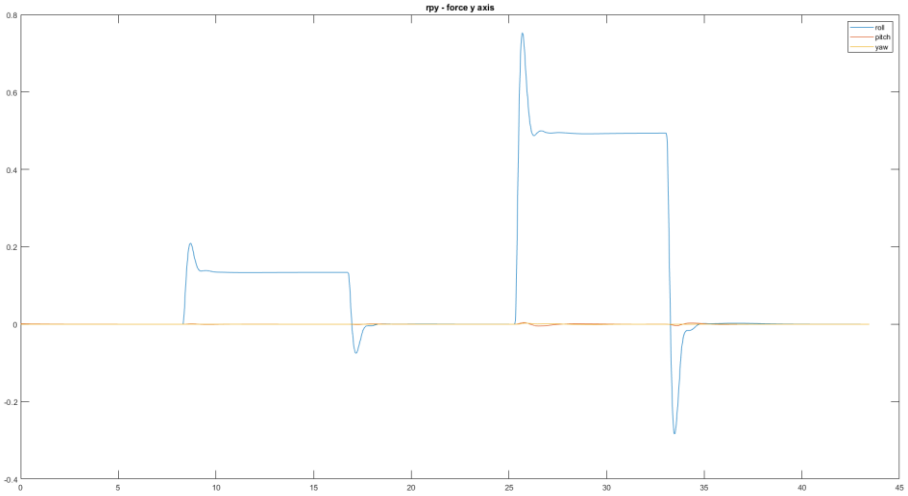
(a)


estimated torque - force y axis
x
y
z

(b)

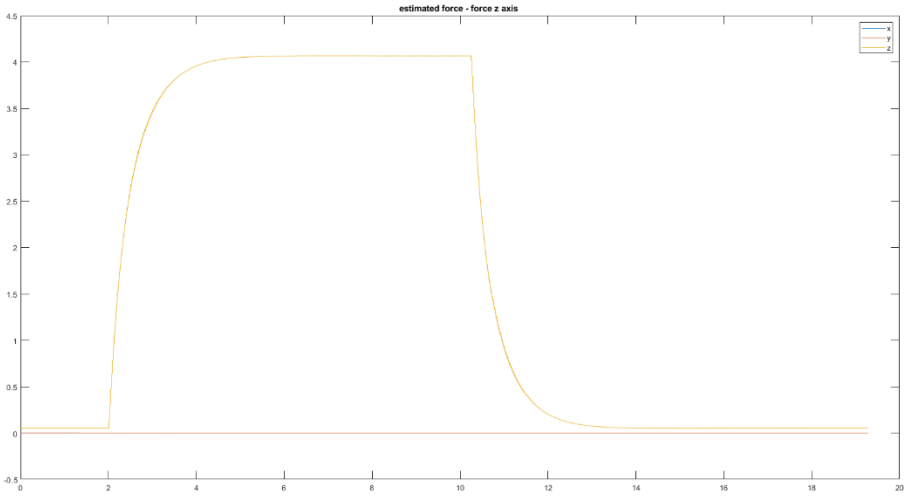

position - force y axis
x
y
z

(c)

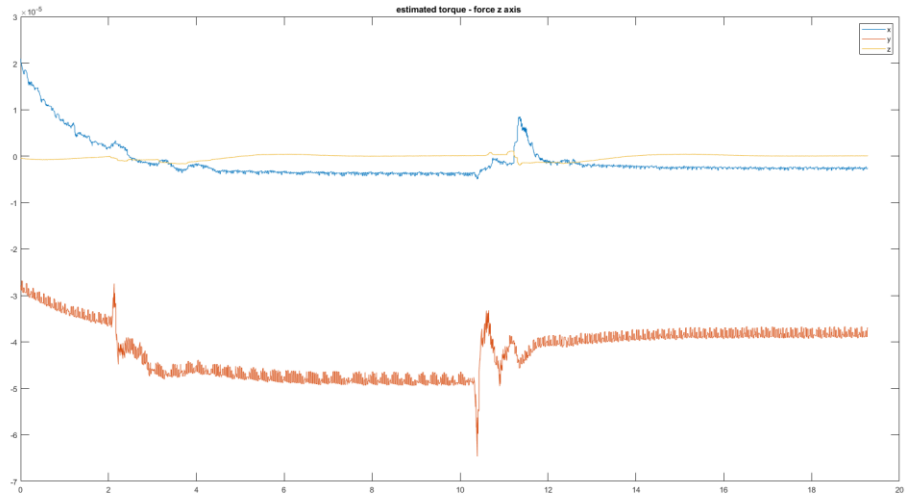


(d)

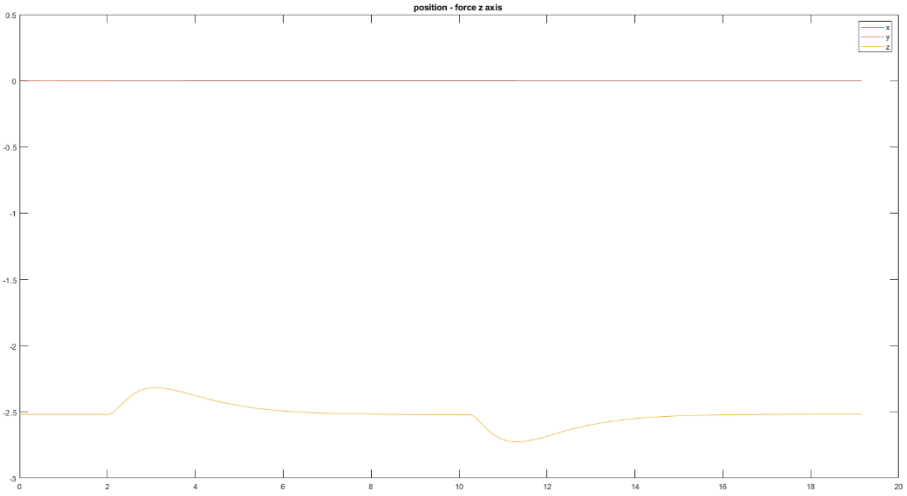
Fig. 2.4 Application and removal of a force along the x-axis of 2N and then 8N



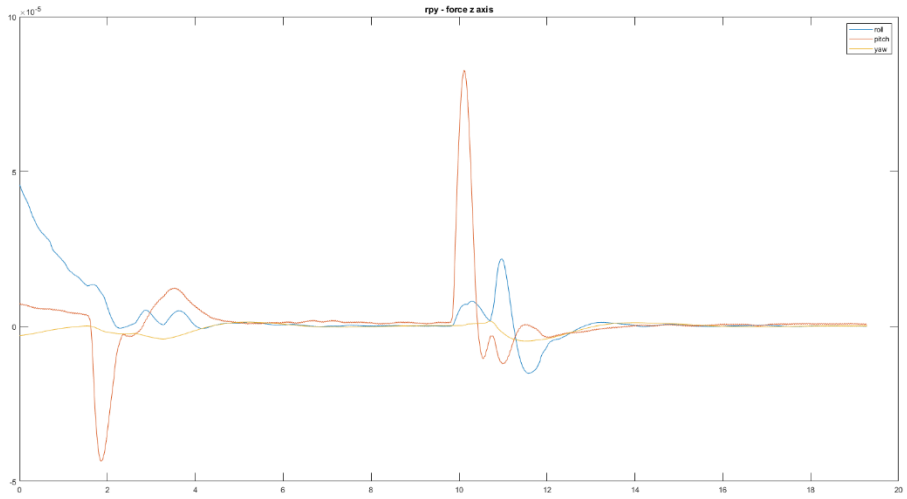
(a)



(b)

