AUVE Lab - Quadrotor control

Raffaele Pumpo, Emanuele Buzzurro

Released: December 15, 2023

1 Introduction

The primary objective of this lab is to achieve stabilized flight of a quadrotor through the implementation of a cascaded control law. The simulation environment utilized for this purpose is GAZEBO.

This report comprehensively details the control strategy employed, elucidates the attained achievements using visual aids such as curves and tables, and provides the necessary code in the form of the `uavcontrol.cpp` file.

Two launch files are provided for distinct simulations:

- `uav_simulation.launch.py:` for simulating the quadrotor
- `uav_simulation_fixed.launch.py:` for simulating the drone attached to a ball joint in its center

It is crucial to note that the latter simulation aids in testing the attitude control law in an initial phase, albeit with modified quadrotor dynamics. Any gains obtained in this fixed simulation might not be directly applicable to the unconstrained drone.

Furthermore, it was essential to run Gazebo for the project to execute correctly, with the closure of Gazebo marking the end of the program. Throughout the lab, vectors and quaternions are represented using the Eigen library, accessible at https://eigen.tuxfamily.org.

2 Description of the system

The quadrotor is described by Fig. 1. The axis of the body frame are such that $\mathbf{x}_b$ is pointing forward and $\mathbf{z}_b$ up; $\mathbf{y}_b$ is then pointing left.

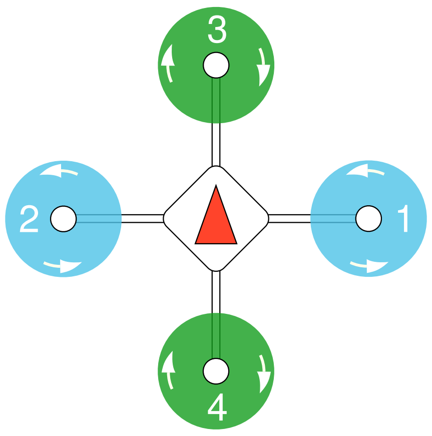


Figure 1: Scheme of the quadrotor (source: docs.px4.io)

The useful variables for this lab are the following (and recalled at the beginning of the "uavcontrol.cpp" file).

- Arm length : 0.17 m
- Rotor thrust gain: $k_t = 5.5\text{e-}6$ N per rpm^2
- Rotor drag gain: $k_d = 3.299\text{e-}7$ Nm per rpm^2
- Drone mass: 1 kg

3 Writing a control law for a quadrotor

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

In the code implementation, we can use

- uav.getPosition() to get the 3D position of the drone
- uav.getorientation() to get the orientation of the drone as a unit quaternion.
- uav.getLinearVelocity() or uav.getAngularVelocity() to respectively get the linear and the angular velocity of the drone.

In order to establish a control law, we followed those three step:

- Write a mixer matrix to compute torques and thrust force of the quadrotor, given a motor rotation speed.
- Write an attitude controller, using a control law based on quaternions.
- Write a position controller.

We can illustrate the controller as following:

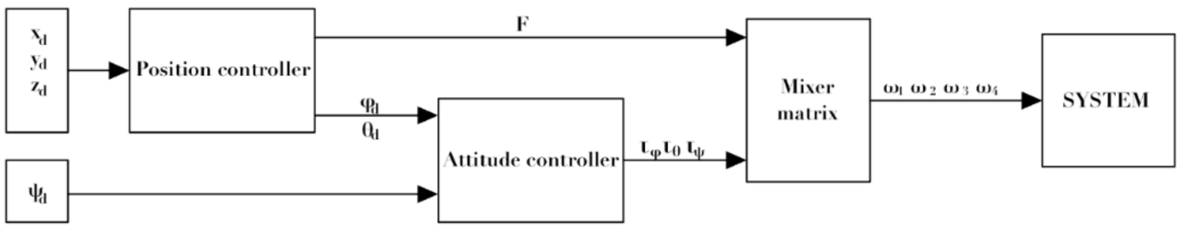


Figure 2: Controller

3.1 Mixer

The input vector that drives the dynamics of the system is represented by a 4-dimensional vector:

$$\begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}$$

where:

- F denotes the thrust produced by the rotors, governing both the altitude and velocity along the z-axis.
- τ_ϕ represents the torque generated by the rotors positioned along the x-axis (motors 3 and 4), influencing the roll angle (torque around the x-axis).
- τ_θ signifies the torque produced by the rotors along the y-axis (motors 1 and 2), controlling the pitch angle (torque around the y-axis).
- τ_ψ is the resultant torque from all four rotors, regulating the yaw angle (torque around the z-axis).

