POLITECNICO
MILANO 1863

AUTOMATION AND CONTROL LABORATORY - FINAL REPORT

Group: Quadcopter

222837 – RICCIARDI Andrea
221278 – CALEFATO Sebastiano
225917 – GABRIELLI Simone
224885 – TARANTINO Matteo
995867 – BARANI Lorenzo

Supervisor

Dr. CAZZULANI Gabriele

Academic Year 2023-2024

Contents

Table of Figures

1) Goals of the project

The aim of the project is to control the attitude of a quadcopter, in particular, due to the drone configuration, we want to design a controller capable of performing a control action on pitch and roll angles in such a way to maintain the quadcopter into an equilibrium position.

2) Theoretical background

The holybro-S500 is a quadcopter formed by four symmetrical rotor-propellers. Each propeller is mounted at the same height on each edge of the quadcopter. Due to a great control of the thrust given by each motor the quadcopter can fulfill the 3D rotational asset and the translational movement. The quadcopter is a system that has six degrees of freedom: three translational and three rotational ones, however in our project the central body of the quadcopter is fixed with a hinge, so the 3 degrees of freedom related to the translational movements are constrained and the drone is just free to rotate in 3D space. In order to perform a complete control action about the rotation on the 3D space we have to consider the 3 Euler angles:

- Roll (φ): is the angle of the rotation around the x axis of the drone
- Pitch (θ):, is the angle of the rotation around the y axis of the drone
- Yaw (ψ):, is the angle of the rotation around the z axis of the drone.

Nevertheless, our physical configuration does not allow the drone to rotate freely in yaw due to the presence of cables.
In order to perform the attitude control of the drone we can use as reference model the "+"or "x" configuration. In the plus configuration the x and y axis are oriented following the arms of the drone while in the "x" configuration these axes are rotated of 45° with respect to the arms of the drone.

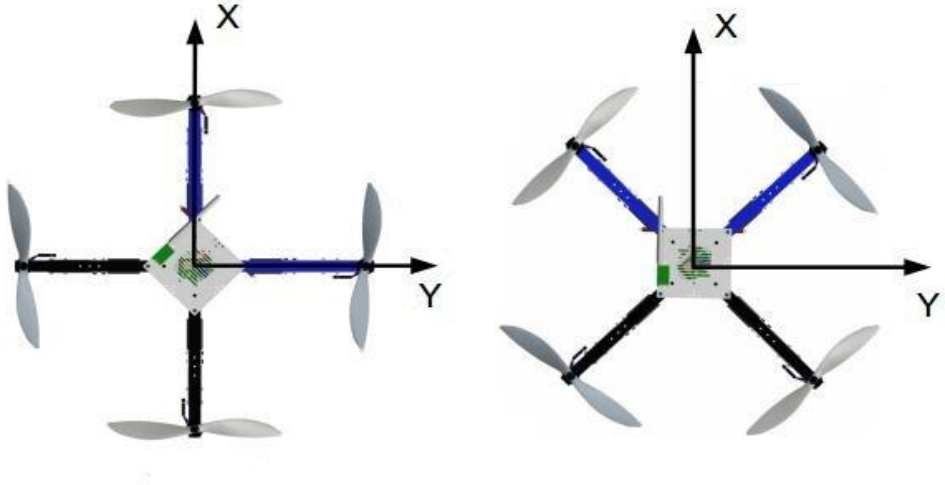


Figure 1: Drone Configurations

To change the reference system for the drone measurements from one to the other configuration, it is necessary to use the rotation matrix. These considerations were made on the reference frame orientation of the gyroscope:

$$R_{+}^{x} = \begin{bmatrix} \cos(-135) & -\sin(-135) & 0 \\ \sin(-135) & \cos(-135) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \frac{\sqrt{2}}{2}\begin{bmatrix} -1 & 1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & \frac{2}{\sqrt{2}} \end{bmatrix}$$

$$R_{x}^{+} = R_{+}^{x\,-1} = R_{+}^{x\,T}$$

3) Sensors and Actuators

3.1 Sensors

The holybro-S500 quadcopter has several sensors [1], between which the are the accelerometer and the gyroscope, which were the mostly used.

3.1.1 Accelerometer

The accelerometer that our quadcopter has installed can measure the acceleration along the 3 directions of motion (x, y and z), hence measuring $a_m = [a_{mx}; a_{my}; a_{mz}]$.
Performing several measurements moving the drone in every direction we were able to identify the reference frame of this sensor, where the x axis, considering the drone still in an "x" configuration, is pointing to the south, while the y axis is pointing to the west, and the z direction is pointing upwards:

Figure 2: Accelerometer Reference Frame

It is therefore evident that the frame of the accelerometer is configured as a left-handed triple.

3.1.2 Gyroscope

The gyroscope installed on our drone can measure the roll, pitch and yaw angular velocities, hence it measures $\omega_m = [\omega_{xm};\ \omega_{ym};\ \omega_{zm}]$.
Performing several measurements moving the drone in every direction we were able to identify the reference frame of this sensor, where the x axis, considering the drone still in an "x" configuration, is pointing to the north, while the y axis is pointing to the est, and the z direction is pointing upwards:

Figure 3: Gyroscope Reference Frame

It is therefore evident that the frame of the gyroscope, like the accelerometer one, is configured as a left-handed triple.

3.2 Actuators

The holybro-S500 quadcopter has 4 actuators, which are the 4 propellers. The motors are arranged as follows:

Figure 4: Motors Configuration

Each of them has a specific PWM/Thrust curve, which we had properly identified.
As to perform those identifications, we needed to setup several experiments to find the desired curve. We firstly performed the automatic calibration of the drone, by using the software "QGroundControl", afterwards we applied a spring on the chosen motor, and we left free the opposite motor, not allowing the spring to contract. We oriented the drone such that the angle between the vertical line and the z axis attached to the body of the drone was positive (counterclockwise direction). The considered spring was correctly characterized by attaching to it a known mass and measuring the elongation of the spring, hence having: $k = m_{known} * g/\Delta_l$.
The remaining two were blocked with a thread, to make our system have only 1 degree of freedom. We used a spring because in this way it was possible to measure its own elongation Δ_l, considering a specific PWM value, hence computing the elastic force given by the spring using this formula:

$$F_{el} = k\Delta_l$$

3.2.1 Motor 1

Considering the 1st propeller, in order to compute the thrust, we firstly measured the values of the acceleration of gravity projected along the x, y and z axis for each PWM value, starting from 1000 and ending in 2000 with a step of 50. As to compute the angle between the vertical line and the z axis attached to the body of the drone, which was needed to compute the elongation of the installed spring given its initial position, we considered those equations:

$$\theta_{xy}^1 = \arcsin\left(\frac{\bar{a}}{g}\right); \theta_z^1 = \arccos\left(\frac{a_z}{g}\right) = \arccos\left(\frac{a_{mz}}{g}\right)$$

They represent the same angle, but computed using different measurements, the 1st using the x and y accelerations and the 2nd using the acceleration along the z axis. Since we were dealing with small angles, the angle obtained using the x and y measurements is more accurate, since it involves the sine instead of the cosine, hence we considered it for our computations ($\theta = \theta_{xy}^1$). Here θ stands for a generic angle, it does not represent the pitch angle.
Those equations come from physical considerations, in fact, considering that the drone is still, it is subject only to the acceleration of gravity g, therefore $a_x^x = a_{mx}$ is the projection of g along the x

axis of the drone and $a_y^x = a_{my}$ is the projection of g along the y axis of the drone. We then projected the 2 measurements onto the x axis of the drone, but considering the "+" configuration, hence we got:

$$a_{xx}^{+} = a_x^x \cos(45°)$$

$$a_{yx}^{+} = a_y^x \sin(45°)$$

Then we summed those 2 components:

$$\bar{a} = a_{xx}^{+} + a_{yx}^{+}$$

Afterward, for obtaining the value of θ, considering that $gsin(\theta) = \bar{a}$, we have the previously stated formula.

Then we used the equation of the moments in order to compute our force (in this case T1). Since we are dealing with little angle, we assumed $\sin(\theta) = \theta$ and $\cos(\theta) = 1$.

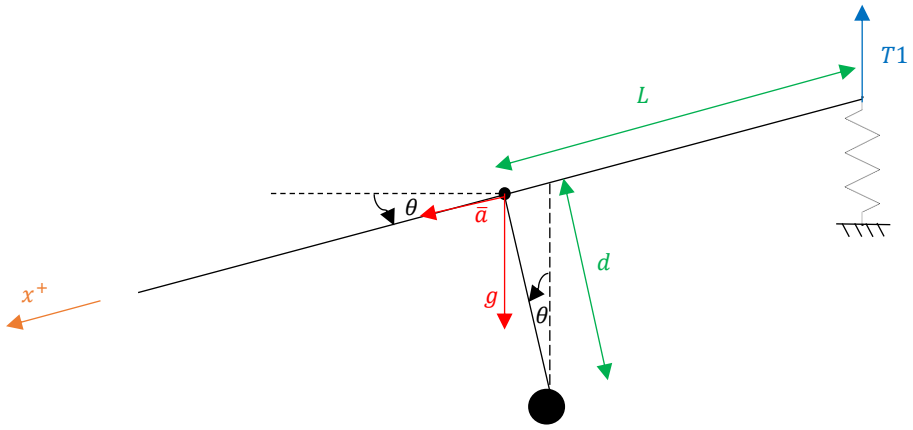


Figure 5: Dynamics of Experiment 1

Static Equilibrium: $(T1 = 0)$

$$k\Delta_{l0}\big(Lcos(\theta_0) - dsin(\theta_0)\big) = mgdsin(\theta_0)$$

$$\Downarrow$$

$$k\Delta_{l0}(L - d\theta_0) = mgd\theta_0$$

$$\Downarrow$$

$$F_{el}^{0} = k\Delta_{l0} = \frac{mgd\theta_0}{(L - d\theta_0)}$$

After having computed the static elastic force we computed the thrust considering the static equilibrium:

Static Equilibrium: $(T1 \neq 0)$

$$(k\Delta_l - T1)\big(Lcos(\theta) - dsin(\theta)\big) = mgdsin(\theta)$$

$$\Downarrow$$

$$(k\Delta_l - T1)(L - d\theta) = mgd\theta$$

Since $\Delta_l = \Delta_{ld} + \Delta_{l0}$:

$$k\Delta_{ld} + F_{el}^{0} - T1 = \frac{mgd\theta}{(L - d\theta)}$$

Hence:

$$T1 = k\Delta_{ld} + F_{el}^{0} - \frac{mgd\theta}{(L - d\theta)}$$

Then, because $\Delta_{ld} = Lsin(\Delta_{\theta_d}) \cong L(\theta - \theta_0)$, we have that:

$$T1 = kL(\theta - \theta_0) + \frac{mgd\theta_0}{(L - d\theta_0)} - \frac{mgd\theta}{(L - d\theta)}$$

In conclusion we computed for each PWM value its associated thrust value.

3.2.2 Motor 2

The considerations for this motor are identical to the ones considered for motor 1, but the only difference is the computation of $\bar{a}$, in fact:

$$a_{xx}^{+} = -a_x^x \cos(45°)$$

$$a_{yx}^{+} = -a_y^x \sin(45°)$$

Hence:

$$\bar{a} = a_{xx}^{+} + a_{xx}^{+}$$

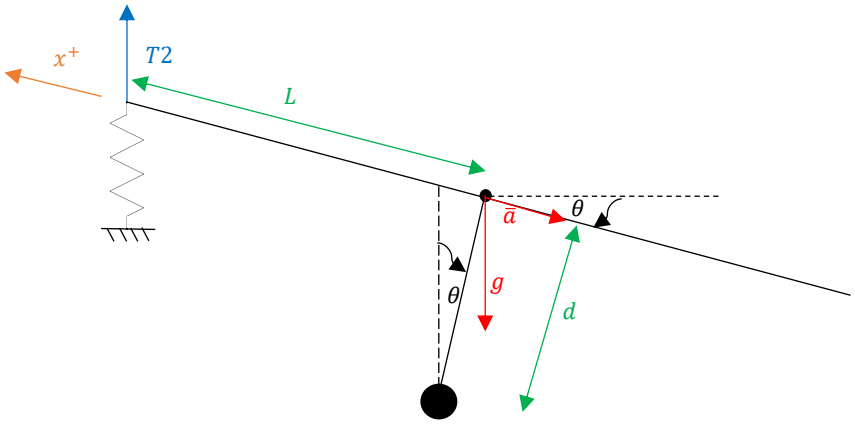


Figure 6: Dynamics of Experiment 2

3.2.3 Motor 3

The considerations for this motor are identical to the ones considered for motor 1, but with the only difference of the computation of $\bar{a}$, in fact:

$$a_{xx}^{+} = a_x^x \cos(45°)$$

$$a_{yx}^{+} = -a_y^x \sin(45°)$$

Hence:

$$\bar{a} = a_{xx}^{+} + a_{xx}^{+}$$

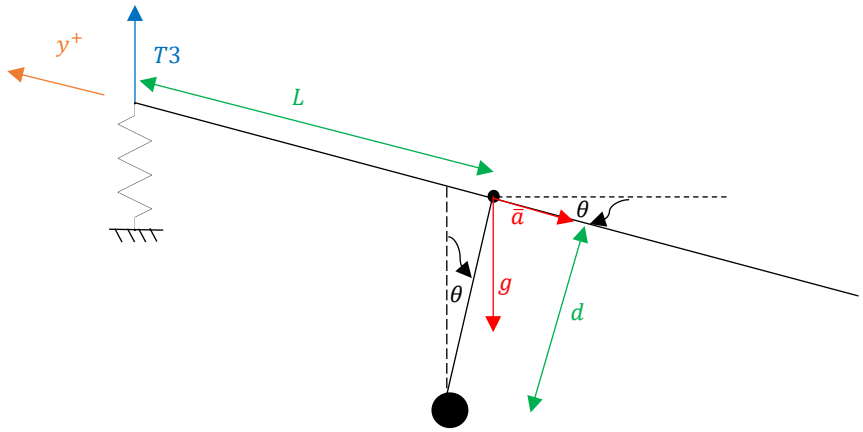


Figure 7: Dynamics of Experiment 3

3.2.4 Motor 4

The considerations for this motor are identical to the ones considered for motor 1, but it differs in the computation of $\bar{a}$, in fact:

$$a_{xx}^{+} = -a_x^x \cos(45°)$$

$$a_{yx}^{+} = a_y^x \sin(45°)$$

Hence:

$$\bar{a} = a_{xx}^{+} + a_{xx}^{+}$$

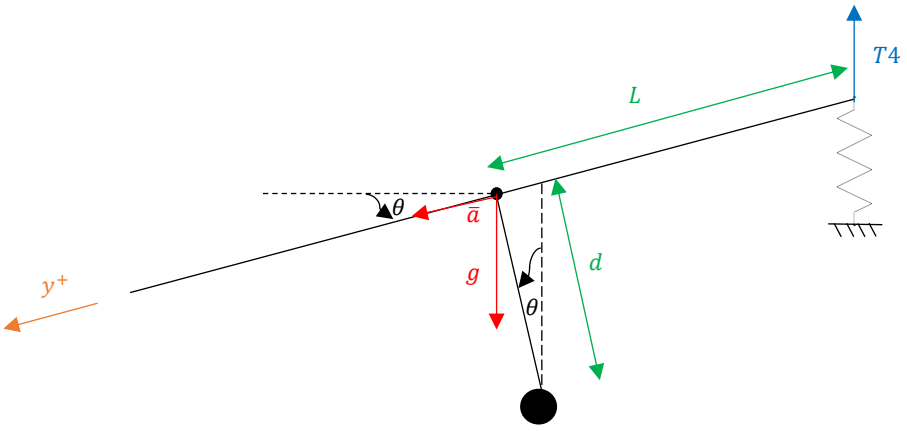


Figure 8: Dynamics of Experiment 4

3.2.5 Curve Fitting

After having collected all the needed data we needed to find the PWM/Thrust curves that interpolated all the points. We used 2 different approaches:

- Neural Networks (NN)
- Look-up Tables

Talking about the NN approach, since a standard model, even a little one, requires a big quantity of data, due to the fact we had 20 rows in our dataset, we performed some data augmentation by creating 2000 artificial samples for each data point. We applied a noise on the Thrust data (that in our case are the inputs), with 0.01 standard deviation, while we did not apply anything to their labels, to simulate a measurement error on the Thrust values. Afterwards we normalized the data between 0 and 1, to make them simpler to interpret for our model.

