DOI: 10.2478/sbeef-2022-0018

QUADCOPTER PROTOTYPE STABILITY ANALYSIS USING MATLAB SIMSCAPE LIBRARY

HAMZA DJIZI[1,2], ZOUBIR ZAHZOUH[3], ABDELAZIZ LAKEHAL[3,*]
[1]Department of Mechanical Engineering, University of Souk Ahras, P.O. Box 1553, Souk Ahras, 41000, Algeria.
[2]INFRA-RES Laboratory, University of Souk Ahras, Algeria.
[3]Laboratoire de Recherche en Électromécanique et Sûreté de Fonctionnement, LRESF Laboratory, University of Souk Ahras, P.O. Box 1553, Souk Ahras, 41000, Algeria.
E-mail (corresponding author): a.lakehal@univ-soukahras.dz

Abstract. ***Nowadays, the use of quadcopters in daily life has become important due to its capabilities and ability to carry out many tasks in many fields like civil, military, industrial, and agricultural fields. The modelling of the quadcopter and deeply understanding its movements is very important to ensure that the simulations of its behaviour are as close as possible to reality and also helps us to design a flight controller. In this work, we used a modern technique on MATLAB (Simscape) to simulate a quadcopter in real-time. At first, we build a quadcopter using Simscape multibody then we simulated the PID regulator, the command algorithms, and the motor model with the applied forces on the body to achieve the global model that we can use to study the movement of the quadcopter on the three-axis which ensure a stable functioning. The results obtained show the stability of the four movements of the quadcopter (roll, pitch, yaw, and altitude).***

Keywords: Quadcopter, Electrical model, PID regulator, Simscape, Stability.

1. INTRODUCTION

Nowadays, with the incredible development of technology, especially electronics and computer science, which led to appear powerful microcontrollers and powerful software, this is what helped people to increase the performance of drones from both sides: software, and hardware. Some important applications of this aircraft are that it can be used in the army, entertainment, surveillance, aerial photography, agriculture, and transportation, where individual goods can be transported too anywhere. The irrigation has become more efficient when using drones to be watering, monitoring fields and detects water pooling or leaks using thermal cameras.

Quadcopters have a different behaviour from other planes, as they have four movements resulting from the rotation of four engines in diverse ways. To obtain a vertical movement, all the engines must rotate at the same speed. As for the other movements there are two types of quadcopters (+ and x), and each type has its own way of controlling [1].

The control of the quadcopter depends mainly on its stability [2-3]. Where a special control unit is designed based on the basic movements made by the aircraft during flight. These movements are controlled by special propellers that produce forces and this shows that the aerodynamic properties of the aircraft are very important as many researchers have studied these properties in depth [4]. In addition to the mechanical vibration characteristics of the propellers which help the quadcopter to fly safely in optimal conditions [5].

Ensuring drone stability is an important task for ensure a safety flying. So, to stabilize a quadcopter there are many different regulators used. The most important and widely used regulators in the field of drones are: PID and LQR [6-7]. In addition, other different hybrid regulators can be used (P-LQR, PD-LQR and PD2-LQR) [8]. However, the PID has the ability to correct the error and apply accurate and optimal control using three control terms: proportional, integral and derivative. As for the LQR is a method that used to find the optimal control action that ensure a high stability and performance to the system by reducing the cost J value using two matrices Q and R, where Q is a square matrix with rows equal the number of states, and R is square matrix with rows equal to the number of inputs. Also, many artificial intelligent techniques based on Artificial Neural Networks (ANN) are used for improving the response of the PID regulator and make the system more reliable [9-10].

To develop a quadcopter model under MATLAB (Simulink), it is important to understand the quadcopter behaviour and its movements (Altitude, Roll, Pitch, and Yaw). In addition to understand the relation between the different equations, there are different ways to simulate a quadcopter model on MATLAB. The different mathematical equations of a quadcopter can be used to create a Simulink model in several ways [11-12]. A mechanical model can be developed based on Simscape Multibody library with the help of an electric library to simulate the motors [13].

In this work, we studied the x-type, where in this type we can control the three movements (roll, pitch, yaw) through pairs of motors. Firstly, to achieve a roll movement, the speed of the two motors on the left must be increased, and the speed of the two motors on the right must be decreased. This applies torque around the x-axis to obtain a rotational movement this movement is coupled with a translational movement along the y-axis. Secondly, to achieve a pitch movement, the speed of the two motors on the front must be increased, and the speed of the two motors on the back must be decreased. This applies torque around the y-axis to obtain a rotational movement this movement is coupled with a translational movement along the x-axis. Finally, the yaw movement can be produced when the two diagonal motors rotate clockwise while the other two diagonal motors rotate counter clock wise. The main objective of the simulation in MATLAB using Simscape library is to study the aircraft performance in a better efficiency, where we can achieve and control the aircraft stability in its four basic movements.

2. EQUATIONS AND METRICS

2.1 The equations used in the model.

To calculate the forces on the three axes generated by each motor, the following equations are used:

$$F_{xx} = TF * \sin(y) \quad (1)$$

$$F_{yy} = TF * \sin(x) \quad (2)$$

$$F_z = \sqrt{((TF * \cos(x))^2 + ((TF * \cos(y))^2} \quad (3)$$

To calculate the thrust force the equation (4) is used:

$$Thrust = D^3 * Pitch * RPM^2 * 10^{-10} \quad (4)$$

2.2 Command Algorithms

$$MM_{11} = TT + RR + PP - YY \quad (5)$$
$$MM_{22} = TT - RR + PP + YY \quad (6)$$
$$MM_{33} = TT - RR - PP - YY \quad (7)$$
$$MM_{44} = TT + RR - PP + YY \quad (8)$$

2.3 PID Equation

The PID regulator, also called PID corrector (proportional, integral, derivative) is a control system that improves the systems performance.

The following equation shows the PID command:

$$uu(tt) = KK_{pp}\, ee(tt) + KK_{ii} \int ee(tt)ddtt + KK_{dd} \frac{dddd}{dddd} \quad (9)$$

After studying and improving the system, we were able to obtain the constants shown in (Table 1), which are the right values for the stability of the system.

Table 1. PID coefficients

PID	Coefficients		
	K_p	K_i	K_d
Roll	0.5	0.4	0.5
Pitch	0.5	0.4	0.5
Yaw	1	0.5	0.9
Altitude	0.2	0.2	0.1

In Table 2, mechanics parameters and metrics are presented.

Table 2. Mechanics parameters and metrics

Parameters	Value / Units
Weight of drone (m)	0.49 (kg)
Speed of motor	(RPM)
Propellers size	D/Pitch –08/4.5 (inches)
Thrust-ForceTF, Fx, Fy, and Fz	one force(Oz), and Newton (N)
x, y, z	Meter (m)
Ix, Iy, Iz	[0.00266, 0.00266, 0.0027] (kg.m^2)
Angles (roll, pitch, yaw)	(rad)
Speed	(m/s)
Mi	motors
T, R, P, Y	(Thrust, Roll, Pitch, Yaw)

3. SIMULATION

Computer or digital simulation refers to executing a computer program to simulate a real and complex physical phenomenon. So, a built in Simscape library under MATLAB used to simulate a prototype of quadcopter. The figures below show the important blocks and the global model.

To study the stability of the quadcopter, the regulator PID is used with the model. Each movement has a specific regulator (Figure 1).

- Roll PID regulator.
- Pitch PID regulator
- Yaw PID regulator
- Altitude PID regulator

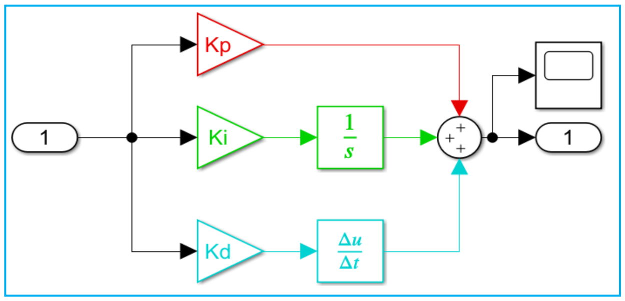


Figure 1. PID regulator model for each movement

Each motor must have a model (electrical model) using electrical tools under Simscape library as shown in the (Figure 2).

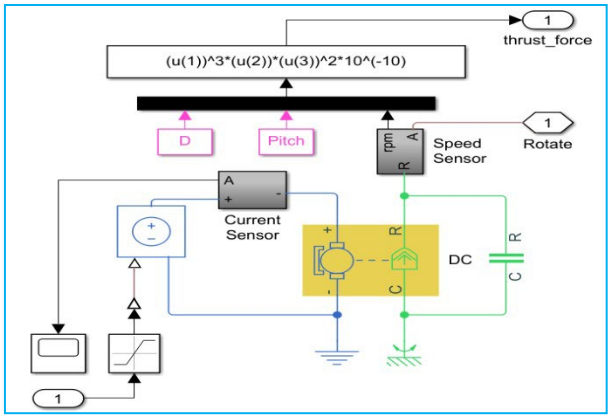


Figure 2. Brushless Direct Current motor (BLDC) motor model

Figure 3 shows how the control commands are linked to control the four movements of the quadcopter, each character for a specific motor (U, L, D, R).

After we calculate the thrust of each motor, we must calculate the force along of each axis. From Figure 4, we see how the forces along of the three axes can be calculated.

The software SolidWorks is used to design the quadcopter structure. At first, the base plate is simulated, arms to support motors then the propellers blades and motors. After that the model is uploaded on MATLAB as shown in the (Figure 5).

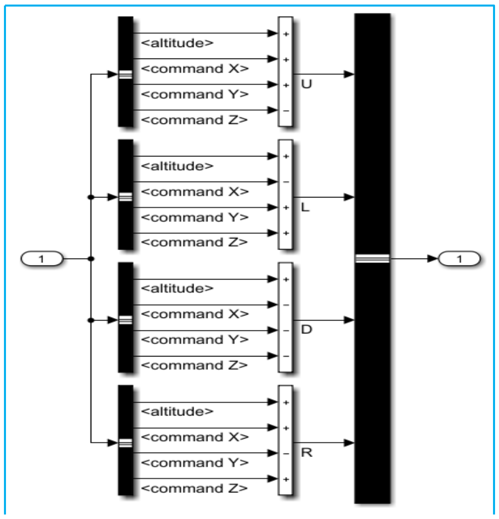


Figure 3. Command Algorithms model

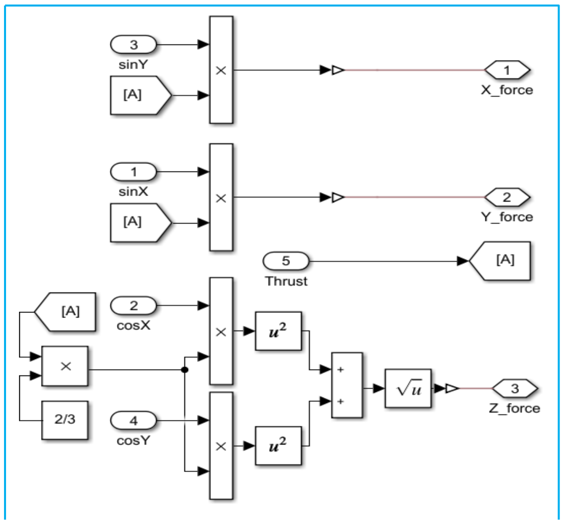


Figure 4. Forces along of the three axes (x, y, and z)

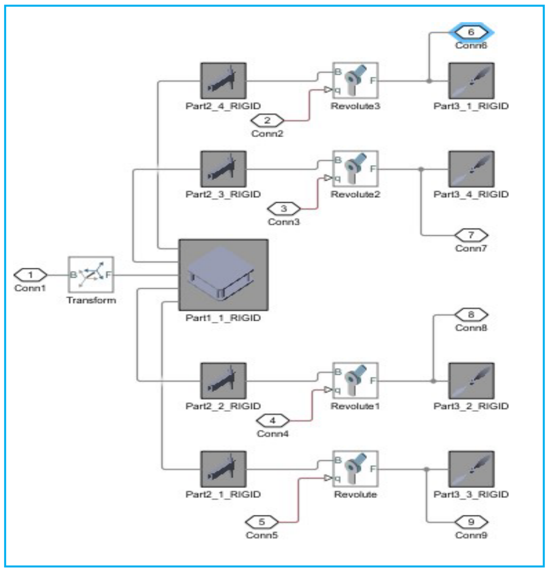


Figure 5. Quadcopter structure model using Simscape multibody.

Figure 6 shows the quadcopter structure in the mechanics explorer under MATLAB environment:

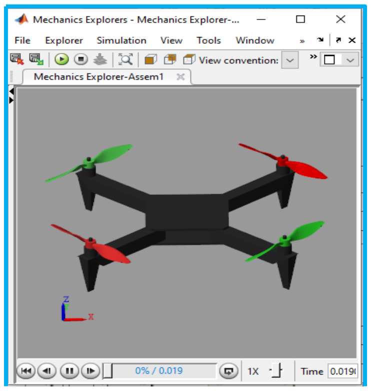

Mechanics Explorers - Mechanics Explorer-...
File Explorer Simulation View Tools Window
View convention:
Mechanics Explorer-Assem1
0% / 0.019
1X
Time 0.019

Figure 6. Quadcopter in Mechanics Explorer

To develop the global model, it is necessary to understand the behaviour of the quadcopter and its movements. It is known that the plane has six (6) degrees of freedom, but only four basic movements. So, it is easy to choose the variables that cause the quadcopter to move in different directions.

- Control the altitude along the vertical z-axis: Altitude.
- Control of rotation around the x-axis: Roll.
- Control of rotation around the y-axis: Pitch.
- Control of rotation around the z-axis: Yaw

And know the relationships between the different blocks and models of the quadcopters (regulator, motors model, command algorithms model, and forces model).

Figure 7 shows the global model of the quadcopter:

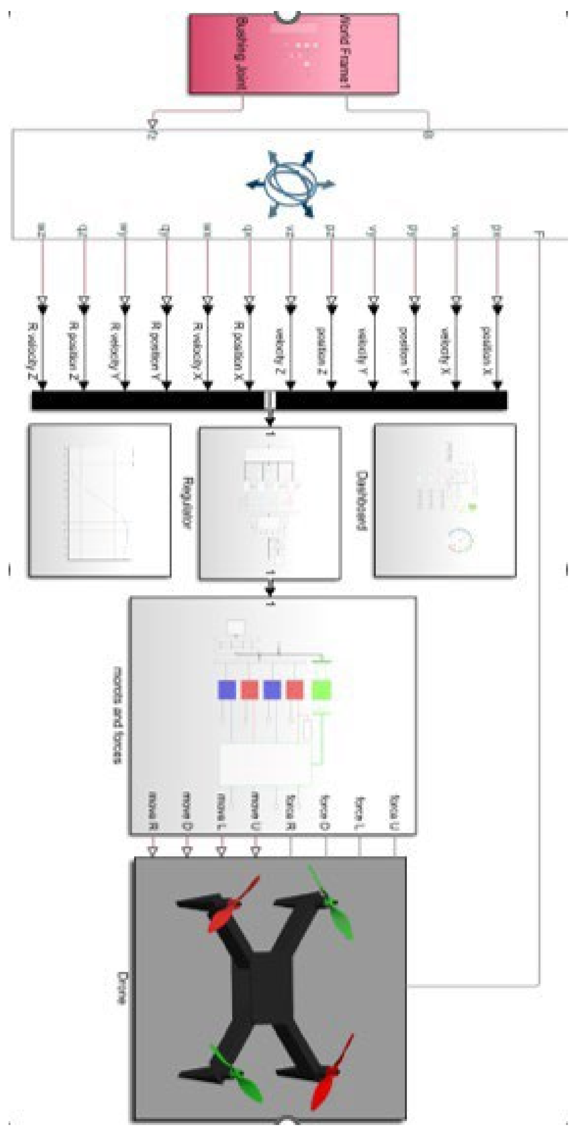


Figure 7. The quadcopter global model

4. RESULTS AND DISCUSSION

After developing the quadcopter model, the most important parameter to be studied is the aircraft stability during the performance of its four basic movements avoiding any external perturbations.

Then, the PID regulators must be tuned and adjusted. There are four movements, so we have four PID regulators. Each regulator has three constants (k_p, k_i, k_d). All these constants are adjusted until optimal stability is gotten (Table 1).

4.1 Altitude results

In order to obtain a vertical movement, it is necessary that all the lifting force is opposed to the force of gravity along the z-axis. On the other hand, the lifting force created by each motor must be the same to prevent overturning of the quadcopter. For this, the thrust produced by motors must be identical.

The up and down movement is obtained by changing the speed of motors rotation. If the thrust force is greater than the weight of the quadcopter, the movement is upward, and if it is less than the weight of the quadcopter, the movement is downward.

In the (Figure 8) we can see the stability on the z-axis. When, we give a command to the drone to climb a height of five meters, the PID regulator adjust the velocity of the quadcopter to stabilize at the given height command after about six seconds, which represent a good result compared with the literature.

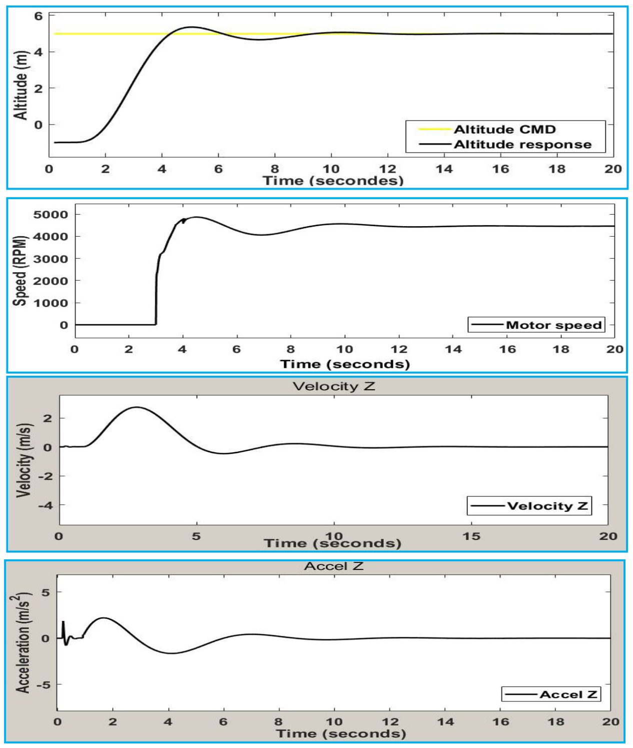
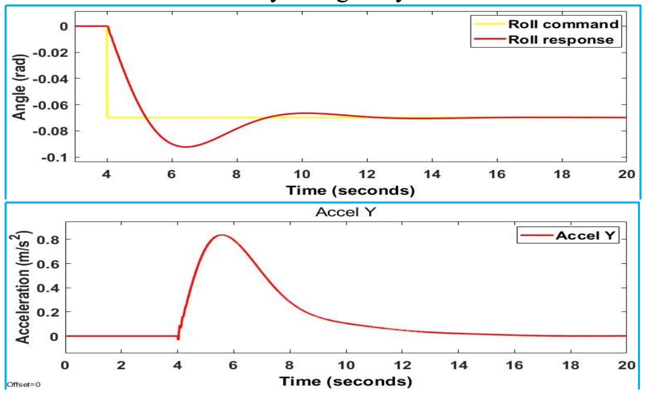


Figure 8. Altitude command and response (Altitude, Motor speed, velocity, and acceleration)

4.2 Roll results

The roll movement is done when a torque is given around the x-axis. This torque rotates the plane at a specific angle. This movement produces another movement along the y-axis. This allows the drone to go left or right.

To stabilize the movement along the y-axis, the PID parameters are tuned for the roll movement. Figure 9 shows the drone's stability along the y-axis.



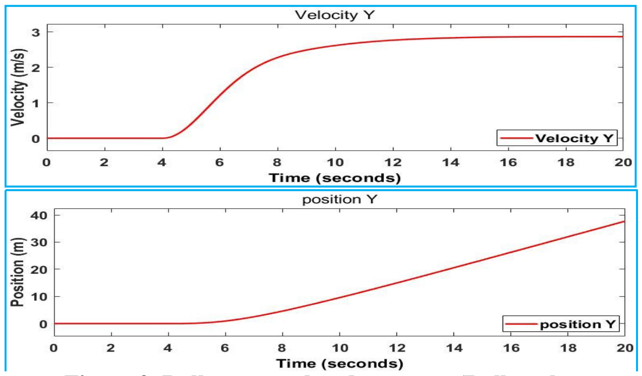


Figure 9. Roll command and response (Roll angle, acceleration, velocity, and position)

4.3 Pitch movement

This movement is similar to that of rolling. When a torque is given around the y-axis, it rotates the plane at a specific angle. This movement produces another movement along the x-axis. This allows the drone to go forward or backward. To stabilize the movement along the x-axis, the PID parameters are tuned for the Pitch movement. Figure 10 shows the drone's stability along the x-axis.

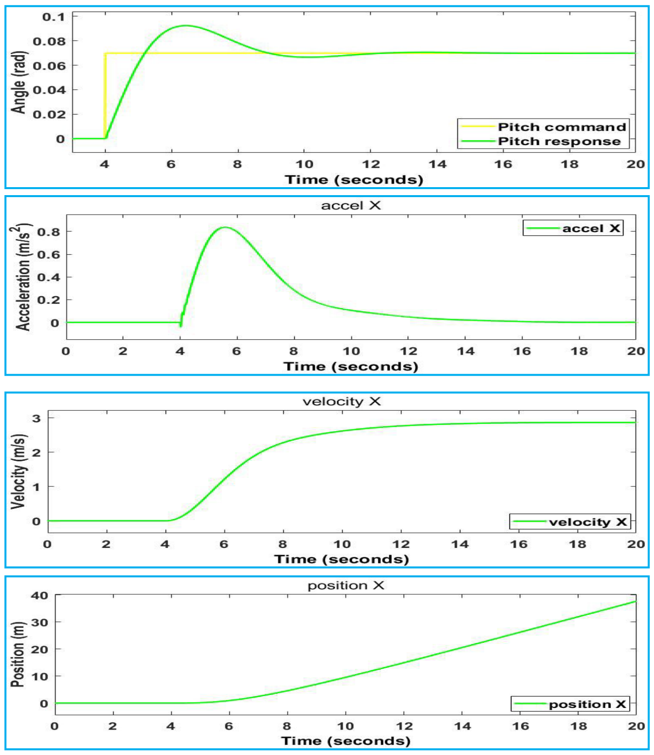


Figure 10. Pitch command and response(Pitch angle, acceleration, velocity, and position)

4.4 Yaw results

The yaw movement makes the drone change her direction. When a torque is given around the z-axis using the diagonals motors, the torque rotates the plane at a specific angle around the z-axis. To stabilize the Yaw movement around the z-axis we have tuned the PID parameters to ensure the stabilization.

Figure 11 shows the drone's stability around the z- axis.

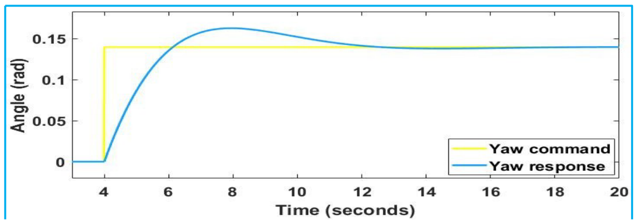


Figure 11. Yaw command and response

5. CONCLUSIONS

The Simulink quadcopter model was introduced using the Simscape library under MATLAB and a PID controller to study the stability of a quadcopter in real time. After giving the motion command and measuring the output, the output error was calculated and then adjusted the error using a PID controller. The results were transmitted through the control algorithms to the kinetic model in the form of electrical energy, which rotates the motors, the latter produces a lift force, and this force can move the quadcopter.

When we ordered the quadcopter to climb to a height of three meters, it stabilized at this height in about seven seconds which is a good result according to the proposed system in this paper and compared to the literature. Also when the quadcopter is ordered to go left or right by giving the Roll angle about six or –six degrees, it stabilized in about eight seconds which also represent a nice result. The third result of this study is when the quadcopter is ordered to go forward or backward by giving the Pitch angle about six or -six degrees, the quadcopter stabilized in about eight seconds. Finally, the quadcopter too stabilized in the yaw movement in about eight second on the one hand and in the other hand the period of time for the drone stability of was so close due to the fact that the design of the quadcopter is symmetric.