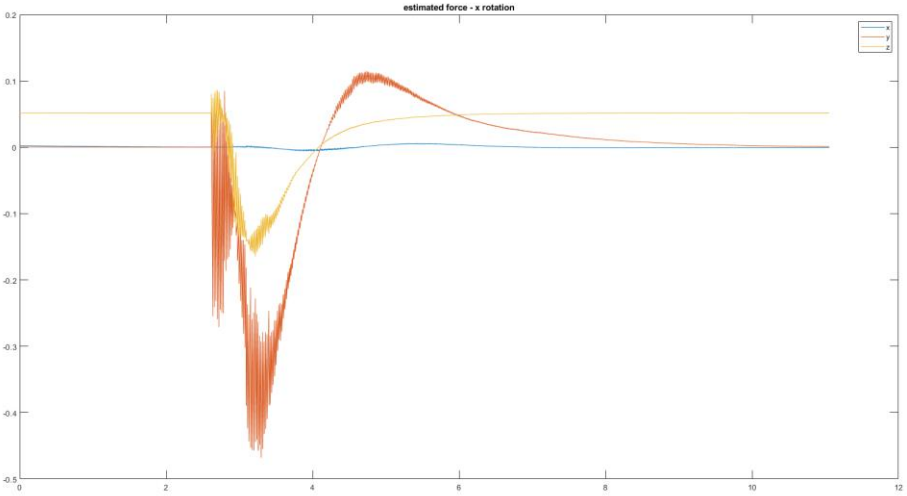


(c)

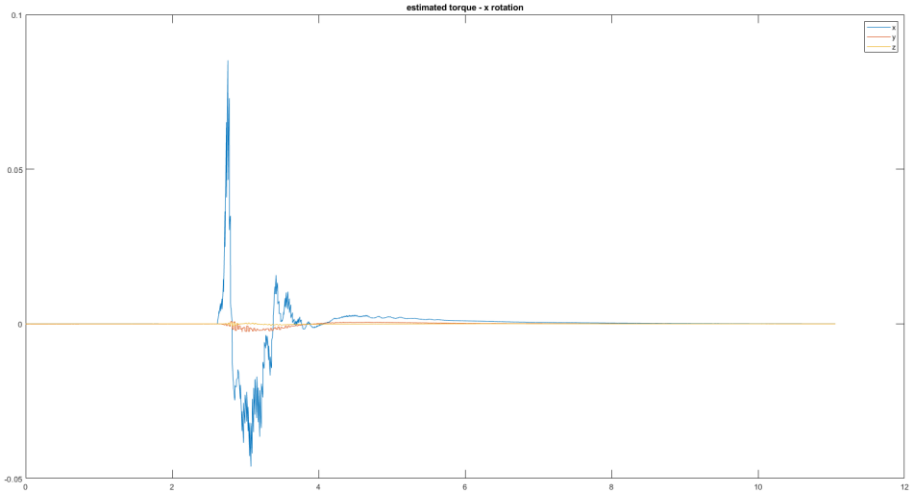


(d)

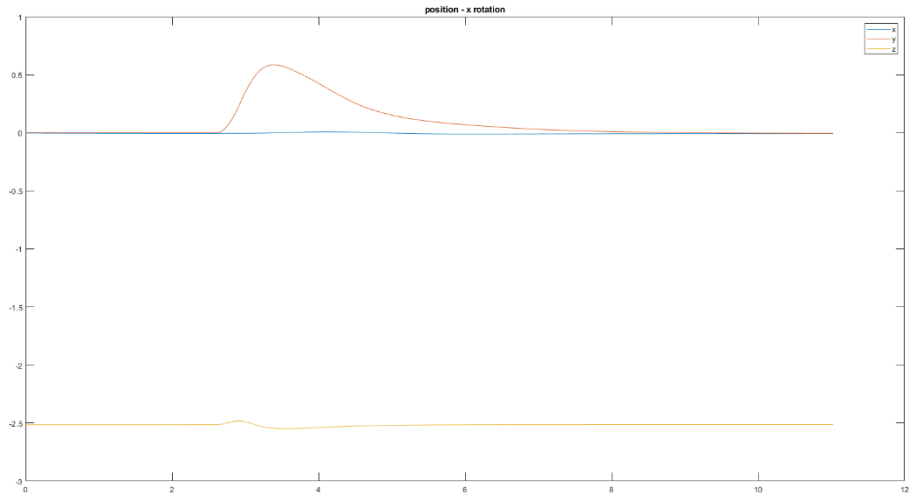
Fig. 2.5 Application and removal of a force along the z-axis of 4N in the direction of the weight force



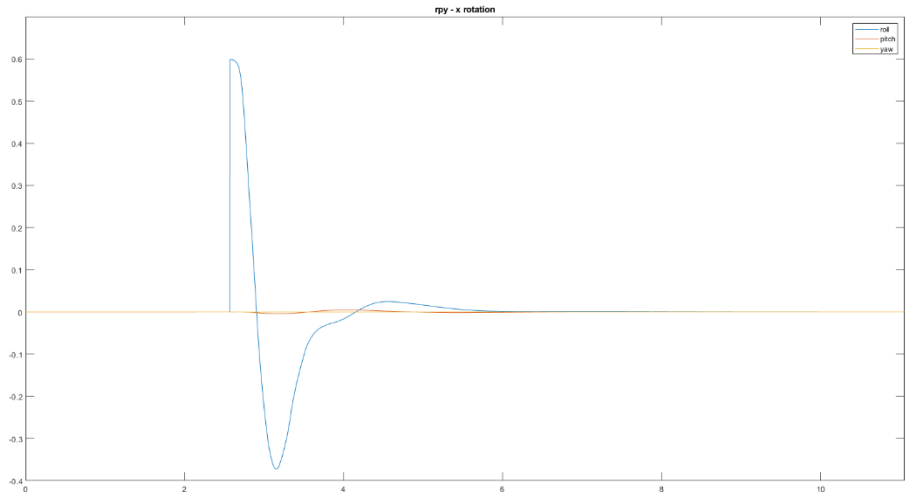
(a)



(b)

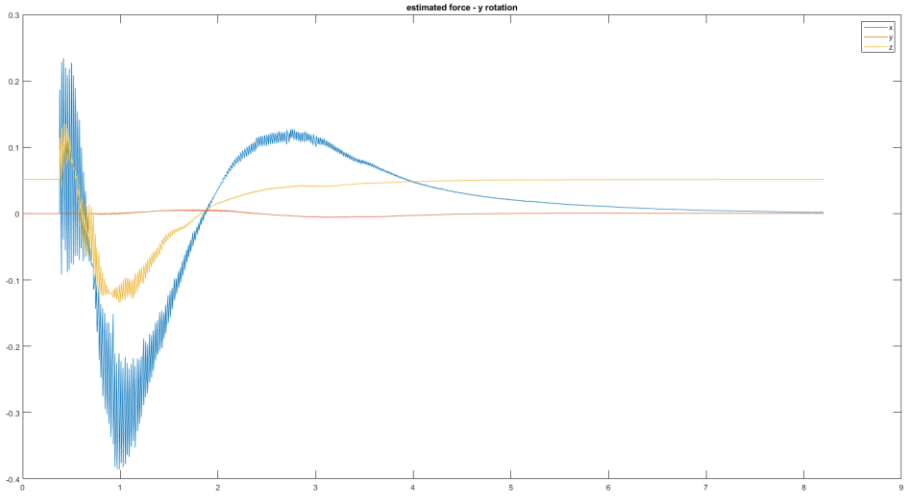


(c)

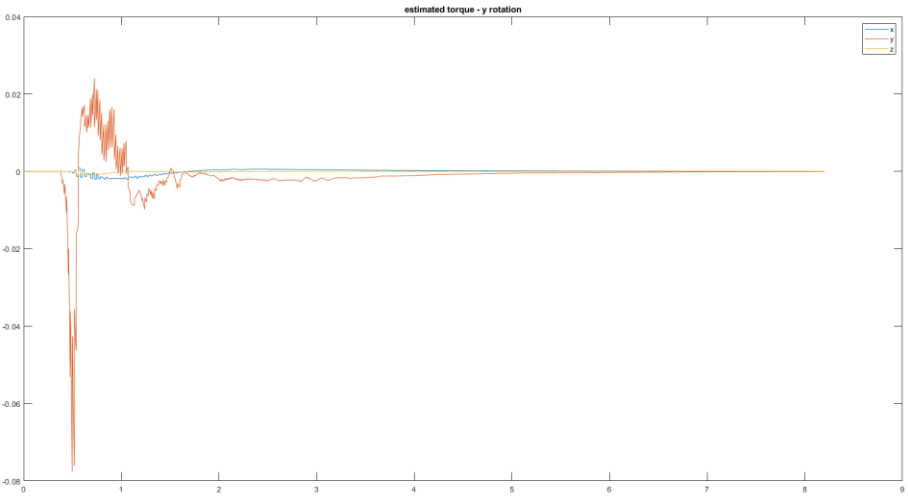


(d)

