© June 2023| IJIRT | Volume 10 Issue 1 | ISSN: 2349-6002

PID Flight Controller for Quadcopter and it’sSimulation

Yash Pawar, Sourabh Prasad, Vedant Wakhare, Kunal Patil, S. D. Bharkad
Department of Electronics and Telecommunication Engineering Government College of Engineering, Aurangabad

Abstract—**Quadcopters, also known as drone, are popular with unmanned aerial vehicles (UAVs). They are widely used in many applications due to their small size and high security. In this article, RC quadcopter is designed and built using PID (Proportional Integral Derivative) controller and simulated using Godot Engine. The simulation in Godot allows for real- time visualization of the quadcopter’s behavior, providing an interactive platform for testing and analyzing different control strategies. The paper involves creating the quadcopter model, implementing the PID controller algorithm, and integrating sensor feedback and actuator dynamics into the simulation. The simulation environment enables the exploration of various flight scenarios, including response to external disturbances and changes in environmental conditions. The pitch, roll, yaw and position response of the quadcopter is obtained and a PID controller is used to stabilize the system response. The operation and performance of the quadcopter and combat algorithms were tested in the simulator and the desired results were obtained.**

I. INTRODUCTION

In recent years, modeling and simulation of quadcopters based on PID controllers has become very popular due to its many applications in many fields such as aerial photography, surveillance, search and rescue and package delivery. A quadcopter, also known as a quadrotor, is an unmanned aerial vehicle (UAV) characterized by its four propellers that generate lift and control its movement.

This presentation is designed to demonstrate the basics of modeling and simulation of a quadcopter using a PID controller. It highlights the importance of understanding quadcopter dynamics, developing control algorithms, and using environmental simulation to analyze and optimize its performance. More research on

Quadcopters can be found in many documents in this field. Design and develop a quadcopter model using a PID controller. The IMU is used to determine the system orientation. In this study, the mathematics of the quadcopter is done by using the MATLAB Simulink model, which uses a PID controller to stabilize the operation of the quadcopter. Quadrotor field slope, roll and yaw slope are taken from MATLAB Simulink.

In our paper, there will be sensors that provide feedback to the required PID controller in the theoretical drone model. Then our flight algorithm is used by the PID controller, and the drone can fly under the control of the user. In general, like drone racing, users need to be fluent, snappy, fast, etc. needs such controls. These can be achieved with the PID setting.

II. OBJECTIVE

The objective of this paper is to develop a comprehensive modeling and simulation framework for a quadcopter using a PID (Proportional-Integral-Derivative) controller. The aim is to achieve a deep understanding of quadcopter dynamics, design an effective control algorithm, and simulate its behavior in a virtual environment.

1) Mathematical Modeling: The first objective is to develop an accurate mathematical model that describesthe physical dynamics of the quadcopter. This involves formulating the equations of motion, considering factors such as thrust, drag, weight, and torques. The model should capture the quadcopter’s translational and rotational motion, as well as its interactions with the environment.
2) PID Controller Design: The second objective is to design a PID controller that can stabilize the quadcopter and enable precise control of its position, orientation, and velocity. This involves determining suitable gains for the proportional, integral, and derivative components of the controller. The goal is to develop a control algorithm that can effectively counteract disturbances and track desired setpoints accurately.
3) Simulation Implementation: The next objective is to implement the quadcopter model and PID controller

in a simulation framework. This involves developing simulation software tools using GODOT Engine and GDscript, to create a virtual environment where the quadcopter's behavior can be simulated and analyzed. The simulation should replicate real-world dynamics, including sensor feedback, actuator responses, and environmental influences.

