

Virtual Reality Headset based Control and Navigation of Drone



By

Muhammad Mehran Javed

(Registration No: 00000328019)

Department Of Robotics & Artificial Intelligence
School Of Mechanical & Manufacturing Engineering
National University of Sciences & Technology (NUST)
Islamabad, Pakistan

(2024)

Virtual Reality Headset based Control and Navigation of Drone



By

Muhammad Mehran Javed

(Registration No: 00000328019)

A thesis submitted to the National University of Sciences and Technology, Islamabad,

in partial fulfillment of the requirements for the degree of

Master of Science in
Robotics & Artificial Intelligence

Supervisor: Dr. Muhammad Tauseef Nasir

Co Supervisor: Dr. Zaib Ali

School Of Mechanical & Manufacturing Engineering
National University of Sciences & Technology
Islamabad, Pakistan

(2024)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Regn No. 00000328019 Muhammad mehran Javed of School of Mechanical & Manufacturing Engineering (SMME) has been vetted by undersigned, found complete in all respects as per NUST Statues/Regulations, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis titled. Virtual reality (VR) headset-based control and navigation of drone.

Signature:



Name (Supervisor): Muhammad Tauseef Nasir

Date: 25/9/24

Signature (HOD):



Date: 25/9/24

Signature (DEAN):



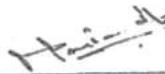
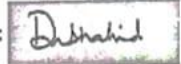

Date: 25/9/24



National University of Sciences & Technology (NUST)
MASTER'S THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: Muhammad mehran Javed (00000328019)
Titled: Virtual reality (VR) headset-based control and navigation of drone, be accepted in partial fulfillment of the requirements
for the award of MS in Robotics & Intelligent Machine Engineering degree.

Examination Committee Members

- | | | |
|----|-------------------------------|--|
| 1. | Name: Zaib Ali | Signature:  |
| 2. | Name: Shahid Ikram Ullah Butt | Signature:  |
| 3. | Name: Muhammad Safdar | Signature:  |

Supervisor: Muhammad Tauseef Nasir

Signature: 

Date: 26-Aug-2024

Head of Department

26-Aug-2024
Date

26-Aug-2024
Date

COUNTERSIGNED


Dean/Principal

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in this thesis, entitled “**Virtual Reality Headset based Control and Navigation of Drone**” was conducted by Mr. **Muhammad Mehran Javed** under the supervision of **Dr. Muhammad Tauseef Nasir**.

No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the **Department of Robotics & Artificial Intelligence** in partial fulfillment of the requirements for the degree of Master of Science in Field of **Robotics and Intelligent Machine Engineering** at **School of Mechanical and Manufacturing Engineering**, National University of Sciences and Technology, Islamabad.

Student Name: Muhammad Mehran Javed

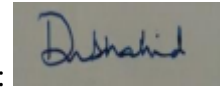
Signature:



Examination Committee:

a) External Examiner 1: Dr. Shahid Ikram Ullah
(Professor & HOD DME)

Signature:



.....

b) External Examiner 2: Dr. Muhammad Safdar
(Associate Professor)

Signature:



.....

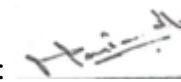
Supervisor Name: Dr. Muhammad Tauseef Nasir

Signature:



Co - Supervisor Name: Dr. Zaib Ali

Signature:



Name of Dean/HOD: Dr. Kunwar Faraz

Signature:



AUTHOR'S DECLARATION

I *Muhammad Mehran Javed* certify that this research work titled “*Virtual Reality Headset based Control and Navigation of Drone*” is my own work and has not been submitted previously by me for taking any degree from National University of Sciences and Technology, Islamabad or anywhere else in the country/ world.

At any time if my statement is found to be incorrect even after I graduate, the university has the right to withdraw my MS degree.

A handwritten signature in blue ink, consisting of a stylized 'M' and 'J' with a star at the end.

Signature of Student

Muhammad Mehran Javed

2020-NUST-MS-R&AI-328019

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled “**Virtual Reality Headset based Control and Navigation of Drone**” is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and National University of Sciences and Technology (NUST), Islamabad towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the University reserves the rights to withdraw/revoke my MS degree and that HEC and NUST, Islamabad has the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized thesis.