We built our model, which is a regression one, by connecting several dense layers of limited dimension, to non-overcomplicate it. We used "ReLU" as activation function and we also applied some L2 regularization., in addition to some Batch Normalization layers.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 32)                64

batch_normalization_16 (Bat  (None, 32)                128
chNormalization)

dense_24 (Dense)             (None, 16)                528

batch_normalization_17 (Bat  (None, 16)                64
chNormalization)

dense_25 (Dense)             (None, 1)                 17

=================================================================
Total params: 801
Trainable params: 705
Non-trainable params: 96
```

Figure 9: Model Summary

Then we trained our models, one for each motor, on 100 epochs, but using as validation split the 20% of the dataset and as callbacks the Early-Stopping.

The models were performing good in test phase, but were impossible to implement on the drone, since the memory of the micro-controller is limited, and the models were too big to be stored. For this reason, we switched using the Look-Up Tables, again one for each motor, by linearly fitting all our data-points.

The obtained curves are:

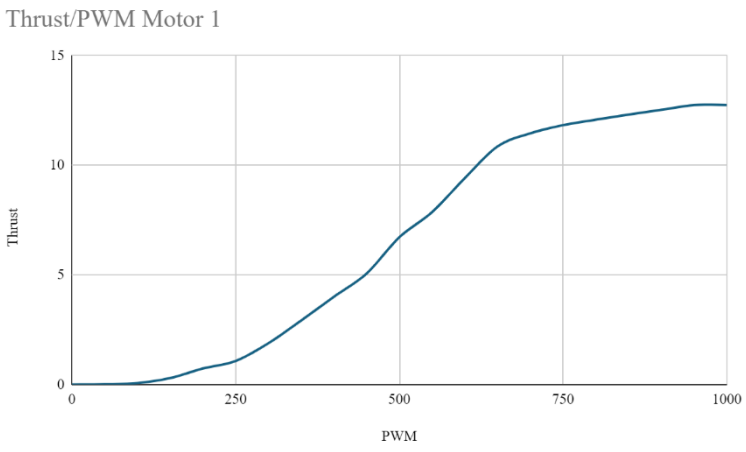


Figure 10: Thrust/PWM Curve Motor 1

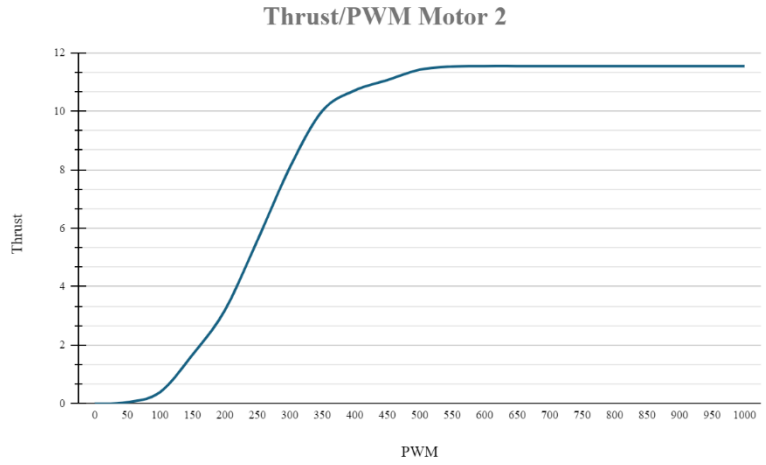


Figure 11: Thrust/PWM Curve Motor 2

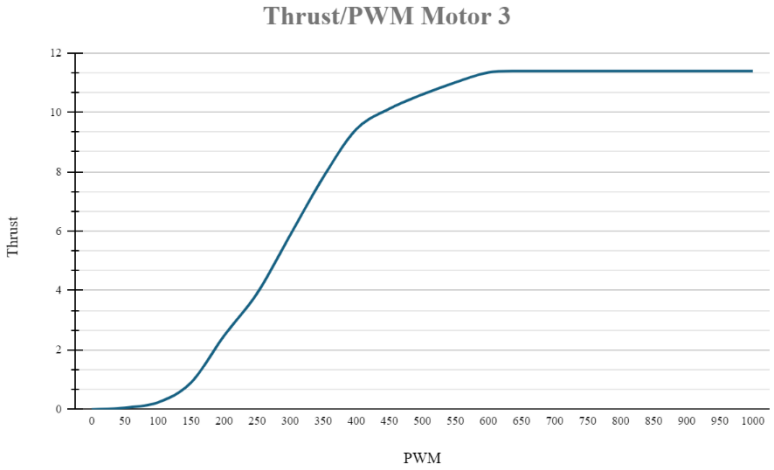


Figure 12: Thrust/PWM Curve Motor 3

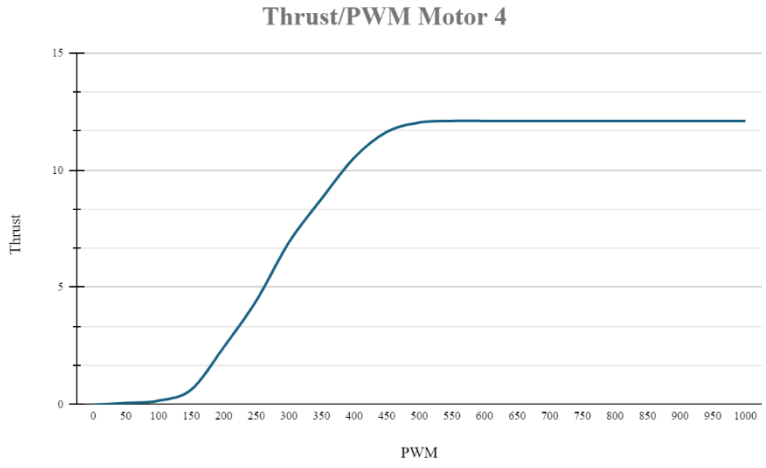


Figure 13: Thrust/PWM Curve Motor 4

4) Model Identification

As first step to characterize the model we considered an orientation like in the figure below in which we see the drone from the lateral view acting a rotation around roll angle in "+" configuration:

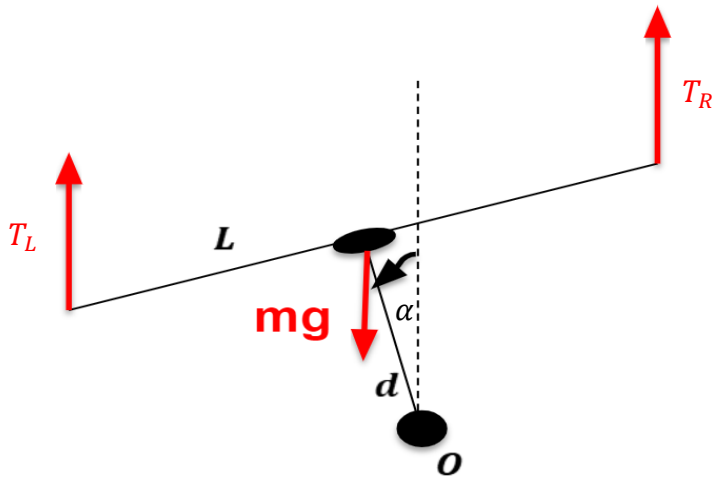


Figure 14: Drone Dynamic Model

The physical model that we adopted for the identification is described by the equations:

$$J_x^+ \ddot{\varphi}^+ = mgdsin(\varphi^+) + T_1\big(Lcos(\varphi^+) - dsin(\varphi^+)\big) - T_2(Lcos(\varphi^+) + dsin(\varphi^+))$$

$$J_y^+ \ddot{\theta}^+ = mgdsin(\theta^+) - T_3\big(Lcos(\theta^+) - dsin(\theta^+)\big) + T_4(Lcos(\theta^+) + dsin(\theta^+))$$

- L is the distance between one motor and the center of the drone, we consider the same L for every motor
- d is the height of the center of gravity of the drone with respect to the hinge
- m is the mass of the quadcopter
- J_x^+ is the moment of inertia of the drone with respect to the x axis considering the "+" configuration (from now on $J_x^+ = J_x$)
- J_y^+ is the moment of inertia of the drone with respect to the y axis considering the "+" configuration (from now on $J_y^+ = J_y$)
- φ^+ is the roll angle considering the "+" configuration
- θ^+ is the pitch angle considering the "+" configuration

We measured d and m with the appropriate tools (balance and meter) and we found the following values:

- $d = 0{,}10\ m$
- $m = 1{,}08\ kg$

In order to get the remaining physical model parameters, and also others that are used later on, we looked at the drone datasheet [2] and we found that:

- $L_{arm} = L = 0.25\ m$
- $m_{body} = 0.610\ kg$
- $m_{motor} = 0.1175\ kg$
- $L_{body} = 0.144\ m$

4.1 Moments of Inertia Identification

For what concerns the moments of inertia around the x and y axes, we fixed two springs with stiffness k, which were previously characterized, in correspondence with the motors of the arm under consideration:

Figure 15: Moments of Inertia long x and y axes Experiment

Considering the x axis of the "+" configuration, letting the system oscillate around the x axis ($\theta = \varphi^{+}$), we measured using the accelerometer the natural frequency of the body. Here θ stands for a generic angle, does not represent the pitch angle. We used the dynamic equation of the equilibrium of the momentum with respect to the pole O:

$$\sum M_O = J_x\ddot{\theta}$$

$$\Downarrow$$

$$J_x\ddot{\theta} = mgdsin(\theta) - k\Delta x\big(Lcos(\theta) + dsin(\theta)\big) - k\Delta x(Lcos(\theta) - dsin(\theta))$$

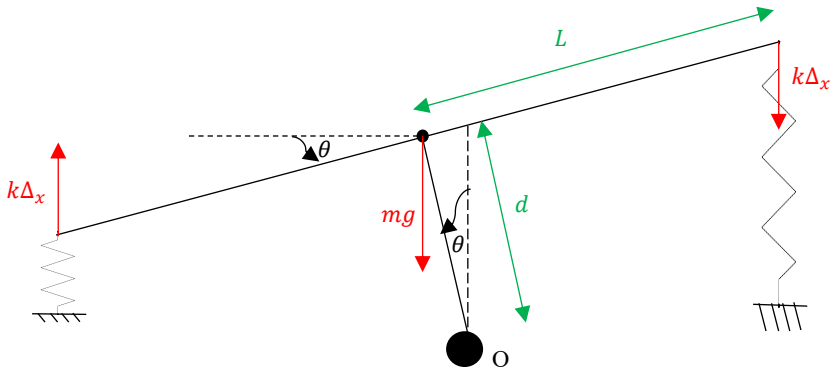


Figure 16: Inertia on x Dynamics

Considering small oscillations of the system and so $\theta \cong 0$, we can rewrite the equation above in the following form:

$$J_x\ddot{\theta} = mgd\theta - k\Delta x(L + d\theta) - k\Delta x(L - d\theta) \text{ (notice that } \Delta x = Lsin(\theta) \cong L\theta)$$

We can rewrite the equation as:

$$J_x\ddot{\theta} = mgd\theta - 2kL^2\theta$$

Now the solution of this differential equation should have the form: $\theta = \theta_0 e^{\lambda t}$ and so we can demonstrate that:

$$(J_x\lambda^2 - mgd + 2kL^2)\theta_0 = 0$$

Since we considered our system as undamped, where $2kL^2 > mgd$, we obtained 2 purely imaginary solutions: $\lambda_{1,2} = \pm i\omega$ where ω is the natural frequency of the considered body. we finally found that:

$$J_x = \frac{2kL^2 - mgd}{\omega_n}$$

For the y axis of the drone, we can consider the same moment of inertia because we considered the drone symmetric respect to its origin.

$$J_x = J_y$$

Then we performed a different experiment to measure the value of the natural frequency of the system, but along the z axis, useful to compute the moment of inertia along the z axis. As to fix the x and y axes, we tied a thread (green ones) on 2 (out of the 4) motors, which were opposite, and we attached the other end of the threads to the wall which was parallel to the considered arm of the chosen motor.
We also connected 2 springs (**black** ones) for each of the chosen opposite motor, following the same direction of the thread, but one following the thread upwards and the other downwards.
To support the drone, hence avoiding the drone to fall in the x or y direction, we used other 2 threads (red ones) which were connecting the remaining 2 motors to the wall.

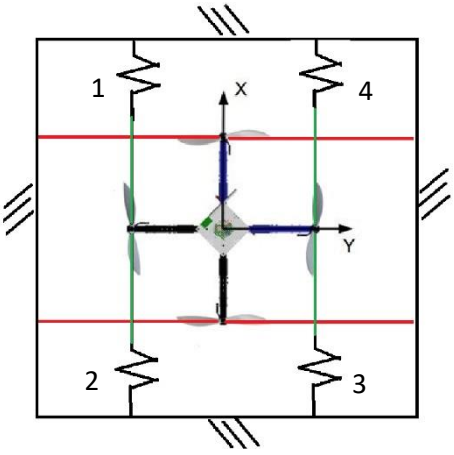


Figure 17: Inertia on z Experiment

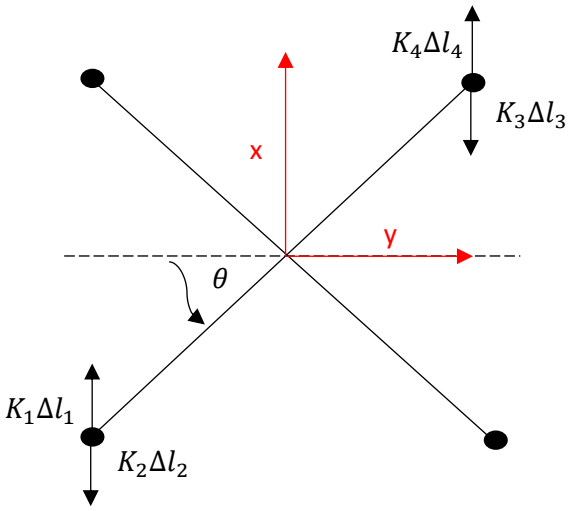
Then, we let the drone oscillate around the z axis and we measured the natural frequency.
In order to compute the moment of inertia we considered that:

$$J_z^+ \ddot{\psi}^+ = J_z^x \ddot{\psi}^x = J_z \ddot{\theta} = \sum M_O$$

Again, here θ stands for a generic angle, does not represent the pitch angle.
Hence, we got that (considering small angles: $Lcos(\theta) \cong L$):

$$J_z \ddot{\theta} = [(K_2 + K_4)(-\Delta_x) - (K_1 + K_3)(\Delta_x)]L$$

where:



$$\Delta l_1 = \Delta l_3 = +\Delta_x$$
$$\Delta l_2 = \Delta l_4 = -\Delta_x$$

Figure 18: Inertia on z Dynamics

Considering that $\Delta_x = Lsin(\theta) \cong L\theta$, we have that:

$$\ddot{\theta} = -\frac{[(K_2 + K_4) + (K_1 + K_3)]L^2\theta}{J_z} \quad becomes \left(\theta = \theta_0 e^{\lambda t}\right): \lambda^2 = -\frac{[(K_2 + K_4) + (K_1 + K_3)]L^2}{J_z}$$

Since $\lambda_{1/2} = \pm i\omega_n$ because we considered our system as undamped, we have that the natural frequency is:

$$\omega_n = \sqrt{\frac{[(K_2 + K_4) + (K_1 + K_3)]L^2}{J_z}}$$

In conclusion:

$$J_z = \frac{[(K_2 + K_4) + (K_1 + K_3)]L^2}{{\omega_n}^2}$$

In order to confirm the measurements for the moments of inertia we adopted some geometrical considerations [3].
We considered the body of the drone as a sphere, with radius equal to the half of the length of the body of the drone:

$$r = \frac{L_{body}}{2}$$

Then we computed the moments of inertia along the x and y axes (assumed to be the same), assuming that the main mass (m_{body}) is modeled like a sphere, while the motors are modeled like point masses. Therefore, the considered inertia was computed by using the Huygens-Steiner theorem as follows:

$$J_x = J_y = \frac{2}{5} m_{body} r^2 + 2{L_{arm}}^2 m_{motor}$$

Then we also computed, using the same geometrical consideration, the moment of inertia along the z axis, which has been obtained using this formula:

$$J_z = \frac{2}{5} m_{body} r^2 + 4{L_{arm}}^2 m_{motor}$$

We noticed that there was a difference between the experimental values and the geometrical ones, hence we used the experimental ones, which performed better ($J_x = J_y = 0.055, J_z = 0.106$).

4.2 Thrust and Torque Parameters Identification

Other relevant quantities that we estimated were the Thrust coefficient C_t and the Torque coefficient C_m.
These coefficients are important for the following relationships:

$$T = C_t \omega^2$$

$$M = C_m \omega^2$$

Hence those coefficients are fundamental to obtain the value of the angular velocity considering the value of the thrust and torque for each motor.
We assumed that the thrust and torque coefficients are the same for each propeller.
As to obtain those coefficients we looked at the motor datasheet, where it is shown the value of the angular velocity with respect to the values of the thrust and torque:

Type	Propeller	Throttle	Voltage (V)	Thrust (g)	Torque (N*m)	Current (A)	RPM	Power (W)	Efficiency (g/W)	Operating Temperature (℃)
AIR2216II-KV920	T1045II	30%	16	210	0.03	1.44	4042	23	9.12	80℃
		35%	16	259	0.04	1.87	4469	30	8.67	
		40%	16	309	0.05	2.29	4855	37	8.45	
		45%	16	373	0.05	2.86	5301	46	8.15	
		50%	16	447	0.06	3.60	5780	58	7.76	
		55%	16	536	0.08	4.53	6298	72	7.39	
		60%	16	628	0.09	5.61	6800	90	7.01	
		65%	16	729	0.10	6.78	7281	108	6.73	
		70%	16	814	0.11	7.92	7679	126	6.44	
		75%	16	906	0.12	9.20	8096	147	6.18	
		80%	16	993	0.14	10.59	8468	169	5.88	
		85%	16	1087	0.15	12.11	8867	193	5.65	
		90%	16	1191	0.16	13.81	9257	219	5.43	
		95%	16	1289	0.18	15.68	9675	249	5.18	
		100%	16	1332	0.18	16.37	9857	260	5.13	

Figure 19: Motor Datasheet [7]

As first thing we converted the RPM value of the angular velocity in rad/s using this formula:

$$\frac{rad}{s} = \frac{RPM}{60} 2\pi$$

Then we created an additive column where we had the value of ω^2 for each row.

Afterwards we converted the thrust force from g to N, using the following formula:

$$1\ g = 0.00981\ N$$

Starting from the estimation of C_t, we used Linear Regression to understand which is the line that approximates all our Thrust/ω^2 points, obtaining this result:

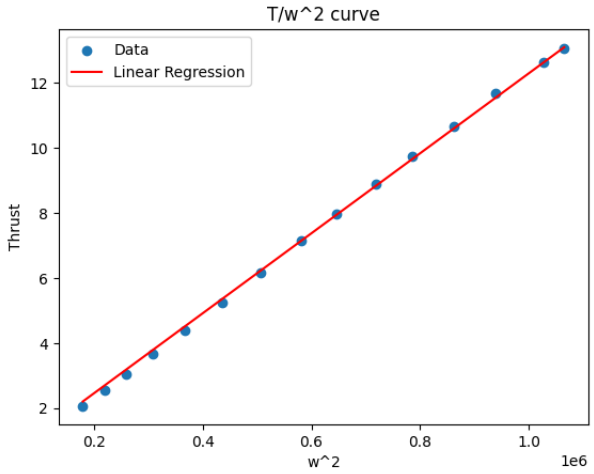


Figure 20: Thrust Coefficient Estimation

We got that all the obtained line is $y = 0.0000122949$x, hence C_t is equal to the slope of the obtained line.

We used the same approach for C_m, but considering all the Torque/ω^2 points, obtaining this result:

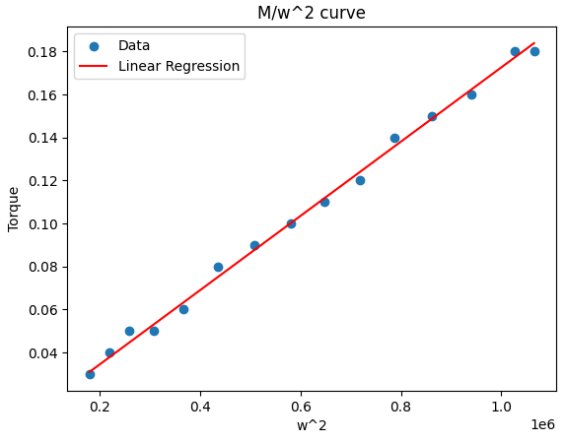


Figure 21: Torque Coefficient Estimation

We got that all the obtained line is $y = 0.000000172576$x, hence C_m is equal to the slope of the obtained line.

5) State Estimation

The states needed to make a proper control design for the attitude control were of course both the angular positions and the angular rates.

The first control architecture was based on the only control of roll and pitch angles.

The sensors available on our quadcopter are IMU sensors, i.e. an accelerometer and a gyroscope. So, we can use both these sensors to get an estimation about the other directly unmeasurable states needed for the control design.

We have two ways to get an estimation about both the pitch and roll angles:

- In the first case, we can consider the measures given by the accelerometer. Indeed, by considering that our quadcopter is constrained to a base, the only linear acceleration measured is the gravitational one.
  Because of this, we can consider the projection of the gravitational acceleration along the three axis and, by doing so, we get the relationship between the angles and the acceleration components of each axis.
  By calling $a_m = [a_{mx}; a_{my}; a_{mz}]$ the accelerometer measures, we have the following:

$$\varphi_m^x = \arcsin\left(\frac{a_{mx}}{g}\right)$$
$$\theta_m^x = -\arcsin\left(\frac{a_{my}}{g\cos(\varphi_m^x)}\right).$$

- The second way consists in a simple integration of the angular rates under the assumption of small angles. If we call $\omega_m = [\omega_{xm};\ \omega_{ym};\ \omega_{zm}]$ the gyroscope measures, the angles are given by:

$$\varphi_m^x = \frac{1}{s}\omega_{xm}$$
$$\theta_m^x = \frac{1}{s}\omega_{ym}$$

The first mentioned method is affected by a high frequency noise due to accelerometer measures, whereas the second one is affected by a low frequency drift, due to the integration of the gyroscope noise.

5.1 Filters

Instead of direct estimation of the needed angles we also considered the usage of filters.

5.1.1 Low Pass Filter (LPF)

As a first estimation of the angles, we decided to use only the gyroscope measures. Indeed, we set up an estimation structure consisting of a simple low pass filter to remove the gyroscope noise and a successive integrator to get the estimation of the angles.

$$LPF(s) = \frac{1}{\tau s + 1} \qquad and \qquad \frac{1}{s}$$

This solution provided us with a good estimation but not that reliable, considering both the drift and the loss of information due to integration and usage of a low pass filter.

5.1.2 Linear Complementary Filter

