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.

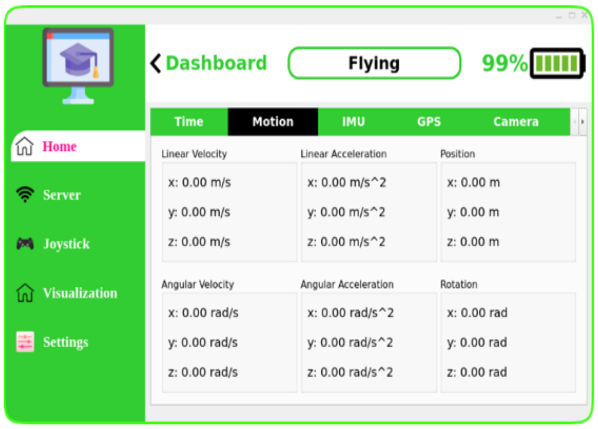
This paper is structured as follows. In section 2, the design aspects of the PyQt6 application are discussed. It covers the architecture, features, and the application, highlighting the development choices and considerations. Section 3 presents the URDF prototype of the quadrotor. It describes the design 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

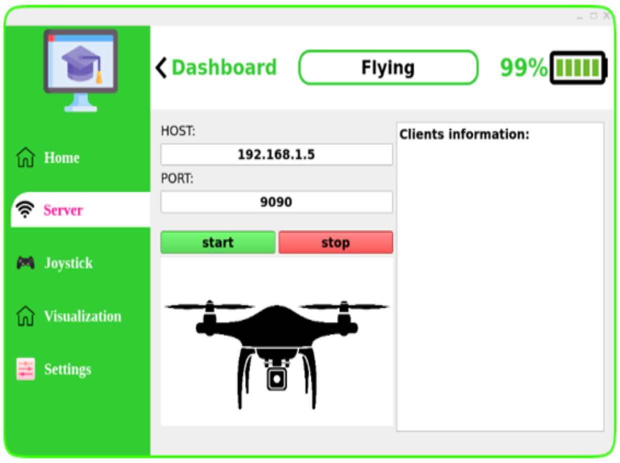
Figure 1. Home screen for real-time monitoring and control of quadrotor system

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

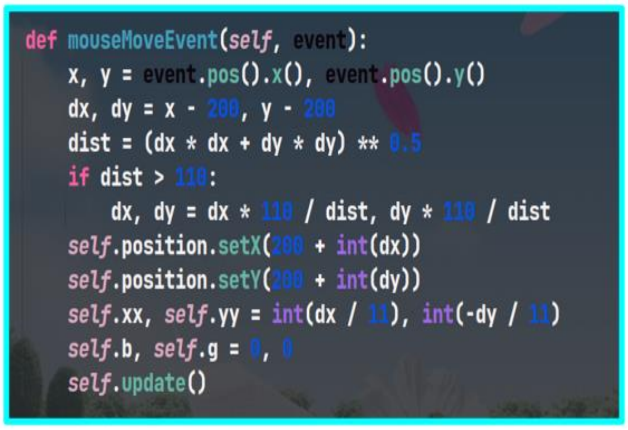
Clients information:

Figure 2. Server screen to start and stop the server

2.3 Joystick screen design

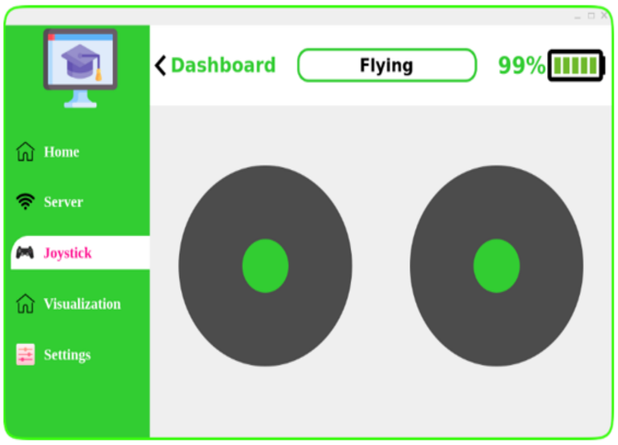
To create a new PyQt6 application featuring two joysticks, the essential first step is to import the necessary libraries, including PyQt6 and math. Two joysticks can then be created using QPainter and paintEvent, with the drawEllipse method to draw the circles required for the joysticks. Then, the two 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.