6. REFRENCES

[1] Khaled Hassan M, Ayman Shehata T, Abdelrady OE. Design and Manufacturing of X-Shape Quadcopter. International Journal of Engineering Research & Science 2022; 8(4): 12–20.

[2] Jaehyun Y, Jaehyeok D. Optimal PID control for hovering stabilization of quadcopter using long short-term memory. Advanced Engineering Informatics 2022; 53: 101679.

[3] Trenev I, Tkachenko A, Kustov A. Movement stabilization of the parrot mambo quadcopter along a given trajectory based on PID controllers. IFAC Papers On Line 2021; 54(13): 227-232.

[4] Polivanov P. A, Sidorenko A. A. Aerodynamic Characteristics of a Quadcopter with Propellers. In AIP Conference Proceedings 2351, 040053, 2021.

[5] Faraz A, Anamika B, Pushpendra K, Pravin P. Patil. Modeling and Mechanical Vibration characteristics analysis of a Quadcopter Propeller using FEA. In IOP Conf. Series: Materials Science and Engineering 577, 012022, 2019.

[6] Leszek C, Krzysztof W. Optimizing PID controller gains to model the performance of a quadcopter. Transportation Research Procedia 2019; 40: 156-169.

[7] Faraz A, Pushpendra K, Anamika B, Pravin P. Patil, Simulation of the Quadcopter Dynamics with LQR based Control, Materials Today: Proceedings, Volume 24, Part 2, 2020, Pages 326- 332.

[8] Mohamad Norherman S, Parvathy R, Nurulasikin M S. An effective proportional-double derivative-linear quadratic regulator controller for quadcopter attitude and altitude control. Automatika 2021; 62(3-4): 415-433.

[9] Yoon G. Y, Yamamoto A, Lim H. O., Mechanism and neural network based on PID control of quadcopter. In 16th International Conference on Control, Automation and Systems (ICCAS), 19-24, 2016.

[10] Nguyen N. P, Mung N. X, Thanh H. L. N. N, Huynh T. T, Lam N. T, Hong S. K. Adaptive Sliding Mode Control for Attitude and Altitude System of a Quadcopter UAV via Neural Network. In IEEE Access, vol. 9; 40076-40085, 2021.

[11] Waqas M, Sakhawat H. Developing of the smart quadcopter with improved fight dynamics and stability. Journal of Electrical Systems and Information Technology 2019; 6:6.

[12] González-Hernández I, Salazar S, Lozano R, Ramírez-Ayala. Real-Time Improvement of a Trajectory-Tracking Control Based on Super-Twisting Algorithm for a Quadrotor Aircraft. Drones 2022; 6: 36.

[13] Budnyaev V. A, Filippov I. F, Vertegel V. V and Dudnikov S. Y. Simulink-based Quadcopter Control System Model. In 1st International Conference Problems of Informatics, Electronics, and Radio Engineering (PIERE), 246-250, 2020.

A quadrotor controlled in real-time using hand gestures and ROS2 multi-node communication within GAZEBO 3D environment

Hamza Djizi

INFRA-RES Laboratory,
Department of Mechanical Engineering,
University of Souk Ahras,
P.O. Box 1553, Souk-Ahras,
41000, Algeria
Email: hamzadjizi@gmail.com

Abdelaziz Lakehal* and Zoubir Zahzouh

Laboratory of Research on Electromechanical and Dependability,
University of Souk Ahras,
P.O. Box 1553, Souk-Ahras,
41000, Algeria
Email: lakehal21@yahoo.fr
Email: z.zahzouh@univ-soukahras.dz
*Corresponding author

Abstract: This paper introduces a novel way of designing and controlling a quadrotor prototype using hand gestures, utilising the Robotic Operating System 2 (ROS2) and GAZEBO11 3D environment. A C++-based plug-in was created for GAZEBO, while the cross-platform pipeline framework Media-Pipe was used to manage the quadrotor's movements through hand gestures. The PID regulator was utilised to enhance the movements' accuracy and responsiveness, leading to a more efficient and precise response to user commands for a better user experience. The obtained results demonstrated that the PID regulator improved the response of the quadrotor to hand gestures with greater accuracy.

Keywords: Robotic Operating System 2; ROS2; quadrotor; GAZEBO; control; communication; hand gestures.

Reference to this paper should be made as follows: Djizi, H., Lakehal, A. and Zahzouh, Z. (xxxx) 'A quadrotor controlled in real-time using hand gestures and ROS2 multi-node communication within GAZEBO 3D environment', *Int. J. Automation and Control*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Hamza Djizi received his Master's degree from the University of Skikda, Algeria, and currently working toward his PhD in Electromechanical Engineering at the University of Souk Ahras, Algeria. His research interests include system control, robotics and artificial intelligence methods.

Abdelaziz Lakehal received his PhD and the Habilitation to supervise research activities in Electromechanical Engineering, in 2013, and 2016 respectively. He is currently a Full Professor at the University of Souk Ahras. His research interests include maintenance, quality reliability and safety, fault diagnosis and prediction, electromechanical engineering.

Zoubir Zahzouh received his Doctorate in Electromechanical Specialty from the Skikda University, Algeria in 2015. He is currently an Associate Professor at the University of Souk Ahras. His areas of interest include power systems optimisation, power quality, power systems protection and renewable energies.

1 Introduction

Quadrotors have rapidly advanced in the last decade with increased hardware and software capabilities, relying more on artificial intelligence for complex tasks (Spanaki et al., 2022). The importance of drones in human society has grown due to their high efficiency and low cost. As a result, researchers and scientists have become highly interested in this type of aircraft. However, a reliable and precise controller is still necessary for these aircraft to perform tasks with minimal errors (Shirzadeh et al., 2021). To meet this need, The ROS system (ROS Docs, 2022) has gained popularity for its high efficiency and powerful software platform, particularly in the industrial field, which makes companies adopt it widely for research and development. This platform helps them move from research and prototyping to deployment and production.

There are two main versions of this system (ROS and ROS2). The community supports ROS on various Linux distributions, although it was first developed on Ubuntu. Robotic Operating System 2 (ROS2) is developed on Ubuntu, Windows, and MacOS (ROS docs). Additionally, an open-source 3D robotics simulator called GAZEBO (GAZEBO Docs, 2022) is used with ROS. This software is based on physics engine and OpenGL rendering, and it supports various sensors such as cameras, lidars, GPS, etc. The importance of GAZEBO appears during the manufacturing process, as it can save time and money to produce a cheaper product (gazebo APIs). There are an increasing demand for ROS applications in automation systems (Erős et al., 2019), particularly in the field of unmanned aerial vehicles (Orozco Soto et al., 2022). ROS helps researchers to develop more reliable solutions to ensure system robustness and autonomous mission success (García and Molina, 2022; Omar, 2022). ROS applications are also being utilised in the field of mobile industrial robots, which can be programmed to carry out repetitive tasks with high efficiency and accuracy (Puck et al., 2020).

One of the main challenges in programming a quadrotor is maintaining stability during flight. This can be achieved by using various regulators, such as PID regulators, that are integrated with the control commands to improve performance and maintain stability (Goud et al., 2022; Trenev et al., 2021). The PID regulator is widely used in industry for automatic process control, including in drones due to its ease of use and advantages (Yoon and Doh, 2022).

However, various controllers were developed to address the stability issues, such as linear controllers, nonlinear controllers, and learning-based control techniques. LQR and hybrid LQR offer higher efficiency and improved the quadrotor performance (Shauqee et al., 2021). The linear quadratic Gaussian controller uses feedback and optimal control theory to regulate and control dynamic systems with high performance and stability (Fessi and Bouallègue, 2019; Kumar et al., 2023). The H_∞ controller is a popular robust control system used in aerospace and automotive industries to achieve high performance and stability for complex linear time-invariant systems (Bellahcene et al., 2021). Model predictive control is an advanced control system used for regulating dynamic systems, especially quadrotors (Meradi et al., 2022; Zhao and Wu, 2023). Fuzzy logic and artificial neural networks are learning-based control techniques that can also improve quadrotor stability and performance (Abedzadeh Maafi et al., 2022; Dey et al., 2022; Guettal et al., 2022; Pakro and Nikkhah, 2022).

When the quadrotor is created and evaluated, it can be fabricated using the on-axis microstereolithography (OMSL) method, which is a method that uses layer-by-layer photopolymerisation to create three-dimensional (3D) components. It employs a laser beam to harden a liquid photopolymer to produce detailed, high-resolution structures. This technique is used to print the 3D quadrotor models created with 3D modelling software. The OMSL printer then uses the design to steer the laser beam into the liquid resin, resulting in the construction of the quadrotor's components (Gandhi et al., 2013). Bulk lithography is also a technology used to produce microelectronic components and integrated circuits. They are a type of photolithography technology that fabricates quadrotor components using light-sensitive materials and masks. By enforcing spatial variation of laser intensity at every location in a single-layer scan, this technique produces a variable depth 3D microstructure (Gandhi and Bhole, 2013).

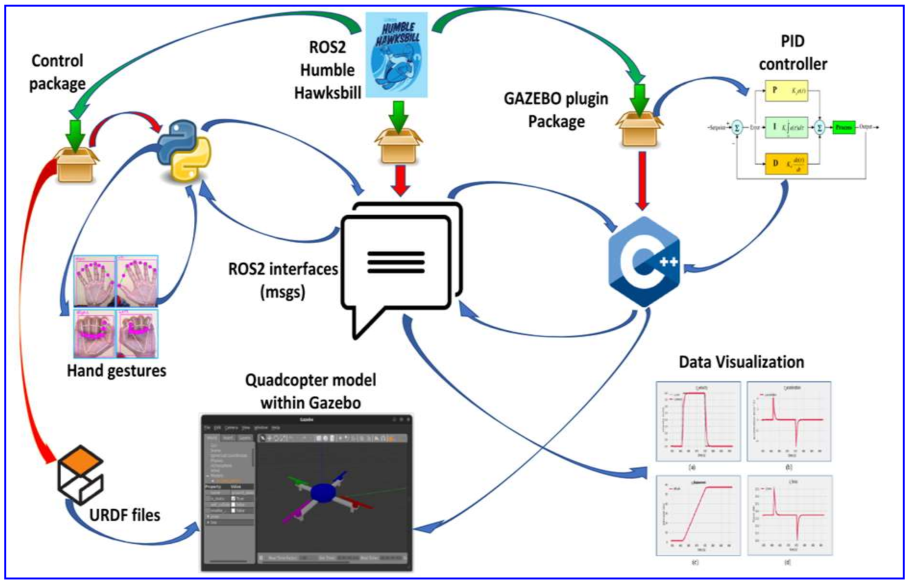
A quadrotor control system that uses Media-Pipe, GAZEBO, and ROS2 is presented in this work (Figure 1). Media-Pipe is a framework for multimodal machine learning pipelines, GAZEBO is a robotics simulation environment, and ROS2 is an open-source robot operating system. Media-Pipe processes quadrotor sensor data and generates control commands based on hand gestures, which are then published and subscribed between the nodes using ROS2, while GAZEBO simulates the 3D model, flight dynamics, and environment of the quadrotor. By combining Media-Pipe, GAZEBO, and ROS2, the quadrotor model can be realistically and scalably tested and evaluated without physical hardware.

Research highlights:

- introduction to quadrotor design and control using ROS2 and GAZEBO
- using MediaPipe framework to control the quadrotor with hand gestures
- PID regulator implementation to improve quadrotor response and stability
- future research and development can be guided by this work.

This study provides a detailed explanation of building a quadrotor model in ROS2 and GAZEBO and controlling it through hand gestures using ROS2 nodes. The quadrotor model's response is enhanced by the PID regulator, which improves stability in four basic movements, indicating the system's effectiveness and potential for future development. However, in Section 2, the ROS2 project architecture is presented. Section 3 details how to create the quadrotor 3D model with URDF. Section 4 describes the development of a C++ GAZEBO plug-in based on Newton's second law of motion to manage the various movements of the quadrotor. Section 5 goes through how to control the quadrotor with hand gestures. Section 6 describes the system's implementation. Section 7 demonstrates and discusses the communication between the different ROS2 nodes to control and monitor the system. The simulation results pertaining to the system are presented in Section 8. Finally, conclusions and future work are mentioned in Section 9.

Figure 1 A graphical abstract that illustrates the process of controlling the quadrotor (see online version for colours)

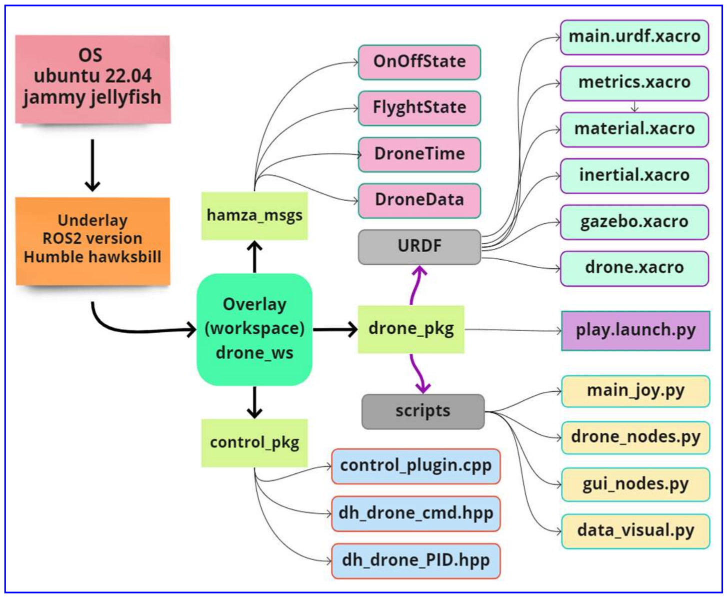


2 Quadrotor project architecture

This project builds a quadrotor using the recommended installation of ROS2 (Humble Hawksbill) on Ubuntu Linux – Jammy Jellyfish (22.04). The project's architecture is illustrated in Figure 2, with the setup environment for installed ROS2 packages and libraries serving as the underlay. Ubuntu Jammy currently supports ROS2 Humble Hawksbill packages. The overlay serves as the setup environment for the workspace packages that were built for the drone project. Three packages were created, each one dedicated to a specific part of the project. The first package, *hamza_msgs*, contains interfaces used for communication between ROS2 applications, including messages, services, and actions. The second package, *drone_pkg*, contains the quadrotor model description designed using a unified robot description format based on Extensible Markup Language (XML). In addition to the python scripts programmed to recognise hand gestures as commands using the Media-Pipe framework. The third package, *control_pkg*, contains a C++ plug-in responsible for controlling the quadrotor's motors, sensors, and other components in GAZEBO11. This plug-in is called from the URDF file. Furthermore, this plug-in is intended to receive commands from the control node, and send data to the visualisation node.

To begin working with ROS2, a new directory must be created to contain the workspace called *drone_ws*. Create packages within the '*src*' folder under the created directory. These packages serve as a container for ROS2 code. The packages must then be built to obtain the generated files for the project. Finally, to launch the packages, the installation file must be sourced using the terminal.

Figure 2 ROS2 project architecture (see online version for colours)

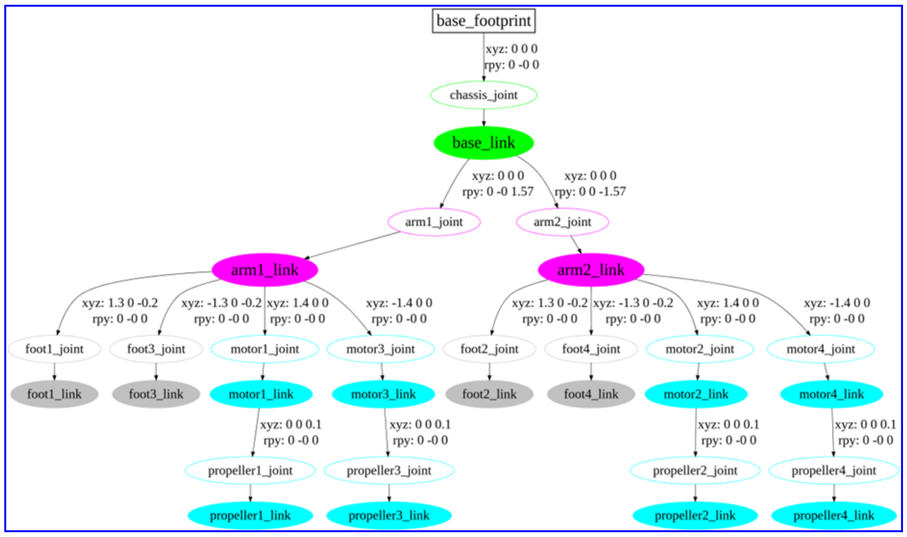


3 Create quadrotor 3D model

GAZEBO is an invaluable tool for robotics development, as it provides a realistic 3D environment to simulate real-world physics and test the performance of robots with various sensors. It is useful when the hardware is unavailable, as it allows developers to build and test robot prototypes in a virtual environment. Even when the hardware is available, GAZEBO is still an important tool for testing robots before they are implemented in the real world. With its open-source nature, GAZEBO is a powerful and efficient tool for robotics development.

URDF is an XML language used with ROS and GAZEBO to create 3D quadrotors, where the quadrotor model of this work consists of links connected between each other through joints, as a parent and a child. Figure 3 demonstrates the different parts of the quadrotor. The rectangles denote links and the ellipses denote the joints. The *base_footprint* is linked to the *base_link* through the fixed *chassis_joint*, then the two arms (*arm1_link* and *arm2_link*) are linked to the *base_link* through two fixed joints (*arm1_joint* and *arm2_joint*), then four *motor_links* and four *foot_links* are linked to the two *arm_links* through fixed joints. Finally, the four *propeller_links* are linked to the four *motor_links* through four continuous joints in order to move the propellers freely. To control the 3D model in the GAZEBO, the C++ plug-in must be added to the URDF files.

Figure 3 Hierarchical graph from quadrotor model (URDF) (see online version for colours)



4 Create GAZEBO plug-in

To control the 3D model in GAZEBO, a new plug-in must be created with the help of the GAZEBO APIs based on C++ language, where this plug-in gives the URDF model greater functionality and more flexibility to connect between ROS2 nodes for sending and receiving data. The plug-in must be developed until the quadrotor can respond to the *on_off*, *take_off*, and *landing* commands, it can also respond to the command velocities (*x_cmd*, *y_cmd*, *z_cmd*, and *yaw_cmd*), and it can fly through different states (*taking_off*, *flying*, *landing*, and *low_battery* state). Additionally, A PID regulator should also be included in the model since it may considerably enhance the quadrotor's reaction and stability during flight. The quadrotor can maintain its ideal attitude and altitude more accurately and consistently by utilising this regulator, as well as providing a more robust response to external disturbances. This ensures the stability of the quadrotor, and it can carry out intended functions with better accuracy and dependability.

The GAZEBO plug-in that controls the quadrotor's movement must be based on the principles and physical laws that govern its movement in 3D space. The forces created by the spinning of the quadrotor's four propellers govern its behaviour. These forces can be employed to control its movement. According to Newton's second law of motion, the quadrotor's acceleration is determined by two variables: the total net force created by the four propellers and the quadrotor's mass. Additionally, the forces created by the propellers must be included when calculating the drag and lift forces experienced by the quadrotor in flight.

4.1 Newton's second law of motion

Newton's second law of motion is used to determine the acceleration of the quadrotor in response to the net force generated by its propellers.

$$\sum F = ma \tag{1}$$

$$F_1 + F_2 + F_3 + F_4 - mg = ma \tag{2}$$

F_i	force generated by each propeller (N)
M	quadrotor's weight (kg)
a	quadrotor acceleration (m/s^2)
g	gravitational acceleration (m/s^2).

4.2 Hovering flight condition

Hovering flight is a stationary flight where the quadrotor remains stable in the air without any movement. This is achieved when the total thrust generated by the propellers is equal to the mass of the quadrotor multiplied by the gravitational force. By maintaining this balance, the quadrotor is able to remain in a stationary position in the air.

$$ma = 0 \tag{3}$$

$$mg = F_1 + F_2 + F_3 + F_4 \tag{4}$$

4.3 Vertical flight condition

The vertical motion of a quadrotor can be achieved by controlling the thrust of its rotors when it ascends and descends. By adjusting the thrust of each rotor, the quadrotor can move up and down, allowing it to generate a vertical force.

- *Ascend:* The quadrotor can ascend when the sum of the forces generated by the propellers exceeds its weight, which is equal to its mass multiplied by the gravitational force.

$$ma > 0 \tag{5}$$

$$mg < F_1 + F_2 + F_3 + F_4 \tag{6}$$

- *Descend:* The quadrotor can descend when the sum of forces generated by the propellers is less than its weight multiplied by the gravitational force.

$$ma < 0 \tag{7}$$

$$mg > F_1 + F_2 + F_3 + F_4 \tag{8}$$

4.4 PID regulator

To improve the performance of the quadrotor for the four movements, the PID regulator should be implemented using a header file containing source code written in C++

programming language. Then it must be added to the main program file (GAZEBO plug-in) using the #include directive. Equation (9) will utilise for calculating the integral using the resulting velocity error between the desired velocity and the current velocity, in addition to the updated running time. Equation (10) will employ for calculating the derivative using the error and the previous error along with the up-to-date running time. Equation (11) calculates the PID response by summing three parameters: proportional, integral, and derivative. Furthermore, the PID regulator must be tuned to ensure optimal performance of the quadrotor. This can be done by adjusting the PID parameters, such as the proportional, integral, and derivative gains.

$$integral = integral + (error * dt) \tag{9}$$

$$derivative = (error - previous\_error)/dt \tag{10}$$

$$PID = K_p * error + K_i * integral + K_d * derivative \tag{11}$$

4.5 Moment of inertia

To design a simple prototype using URDF files, boxes and cylinders can be used as components of the quadrotor. Each component's inertia matrix can be calculated from its mass, length, width, height, and radius based on box and cylinder characteristics. The inertia matrix can be calculated using the two following formulas:

1 Inertia matrix of a box:

$$I_{box} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}m\left(y^2+z^2\right) & 0 & 0 \\ 0 & \frac{1}{12}m\left(x^2+z^2\right) & 0 \\ 0 & 0 & \frac{1}{12}m\left(x^2+y^2\right) \end{bmatrix} \tag{12}$$

2 Inertia matrix of a cylinder:

$$I_{cylinder} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} \frac{1}{12}m\left(3R^2-L^2\right) & 0 & 0 \\ 0 & \frac{1}{12}m\left(3R^2+L^2\right) & 0 \\ 0 & 0 & \frac{1}{2}mR^2 \end{bmatrix} \tag{13}$$

Where Table 1 presents the physical quantities used to calculate the inertia matrix.

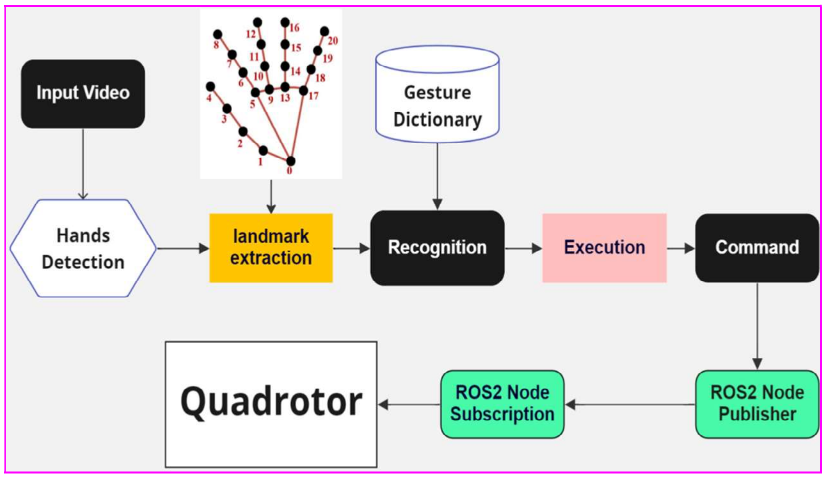
5 Hand gestures control node

The quadrotor control will use ROS2's closed-loop system and real-time computing, which is a crucial characteristic of autonomous vehicles. The quadrotor prototype will be tested and verified for successful takeoff, flight, and landing using hand gestures, utilising the Python Media-Pipe library's powerful capabilities. This library allows developers to rapidly create real-time applications that can detect and track hand movements. The Python Media-Pipe library is an excellent choice for hand gesture-based applications due to its robust feature set. Developed by Google, Media-Pipe is an open-source framework that simplifies the creation of custom machine-learning solutions for live and streaming media. The framework enhances computer vision applications and has several advantages, including being open-source, free, and cross-platform compatible with Android, iOS, Mac, web, and Linux. Media-Pipe also supports common hardware such as GPUs, CPUs, and TPUs, which enables fast ML inference and video processing. This framework offers a variety of machine-learning solutions, such as FaceDetection, FaceMesh, ObjectDetection, HandsDetection, and more.