As second step of our estimation phase we chose to combine both the good qualities of the methods mentioned in the very beginning. In particular, we could combine the information at low frequency by using the accelerometer measurements and the information at high frequencies by using the gyroscope measurements.

To do so, we implemented the well-known linear complementary filter, that consists in a linear combination between two complementary filters, i.e. a low pass filter and a high pass filter. The LPF is used to remove the high frequency noise of the accelerometer measures whereas, the HPF is used to remove the low frequency drift due to integration of the gyroscope measures.

By implementing it, we have:

$$\varphi^x(s) = \frac{1}{\tau s + 1}\arcsin\left(\frac{a_{mx}}{g}\right) + \frac{\tau s}{\tau s + 1}\left(\frac{1}{s}\omega_{xm}\right)$$

$$\theta^x(s) = -\frac{1}{\tau s + 1}\arcsin\left(\frac{a_{my}}{g\cos\left(\theta^x_{x\,meas}\right)}\right) + \frac{\tau s}{\tau s + 1}\left(\frac{1}{s}\omega_{ym}\right)$$

Whereas, for the angular rates estimation a simple low pass filter has been applied to the gyroscope measurements.

5.1.3 Extended Kalman Filter 1

As a further step towards optimal estimation, we have then decided to develop a Kalman filter, more precisely, its non-linear version: the EKF.
The EKF mainly differs from its linear version in approximating the non-linearity of the equation through a first order truncation of the Taylor series.
To implement an EKF we have two main steps:

- 1st step prediction:

  As well known, the first step of the EKF consists in predicting the state at the new time instant by knowing both the dynamics of the states and their value at present time.
  In our case, since we have a non-linear system, we had to use a non-linear model for prediction as follows:

  State prediction:

  $$x_{k+1|k} = f\left(x_{k|k}\right)$$

  Covariance prediction:

  $$P_{k+1|k} = FP_{k|k}F^T + Q$$

  Where:

  $$F = \frac{\partial f}{\partial x}|_{x_{k|k}}$$

  Being x the state, f the non-linear dynamics, P the estimation error covariance, Q the process noise covariance and F the non-linear dynamics Jacobian.

- 2nd step correction:

  In this second step instead, the measurements of the sensors are used to correct the prediction of the states.

The correction happens as follows:
Innovation:

$$y = z - h(x_{k+1|k})$$

Gain computation:

$$S = HP_{k+1|k}H^T + R$$
$$K = PH^T S^{-1}$$

State correction:

$$x_{k+1|k+1} = x_{k+1|k} + Ky$$

Covariance correction:

$$P_{k+1|k+1} = (I - KH)P_{k+1|k}$$

Where:

$$H = \frac{\partial h}{\partial x}$$

Being z the measurement, R the measurement noise covariance, h the measurement equation and H the Jacobian of the measurement equation.

In our specific case, we implemented as first filter, the one filter that used our simplest non-linear model *(Pole Placement)*.

As a first choice, we chose to use as measurements just the angular rate ones. This solution implied a drift over time of the angular position and high dependence of the position estimation over the initial conditions.
This led us to also consider the angular position measurements by using the accelerometer.

Once the model had been chosen, we had to tune the filter. In particular, we had to choose the values of Q and R, that represent, respectively, our uncertainty on the model and on the measurements.
For the choice of Q values, we took into consideration the lack of precision of our model, due to its simplicity. We indeed chose the following values:

$$Q = \begin{bmatrix} 1e-7 & 0 & 0 & 0 \\ 0 & 1e-4 & 0 & 0 \\ 0 & 0 & 1e-10 & 0 \\ 0 & 0 & 0 & 1e-7 \end{bmatrix}$$

As we can see from our choice, we simply tuned Q considering the uncertainty on the velocity dynamics (elements $Q(2,2)$ and $Q(4,4)$) while we have a lower uncertainty on the position due to very simple model we had, which is a simple integrator (elements $Q(1,1)$ and $Q(3,3)$).
For the tuning of the measurement covariance R, instead, we have estimated the variance on both the gyroscope and the accelerometer and used them as values on R, i.e.:

$$R = \begin{bmatrix} 1.9365481e-8 & 0 & 0 & 0 \\ 0 & 2.7827391e-6 & 0 & 0 \\ 0 & 0 & 1.8091231e-8 & 0 \\ 0 & 0 & 0 & 2.7182404e-6 \end{bmatrix}$$

This filter has been used for the simplest control structures with good performances.

5.1.4 Extended Kalman Filter 2

The last filter has been developed for more complex control structures. For this scope, we have indeed used a different, more complex, model *(Feedback Linearization)*.

In this case, Q and R were augmented due to the higher number of states considered and, once again, R was tuned considering the estimated variance values whereas Q was tuned considering the, lower, uncertainty of the model, while the first one depends on the model, the second one depends only on the sensor used. For this reason, we used the following Q:

$$Q = \begin{bmatrix} 5e-16 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e-6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5e-16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e-6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e-13 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5e-7 \end{bmatrix}$$

While R became:

$$R = \begin{bmatrix} 1.9365e-8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2.7827e-6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.8091e-8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.7182e-6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.9766e-6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5.5874e-6 \end{bmatrix}$$

For what regards initial conditions instead, we considered the zero for the estimated state, while we considered a possible error on it by taking into account:

$$P = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

The same was done for the first EKF by considering appropriate dimensions for both the state and the covariance matrix.

5.1.4 EKF Results

The obtained results are the following:

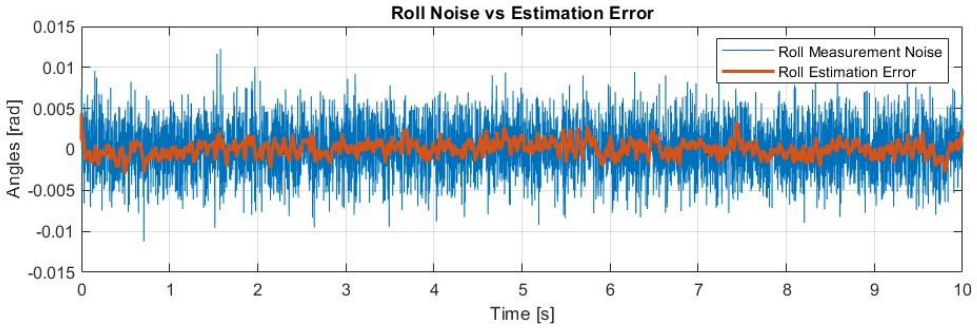


Figure 22: Roll Measurement Noise vs Estimation Error

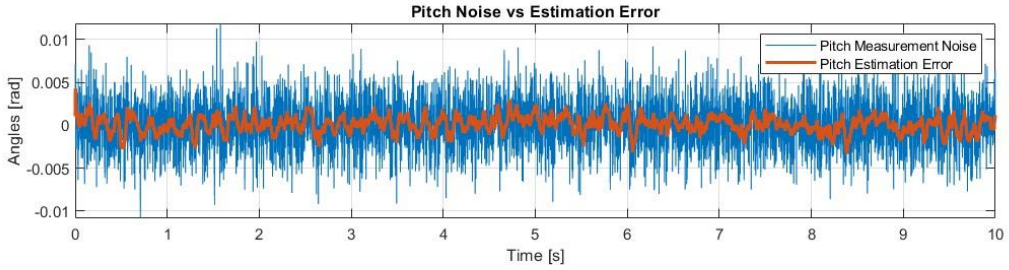


Figure 23: Pitch Measurement Noise vs Estimation Error

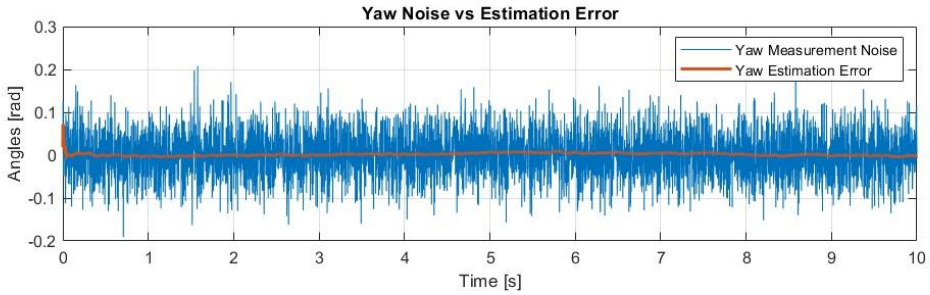


Figure 24: Yaw Measurement Noise vs Estimation Error

These figures show us a good estimation of the states and a good filtering of the noise.

6) Drone Controllers

After all the model identification part we were able to apply some controllers on our plant.

6.1 Pole Placement

The first controller that we applied to our drone was the Pole Placement control. It is a method used in control theory to design a controller for a system by placing the poles of the system's transfer function at desired locations in the complex plane. It is a regulation control.
As first thing we had to find the model on which computing the state feedback gain to place the poles where desired. We decided to design this model in the "+" configuration.
We considered 4 state equations, involving:

- The roll angle in the "+" configuration $\varphi^+ = \varphi = x_1$
- The roll angular velocity in the "+" configuration $\dot{\varphi}^+ = \dot{\varphi} = x_2$
- The pitch angle in the "+" configuration $\theta^+ = \theta = x_3$
- The pitch angular velocity in the "+" configuration $\dot{\theta}^+ = \dot{\theta} = x_4$

We considered 4 inputs:

- Thrust of motor 1 $T_1 = u_1$
- Thrust of motor 2 $T_2 = u_2$
- Thrust of motor 3 $T_3 = u_3$
- Thrust of motor 4 $T_4 = u_4$

Then we considered 2 outputs:

- The roll angular velocity in the "x" configuration $\dot{\varphi}^x = y_1$
- The pitch angular velocity in the "x" configuration $\theta^x = y_2$

As already said in the model identification part we considered the following equations:

$$J_x^+ \ddot{\varphi}^+ = mgdsin(\varphi^+) + T_1\big(Lcos(\varphi^+) - dsin(\varphi^+)\big) - T_2(Lcos(\varphi^+) + dsin(\varphi^+))$$

$$J_y^+ \ddot{\theta}^+ = mgdsin(\theta^+) - T_3\big(Lcos(\theta^+) - dsin(\theta^+)\big) + T_4(Lcos(\theta^+) + dsin(\theta^+))$$

Those can be rewritten as:

$$J_x^+ \dot{x_2} = mgdsin(x_1) + u_1\big(Lcos(x_1) - dsin(x_1)\big) - u_2(Lcos(x_1) + dsin(x_1))$$

$$J_y^+ \dot{x_4} = mgdsin(x_3) - u_3\big(Lcos(x_3) - dsin(x_3)\big) + u_4(Lcos(x_3) + dsin(x_3))$$

Then, since the PP is a linear control, we must linearize our model, which is a nonlinear one, around the equilibrium $\bar{x} = \overline{x_1} = \overline{x_2} = \overline{x_3} = \overline{x_4} = 0$ *and* $\bar{u} = \overline{u_1} = \overline{u_2} = \overline{u_3} = \overline{u_4} = \bar{U}$. We chose $\bar{U} = 3$ since we wanted the motors to stay in the linear section of the Thrust/PWM curve.

The output equations are already linear, hence do not need any linearization:

$$y_1 = \frac{\sqrt{2}}{2}(-x_2 + x_4)$$
$$y_2 = \frac{\sqrt{2}}{2}(-x_2 - x_4)$$

They come from the usage of the rotation matrix R_+^x on the angular velocities.

The full nonlinear model is:

$$\dot{x_1} = x_2$$
$$\dot{x_2} = \frac{mgd}{J_x^+} sin(x_1) + \frac{u_1}{J_x^+}\big(Lcos(x_1) - dsin(x_1)\big) - \frac{u_2}{J_x^+}(Lcos(x_1) + dsin(x_1))$$
$$\dot{x_3} = x_4$$
$$\dot{x_4} = \frac{mgd}{J_y^+} sin(x_3) - \frac{u_3}{J_y^+}\big(Lcos(x_3) - dsin(x_3)\big) + \frac{u_4}{J_y^+}(Lcos(x_3) + dsin(x_3))$$
$$y_1 = \frac{\sqrt{2}}{2}(-x_2 + x_4)$$
$$y_2 = \frac{\sqrt{2}}{2}(-x_2 - x_4)$$

Therefore, the linearized model is:

$$\partial\dot{x}_1 = \partial x_2$$
$$\partial\dot{x}_2 = \frac{(mgd - 2\bar{U}d)}{J_x^+}\partial x_1 + \frac{L}{J_x^+}\partial u_1 - \frac{L}{J_x^+}\partial u_2$$
$$\partial\dot{x}_3 = \partial x_4$$
$$\partial\dot{x}_4 = \frac{(mgd - 2\bar{U}d)}{J_y^+}\partial x_3 - \frac{L}{J_y^+}\partial u_3 + \frac{L}{J_y^+}\partial u_4$$
$$\partial y_1 = \frac{\sqrt{2}}{2}(-\partial x_2 + \partial x_4)$$
$$\partial y_2 = \frac{\sqrt{2}}{2}(-\partial x_2 - \partial x_4)$$

Hence the linearized system matrices are:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{mgd - 2\bar{U}d}{J_x^+} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{mgd - 2\bar{U}d}{J_y^+} & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{L}{J_x^+} & \frac{-L}{J_x^+} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-L}{J_y^+} & \frac{L}{J_y^+} \end{bmatrix}$$

$$C = \frac{\sqrt{2}}{2}\begin{bmatrix} 0 & -1 & 0 & 1 \\ 0 & -1 & 0 & -1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Then, since the model must be discretized in order to use it in our overall scheme, we just used the Matlab function $c2d$:

$$dsys = c2d(ss(A, B, C, D), Ts, 'zoh');$$

The sampling time Ts was chosen to be the same of the sampling time of our sensors.
Then we extracted the F, G, H and I matrices by using the Matlab function $ssdata$.

We decided to place all the 4 poles in 0.96, which is a position that is a compromise between reducing the control effort and stability, being robust to high frequency disturbances. We found this position through a trial-and-error procedure, noticing that faster poles caused the control to be too reactive to disturbances, while slower ones caused the control to be too slow, and not to reach the desired equilibrium.

In conclusion we used the Matlab function place to get our desired state feedback gain:

$$K_{PP} = place(F, G, poles);$$

Talking about the Simulink scheme, starting from the measurement part, hence the state feedback, we got the measurements of $\dot{\varphi}^x$ and $\dot{\theta}^x$ from the gyroscope and we got the measurements of φ^x and θ^x from the vehicle attitude block + quat2eul block, in fact the vehicle attitude block returns us the quaternion values for the drone, which are converted in the Euler angles in the "x" configuration through the quat2eul function block.
Then, to obtain the values of the angular velocities and angles in the "+" configuration, which represent our states, we used a gain block with the rotation matrix $R_x^+ = R_+^{x^T}$ inside to convert them from the "x" to the "+".
The rotation matrices were applied separately for the angular velocities and for the angles.
With those block operations we obtained our 4 needed state measurements, which were given as input to another gain block, which represents the multiplication by the obtained K_{PP} matrix.
Still, it was impossible to feed those control inputs to the plant, but they had to be correctly mixed and converted.
In order to mix them, we considered that the control inputs represent the thrust values requested for each motor, hence they have to be positive: $u_i \geq 0 \; for \; i = 1,2,3,4$.

For this reason, we inserted a mixing function block that does this job.
This mixing function considers the 4 motors as grouped in 2 couples, motor 1 with motor 2, motor 3 with motor 4. This is because the coupled motors are opposite and represent an axis of the quadcopter. It was programmed to avoid having negative inputs, in fact if one of the 4 inputs is negative then we have that the negative input goes to 0 by summing to it the absolute value of itself, but summing also to the opposite motor, hence the other in the couple, the same absolute value of the negative input. As an example, if $u_1 = -3$, then $u_1 = u_1 + |u_1| = 0$ and $u_2 = u_2 + |u_1|$.