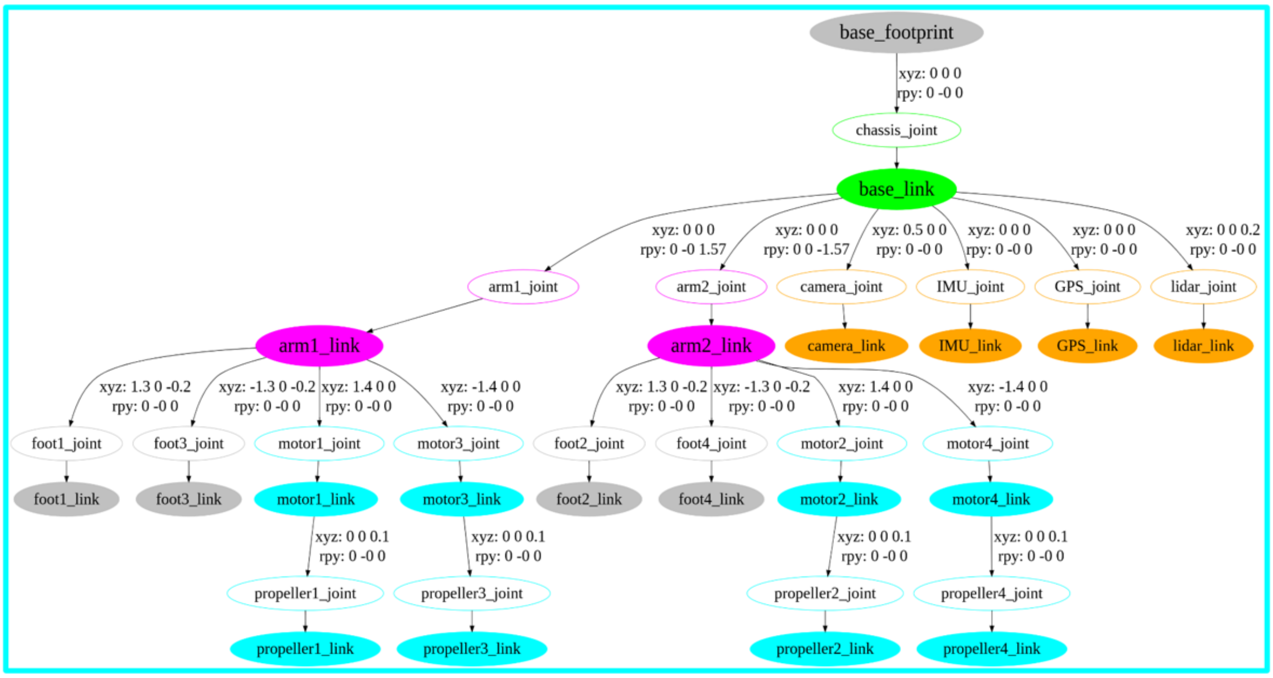


Figure 4. Graphical representation of the hierarchical quadrotor system including the sensors

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.

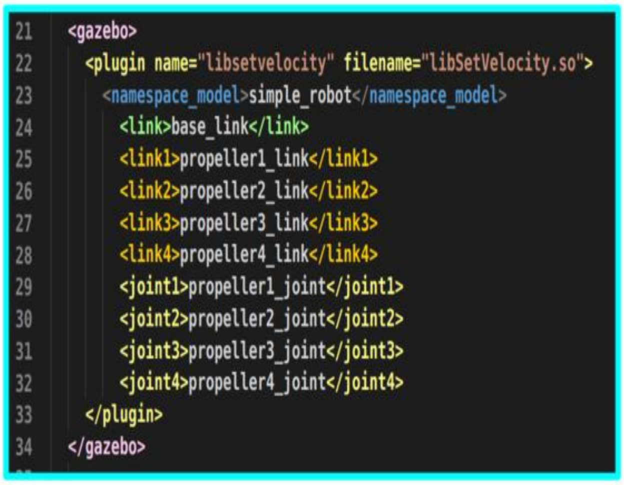
The plugin is based on the fundamental principles of physics, specifically Newton's second law of motion, which is instrumental in governing the dynamics of the quadrotor multibody system. In addition, different flight conditions are considered, such as hovering, ascending, descending, taking off, and 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