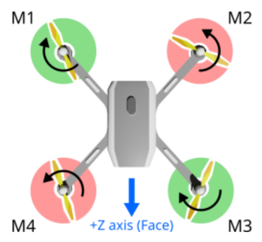
4) Performance Evaluation: The fourth objective is to evaluate the performance of the quadcopter and the PID controller through simulation experiments. This includes assessing stability, responsiveness, tracking accuracy,and robustness to disturbances. By analyzing the simulation results, we can gain insights into the strengths and limitations of the PID controller design, identify areas for improvement, and refine the control strategy.
5) Optimization and Iteration: The final objective is to optimize the PID controller parameters and iterate on the model and simulation framework based on the evaluation results. By fine-tuning the gains, adjusting control strategies, and refining the model, we aim to achieve optimal quadcopter performance and control accuracy.

Through the successful completion of these objectives, this paper aims to advance the understanding and capabilities of quadcopter modeling and control using PID controllers. The outcomes will contribute to the development of more efficient and reliable quadcopter systems, enabling their application in various fields such as aerial surveillance, mapping, inspection, and autonomous operations.

III. LITERATURE REVIEW

Quadcopter

The flight of a drone is achieved through the rotation of four vertical-axis propellers (rotors) positioned at the corners of a square configuration. According to Newton's second law, when the drone hovers at a constant altitude, the upward thrust generated by the rotors balances the downward grav- itational force acting on the drone's airframe. By adjustingthe total power of the rotors, the drone can accelerate or decelerate vertically, causing a change in its upward thrust and consequently altering its altitude [4]. When the thrust force exceeds the force of gravity, the drone experiences an initial acceleration. However, during this acceleration, the potential energy of the drone increases, requiring the motors to perform work in supplying the additional energy. Consequently, the rate of ascent is constrained by the ability of the motors to deliver this energy effectively, which limits the drone's rate of climb.



In this quadcopter configuration, the rotors are divided into two sets: red rotors rotating counterclockwise and green rotors rotating clockwise. By having these two sets of rotors rotating in opposite directions, the total angular momentum of the quadcopter remains zero. Angular momentum, similar to linear momentum, is calculated by multiplying the angular velocity by the moment of inertia. The moment of inertia is a property that relates to rotation and is akin to mass in linear motion. Although it can be complex, for our purposes, understanding that angular momentum depends on rotor speed is sufficient [7]. Every movement of the quadcopter is achieved by adjust- ing the spin rate of one or more rotors. This adjustment is made possible by a controller that can increase or decreasethe voltage supplied to each motor. Alternatively, with the assistance of a computer control system, the pilot can simply manipulate a joystick, allowing the computer to handle the rotor adjustments. The quadcopter's onboard accelerometer and gyroscope contribute to flight ease and stability by making precise power adjustments to each rotor. These adjustments en- sure the quadcopter responds to subtle changes and movements as required [7].

Roll: In terms of lateral movement, the quadcopter can shift left or right relative to its front. This movement is commonly referred to as rolling. To roll the quadcopter to the left, the lift generated by the motors on the right side is increased, while simultaneously reducing the lift on the motors located on the left side. Conversely, to roll the quadcopter to the right, the opposite action is taken. The lift is increased on the motors on the left side while decreasing the lift on the motors on the right side. These differential adjustments in lift between the left and right sides enable the quadcopter to roll and move laterally as desired.

Pitch: To achieve forward or backward movement, the quad- copter can pitch either towards or away from the observer. When the drone pitches forward (moves towards the observer), the power supplied to the rear motors is increased. This creates a net forward force that causes the nose of the drone to pitch downward. To maintain the conservation of angular momentum, the power applied to the two front motors is simultaneously decreased. This differential adjustment in power between the rear and front motors allows the quadcopter to pitch forward and move in the desired direction. Conversely, to pitch the drone backward (move away from the observer), the exact opposite action is taken. The power applied to the rear motors is reduced, while the power to the front motorsis increased, resulting in a net backward force and a pitch upward motion of the drone.

Yaw: To pivot or turn the quadcopter from the left or right, it can perform a yaw motion, which involves rotating around its center axis. When the drone yaws clockwise, the lift on the motors that rotate in an anticlockwise direction is increased. Simultaneously, the lift on the motors rotating clockwise is decreased. This adjustment is made to ensure that the net force in both the upward and downward directions remains balanced. By increasing the lift on the anticlockwise rotating motors and decreasing the lift on the clockwise rotating motors, an anticlockwise torque is generated. This torque allows the quadcopter to rotate in a clockwise direction while conserving angular momentum.