To transform this control space vector into rotor velocities, a mixer matrix is employed. In accordance with the body-fixed reference frame, the equations expressing the control inputs in terms of rotor velocities enable the construction of the Mixer matrix. These equations are as follows:

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

This set of equations, can also be expressed in the following matrix form:

$$\begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} kt & kt & kt & kt \\ -ktl & ktl & 0 & 0 \\ 0 & 0 & -ktl & ktl \\ -kd & -kd & kd & kd \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

By inverting the mixer matrix, we obtain the inverse of the mixer matrix M^{-1} as follows:

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = M^{-1} \begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \quad (3)$$

In order to express rotor velocities in terms of control inputs, we need to invert the relationship. The resulting matrix facilitates this transformation.

Additionally, it's important to note that the velocities were calculated squared in the preceding equations. To obtain the actual rotor velocities, it is necessary to apply the square root to each velocity input. This step ensures that the control inputs are correctly translated into the rotor velocities governing the quadrotor's motion.

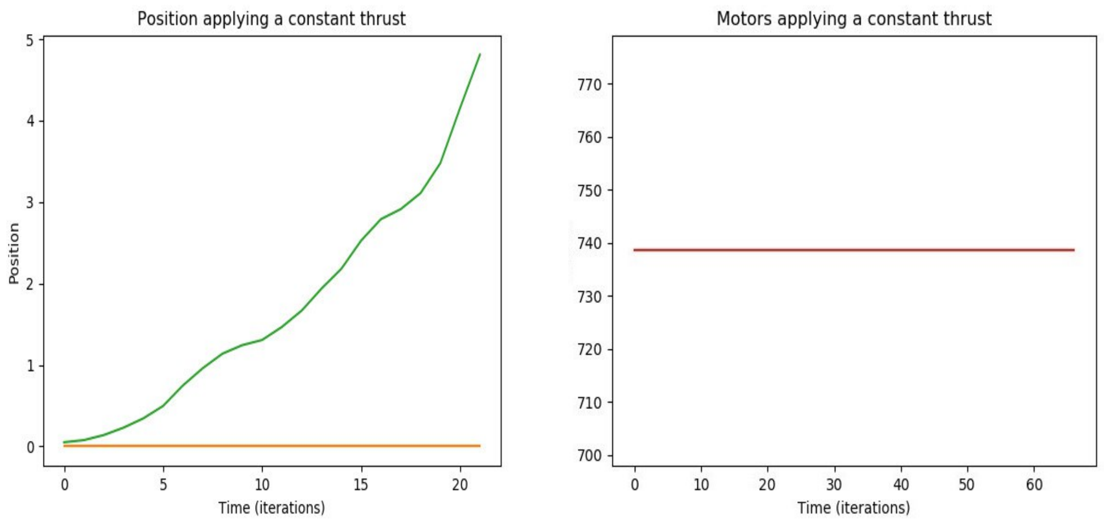
3.1.1 Mixer Test

The mixer undergoes a basic test in the script's test_mixer function. This test involves setting a specific thrust value and observing the resulting motor commands. This process ensures that the mixer functions as intended and provides the expected response to control inputs.

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

$$\mathbf{u} = \begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 12 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The resulting position, applying a constant thrust, and the thrust itself applied by the motors are shown in the figure below:



(a) Positions (b) Motors

Figure 3: Positions and Motors values obtained with F = 12

As long as the drone's mass is 1 Kg, by applying $F = 12\,\mathrm{N}$, the drone started hovering at a determined altitude along the z-axis.
Applying an input test constant, as a thrust , we can easily see from the figure that only the z-position changes in time, x,y are constantly 0, and all the motors have equal and constant value, as expected for a constant input with only thrust force, we can conclude that the mixer matrix has been built correctly.