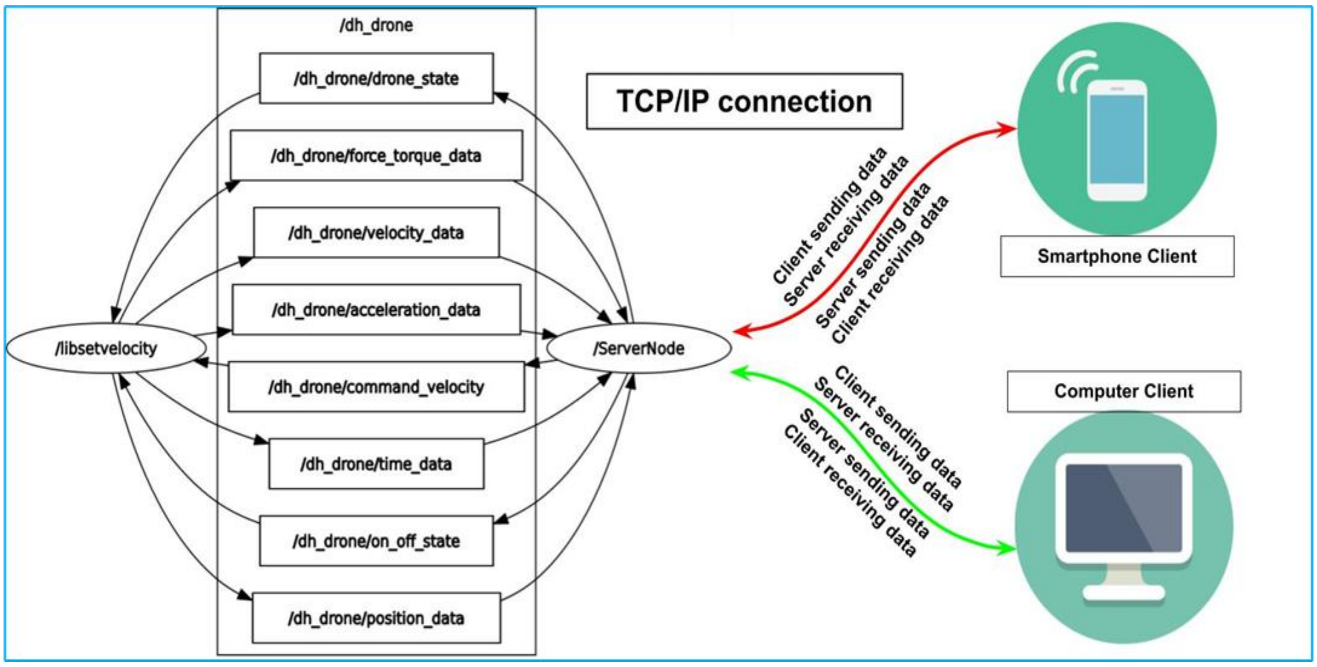


Figure 6. A graphical representation of the ROS2 network and communication model, demonstrating server-client communication and TCP/IP protocols with ROS2 nodes

5. IMPLEMENTATION AND COMMUNICATION

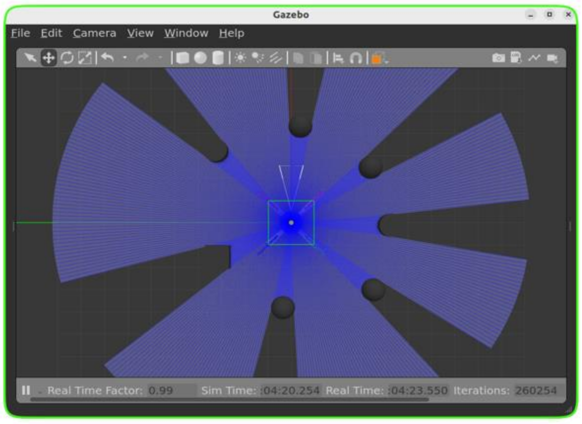
The process of constructing the system involves creating three packages within the source directory of the workspace: quadrotor_pkg, msgs_pkg, and gazebo_plugin_pkg. Each of the packages encompasses different dependences necessary for the system's functionality. The quadrotor_pkg contains the essential 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.