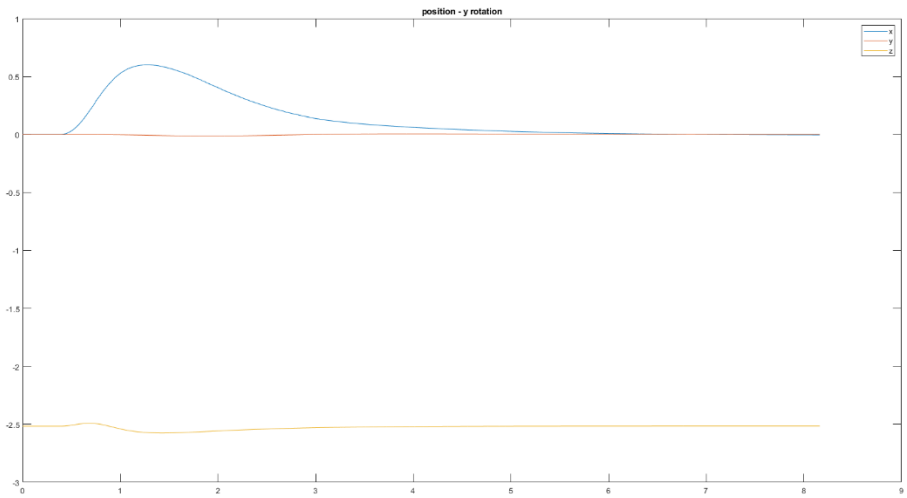
Fig. 2.6 Rotating about the x-axis


estimated force - y rotation
x
y
z

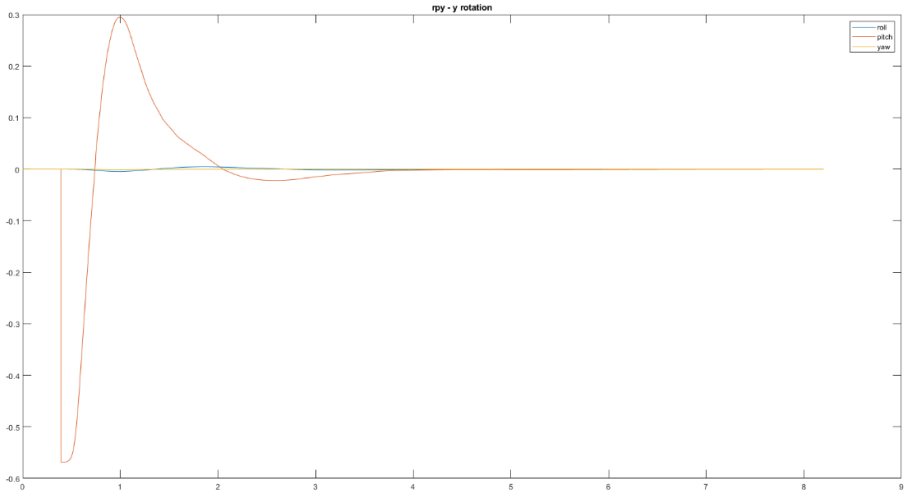
(a)


estimated torque - y rotation
x
y
z

(b)

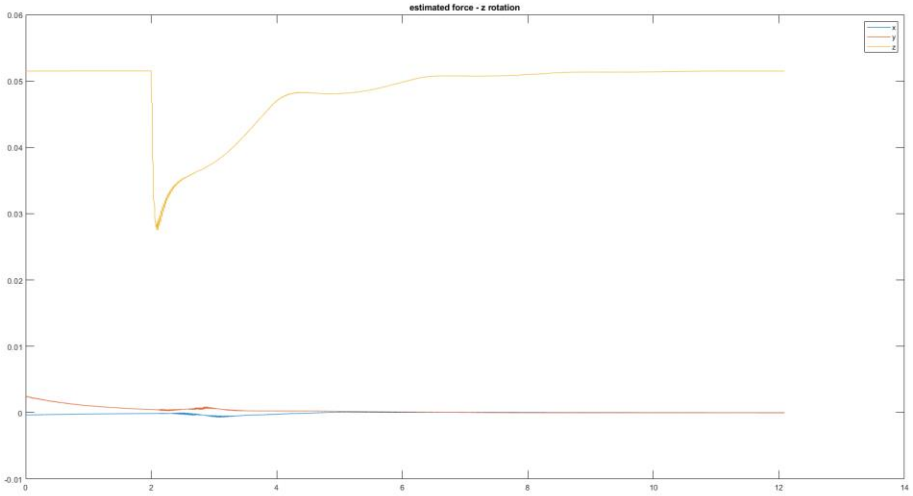

position - y rotation
x
y
z

(c)

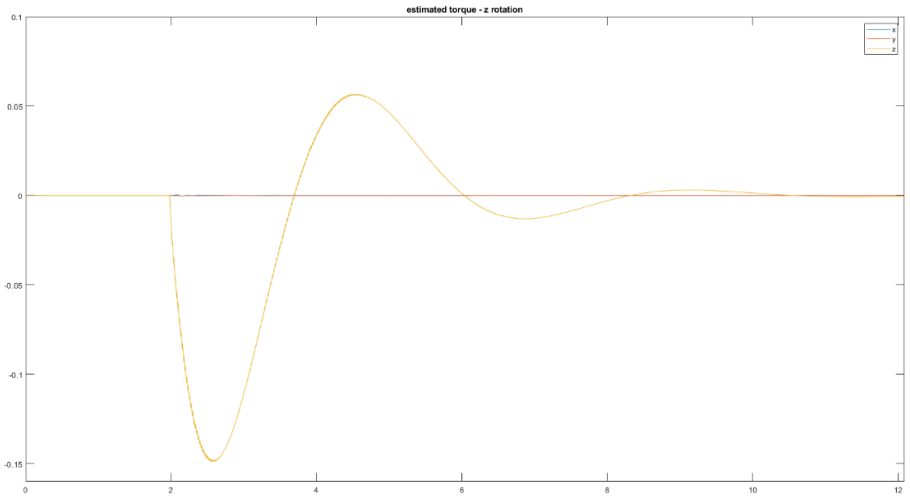


(d)