3.2 Attitude controller

The Attitude controller, positioned in the inner loop, assumes the crucial role of directing the drone's orientation through the implementation of a PD controller driven by torques. Notably, the controller utilizes quaternion error as a key parameter, necessitating an understanding of both quaternion multiplication and inversion.

To quantify the disparity between the desired orientation $\mathbf{q}_d$ and the current orientation $\mathbf{q}$, the quaternion error $\tilde{\mathbf{q}}$ is defined as:

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

The ensuing control law serves as a means to converge towards the target orientation:

$$\tau = -\mathbf{K}_d\mathbf{w} + \mathbf{K}_p \text{sign}(\tilde{\mathbf{q}}_r)\tilde{\mathbf{q}}_i$$

where:

- τ represents the drone's input torque.
- $\mathbf{w}$ signifies the current angular velocity in the body frame.
- $\tilde{\mathbf{q}}_r$ denotes the real scalar part of the quaternion error.
- $\tilde{\mathbf{q}}_i$ corresponds to the imaginary part of the quaternion error, represented as a vector.
- $\mathbf{K}_p$ and $\mathbf{K}_d$ are diagonal gain matrices.

This control strategy, incorporating proportional and derivative gains, orchestrates the drone's response to discrepancies in orientation, ensuring a dynamic and accurate adjustment towards the desired attitude.

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.

3.3 Position controller

The goal of the position controller block is to ensure the stability of the quadrotor and the reachability of the desired position.
This block can be considered as the outer loop of the controller, indeed it generates for each time instant the required amplitude of the force and the desired angle configuration, the latter input of the inner loop.
Then, the primary objective of the position controller is to determine the optimal force magnitude, a scalar value that governs the thrust generated by the rotors. This scalar value subsequently serves as the input for the mixer matrix. Concurrently, the orientation of the force becomes an input for the lower-level attitude controller. In essence, the position controller strategically regulates the thrust magnitude, while the force orientation is directed to the lower-level controller responsible for managing the UAV's attitude.
For this controller, we will use a simple PD control with compensation of gravity.

$$\mathbf{f} = \mathbf{K}_d(\mathbf{v}_d - \mathbf{v}) + \mathbf{K}_p(\mathbf{p}_d - \mathbf{p}) - m\mathbf{g} \tag{1}$$

where

- $\mathbf{f}$ is 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

Once, the vector force $\mathbf{f}$ is obtained, the amplitude, input of the mixer block is obtained simply doing its norm.
Instead the orientation, input for the attitude controller, describes the orientation that the drone has to reach in such a way that it can successively generates the thrust force considered along the z-axis.
Then the Rotation desired is computed as following:

$$R_d = \left[(y \times \frac{\mathbf{f}}{\|\mathbf{f}\|})^T,\ (\frac{\mathbf{f}}{\|\mathbf{f}\|} \times (y \times \mathbf{f}))^T,\ \left(\frac{\mathbf{f}}{\|\mathbf{f}\|}\right)^T \right]$$

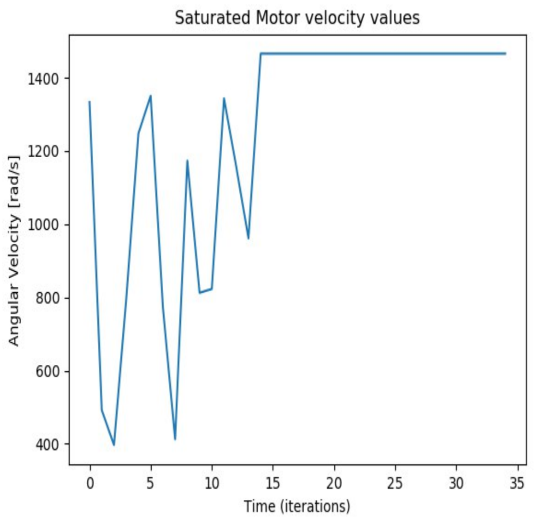
Doing this we obtain the rotation desired considering that the force generated by the PD controller must be along the z-axis, the x is computed before as the cross product between the $y = [0 \quad 1 \quad 0]$ that is the y-unit vector of the fixed frame of the drone, and then normalizing the x-vector obtained and doing again a cross product with the z-vector we will obtain the orthogonal frame corresponding to the desired one.
Finally, the Rotation matrix is converted in quaternion for a better continuous controller.