Mathematical Form

The roll, pitch and yaw movement equations for the quadcopter. [1]

$$\ddot{\phi} = \frac{I_{yy} I_{zz}}{I_{xx}} \vartheta\psi - \frac{J_{tp}}{I_{xx}} \vartheta\omega + l \frac{U_2}{I_{xx}}$$

$$\ddot{\vartheta} = \frac{I_{xx} I_{zz}}{I_{yy}} \vartheta\psi - \frac{J_{tp}}{I_{yy}} \vartheta\omega + l \frac{U_3}{I_{yy}}$$

$$\ddot{\Phi} = \frac{I_{xx} I_{yy}}{I_{zz}} \vartheta\psi - l \frac{U_4}{I_{zz}}$$

This then influences the control architecture of the quadcopter using PID blocks.

PID Controller

In various industrial control systems, the proportional-integral-derivative (PID) controller is widely employed due to its versatility and ability to be optimized for specific control systems. The PID controller is the most commonly used algorithm in controller design and is extensively utilized across various industries [3]. The PID controllers found in industrial applications can be categorized into three basic types: parallel, serial, and mixed configurations. The design velocity algorithm, also known as the incremental algorithm,is often employed for implementing the PID controller [3].

PID controllers offer a comprehensive set of dynamics that enable effective control in real-world applications. The derivative mode provides a fast response to changes in the controller input, the integral mode adjusts the control signal to reduce the error, and the proportional mode ensures suitable action within the control error range to eliminate oscillations. By incorporating the derivative mode, system stability is improved, allowing for increased gain (K) and decreased integral time constant ($T - i$), which enhances the speed of the controller's response [8].

PID controllers are extensively utilized in the process industry, with a significant majority of control systems relying on them. Reports suggest that approximately 98% of control loops in the pulp and paper industries employ single-input single-output PI controllers, while more than 95% of controllers in process control applications are of the PID type [8]. The PID controller offers the advantages of proportional, derivative, and integral control actions combined, making it a popular choice for control systems.

IV. PROPOSED METHOD

The algorithm uses a set-point and process variable to calculate an error, which is then used to determine the control output.

The control output is calculated by combining proportional, integral, and derivative terms, which are weighted by respective gain values (K_p, K_i, and K_d). The algorithm involves calculating the PID for all the movements i.e thrust, roll, pitch and yaw, then combining them using motor mixing algorithm and we get the desired output. This control loop iteratively updates the control output based on the current error and previous error. The control output is applied to the quadcopter's motors to adjust their speeds and achieve the desired position or orientation. Proper tuning of the PID gains is crucial for stable and responsive control.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \quad (1)$$

PID Algorithm

The purpose of a PID controller is to control a system by adjusting an output variable based on the difference between a desired setpoint and the actual value of a process variable. The controller uses three terms to calculate the output: proportional, integral, and derivative.

```
class PID:
    var GAIN
    var error
    var previous_error
    var error_derivative
    var error_integral
    var pid

  func _init(_GAIN):
    GAIN = _GAIN
    previous_error = 0.0
    error_integral = 0.0

  func calculate_PID(_error, delta):
    error = _error
    error_derivative = (error - previous_error) /
     delta
    error_integral += error * delta
    previous_error = error

    pid = error * GAIN.x + error_derivative * GAIN.z
      + error_integral * GAIN.y
    return pid
```

Listing 1. GDScript class for PID algorithm.

The implementation starts with a PID class that has five instance variables: `GAIN`, `error`, `previous_error`, `error_derivative`, and `error_integral`. The `GAIN` variable is a Vector3 that contains the coefficients for the proportional, integral, and derivative terms. The error variable represents the difference the setpoint (desired value) and the process variable. The `previous_error` variable is used to calculate the derivative term. The `error_derivative` variable is the derivative term of the `error`, and the `error_integral` variable is the integral term of the `error`.