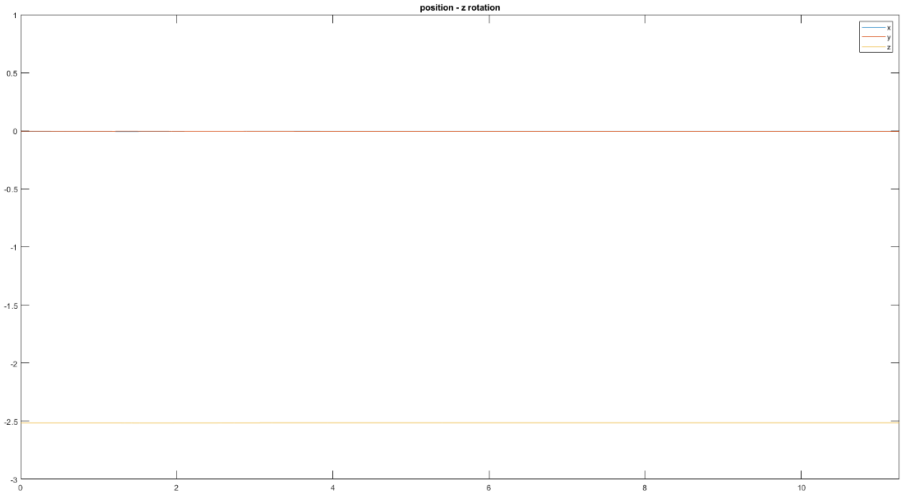
Fig. 2.7 Rotating about the y-axis



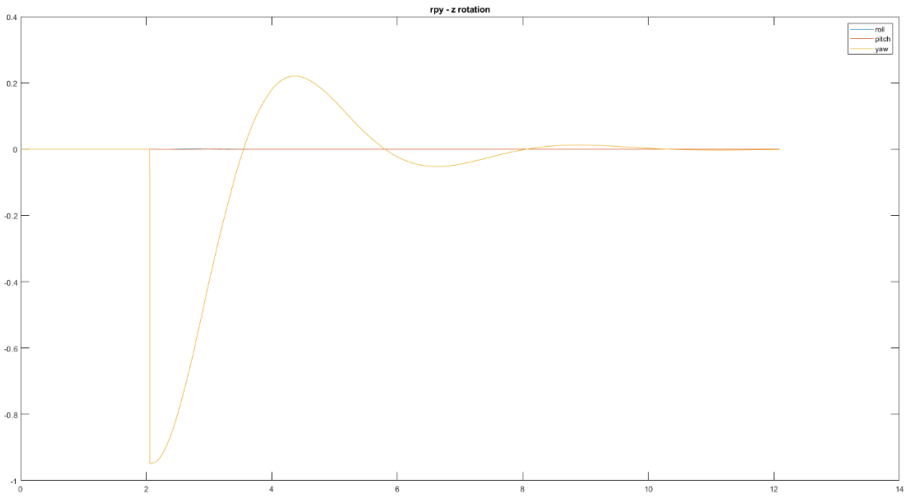
(a)



(b)



(c)



(d)

Fig. 2.8 Rotating about the z-axis

From the graphs it is possible to see how the low-level control works well because it manages to correct all the perturbations that the system may suffer by bringing the drone back to the desired position.

3. Planner - FSR

For the online planner, the method with artificial potentials was used. It consists in calculating for each position of the quadcopter a force, which is the result of an attractive and a repulsive force and then it is translate by the controller into the desired acceleration. The force has two components, attractive one is calculate considering the distance from the point to be reached to the drone and repulsive one taking considering proximity to obstacles.

3.1 Attrattive force

$$\begin{cases} f_a = K_a e + K_{a_{vel}} \dot{e} & \left||e|\right| \leq 1 \\ f_a = K_a \dfrac{e}{||e||} + K_{a_{vel}} \dot{e} & \left||e|\right| > 1 \end{cases} \tag{3.1}$$

where:
$e = P_{goal} - P_{UAV}$ is the vector of the position error between the quadcopter and the goal position.

$\dot{e} = -\dot{P}_{UAV}$ is the speed error vector. At the end point, zero velocity along all axes is desired.
The parameters used for constants are as follows:

$$K_a = 3 \qquad K_{a_{vel}} = 4 \tag{3.2}$$

During the approach phase to a window obstacle, only the attractive force is calculated as it is necessary for the drone to get very close to the obstacle. During the approach, the attractive force is limited to prevent the drone from crashing with the obstacle.

3.2 Repulsive force

The repulsive force has the goal to create a repulsive barrier around obstacles. For each obstacle the repulsive force is:

$$\begin{cases} f_r = \dfrac{K_r}{\eta^2}\left(\dfrac{1}{\eta} + \dfrac{1}{\eta_0}\right)^{\gamma-1} \nabla\eta & \eta \leq \eta_0 \\ f_r = 0 & \eta > \eta_0 \end{cases} \tag{3.3}$$

where:
η is the distance to the nearest obstacle
η_0 is the radius of influence, which is the minimum radius from which the presence of an obstacle is considered.

The parameters are:

$$K_r = 1 \quad \gamma = 2 \quad \eta_0 = 2m \tag{3.4}$$

From the tests carried out, in the event that the point to be reached is within the range of influence, it was not reached because the repulsive force is greater than the attractive one. To solve this problem the repulsive force is multiplied by the distance from the target, when this is less than 1*m* but greater than the radius of the drone.

3.3 Total force

The total force is the sum of the attractive force and the repulsive one.

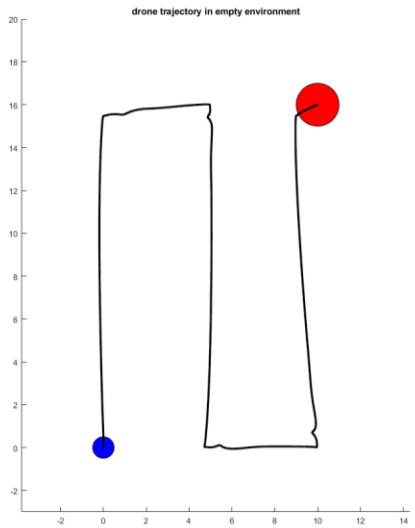
$$F_T = f_a + f_r \tag{3.5}$$

3.3.1 Local minima

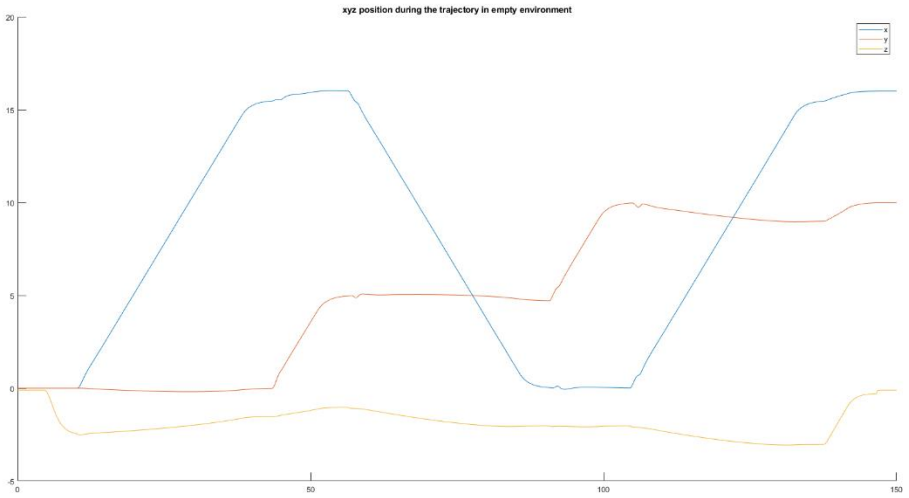
In some cases it is possible to verify that the total force is zero or in any case very small but the quadcopter is not in the goal position and not even close to it, in these cases it is in the presence of a local minimum. To get out of a local minimum, a random force is applied to the quadcopter on the XY plane. If it is not sufficient, it is applied again until the UAV is at a point where the total force is high enough, hoping to move the robot to a path that leads it to the desired point.

3.4 Test

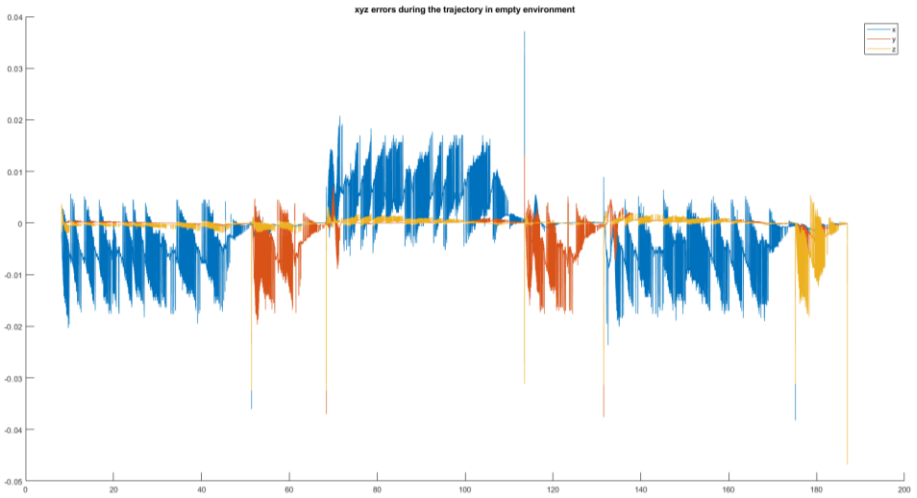
The first reported test is carried out without any obstacle.