```
function [y1,y2,y3,y4] = mix(u1,u2,u3,u4)

if u1 < 0
    u2 = u2 + abs(u1);
    u1 = u1+abs(u1);
end
if u2 < 0
    u2 = u2 + abs(u2);
    u1 = u1 + abs(u2);
end
if u3 < 0
    u3 = u3 + abs(u3);
    u4 = u4 + abs(u3);
end
if u4 < 0
    u4 = u4 + abs(u4);
    u3 = u3 + abs(u4);
end
 y1 = u1;
 y2 = u2;
 y3 = u3;
 y4 = u4;
```

Figure 25: Mixing Function PP

After the mixing function we added a constant term, which represents $\bar{U}$, which we chose as 3. This was done in order to obtain u from ∂u, in fact: $u = \partial u + \bar{U}$.
In conclusion we converted the mixed thrust results in PWM values for each motor by using the previously mentioned look-up tables.
After each table we put a saturation block to limit the PWM value between 0 and 1000, for then summing up to each PWM input the value 1000, to shift the interval [0:1000] to [1000:2000], which is the correct input interval for our plant inputs.

6.1.1 PP Performances

As far as the performances are concerned, we used as proportional gain the following:

$$K_{PP} = \begin{bmatrix} 54.5329309 & 5.4477749 & 0.6949964 & 0.0350342 \\ -54.5329309 & -5.4477749 & -0.6949964 & -0.0350342 \\ -0.5684230 & -0.0286606 & -54.0009370 & -5.4209439 \\ 0.5684230 & 0.0286606 & 54.0009370 & 5.4209439 \end{bmatrix}$$

Considering that the equilibrium is in 0, we have following time histories:

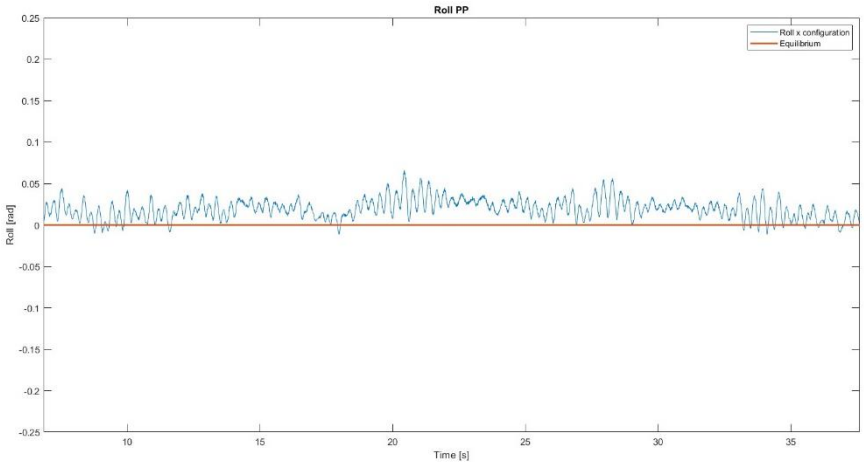


Figure 26: Roll Time History PP

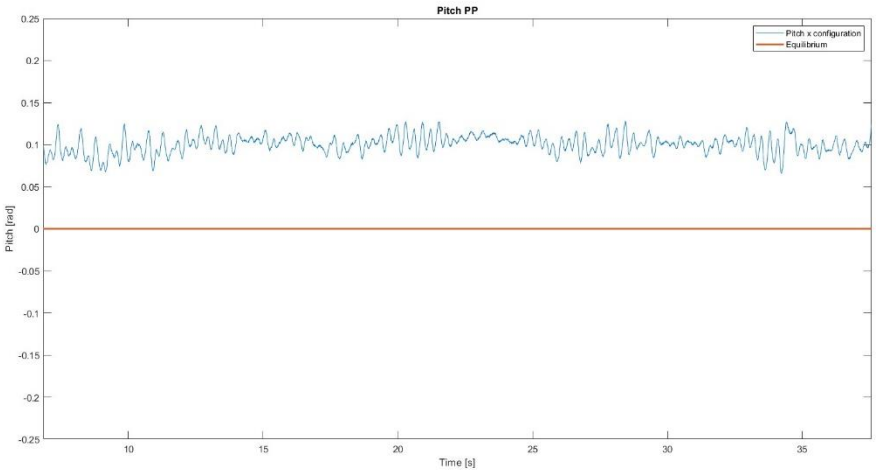
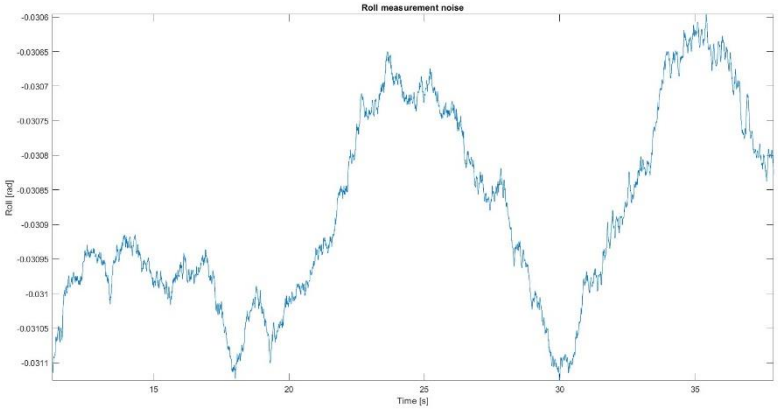


Figure 27: Pitch Time History PP

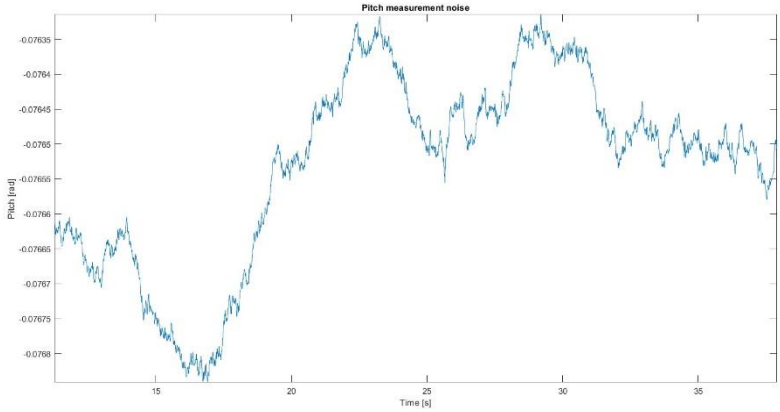
As we can see, the pitch time history is not perfectly converging to the equilibrium. It is happening since the model that we considered, which is a linearized one, does not perfectly describe the real system, which is nonlinear.
Looking at the time histories, we can notice that there are some oscillations. To analyze the origin of the oscillations we measured the pitch and roll without applying the control to get the noise characteristics (i.e. standard deviation and variance).
Indeed:



$$\sigma^2 = 1.9365482e - 08$$

$$\sigma = 0.00013915991$$

Figure 28: Roll measurement noise



$$\sigma^2 = 1.8091232e - 08$$

$$\sigma = 0.00013450364$$

Figure 29: Pitch Measurement Noise

From a first analysis, we concluded that, since the measurement noise is an additive one, it is feedbacked together with the measurements and it is multiplied by the gain. This introduces errors in the control action, which is reacting to the noise, generating the oscillations that we are dealing with. Other causes of the oscillations are the not perfect identification of the system parameters and the approximation in the design of the model. In particular, talking about the parameters mismatch, the inertia could influence the most the closed loop response, while for the model approximation, we neglected some effects given by the presence of the cables connected to the drone, and by the structural friction introduced by the spherical hinge. We also neglected some other complex dynamics like the gyroscopic torques and aerodynamic effects. Despite this, the considered oscillations are in the order of 0.025 rad, which is equal to 1.43 degrees, hence they are quite low valued and do not compromise the stability of the system.

6.2 LQ Control

The second control we used is the LQR, which is a method in control theory for designing a controller that optimizes the performance of a system by minimizing a cost function. The cost function typically includes terms for the state variables and the control inputs, weighted by matrices to balance performance and control effort. The goal of LQR is to find the optimal state feedback gains that result in the best trade-off between minimizing deviations from desired states and minimizing the control effort, leading to an efficient and stable system response.
The considered model was the same used in the Pole Placement, hence the only difference between the 2 control approaches stands in the control gain, which we called K_{LQ}.
As to obtain the value for the gain matrix K_{LQ} we needed to define the R and Q matrices, which are respectively the weight matrices for the inputs and for the states.
In order to find the optimal values for those matrices we wrote a script that allowed us to find the best R and Q combination, by making several assessments on the eigenvalues of the obtained closed loop system.
We firstly set the R matrix as:

$$R = eye(4);$$

We kept it till the end, since the relevant information for the control is the ratio between the R and Q matrices.
We initialized the Q matrix as:

$$Q = 10000 * eye(4);$$

Afterwards we computed the gain K_{LQ} by using the $lqrd$ function:

$$K_{LQ} = lqrd(A, B, Q, R, Ts)$$

This function directly discretizes the continuous state space system and computes the discrete time control gain. With that gain we computed the closed loop eigenvalues:

$$eigenvalues = eig\big(F - G * K_{LQ}\big);$$

Then we initialized another matrix which is the Q_temp, which will be used for comparison purposes.

Successively we iterated several times to randomly set the diagonal values of Q_temp between 0 and 1000000, for then computing the temporary gain K_{LQ}_temp and its associated temporary eigenvalues $eigenvalues\_temp$.

Subsequently we assessed whether the eigenvalues associated with the angles were between the values 0.9 and 0.98, which is fundamental to achieve a tradeoff between performance and robustness. If this condition held then we set:

$$Q = Q\_temp;$$

In conclusion we tested the best Q obtained by using the above script on the physical model and we kept the best performing one. Talking about the Simulink scheme, due to the state feedback nature of the considered control, we adopted the same scheme described in the Pole Placement paragraph, by only changing the control gain to K_{LQ}.

6.2.1 LQR Performances

As far as the performances are concerned, we used as R and Q matrices the following:

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} 632549.7 & 0 & 0 & 0 \\ 0 & 11393.1 & 0 & 0 \\ 0 & 0 & 778447.6 & 0 \\ 0 & 0 & 0 & 12170.8 \end{bmatrix}$$

Obtaining the following gain matrix:

$$K_{LQ} = \begin{bmatrix} 341.8209563 & 46.7300842 & -1.1386502e-12 & -1.3130135e-14 \\ -341.8209563 & -46.7300842 & -3.5784784e-12 & -7.8227042e-15 \\ 1.4464951e-14 & -1.71882221e-15 & -373.6305159 & -47.6503389 \\ 4.3733912e-13 & 3.0770812e-16 & 373.6305159 & 47.6503389 \end{bmatrix}$$

Considering that the equilibrium is in 0, we have following time histories:

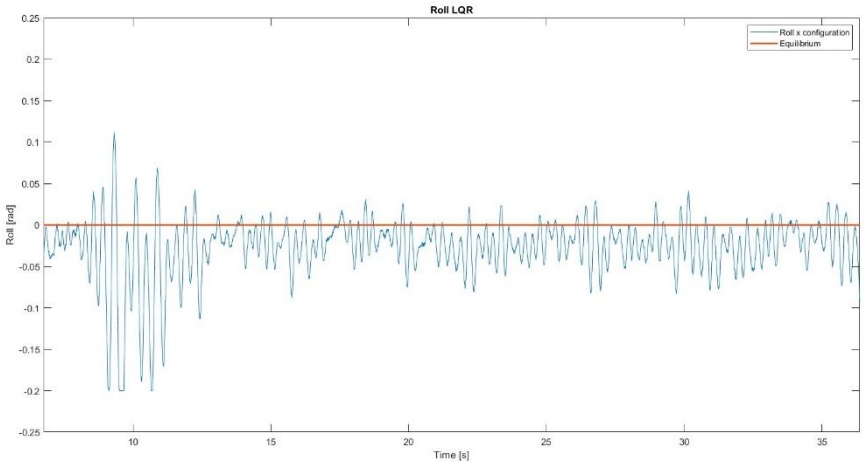


Figure 30: Roll Time History LQR

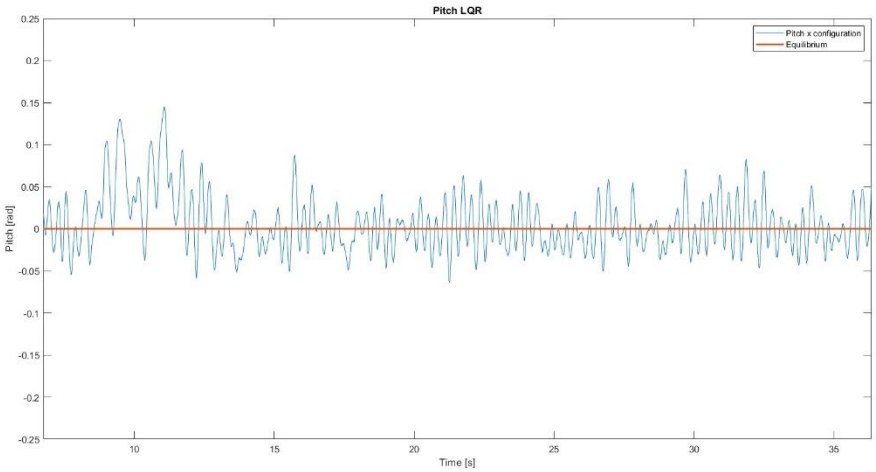


Figure 31: Pitch Time History LQR

Differently from the PP, the LQR approach converges better to the equilibrium, in fact there is no angle that converges to a value which is different from 0. This is due to the higher performance of the chosen algorithm. Still there are some oscillations. The nature of those oscillations is the same explained in the PP section. They are more relevant with respect to the pole placement approach, since the K_{LQ} gain is greater than the pole placement gain, hence the error is more amplified with respect to the PP approach, therefore the controller is more reactive and sensitive to the noise, causing greater oscillations. In conclusion, the greater is the gain, the greater are the oscillations.

6.3 Cascade PID Control

The third control that we implemented is the Cascade PID control, which is a control strategy that uses two or more nested PID (Proportional-Integral-Derivative) controllers arranged in a series configuration. The primary (outer) loop controls the main process variable, while the secondary (inner) loop controls an intermediate variable that influences the primary variable. The inner loop typically provides faster response to disturbances, improving the overall performance and stability of the system. This hierarchical structure allows for better handling of complex processes with multiple dynamic components.

We considered our quadcopter that has four inputs and six outputs and is a second order underactuated system [4], showing the nonlinear characteristics upon flying at high speed, and affected by all kinds of disturbances and parameters change in the process of flight. The idea behind the cascade PID controller consist in designing an outer ring to control the angular position and an inner ring to control the angular velocity.

Before establishing the model for our quadcopter, we noticed that motors 1 and 2 rotate counterclockwise, while motor 3 and 4 rotate clockwise. The relationship between torques on roll, pitch and yaw with thrusts is described by the following formulas:

$$T_{tot} = T_1 + T_2 + T_3 + T_4$$

$$\tau_{roll} = L\frac{\sqrt{2}}{2}(-T_1 + T_2 + T_3 - T_4)$$

$$\tau_{pitch} = L\frac{\sqrt{2}}{2}(T_1 - T_2 + T_3 - T_4)$$

$$\tau_{yaw} = M_1 + M_2 - M_3 - M_4$$

These expressions are due to physical considerations on our drone, in fact, considering the following figure:

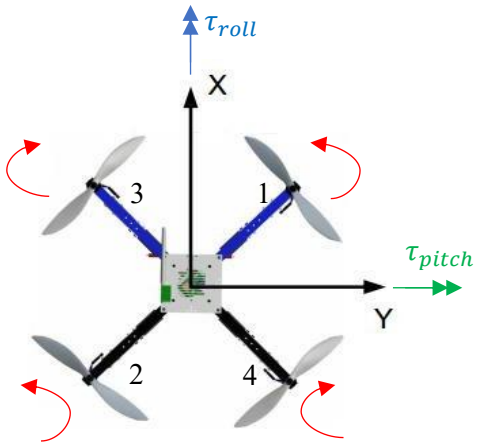


Figure 32: Physical Considerations Torques PID

For the equation of the moments for roll, hence τ_{roll}, motors 3 and 2 generate a positive torque, while motors 1 and 4 a negative one.
For the equation of the moments for pitch, hence τ_{pitch}, motors 1 and 3 generate a positive torque, while motors 2 and 4 a negative one.
For the equation of the moments for yaw, hence τ_{yaw}, motors 1 and 2 generate a positive torque, while motors 3 and 4 a negative one.
Since the Cascade PID control is applied considering the drone in the "x" configuration, we needed to compute the values of the inertia in that configuration.
Due to the fact that we have the values of the inertia in the "+" configuration, to convert them to the "x" one we have to consider the following equation:

$$J^x = R_+^x J^+ R_+^{x^T}$$

After the computations we noticed that $J^x = J^+$. This is true due to the symmetry of the drone.

Afterwards we considered the dynamic equations of the moments and of the forces and we obtained the following formulas:

$$\ddot{x} = \frac{(\sin(\varphi^x)\sin(\psi^x) + \cos(\varphi^x)\sin(\theta^x)\cos(\psi^x))T_{tot}}{m}$$

$$\ddot{y} = \frac{(\cos(\varphi^x)\sin(\theta^x)\sin(\psi^x) - \sin(\varphi^x)\cos(\psi^x))T_{tot}}{m}$$

$$\ddot{z} = \frac{(mg - \cos(\varphi^x)\cos(\theta^x))T_{tot}}{m}$$

$$\ddot{\varphi}^x = \frac{\tau_{roll} + \dot{\theta}^x\dot{\psi}^x(J_y - J_z)}{J_x}$$

$$\ddot{\theta}_y^x = \frac{\tau_{pitch} + \dot{\varphi}^x\dot{\psi}^x(J_z - J_x)}{J_y}$$

$$\ddot{\theta}_z^x = \frac{\tau_{yaw} + \dot{\varphi}^x\dot{\theta}^x(J_x - J_y)}{J_z}$$

The inputs of the above model were the four angular speeds of the propellers Ω_i $i = 1,2,3,4$, in fact:

$$T_i = C_t {\Omega_i}^2 \; for \; i = 1,2,3,4$$
$$M_i = C_m {\Omega_i}^2 \; for \; i = 1,2,3,4$$

Therefore, the equations are dynamically coupled.
To tune more easily the PIDs, it was better to have a G(s) matrix that was diagonal; it meant having the input of the equations decoupled, hence each equation should have only one input, therefore having 6 decoupled SISO systems.
The used mapping is the following:

$$U_1 = T_1 + T_2 + T_3 + T_4$$
$$U_2 = (-T_1 + T_2 + T_3 - T_4)$$
$$U_3 = (T_1 - T_2 + T_3 - T_4)$$
$$U_4 = M_1 + M_2 - M_3 - M_4$$

To get the transfer functions to tune the PIDs, we needed to linearize the equations of the system.
We linearized around the zero-equilibrium point, that is $\bar{x} = \bar{y} = \bar{z} = 0$ and $\overline{\theta_x^x} = \overline{\theta_y^x} = \overline{\theta_z^x} = 0$
The linearized system is, neglecting the $\ddot{x}$ and $\ddot{y}$ equations:

$$\ddot{z} = g - \frac{T_{tot}}{m} = g - \frac{T_1 + T_2 + T_3 + T_4}{m}$$

$$\ddot{\varphi}^x = \frac{\tau_{roll}}{J_x} = \frac{L * \frac{\sqrt{2}}{2} * (-T_1 + T_2 + T_3 - T_4)}{J_x}$$

$$\ddot{\theta}^x = \frac{\tau_{pitch}}{J_y} = \frac{L * \frac{\sqrt{2}}{2} * (T_1 - T_2 + T_3 - T_4)}{J_y}$$

$$\ddot{\psi}^x = \frac{\tau_{yaw}}{J_z} = \frac{M_1 + M_2 - M_3 - M_4}{J_z}$$

Substituting with the before mentioned inputs, the linearized system becomes:

$$\ddot{z} = g - \frac{U_1}{m}$$
$$\ddot{\varphi}^x = L * \frac{\sqrt{2}}{2} \frac{U_2}{J_x}$$
$$\ddot{\theta}^x = L * \frac{\sqrt{2}}{2} \frac{U_3}{J_y}$$
$$\ddot{\psi}^x = \frac{U_4}{J_z}$$

We neglected the $\ddot{x}$ and $\ddot{y}$ equations since our drone is not able to move, due to the hinge. Still, we considered the equation of $\ddot{z}$, because it is useful to formalize the decoupled model.
The transfer function of the 4Input-4Output system is the following:

$$G_1(s) = \begin{bmatrix} \frac{1}{ms} & 0 & 0 & 0 \\ 0 & \frac{L * \frac{\sqrt{2}}{2}}{J_x s} & 0 & 0 \\ 0 & 0 & \frac{L * \frac{\sqrt{2}}{2}}{J_y s} & 0 \\ 0 & 0 & 0 & \frac{1}{J_z s} \end{bmatrix}$$

Since we implemented a cascade PID loop for the roll, pitch and yaw angles separately, we considered the following notation: $R_{1_i}(s)$ stands for the inner loop PID regulator for the i_{th} variable ($i = 1$ is for the roll angle, $i = 2$ is for the pitch angle and $i = 3$ is for the yaw angle), $R_{2_i}(s)$ stands for the outer loop PID regulator for the i_{th} variable. We obtained the regulators $R_{1_i}(s)$ by tuning the PID on the above transfer functions $G_{1_i}(s)$, that are the diagonal terms of $G_1(s)$. The inner loop should be way faster than the outer.
For the outer loop the controllers $R_{2_i}(s)$ should be tuned on the closed loop transfer function:

$$\frac{1}{s}\frac{R_{1_i}(s)G_{1_i}(s)}{1 + R_{1_i}(s)G_{1_i}(s)} \cong \frac{1}{s}$$

But because the inner is way faster than the outer, the transfer function is approximately equal to $\frac{1}{s}$.
The block scheme that we have therefore created is, considering one of the three angles i.e. θ^x:

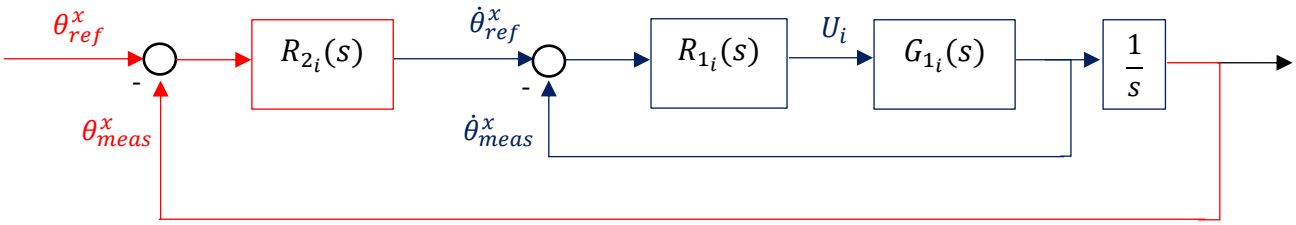


Figure 33: Pitch Cascade PID Control Scheme

For our implementation, we used discrete time PID tuned on the discretized plant transfer functions $G_{1_{discrete}}(z)$.
The outputs of the inner controllers were then combined in a "motor mixer" function that combined them with the appropriate signs, given by the inverse relationship between $U = [U_1\ U_2\ U_3\ U_4]^T$ and the vector $T = [T_1\ T_2\ T_3\ T_4]^T$ with appropriate saturations of the input and the output of the "motor mixer" function.
Talking about the Simulink scheme, we have that the output feedback is the same used in the pole placement (the state feedback part), but with two differences:

- We did not use the rotation matrix to convert the measurements from the "x" configuration to the "+" one, since the control is based on the "x" configuration.
- We used the measurements of the yaw angular velocity too for the control loop.

We did not use the yaw angle measurements since we implemented only the velocity loop for yaw, due to the fact that it is less relevant with respect to the other two loops.
Differently from PP and LQR, since the Cascade PID control allowed us to define the references that our system has to follow, we introduced a scheme that allowed us to manually insert the values of

desired roll and pitch angles in the "+" configuration through the usage of a slider. Those references were then converted in the "x" configuration through the usage of the rotation matrix R_{+}^{x}.

All those measurements and references were put in input to a subsystem block, in which it is contained the whole control logic. In addition to those subsystem inputs, we also inserted another one which is the Total Thrust value, which worked like an offset for the mixing function. It can be seen as the U_1 input. The thrust value worked like an offset and it was set to 0.7, because we wanted the motors to work in the linear zone of their PWM/Thrust curve.

Inside the subsystem block we put the Cascade PID logic, hence we inserted the three control loops for the roll, pitch, and yaw, that were based on the before mentioned block scheme.
The chosen $R_{2_i}(s)$ controller is a P controller, while the $R_{1_i}(s)$ is a PID one, with filtered derivative action. They were tuned properly following the before defined tuning logic.
We also saturated the roll and pitch control actions between -1 and 1, while for the yaw is between -0.5 and 0.5, to give less importance to this loop. Those are values that were obtained by applying a trial-and-error approach.
Afterwards we mixed the three outputs of the control subsystem together with the total thrust value by using a function block to get the correct PWM values.
It basically inverts, as already described, the relationship between T and U, for then properly scaling those values in the PWM interval using a scaling factor and a bias term:

```
function [M1, M2, M3, M4] = motor_mixer(Roll, Pitch, Yaw, Thrust)
%Function description:
%  Control allocation. The quadrotor type is X-configuration,
%  and the airframe is as follows:
%3↓   1↑
 %  \ /
 %  / \
 %2↑   4↓
%Input:
%  Roll, Pitch, Yaw: attitude controller output.
%  Thrust

idle_PWM = 1000;
scale = 1000;


M1 = (Thrust - Roll + Pitch + Yaw) * scale + idle_PWM;
M2 = (Thrust + Roll - Pitch + Yaw) * scale + idle_PWM;
M3 = (Thrust + Roll + Pitch - Yaw) * scale + idle_PWM;
M4 = (Thrust - Roll - Pitch - Yaw) * scale + idle_PWM;
```

Figure 34: Cascade PID Mixing Function

This works because the PIDs are based on the error signal and so they adapt the control actions to minimize the error between the reference and the measured values.

6.3.1 Cascade PID Performances

As far as the performances are concerned, we used the following parameters for our PIDs:

	Roll Inner Loop	Roll Outer Loop	Pitch Inner Loop	Pitch Outer Loop	Yaw Loop
P	3.4962112	6.2358003	3.4962112	6.2358003	0.7497068
I	6.1632505	0	6.1632505	0	1.3896892
D	0.0682371	0	0.0682371	0	−0.0043435
N	35.815475	0	35.815475	0	17.0578345

Considering a 0 reference for both roll and pitch, we obtained the following time histories:

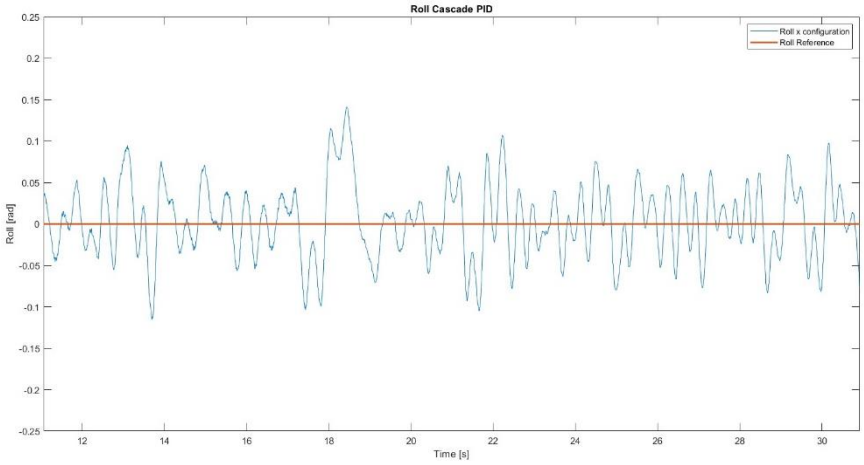


Figure 35: Roll Time History Cascade PID

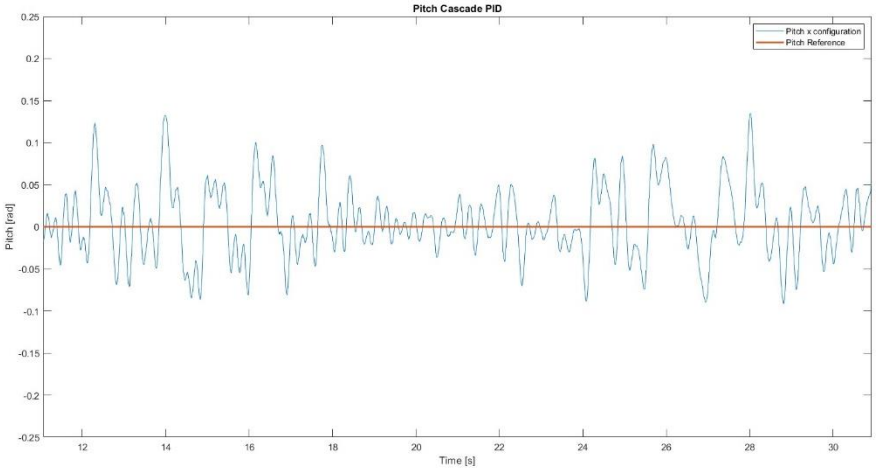


Figure 36: Pitch Time History Cascade PID

As we can see, the Cascade PID oscillations are greater than the ones in the PP control and LQR, but they have a lower frequency. This is due to design choices, in fact the PIDs were tuned in order to be more robust (greater phase margin) and with a higher bandwidth. In addition to the measurement noise consideration explained before, since we considered a high bandwidth, the controller might try to follow the high frequency noise, resulting in a nervous behavior. Nevertheless, the stability of the system is not compromised, and the considered references are well followed.

6.4 Backstepping control

The fourth control we used is Backstepping, which is a recursive design methodology used in control theory to stabilize nonlinear systems. It involves breaking down a complex system into simpler, lower-dimensional subsystems and designing control laws for each subsystem step-by-step. By iteratively applying Lyapunov functions and designing intermediate control inputs, backstepping systematically constructs a stabilizing control law for the overall system. This approach is particularly effective for dealing with nonlinearities and uncertainties in the system dynamics.
This control was entirely based on the dynamic model of our quadcopter.
The Backstepping control is applied considering the drone in the “x” configuration.

The considered dynamic equations for the moments in matrix form are:

$$J^x \underline{\ddot{\alpha}}^x = -\underline{\dot{\alpha}}^x \times \left(J^x \underline{\dot{\alpha}}^x\right) + \underline{\tau}$$

Where:

$$\underline{\alpha}^x = [\varphi^x \quad \theta^x \quad \psi^x]^T$$
$$\underline{\dot{\alpha}}^x = [\dot{\varphi}^x \quad \dot{\theta}^x \quad \dot{\psi}^x]^T$$
$$\underline{\ddot{\alpha}}^x = [\ddot{\varphi}^x \quad \ddot{\theta}^x \quad \ddot{\psi}^x]^T$$
$$J^x = \begin{bmatrix} J_x^x & 0 & 0 \\ 0 & J_y^x & 0 \\ 0 & 0 & J_z^x \end{bmatrix} = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}$$

They were rewritten as [5]:

$$\ddot{\varphi}^x = \dot{\theta}^x \dot{\psi}^x \left(\frac{J_y - J_z}{J_x}\right) + \frac{1}{J_x} U_2$$
$$\ddot{\theta}^x = \dot{\varphi}^x \dot{\psi}^x \left(\frac{J_z - J_x}{J_y}\right) + \frac{1}{J_y} U_3$$
$$\ddot{\psi}^x = \dot{\theta}^x \dot{\varphi}^x \left(\frac{J_x - J_y}{J_z}\right) + \frac{1}{J_z} U_4$$

The considered dynamic equations for the forces are:

$$\ddot{z} = -g + \frac{(cos\varphi^x \, cos\theta^x)}{m} U_1$$
$$\ddot{x} = \frac{(cos\varphi^x sin\theta^x cos\psi^x + sin\varphi^x sin\psi^x)}{m} U_1$$
$$\ddot{y} = \frac{(cos\varphi^x sin\theta^x sin\psi^x - sin\varphi^x cos\psi^x)}{m} U_1$$

Where (like in the Cascade PID approach):

$$U_1 = T_1 + T_2 + T_3 + T_4$$
$$U_2 = \frac{L\sqrt{2}}{2}(-T_1 + T_2 + T_3 - T_4)$$
$$U_3 = \frac{L\sqrt{2}}{2}(T_1 - T_2 + T_3 - T_4)$$
$$U_4 = M_1 + M_2 - M_3 - M_4$$
$$\Omega = \Omega_1 + \Omega_2 - \Omega_3 - \Omega_4$$

In the model we neglected the rotor (propulsion group) inertia J_r. Indeed:

$$J_r = J_p - J_m r = 0$$

where J_p is the propeller inertia, while J_m is the motor inertia.
Then we used the previously quoted equations to write the state space model of the drone, neglecting the equations for the forces along x and y, hence we neglected the equations of $\ddot{x}$ and $\ddot{y}$. This was

done since our drone is not able to move, due to the hinge. Still, we considered the equation of $\ddot{z}$, since it was useful for the mixing function.
We considered as states:

$$x_1 = \varphi^x$$
$$x_2 = \dot{x_1} = \dot{\varphi}^x$$
$$x_3 = \theta^x$$
$$x_4 = \dot{x_3} = \dot{\theta}^x$$
$$x_5 = \psi^x$$
$$x_6 = \dot{x_5} = \dot{\psi}^x$$
$$x_7 = z$$
$$x_8 = \dot{x_7} = \dot{z}$$

Considering that:

$$\dot{X} = f(X, U)$$

where: $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}^T$ and $U = \{U_1, U_2, U_3, U_4\}^T$, we had that:

$$f(X,U) = \begin{Bmatrix} x_2 \\ x_4 x_6 a_1 + b_1 U_2 \\ x_4 \\ x_2 x_6 a_3 + b_2 U_3 \\ x_6 \\ x_4 x_2 a_5 + b_3 U_4 \\ x_8 \\ -g + \dfrac{cosx_1\ cosx_3}{m} U_1 \end{Bmatrix}$$

Where:

$$a_1 = \frac{J_y - J_z}{J_x} \quad b_1 = \frac{1}{J_x}$$
$$a_3 = \frac{J_z - J_x}{J_y} \quad b_2 = \frac{1}{J_y}$$
$$a_5 = \frac{J_x - J_y}{J_z} \quad b_3 = \frac{1}{J_z}$$

As we can see in $f(X,U)$, the angles and their time derivatives do not depend on translation components. On the other hand, the translations depend on the angles, as we understood by looking at the last row of $f(X,U)$. The control scheme advocated for the overall system is then logically divided in a position controller and a rotation controller, where the rotation controller is the inner one:

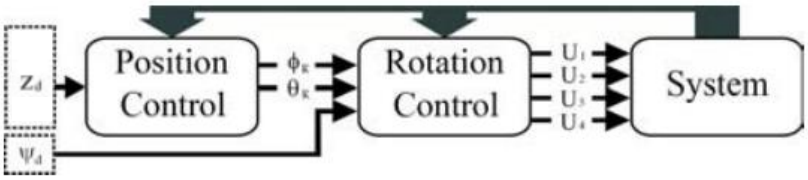


Figure 37: Backstepping Conceptual Control Scheme

As already mentioned, we are only interested in controlling the attitude, as our quadrotor is constrained by a spherical hinge and has only 3 DOF, the attitude indeed.
Using the backstepping approach, one can synthesize the control law forcing the system to follow the desired trajectory and this is done using the Lyapunov theory.

We considered the roll tracking-error:

$$z_1 = x_{1_d} - x_1$$

And we used the Lyapunov theorem by considering the Lyapunov function z_1 positive definite and it's time derivative negative semi-definite:

$$V(z_1) = \frac{1}{2} z_1^2$$

$$\dot{V}(z_1) = z_1(\dot{x}_{1_d} - x_2)$$

The stabilization of z_1 can be obtained by introducing a virtual control input x_2:

$$x_2 = \dot{x_{1_d}} + \alpha_1 z_1 \text{ with } \alpha_1 > 0\,,$$

$$\text{to obtain: } \dot{V}(z_1) = -\alpha_1 z_1^2$$

We proceeded with a variable change by making:

$$z_2 = x_2 - \dot{x_{1_d}} - \alpha_1 z_1$$

For the second step we considered the augmented Lyapunov function:

$$V(z_1, z_2) = \frac{1}{2}(z_1^2 + z_2^2)$$

and its time derivative is then:

$$\dot{V}(z_1, z_2) = z_2(a_1 x_4 x_6 + a_2 x_4 \Omega + b_1 U_2) - z_2(\ddot{x_{1_d}} - \alpha_1(z_2 + \alpha_1 z_1)) - z_1 z_2 - \alpha_1 z_1^2$$

The control input U_2 was then extracted $(\ddot{x}_{1,2,3_d} = 0)$, satisfying $\dot{V}(z_1, z_2) < 0$:

$$U_2 = \frac{1}{b_1}(z_1 - a_1 x_4 x_6 - \alpha_2 x_4 \Omega - \alpha_1(z_2 + \alpha_1 z_1) - \alpha_2 z_2)$$

Where the term $\alpha_2 z_2$ (with $\alpha_2 > 0$) was added to stabilize z_1.

The same steps were followed to extract U_3 and U_4 ($\alpha_3, \alpha_4, \alpha_5, \alpha_6 > 0$):

$$U_3 = \frac{1}{b_2}(z_3 - a_3 x_2 x_6 - \alpha_4 x_2 \Omega - \alpha_3(z_4 + \alpha_3 z_3) - \alpha_4 z_4)$$

$$U_4 = \frac{1}{b_3}(z_5 - a_5 x_2 x_4 - \alpha_5(z_6 + \alpha_5 z_5) - \alpha_6 z_6)$$

$$\begin{cases} z_3 = x_{3_d} - x_3 \\ z_4 = x_4 - \dot{x}_{3_d} - \alpha_3 z_3 \\ z_5 = x_{5_d} - x_5 \\ z_6 = x_6 - \dot{x}_{5_d} - \alpha_5 z_5 \end{cases}$$

Because we needed to re-map the vector $U = [U_1\ U_2\ U_3\ U_4]^T$ into $T = [T_1\ T_2\ T_3\ T_4]^T$, we needed the expression of U_1 too, that is needed for the altitude control. Indeed what is typically done is altitude+attitude control.

The altitude control action U_1 was obtained using the same approach described before ($\alpha_7, \alpha_8 > 0$):

$$U_1 = \frac{m}{cosx_1\ cosx_3}\ (z_7 + g\ -\ \alpha_7(z_8\ +\ \alpha_7 z_7)\ -\ \alpha_8 z_8)$$

$$\begin{cases} z_7 = x_{7_d} - x_7 \\ z_8 = x_8 - \dot{x}_{7_d} - \alpha_7 z_7 \end{cases}$$

Once we had the U vector evaluated by the controller, we needed to re-map it into the T vector.

Considering that in matrix form:

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -k_1 & k_1 & k_1 & -k_1 \\ k_1 & -k_1 & k_1 & -k_1 \\ k_2 & k_2 & -k_2 & -k_2 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = T\_to\_U * T$$

Where:

$$k_1 = L\frac{\sqrt{2}}{2}$$

$$k_2 = \frac{C_m}{C_t}$$

The inverse mapping is simply obtained by taking the inverse:

$$quadParams.MotorMixer = inv(T\_to\_U);$$

Where:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = quadParams.MotorMixer * \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

Once we found the control action in terms of vector T, using the look-up tables we obtained the values in terms of PWM, which were given to the motors to actually control the quadrotor.
Talking about the simulink scheme, the output feedback is the same used in the Cascade PID, but with the only difference that we used the measurements/estimation of the yaw angle too for the computation of the control inputs.
Like the Cascade PID scheme, the Backstepping control allowed us to define the references that our system has to follow, hence we introduced the same reference scheme that we used in the Cascade PID Simulink.
We put in input all the references and the measurements in a subsystem block, in which is defined the whole Backstepping logic.
We put both the reference and the measurements of z as 0, since we did not want the drone to increase its altitude, which is always 0 due to the spherical hinge constraint.
Talking about the Backstepping logic subsystem, we put all the references and measurements in input to a Matlab function block, which computes the values of the control inputs. We put in input to the

function block also the value $\Omega\ (= \omega_m)$, which was computed by using the formula stated above, where we obtained the values of the angular velocity of the propellers by making the square root of the absolute value of the ratio between the i^{th} thrust (obtained after the mixing function) and the thrust coefficient C_t.
Before putting the obtained value Ω in our function block, we delayed it with a delay block, in order to not induce our scheme to an algebraic loop.
As far as the function block is concerned, it trivially allows to compute the control inputs, given all the measurements and references (and the value Ω), using the previously described equations.
We trivially tuned the α_i $(i = 1,2,3,4,5,6,7,8)$ by using a trial and error approach, and we found that the best performing were:

```
alpha_angle=50;
alpha_angleRate=1\10;

alpha1=alpha_angle;
alpha2=alpha_angleRate;
alpha3=alpha_angle;
alpha4=alpha_angleRate;
alpha5=alpha_angle;
alpha6=alpha_angleRate;
alpha7=alpha_angle;
alpha8=alpha_angleRate;
```

Figure 38: Parameters Tuning Backstepping

We used smaller α_i for the velocity states $(i = 2,4,6,8)$, since we wanted to make the position control faster than the velocity one in terms of Lyapunov functions.
The rest of the function block just implements the Backstepping logic that we previously defined.

After the function block, we implemented another function block which was performing the motor mixing, hence, as described before, we just invert the T_to_U and we multiply it with the vector of the obtained control inputs U.

Once we obtained the values of the T vector we needed to make sure that they were all positive, therefore we put the same mixer function used in PP and in LQR after the before mentioned mixing function.

In conclusion we had that the output scheme is the same of all the other controllers, hence we had the conversion in PWM values of all the thrusts through the usage of the look-up tables.

6.4.1 Backstepping Performances

As far as the performances are concerned, we used the previously considered alphas and we obtained the following time histories (changing the reference to follow):

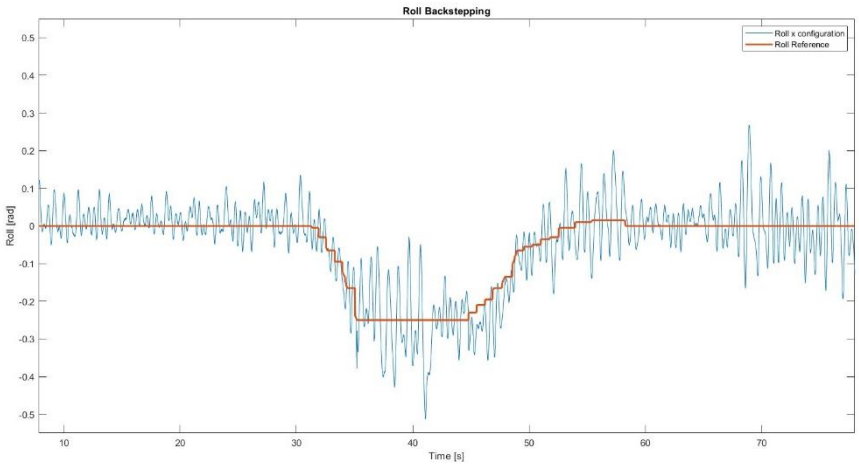


Figure 39: Roll Time History Backstepping

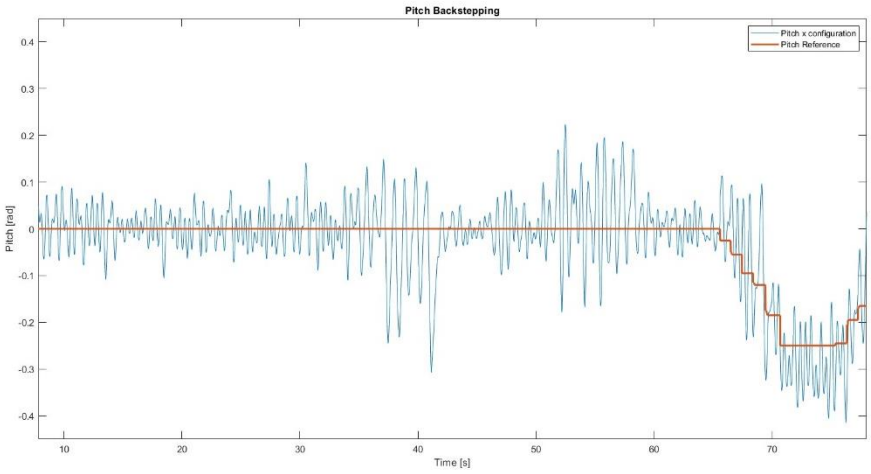


Figure 40: Pitch Time History Backstepping

As we can see, the controller is very responsive to the given reference, and results in having better performances with respect to the previous controllers. This is due to the non-linear nature of the control action, which directly acts on the nonlinear system. Another reason is that we considered a more complex model with respect to the others, still neglecting the presence of the cables and the structural friction introduced by the spherical hinge.
Another advantage of this control strategy is the possibility to track a given reference, while PP and LQR must be tuned on a given equilibrium and can be used only for regulation purposes.
The Cascade PID could in principle follow the input trajectories, but we decided to track a moving reference only for this control and for the Proportional Feedback Linearization control since they are better performing.

Another reference trajectory we wanted the controller to follow is a circle. The performances were the following:

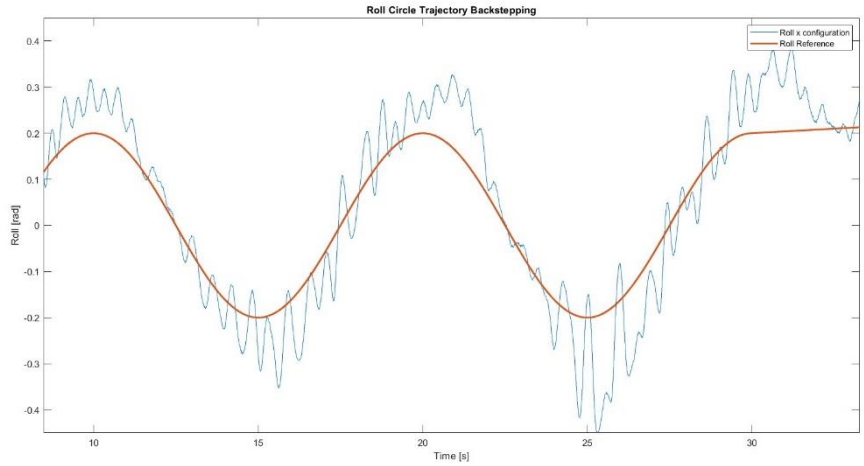


Figure 41: Roll Circle Trajectory Backstepping

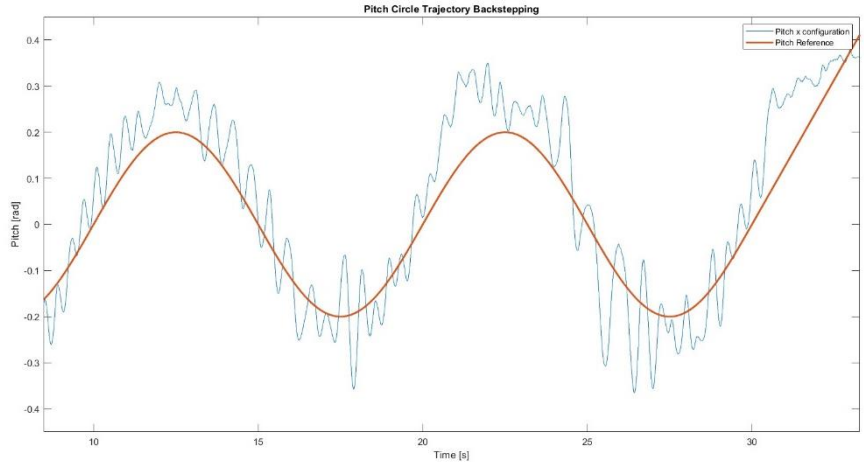


Figure 42: Pitch Circle Trajectory Backstepping

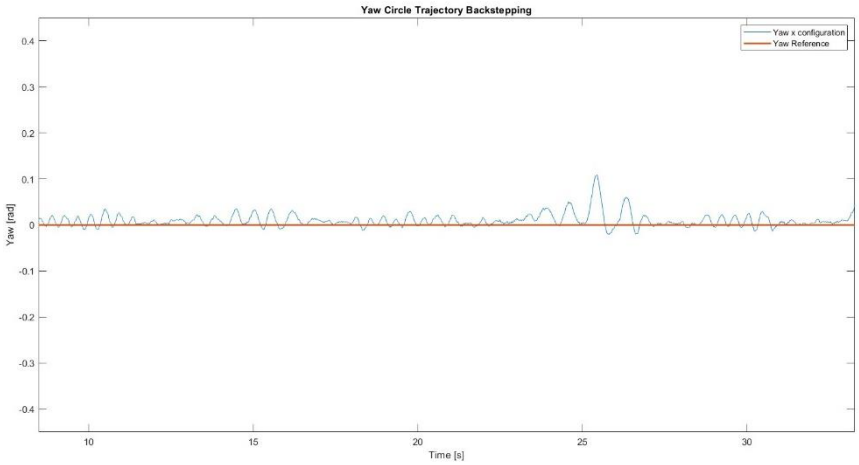


Figure 43: Yaw Circle Trajectory Backstepping

As we can see the trajectory is well followed, but the oscillations are increased, due to the complexity of the requested reference. Obviously they are more relevant for roll and pitch, due the higher complexity of the control laws related to them, while for yaw are approximately negligible.

6.5 Proportional Feedback Linearization Control

The last control that we applied is the Feedback Linearization, which is a technique used in nonlinear control systems to transform a nonlinear system into an equivalent linear system through a change of variables and a suitable control input. By applying a nonlinear state transformation and a nonlinear feedback control law, the original nonlinear dynamics are "canceled out," resulting in a system that behaves like a linear system. This allows the use of linear control methods to achieve desired performance, making it easier to design controllers for complex nonlinear systems.
The dynamic equations of our model, based upon the “x” configuration are [6]:

$$\ddot{\varphi}^x = \dot{\theta}^x\dot{\psi}^x\left(\frac{J_y - J_z}{J_x}\right) + \frac{1}{J_x}U_2$$
$$\ddot{\theta}^x = \dot{\varphi}^x\dot{\psi}^x\left(\frac{J_z - J_x}{J_y}\right) + \frac{1}{J_y}U_3$$
$$\ddot{\psi}^x = \dot{\theta}^x\dot{\varphi}^x\left(\frac{J_x - J_y}{J_z}\right) + \frac{1}{J_z}U_4$$
$$\ddot{z} = -g + \frac{(cos\varphi^x\, cos\theta^x)}{m}U_1$$
$$\ddot{x} = \frac{(cos\varphi^x sin\theta^x cos\psi^x + sin\varphi^x sin\psi^x)}{m}U_1$$
$$\ddot{y} = \frac{(cos\varphi^x sin\theta^x sin\psi^x - sin\varphi^x cos\psi^x)}{m}U_1$$

Where (like the Cascade PID and Backstepping):

$$U_1 = T_1 + T_2 + T_3 + T_4$$
$$U_2 = \frac{L\sqrt{2}}{2}(-T_1 + T_2 + T_3 - T_4)$$
$$U_3 = \frac{L\sqrt{2}}{2}(T_1 - T_2 + T_3 - T_4)$$
$$U_4 = M_1 + M_2 - M_3 - M_4$$
$$\Omega = \Omega_1 + \Omega_2 - \Omega_3 - \Omega_4$$

We also neglected the rotor (propulsion group) inertia J_r. Indeed:

$$J_r = J_p - J_m r = 0$$

where J_p is the propeller inertia, while J_m is the motor inertia.
The considered model is the same used for Backstepping.
Then we re-wrote the system in state space representation, but we also considered, differently from the Backstepping approach, also the equations for the position x and y and their respective velocities, which are useful for the division in subsystems:

$$\begin{array}{cc} x_1 = \varphi & x_7 = z \\ x_2 = \dot{x_1} = \dot{\varphi}^x & x_8 = \dot{x_7} = \dot{z} \\ x_3 = \theta^x & x_9 = x \\ x_4 = \dot{x_3} = \dot{\theta}^x & x_{10} = \dot{x_9} = \dot{x} \\ x_5 = \psi^x & x_{11} = y \\ x_6 = \dot{x_5} = \dot{\psi}^x & x_{12} = \dot{x_{11}} = \dot{y} \end{array}$$

Considering that:

$$\dot{X} = f(X, U)$$

where: $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}\}^T$ and $U = \{U_1, U_2, U_3, U_4\}^T$, we had that:

$$f(X,U) = \begin{Bmatrix} x_2 \\ x_4 x_6 a_1 + b_1 U_2 \\ x_4 \\ x_2 x_6 a_3 + b_2 U_3 \\ x_6 \\ x_4 x_2 a_5 + b_3 U_4 \\ x_8 \\ -g + \frac{cosx_1\, cosx_3}{m} U_1 \\ x_{10} \\ \frac{u_x}{m} U_1 \\ x_{12} \\ \frac{u_y}{m} U_1 \end{Bmatrix}$$

Where:

$$\begin{array}{cc} a_1 = \dfrac{J_y - J_z}{J_x} & b_1 = \dfrac{1}{J_x} \\ a_3 = \dfrac{J_z - J_x}{J_y} & b_2 = \dfrac{1}{J_y} \\ a_5 = \dfrac{J_x - J_y}{J_z} & b_3 = \dfrac{1}{J_z} \end{array}$$

And:

$$u_x = cosx_1\, sinx_3\, cosx_5 + sinx_1\, sinx_5$$
$$u_y = cosx_1\, sinx_3\, sinx_5 - sinx_1\, cosx_5$$

As we can see in $f(X,U)$, the angles and their time derivatives do not depend on translation components. On the other hand, the translations depend on the angles. The control scheme advocated for the overall system is then logically divided into a position controller and an attitude controller, where the rotation controller is the inner one.
We are only interested in controlling the attitude, as our quadrotor is constrained by a spherical hinge and has only 3 DOF, the attitude indeed.
In order to apply the Feedback Linearization we divided our model in 2 sub-models:

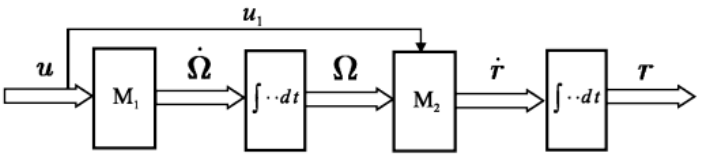


Figure 44: Overall model of the quadrotor dynamics Feedback Linearization

Talking about the sub-model M_1, which describes the angular rates dynamics of the quadrotor, we considered only those state equations:

$$\dot{x}_2 = x_4 x_6 a_1 + b_1 U_2$$

$$\dot{x}_4 = x_2 x_6 a_3 + b_2 U_3$$

$$\dot{x}_6 = x_4 x_2 a_5 + b_3 U_4$$

We obtained The Euler angles x_1, x_3, x_5 by pure integration.

Instead, the sub-model M_2 describes the velocities dynamics of the quadrotor, hence we considered those state equations:

$$\dot{x}_8 = -g + \frac{cosx_1\ cosx_3}{m} U_1$$

$$\dot{x}_{10} = \frac{u_x}{m} U_1$$

$$\dot{x}_{12} = \frac{u_y}{m} U_1$$

We obtained the states x_7, x_9, x_{11} by pure integration since they represent the positions.

Generally, a UAV system contains two main control loops. The first main and underlying control loop is the vehicle control loop. The second main loop is the mission control loop that comprises the stabilized vehicle as a platform for mission related sensors and actuators and the mission control system. The mission control loop computes the desired flight path, e.g. given by waypoints, and commands current required movements to the vehicle control loop, direct position control or desired velocity vector.
We chose the desired velocity approach. However, both the approaches refer to the outer loop, as can be seen in the following figure. The output of C_2 refers to the inner loop, that is an attitude control loop:

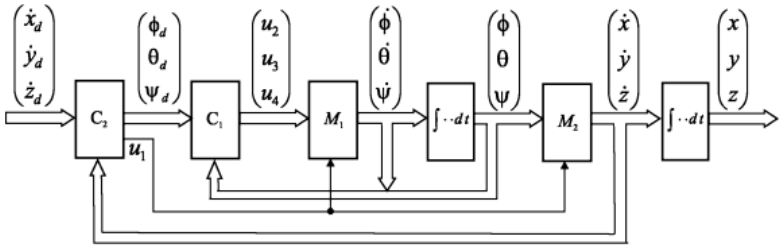


Figure 45: Nested Structure of the UAV Attitude Control Feedback Linearization