The `_init()` function is called the when the PID object is created, and its sets the `GAIN`, `previous_error`, and `error_integral` variables to their initial values.

The `calculate_PID()` function is where the actual PID calculation is performed. It takes two arguments: `_error`, which is the current error between the setpoint and the process variable, and `delta`, which is the time elapsed since the last calculation. The function first updates the `error_derivative` and `error_integral` variables by calculating the difference between the current error and the previous error, and addign the current error multiplied by `delta` to the `error_integral`. The `previous_error` variable is then updated to the current error.

Finally, the `pid` variable is calculated by multiplying the `error` by the proportional coefficient (`GAIN.x`), adding the `error_derivative` multiplied by the derivative coefficient (`GAIN.z`), and adding the `error_integral` multiplied by the integral coefficient (`GAIN.y`). This value is returned by the function as the output of the PID controller.

Control Architecture

The software architecture of the control program implemented on the micro-controller as seen in [10] can be further modified as per this papers requirements. The pitch, roll, and yaw are the three axes that control the orientation and movement of a drone. The pitch axis controls the forward and backward movement of the drone, the roll axis controls the left and right movement of the drone, and the yaw axis controls the rotation of the drone around its vertical axis.

To use the PID controller to control a drone's pitch, roll, and yaw movements, we need to measure the error between the desired orientation of the drone and its actual orientation.

For the pitch axis, the error would be the difference between the desired pitch angle and the actual pitch angle. Similarly,for the roll axis, the error would be the difference between the desired roll angle and the actual roll angle. For the yaw axis, the error would be the difference between the desired yaw rate and the actual yaw rate.

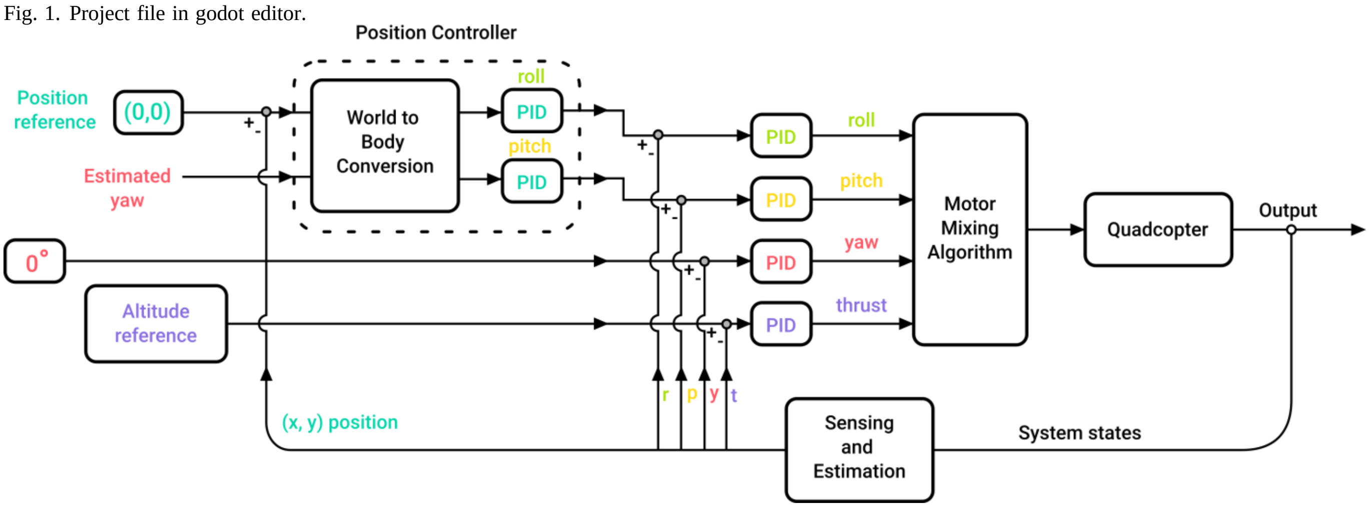
We can then use the `calculate_PID()` function to calcu- late the output for each axis. The output values can be sentto the drone's flight controller to adjust the motor speed and control the pitch, roll, and yaw movements of the drone. The PID controller continuously monitors the error and adjusts the output values to minimize the error and maintain the desired orientation of the drone.