Table 1 Physical quantities used for inertia matrix

Type	*Symbol*	*Description*	*Physical quantity*	*Unit*
Box	I_{box}	Box moment of inertia	MomentOfInertia	kg.m^2
	m	Mass	Mass	kg
	x	Length	Length	m
	y	Width	Width	m
	z	Height	Height	m
Cylinder	$I_{cylinder}$	Cylinder moment of inertia	MomentOfInertia	kg.m^2
	m	Mass	Mass	kg
	L	Length	Length	m
	R	Radius	Radius	m

Figure 4 Diagram of the steps followed for quadrotor control using hand gestures (see online version for colours)



The quadcopter control will be based on the HandDetection solution in ROS2 environment. The control commands will be executed according to the hand gestures, which will be interpreted using 21 3D coordinates. The x and y values were obtained from the size of the video, while the *z* value was taken from the image depth how the 21 landmarks must be taken is shown in Figure 4.

The control node is dedicated to interpreting commands from hand gestures using the latest artificial intelligence and computer vision technology. To do this, the Media-Pipe library from the Python language is used to detect human hands, extracting 21 landmarks for each hand. These landmarks are then used to recognise and classify the hand gestures, which are then interpreted and published as commands to the quadrotor node subscriptions through topics based on certain conditions. This allows for a seamless and intuitive way to control the quadrotor with just the movement of hands. The diagram presented in Figure 4 illustrates the architecture for utilising Media-Pipe hand gestures with ROS2 to control the quadrotor model.

5.1 *Taking-off or landing*

To quickly and easily get the quadrotor in the air, two-handed gestures can be used. Firstly, ensure the quadcopter is powered on and your hands are in the correct position in front of the computer camera. Figure 5(a) shows how to use a two-handed gesture to initiate take-off, and closing both hands together can initiate landing while in flight as shown in Figure 5(b). This will signal the quadrotor to begin its descent and land on the ground.

5.2 *Forward, backward, upward, or downward movement*

To control the quadrotor's movement, a right-handed gesture is utilised to move it forward or backward, and a left-handed gesture is used to move it upward or downward. The system's functionality should be tested by performing the appropriate gesture and observing the quadrotor's response. If it responds as expected, the system is deemed operational. The hand gestures for each movement are illustrated in Figure 5: forward [Figure 5(g)], backward [Figure 5(h)], upward [Figure 5(i)], and downward [Figure 5(j)].

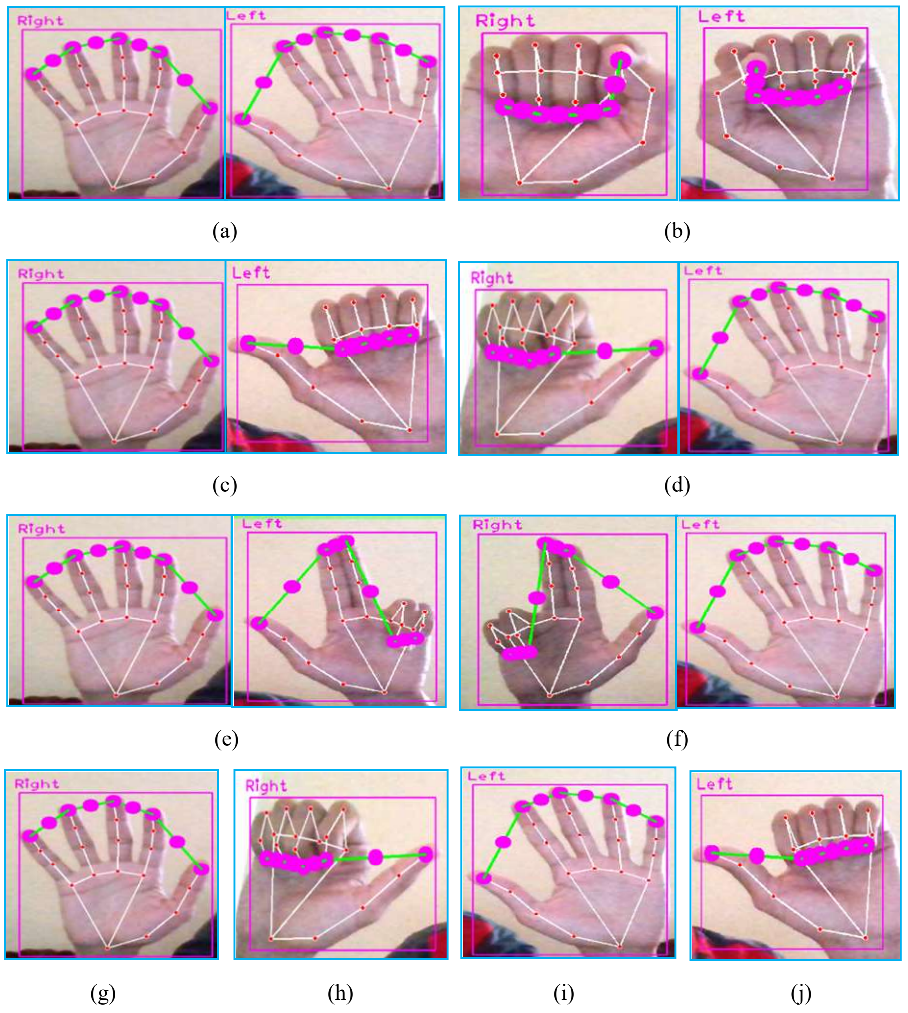
5.3 *Right or left movement*

The Media-Pipe framework allows for simple manipulation of the quadrotor with a few hand gestures, enabling left or right movements, and enhancing the overall interactive and engaging experience for the user. As demonstrated in Figure 5(c), the gesture for causing the quadrotor to move left is a simple movement of both hands. Similarly, Figure 5(d) depicts the gesture for the quadrotor to move right, which is also a simple movement of both hands.

5.4 *Yaw movement*

The yaw motion can be intuitively controlled through hand gestures. As illustrated in Figure 5(e), the gesture of both hands will result in a counterclockwise rotation around the *z*-axis for the quadrotor, and as shown in Figure 5(f), the gesture of both hands will result in a clockwise rotation. By performing these gestures in front of the camera, the user can easily and efficiently control the rotational movement (yaw) of the quadrotor.

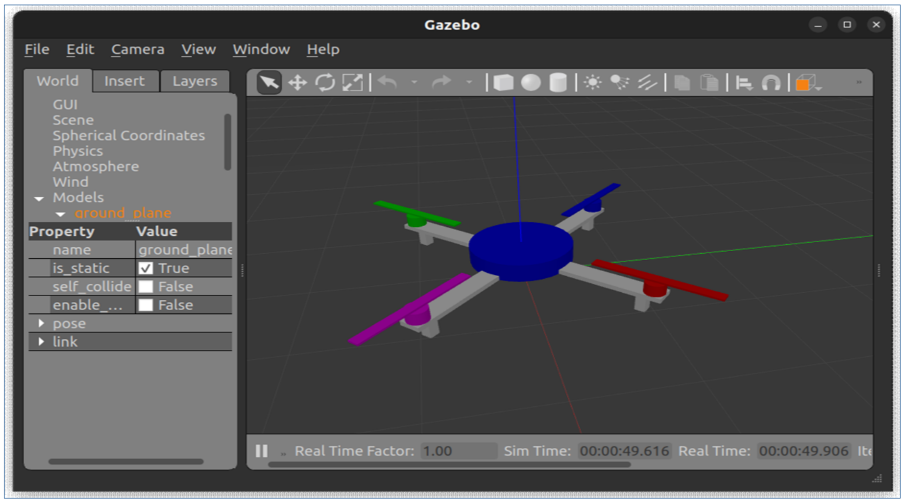
Figure 5 Hand gestures for control the quadrotor (see online version for colours)



6 Implementation

To integrate the quadrotor's model and control programs, they must be connected through ROS2 interfaces for communication and control. The ROS2 tools can be utilised to verify and observe the system's performance. The project must be built with the colcon tool, sourced, and launched with the ROS2 launch command. Then, the quadrotor model must be spawned in GAZEBO using the robot_description topic (Figure 6). Simulations must be conducted to evaluate the system's outcomes and ensure its proper operation.

Figure 6 3D model under GAZEBO11 (see online version for colours)



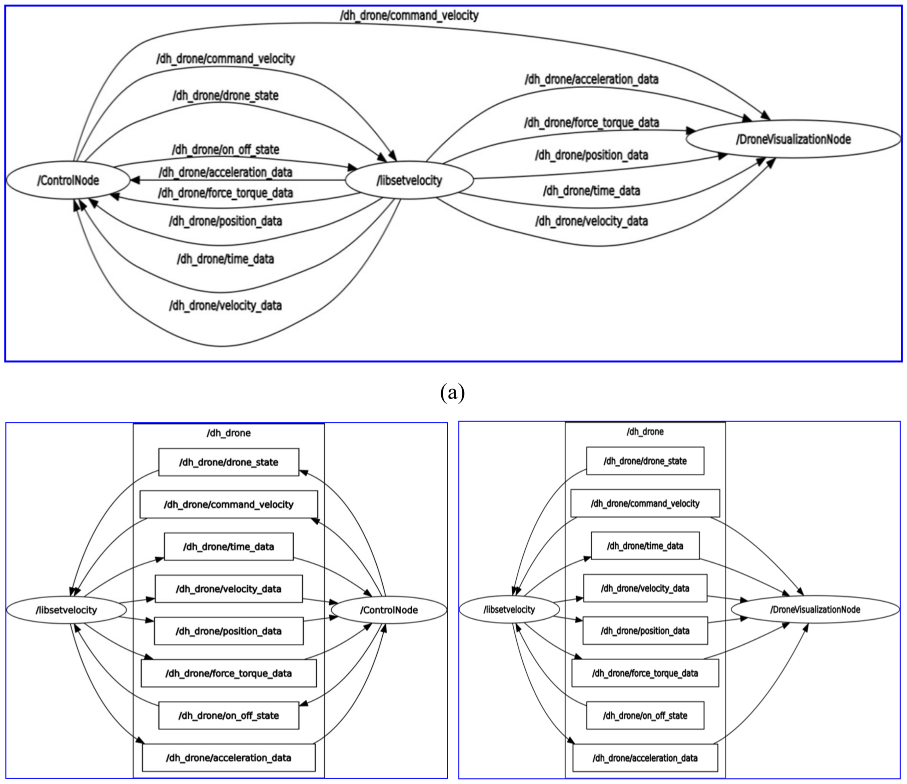
7 Communication between ROS2 nodes

Nodes are the executable files that make things happen in ROS2, these nodes can send and receive data to the other nodes using topics, services, actions, or parameters. Topics are a vital element that acts as a bus for nodes to exchange messages using publishers and subscribers. A node can publish messages to any number of topics simultaneously and can also have subscriptions to any number of topics. When the project is launched, three main nodes must be running: The control node, The GAZEBO node (libsetvelocity), and the visualisation node, as shown in Figure 7(a). The control node is responsible for controlling the quadrotor's motion, the GAZEBO node is responsible for simulating the quadrotor's environment, and the visualisation node is responsible for displaying the quadrotor's state.

Figure 7(b) illustrates the communication between the two nodes [GAZEBO node (*libsetvelocity*), and control node] using topics. The control node transmits the hand gesture commands to the gazebo node. The *on_off_state* command is transmitted to turn on the quadrotor through the */dh_drone/on_off_state* topic. When the drone is running, the control node transmits another command for taking off the drone using the */dh_drone/drone_state* topic. When the quadrotor is flying, the control node can transmit the velocity commands through the */dh_drone/command*_velocity topic. The gazebo node transmits back the drone flight data through data topics to the control node, allowing for real-time controlling of the quadrotor's performance.

To enhance and monitor the quadrotor's performance, the animation function (FuncAnimation) from the Matplotlib library in Python must be used to plot the position, velocity, acceleration, force, and torque curves. This function enables the creation of a real-time animation based on the flight time from *Gazebo_sim*. The visualisation node plots the position, force, torque, acceleration, and time data received from the GAZEBO node (*/libsetvilocity*) as shown in Figure 7(c) for each movement. This data can be used to analyse the quadrotor’s performance and identify areas of improvement. For example, the acceleration data can be used to determine the quadrotor’s responsiveness to commands, while the force and torque data can be used to measure the amount of power being generated by the motors. By analysing this data, the quadrotor’s performance can be improved by making adjustments to the motor power, the weight of the quadrotor, or the chassis design.

Figure 7 (a) Multi-node communication graph (b) Communication between control node and gazebo node (c) Communication between gazebo node and visualisation node



(a)

(b) (c)

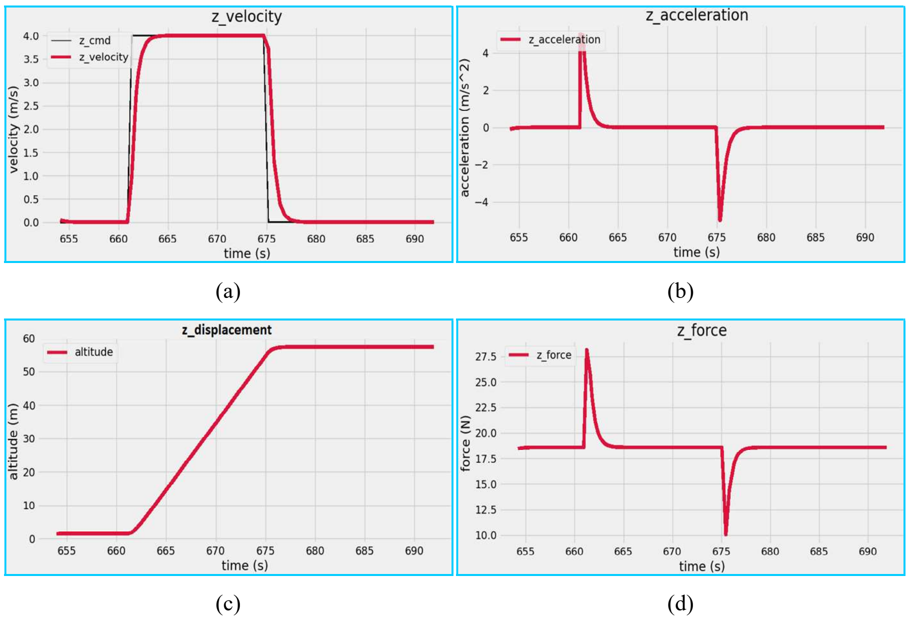
8 Results and discussion

The obtained results show that the system control is able to accurately detect and respond to the various hand gestures, allowing the quadrotor to take off, land, and perform various movements such as hovering, turning, and ascending/descending as instructed. Furthermore, the system is able to respond to the hand gestures in real-time, providing a smooth and responsive experience.

8.1 Vertical movement results

The quadrotor's exceptional altitude performance is demonstrated. The capacity to react swiftly and precisely to orders is proven, as seen in Figure 8(a), which illustrates velocity. According to Newton's second rule, the force is directly proportional to acceleration, as shown in Figures 8(d) and (b) which display the total thrust produced by the four motors and acceleration. The displacement variable [Figure 8(c)] shows how the location varies in response to the required velocity. As seen, movement starts when the velocity is set to 4 m/s and ends when the velocity is set to 0 m/s. The PID regulator's results are satisfactory and comparable to those reported in the literature. This contribution facilitates the quadrotor control by users.

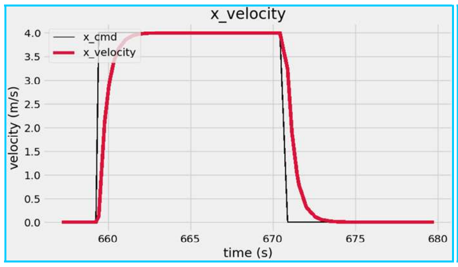
Figure 8 Altitude results, (a) velocity (b) acceleration (c) displacement (d) force (see online version for colours)



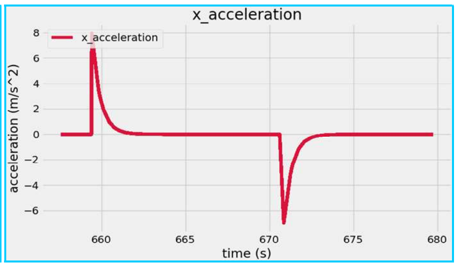
8.2 Results of forward and backward movement

Figure 9(a)-(d) depict the quadrotor's movement along the *x*-axis, displaying velocity, acceleration, position, and force results. The quadrotor's stability during motion is reflected in the velocity results, with a rise time of under 2 seconds and no overshoot. The changes in acceleration over time are presented similarly, as they are reflected in generated force variations (Newton's second law). The position changes in response to velocity commands are shown in Figure 9(c), where the quadrotor reaches its desired position in a few seconds. Force variations along the *x*-axis are consistent with acceleration and velocity. These results prove the quadrotor's successful operation.

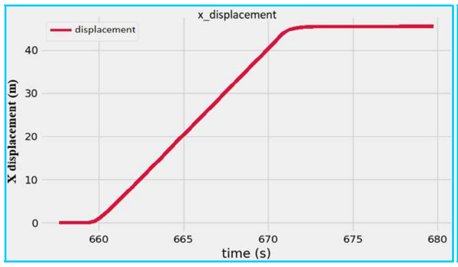
Figure 9 Linear *x* results, (a) velocity (b) acceleration (c) displacement (d) force (see online version for colours)



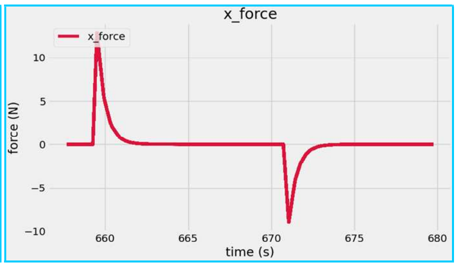
(a)



(b)

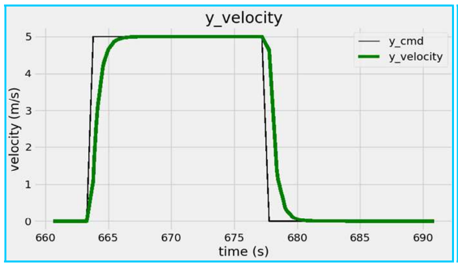


(c)

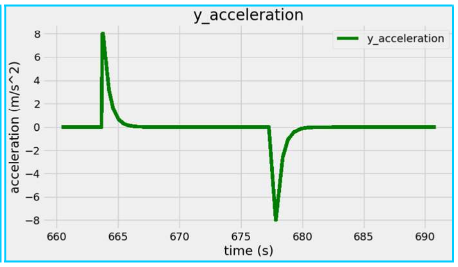


(d)

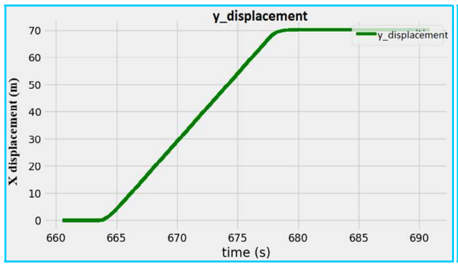
Figure 10 Linear *y* results, (a) velocity (b) acceleration (c) displacement (d) force (see online version for colours)



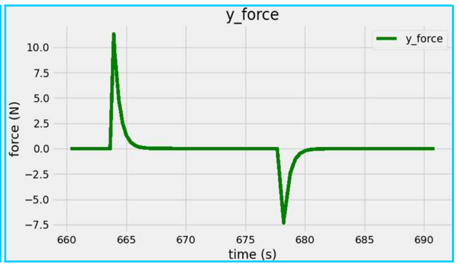
(a)



(b)



(c)



(d)

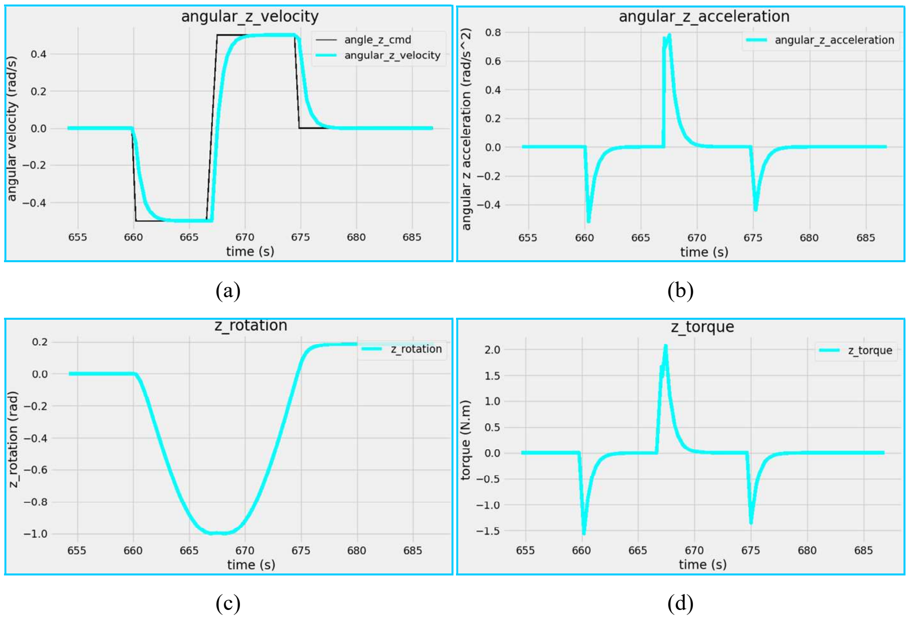
8.3 Results of right and left movement

Similar to the *x*-axis, the movement along the *y*-axis is shown in Figure 10, including velocity, acceleration, position, and force. The velocity figure shows the quadrotor's stability during motion with a small rise time of fewer than 2 seconds and no overshoot. Changes in acceleration over time are illustrated in Figure 10(b), which are reflected in generated force changes according to Newton's second law of motion. Finally, displacement shows the changes in position along the *y*-axis in response to the velocity command over time, indicating that the quadrotor can maintain a stable trajectory along the *y*-axis.

8.4 Yaw movement results

Figure 11 presents the impressive performance of the quadrotor in controlling angular motion around the *z*-axis (yaw motion). The figure includes results for angular velocity, angular acceleration, rotation, and torque generated around the *z*-axis. The results demonstrate the stability in rotational motion around the *z*-axis over time and the efficient response to control commands. The acceleration around the *z*-axis shows the quadrotor's ability to quickly and accurately adjust angular velocity in response to commands. Furthermore, the quadrotor's ability to generate sufficient torque to adjust angular velocity and acceleration is presented in Figure 11(d). Overall, the results show that the developed quadrotor has an impressive ability to accurately and efficiently control the angular motion around the *z*-axis.