In our setup, we considered only the inner loop and we manually set the desired Euler Angles, to test the correctness of the attitude control system.
The corresponding dynamic model comprises the first six equations of the state variable model, which is a series of the nonlinear sub model M_1 and an integrator.
In our model we neglected the gyroscopic effect, but in theory the controller C_1 can stabilize the attitude also with gyroscopic terms.
The considered equations are:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_4 x_6 a_1 + b_1 U_2 \\ x_4 \\ x_2 x_6 a_3 + b_2 U_3 \\ x_6 \\ x_4 x_2 a_5 + b_3 U_4 \end{bmatrix}$$

We applied a feedback linearization to obtain a linear system:

$$U_2 = f_2(x_2, x_4, x_6) + U_2^*$$

$$U_3 = f_3(x_2, x_4, x_6) + U_3^*$$

$$U_4 = f_4(x_2, x_4, x_6) + U_4^*$$

With the new input variables U_2^*, U_3^*, U_4^*.
In order to obtain a linear system, the following conditions had to be fulfilled:

$$x_4 x_6 a_1 + b_1 f_2(x_2, x_4, x_6) = K_2 x_2$$

$$x_2 x_6 a_3 + b_2 f_3(x_2, x_4, x_6) = K_3 x_4$$

$$x_4 x_2 a_5 + b_3 f_4(x_2, x_4, x_6) = K_4 x_6$$

Therefore, the nonlinear feedback linearization law is the following:

$$f_2(x_2, x_4, x_6) = \frac{1}{b_1}(K_2 x_2 - x_4 x_6 a_1)$$

$$f_3(x_2, x_4, x_6) = \frac{1}{b_2}(K_3 x_4 - x_2 x_6 a_3)$$

$$f_4(x_2, x_4, x_6) = \frac{1}{b_3}(K_4 x_6 - x_4 x_2 a_5)$$

Using this feedback turns into the linear and decoupled system:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} x_2 \\ K_2 x_2 + b_1 U_2^* \\ x_4 \\ K_3 x_4 + b_2 U_3^* \\ x_6 \\ K_4 x_6 + b_3 U_4^* \end{bmatrix}$$

It can be shown that the resulting linear closed-loop system is stable even if we consider the gyroscopic terms.
The dynamics of the angles using the linear dynamics and neglecting the gyroscopic terms were described by linear decoupled differential equations of second order.

For example, for the roll dynamics we had ($x_1 = \varphi^x, x_2 = \dot{\varphi}^x$):

$$\ddot{\varphi}^x = K_2 \dot{\varphi}^x + b_1 U_2^*$$

That is:

$$\ddot{x_1} = K_2 \dot{x_1} + b_1 U_2^*$$

If x_{1_d} is the desired roll angle, the application of a linear controller $U_2^* = w_2 * (x_{1_d} - x_1)$ with constant parameter w_2 leads to a closed-loop system of second order with the following transfer function:

$$F(s) = \frac{X_1(s)}{X_{1_d}(s)} = \frac{w_2}{\frac{1}{b_1} s^2 - \frac{K_2}{b_1} s + w_2}$$

The same considerations held for the other angles with linear controllers $U_3^* = w_3 * (x_{3_d} - x_3)$ and $U_3^* = w_3 * (x_{5_d} - x_5)$.

The dynamics of these closed-loop systems were easily defined by the adjustment of the pairs of parameters $(K_2, w_2), (K_3, w_3), (K_4, w_4)$, respectively, with the only limitation that the parameters K_2, K_3, K_4 must be negative.
A possible choice is: $K_2 = -80$ and with $w_2 = \left(\frac{K_2}{2}\right)^2 * \frac{1}{b_1}$ for zero overshoot since the poles resulted to be negative and real valued. We chose the best performing by using a trial-and-error approach.

Talking about the Simulink scheme, the output feedback and the reference choice part is the same of the Backstepping, but we did not put in input to our subsystem block, where the whole control logic is contained, the desired altitude and measured altitude, which are both 0 for the same reasons specified in the Backstepping section.

Afterwards we put all the measurements and the reference in input to a subsystem in which is contained the full control logic.
In this subsystem we firstly put the function which computes the values of U_2^*, U_3^* and U_4^* considering the previously defined logic and then a second function block which computes the values of U_2, U_3 and U_4.The control input values are then put in input to a mixer function, which converts them to the needed thrust values. The conversion is the same used in the Backstepping control. The control input

U_1 is trivially put to 0 since the reference value and "measured" values are both 0, like already explained.
In conclusion we had that the output scheme is the same of all the other controllers, hence we had the conversion in PWM values of all the thrusts through the usage of the look-up tables.

6.5.1 Feedback Linearization Performances

As far as the performances are concerned, we chose to use the previously described proportional gains:

$$K_2 = -80; K_3 = -80; 0\ K_4 = -80$$

We obtained the following time histories (directly considering a circle trajectory):

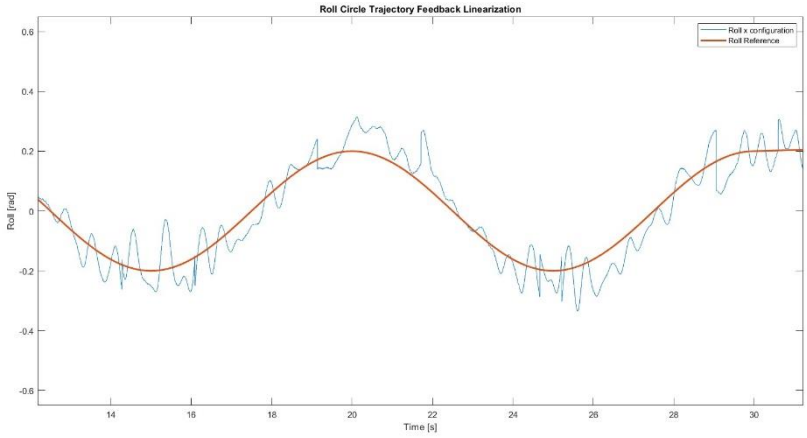


Figure 46: Roll Circle Trajectory Proportional Feedback Linearization

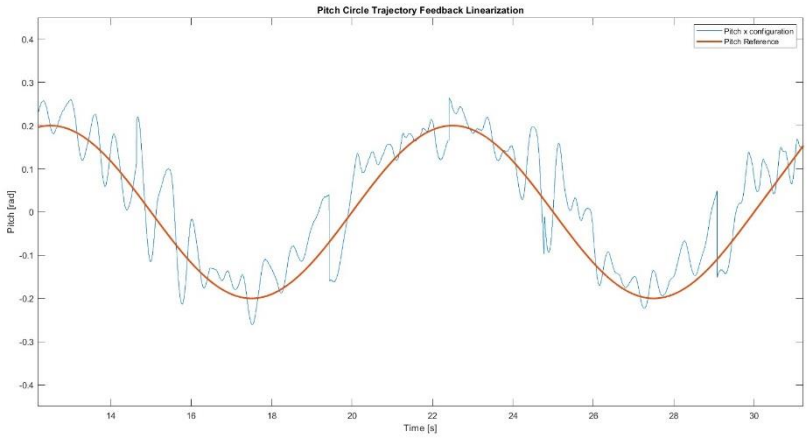


Figure 47: Pitch Circle Trajectory Proportional Feedback Linearization

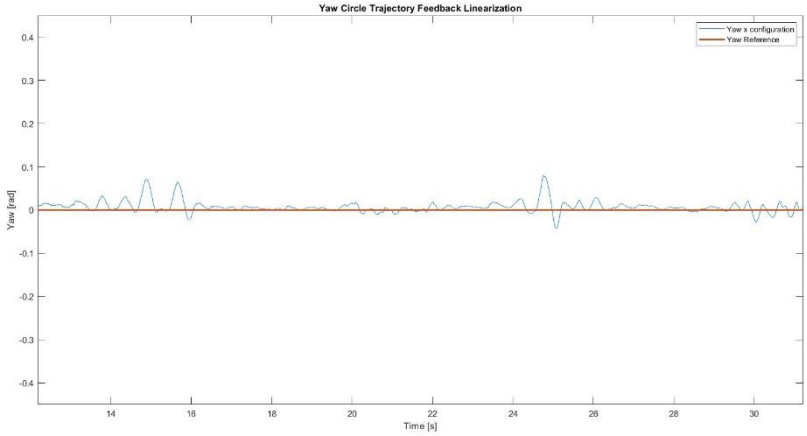


Figure 48: Yaw Circle Trajectory Proportional Feedback Linearization

As we can see, the results are very good in terms of trajectory following, and the oscillations are quite low. This is mainly due to the nature of the Feedback Linearization. Indeed, the nonlinear control action can compensate the nonlinearities in the system, making it act like a linear system. On top of that, the proportional controller can work on the error guaranteeing good performance.

With respect to the Backstepping control, considering the circle reference trajectory, we have smaller oscillations, since the proportional controller can compensate better for the error between the reference and the measurements of the angles. In fact, the Backstepping control law is based on the Lyapunov theory and is more dependent on the identified parameters and the mismatch between the identified parameters and the real ones could lead to greater oscillations.

6.6 Feedback Linearization Control with linear controllers

It is an alternative version of the Feedback Linearization but applying linear controllers like PP and LQ.
After the linearizing control law had been applied, we built a linear control technique on top of it.
The linearizing control law is the following:

$$x_4 x_6 a_1 + b_1 U_2 = U_2^*$$

$$x_2 x_6 a_3 + b_2 U_3 = U_3^*$$

$$x_4 x_2 a_5 + b_3 U_4 = U_4^*$$

From which we obtained:

$$U_2 = f_2(x_2, x_4, x_6) + U_2^* = \frac{-x_4 x_6 a_1}{b_1} + U_2^*$$

$$U_3 = f_3(x_2, x_4, x_6) + U_3^* = \frac{-x_2 x_6 a_3}{b_2} + U_3^*$$

$$U_4 = f_4(x_2, x_4, x_6) + U_4^* = \frac{-x_4 x_2 a_5}{b_3} + U_4^*$$

Since the resulting system was a linear one, we obtained the state space representation, and we built a linear control technique based on it. From the state space representation point of view, we got the following:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} x_2 \\ U_2^* \\ x_4 \\ U_3^* \\ x_6 \\ U_4^* \end{bmatrix}$$

That can be re-written in compact form as:

$$\dot{x} = Ax + Bu$$

Where:

$x = [x_1\ x_2\ x_3\ x_4\ x_5\ x_6]^T$ and $u = [U_2^*\ U_3^*\ U_4^*]^T$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

6.6.1 Feedback Linearization Control with Pole placement

As a first controller that we implemented in our structure, we chose the pole placement controller. It is a simple regulation controller that consists in a state-feedback, built such that the poles of the overall system are placed in a desired position.

Once again, as the linearized case, the poles are placed based on a trade-off between performances and robustness to noise. The ones chosen are:

$$poles = [0.96; 0.96; 0.96; 0.96; 0.96; 0.96]$$

The results show a better performance of the system in terms of steady-state error and resonance peaks with respect to the linearized case.

6.6.2 Feedback Linearization Control with LQ controller

The second controller that we implemented on our system was an LQ controller. It consists again of a state-feedback whose gain is computed by optimizing a cost function whose parameters, Q and R, depend on the relative importance given to state convergence and control effort. The values of these parameters were chosen thanks to the previously explained script based on the performances we imposed.

The results show very good performances as expected from a control applied to a linear system.