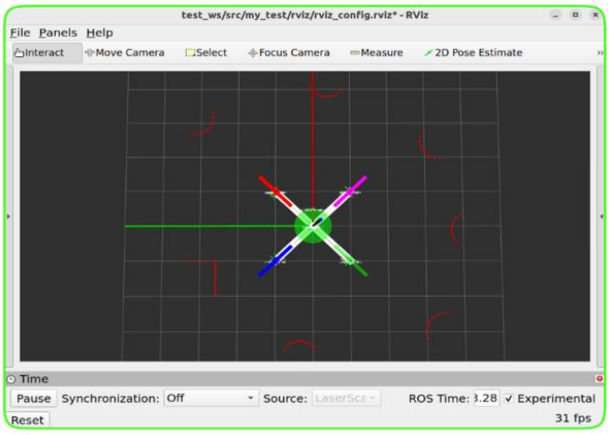
The communication between ROS2 nodes in the quadrotor system is essential. The server node, responsible for receiving clients input and sending commands to the quadrotor node, communicates with the quadrotor node via a series of topic subscriptions and publishers (Figure 6). The quadrotor node provides 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.



(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.

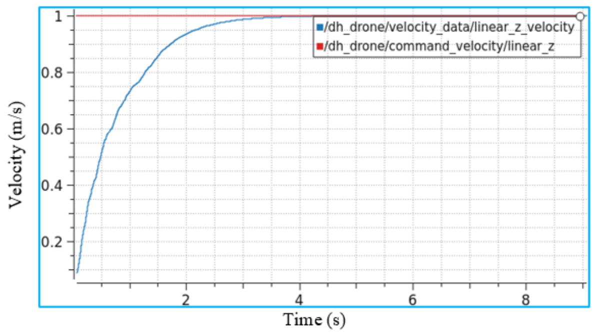
The simulation results were highly encouraging, with the system effectively detecting and calculating distances to obstacles. These findings underscore the potential for developing an advanced obstacle avoidance algorithm, enabling the quadrotor to navigate complex environments while prioritizing safety.

6. RESULTS AND DISCUSSION

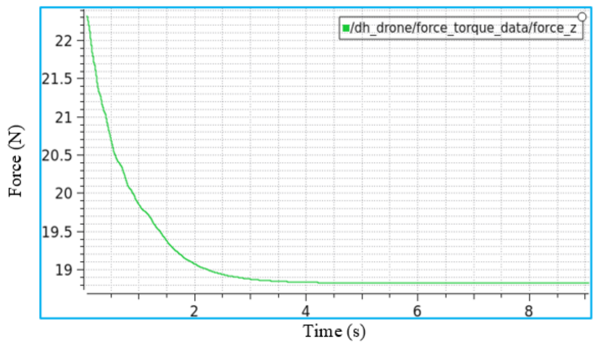
To obtain the results that confirm the system's effectiveness, the Plotjuggler tool, an open-source tool renowned for visualizing real-time data in ROS2 applications, is employed. This tool enables quickly and easily plotting data from different topics and nodes in a ROS2 system. Plotjuggler works by 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.

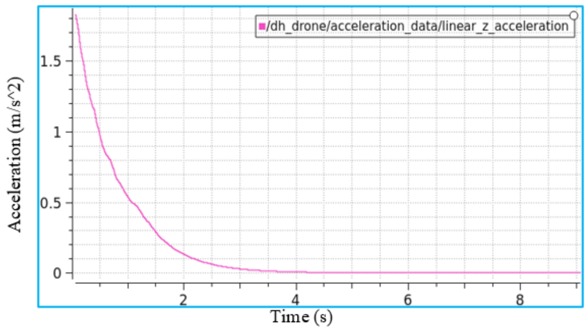
Moreover, the comprehensive analysis of the quadrotor system encompassed an evaluation of key performance metrics and criteria, including velocity, acceleration, force, and motion control, along the x, y, and z axes, as well as yaw motion. The illustrated results in Figures 8, 9, 10, and 11 highlight 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