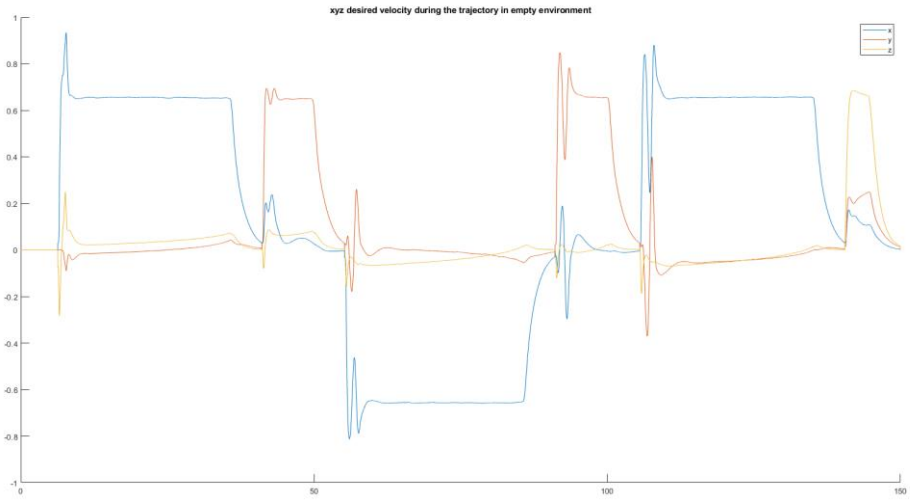
(a)


xyz position during the trajectory in empty environment
x
y
z
20
15
10
5
0
-5
0
50
100
150

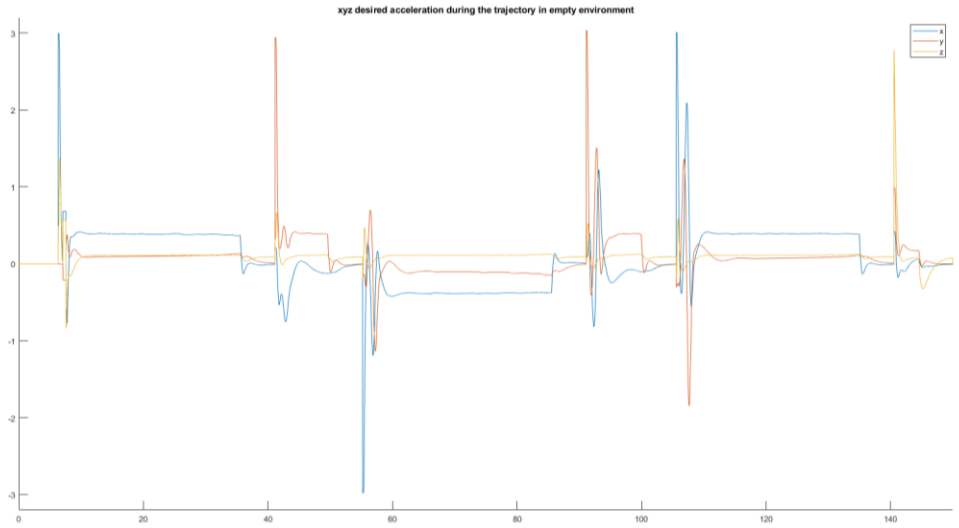
(b)


xyz errors during the trajectory in empty environment
x
y
z
0.04
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
0
20
40
60
80
100
120
140
160
180
200

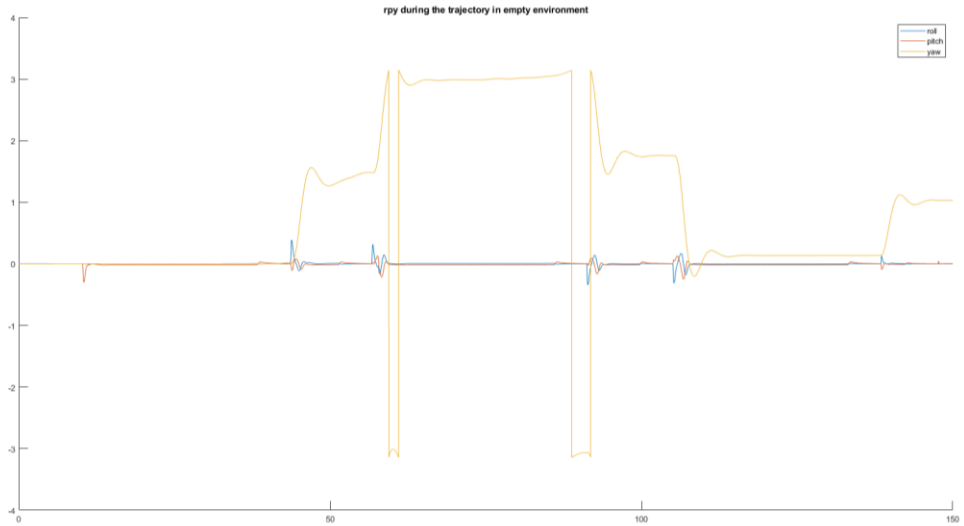
(c)


xyz desired velocity during the trajectory in empty environment
x
y
z
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
50
100
150

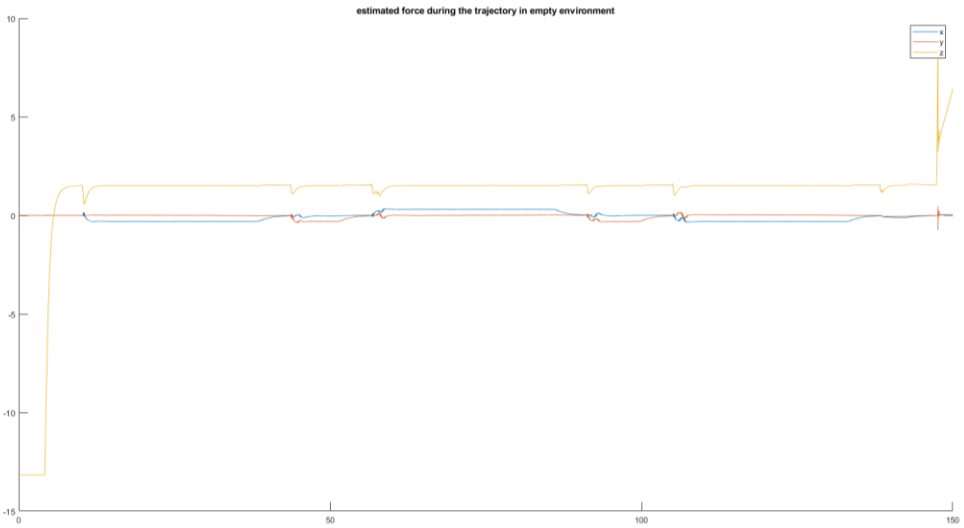
(d)