Desired Drone State

```
extends Spatial
var ds = Vector3.zero
var dyaw = 0.0

const XSPEED = 4
```

Fig. 1. Project file in godot editor.



```
    cosnt YSPEED = 3
    const ZSPEED = 4
    const YAWSPEED = 1

    func _ready():
        pass

    func _process():
        # Controller input
        ds.x = Input.get_axis("right", "left") *
XSPEED
        ds.y = -Input.get_axis("up", "down") *
YSPEED
        ds.z = -Input.get_axis("forward", "backward
") * ZSPEED
        dyaw = Input.get_axis("yaw_right", "yaw_left
") * YAWSPEED

        if ds != Vector3.ZERO or dyaw != 0.0:
            self.translate(ds * delta)
            self.rotate(get_transform().basis.y.
normalized(), dyaw * delta)

        if self.global_transform.origin.y < 0.2:
            self.global_transform.origin.y = 0.2
```

Listing 2. Script for taking in user input.

The script extends the `Spatial` class, which is base class for spatial nodes in Godot Engine. Variables are declared and initialized:

- `ds` is a `Vector3` variable initialized to `(0, 0, 0)`, representing the change in position along the *x*, *y*, and *z* axes.
- `dyaw` is a floating-point variable initialized to `0.0`, representing the change in rotation around the y-axis (yaw).

- `XSEEPD`, `YSPEED`, `ZSPEED` are constants representing the speed factors for movement and rotation.

The `_ready` function is called when the script and its associated node are rady for use. In this case, the function is empty and does nothing. The `_process` function is called every frame to update the script's logic.

The input values from the controller are retrieved and used to update the `ds` vector:

- `Input.get_axis` retrieves the input axis value for the given axis names. In this case, ”right”, ”left”, ”up”, ”down”, ”forward”, and ”backward” are used to get input values along the respective axes.
- The input values are multiplied by the predefined speed constants (`XSPEED`, `YSPEED`, `ZSPEED`) and assigned to the corresponding components of the `ds` vector.

The input value for yaw rotation is retrieved and multiplied by the `YAWSPEED` constant, updating the `dyaw` variable.

If there is any non-zero movement (`ds` is not zero) or rotation (`dyaw` is not zero):

- The node's position is translated (`self.translate`) by the product of the movement vector `ds` and the time elapsed since the last frame (`delta`).
- The node is rotated (`self.rotate`) around the y-axis (`get_transform().basis.y.normalized()`) by the product of the rotation speed `dyaw` and the time elapsed since the last frame (`delta`).

Drone PID Flight Algorithm

```
    extends RigidBody

```

```
onready var desired_drone_state = get_node("root
/World/Desired_Drone_State")

var altitude : PID
var x_axis : PID
var roll_angle : PID

var thrust = 0.0
var roll_x = 0.0
var roll = 0.0

func _ready():
    altitude = PID.new(Vector3(30, 3, 2))
    x_axis = PID.new(Vector3(3, 3, 0))
    roll_angle = PID.new(Vector3(0.5, 0.01,
0.07))

    pass

func _process():
    thrust = altitude.calculate_PID(
desired_drone_state.translation.y - self.
translation.y, delta)
    roll_x = x_axis.calculate_PID(
desired_drone_state.translation.x - self.
translation.x, delta)

    var alpha = roll_x
    var omega = previous_omega + alpha * delta
    var delta_theta = omega * delta + alpha *
delta * delta / 2
    previous_omega = omega

    theta += delta_theta
    roll = roll_angle.calculate_PID((theta *
SCALE) - self.rotation_degrees.z, delta)

    var motorFR = thrust + roll
    var motorFL = thrust - roll
    var motorBR = thrust + roll
    var motorBL = thrust - roll

    var posFR = Vector3(-1, 0, 1)
    var posFL = Vector3(1, 0, 1)
    var posBR = Vector3(-1, 0, -1)
    var posBL = Vector3(1, 0, -1)

    add_force(self.get_transform().basis.y *
motorFR, self.global_transform.origin + posFR)
    add_force(self.get_transform().basis.y *
motorFL, self.global_transform.origin + posFL)
    add_force(self.get_transform().basis.y *
motorBR, self.global_transform.origin + posBR)
    add_force(self.get_transform().basis.y *
motorBL, self.global_transform.origin + posBL)

    pass
```