NOTE: The oscillations in all the time histories depicted in chapter 6 are also due to the plotting methodology, in fact we are plotting the noisy measurements over time, in which we can clearly see the effect of both high frequency measurement noise and the noise introduced by the rotation of the propellers. Another notable detail is that we tuned our controllers in order to allow the control action to be under a certain limit, to deal with the saturation of the actuators.

7) Performance Improvement

Our final goal was to reduce the oscillations in the time histories of the controls. In order to achieve this result, we implemented the Extended Kalman Filter together with the controls, with better tuning of the parameters. We reduced the bandwidth of each one of our controllers and we made each controller less reactive, hence less sensitive to noise and disturbances. However, since we were plotting out results in the last days, due to memory and plant issues we were not able to retrieve the desired time histories.

For this reason, we decided to work only in simulation, and we proved that the implementation of those changes allowed us to improve our performance.
In the following sections, we put on the left the closed-loop responses of the considered controls without filtering the noise, while on the right the ones with the EKF.

7.1 PP Improvement

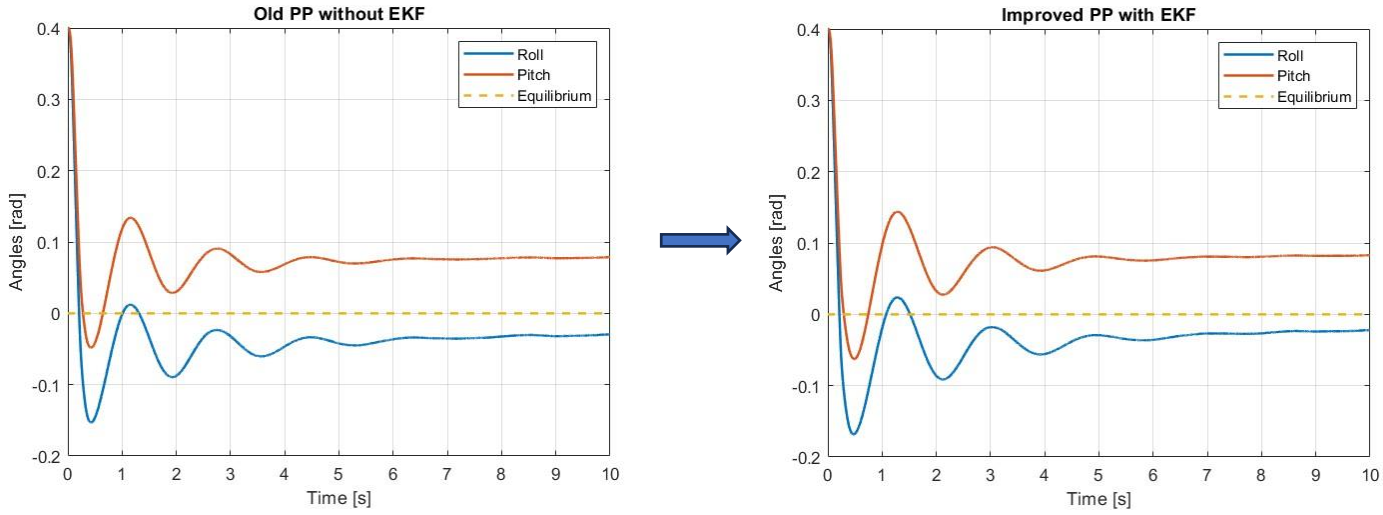


Figure 49: PP Improvement

	OLD	NEW
***Roll Settling Time* (0.017 *rad range*)**	$> 10\ s$	$> 10\ s$
Roll Steady State Error	0.029592	0.022245
***Pitch Settling Time* (0.017 *rad range*):**	$> 10\ s$	$> 10\ s$
Pitch Steady State Error	0.078574	0.082823

7.2 LQ Improvement

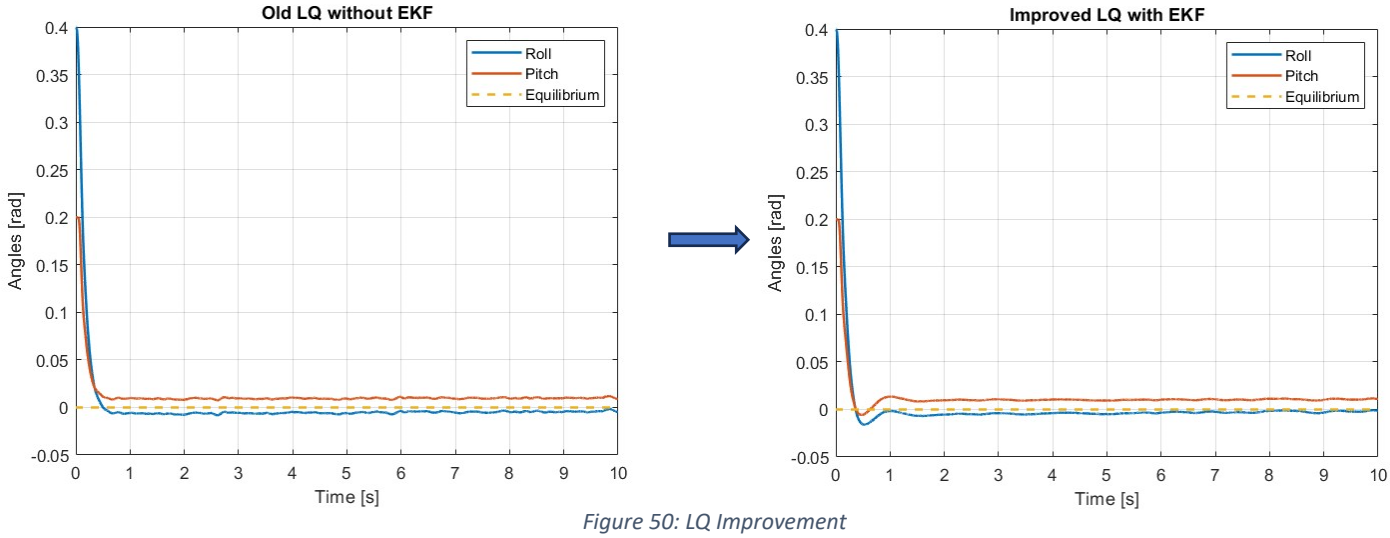


Figure 50: LQ Improvement

	OLD	NEW
***Roll Settling Time* (0.017 *rad range*)**	$0.36\ s$	$0.312\ s$
Roll Steady State Error	0.0049994	0.0013895
***Pitch Settling Time* (0.017 *rad range*):**	$0.386\ s$	$0.291\ s$
Pitch Steady State Error	0.0089351	0.010876

7.3 Cascade PID Improvement

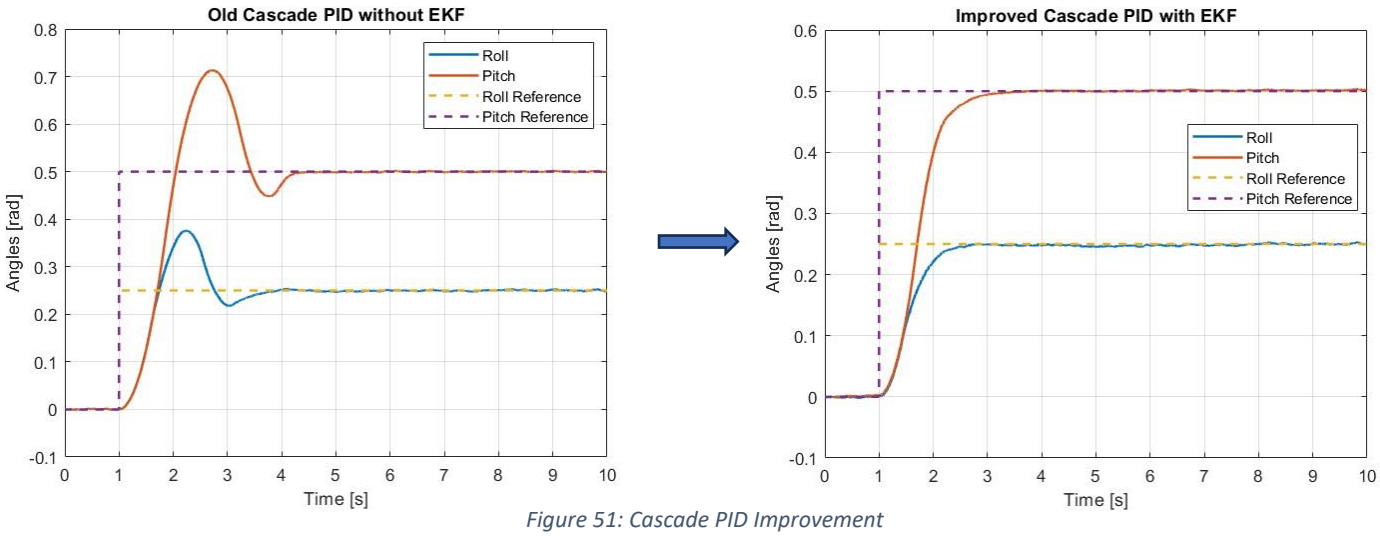


Figure 51: Cascade PID Improvement

	OLD	**NEW**
***Roll Settling Time* (0.017 *rad range*)**	2.304 s	1.133 s
Roll Steady State Error	0.0015867	0.00022935
***Pitch Settling Time* (0.017 *rad range*):**	3.051 s	1.637s
Pitch Steady State Error	0.0027949	0.0018736

It had been re-tuned using the following parameters:

	Roll Inner Loop	Roll Outer Loop	Pitch Inner Loop	Pitch Outer Loop	Yaw Loop
P	7.8292876	2.8094800	7.8292876	2.8094800	7.29164973
I	30.6143547	0	30.6143547	0	128.9779068
D	−0.0391617	0	−0.0391617	0	0.0102788
N	24.5406262	0	24.5406262	0	308.6585022

7.4 Backstepping Improvement

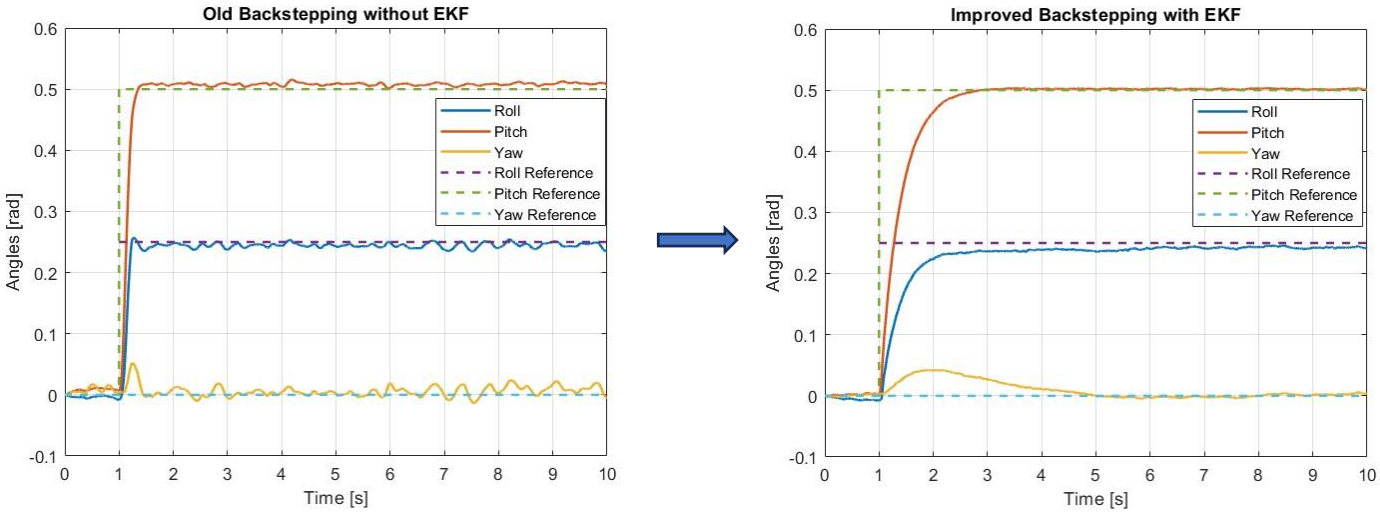


Figure 52: Backstepping Improvement

	OLD	NEW
***Roll Settling Time* (0.017 *rad range*)**	0.213 s	1.433 s
Roll Steady State Error	0.015185	0.0085451
***Pitch Settling Time* (0.017 *rad range*):**	0.29 s	1.224 s
Pitch Steady State Error	0.0075184	0.0018689
***Yaw Settling Time* (0.017 *rad range*):**	8.856 s	2.522 s
Yaw Steady State Error	0.0045515	0.0036519

It had been retuned using the following parameters:

$$\alpha_1 = \alpha_3 = \alpha_5 = \alpha_7 = 200$$

$$\alpha_2 = \alpha_4 = \alpha_6 = \alpha_8 = 1/3$$

7.5 Proportional Feedback Linearization Improvement

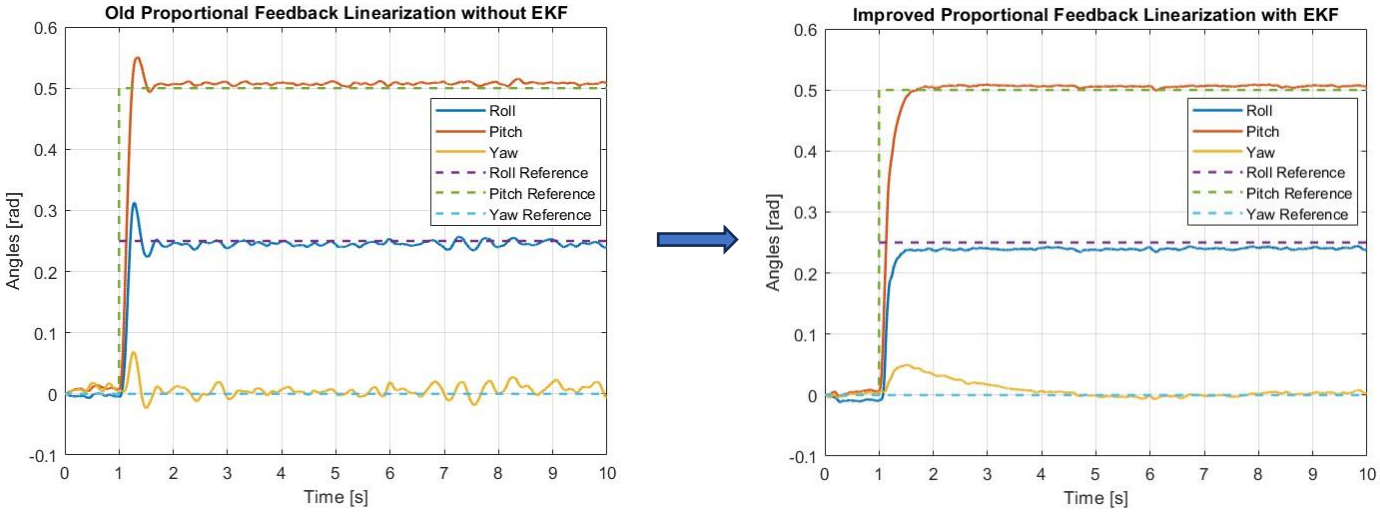


Figure 53: Proportional Feedback Linearization Improvement

	OLD	NEW
***Roll Settling Time* (0.017 *rad range*)**	0.587 s	0.424 s
Roll Steady State Error	0.010718	0.010887
***Pitch Settling Time* (0.017 *rad range*):**	0.464 s	0.468 s
Pitch Steady State Error	0.0085246	0.0045669
***Yaw Settling Time* (0.017 *rad range*):**	8.856 s	1.981 s
Yaw Steady State Error	0.0062478	0.0025638

7.6 PP Feedback Linearization Improvement

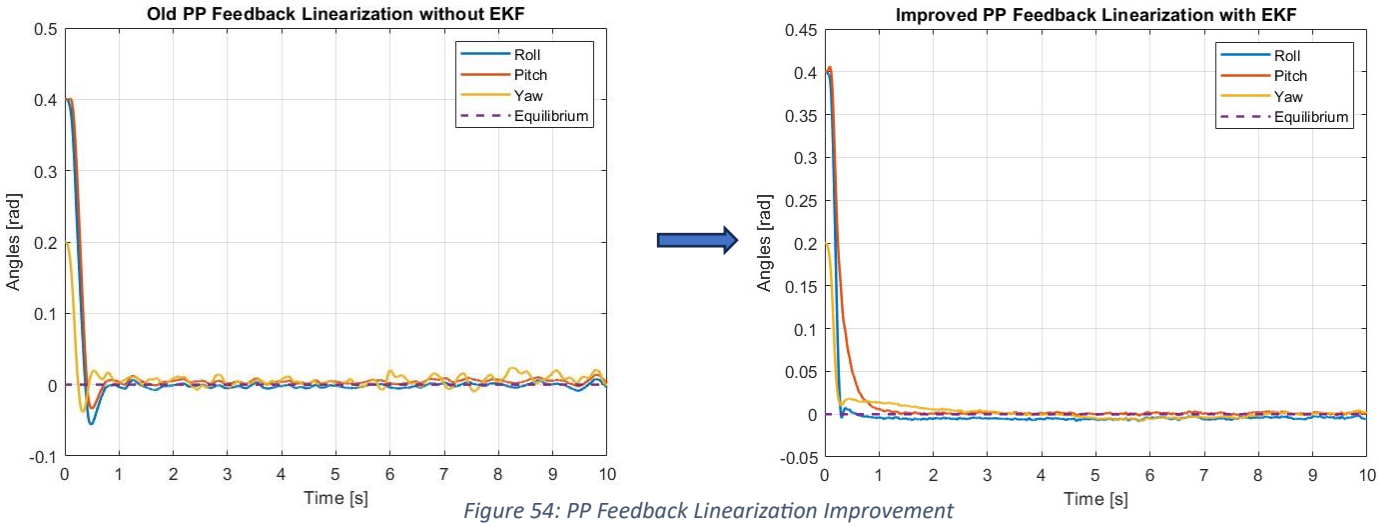


Figure 54: PP Feedback Linearization Improvement

	OLD	NEW
***Roll Settling Time* (0.017 *rad range*)**	0.673 s	0.276 s
Roll Steady State Error	0.0041762	0.0054673
***Pitch Settling Time* (0.017 *rad range*):**	0.614 s	0.715 s
Pitch Steady State Error	0.0022411	0.0013215
***Yaw Settling Time* (0.017 *rad range*):**	9.86 s	0.478 s
Yaw Steady State Error	0.00058225	0.0021671

7.7 LQ Feedback Linearization Improvement

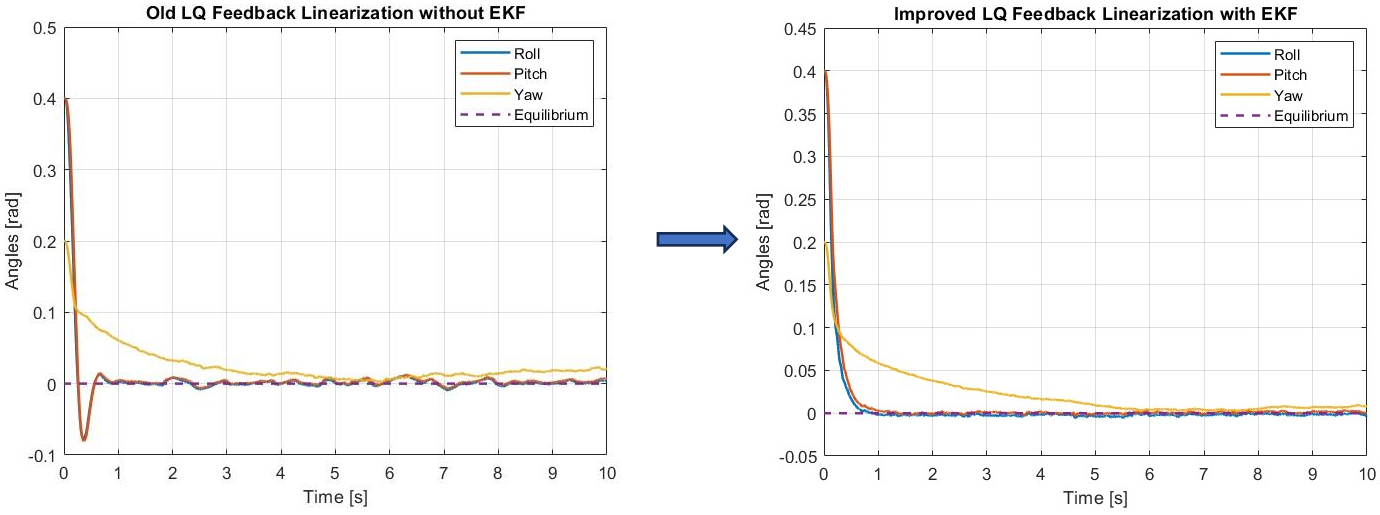


Figure 55: LQ Feedback Linearization Improvement

	OLD	NEW
***Roll Settling Time* (0.017 *rad range*)**	0.51 s	0.494 s
Roll Steady State Error	0.0045706	0.002061
***Pitch Settling Time* (0.017 *rad range*):**	0.519 s	0.565 s
Pitch Steady State Error	0.0078333	0.00074255
***Yaw Settling Time* (0.017 *rad range*):**	> 10 s	3.819 s
Yaw Steady State Error	0.020639	0.0079503

8) Conclusions

During the laboratory experience, we implemented several controllers, spacing from linear ones, like the PP and LQR, to more complex ones, like Backstepping and Feedback Linearization.
A common factor between all the controls is the model identification, which must be correctly done in order to guarantee the required performance.

Talking about linear controllers (PP and LQR), they are good at regulation, but do not achieve the perfect convergence to the equilibrium.

For this reason, we decided to implement a more complex control scheme, which is the Cascade PID. The advantage with respect to the previous ones is that it allows us to define a desired trajectory and it is more robust.
Still, a perfect convergence to the desired reference is not fully achieved, since it is based on the linearization of a complex nonlinear model, and its performance is not the best achievable. The same applies for PP and LQR. In addition to this, the application range of those controllers is limited, since they are constrained to work only around the considered equilibrium.
For this reason, we developed more complex controllers, which are Backstepping and Feedback Linearization. They are nonlinear control techniques; therefore, they do not require any linearization around a given equilibrium.

Talking about Backstepping, the main advantages of this control are that it allows to perform, like the Cascade PID, trajectory following, and it is performing better, since it can compensate for the nonlinear dynamics.

The same can be said for the Proportional Feedback Linearization.

The last controllers that we developed were the PP Feedback Linearization and the LQ Feedback Linearization. They compensate for the nonlinear dynamics too, but they are suitable for regulation purposes.
The introduction of the Feedback Linearization law in the PP and LQ controls makes a great difference in terms of steady state error, which is much lower with respect to standard PP and LQ.
In fact:

$$Pitch\ Steady\ State\ Error_{Standard\ PP} = 0.082823$$
$$Pitch\ Steady\ State\ Error_{Feedback\ Linearization\ PP} = 0.0013215$$

As far as the real-world application of our controls is concerned, we could implement a control algorithm that switches between different control techniques, based on the drone's operational requirements. In fact, when the drone must navigate to a specific position following an imposed trajectory, we could use one between Backstepping and Proportional Feedback Linearization, which are best performing in reference tracking. While, when the drone needs to hover in place to perform tasks like package delivery or high-resolution picture taking, we could switch to a regulation algorithm between PP Feedback Linearization or LQ Feedback Linearization, which are more stable around the equilibrium.

In conclusion, since there are some uncertainties on the identified parameters, a future development could be the implementation of an Adaptive Control algorithm like Model Reference Adaptive Control or VRFT.

Bibliography

[1] X. Dai, «Lesson 03 Experimental Platform Usage,» in *Multicopter Design and Control Practice-A Series Experiments Based on MATLAB and Pixhawk*.

[2] HolyBro, «HolyBro,» [Online]. Available: https://holybro.com/products/x500-v2-kits.

[3] J. C. P. J. A. Armando S. Sanca, «Dynamic Modeling of a Quadrotor Aerial Veichle with Nonlinear Input,» 2008.

[4] X. R. Z. W. Y. X. K. W. Nengsheng Bao, «Research on attitude controller of quadcopter based on cascade PID control algorithm,» 2017.

[5] S. B. a. R. Siegwart, «Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor,» Barcelona, 2005.

[6] H. Voos, «Nonlinear Control of a Quadrotor Micro-UAV using Feedback-Linearization,» Malaga, 2009.

[7] AIR2216II, «AIR2216II Datasheet,» [Online]. Available: https://cdn.shopify.com/s/files/1/0604/5905/7341/files/X500MotorSpec.png?v=1678791632.