Figure 11 Yaw results, (a) angular velocity (b) angular acceleration (c) rotation (d) torque (see online version for colours)



9 Conclusions

In this work, the operation of a 3D quadrotor model required the use of ROS2, GAZEBO 3D environment, Media-Pipe framework, and hand gestures. However, combining multiple programming languages such as C++, Python, and URDF proved challenging and time-consuming. To manage the quadrotor through multi-node communication in ROS2, it was crucial to have a clear understanding of its behaviour and articulate physical concepts in a simple manner. The system was designed with three nodes, each with a specific purpose: the first node received and translated hand gestures, the second node handled the quadrotor model, and the third node visualised data through animated graphs. The PID regulator was a critical component of the control system, where the results showed that the system could respond precisely to hand gestures, accomplishing the desired tasks with great accuracy. Overall, the system demonstrated the effectiveness of combining these technologies to operate a 3D quadrotor model.

Future work will involve further development of the quadrotor, creating a professional model using SolidWorks, converting it to URDF files, controlling it using ROS2 and GAZEBO, and integrating numerous sensors. The system will contain many exciting possibilities, involving machine learning and computer vision and integrating cutting-edge technology to make the system more interactive.

References

Abedzadeh Maafi, R., Etemadi Haghighi, S. and Mahmoodabadi, M.J. (2022) 'Pareto optimal design of a fuzzy adaptive sliding mode controller for a three-link model of a biped robot via the multi-objective improved team game algorithm', *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, Vol. 44, No. 9, p.428, DOI: 10.1007/s40430-022-03719-0.

Bellahcene, Z., Bouhamida, M., Denai, M. et al. (2021) 'Adaptive neural network-based robust H_∞ tracking control of a quadrotor UAV under wind disturbances', *International Journal of Automation and Control*, Vol. 15, No. 1, p.28, DOI: 10.1504/IJAAC.2021.111747.

Dey, C., Mudi, R.K. and De Maity, R.R. (2022) 'Stable optimal self-tuning interval type-2 fuzzy controller for servo position control system', *International Journal of Automation and Control*, Vol. 16, No. 5, p.594, DOI: 10.1504/IJAAC.2022.10048165.

Erős, E., Dahl, M., Bengtsson, K. et al. (2019) 'A ROS2 based communication architecture for control in collaborative and intelligent automation systems', *Procedia Manufacturing*, Vol. 38, pp.349–357, DOI: 10.1016/j.promfg.2020.01.045.

Fessi, R. and Bouallègue, S. (2019) 'LQG controller design for a quadrotor UAV based on particle swarm optimisation', *International Journal of Automation and Control*, Vol. 13, No. 5, p.569, DOI: 10.1504/IJAAC.2019.101910.

Gandhi, P. and Bhole, K. (2013) 'Characterization of 'bulk lithography' process for fabrication of three-dimensional microstructures', *Journal of Micro and Nano-Manufacturing*, Vol. 1, No. 4, p.41002, DOI: 10.1115/1.4025461.

Gandhi, P., Deshmukh, S., Ramtekkar, R. et al. (2013) ''On-axis' linear focused spot scanning microstereolithography system: optomechatronic design, analysis and development', *Journal of Advanced Manufacturing Systems*, Vol. 12, No. 1, pp.43–68, DOI: 10.1142/S0219686713500030.

García, J. and Molina, J.M. (2022) 'Simulation in real conditions of navigation and obstacle avoidance with PX4/GAZEBO platform', *Personal and Ubiquitous Computing*, Vol. 26, No. 4, pp.1171–1191, DOI: 10.1007/s00779-019-01356-4.

GAZEBO Docs (2022) [online] https://classic.gazebosim.org/api.

Goud, E.C., Rao, A.S. and Chidambaram, M. (2022) 'Novel design of PID controllers for minimum and non-minimum phase time delay processes for enhanced performance', *International Journal of Automation and Control*, Vol. 16, No. 6, p.689, DOI: 10.1504/IJAAC.2022.10050111.

Guettal, L., Chelihi, A., Ajgou, R. et al. (2022) 'Robust tracking control for quadrotor with unknown nonlinear dynamics using adaptive neural network based fractional-order backstepping control', *Journal of the Franklin Institute*, Vol. 359, No. 14, pp.7337–7364, DOI: 10.1016/j.jfranklin.2022.07.043.

Kumar, A., Mohan Singh, B., Kumar, R. et al. (2023) 'Development of modified LQG controller for mitigation of seismic vibrations using swarm intelligence', *International Journal of Automation and Control*, Vol. 17, No. 1, p.1, DOI: 10.1504/IJAAC.2023.10049079.

Meradi, D., Benselama, Z.A., Hedjar, R. et al. (2022) 'Quaternion-based nonlinear MPC for quadrotor's trajectory tracking and obstacles avoidance', in *2022 2nd International Conference on Advanced Electrical Engineering (ICAEE)*, IEEE, Constantine, Algeria, 29 October, pp.1–6, DOI: 10.1109/ICAEE53772.2022.9962052.

Omar, H.M. (2022) 'Hardware-in-the-loop simulation of time-delayed anti-swing controller for quadrotor with suspended load', *Applied Sciences*, Vol. 12, No. 3, p.1706, DOI: 10.3390/app12031706.

Orozco Soto, S.M., Ruggiero, F. and Lippiello, V. (2022) 'Globally attractive hyperbolic control for the robust flight of an actively tilting quadrotor', *Drones*, Vol. 6, No. 12, p.373, DOI: 10.3390/drones6120373.

Pakro, F. and Nikkhah, A.A. (2022) 'A fuzzy adaptive controller design for integrated guidance and control of a nonlinear model helicopter', *International Journal of Dynamics and Control*, DOI: 10.1007/s40435-022-00993-7.

Puck, L., Keller, P., Schnell, T. et al. (2020) 'Distributed and synchronized setup towards real-time robotic control using ROS2 on Linux', in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, IEEE, Hong Kong, August, pp.1287–1293, DOI: 10.1109/CASE48305.2020.9217010.

ROS Docs (2022) [online] https://docs.ros.org.

Shauqee, M.N., Rajendran, P. and Suhadis, N.M. (2021) 'An effective proportional-double derivative-linear quadratic regulator controller for quadcopter attitude and altitude control', *Automatika*, Vol. 62, Nos. 3–4, pp.415–433, DOI: 10.1080/00051144.2021.1981527.

Shirzadeh, M., Amirkhani, A., Tork, N. et al. (2021) 'Trajectory tracking of a quadrotor using a robust adaptive type-2 fuzzy neural controller optimized by cuckoo algorithm', *ISA Transactions*, Vol. 114, pp.171–190, DOI: 10.1016/j.isatra.2020.12.047.

Spanaki, K., Karafili, E., Sivarajah, U. et al. (2022) 'Artificial intelligence and food security: swarm intelligence of AgriTech drones for smart AgriFood operations', *Production Planning & Control*, Vol. 33, No. 16, pp.1498–1516, DOI: 10.1080/09537287.2021.1882688.

Trenev, I., Tkachenko, A. and Kustov, A. (2021) 'Movement stabilization of the parrot mambo quadcopter along a given trajectory based on PID controllers', *IFAC-PapersOnLine*, Vol. 54, No. 13, pp.227–232, DOI: 10.1016/j.ifacol.2021.10.450.

Yoon, J. and Doh, J. (2022) 'Optimal PID control for hovering stabilization of quadcopter using long short term memory', *Advanced Engineering Informatics*, Vol. 53, p.101679, DOI: 10.1016/j.aei.2022.101679.

Zhao, D. and Wu, R. (2023) 'Lyapunov-based MPC for nonlinear process with on-line triggered linearized model', *International Journal of Automation and Control*, Vol. 17, No. 1, p.1, DOI: 10.1504/IJAAC.2023.10046543.

International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 56, No. 3, June, 2023, pp. 415-424
Journal homepage: http://iieta.org/journals/jesa

A PyQt6-Based Platform for Real-Time Control and Monitoring of a Quadrotor Multibody System Using ROS2 and Gazebo

Hamza Djizi[1*], Zoubir Zahzouh[2]

[1] Department of Mechanical Engineering, University of Souk Ahras, Souk-Ahras 41000, Algeria
[2] Laboratoire de Recherche en Électromécanique et Sûreté de Fonctionnement, LRESF Laboratory University of Souk Ahras, Souk-Ahras 41000, Algeria

Corresponding Author Email: hamzadjizi@gmail.com

https://doi.org/10.18280/jesa.560308

Received: 13 April 2023
Accepted: 12 June 2023

Keywords:
quadrotor multibody system, ROS2, GAZEBO, control, server-client, PyQt6, proportional-derivative (PD) controller, real-time control

ABSTRACT

This paper presents a PyQt6 server-based application design for controlling a quadrotor multibody system in a simulated environment using the Gazebo 3D model and ROS2 on Linux. The combination of PyQt6 with ROS2 offers an intuitive graphical interface that simplifies access to control parameters and flight modes. The system incorporates a unique Gazebo plugin that connects to a proportional-derivative (PD) controller, providing stable quadrotor flight control. Notably, this plugin facilitates precise quadrotor movements and establishes reliable communication between the server and quadrotor, distinguishing it from other plugins. Moreover, simulation results demonstrate the effectiveness of the proposed PyQt6 server-based application in real-time quadrotor control. The results exemplify the system's capability to achieve stable and precise quadrotor movement by effectively controlling motion along the three axes (x, y, and z) along with yaw. However, the primary contribution of the system presented in this paper lies in the development of a robust PyQt6 server-based application designed to control a quadrotor multibody system. Furthermore, the system exhibits inherent potential for extension to encompass the control of a physical quadrotor, thereby substantiating its viability in real-world applications.

1. INTRODUCTION

Quadrotors are a type of unmanned aerial vehicle (UAV) with four motors, allowing them to generate force and torque. Although possessing six degrees of freedom, only the four actuators are required to control all fundamental movements. Despite this, there is instability and limited maneuverability because of the low number of actuators. Consequently, researchers have developed advanced control algorithms and feedback systems, such as linear and non-linear controllers, to enable precise control command adjustment by considering the system's input and output data. Integrating these controllers into the quadrotor's underactuated system can significantly enhance its stability and maneuverability. For instance, linear and non-linear controllers can utilize the output data from the system to determine the necessary force and torque before modifying the input instruction accordingly. Thus, this allows the quadrotor to remain stable and agile even in challenging flying conditions.

Quadrotors have recently been employed in research and development as multibody systems, which are structures fabricated of several bodies or parts connected by joints. To investigate and model these systems, researchers frequently utilize software like Gazebo, which enables them to build virtual worlds for their quadrotors and other robots, to test and model these systems. Researchers may test various control algorithms and replicate real-world situations using Gazebo Without risking damage to their physical quadrotors. The advancement of robotic systems and the creation of quadrotors both depend heavily on this technology.

Researchers use various simulation software tools to analyze and optimize the multibody systems' performance, including Webots, is a robotics simulation software is used to create realistic simulations of robots and virtual environments [1, 2]. It supports multiple robot models, such as robots and drones. This software used to facilitate the design and test of complex robotic systems. SimMechanics is multibody dynamics simulation software designed by Mathworks company, and it utilized to model and simulate mechanical systems. With this software, researchers can analyze and optimize the performance of their robots [3, 4]. ADAMS (Automatic Dynamic Analysis of Mechanical Systems) is a multibody dynamics simulation software used to model and analyze mechanical systems [5, 6]. It allows researchers to design accurate models such as robots and drones [7, 8]. GAZEBO is an open-source robotics simulation software is used to design and simulate robots in a realistic environment [9]. This 3D software allows researchers to program and add custom plugins to handle the different aspects of their robots [10]. It also provides advanced tools for simulating complex environments, including physics engines, sensors, and controllers. GAZEBO works with ROS to offer a complete solution for designing, emulating, and testing robotic systems. These software packages include features like physics-based modeling, visualization, and control design, all of which help the development of UAVs. Its combination enables an ample understanding of the quadrotor's behavior, easing the design for efficient and safe solutions.

As for the controllers, researchers use different kinds of controllers, including PID and PD controllers are among the most widely used control algorithms due to their simplicity and effectiveness [11, 12]. LQR and MPC are more advanced techniques that provide optimal control of systems with constraints [13]. SMC is a nonlinear control technique that offers robustness to disturbances and uncertainties [14]. Fuzzy Logic and Neural Networks are intelligent control techniques that allow for nonlinear mapping of inputs to outputs [15, 16]. Backstepping is a recursive design method for designing controllers for nonlinear systems [17]. The Linearized Controller is a technique used to approximate nonlinear systems by a linear one around a given operating point [18]. The selection of a control strategy relies on the system's characteristics and the particular demands of the application.

With the integration of controllers, ROS2 has become a leading platform for managing complicated robotic systems because of its streamlined features. Even though ROS2 and its tools have made tremendous advancements, there is still room for improvement, especially on the side of quadrotors. Thus, this work includes designing a platform for monitoring and controlling a quadcopter that utilizes the recently released PyQt6 toolkit. Choosing this framework was due to PyQt5's use for developing several well-known ROS2 tools, including RQT and RVIZ2. This work intends to extend the capabilities of ROS2 by utilizing PyQt6's sophisticated features for controlling and monitoring a quadrotor using modern user interfaces. Hence, we chose this framework over PyQt5 due to its superior feature set, enhanced performance, ongoing development and support within the presence of well-organized widgets and functions. However, integrating a server-client architecture will substantially contribute when developing a robust ROS2 control network, facilitating the control of the quadrotor through other devices. This network tool is a comprehensive and intuitive user interface that can provide real-time feedback on the quadrotor's performance. By leveraging PyQt6's networking capabilities, the server-client architecture could allow multiple users to monitor and control the quadrotor simultaneously. To summarize, this project aims to demonstrate the immense potential of combining ROS2 with PyQt6 to build a platform for monitoring and controlling an intelligent quadrotor equipped with several sensors, such as a depth camera, lidar, IMU, and GPS.

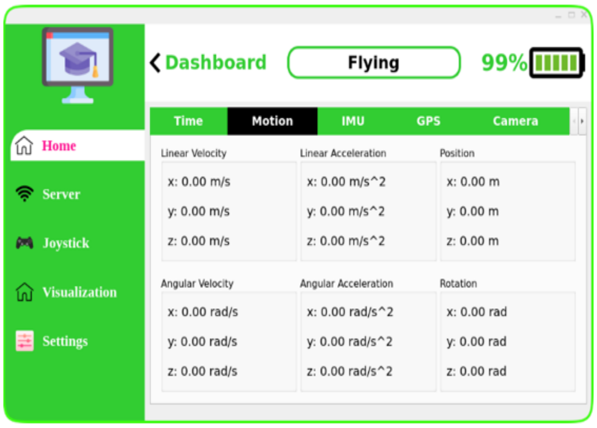
This paper is structured as follows. In section 2, the design aspects of the PyQt6 application are discussed. It covers the architecture, features, and the application, highlighting the development choices and considerations. Section 3 presents the URDF prototype of the quadrotor. It describes the design and modeling of the quadrotor using URDF, including its physical components, such as the motors and sensors. Section 4 delves into the development of a new plugin for Gazebo. The plugin enhances the capabilities of Gazebo for simulating and interacting with the quadrotor model developed in the previous section. Section 5 focuses on the overall implementation of the system and the communication protocols involved. It covers the integration of the PyQt6 application, the URDF quadrotor model, and the Gazebo plugin. In Section 6, the results obtained from the system implementation are presented and analyzed. It includes performance metrics and simulations. The final section delivers a summary of the paper, highlighting the pivotal contributions, the accomplishments, and the implications of this study.

2. PYQT6-BASED APPLICATION DESIGN

Recently, drones have gained immense popularity due to their numerous applications across industries. However, controlling a quadrotor can be challenging and requires expertise in various domains, such as robotics, control systems, and software engineering. In ROS2, there are multiple programs available for quadrotor control, including the "rqt robot steering" package with a PyQt5-based GUI and the “teleop_twist_keyboard” package with a command-line interface (CLI). These interfaces enhance the flexibility and usability of quadrotor control. However, they have limitations in their effectiveness for controlling the quadrotor. Hence, a user-friendly PyQt6 application will be created using the ROS2 network to overcome this problem and make operating a quadrotor more practical and effective. The primary objective of the application is to provide an intuitive and robust interface for controlling the quadrotor. The app features will be divided into several main sections, such as the home section for monitoring the quadrotor status, the server section for making or breaking connections with other devices like computers or smartphones, the joystick section for controlling the quadrotor, the visualization section for viewing various data, and the settings section for further customizing the application. By employing this application, users can efficiently operate the quadrotor and leverage the benefits of this technology.

2.1 Home screen design

The designed home screen in PyQt6 features a clear and concise layout (Figure 1). The current time is displayed in a large, easy-to-read font in the time section, providing users with real-time updates. The quadrotor's displacement, velocity, and acceleration are presented in a visually appealing manner, offering a clear overview of its movement dynamics. IMU data, including orientation and rotation, is displayed in a separate section for comprehensive understanding the motion of the quadrotor. The GPS data, including location and altitude, is displayed to provide essential positioning information. Finally, lidar data and distance measurements is presented, enabling an accurate assessment of the quadrotor's surroundings. Overall, the home screen design in PyQt6 is optimized for efficient monitoring of vital quadrotor data while maintaining a visually appealing and user-friendly interface.



Linear Velocity	Linear Acceleration	Position
x: 0.00 m/s	x: 0.00 m/s^2	x: 0.00 m
y: 0.00 m/s	y: 0.00 m/s^2	y: 0.00 m
z: 0.00 m/s	z: 0.00 m/s^2	z: 0.00 m

Angular Velocity	Angular Acceleration	Rotation
x: 0.00 rad/s	x: 0.00 rad/s^2	x: 0.00 rad
y: 0.00 rad/s	y: 0.00 rad/s^2	y: 0.00 rad
z: 0.00 rad/s	z: 0.00 rad/s^2	z: 0.00 rad

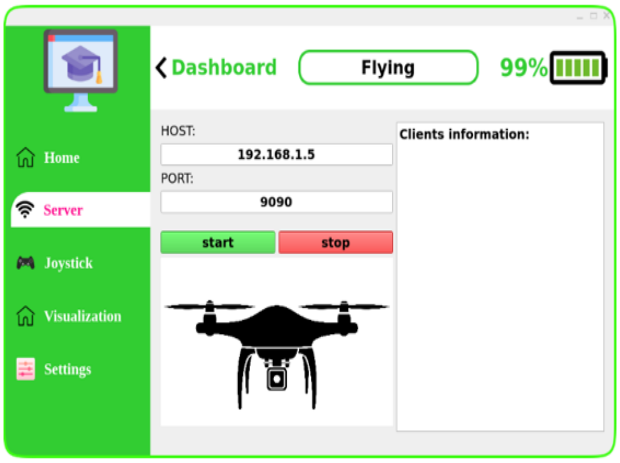
Figure 1. Home screen for real-time monitoring and control of quadrotor system

The application home screen is designed to update in real time, utilizing the spin function in the ROS2 node and the threading protocol. By employing the spin function, the node can remain active and continuously processes incoming messages and events from the ROS2 network. It allows for immediate updates on the home screen, ensuring that every change or new data are promptly displayed to the user. The threading protocol further enhances this capability by running the spin function on a separate thread, enabling concurrent execution and preventing potential delays or freezes in the user interface. Consequently, the home screen maintains a dynamic and up-to-date representation of the quadrotor's status, providing users with real-time information and facilitating efficient control and monitor of the system.

2.2 Server screen design

To create a server using the socket package in Python, we can start by specifying a host and port number, which the server will use to connect clients. The server then can be written to include two buttons: a "start" button and a "stop" button, created using the PyQt6 library widgets. Clicking the "start" button initiates the server and begins listening for incoming connections. Meanwhile, clicking the "stop" button shuts down the server and disconnects the active clients. Additionally, to Secure reliable and ordered data transmission, the server can utilize the TCP (Transmission Control Protocol) protocol instead of UDP. TCP provides reliable, connection-oriented communication, guaranteeing delivery and in-order arrival of data packets. By combining the socket package, PyQt6 library, and TCP protocol, a robust and user-friendly server that facilitates secure and reliable communication and data transfer between multiple devices can be designed (Figure 2).

To ensure real-time control addressing potential latency issues and implementing appropriate measures is crucial. The system may encounter challenges like data overlapping and connectivity issues. Thus, to address these concerns, the application utilizes the try function in Python. By employing this function, the application can effectively handle and disregard any latency problems that may arise, allowing the system to maintain smooth functionality despite intermittent delays or disruptions. As a result, the application provides a seamless and uninterrupted real-time control experience, significantly enhancing the system's reliability and responsiveness.



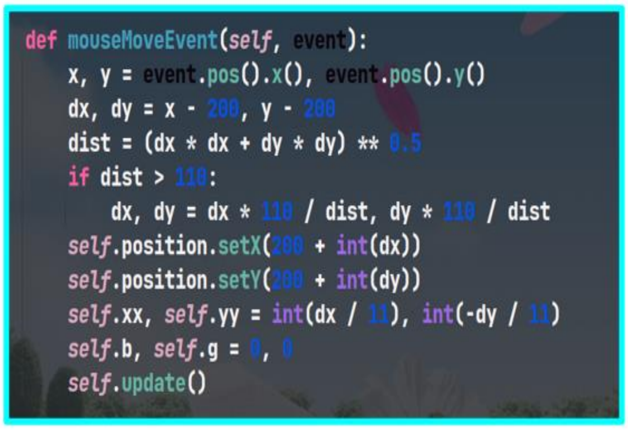
HOST:
192.168.1.5
PORT:
9090
Clients information:

Figure 2. Server screen to start and stop the server

2.3 Joystick screen design

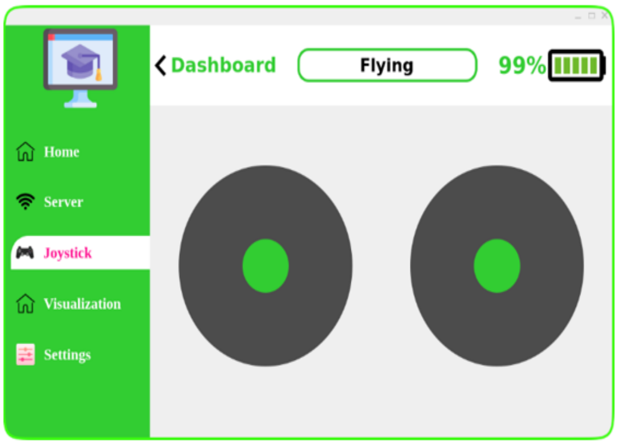
To create a new PyQt6 application featuring two joysticks, the essential first step is to import the necessary libraries, including PyQt6 and math. Two joysticks can then be created using QPainter and paintEvent, with the drawEllipse method to draw the circles required for the joysticks. Then, the two joysticks should be placed in a single widget using QHBoxLayout and included in a new QWidget class. Three interactive functions, mousePressEvent, mouseMoveEvent, and mouseReleaseEvent, can then be added to detect mouse clicks, movements, and releases to enhance the app's functionality. Finally, to restrict the motion of the joysticks to within the circle's boundaries, the application of the distance formula is necessary (Figure 3). This technique effectively constrains the joysticks' movement within the prescribed circle. Following these steps helps us to design an interactive PyQt6 application with two joysticks to control the quadrotor.

Once the joystick for quadcopter control is created, the joystick data needs to be scaled to manage the quadcopter's four fundamental movements. These movements include yaw, forward and backward, left and right, up, and down. Mapping the input values from the joystick to the required output range for each movement includes scaling the joystick data. It ensures that the quadrotor flies in a predictable and regulated manner, precisely translating the operator's motion to the quadrotor's actions. It is possible to control the quadrotor's movements precisely and quickly by sending the scaled joystick data to the quadrotor control system.



```
def mouseMoveEvent(self, event):
    x, y = event.pos().x(), event.pos().y()
    dx, dy = x - 200, y - 200
    dist = (dx * dx + dy * dy) ** 0.5
    if dist > 110:
        dx, dy = dx * 110 / dist, dy * 110 / dist
    self.position.setX(200 + int(dx))
    self.position.setY(200 + int(dy))
    self.xx, self.yy = int(dx / 11), int(-dy / 11)
    self.b, self.g = 0, 0
    self.update()
```

(a) the program that controls how the joystick moves



(b) Joystick screen design

Figure 3. The joystick screen on which the quadrotor is controlled

In order to achieve smooth control, the joystick's sensitivity has been fine-tuned to strike a balance between being too sensitive or less sensitive. This optimization ensures that the quadrotor responds accurately to even subtle movements of the joystick, enabling precise control over its motion. The sensitivity is adjusted based on the quadrotor's linear velocity along the three axes and its angular velocity around the z-axis. The linear velocity is constrained within the range of -8 to 8m/s, while the angular velocity is confined to -0.4 to 0.4rad/s. As for the responsiveness, the joystick's responsiveness is significantly improved by utilizing the DDS (Data Distribution Service) protocol. DDS facilitates efficient real-time data exchange, ensures reliable and secure communication, reduces latency, and maximizes the joystick's responsiveness for accurate on-screen actions.