4 Saturation of the motors

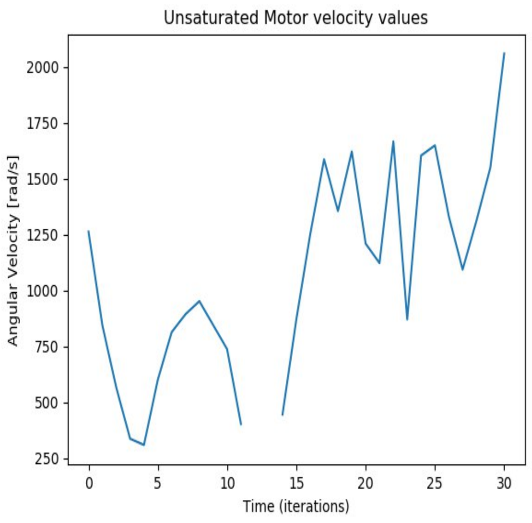
Motor saturation is the process of constraining the control signals sent to each motor to prevent them from exceeding safe operational limits. This is particularly important in scenarios where the control algorithm generates commands that, if unbounded, could lead to overdriving the motors or causing instability.
Then, the motor saturation, which refers to limiting the output within a certain range, is a critical aspect of this control strategy.
In the lab this constraint has been considered in the following way:

- If a motor command is less than 0, it is set to 0. This ensures that the motor output remains non-negative, preventing issues associated with negative thrust.
- If a motor command exceeds the upper limit of 1466, it is capped at 1466. This upper limit represents a saturation threshold beyond which the motor output is constrained, safeguarding against excessive thrust that might compromise the quadrotor's stability or exceed hardware capabilities.

The behaviour of each motor can be seen in the following images:



(a) Saturated



(b) Unsaturated

Figure 4: Joint values of the Cosserat rod with implicit integrator

We can easily see that the value of each motor saturated is always a positive value and below 1466, maximum value that the motors can produce, unlike to the unsaturated ones that is largely above the maximum value, and we can see also some problem of jumps in the motors value.

Motor saturation is essential for preventing undesirable behavior in a quadrotor, such as excessive thrust or control signals that could lead to mechanical stress. By incorporating saturation limits, the control system promotes safer and more controlled flight, contributing to the overall reliability and longevity of the quadrotor platform. The specific limits set in the code should align with the physical characteristics and specifications of the quadrotor's motors. Adjusting these limits appropriately ensures optimal performance and mitigates the risk of damage or instability.

5 Trajectory Implementation

In the code two type of trajectories have been implemented, one in which the drone has to reach a desired position and another in which it has to follow a circular trajectories.
The choice between which one the drone has to follow can be selected through the parameter **method**, as following:

- false = Desired position trajectory
- true = Circular trajectories

The generateCircularTrajector function encapsulates a method for generating a circular trajectory in three-dimensional space. This trajectory is characterized by a circular path along the xy-plane, with a fixed height in the z-axis. The key parameters influencing the trajectory are the radius (r) of the circle and the angular velocity (ω).
The mathematical idea of the circular trajectory is in parametric equations describing the position of a point moving along the circumference of a circle over time.

The equations governing the circular trajectory in the xy-plane are as follows:

$$x(t) = r \cdot \cos(\omega t)$$
$$y(t) = r \cdot \sin(\omega t)$$
$$z(t) = \text{constant} \quad \text{(Fixed height)}$$

Where:

- $x(t)$, $y(t)$, and $z(t)$ represent the desired position of the point in three-dimensional space at time t.
- r is the radius of the circle.
- ω is the angular velocity, determining the rate of rotation.

Then, the desired velocity vectors at a given time (t) using the parametric equations. Specifically:

$$V_x(t) = -r \cdot \omega \cdot \sin(\omega t)$$
$$V_y(t) = r \cdot \omega \cdot \cos(\omega t)$$
$$V_z(t) = 0.0$$

Finally, different parameters can be changed:

- Radius (r): Determines the size of the circular trajectory. Adjusting this parameter modifies the distance of the trajectory from the origin.
- Angular Velocity (ω): Dictates the speed at which the circular trajectory is traversed. A higher angular velocity results in a faster rotation.