Listing 3. Drone PID flight algorithm script.

This script extends the `RigidBody` class in the Godot Engine. The functionality of the `RigidBody` class, which is typically used to represent objects with physics therefore, we here use this class to model our virtual drone as we will be using Godot physical engine to simulate real gravity and collisions and apply forces on the drone to pilot it. We declared a variable named `desired_drone_state` and assigned it the reference to a node in the scene graph hierarchy. We have a node with the path "root/World/Desired Drone_ State"_in the scene, and the `get_node` function is used to retrieve a reference to that node. Then we declared variables of type `PID` for controlling the altitude, X and Y axis position, andyaw angle of the drone. `PID` is the custom class we wrote for computing the PID of variables in our flight algorithm. We declared variables to store the thrust, X-axis and Y-axis input, and roll and pitch angle output for the drone, and finally yaw angle. They are initialized with initial values of 0.0.

The `_ready()` function is a function that is automatically called when the script instance is ready to be used. In this func-tion, the PID controllers are initialized with specific parametersby calling the `PID.new()` function. The exact parameters passed to the `PID.new()` function are the variables thrust, X and Y axis movement and the angles yaw, roll, and pitch.

The `_process()` function is automatically called on ev-ery frame update. Within this function, the PID con-trollers are used to calculate control signals for the drone's thrust and roll. Here we calculate the thrust control signal using the altitude PID controller. It sub- tracts the current altitude (Y-coordinate) of the drone (`self.translation.y`) from the desired altitude speci- fied by `desired_drone_state.translation.y` and passes the result to the `calculate_PID()` function of the `altitude` PID controller. The `delta` parameter representsthe time elapsed since the last frame update.

The X-axis roll control signal is calculated in`roll_x` using the X-axis PID controller. We then subtract the current X-coordinate of the drone (`self.translation`) from the desired X-coordinate specified by `desired_drone_state.translation.x` and passes the result to the `calculate_PID()` function ofthe `x_axis` PID controller.

In the next few lines of the script, we calculate the roll angle of the drone based on the roll control signal (`roll_x`). It uses an iterative calculation process where the roll angle (`theta`) is updated based on the change in time (`delta`) and the previous angular velocity (`prevous_omega`). The `delta_theta` represents the change in angle during the current frame update. Finally, we calculate the roll control signal using the rollangle PID controller. It subtracts the current roll angle ofthe drone (`self.rotation_degrees.z`) from the desiredroll angle, scaled by a factor of `SCALE`, and passes the resultto the `calculate_PID()` function.

Simulator

The designed flight algorithm using PID will be implemented and simulated using a physics engine called Godot Engine. Using which we developed a software that simulates realtime realistic drone flight. The user pilots the drone using a remote controller. On this virtual drone we test our quadcopter flight algorithm. Building upon the methods of [5] and [6].

V. RESULTS

During the modeling of the quadcopter, several parameters were considered, including mass, arm length, radius, motor torque, and motor speed. To simplify the modeling process, certain assumptions were made. Firstly, it was assumed that the quadcopter's structure is rigid and symmetric. Secondly, the quadcopter's load-heavy point was assumed to be located at its center of mass. Lastly, the vibration effects in each propeller were neglected.

