Master in Control and Robotics

AUVE Lab 3 Report

Hierarchical control of an Unmanned Aerial Vehicle

Authors:
Conti Fabio, Pagano Francesco

Contents

1 Introduction

1.1 Context

The purpose of this lab is to perform a stabilized flight of a quadrotor using a cascaded control law. The software architecture of the robot is built with ROS2 and it is then simulated using GAZEBO. The quadrotor used for the simulation is represented in the following picture:

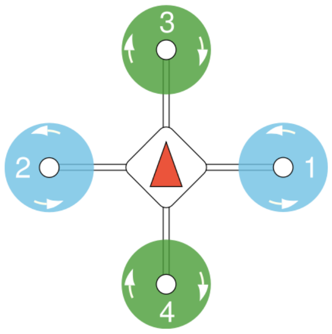


Figure 1: Scheme of the quadrotor

The axis of the body frame are such that x_b is pointing forward and z_b up; y_b is then pointing left. The useful variables for this lab are the following:

- Arm length : 0.17 m
- Rotor thrust gain: $kt = 5.5 * 10^{-6}$ N per rpm^2
- Rotor drag gain: $kd = 3.299 * 10^{-7}$ Nm per rpm^2
- Drone mass: $1kg$

In the context of this lab, we assume that you have a precise measure of the quadrotor configuration in 6D, and of the associated velocities. You can control the UAV via the rotation speed of each of the four motors (in rpm).

1.2 Objective

The objective of this lab is to create a control law capable of controlling the quadrotor's flight. Cascade control is a control system architecture in which two or more controllers operate in series, with the output of one controller serving as the input to the next. This type of control architecture is often used in quadrotor control because it allows for the implementation of separate controllers for different aspects of the quadrotor's motion, such as position and attitude. This can make the control system more flexible and efficient, allowing the quadrotor to respond more accurately to a wide range of control inputs and disturbances. Additionally, cascade control can make it easier to account for non-linearities in the system and to compensate for uncertainties and disturbances. In order to establish a control law of this kind, you must follow those three step:

- Write a *mixer-matrix* to compute the rotors velocities $\omega_1, \omega_2, \omega_3, \omega_4$ starting from the torques around the three axis and thrust force of the quadrotor.
- Write an attitude controller, using a control law based on quaternions.
- Write a position controller.

An example of this control scheme is shown in the following figure:

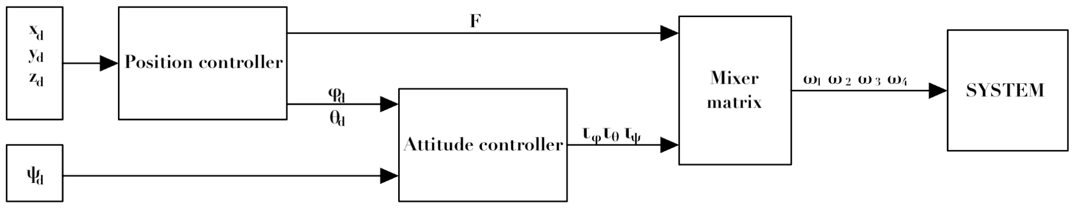


Figure 2: Scheme of the quadrotor's control law

Where:

- $[x_d, y_d, z_d]$: desired position.
- $[\psi_d, \phi_d, \theta_d]$: desired angle configuration. Yaw, pitch, roll angles.
- F: thrust force.
- $[\tau_\psi, \tau_\phi, \tau_\theta]$: torque vector around the drone's axis.
- $[\omega_1, \omega_2, \omega_3, \omega_4]$: angular velocities of the rotors.

The design process for each controller component and the reasoning behind the decisions made are covered in detail in the following sections.

2 Mixer-matrix

The input vector for such a system is a 4-dimensional vector: $\begin{bmatrix} U_1 \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}$ where:

- U_1 is the thrust generated by the rotors and it controls the altitude and the velocity of the quadrotor along the z-axis.
- τ_ϕ is the torque generated by the rotors positioned along the x-axis (motors 3, 4) and it controls the roll angle which is the torque around the x-axis;
- τ_θ is the torque generated by the rotors positioned along the y-axis (motors 1, 2) and it controls the pitch angle which is the torque around the y-axis;
- τ_ψ is the resultant torque generated by the 4 rotors and it controls the yaw angle which is the torque around the z-axis;