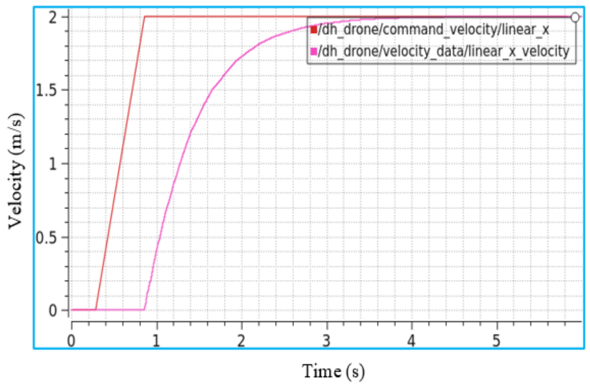
(c) Results of altitude: acceleration response

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

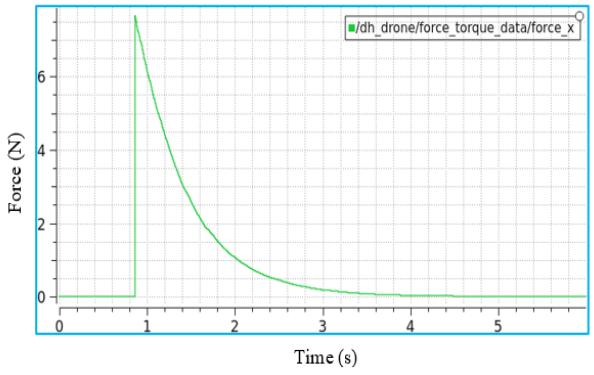
Quantitative analysis of the quadrotor's performance unveils promised statistics. During the evaluation process, the quadrotor exhibited outstanding capabilities in various domains. Along the z-axis, according to the given velocity of 1m/s. The quadrotor demonstrated swift acceleration, reaching an average of 1.8m/s². 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/s², 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.

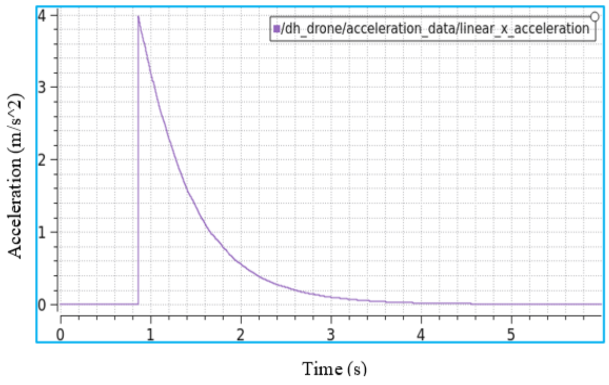
In summary, the impressive obtained statistics suggest that the quadrotor system exhibits minimal deviation in control. The consistent values for velocity, acceleration, force, and torque highlight its reliability and precision, ensuring stable and accurate flight maneuvers. Thus, the analysis confirms 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