However, during the design phase, the quadcopter exhibited instability when different values of the proportional, integral, and derivative gains for roll and pitch control were used. Through a process of trial and error, the optimal PID gains for stability were determined. For the roll control, a proportional gain of 1.3, an integral gain of 0.04, and a derivative gain of 18 were found to yield stability. Similarly, for pitch control, a proportional gain of 1.5, an integral gain of 0.05, and a derivative gain of 15 were determined to be stable.

Various behaviors and performance characteristics were evaluated by considering different PID parameter values. These evaluations helped in identifying the specific PID gains that provided stability and desired control performance for the quadcopter.

Behaviour considering only Proportional Gain (P)
When the proportional gain is set to a low value, such as less than 1.3, the quadcopter's response is unstable. This instability

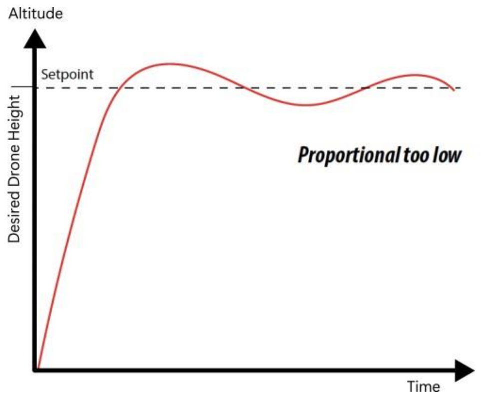


Fig. 2. Altitude graph with low P gain.

could manifest as oscillations or erratic behavior. Increasing the proportional gain to 1.3 stabilizes the quadcopter and improves its performance. Moreover, if the proportional gainis set too high, the quadcopter might exhibit high-frequency oscillations, which could lead to instability.

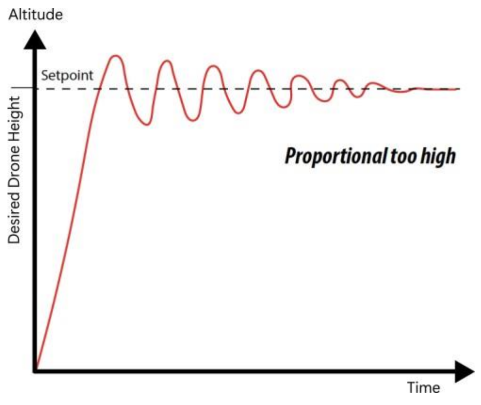


Fig. 3. Altitude graph with high P gain.

Behaviour with Proportional and Integral Gain (I):

Considering only the proportional and integral gains (P and I), the quadcopter remains unstable unless the integral gain is set to a certain value. With an integral gain of 0.04 for roll and 0.05 for pitch, the quadcopter achieves stability. The integral term helps eliminate steady-state errors and ensures the quadcopter can maintain its desired position.

Derivative Gain (D)
Including the derivative gain (D) further enhances the stability and performance of the quadcopter. With a derivative gain of 18 for roll and 15 for pitch, the quadcopter remains stable in the simulation. The derivative term provides damping to the system and helps reduce overshoot and oscillations.

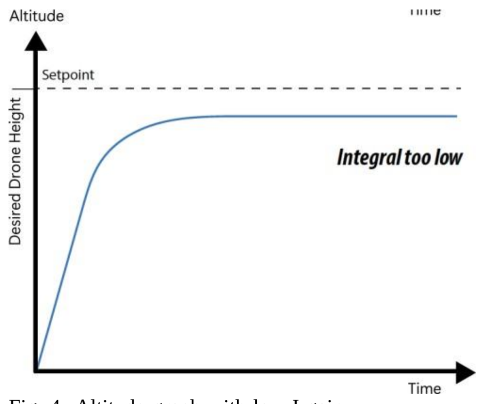


Fig. 4. Altitude graph with low I gain.