The mixer matrix is such that it brings this control space vector into the rotor velocities one. According to the body-fixed reference frame, the equations that express the above-mentioned control inputs as a function of the rotors' velocities allowed us to correctly build the Mixer matrix. The equations are the following:

$$\begin{aligned} U_1 &= k_t(\omega_3^2 + \omega_4^2 + \omega_1^2 + \omega_2^2) \\ \tau_\phi &= F_1 l - F_2 l = k_t(\omega_1^2 - \omega_2^2) \\ \tau_\theta &= F_4 l - F_3 l = k_t(\omega_4^2 - \omega_3^2) \\ \tau_\psi &= \tau_3 + \tau_4 - \tau_1 - \tau_2 = k_d(\omega_3^2 + \omega_4^2 - \omega_1^2 - \omega_2^2) \end{aligned} \tag{1}$$

By writing this set of equations in matrix form we obtain:

$$\begin{bmatrix} U_1 \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = P \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \begin{bmatrix} k_t & k_t & k_t & k_t \\ -k_t l & k_t l & 0 & 0 \\ 0 & 0 & -k_t l & k_t l \\ -k_d & -k_d & k_d & k_d \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{2}$$

By inverting P, we obtain the mixer Matrix M:

$$\begin{bmatrix}\omega_1^2\\ \omega_2^2\\ \omega_3^2\\ \omega_4^2\end{bmatrix} = M \begin{bmatrix}U_1\\ \tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} \tag{3}$$

Until now, velocities inputs have been calculated elevated to the square power. Therefore, it is necessary to apply the square root to each velocity input.

2.1 Mixer Matrix test

We tested the correctness of the mixer matrix by applying an input vector containing only a desired thrust such that the drone should start hoovering:

$$u = \begin{bmatrix}U_1\\ \tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} = \begin{bmatrix}10\\ 0\\ 0\\ 0\end{bmatrix}$$

As long as the drone's mass is 1 Kg, by applying $U_1 = 10$ N the drone started hoovering at a determined quote along the z-axis. The resulting rotors' velocities are given by the equation number 3. Afterwards, since we were provided with another simulation environment in which a virtual ball joint constraints the drone's motion we tried to apply control vectors containing also some desired torque. By applying a desired positive torque along a desired axis we were able to see how the drone started rotating around the chosen axis.

3 Attitude controller

The Attitude controller is the block that implements a PD controller that sets the desired drone's orientation using torques. However, the control law used in this implementation does not make use of canonical Euler angles but instead, the *quaternion error* is used to reach the goal. Generally, quaternions are preferred over Euler angles in robotics applications because they are more efficient, easier to work with mathematically, and do not suffer from *gimbal lock*. In a cascade control scheme for a quadrotor robot, the attitude controller must converge faster than the position controller because the attitude of the quadrotor directly affects its translational motion. If the attitude controller does not converge quickly enough, small errors in the quadrotor's orientation can accumulate and lead to large errors in its position. This can cause the position controller to become unstable or to produce undesirable motion. By ensuring that the attitude controller converges faster than the position controller, the cascade control scheme can help to stabilize the quadrotor and maintain its desired position and orientation. This can improve the performance and reliability of the quadrotor's control system, and allow it to respond more quickly and accurately to changes in its environment.
Given a desired orientation $\mathbf{q}_d$ and the current orientation $\mathbf{q}$ the quaternion error is defined as:

$$\tilde{\mathbf{q}} = \mathbf{q}^{-1}\mathbf{q}_d \tag{4}$$

Then, the Proportional Derivative Controller we used for this control scheme has the following form:

$$\tau = -\mathbf{K_d}\omega + \mathbf{K_p} sign(\tilde{\mathbf{q}}_r)\tilde{\mathbf{q}}_i \tag{5}$$

Where:

- $\mathbf{K_d}$ and $\mathbf{K_p}$: positive definite diagonal gain matrices.
- ω: its current angular velocity
- $\tilde{\mathbf{q}}_r$ and $\tilde{\mathbf{q}}_i$: which are respectively the real scalar part and the imaginary part of the quaternion error, represented as a vector.

- τ: drone input torque.

Given the cascade form of the robot controller, the input of the current attitude control block will come from an higher level controller called *Position controller.* This block will output the desired angle configuration which represents the rotation the attitude control block will try to apply on the system in terms of torque.