However, the quadrotor system receives joystick data through the integration of two ROS2 nodes. The first node, integrated with the PyQt6 application, receives and publishes joystick data via topics. The second node which is integrated with the Gazebo plugin, actively spins and receives real-time data through subscriptions. This data is subsequently utilized to control and execute actions on the motors.

3. DEVELOPING GAZEBO 3D MODEL

Gazebo, with its realistic physics, sensor simulation, control integration, and flexibility, plays a crucial role in developing 3D quadrotors using URDF and SDF. As an open-source tool, it has demonstrated its value in creating and evaluating robotic systems, driving advancements in robotics. Its precise quadrotor dynamics simulation and seamless URDF and SDF integration empower developers to design, test, and enhance 3D quadrotor systems in a simulated environment.

Creating a multi-body quadrotor using URDF and xacro involves several steps. First, the basic structure of the quadrotor, such as the body shape, the length of arms, and the number of rotors, need to be defined in the URDF file. Next, the joints that connect the different components of the quadrotor, such as the rotors and the body, need to be specified. These joints enable the quadrotor to move and articulate realistically. After defining the basic structure and joints, sensors can be added to the quadrotor model. Four commonly used sensors are the LIDAR, depth camera, IMU, and GPS. Each sensor in Gazebo is connected to the quadrotor through specific joints and associated links. For instance, the camera utilizes the camera_link as its reference frame. The IMU relies on the IMU_link. The GPS is connected through the GPS_joint and references the GPS_link, while the LIDAR sensor uses the lidar_link as its reference frame and is connected via the lidar_joint. This well-defined linkage enables accurate positioning and interaction between the quadrotor and its various sensors within the simulation.

The LIDAR sensor creates a 3D point cloud of the surroundings by measuring the distances to objects in the environment using lasers. Similarly, the depth camera produces a detailed depth map of the quadrotor's surroundings by employing infrared sensors to measure distance. Autonomous systems often rely on these sensors for accurate, reliable navigation and obstacle avoidance. The quadrotor's acceleration, angular velocity, and orientation are all measured using the inertial measurement unit or IMU. When there are outside disturbances present, this sensor is crucial for maintaining the quadrotor's orientation and stabilization. Finally, location and velocity data are provided by the GPS sensor. In outdoor conditions, this sensor is helpful with its ability to provide real-time location information. The GPS sensor is a valuable tool for navigation, localization, and mapping applications. Thus, integrating these sensors into a multi-body quadrotor URDF model can improve its functionality and let it fly by itself in a wide range of settings. The combination of these sensors may introduce potential obstacles, including simulation limitations, computational performance considerations, and challenges related to data integration and synchronization. These factors need to be carefully addressed to ensure accurate and reliable sensor fusion within the system.

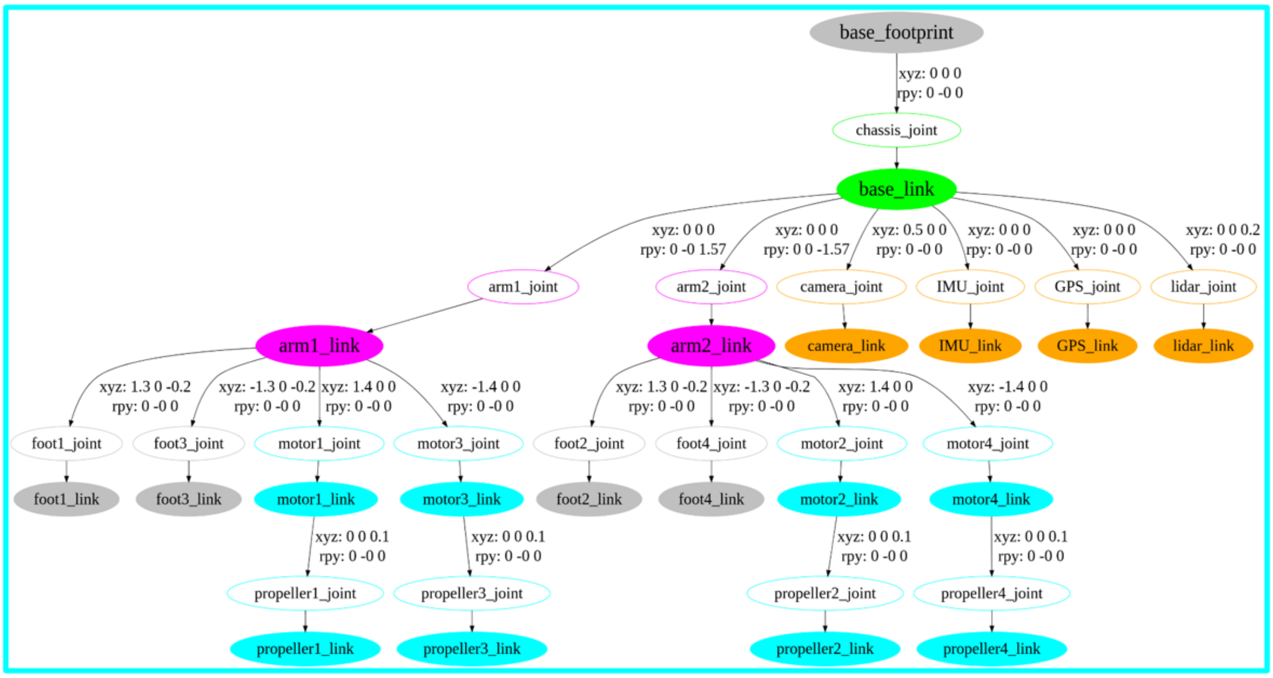


Figure 4. Graphical representation of the hierarchical quadrotor system including the sensors

Figure 4 illustrates the design of a quadrotor model in URDF format, which includes the four rotors and the main body of the quadrotor. The sensors: LIDAR, depth camera, IMU, and GPS, have been linked to the quadrotor design using visual tags in URDF. Through these tags, the sensors can be positioned and oriented precisely within the quadrotor's frame of reference, enabling them to provide critical data to the quadrotor's control systems. The depth camera is positioned at the front of the quadrotor to improve obstacle detection and 3D mapping capabilities, aided by the centrally located lidar. Meanwhile, the IMU and GPS at the center measure the quadrotor's acceleration, velocity, and position. By integrating these sensors into the quadrotor's design in URDF, the quadrotor can accurately perceive its environment and navigate through it with precision and stability. In addition, the URDF quadrotor system must be built on TF2, a potent tool that enables us to track coordinate frames in a ROS2 network. TF2 (Transform Library 2) is a software library that provides a mechanism for managing coordinate frame transformations. It allows for the conversion and alignment of coordinate frames between different sensors, robots, or platforms within a distributed system. TF2 also enables the utilization of sensor data to detect a robot's limits through coordinate frame transformations. Sensor data, acquired from LIDAR, cameras, or proximity sensors, undergoes transformation to the robot's base frame using TF2. Collision detection algorithms assess whether the robot approaches or surpasses limits by checking for obstacles, proximity to objects, or joint angles. Feedback from sensed limits aids in adjusting trajectory and actions through motion planning algorithms, ensuring real-time limit awareness within the ROS2 network.

In order to ensure successful operation of the quadrotor system, it is crucial to establish accurate coordination and linkage among multiple connections and joints. With the aid of TF2, we can create a hierarchy between these elements and precisely determine their locations and orientations concerning one another. It is essential for maintaining the quadrotor's stability, predictability, and control of its movements. The quadrotor system can connect and cooperate with other nodes in the network thanks to TF2's smooth integration into a broader ROS2 network.

4. DEVELOPING GAZEBO PLUGIN

The design process of a new C++ based plugin involves specifying the quadrotor's behavior and implementing it using GAZEBO's API. Importing GAZEBO libraries, ROS2 functions, ROS2 interfaces, and the PD function is necessary for controlling the quadrotor's movement. The main file code must include constants for maximum height, speed, and battery duration, as well as variables for the quadrotor's state, position, velocity, thrust, time, and torque. Functions for controlling linear and angular velocities and adding linear force are also essential. Accurate definitions of links and joints are crucial as they define the quadrotor's physical structure, including position, orientation, base frame, motors, propellers, and sensors. Matching the link and joint names with the URDF files ensures realistic movements and appropriate responses to external forces, maintaining the integrity of the quadrotor's components.

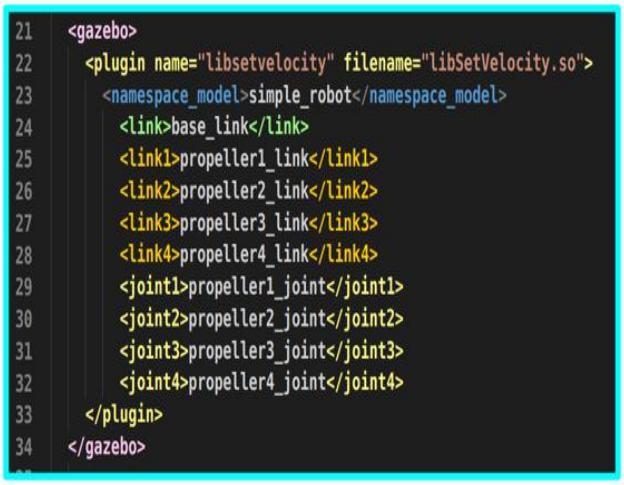
The plugin is based on the fundamental principles of physics, specifically Newton's second law of motion, which is instrumental in governing the dynamics of the quadrotor multibody system. In addition, different flight conditions are considered, such as hovering, ascending, descending, taking off, and landing, and adjusts the control inputs accordingly. The plugin also includes a Proportional-Derivative (PD) controller, which enables precise control over the quadrotor's movements. The PD gains are meticulously and precisely tuned using a dedicated PyQt6 interface, significantly enhancing the quadrotor's prompt response to commands.

As a ROS2 node, the plugin is equipped with subscriptions and publishers, thus enabling communication with other nodes within the ROS2 network. Moreover, the plugin presents interfaces for various data types, including AccelerationData, DroneState, DroneTime, VelocityData, ForceTorqueData, OnOffState, and PositionData. Each of these interfaces encompasses essential data types, including float32, int32, and string, to fulfill the requirement of the quadrotor. These interfaces are transmitted between the nodes using topics, enabling seamless access and data manipulation by other nodes in the network. These topics includes: */dh_drone/drone_state*, */dh_drone/force_torque_data*, */dh_drone/velocity_data*, */dh_drone/acceleration_data*, */dh_drone/command_velocity*, */dh_drone/time_data*, */dh_drone/on_off_state*, */dh_drone/position_data*.

However, the communication between nodes using topics and interfaces enhances the plugin's importance as a powerful tool for effectively controlling the quadrotor system.

In order to integrate the GAZEBO plugin with the URDF quadrotor model, we must include a new section to the main URDF file. This section defines the movable links and joints of the quadrotor, and contains the necessary plugin to control the system (Figure 5). The plugin will then apply forces and torques to the links that can be moved, allowing for precise control over the quadrotor's movements. This process is essential to ensure the proper functioning of the plugin with the quadrotor model, providing accurate and reliable communication throughout the simulation.

The main URDF file serves as the central definition for the quadrotor multibody system, encompassing various components through the utilization of the xacro macro language. This facilitates the inclusion of quadrotor parts such as drone, drone constants, inertial macros, lidar, GPS, camera, and imu. In conjunction with the Gazebo plugin, these individual xacro files collectively define the structure and characteristics of the quadrotor within the URDF specification.



```
<gazebo>
  <plugin name="libsetvelocity" filename="libSetVelocity.so">
    <namespace_model>simple_robot</namespace_model>
      <link>base_link</link>
      <link1>propeller1_link</link1>
      <link2>propeller2_link</link2>
      <link3>propeller3_link</link3>
      <link4>propeller4_link</link4>
      <joint1>propeller1_joint</joint1>
      <joint2>propeller2_joint</joint2>
      <joint3>propeller3_joint</joint3>
      <joint4>propeller4_joint</joint4>
  </plugin>
</gazebo>
```

Figure 5. Integration of Gazebo plugin into the main URDF file

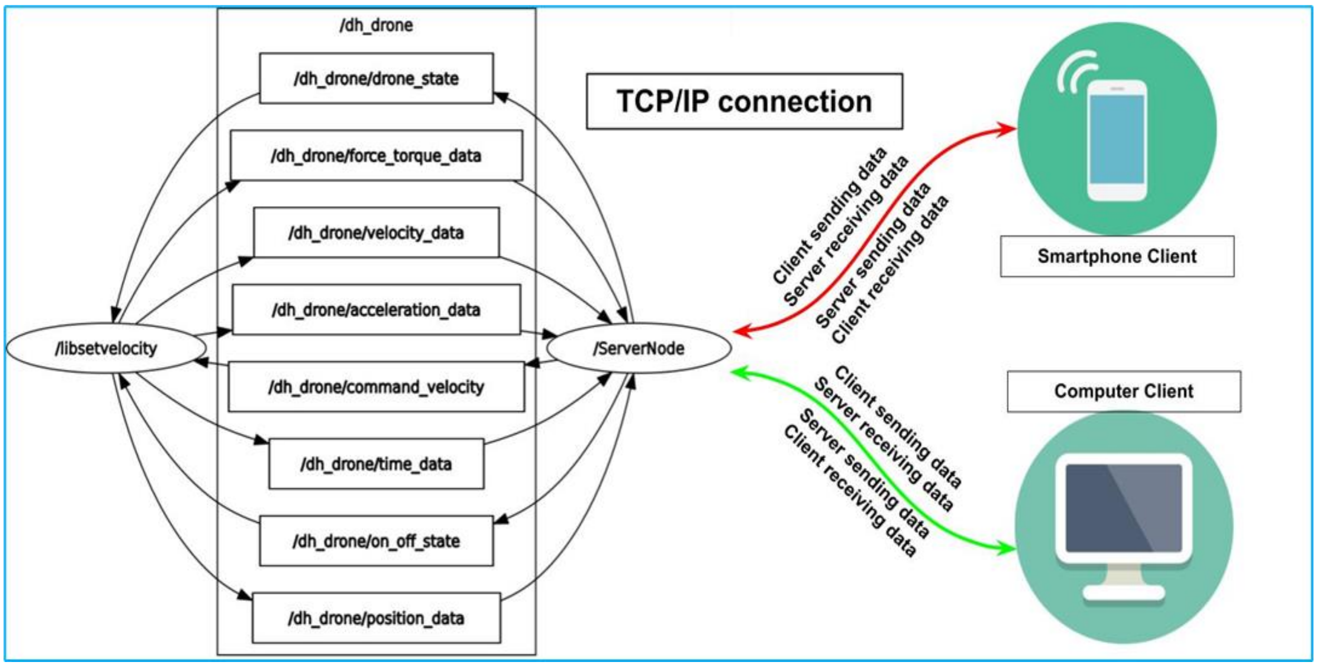


Figure 6. A graphical representation of the ROS2 network and communication model, demonstrating server-client communication and TCP/IP protocols with ROS2 nodes

5. IMPLEMENTATION AND COMMUNICATION

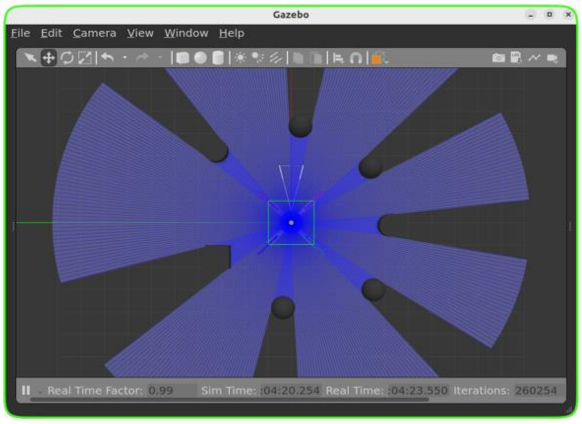
The process of constructing the system involves creating three packages within the source directory of the workspace: quadrotor_pkg, msgs_pkg, and gazebo_plugin_pkg. Each of the packages encompasses different dependencies necessary for the system's functionality. The quadrotor_pkg contains the essential components such as URDF files, launch files, and Python scripts responsible for node creation and the PyQt6 application. The msgs_pkg is dedicated to housing the required interfaces for seamless system operation. The gazebo_plugin_pkg incorporates the Gazebo plugin, which, after project building, will be exported to ensure optimal integration with the system.

The implementation of the system will be done after finishing the main programs, including the gazebo plugin, quadrotor URDF design, and PyQt6 application, and initializing a ROS2 project. It is essential to create the necessary packages for the network. This step involves creating a new package, defining the dependencies and message types, and setting up the nodes for communication. Thorough testing of the quadrotor system is required following the launch of the ROS2 project. This process includes verifying that the gazebo plugin can control the quadrotor in different flight conditions, that the PyQt6 application can interface with the plugin to provide user control, and that the ROS2 network is correctly working, allowing nodes to communicate and exchange data. Thorough testing is imperative to ascertain the quadrotor system's reliability, accuracy, and suitability for real-time applications, instilling confidence in its performance.

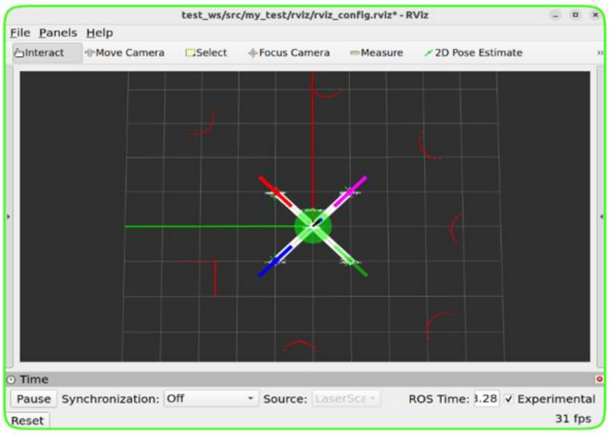
The communication between ROS2 nodes in the quadrotor system is essential. The server node, responsible for receiving clients input and sending commands to the quadrotor node, communicates with the quadrotor node via a series of topic subscriptions and publishers (Figure 6). The quadrotor node provides the server node with data related to the drone's state, force and torque information, velocity data, acceleration data, time data, on-off state, and position data. Thus, this data is then utilized by the server node to generate commands, which are subsequently sent back to the quadrotor node for precise control over its movements.

Moreover, the server node, which was programmed using Python with the help of the socket and struct libraries, uses the TCP/IP protocol to communicate with clients. Data about the quadrotor status, force and torque data, velocity and acceleration data, time data, on-off state, and location data are all transmitted via the server node. The quadrotor may then be moved and updated in its status for clients. This protocol enables multiple applications for the quadrotor system by offering adaptable and dependable communication between the server node and clients. Thus, the quadrotor system may work effectively and correctly thanks to the TCP/IP protocol and excellent communication between nodes using DDS (Data Distribution Service), making it a precious tool for many real-time applications.

The quadrotor with LiDAR simulation was conducted in a controlled testing environment within Gazebo and ROS2. The specific setup involves placing various obstacles, such as static objects, within the simulated environment (Figure 7). The testing procedures consisted of executing predefined flight paths and maneuvers while collecting LiDAR sensor data.


Gazebo
File Edit Camera View Window Help
Real Time Factor: 0.99 Sim Time: 04:20.254 Real Time: 04:23.550 Iterations: 260254

(a) Quadrotor model on gazebo with the lidar sensor



(b) Lidar sensor data on RViz2

Figure 7. Visualization of LiDAR sensor data using RViz2: a graphical representation of point cloud data captured by the sensor

Throughout the simulation, the quadrotor's LiDAR sensor accurately detected and calculated the distances between the quadrotor and the objects in its surroundings, providing measurements in meters. This distance measurement is a critical metric for evaluating the system's effectiveness. The assessment of performance criteria encompassed analyzing the accuracy of distance calculations, the speed of obstacle detection, and the system responsiveness.

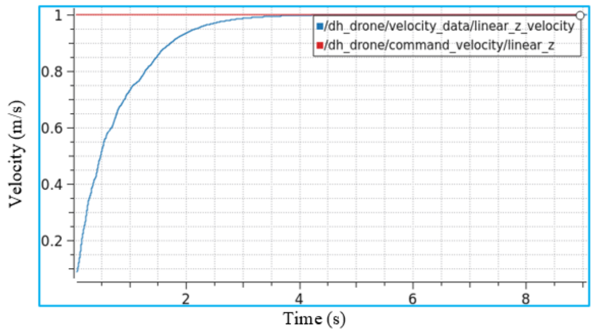
The simulation results were highly encouraging, with the system effectively detecting and calculating distances to obstacles. These findings underscore the potential for developing an advanced obstacle avoidance algorithm, enabling the quadrotor to navigate complex environments while prioritizing safety.

6. RESULTS AND DISCUSSION

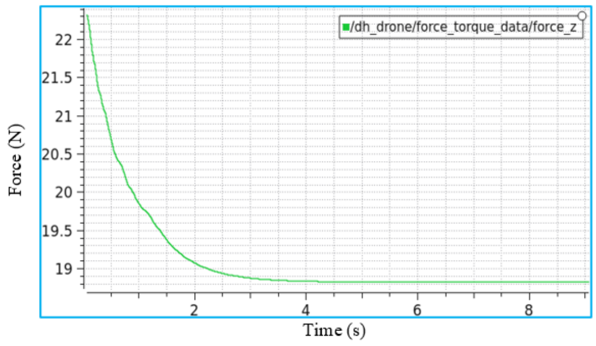
To obtain the results that confirm the system's effectiveness, the Plotjuggler tool, an open-source tool renowned for visualizing real-time data in ROS2 applications, is employed. This tool enables quickly and easily plotting data from different topics and nodes in a ROS2 system. Plotjuggler works by subscribing to ROS2 topics and receiving data in real-time. It then uses customizable plots to display this data intuitively and interactively. With Plotjuggler, users can easily monitor and debug their ROS2 applications. They also can gain insights into the behavior of the system. It can also help them to improve the performance and reliability of systems.

However, Plotjuggler listens to the integrated node within the Gazebo plugin, along with the node that publishes velocity commands. It acquires data from the following topics: /dh_drone/force_torque_data, /dh_drone/velocity_data, /dh_drone/acceleration_data, /dh_drone/command_velocity, /dh_drone/time_data, and /dh_drone/position_data. These topics transmit interfaces containing data types such as int32 and float32, which describe quadrotor parameters, including velocity, acceleration, position, torque, and force.

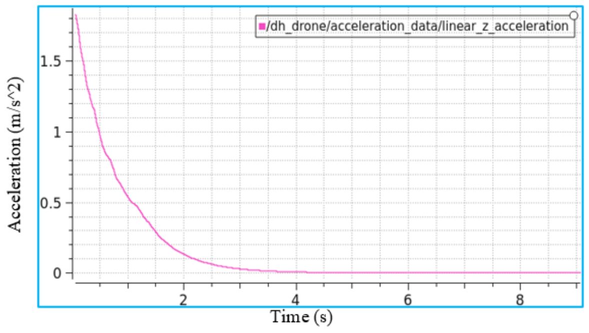
Moreover, the comprehensive analysis of the quadrotor system encompassed an evaluation of key performance metrics and criteria, including velocity, acceleration, force, and motion control, along the x, y, and z axes, as well as yaw motion. The illustrated results in Figures 8, 9, 10, and 11 highlight the quadrotor's exceptional maneuverability and control achieved through the developed control system. Notably, the quadrotor demonstrates precise movements along the x, y, and z axes, ensuring high control levels and stability during flight. Furthermore, its yaw motion, enabling rotation around the vertical axis, exhibits remarkable responsiveness and accuracy. These outcomes can be attributed to the meticulous control of the four rotors, facilitating precise adjustments to the quadrotor's thrust and orientation. The evaluation process considered various factors, including response time, stability, and control accuracy, which play a role to the system's outstanding performance.