xyz desired acceleration during the trajectory in empty environment
x
y
z

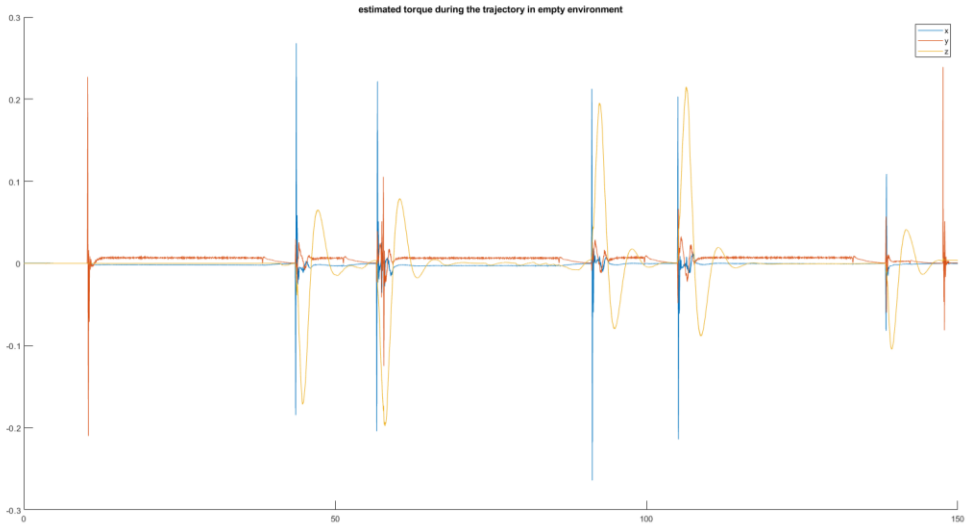
(e)


rpy during the trajectory in empty environment
roll
pitch
yaw

(f)


estimated force during the trajectory in empty environment
x
y
z

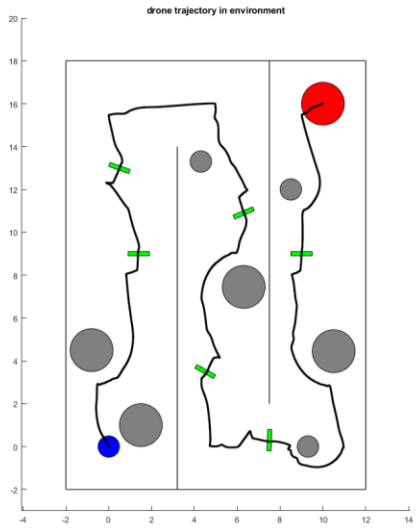
(g)



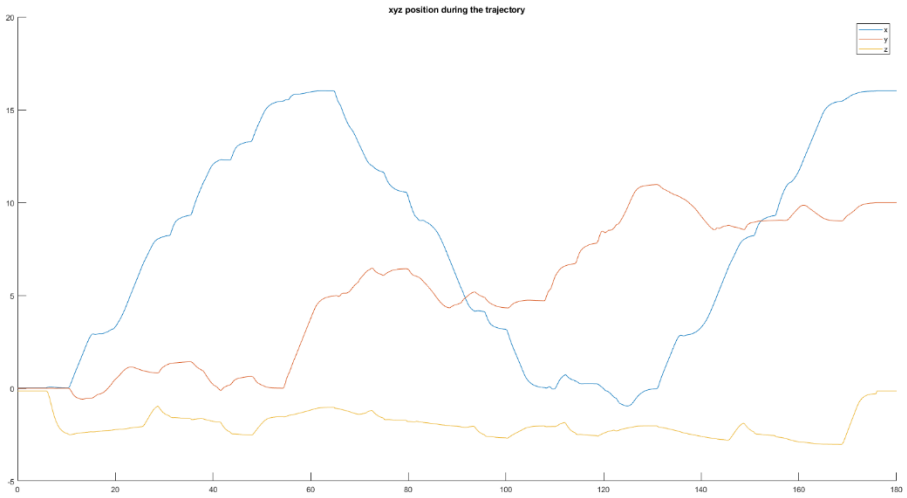
(h)

Fig. 3.1 Simulation of trajectory in an empty environment

In this simulation, the graphs were obtained from the simulation of the full trajectory, which provides for the departure from the start platform to the arrival platform passing through the waypoints without colliding with walls and obstacles, as well as passing inside the windows.



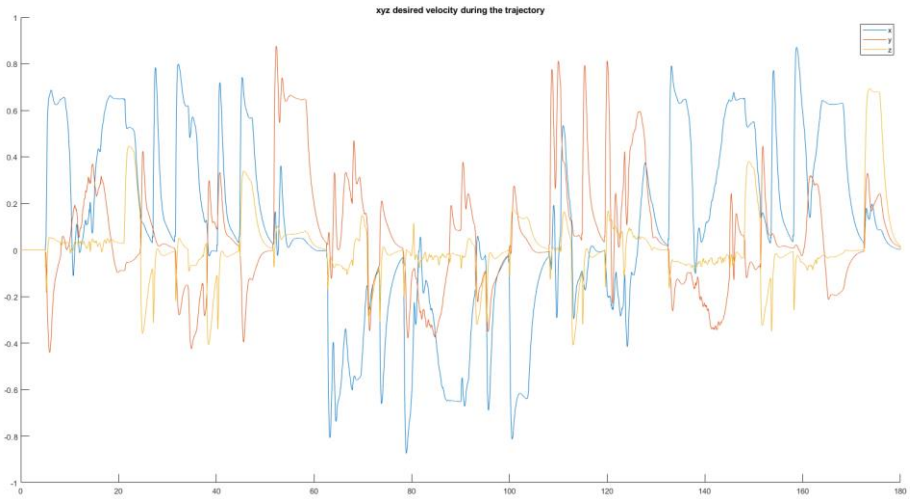
(a)


xyz position during the trajectory
x
y
z

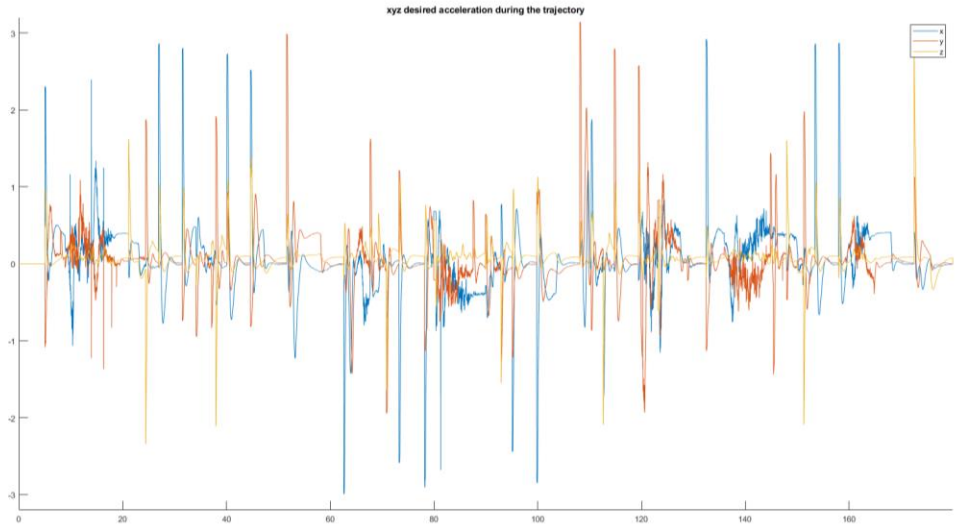
(b)


xyz errors during the trajectory in empty environment
x
y
z

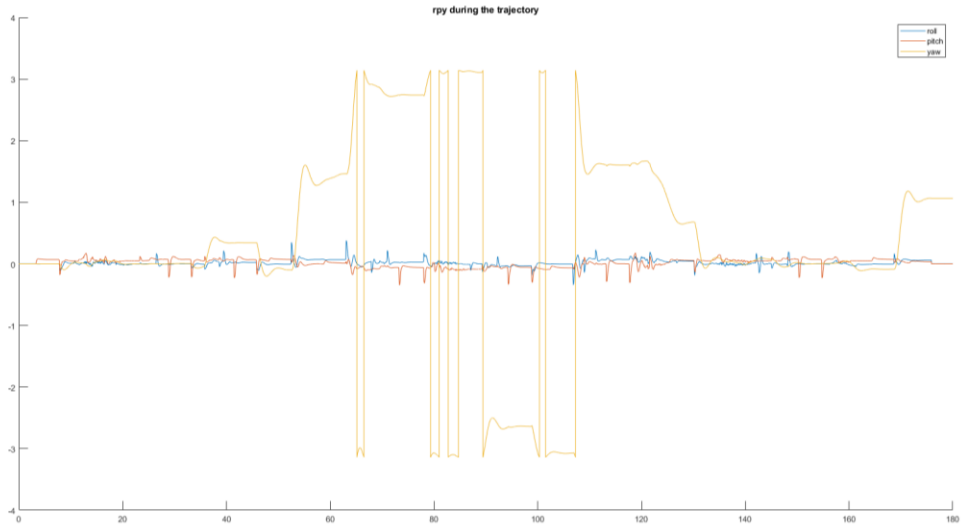
(c)