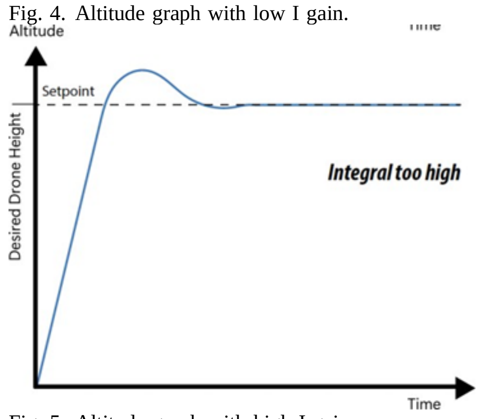


Fig. 5. Altitude graph with high I gain.

VI. CONCLUSION AND FUTURE

Conlcusion

In conclusion, simulation is a critical tool in the development and testing of a PID flight controller for a quadcopter. It allows developers to test and refine the controller's performance in a controlled environment before deploying it on a physical quadcopter. Simulation also enables developers to test the controller's performance under a wide range of flight conditions and to quickly iterate on the design to optimize its performance. It is also an efficient way to identify and troubleshoot any issues in the controller's design and to evaluate the impact of changes to the controller's parameters. Overall, simulation is an essential component in the development and testing of a PID flight controller for a quadcopter, providing a safe and efficient means to optimize the controller's design, test its performance under various flight conditions, and ensure that it meets the required performance specifications.

Future Scope

In conclusion, the future scope of a PID flight controller for a quadcopter is vast, with ongoing research and development efforts focused on improving the performance and capabilities of these controllers. Continued advancements in control algorithms, sensor fusion, autonomous flight, energy efficiency, and miniaturization could lead to more capable and versatile quadcopters in the future. [9]

REFERENCES

[1] Vishwanadhapalli Praveen and Anju S. Pillai. *Modeling and Simulation of Quadcopter usign PID Controller*. IJCTA, 9(15), 2016, pp. 7151-7158.

[2] Jared Maltos. *Simple physics behind the flight of a drone*. Physics 211-F04

[3] K Smriti Rao, Ravi Msihra. *Comparative study of P, PI, and PID controller for speed control of VSI-fed induction motor*. 2014 IJEDR, ISSN: 2321-9939.

[4] S. Jeremia, E. Kuantama and J. Pangaribuan,*Design and construction of remote-controlled quad-copter based on STC12C5624AD*, 2012 In- ternational Conference on System Engineering and Technology (ICSET), Bandung, Indonesia, 2012, pp. 1-6, doi: 10.1109/ICSEngT.2012.6339317.

[5] K. Patel and J. Barve, *Modeling, simulation and control study for the quad-copter UAV*, 2014 9th International Conference on Industrial and Information Systems (ICIIS), Gwalior, India, 2014, pp. 1-6, doi: 10.1109/ICIINFS.2014.7036590.

[6] S. Sohail, S. Nasim and N. H. Khan, *Modeling, controlling and stability of UAV Quad Copter*, 2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT), Karachi, Pakistan, 2017, pp. 1-8, doi: 10.1109/ICIEECT.2017.7916559.

[7] O. Arrieta, D. Campos, J. D. Rojas, M. Barbu and R. Vilanova, *Multi- optimization approach for PID control on Drone roll-pitch orientation*, 2022 23rd International Carpathian Control Conference (ICCC), Sinaia, Romania, 2022, pp. 227-232, doi: 10.1109/ICCC54292.2022.9805919.

[8] A. Iyer and H. O. Bansal, *Modelling, Simulation, and Implementation of PID Controller on Quadrotors*, 2021 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2021, pp. 1-7, doi: 10.1109/ICCCI50826.2021.9402301.

[9] K.J. Åström, T. Hägglund, *The future of PID control*, Control Engineering

Practice, Volume 9, Issue 11, 2001, Pages 1163-1175, ISSN 0967-0661.

[10] D. D. Timis and E. H. Dulf, *Software implementation and testof an advanced robust control applied to a quad-copter*, 2020 IEEE International Conference on Automation, Quality and Test- ing, Robotics (AQTR), Cluj-Napoca, Romania, 2020, pp. 1-4, doi: 10.1109/AQTR49680.2020.9130018.