(a) Results of altitude: velocity command and response



(b) Results of altitude: generated force response



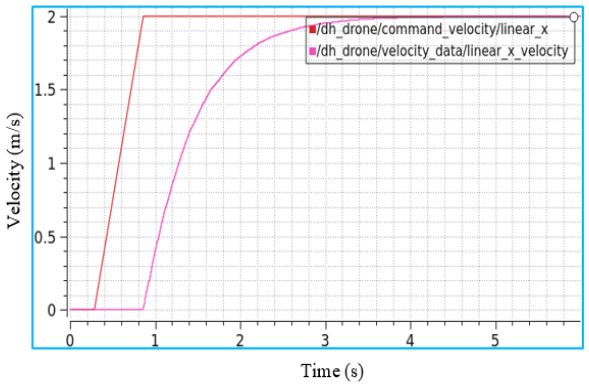
(c) Results of altitude: acceleration response

Figure 8. Results of optimized altitude control of quadrotor achieved: precise control commands enhance velocity, force, and acceleration

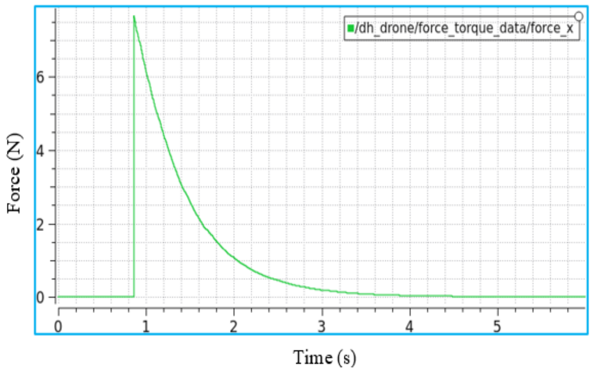
Quantitative analysis of the quadrotor's performance unveils promised statistics. During the evaluation process, the quadrotor exhibited outstanding capabilities in various domains. Along the z-axis, according to the given velocity of 1m/s. The quadrotor demonstrated swift acceleration, reaching an average of 1.8m/s^2. In terms of force, it exerted an average thrust of 22.3 N. Moving on to the motion along the x and y axes, the evaluation velocity stood at 2m/s. The quadrotor showcased rapid acceleration, averaging at 4m/s^2, while exerting an average force of 7.5 N. As for the yaw motion, the evaluation velocity measured 0.2rad/s. Remarkably, the quadrotor demonstrated swift acceleration, reaching an average of 0.33rad/s², complemented by an average torque of 0.72 N.m.

Despite the satisfactory results, it is crucial to acknowledge that the quadrotor system still possesses certain limitations that present opportunities for further development from various aspects. Firstly, in terms of velocity, although the quadrotor reached a maximum speed of 8m/s, exploring methods to enhance its velocity and stability performance would open doors to applications that demand higher velocities and swift maneuverability. Additionally, while the quadrotor demonstrated rapid acceleration, improvements can be made to improve its agility and responsiveness, enhancing its ability to navigate seamlessly through complex environments replete with dynamic obstacles. Moreover, increasing the force and thrust capabilities of the quadrotor would enable it to handle more demanding tasks and payloads, expanding its range of potential applications. Furthermore, refining the control algorithms and mechanisms associated with the yaw motion can contribute to better stability and precision during rotational maneuvers; and ensure the quadrotor's adaptability in scenarios requiring intricate movements. It is through addressing these limitations and pursuing further advancements that the quadrotor system can continue to evolve and achieve new heights of performance and versatility.

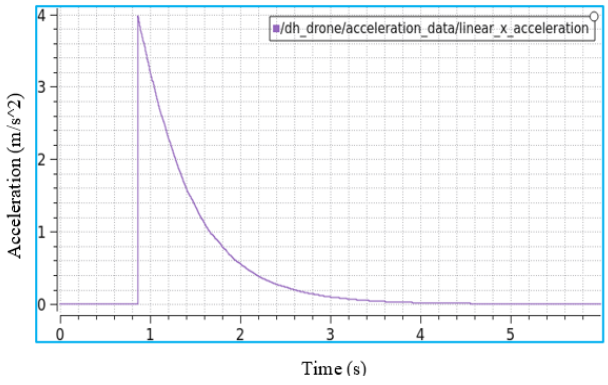
In summary, the impressive obtained statistics suggest that the quadrotor system exhibits minimal deviation in control. The consistent values for velocity, acceleration, force, and torque highlight its reliability and precision, ensuring stable and accurate flight maneuvers. Thus, the analysis confirms the quadrotor system's exceptional attributes and establishes its suitability for multiple applications such as Aerial Surveillance and Monitoring, Search and Rescue Operations, Industrial Inspections, and Agriculture and Crop Monitoring.



(a) Results of x motion: velocity command and response

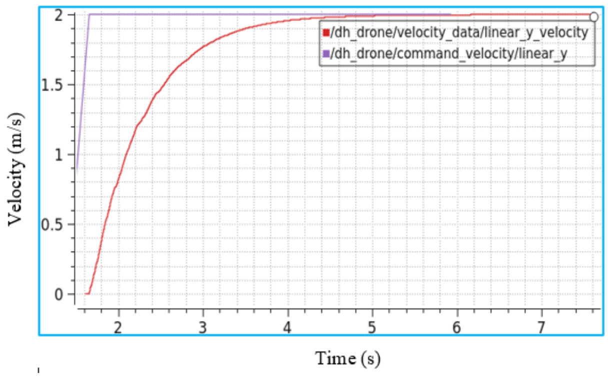


(b) Results of x motion: generated force response

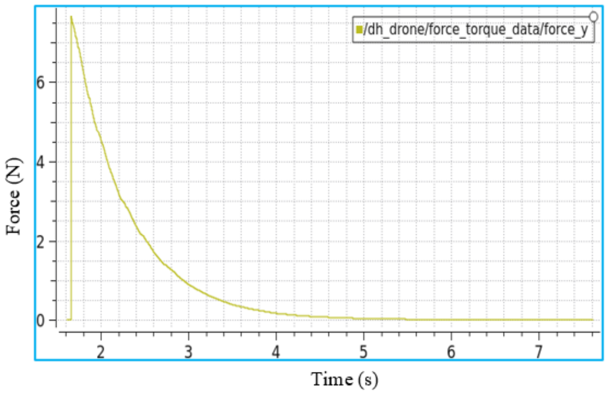


(c) Results of x motion: acceleration response

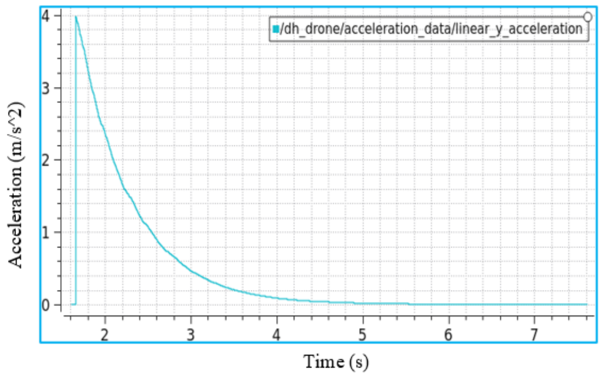
Figure 9. Results of optimized x motion control of quadrotor achieved: precise control commands enhance velocity, force, and acceleration



(a) Results of y motion: velocity command and response

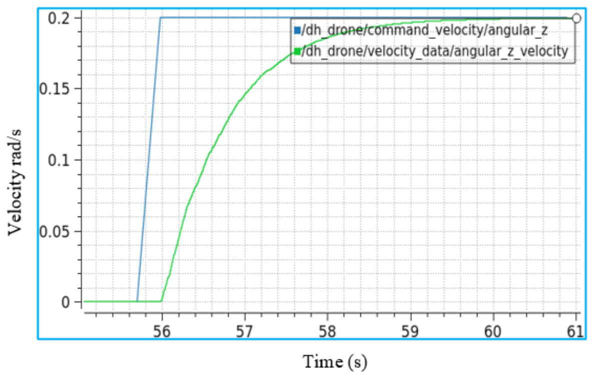


(b) Results of y motion: generated force response

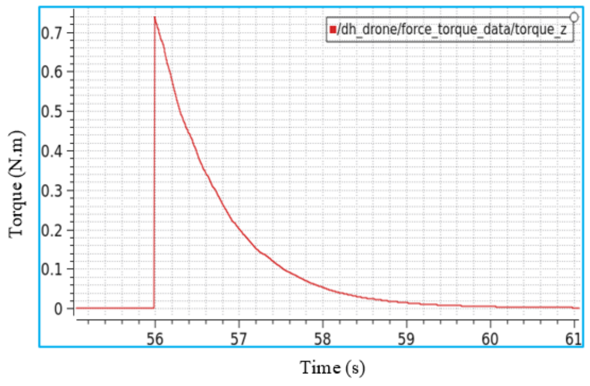


(c) Results of y motion: acceleration response

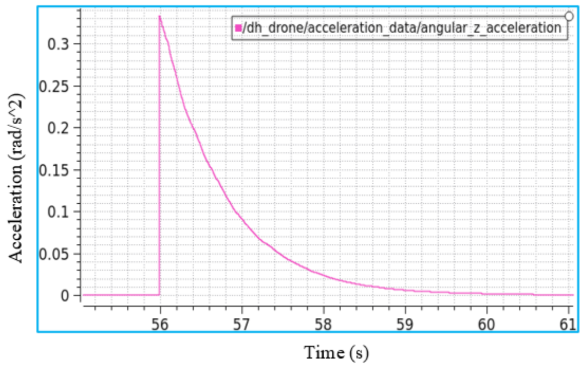
Figure 10. Results of optimized y motion control of quadrotor achieved: precise control commands enhance velocity, force, and acceleration



(a) Results of yaw motion: velocity command and response



(b) Results of yaw motion: generated torque response



(c) Results of yaw motion: acceleration response

Figure 11. Results of optimized yaw motion control of quadrotor achieved: precise control commands enhance velocity, force, and acceleration

7. CONCLUSION

In this work, the PyQt6 application played a crucial role in conveniently managing the quadrotor's movements, demonstrating advancements in robotics and control systems. It enabled effective control of the quadrotor's motions, contributing to project success and enhancing maneuverability in quadrotor multibody systems. The findings collected in this study affirm the system's efficacy in governing the quadrotor's motions, including motion along the three axes (x, y, and z), in addition to the yaw motion. Moreover, this project significantly contributes to the field of quadrotor multibody systems, paving the way for further advancements in the control and monitoring of complex systems. While this work highlights several achievements, it is essential to acknowledge its limitations. Scalability, robustness, and adaptability to different environments are some of the challenges that future researchers should consider. By addressing these aspects, the system has the potential for further enhancement to cater to the requirements of diverse applications. By setting sights on the future, many projects can leverage advanced technologies like SLAM and yolov8 to enhance the quadrotor's intelligence in obstacle avoidance, mapping, and object detection. The integration of machine learning techniques holds the potential for achieving autonomous quadrotor operation, thereby driving notable progress in the sector of quadrotor multibody systems. Overall, this project has successfully contributed to the control systems of quadrotors, with implications extending beyond this specific domain. The potential impact encompasses the development of autonomous aerial vehicles, enhancing search and rescue operations, and enabling remote sensing applications.

REFERENCES

[1] Ugurlu, H.I., Pham, X.H., Kayacan, E. (2022). Sim-to-real deep reinforcement learning for safe end-to-end planning of aerial robots. Robotics, 11(5): 109. https://doi.org/10.3390/robotics11050109

[2] Vazquez, E.M.C., Merino, A.D.P., Torres, C.M., Etcheverry, G. (2019). Simulación de un cuadri-rotor en el software webots. Memorias del Congreso Nacional de Control Automático, 67-72.

[3] Jatsun, S., Lushnikov, B., Emelyanova, O., Leon, A.S.M. (2021). Synthesis of simmechanics model of quadcopter using solidworks cad translator function. In Proceedings of 15th International Conference on Electromechanics and Robotics" Zavalishin's Readings", Springer Singapore, 125-137. https://doi.org/10.1007/978-981-15-5580-0_10

[4] Lv, Z.Y., Zhao, Q., Li, S.M., Wu, Y.H. (2022). Finite-time control design for a quadrotor transporting a slung load. Control Engineering Practice, 122: 105082. https://doi.org/10.1016/j.conengprac.2022.105082

[5] Xin, H.B. (2021). Design and analysis of retractable structure of new quadrotor landing gear. In Journal of Physics: Conference Series, IOP Publishing, 1750(1): 012022. https://doi.org/10.1088/1742-6596/1750/1/012022

[6] Xu, J.L., Hao, Y.P., Wang, S.T. (2022). Flight control simulation and flight test of foldable rotor UAV. In Journal of Physics: Conference Series, IOP Publishing, 2252(1): 012052. https://doi.org/10.1088/1742-6596/2252/1/012052

[7] Hamed, A., Fanni, M., Ahmed, S., Sameh, A. (2020). Hybrid guidance of quadrotor manipulation system for indoor-outdoor active tasks. International Journal of Mechanical & Mechatronics Engineering, 20(04): 1-12.

[8] Six, D., Briot, S., Chriette, A., Martinet, P. (2017). The kinematics, dynamics and control of a flying parallel robot with three quadrotors. IEEE Robotics and Automation Letters, 3(1): 559-566. https://doi.org/10.1109/LRA.2017.2774920

[9] Zhu, J.C., Xu, C. (2017). A comprehensive simulation testbench for aerial robot in dynamic scenario using gazebo-ros. In 2017 Chinese Automation Congress (CAC), IEEE, 7664-7669. https://doi.org/10.1109/CAC.2017.8244165

[10] Xie, Y.C., Li, Y.Z., Dong, W. (2022). Behavior

prediction based trust evaluation for adaptive consensus of quadrotors. Drones, 6(12): 371. https://doi.org/10.3390/drones6120371

[11] Labbadi, M., Cherkaoui, M., El Houm, Y., Guisser, M. (2019). A comparative analysis of control strategies for stabilizing a quadrotor. In Information Systems and Technologies to Support Learning: Proceedings of EMENA-ISTL, Springer International Publishing, 111: 625-630. https://doi.org/10.1007/978-3-030-03577-8_68

[12] Shauqee, M.N., Rajendran, P., Suhadis, N.M. (2021). An effective proportional-double derivative-linear quadratic regulator controller for quadcopter attitude and altitude control. Automatika: Časopis za Automatiku, Mjerenje, Elektroniku, Računarstvo i Komunikacije, 62(3-4): 415-433. https://doi.org/10.1080/00051144.2021.1981527

[13] Okasha, M., Kralev, J., Islam, M. (2022). Design and experimental comparison of PID, LQR and MPC stabilizing controllers for parrot mambo mini-drone. Aerospace, 9(6): 298. https://doi.org/10.3390/aerospace9060298

[14] Huang, S.R., Yang, Y.N. (2022). Adaptive neural-network-based nonsingular fast terminal sliding mode control for a quadrotor with dynamic uncertainty. Drones, 6(8): 206. https://doi.org/10.3390/drones6080206

[15] Ginting, A.H., Doo, S.Y., Pollo, D.E., Djahi, H.J., Mauboy, E.R. (2022). Attitude control of a quadrotor with fuzzy logic controller on so (3). Journal of Robotics and Control (JRC), 3(1): 101-106. https://doi.org/10.18196/jrc.v3i1.12956

[16] Li, J.H., Mou, S.H., Zhang, D.H. (2021). A novel adaptive robust control algorithm for quadrotor UAV. In 2021 6th International Conference on Robotics and Automation Engineering (ICRAE), IEEE, 50-54. https://doi.org/10.1109/ICRAE53653.2021.9657806

[17] García, O., Ordaz, P., Santos-Sánchez, O.J., Salazar, S., Lozano, R. (2019). Backstepping and robust control for a quadrotor in outdoors environments: An experimental approach. IEEE Access, 7: 40636-40648. https://doi.org/10.1109/ACCESS.2019.2906861

[18] Shakeel, T., Arshad, J., Jaffery, M.H., Rehman, A.U., Eldin, E.T., Ghamry, N.A., Shafiq, M. (2022). A comparative study of control methods for X3D quadrotor feedback trajectory control. Applied Sciences, 12(18): 9254. https://doi.org/10.3390/app12189254

DOI: 10.2478/sbeef-2023-0003

QUADCOPTER PROTOTYPE STABILITY ASSESSMENT WITH PID CONTROLLER AND EULER-LAGRANGE APPROACH

[1,2]HAMZA DJIZI, [3]ZOUBIR ZAHZOUH, [4]AZZEDINE BOUZAOUIT
[1]Department of Mechanical Engineering, Mohamed Cherif Messaadia University, P.O. Box 1553, Souk-Ahras, 41000, Algeria.
[2]INFRA-RES Laboratory, Mohamed Cherif Messaadia University, Algeria
[3]Laboratoire de Recherche en Électromécanique et Sûreté de Fonctionnement, LRESF Laboratory, Mohamed Chérif Messaadia University, P.O. Box 1553, Souk-Ahras, 41000, Algeria.
[4]University of 20 August 1955 Skikda, Algeria
Email: hamzadjizi@gmail.com

Abstract. ***The increasing use of drones in various fields has led to their popularity in developed countries due to their ease of use and manufacture. This Miniature Pilotless Aircraft has numerous beneficial usages such as express shipping, gathering information, crop monitoring, cargo transport, storm tracking, geographic mapping of inaccessible terrain, search and rescue operations, among others. This study aims to investigate the stability of a quadcopter through simulations based on the mathematical model that describes the quadcopter's dynamic and flight mechanics, using the Euler-Lagrange approach. It conducts simulations in MATLAB and present the principles that govern quadcopter stability, focusing on setting the PID coefficients to achieve optimal stability. This study provides insights into the principles of drone mechanics and stability, enabling us to better understand the quadcopter's behavior and performance.***

Keywords: simulation, quadcopter, command, stability, PID

1. INTRODUCTION

Quadcopters have spread quickly across a variety of sectors because to their adaptability and simplicity of usage, making them a popular alternative for many companies looking for effective solutions. Quadcopters may carry out a wide range of tasks as an Unmanned Aerial Vehicle (UAV), including package delivery, crop monitoring, search and rescue missions, and aerial videography. Quadcopters are essential in current operations because of their small size and agility, which allow them to negotiate difficult terrain and reach remote areas. As a result, companies and organizations all over the world have embraced this technology as a practical way to increase the effectiveness, speed, and accuracy of their operations.

The civilian drone market has seen a recent influx of new models, with many of these multi-rotors utilizing advanced and sophisticated technologies previously unexplored in the industry. The incorporation of high-precision technologies, particularly in the areas of tracking, recognition, and obstacle avoidance, has allowed for greater functionality and efficiency in drone operations. These cutting-edge technologies have opened up new possibilities for a wide range of applications, from aerial surveying and inspection to precision agriculture and emergency response. The introduction of these advanced features has further expanded the potential uses of civilian drones, making them an increasingly popular choice for various industries seeking innovative solutions. Quadrotors are highly maneuverable aerial vehicles that need complex modeling approaches for control and optimization. Since it allows for the introduction of nonlinear dynamics and external forces, the Euler-Lagrange technique is often employed for modeling quadrotors [1-2]. The Newton-Euler technique is also employed since it is based on Newton's principles of motion and gives a thorough knowledge of the quadrotor's motion [3-4]. The Hamiltonian technique is used to derive the quadrotor's equations of motion in an energy-efficient manner [5-6]. The state space technique may be used to model and control a system using linear equations [7]. The linearization method is frequently used to simulate the nonlinear dynamics of a quadrotor around an operational point [8]. Finally, multibody system approach serves as a crucial tool in accurately modeling the quadrotor's dynamic behavior and its intricate interactions with the environment, taking into consideration the complex movements of its various parts. This approach is applied in different software such as GAZEBO, Webots, SimMechanics, and ADAMS [9-10].

The present article aims to examine the stability of a novel prototype of a quadcopter by employing the Euler language approach and utilizing the Proportional-Integral-Derivative (PID) regulator on MATLAB (Figure 1). The study will go through the fundamental movements of the quadcopter, namely roll, pitch, yaw, and altitude, to obtain a comprehensive understanding of the dynamics and performance of the device. Overall, this study represents an important step towards improving the stability and performance of quadcopters, which have become increasingly important for various applications, including aerial photography, surveillance, and transportation. By gaining a deeper understanding of the quadcopter's dynamics and behavior, we can develop more effective control strategies and improve the safety and reliability of these devices.

The objective of this study is to investigate the stability of a new quadcopter prototype using the Euler language technique and the Proportional-Integral-Derivative (PID) regulator on MATLAB. The research will go through the fundamental movements of the quadcopter, including: roll, pitch, yaw, and altitude, in order to understande the quadrotors dynamics and performance. Overall, this work offers a significant step in improving the stability and performance of quadcopters, which have become more relevant for a variety of applications. However, Increasing the safety and dependability of these aircrafts can be achieved by better understanding the dynamics and behavior of quadcopters.

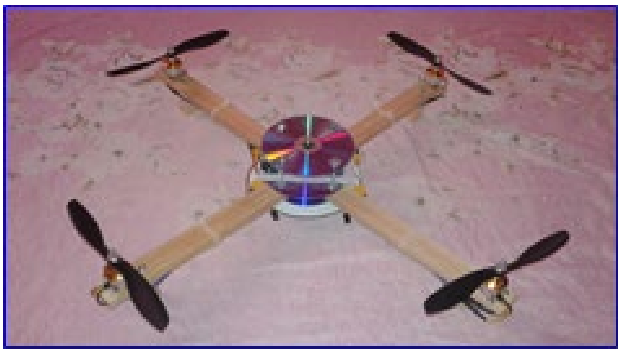


Figure 1. Quadrotor prototype.

2. QUADROTOR DYNAMICS AND REFERENCES

Quadcopters utilize rotor speed variation to execute fundamental movements. Tilting the quadcopter in the direction of a slower rotor results in translation along the corresponding axis, which is the basis of pitch and roll movements. Additionally, quadcopters are capable of vertical movement and rotation around the Z-axis, known as yaw, as depicted in Figure 2. These four movements, namely roll, pitch, vertical movement, and yaw, are controlled by the torque applied by the motors and are sufficient to manipulate the quadcopter's six degrees of freedom.

The quadcopters are governed by the laws of physics and aerodynamics. These unmanned aerial vehicles rely on the variation of rotor speeds to produce the basic movements required for their operation, namely roll, pitch, yaw, and altitude control. To describe the flight dynamics of quadcopters mathematically, two references are used: a fixed reference linked to the Earth and a mobile reference with its origin at the drone's center of gravity. The transformation matrix R is used to convert between these references and contains information about the orientation and position of the movable reference relative to the fixed reference. By modeling the quadcopter's flight dynamics, we can gain insight into its behavior and performance under different conditions, which can be used to improve its design and control strategies. The mathematical model can also be used to simulate the quadcopter's behavior and test various control algorithms and maneuvers in a virtual environment before applying them to real-world scenarios.

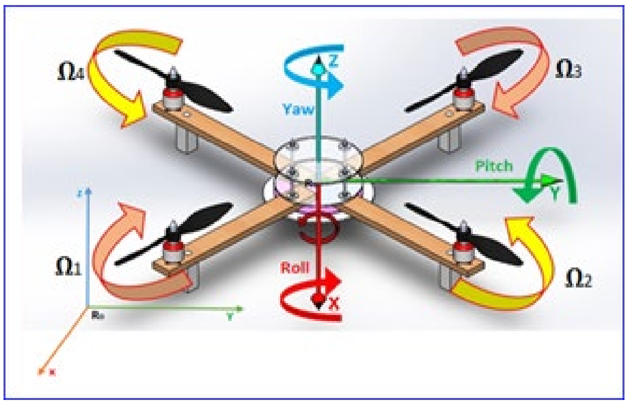


Figure 2. Quadrotor references and movements.