xyz desired velocity during the trajectory
x
y
z

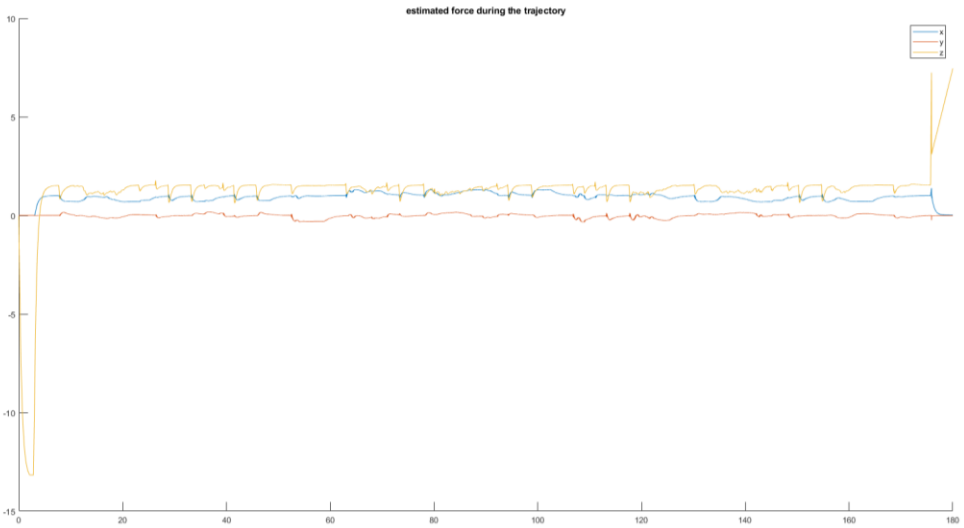
(d)


xyz desired acceleration during the trajectory
x
y
z

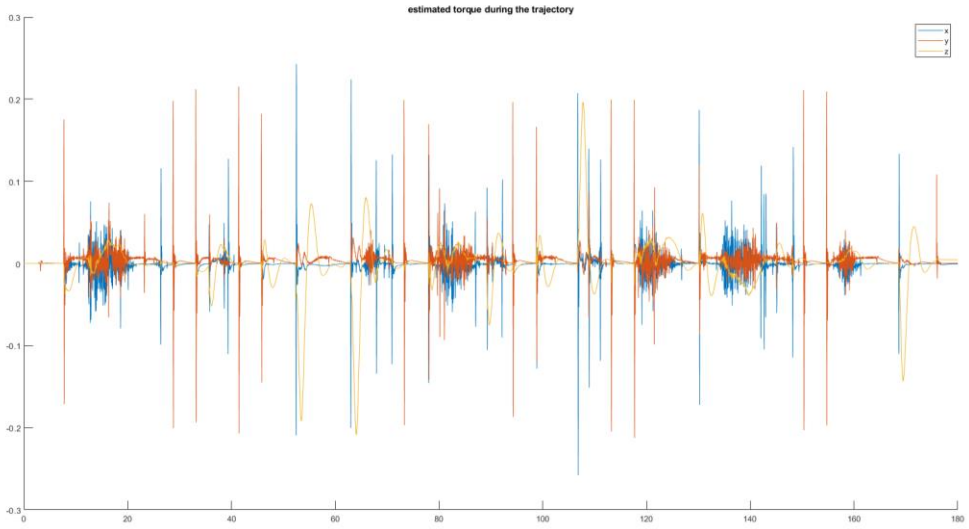
(e)


rpy during the trajectory
roll
pitch
yaw

(f)


estimated force during the trajectory
x
y
z

(g)



(h)

Fig. 3.2 Simulation of trajectory in the environment

4. Overview - RL

In this paragraph will analyze in little detail the structure of the control program implemented in order to understand how the different parties work together. The first step was to generate the environment, this was possible through the Gazebo Building Editor, which allowed to create the room with the different walls and then this environment was saved as an SDF Object file. Some elements have been imported manually, such as solid obstacles and window obstacles. Some textures have been changed. In a ".world" file, all the different components have been assembled and set the most appropriate position and orientation for each of them to create the test environment.
The next step was to insert the quadcopter. The UAV used was imported from the RotorS Library and the camera and Lidar were added to the model, one for the recognition of markers and the other to measure the distance from obstacles.
The files identified above are all placed in the "my_scenario" folder, it also contains the "launch" files that have the task of starting Gazebo with the world and the drone. An envairoment without walls and obstacles were generated to carry out tests. You then have another "launch" file to start the Aruco marker recognizer and load the parameters used by the controller and planner.
After, it is possible to analyze the controller and the planner in two different packets, which communicate with each other through a communication protocol. You can view the schema of the project structure below.
To simulate the constant force of 1N required by the project, a constant acceleration along the X axis of $0{,}66\,\frac{m}{s^2}$ was added. This acceleration was calculated by dividing the force required by the real mass of the drone.

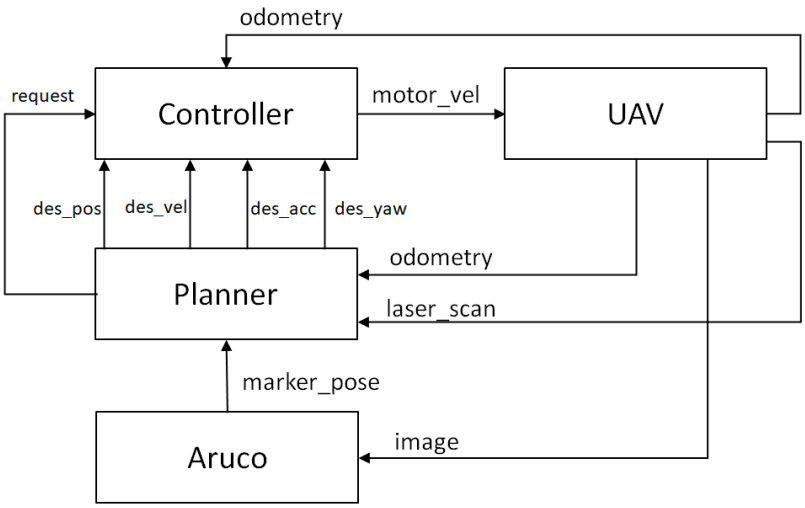


Fig. 4.1 ROS project structure

The external libraries used, in addition to those for message communication, are: Eigen for use with arrays and tf to simplify transformations between rotation matrices and corners in RPY.

5. Controller - RL

The "controller" package implements low-level control for the drone and implements control through keyboard input to guide the drone as if it were remote-controlled. The low-level control loop is performed cyclically with a frequency of 1kHz via the "*vel_ctrl"* thread, which calculates the matrices and all the equations necessary to compute the speed of the propellers. There is also a function to stop the propellers, this function can be activated with keyboard control or automatically, at the end of the landing phase.

5.1 Control loop

The control cycle works at a frequency of 1kHz to eliminate the position error between the drone and the points received from the planner or received from the keyboard. The takeoff and landing phase is also managed by the controller. As seen in chapter 2, where the chosen control was illustrated, it was necessary to insert an estimator to correct errors due to external forces or incorrect parameters. For the estimator, a dedicated function has been created which estimates the external forces and pairs. The final result of the calculation by the controller is the speed of the propellers that is communicated through the topic "iris/cmd_motor_vel" to allow simulation via Gazibo.