4 Position controller

The **position controller** is another important component of a quadrotor's control system. The goal of such a block is to enable the UAV to maintain a stable and consistent position, which is essential for a safe and successful flight. This is the highest-level portion of the overall controller. It aims at generating the required and desired angle configuration to input into the *attitude controller*. The position controller aims to find the desired force whose scalar value will control the thrust the rotors will produce. Indeed, this value will be the input of the mixer matrix. Instead, The force orientation will be an input for the lower-lever attitude controller. The force value follows the PD control law:

$$\mathbf{f} = \mathbf{K_d}(\mathbf{v_d} - \mathbf{v}) + \mathbf{K_p}(\mathbf{p_d} - \mathbf{p}) - m\mathbf{g} \tag{6}$$

where:

- $\mathbf{f}$ s the desired force provided by the drone
- $\mathbf{p_d}$ and $\mathbf{p}$ are respectively the desired and current position of the drone
- $\mathbf{v_d}$ and $\mathbf{v}$ are respectively the desired and current velocity of the drone
- m is the mass of the drone and $\mathbf{g}$ the gravity vector

Afterward, the scalar value of the force is extracted and set by reference to the corresponding f variable in the code as follows:

$$f = \|\mathbf{f}\|$$

The orientation the drone has to match w.r.t. the fixed frame **before** the position controller sets the desired thrust is expressed by the following vector:

$$\mathbf{R_d} = \begin{bmatrix} (\mathbf{y} \times \frac{\mathbf{f}}{f})^T & (\frac{\mathbf{f}}{f} \times (\mathbf{y} \times \frac{\mathbf{f}}{f}))^T & (\frac{\mathbf{f}}{f})^T \end{bmatrix}$$

where:

1. The third column is the direction of the desired force which is indeed the z-axis that the body-fixed frame must reach.
2. The second column is the y-axis that the body-fixed frame must reach.
3. The first column is the x-axis that the body-fixed frame must reach.
4. $\mathbf{y}$ is the unit vector along the y-axis in the fixed frame.
5. f is the desired scalar force
6. $\mathbf{f}$ is the desired force vector

Since the attitude controller works in terms of quaternions, $\mathbf{R_d}$ is converted into a quaternion and sent out by the position controller as output.

5 Conclusions and Results

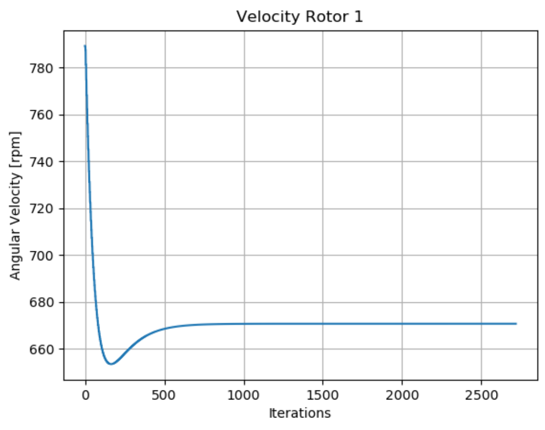
A quadrotor is an underactuated system since it has 6 DoF (three translations and three rotations) and only 4-dimensional control input. The hierarchical control structure allows controlling such a system by exploiting two controllers that can be designed and tuned separately according to the performance requirements.
To perform an analysis of the designed controller performances we defined three different commands:

1. Reaching a point that will make the drone fly vertically;
2. Reaching a point that will make the drone shift towards a single direction;
3. Reaching a point that will make the drone fly diagonally;

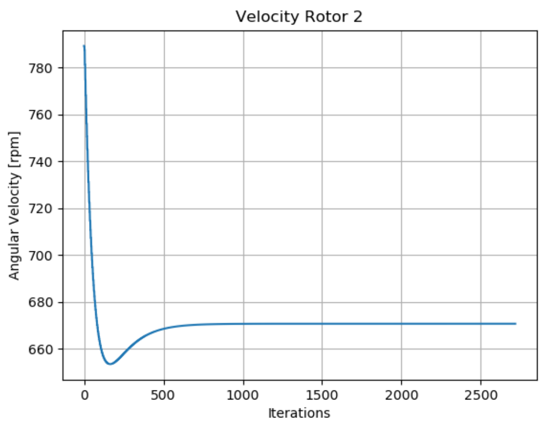
For each of these points, we decided to plot the rotor's velocities $(\omega_1, \omega_2, \omega_3, \omega_4)$, the position control errors $(x - x_d, y - y_d, z - z_d)$ and the orientation error as well (rotation angle error which is the real part of the quaternion vector).