3. QUADROTOR EULER LAGRANGE MODEL

The rotation matrix, which describes the orientation of the quadcopter's movable reference frame relative to the fixed reference frame, can be obtained using Euler angles. Specifically, the rotation matrix is constructed by performing rotations around the X, Y, and Z axes, each by a respective angle of ϕ, θ, and ψ. These rotations correspond to the quadcopter's roll, pitch, and yaw movements, respectively, and are fundamental to controlling its position and orientation in three-dimensional space. The Euler angle approach is a powerful mathematical tool for simulating and analyzing dynamic systems, and its application to quadcopter flight dynamics enables us to study and optimize the performance of these devices. By understanding the relationship between the Euler angles and the quadcopter's movements, we can develop more effective control strategies and improve the safety and reliability of quadcopters in various applications.

Rotation matrix:
$R = R\ (\phi, \theta, \psi) =$

$$\begin{bmatrix} c\theta c\psi & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ c\theta s\psi & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \quad (1)$$

Lift is the force that allows the quadcopter to rise if it at least equals drag. It creates in the direction of the X and Y axes, the following two moments.

$$\tau_x = bl(\Omega_1^2 + \Omega_4^2 - \Omega_2^2 - \Omega_3^2) \quad (2)$$

$$\tau_y = bl(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2) \quad (3)$$

Thrust coefficient: to calculate the thrust coefficient we use this equation:

$$b = \frac{F_{moteur}}{\Omega_{moteur}^2} \quad (4)$$

The drag is the result of the friction of the air on the quadcopter, it is opposed to the lift. She creates a vertical moment.

$$\tau_z = d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \quad (5)$$

To calculate the drag coefficient of a quadcopter, it is necessary to fix the quadcopter at its center of gravity and apply rotation to two opposing motors to initiate rotation around the vertical axis. By measuring the complete cycle time t, the drag coefficient can be determined using the following equation.

$$d = \frac{\pi I_z}{2\Omega^2 t^2} \tag{6}$$

When the quadcopter is rotating on two axes, this rotation generates a force that appears on the third axis and tends to resist the movements of the quadcopter (gyroscope effect).

$$\tau_{gx} = I_{rotor}\omega_y(\Omega_1 + \Omega_4 - \Omega_2 - \Omega_3) \tag{7}$$
$$\tau_{gy} = I_{rotor}\omega_x(\Omega_1 + \Omega_2 - \Omega_3 - \Omega_4) \tag{8}$$

To generate the transfer equations for a motor, it can be represented as a simple RLC circuit. By neglecting losses, we can derive the following equation. This equation is important for understanding the motor's behavior and response to various inputs, which is essential for designing control systems that effectively regulate the quadcopter's movements.

$$H(s) = \frac{K}{K^2 + RJs} \tag{9}$$

To determine the moments of inertia of a quadcopter, we can treat it as a solid body with a fixed mass and its axes parallel to the main axes of inertia. This allows us to model it as a rectangular parallelepiped with mass M and dimensions L, W, and H. The motors can be modeled as cylinders with mass m, height h, radius R, and located at a distance of l from the center of gravity. By accurately calculating the moments of inertia, we can better understand the quadcopter's rotational behavior and design control algorithms that ensure stable and precise movements.

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{10}$$

The moments of inertia of a rectangular parallelepiped are modeled by the following equations:

$$I_X = \frac{m_{pr}}{12}(W^2 + H^2) + m_c\left(R^2 + \frac{h^2}{3}\right) + 2m_c l^2 \tag{11}$$
$$I_Y = \frac{m_{pr}}{12}(L^2 + H^2) + m_c\left(R^2 + \frac{h^2}{3}\right) + 2m_c l^2 \tag{12}$$
$$I_Z = \frac{m_{pr}}{12}(L^2 + W^2) + 2m_c R^2 + 2m_c l^2 \tag{13}$$

The Lagrange formula can be used to obtain the angular accelerations of a quadrotor by using the equations above. These equations take into account the forces acting on the quadrotor, such as thrust and drag, as well as the moments that cause it to rotate.

$$L = T - U \tag{14}$$

With:

$$T = \frac{1}{2} mV^2 \tag{15}$$
$$U = \int[-g\sin\theta]x\,dm + \int[g\sin\phi\cos\theta]y\,dm + \int[g\cos\phi\cos\theta]z\,dm \tag{16}$$

Angular accelerations:

$$\ddot{\phi} = \frac{I_{rotor}\dot{\theta}(\Omega_1+\Omega_4-\Omega_2-\Omega_3)}{I_x} + \frac{1}{I_x}U_2 + \frac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} \tag{17}$$
$$\ddot{\theta} = \frac{I_{rotor}\dot{\phi}(\Omega_1+\Omega_2-\Omega_3-\Omega_4)}{I_y} + \frac{1}{I_y}U_3 + \frac{I_z - I_x}{I_y}\dot{\psi}\dot{\phi} \tag{18}$$
$$\ddot{\psi} = \frac{1}{I_z}U_4 + \frac{I_x - I_y}{I_z}\dot{\theta}\dot{\phi} \tag{19}$$

Linear accelerations:

$$\ddot{x} = \frac{1}{m}(\cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi)U_1 \tag{20}$$
$$\ddot{y} = \frac{1}{m}(\cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi)U_1 \tag{21}$$
$$\ddot{z} = \frac{1}{m}(\cos\phi\cos\theta)U_1 - g \tag{22}$$

With:

$$U_1 = \sum_{i=1}^{4} b * T_i = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{23}$$
$$U_2 = b.l(\Omega_1^2 - \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \tag{24}$$
$$U_3 = b.l(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2) \tag{25}$$
$$U_4 = d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \tag{26}$$

Control algorithmes :

$$M_1 = T + R + P - Y \tag{27}$$
$$M_2 = T - R + P + Y \tag{28}$$
$$M_3 = T - R - P - Y \tag{29}$$
$$M_4 = T + R - P + Y \tag{30}$$

4. SIMULATION

To model a quadcopter using MATLAB (Simulink), it's important to understand its behavior and movements. With six degrees of freedom, but only four motors, we can control four of the six degrees, including altitude, roll, pitch, and yaw. Understanding the relationships between equations, including motor thrust, angular acceleration, gyroscopic effect, control, and displacement equations, is crucial in establishing a reliable model. Through the use of MATLAB (Simulink) in order to determine the constants of the PID and ensure stabilization of the quadcopter on all three axes of roll, pitch, yaw, and altitude. Figure 3 and 4 illustrates the quadrotor model in matlab simulink. Table 1 presents the quadrotor constates used for the simulation.

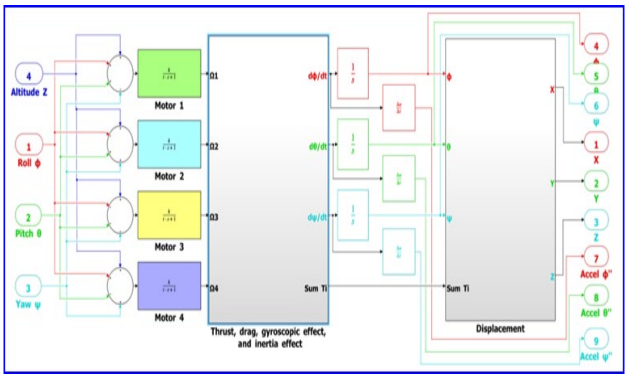


Figure 3. Quadrotor Simulink model

Table 1. Quadrotor parameters

Param	Value	Param	Value
Ix	$4.8 * 10^{-2}$ (Kg.m^2)	l	0.275 (m)
Iy	$4.8 * 10^{-2}$ (Kg.m^2)	d	$7.5 * 10^{-6}$ (Kg.m.rad^{-2})
Iz	$8.5 * 10^{-2}$ (Kg.m^2)	b	$3.1 * 10^{-6}$ (Kg.m.rad^{-2})
m	1.34(kg)	K	230.36 (rad/s)
I_{rotor}	$3.1 * 10^{-5}$ (Kg.m^2)	***t***	0.15 s

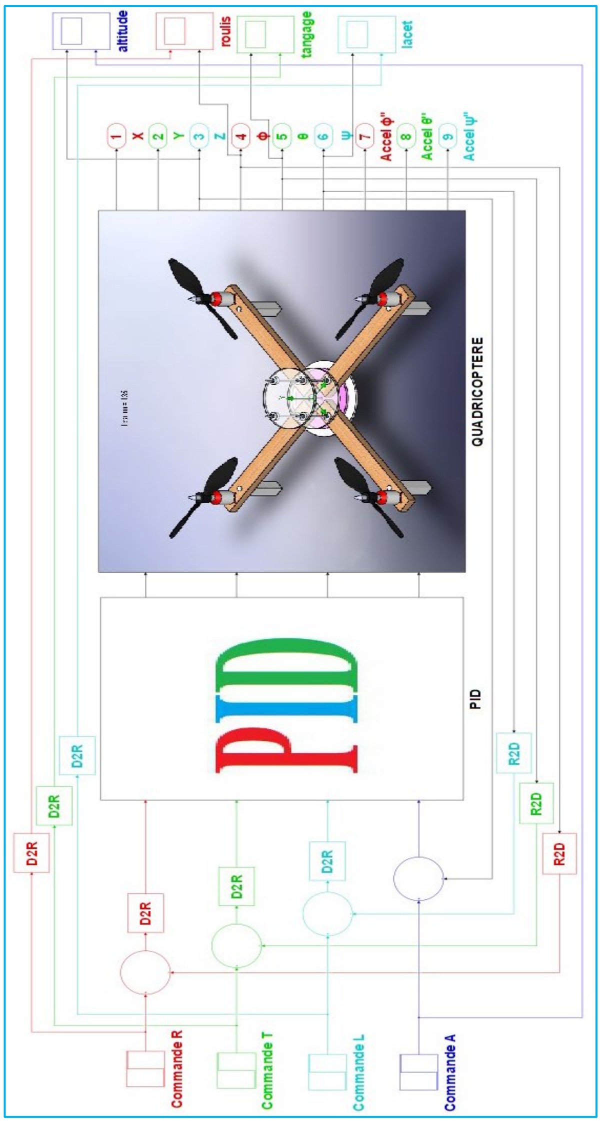


Figure 4. Quadrotor Simulink model with PIDs.

Manually adjusting the coefficients of the PID is a challenging task as it requires adjusting three coefficients at the same time with numerous possible combinations. The process starts with adjusting the Kp coefficient to improve the response time of the system, followed by adjusting the Ki coefficient to eliminate errors and ensure a quick and accurate response. Lastly, the Kd coefficient is adjusted to increase system stability by minimizing oscillations. The optimal values for each PID, including roll, pitch, yaw, and altitude, are provided in Table 2.

Table 2. PID parameters.

Roll		Pitch		Yaw		Altitude	
Param	Value	Param	Value	Param	Value	Param	Value
Kp	1	Kp	1	Kp	0.8	Kp	3
Ki	0.01	Ki	0.01	Ki	0.01	Ki	0.85
Kd	1.2	Kd	1	Kd	1.1	Kd	3

5. RESULTS AND DISCUSSION

Following the simulation of the quadrotor using MATLAB Simulink, we were able to obtain valuable results that depict the system's stability across all four movements, including roll, pitch, yaw, and altitude. In addition, the quadrotor's movements along the X and Y axes, based on its roll and pitch movements, were also evaluated. The results presented in Figures (5, 6, 7 and 8), offer comprehensive visual representations of the quadrotor's dynamic behavior and provide significant insights for the purpose of further analysis and control optimization.

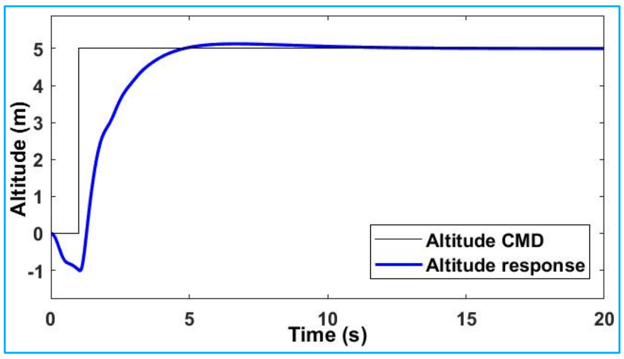


Figure 5. Altitude results.

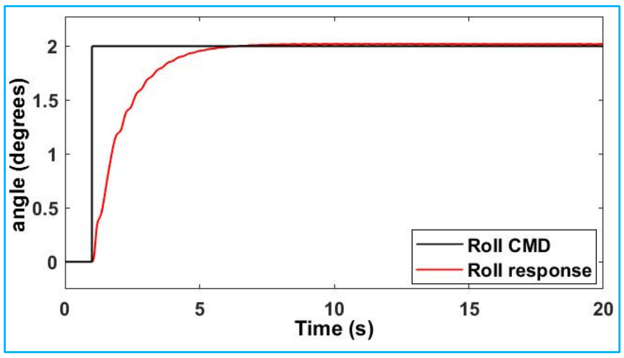


Figure 6. Roll results.

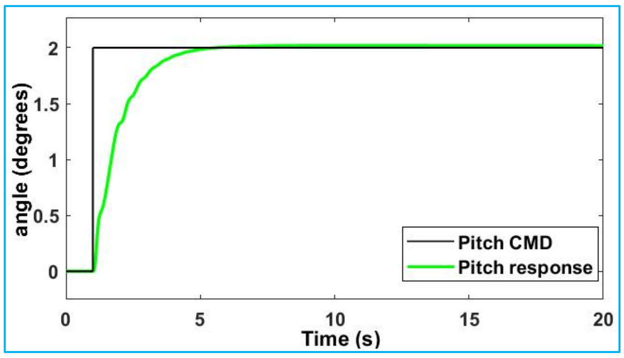


Figure 7. Pitch results.

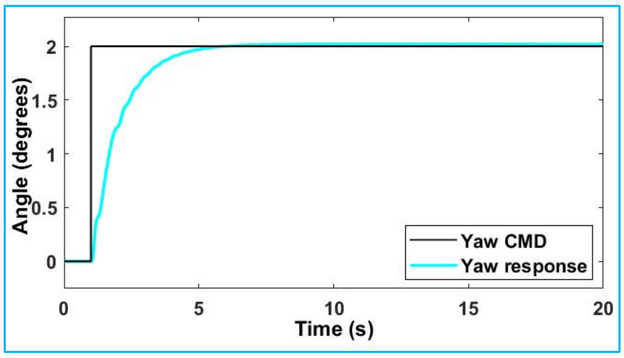


Figure 8. Yaw results.

6. CONCLUSION

In this study, The quadrotor model was simulated using the Euler-Lagrange technique in MATLAB Simulink, and its stability was tested using a Proportional-Integral-Derivative (PID) controller. The simulation results confirmed the PID controller's usefulness in controlling the quadrotor's motion and preserving its stability throughout flight. The study also emphasizes the significance of the Euler-Lagrange technique in adequately describing the complicated dynamics of the quadrotor. Overall, this work gives useful insights for the development and improvement of quadrotor control systems and emphasizes the need of using modern simulation tools like MATLAB to explore the dynamics of complex systems.

7. NOMENCLATURE

ENGLISH LETTERS

I_x: The moment of inertia along the X axis (Kg.m²).
I_y: The moment of inertia along the Y axis in (Kg.m²).
I_z: The moment of inertia along the Z axis in (Kg.m²).
I_{rotor}: The moment of inertia around the motor in (Kg.m²).
d: The drag coefficient (kg. m. rad^{-2}).
b: The thrust coefficient (kg. m. rad^{-2}).
l: The distance between the motor and the centre of gravity of the quadcopter (m).
Kp, Ki, Kd: The gains of proportional, integrals, derivatives.
b: The thrust coefficient in (kg.m/rad^2).
m_{pr}: The weight of the rectangular parallelepiped (kg).
m_c: The mass of the cylinder (kg).
W: The width of the rectangular parallelepiped (m).
h: The height of the cylinder (m).
H: The height of the rectangular parallelepiped (m).
K: The gain of the motor in (V.s / rad).
R: The internal resistance of the motor in (ohm).
J: The inertia of the rotor in (g.cm²).
Mi: Motors
T: Thrust
R: Roll
P: Pitch
Y: Yaw
C: cos
S: sin

GREEK SYMBOLS

ϕ: The angle of rotation around the 'X' axis (Roll) in (rad).
$\boldsymbol{\theta}$: The angle of rotation around the 'Y' axis (Pitch) in (rad).
$\boldsymbol{\Psi}$: The angle of rotation around the 'Z' (Yaw) axis in (rad).
$\boldsymbol{\Omega_i}$: The otors speed in (rad / s).
τ: The time constant of the motors (s).

8. REFERENCES

[1] S. Wang, A. Polyakov, and G. Zheng, "Quadrotor stabilization under time and space constraints using implicit PID controller," *Journal of the Franklin Institute*, vol. 359, no. 4, pp. 1505–1530, Mar. 2022, doi: 10.1016/j.jfranklin.2022.01.002.

[2] J. E. Lavín-Delgado, Z. Zamudio Beltrán, J. F. Gómez-Aguilar, and E. Pérez-Careta, "Controlling a quadrotor UAV by means of a fractional nested saturation control," *Advances in Space Research*, p. S0273117722009619, Oct. 2022, doi: 10.1016/j.asr.2022.10.023.

[3] C. Sun, M. Liu, C. Liu, X. Feng, and H. Wu, "An Industrial Quadrotor UAV Control Method Based on Fuzzy Adaptive Linear Active Disturbance Rejection Control," *Electronics*, vol. 10, no. 4, p. 376, Feb. 2021, doi: 10.3390/electronics10040376.

[4] H. Heidari and M. Saska, "Trajectory Planning of Quadrotor Systems for Various Objective Functions," *Robotica*, vol. 39, no. 1, pp. 137–152, Jan. 2021, doi: 10.1017/S0263574720000247.

[5] M. Zare, F. Pazooki, and S. Etemadi Haghighi, "Hybrid controller of Lyapunov-based and nonlinear fuzzy-sliding mode for a quadrotor slung load system," *Engineering Science and Technology, an International Journal*, vol. 29, p. 101038, May 2022, doi: 10.1016/j.jestch.2021.07.001.

[6] T. Duong and N. Atanasov, "Adaptive Control of SE(3) Hamiltonian Dynamics with Learned Disturbance Features." arXiv, Mar. 22, 2022. Accessed: Jan. 24, 2023. [Online]. Available: http://arxiv.org/abs/2109.09974

[7] O. Kose and T. Oktay, "Combined Quadrotor Autopilot System and Differential Morphing System Design," *Journal of Aviation*, Dec. 2021, doi: 10.30518/jav.856436.

[8] L. Martins, C. Cardeira, and P. Oliveira, "Feedback Linearization with Zero Dynamics Stabilization for Quadrotor Control," *J Intell Robot Syst*, vol. 101, no. 1, p. 7, Jan. 2021, doi: 10.1007/s10846-020-01265-2.

[9] S. De and D. Guida, "Control design for an under-actuated UAV model," *FME Transactions*, vol. 46, no. 4, pp. 443–452, 2018, doi: 10.5937/fmet1804443D.

[10] T. Avant, U. Lee, B. Katona, and K. Morgansen, "Dynamics, Hover Configurations, and Rotor Failure Restabilization of a Morphing Quadrotor," in *2018 Annual American Control Conference (ACC)*, Milwaukee, WI: IEEE, Jun. 2018, pp. 4855–4862. doi: 10.23919/ACC.2018.8431628.

DOI: 10.2478/sbeef-2023-0002

LABVIEW AND REMOTEXY INTEGRATION FOR QUADROTOR STABILIZATION AND CONTROL

[1,2]HAMZA DJIZI, [3]ZOUBIR ZAHZOUH, [4]AZZEDINE BOUZAOUIT
[1]Department of Mechanical Engineering, Mohamed Cherif Messaadia University, P.O. Box 1553, Souk-Ahras, 41000, Algeria.
[2]INFRA-RES Laboratory, Mohamed Cherif Messaadia University, Algeria
[3]Laboratoire de Recherche en Électromécanique et Sûreté de Fonctionnement, LRESF Laboratory, Mohamed Chérif Messaadia University, P.O. Box 1553, Souk-Ahras, 41000, Algeria.
[4]University of 20 August 1955 Skikda, Algeria
E-mail: hamzadjizi@gmail.com

Abstract. ***Nowadays, Small quadcopters have made significant advancements in recent years, thanks to the development of control systems, the availability of sensors, and affordable and reliable materials for their production. Additionally, programs have been developed to model and analyze these aircraft before production. The professional applications of quadcopters are seemingly endless due to their many advantages. The aim of this research is to build a quadcopter and test its stability utilizing Arduino Mega, IMU sensor (Inertial Measurement Unit) and MPU-6050 in LabVIEW environment. The objective is to select the suitable PID parameters and create a remote-control program that can be operated using a smartphone and RemoteXY app on Android OS.***

Keywords: quadcopter, control, LabVIEW, Arduino, sensor.

1. INTRODUCTION

Quadcopters with sensors have developed in terms of their ability to perceive their surroundings, allowing them to fly in appropriate conditions and maintain balance. They consist of four motors with propellers attached to a wooden or other material cross, with each motor connected to an electronic speed controller (ESC) controlled by a control card. The control card receives its commands from a radio control receiver or a smartphone. Designing and building a quadcopter is less complicated than a standard helicopter, but still requires careful consideration of the parts and assembly during the design process. When editing, it is crucial to consider some general points based on experience.

One of the most challenging mounting locations when designing a custom frame is where the motors and frame meet since the four mounting holes there need to be set precisely. Any extra parts should preferably be arranged symmetrically around an axis to make it easy to determine the quadcopter's centre of gravity. The middle of the circle that connects all motors should ideally house the controller (Arduino Mega). As it is heavy enough for the quadcopter, the battery should also be placed in the middle of the device.

Quadrotor control has been an area of interest for many researchers in recent years due to its potential for applications in a wide range of fields such as aerial photography, environmental monitoring, and search-and-rescue operations [1]. PID (Proportional-Integral-Derivative) controllers have been widely used in quadrotor control due to their simplicity and effectiveness in providing stable flight control [2-3]. Previous studies have shown that PID controllers can achieve accurate and stable flight control for quadrotors under various conditions, including disturbances and changing environments [4]. However, researchers have been proposed several regulators for quadrotor control, both linear and nonlinear. Linear regulators such as PD (Proportional-Derivative) and LQR (Linear Quadratic Regulator) have been widely used due to their simplicity and effectiveness in achieving stable flight control [5-6]. Nonlinear regulators such as MPC (Model Predictive Control) [7], SMC (Sliding Mode Control)[8], and fuzzy logic controllers have also shown promising results in achieving stable and robust control of quadrotors under various conditions [9]. Moreover, the use of neural networks has been investigated in quadrotor control as they have the ability to learn complex nonlinear relationships between the inputs and outputs of the system [10].

In this work, we will try to study the stability of a quadrotor using LabVIEW. To achieave that, we need a range of components, including a wooden frame for constructing the quadcopter, an Arduino Mega 2560 microcontroller, four A2212-6T 2200KV motors, four 30A electronic speed controllers (ESC), four 8045 propellers, an MPU-6050 gyroscope, ESP8266 Wi-Fi module, wires, a 3S 4000mAh Li-Po battery, an IMAX B6 Li-Po balance charger (Figure 1). In addition to LabVIEW and an Android OS smartphone equipped with the RemoteXY app. However, Section 2 provides a detailed guide on how to construct a quadrotor and link its components together. Section 3 delves into the control equations and how they can be used to control the quadrotor's movement. Section 4 explains how to implement the control on a LabVIEW environment, while Section 5 discusses the use of RemoteXY to control the quadrotor. Section 6 presents the results and discusses their implications, while the final section, the conclusion, summarizes the article's key points and emphasizes the significance of the findings.