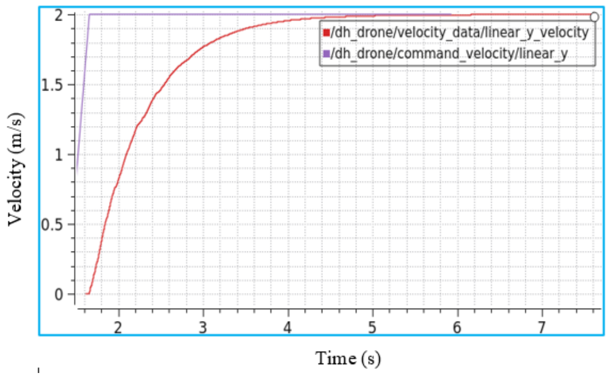


(b) Results of x motion: generated force response

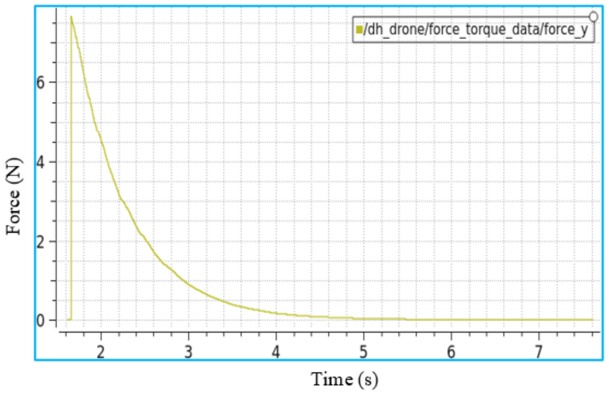


(c) Results of x motion: acceleration response

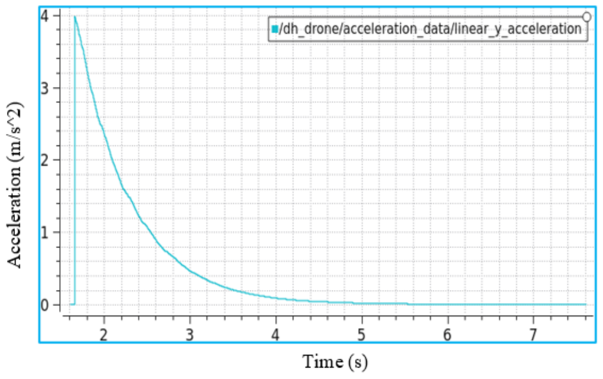
Figure 9. Results of optimized x motion control of quadrotor achieved: precise control commands enhance velocity, force, and acceleration



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

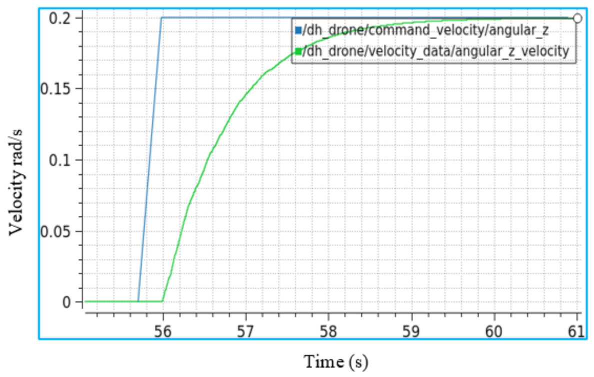


(b) Results of y motion: generated force response

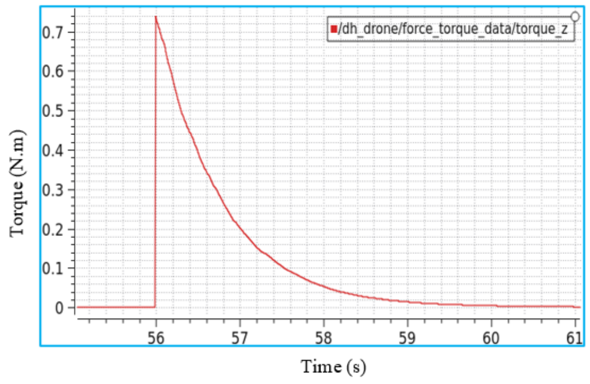


(c) Results of y motion: acceleration response

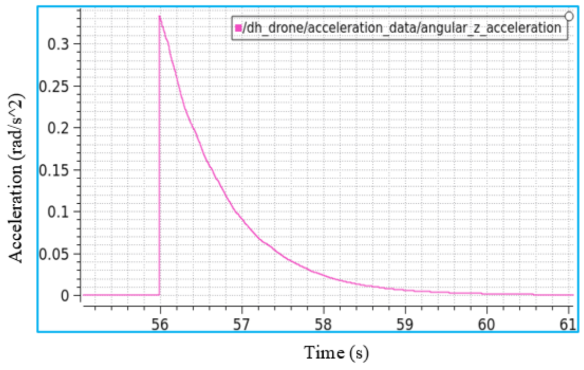
Figure 10. Results of optimized y motion control of quadrotor achieved: precise control commands enhance velocity, force, and acceleration



(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