5.1 Vertical take-off point

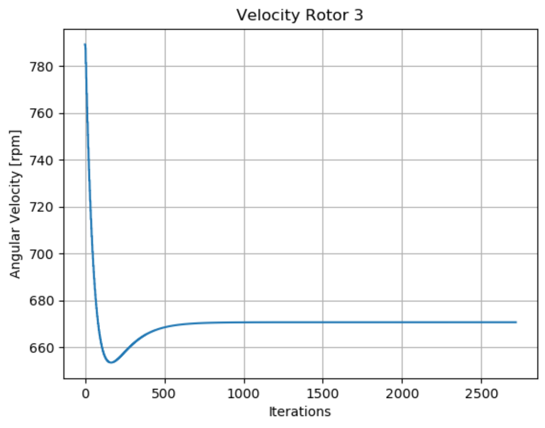
Here are the rotors' velocities:



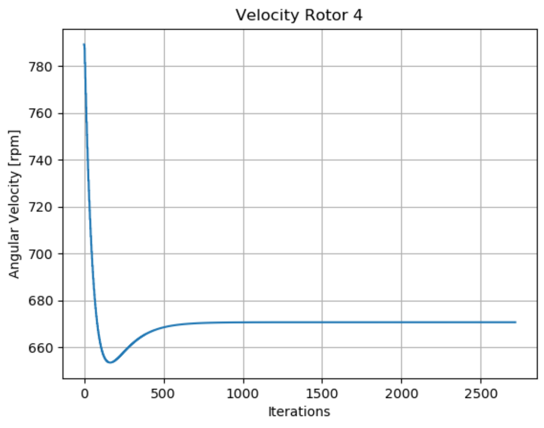
(a) First rotor angular velocity



(b) Second rotor angular velocity



(c) Third rotor angular velocity



(d) Fourth rotor angular velocity

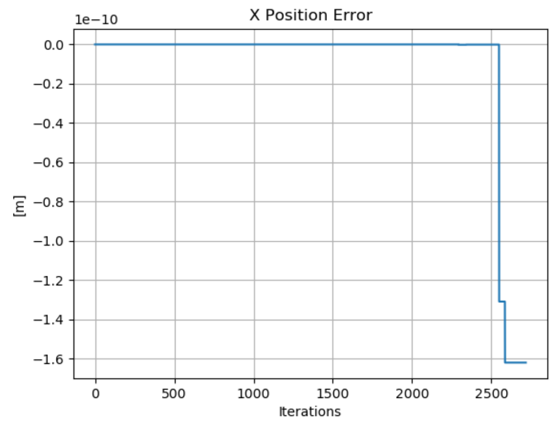
Figure 3: Rotors' velocities in the vertical take-off

Since no sideways motion is required to perform in such a case, the rotors' velocities resulted in being the same for each rotor for the vertical take-off. Indeed, without considering the input torques, the rotor's velocities follow the equation:

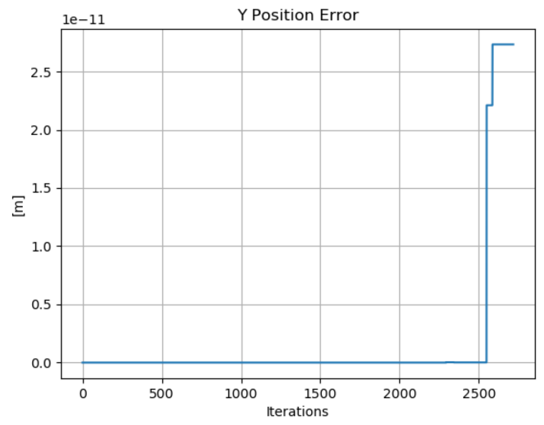
$$\omega_i = \sqrt{\frac{1}{4k_t} U_1} \tag{7}$$

for $i = 1, 2, 3, 4$.

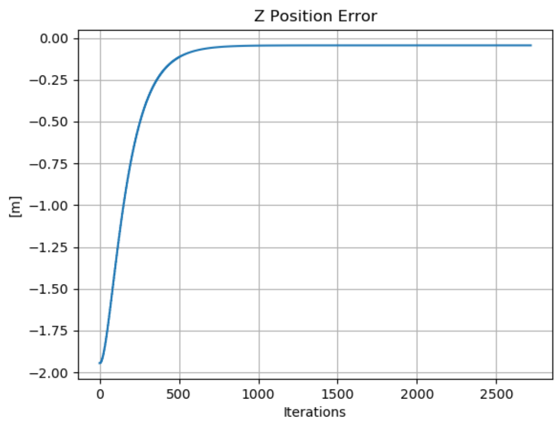
Here are both the control errors in translation and orientation:



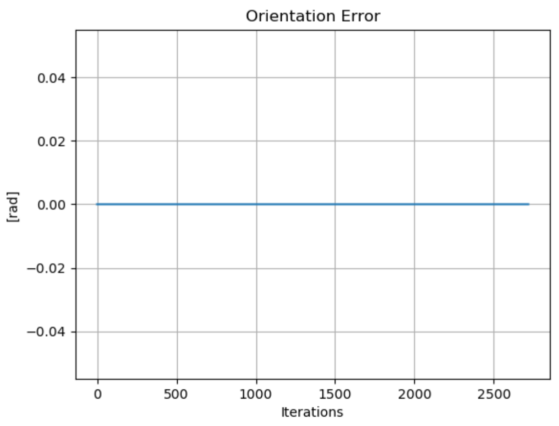
(a) X position error



(b) Y position error



(c) Z position error



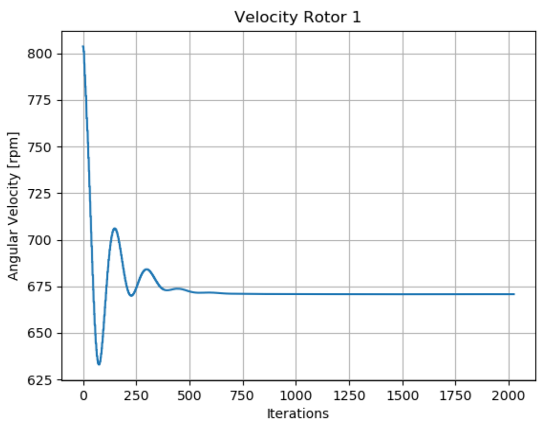
(d) Orientation error

Figure 4: Position and control errors in the vertical take-off

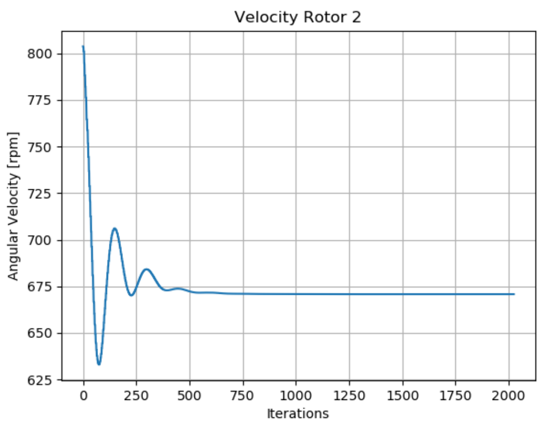
As expected, the only interesting error dynamic is the one related to the height. This error is minimized without any overshoot but with a steady-state error different from zero which could be fixed by tuning the PD gains in a different way.

5.2 Sideways Motion point

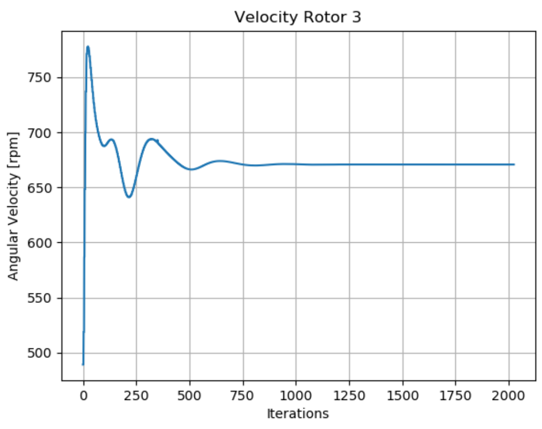
Here are the rotors' velocities:



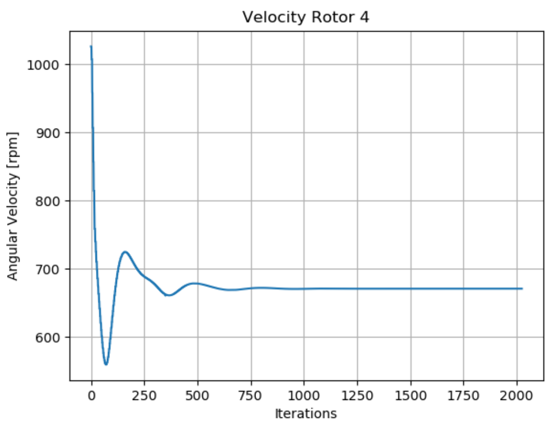
(a) First rotor angular velocity



(b) Second rotor angular velocity



(c) Third rotor angular velocity

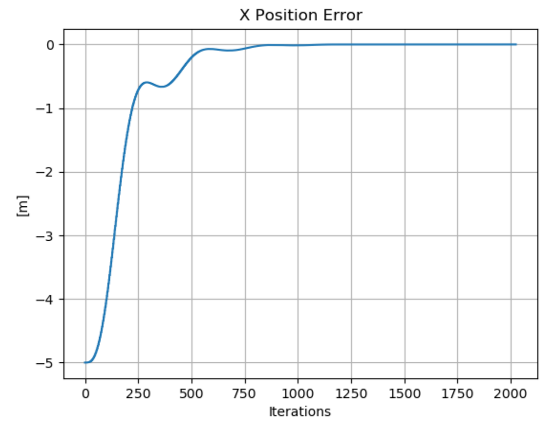


(d) Fourth rotor angular velocity

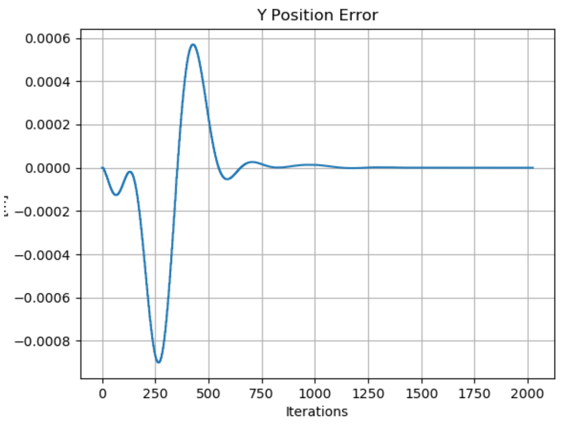
Figure 5: Rotors' velocities in the sideways motion

In the case of the sideways motion, we can see many more ripples in the convergence dynamic of the rotor velocities. This kind of behavior is given by the fact that this motion is way more complex than the one set before. In this case, the drone has to quickly tilt to the correct orientation and only then apply the thrust to move to the desired position. Therefore, the angular velocity of every rotor will have a different dynamic since the coordinates of the point of convergence are not located on the vertical axis anymore.

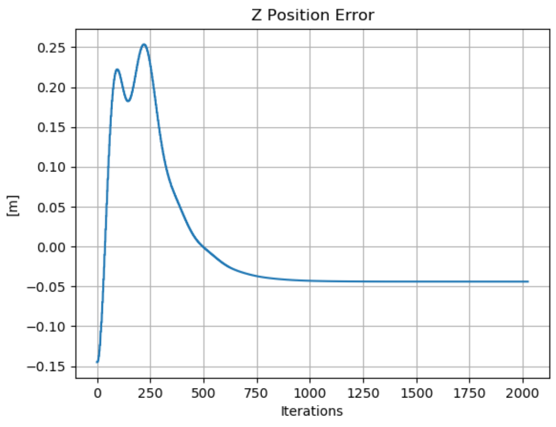
Here are both the control errors in translation and orientation:



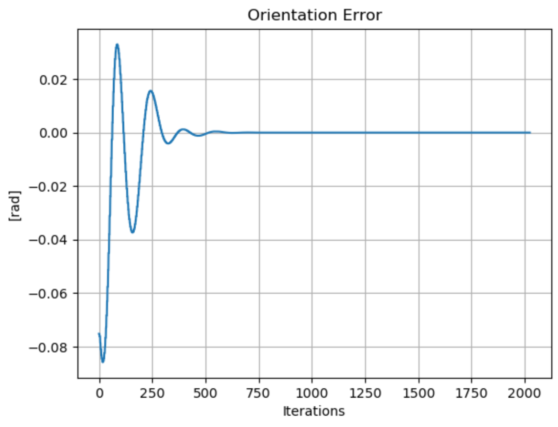
(a) X position error



(b) Y position error



(c) Z position error



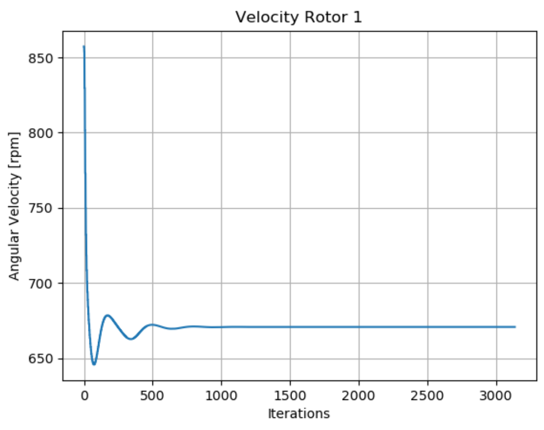
(d) Orientation error

Figure 6: Position and control errors in the sideways motion

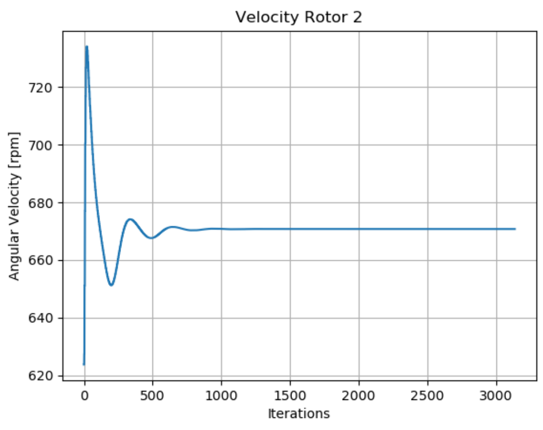
As we can see also for the position errors, many more ripples are showing. *The X position error* since it is the direction over which the drone moves the most it is also the one with the highest amplitude in terms of error. It is also noticeable the converging dynamic of the drone orientation. To reach this setpoint, the robot has to tilt and later get back in the flat orientation position. This is why we can now see overshoots in its behavior. As we can see from the steepness of the angle graph's first overshoot, the dynamic of the drone's orientation is faster than the position graphs. This characteristic is fundamental for the correct functioning of the cascade control system we developed.

5.3 Diagonal Motion point

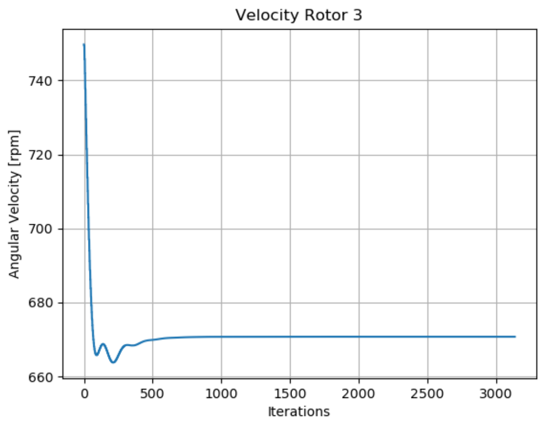
Here are the rotors' velocities:



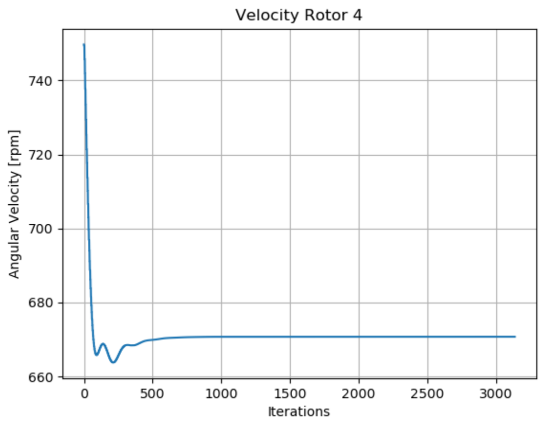
(a) First rotor angular velocity



(b) Second rotor angular velocity



(c) Third rotor angular velocity

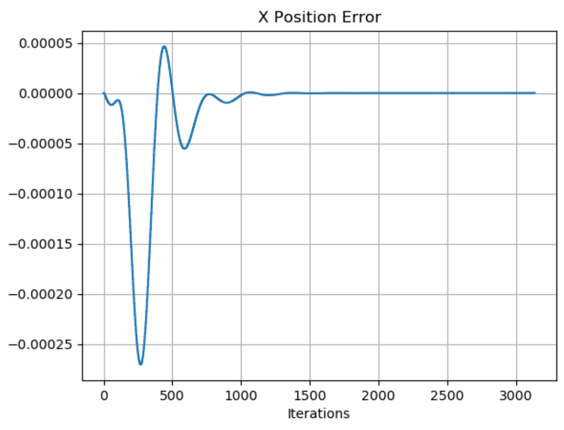


(d) Fourth rotor angular velocity

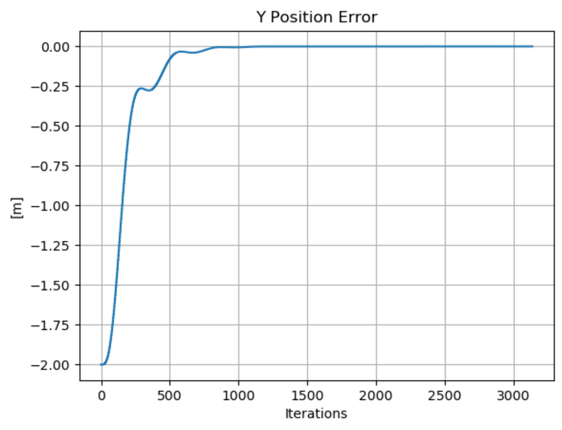
Figure 7: Rotors' velocities in the diagonal motion

As we can clearly see by comparing the amplitude of the curves we can easily tell how the *rotor 1* produced a higher initial thrust compared to the other motors to quickly tilt the drone to guide it to the right spot up. Though, the rotation speed was similar among all the rotors because the drone had to acquire altitude to reach the final position.

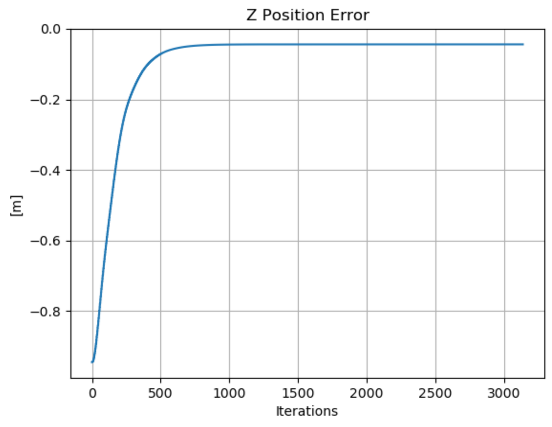
Here are both the control errors in translation and orientation:



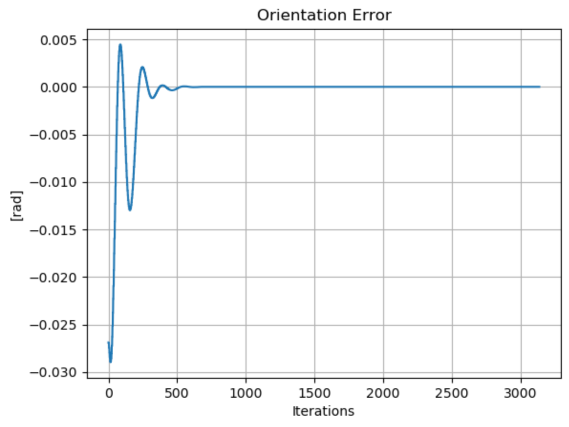
(a) X position error



(b) Y position error



(c) Z position error



(d) Orientation error

Figure 8: Position and control errors in the diagonal motion

In this case, the orientation dynamic has a similar aspect compared to the previous motion since the angle has to tilt in this case too. The same comparison holds also for the dynamic of the previous *X position error* and the current *Y position error*. The **Z position error** is smooth just like in the hoovering flight given a similar task.

5.4 Conclusions

Given the results obtained in the simulation, we can conclude that the designed PD controllers work correctly when it comes to assigning set points for the drone to reach. One advantage of using a PID controller over a PD controller would be that the integral term in the PID control signal could help to eliminate steady-state error in the system. This is because the integral term is proportional to the accumulated error over time, and can therefore help to drive the drone towards the setpoint even if the error does not change significantly. This can be particularly useful in situations where the system is subject to external disturbances like the wind that could cause the error to fluctuate around the setpoint, as the integral term can help to cancel out these disturbances and maintain the system close to the setpoint. Another advantage of using a PID controller is that it can provide more precise control of the system than a PD controller. This is because the derivative term in the PID control signal can help to anticipate and compensate for changes in the error over time, which can lead to faster and more stable convergence to the setpoint. Those kinds of regulators (PD, PID) are the most straightforward linear regulator and is most commonly used

in industrial applications due to their simplicity. Indeed, PID controllers allow for direct tuning of its proportional, integral, and derivative parameters to optimize the controller's performance, which allows for greater flexibility in adapting the controller to the specific needs of the system being controlled. As we can see from Fig 4, 6, 8 the PD controller generates oscillatory behaviours in minimizing the error.
More sophisticated controllers such as Optimal Controllers (LQR) or Model Predictive Control (MPC) could be used if the oscillatory behaviour is not acceptable.