We can see the circular trajectory of the drone as following:

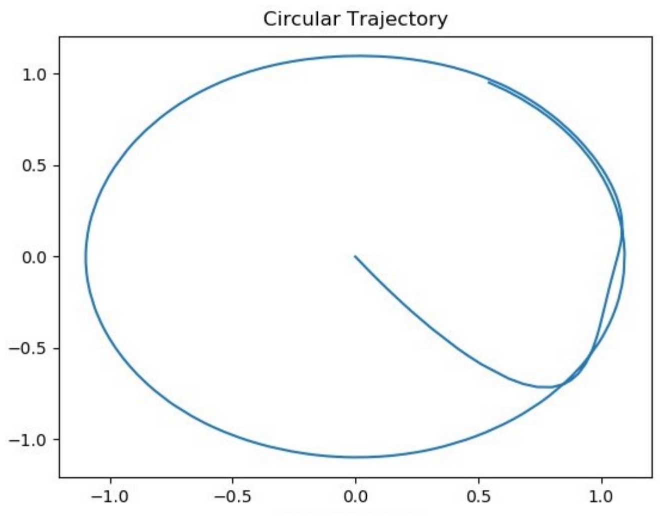


Figure 5: Circular trajectory

We have considered the initial position as the center of the circle, indeed the robot firstly reaches the bound of the circle then starts its trajectories.

6 Results and Conclusions

Our investigation into quadrotor control dynamics unveils a comprehensive understanding through three illuminating graphs, each capturing distinct facets of orientation and position error evolution.

The initial graph depicts the repercussions of employing poorly tuned gains in the quadrotor control system. The pronounced presence of high-frequency oscillations over time reveals the system's struggle to converge to zero orientation error, as shown in the figure below:

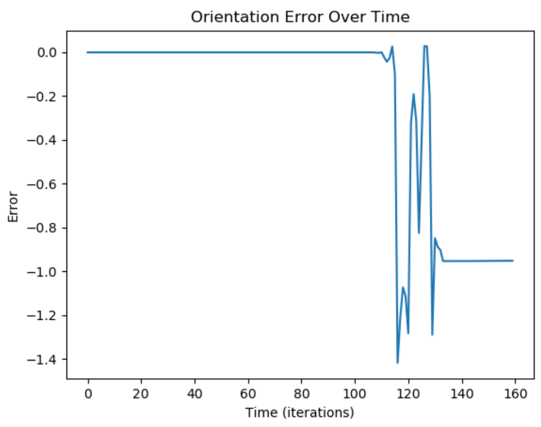


Figure 6: Orientation error with bad tuned gains

This vivid illustration serves as a poignant reminder of the critical interplay between gains and stability. The excessive oscillations underscore the need for meticulous tuning to strike a balance between system responsiveness and robustness.

A stark contrast emerges in the second graph, where the quadrotor's orientation error unfolds over time with well-tuned parameters:

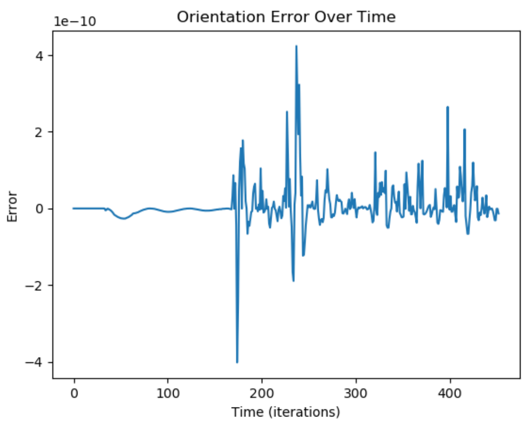


Figure 7: Orientation error with well tuned gains

The significant reduction in oscillations and the near-zero convergence of error underscore the transformative power of precise parameter tuning. This outcome not only validates the efficacy of the tuning process but also highlights the system's ability to achieve stable and accurate orientation control.