5.2 Estimator

The estimator works at a frequency of 1000Hz.
All the matrices necessary for the calculation of the estimate are obtained in the callback of the odometry because they depend only on the odometry itself. The thrust and desired torques are calculated in the control cycle. A Boolean variable is used to indicate that the estimator is ready. To be ready, wait two seconds from the start of the prediction, this variable is used to start the control cycle. This ensures that, as previously mentioned in chapter 2, the estimator must necessarily start before the takeoff. The result of the calculations is saved in two variables, one for force and one for torque and they are used in the control loop to calculate the speed of the different propellers to obtain the desired movement.

5.3 Keyboard input

A function was implemented to allow the movement of the remote-controlled drone, without initial planning. Using this control method, it was possible to carry out tests of the performance of the low-level control and then left it available. Through the use of this function is possible: to move the drone with relative displacements of 0.05m along the X and Y axes, 0.25m along the Z axis, rotate it around the Z axis by a relative angle equal to 10 ° and to make a takeoff that is the ascension to 2.5m. The movements that are imposed with this function do not give any reference on the desired speed and acceleration, thus not obtaining the smooth movements of the drone.

6. Planner – RL

The goal of the planner is to provide the control loop with the positions to be obtained by the drone so that it starts from the initial point and reaches the final one passing between the keypoints without crashing with the full obstacles and passing between the window ones.
The first thing that the planner does is send the references to the controller to make the takeoff, a height of 2.5m is imposed, keeping X and Y constant and set zero speed and accelerations desired. Once the take-off process is finished, the planner acquires from the waypoints carrier the point to be reached and begins to calculate the potential force considering the position where the drone is located, that of the waypoint to be reached and any obstacle nearby.
The planner has a vector in which the positions of the waypoints are saved and an index that indicates the position of the one currently to be reached. Callbacks for obstacle recognition and marker detection were then implemented. Once you reach the last waypoint you make a landing phase and when you are close to the landing base you send to the controller, through a message, the command to turn off the engines.

6.1 Planner loop

The planner cycle calculates with a frequency of 100Hz the attractive and repulsive force to derive the total one. The desired acceleration is therefore calculated and allows to derive the desired speed and position. The control loop performs a different procedure if the drone camera frames a marker and it is at a distance less than or equal to 3 meters. When this happens, the process of approaching and then passing inside the window begins. While the drone does not detect the marker, the potential force guides the drone to the waypoint following the last reached, or the first in the initial case and imposes a desired yaw on the drone.
The orientation of the drone is always directed towards the waypoint to be reached, after reaching the desired point and updating the next there were rapid changes of desired orientation, a function was implemented that guarantees a gradual increase or decrease of the angle until the desired direction is reached in order not to cause strong oscillations of the drone. When the drone is close to the point to be reached, it no longer changes the orientation, keeping the last calculated when it was at a distance of 0.2m.
During the execution of the planner, the Aruco callback is always active, which acquires data on the presence of a marker and its position. From the position of the marker we obtain the center of the window in which the drone must pass. In the event that the drone is at a distance of less than 2.5m from the center of the window, the planner starts a procedure that leads to the passage of the drone through it.
The procedure involves the following steps:
- orientation of the drone in parallel to the obstacle and approach to the marker in order to make more accurate readings and better estimate the position of the window as from some tests carried out, the reading by the Aruco package of the position and orientation of the marker were not precise if you were not very close to the marker
- update the position of the window and move towards a point close to the window and 0.7 m away along the axis perpendicular to the window
-reached the point described in the previous step, it is necessary to move along the axis perpendicular to the window and distant 0.4m from it.

The passage from one step to another of the procedure for passing through the window is indicated by some Boolean variables that are shown below:
- win_obstacle_appr_agg, this variable becomes true in case the drone displays the marker for the first time at a distance of less than 2.5m
- next_aggiornamento becomes true as soon as the drone displays the marker at a distance of less than 2.5m, indicating that an update of the marker position is needed to improve the estimation of the position of the center of the window.

- win_obstacle_appr, this variable becomes true when the drone is sufficiently close to the window and then to the marker and has updated the position of the center of the window more accurately, so that this variable indicates the possibility of the drone to move towards the point that is on the axis of the window before passing inside.

- win_obstacle_pass, this variable becomes true to authorize the passage of the window, that is, when the drone is in the direction of it.

After passing through a window, the drone returns to calculate the trajectory to be followed using the potential force and having as its objective the waypoint that it wanted to reach before the window crossing operations.

6.2 Aruco and laser callback

To move avoiding obstacles or to cross them, the planner uses the presence of a camera and a laser. The data from the camera is used by the Aruco package that estimates the position and orientation of the marker, if present within the image. Inside the planner you have the callback dedicated to the laser that is used to estimate the nearest obstacle and in particular the direction of the point at a shorter distance. Another callback, receives the data from the Aruco packet and processes them in order to estimate the points needed to carry out the different steps for the passage inside the window after making transformations, to bring the coordinates acquired by the camera back into world-frame coordinates.
Through the RViz application it is possible to view in a simpler way the data that are acquired by the camera and then processed to recognize the marker. In the figure it is possible to see how the software was able to identify the marker and also to identify the position, represented by the center of the axes and the orientation that determines their spatial arrangement.

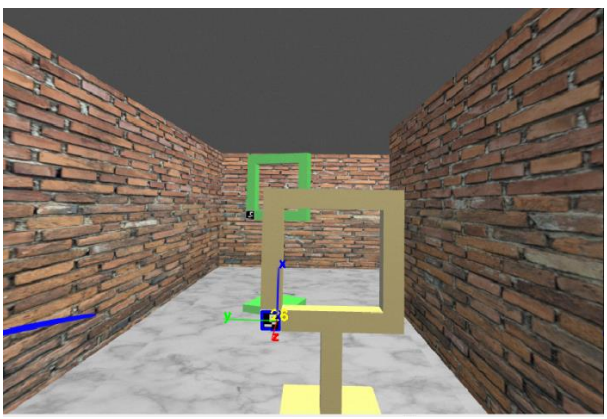
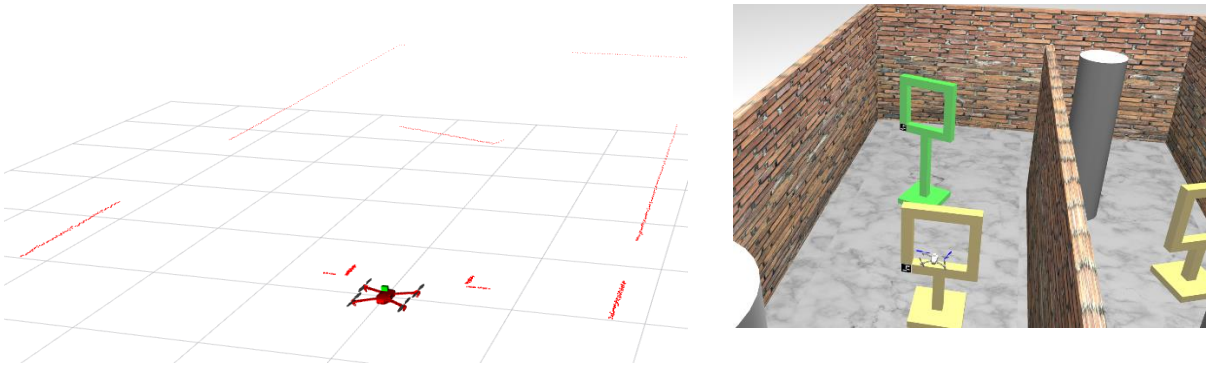


Fig. 6.1 Aruco package feedback in image form

RViz also allows you to have a graphic view of the different points identified by the laser and in the image shown you can observe a snapshot in which you can see in red the points identified by the laser, they can be easily identified in the nearby image in which the drone is shown in the environment. In fact, red dots correspond to walls and obstacles.



(a) (b)

Fig. 6.2 (a) Rviz view of LiDAR feedback; (b) View of drone in enviroment