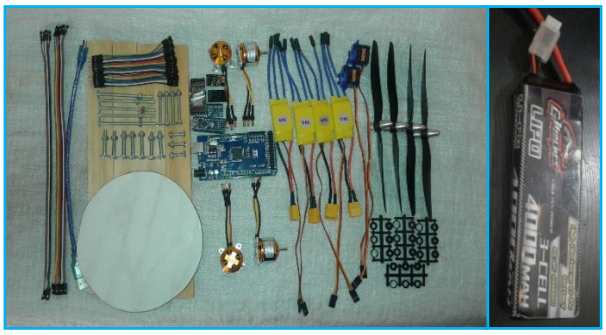


Figure 1. The quadcopter and its basic components.

2. CONSTRUCT THE QUADCOPTER AND LINK ITS COMPONENTS

The quadcopter will be developed in steps, starting with the mechanical assembly and continuing with the functional testing of the parts. The quadcopter's whole structure will be created by connecting the electrical components using connection cables (Jumper-Wire), then assembling them onto the frame. Also, to ensure that the quadcopter can handle the intended weight or achieve the desired thrust, The motors will be selected based on their capabilities to provide the necessary thrust for the quadcopter's weight and flight performance.

Developing an effective command interface in LabVIEW or an interface in Android using the RemoteXY website, it is imperative to gain a comprehensive understanding of the operational and connection mechanisms of the various components of the quadcopter, including but not limited to Arduino, motors, gyroscope, Wi-Fi, and ESC. Only by thoroughly comprehending the functioning of these components and their interplay can we design and implement a robust interface that can cater to the specific needs of the quadcopter and enable efficient control and maneuverability.

Connecting the fundamental electrical components will be established according to the diagram presented in Figure 2, which illustrates the optimal way to link the Arduino Mega, the four A2212-6T 2200KV motors, the MPU-6050 gyroscope, the ESP8266 Wi-Fi module, the four 30A electronic speed controllers (ESC), the Li-Pro battery (3S, 4000 MAH), and the necessary Jumper-Wire cables to ensure the proper functioning of the quadcopter.

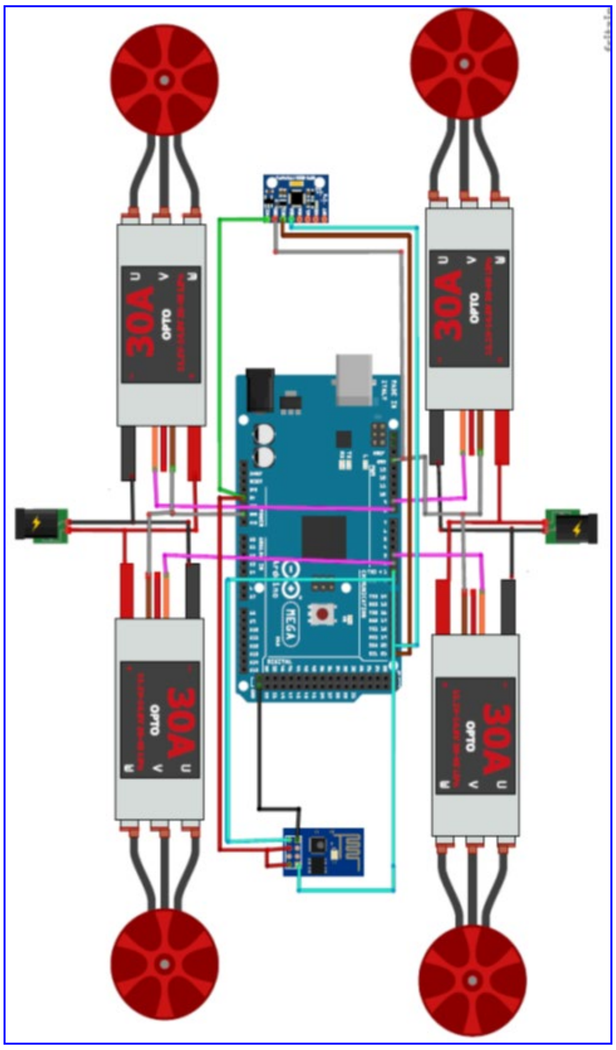


Figure 2. Linking the electronic components.

After assembling the quadrotor, it's essential to carry out a series of tests on the various components to ensure everything is working correctly (Figure 3). This includes establishing a stable WiFi connection using the esp8266 module, initializing the ESC to optimize the performance of the brushless motors, and calibrating the gyroscope to remove any offsets in the six axes. These steps are critical for ensuring a smooth and stable flight, and they must be performed carefully and thoroughly to minimize the risk of any malfunctions or accidents during operation. Once all of these tasks are completed successfully, the quadrotor should be ready for programming and testing and further optimization.

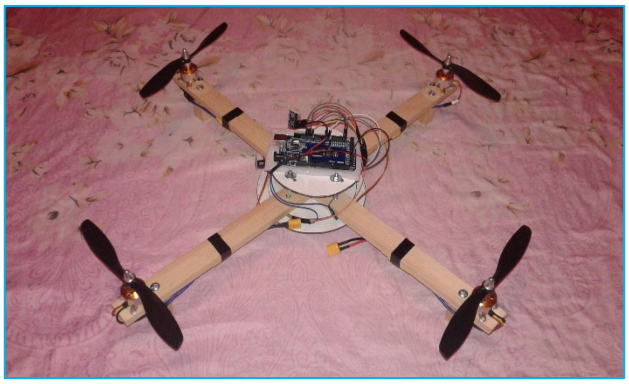


Figure 3. The quadcopter model.

3. CONTROL AND EQUATIONS

Accelerometers and gyroscopes are two commonly used sensors in quadrotors to achieve stabilization and control. The accelerometer measures the linear acceleration of the quadrotor in all three axes. Based on this information, the control system can calculate the orientation of the quadrotor and adjust the motor thrust accordingly to maintain stability. Gyroscopes, on the other hand, measure the angular velocity of the quadrotor around all three axes. By integrating the gyroscopic data, the system can calculate the orientation of the quadrotor over time. Combining the accelerometer and gyroscope data, the control system can achieve accurate attitude estimation of the quadrotor and make necessary adjustments to the motor thrust to stabilize and control the device. The integration of these two sensors provides a robust and reliable control system for quadrotors, enabling them to perform complex maneuvers and maintain stability in various flight conditions.

To comprehensively assess the dynamics and stability of a quadcopter, determining the angles of both accelerometer and gyroscope is essential. To accurately calculate the angles of the gyroscope, employing the equations below is crucial.

$$AngleX = \operatorname{atan}\left(\frac{Y}{\sqrt{X^2+Z^2}}\right) \quad (1)$$

$$AngleY = \operatorname{atan}\left(\frac{X}{\sqrt{Y^2+Z^2}}\right) \quad (2)$$

Accelerometer angles:

$$acc\_angle\_x = atan((aY/16384.0)/sqrt(pow((aX/16384.0),2) + pow((aZ/16384.0),2))) * rad\_to\_deg; \quad (3)$$

$$acc\_angle\_y = atan(-1 * (aX/16384.0)/sqrt(pow((aY/16384.0),2)\ pow((aZ/16384.0),2))) * rad\_to\_deg; \quad (4)$$

$$acc\_angle\_z = atan(sqrt(pow((aY/16384.0),2) + pow((aX/16384.0),2))/(aZ16384.0)) * rad\_to\_deg; \quad (5)$$

Gyroscope Angles:

$$Gyro\_angle\_x = gX/131.0; \quad (6)$$

$$Gyro\_angle\_y = gY/131.0; \quad (7)$$

$$Gyro\_angle\_z = gY/131.0; \quad (8)$$

Total Angles:

$$totale\_angle\_x = 0.98 * (total\_angle\_x + Gyro\_angle\_x * elapsedtime) + 0.02 * acc\_angle\_x; \quad (9)$$

$$totale\_angle\_y = 0.98 * (total\_angle\_y + Gyro\_angle\_y * elapsedtime) + 0.02 * acc\_angle\_y; \quad (10)$$

$$totale\_angle\_z = totale\_angle\_z + Gyro\_angle\_z * ellapsedtime \quad (11)$$

Error calculation:

$$error\_x = totale\_angle\_x - desired\_angle\_x \quad (12)$$

$$error\_y = totale\_angle\_y - desired\_angle\_y \quad (13)$$

$$error\_z = totale\_angle\_z - desired\_angle\_z \quad (14)$$

PID equations:

$$pid\_p = kp * erreur; \quad (15)$$

$$pid\_d = kd * ((erreur - erreur\_précédent)/temps\_écoulé); \quad (16)$$

$$PID = pid\_p + pid\_i + pid\_d; \quad (17)$$

With:
aX, aY, aZ: accelerometer variables (axes)
gZ, gY, gZ: gyroscope variables (axes)
kp, ki, kd: PID parameters

4. CONTROL ON LABVIEW ENVIRONMENT

The implementation of a command interface utilizing LabVIEW has facilitated the study of the quadcopter's distinct movements in real-life scenarios. The simulation results derived from this analysis have enabled us to accurately maintain the speed and stability of the quadcopter, thus ensuring optimal flight conditions. The successful integration of these findings led to the development of an ideal interface under the Android platform, allowing for remote control of the quadcopter with unparalleled precision and reliability.

The implementation of the LabVIEW interface was executed through a systematic construction of the block diagram, which was employed to configure all essential functions required for the comprehensive control of the quadcopter's various components, including the Arduino, motors, and MPU6050 (Figure 4). Beginning with the development of the PIDs diagram, algorithm mixer diagram, and followed by the motors and MPU6050 diagrams, the block diagram that based on the equations above enables the creation of a seamless control interface, which represented in Figure 5.

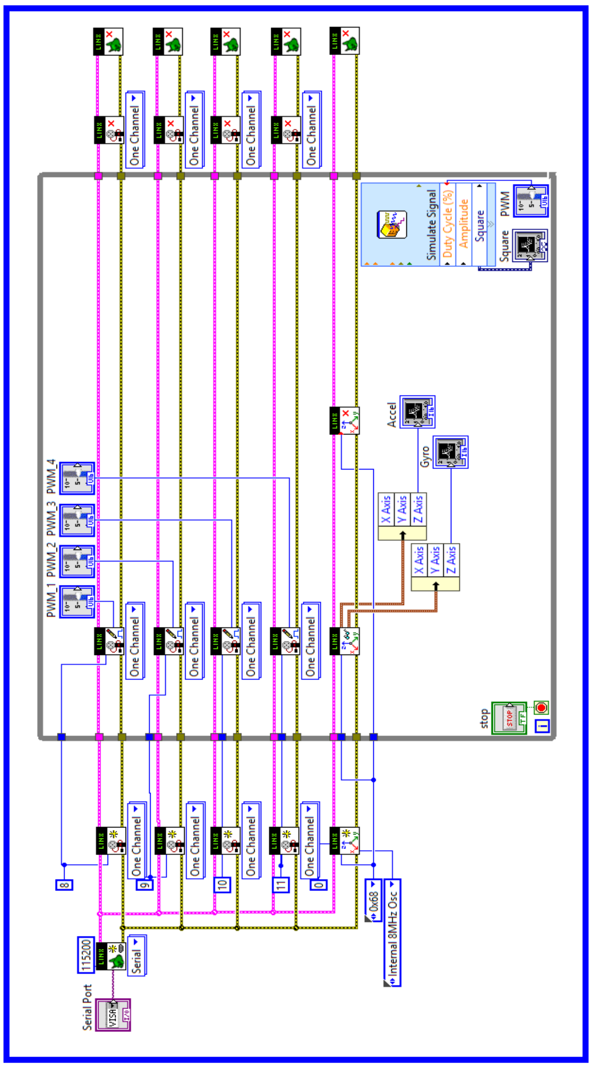


Figure 4. Brushless Motors and gyroscope control diagram (MPU-6050).

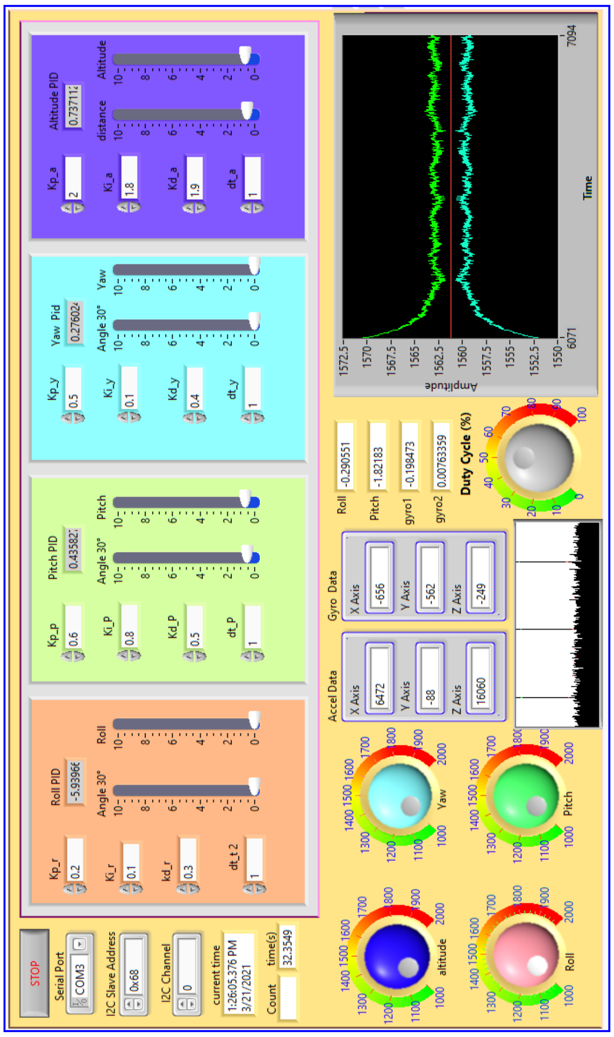


Figure 5. The Control graphical interface on LabVIEW.

5. CONTROL USING ROMOTXY

In order to program and establish a connection between the Arduino and the website (remotexy.com) and its corresponding phone application (Figure 6). A Wi-Fi connection was selected along with an Arduino Mega device, Wifi-ESP8266 connection module, and the Arduino IDE development environment. To enable communication between the smartphone and the controller (Arduino Mega), it was necessary to connect it to the Wi-Fi-ESP8266 module via the UART inputs (Rx, Tx). The ESP8266 module was then configured as a standalone Wi-Fi access point, requiring no connection with an existing Wi-Fi network for operation. The smartphone must be connected to the access point created to establish a connection. When programming the controller, the equations and variables mentioned about will be utilized to calculate the PID values and the appropriate pulse width modulation (PWM) to enable precise control over each movement.

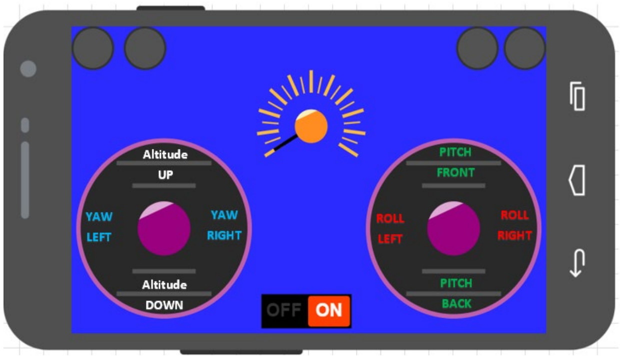


Figure 6. RemoteXY Command interface on smartphone.

6. RESULTS AND DISCUSSION

The analysis of the quadcopter and its real-time movements, while similar to its simulation counterpart under MATLAB, presents variations due to the potential for measurement inaccuracies and manufacturing discrepancies. As such, the current study employed the LabVIEW environment to investigate the movement of the quadcopter and derive precise values for the essential PIDs parameters, which are crucial for subsequent programming of the controller (Table 1).

Table 1. PID regulator parameters

PID_Roll		PID_Pitch		PID_Yaw		PID_Altitude	
Param	Value	Param	Value	Param	Value	Param	Value
Kp	0.2	Kp	0.6	Kp	0.5	Kp	2
Ki	0.1	Ki	0.8	Ki	0.1	Ki	1.8
Kd	0.3	Kd	0.5	Kd	0.4	Kd	1.5

The simulation results for the quadrotor system using the PID regulator on LabVIEW were highly successful. The altitude results demonstrated consistent and stable flight at the desired height, with minimal oscillations and deviations from the setpoint (Figure 7). The roll and pitch results were also impressive, with the quadrotor maintaining a level and balanced flight despite varying wind conditions and disturbances (Figure 8 and 9). The yaw results were equally noteworthy, showcasing the system's ability to maintain a stable heading and respond quickly to changes in orientation (Figure 10). Overall, the simulation demonstrated the effectiveness of the PID regulator in controlling the quadrotor's movements and ensuring smooth, stable flight performance. These results provide valuable insights for further development and optimization of quadrotor systems for a variety of applications

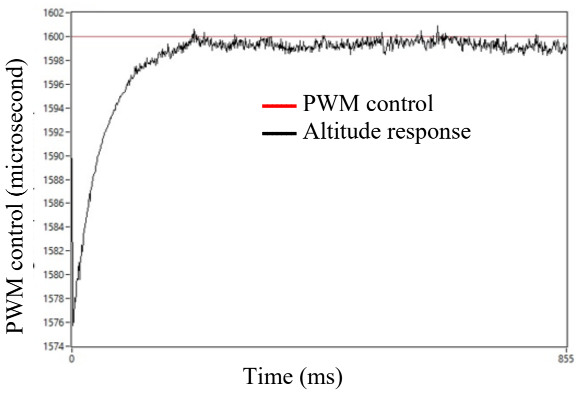


Figure 7. System response for Altitude movement

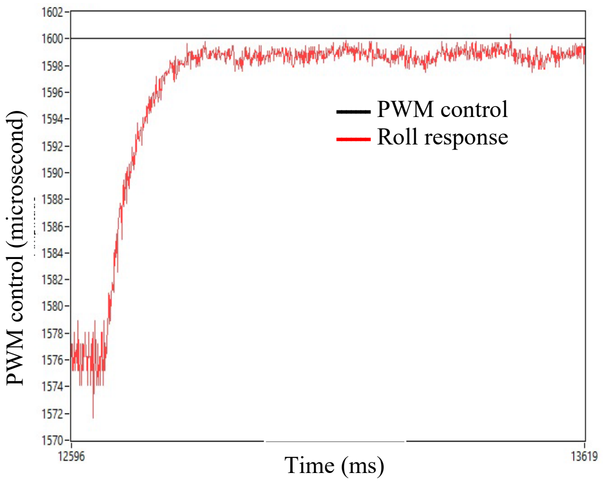


Figure 8. System response for Roll movement

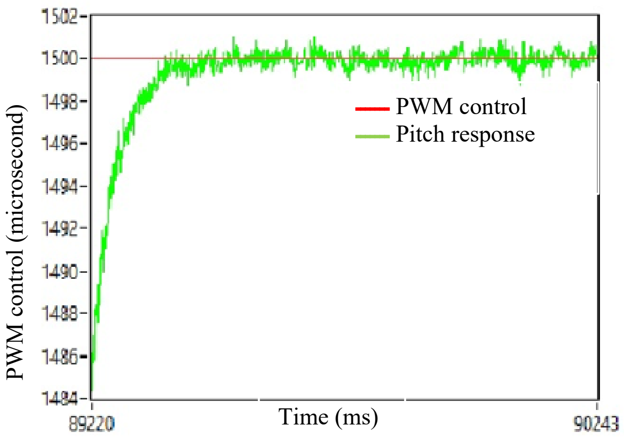


Figure 9. System response for Pitch movement

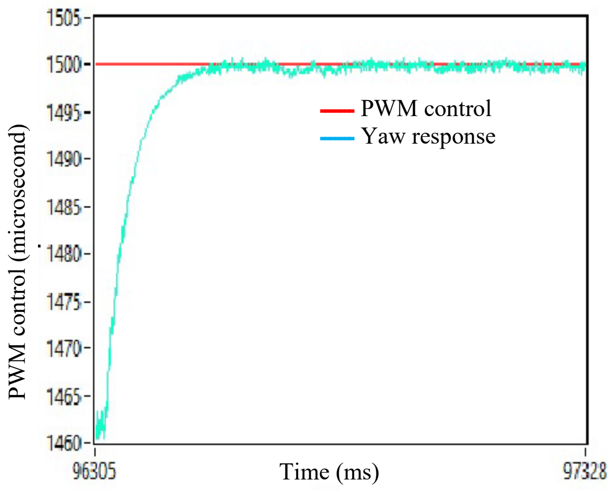


Figure 10. System response for Yaw movement

7. CONCLUSION

This study has successfully presented a system for controlling the various movements of the quadcopter and analyzed the stability of each motion using a PID regulator in the LabVIEW environment. The experimental results demonstrated that the stability of each movement is closely correlated with the speed of the motors. The practical study conducted on LabVIEW enabled us to establish appropriate PID parameters for subsequent controller programming. The IMU sensor (Inertial Measurement Unit, MPU-6050) was identified as a key component for quadcopter controller programming since it can accurately measure all the basic movements of the quadcopter. By utilizing these measurements, we can effectively control and stabilize the quadcopter with remarkable efficiency using Smartphone (RemotXY).

8. REFERENCES

[1] D. Asadi, "Partial Engine Fault Detection and Control of a Quadrotor Considering Model Uncertainty," *Turkish Journal of Engineering*, Mar. 2021, doi: 10.31127/tuje.843607.

[2] L. Jin, Y. Lou, L.-A. Chen, and Q. Lu, "The Unified Tracking Controller for a Tilt-Rotor Unmanned Aerial Vehicle Based on the Dual Quaternion," in *2022 IEEE International Conference on Unmanned Systems (ICUS)*, Oct. 2022, pp. 1356–1363. doi: 10.1109/ICUS55513.2022.9986880.

[3] X. Lu and Z. Xing, "Application of IoT Quadrotor Dynamics Simulation," *Electronics*, vol. 11, no. 4, p. 590, Feb. 2022, doi: 10.3390/electronics11040590.

[4] B. Jiang, B. Li, W. Zhou, L.-Y. Lo, C.-K. Chen, and C.-Y. Wen, "Neural Network Based Model Predictive Control for a Quadrotor UAV," *Aerospace*, vol. 9, no. 8, p. 460, Aug. 2022, doi: 10.3390/aerospace9080460.

[5] S. Martini, S. Sonmez, A. Rizzo, M. Stefanovic, M. J. Rutherford, and K. P. Valavanis, "Euler-Lagrange Modeling and Control of Quadrotor UAV with Aerodynamic Compensation," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, Dubrovnik, Croatia: IEEE, Jun. 2022, pp. 369–377. doi: 10.1109/ICUAS54217.2022.9836215.

[6] L. Martins, C. Cardeira, and P. Oliveira, "Feedback Linearization with Zero Dynamics Stabilization for Quadrotor Control," *J Intell Robot Syst*, vol. 101, no. 1, p. 7, Jan. 2021, doi: 10.1007/s10846-020-01265-2.

[7] D. Yan, W. Zhang, and H. Chen, "Design of a Multi-Constraint Formation Controller Based on Improved MPC and Consensus for Quadrotors," *Aerospace*, vol. 9, no. 2, p. 94, Feb. 2022, doi: 10.3390/aerospace9020094.

[8] F. Ma, Z. Yang, and P. Ji, "Sliding Mode Controller Based on the Extended State Observer for Plant-Protection Quadrotor Unmanned Aerial Vehicles," *Mathematics*, vol. 10, no. 8, p. 1346, Apr. 2022, doi: 10.3390/math10081346.

[9] K. Liu, R. Wang, S. Dong, and X. Wang, "Adaptive Fuzzy Finite-time Attitude Controller Design for Quadrotor UAV with External Disturbances and Uncertain Dynamics," in *2022 8th International Conference on Control, Automation and Robotics (ICCAR)*, Xiamen, China: IEEE, Apr. 2022, pp. 363–368. doi: 10.1109/ICCAR55106.2022.9782598.

[10] J. Li, S. Mou, and D. Zhang, "A Novel Adaptive Robust Control Algorithm for Quadrotor UAV," in *2021 6th International Conference on Robotics and Automation Engineering (ICRAE)*, Guangzhou, China: IEEE, Nov. 2021, pp. 50–54. doi: 10.1109/ICRAE53653.2021.9657806.