Going into the realm of position control, the third graph charts the trajectory of the quadrotor as it navigates towards and maintains the desired position:

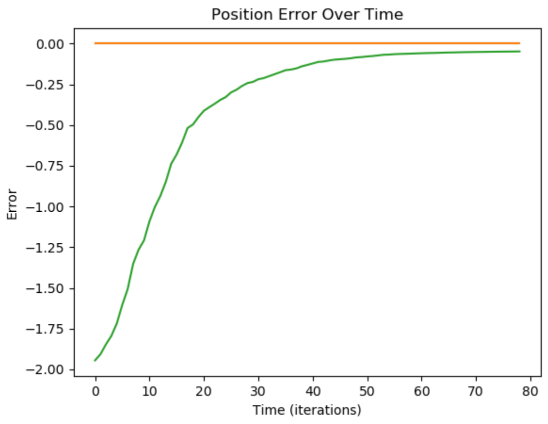
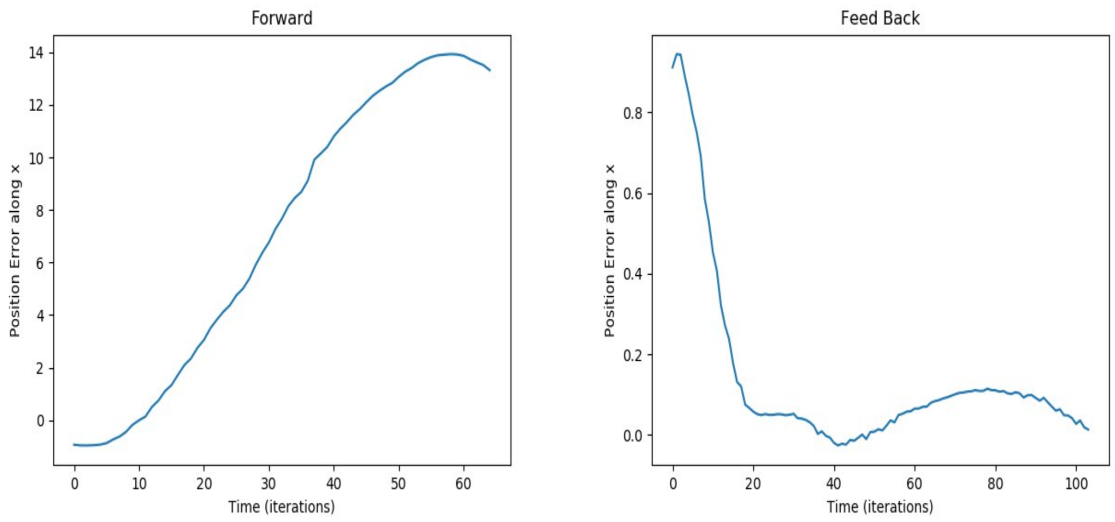


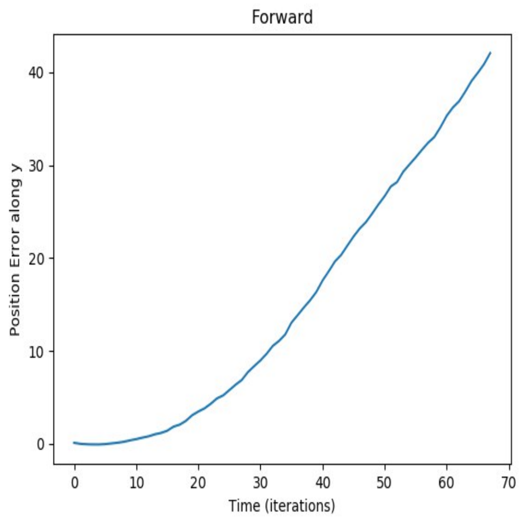
Figure 8: Position error

This plot emphasizes the cascading impact of well-tuned parameters, not just on orientation control but also on the seamless convergence to a predefined position. The steady reduction in position error over time signifies the robustness and precision afforded by a finely tuned control system, showcasing the system's adaptability in achieving spatial accuracy.
Adding another layer to our exploration, we introduce a double plot illustrating the position error on a circular trajectory:

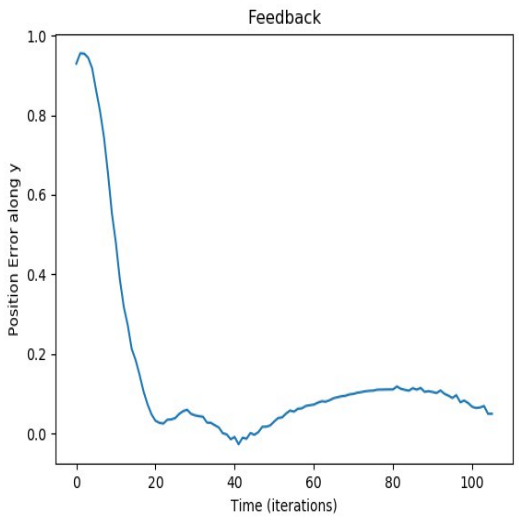


(a) X position feed-forward

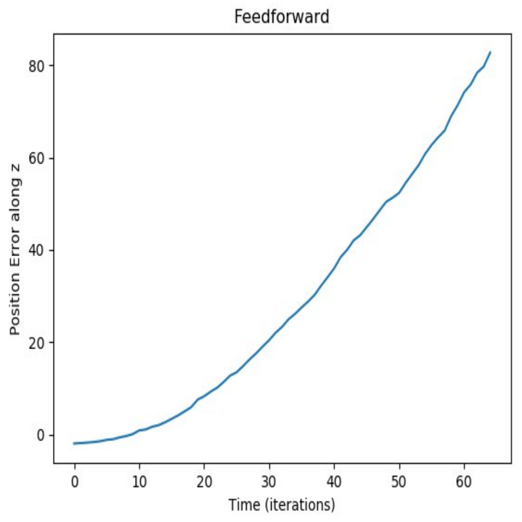
(b) X position feed-back



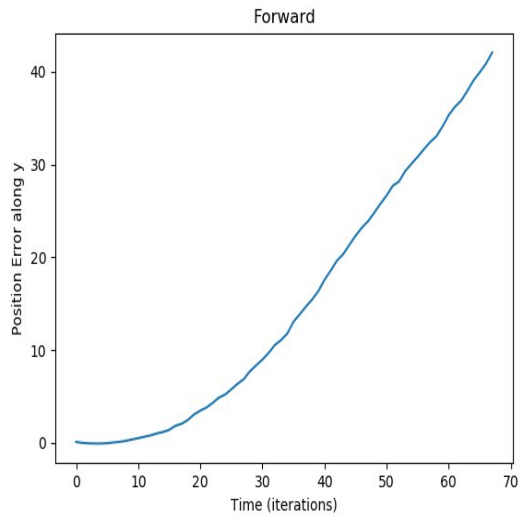
(a) Y position feed-forward



(b) Y position feed-back



(a) Z position feed-forward



(b) Z position feed-back

Figure 11: Comparison between feed-forward and feed-back position error

This dual representation captures the stark contrast between employing a feed-forward approach, resulting in divergence from the correct solution, and a feed-back approach, showcasing correct convergence.
The first segment of the double plot sheds light on the consequences of adopting a feed-forward approach in the context of a circular trajectory. The position error over time reveals a noticeable divergence from the intended solution. This discrepancy serves as a cautionary observation, highlighting the challenges associated with relying solely on feed-forward mechanisms in dynamic scenarios. The graph accentuates the need for a more adaptive and feedback-driven control strategy to mitigate deviations from the desired trajectory.
Contrastingly, the second plot demonstrates the effectiveness of a feed-back approach in rectifying the position error on the same circular trajectory. Here, the system exhibits a commendable convergence towards the correct solution. The consistent reduction in error over time underscores the value of real-time feedback mechanisms in steering the quadrotor towards accurate trajectory tracking. This success story reinforces the adaptability and precision that a well-designed feedback control system brings to the table.

In summary, our exploration of quadrotor control through these graphs underscores the fundamental role of parameter tuning in shaping system performance. The journey from poorly tuned gains with erratic oscillations to well-tuned parameters with stability and precision paints a vivid picture of the transformative effects of tuning.