Student Signature: _____

Name: Muhammad Mehran Javed

DEDICATION

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment.

ACKNOWLEDGEMENTS

In the infinite grace and wisdom of Allah Subhana-Watala, I find my strength and guidance. It is with His divine light that I have been blessed and guided throughout this academic Endeavor and in all facets of my life. To Him, I owe all gratitude, for it is through His will that I have been fortunate to be surrounded by people who have been pillars of support and guidance.

My deepest appreciation goes to my cherished mother, who has been an ocean of love, resilience, and patience. Her Unconditional faith in me and her prayers have been the bedrock on which I have built my aspirations.

To my dear siblings, who have walked with me through thick and thin, your belief in me and your constant encouragement have been invaluable. I am also profoundly grateful to my supervisor, Dr. Muhammad Tauseef Nasir, who has been a mentor for excellence. His expertise, insights, and dedication have significantly shaped this work. Throughout the journey of this thesis, he provided invaluable guidance, constructive criticism, and a reservoir of patience. His unwavering support and wisdom have been pivotal in turning my aspirations into reality.

A special note of thanks to my friends who have been an enduring source of safety, laughter, and reassurance. Their constant presence and belief in my capabilities, especially during moments of self-doubt, have been immensely uplifting.

Lastly, every soul who has contributed, even in the slightest way, to my academic journey has my heartfelt gratitude. Whether it was a word of encouragement, a gesture of support, or a piece of advice, I cherish it deeply. To all, may Allah shower His countless blessings upon you, and may He guide us in all our endeavors.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	IX
TABLE OF CONTENTS	X
LIST OF TABLES	XI
LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS	XIII
ABSTRACT	XIV
CHAPTER 1: INTRODUCTION	1
1.1 Background:	1
1.2 Integration of VR with Robotics: Key Use Cases and Statistics	4
1.3 Thesis Organization:	5
CHAPTER 2: LITERATURE REVIEW	6
CHAPTER 3: RESEARCH OBJECTIVES	10
3.1 Objectives Overview:	10
3.2 Contributions	12
CHAPTER 4: METHODOLOGY AND WORKING	14
4.1 System Architecture	14
4.2 System Workflow	16
4.3 Implementation Details	17
4.4 Key Components in Unity and ROS Integration	20
4.5 Unity integration with Drone:	24
CHAPTER 5: RESULT AND DISCUSSION	27
5.1 Drone Control in Simulation:	27
5.2 Drone Control in Real Time:	28
5.3 User Feedback on VR-Based Drone Control System:	30
CHAPTER 6: CONCLUSION	32
APPENDIX A	35
7.1 Unity Side	35
7.2 ROS Side	41
REFERENCES	61

LIST OF TABLES

Table 1 - Main features of Oculus Quest 2 2

LIST OF FIGURES

Figure 1 - Oculus Quest 2. From the left, the head-mounted display, the left controller (Controller).	2
Figure 2 - Block Diagram representing overall system.	20
Figure 3 - Overall System Architecture.	20
Figure 4 - Unity side before running simulation.	25
Figure 5 - ROS side before running simulation.	26
Figure 6 - Left side window shows Unity scene and right-side window shows ROS side after simulation runs.	27
Figure 7 - User using controller to control drone in simulation.	28
Figure 8 - User using controller to control drone in real time	28
Figure 9 - Plot showing user input velocities and drone velocities in real time	29
Figure 10 - Feedback of user interacting with simulation.	30
Figure 11 - Feedback of user interacting with real-time.	31

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

ROS	Robot Operating System
HMI	Human-Machine Interface
VR	Virtual Reality
UAV	Unmanned Aerial Vehicle
MQTT	Message Queuing Telemetry Transport

ABSTRACT

Virtual reality is a new technology that has been used globally. Users across the globe are integrating VR into the field of robotics. The integration of Virtual Reality with Unmanned Aerial Vehicles (UAVs) helps users using their drone control abilities to perform drone operations in a safe side. This study focuses on the integration of Oculus Quest 2 VR headset to improve the operator's experience in both simulated and real-world. To evaluate the effectiveness of this integration, a comparative analysis was carried out between the Oculus Quest 2 and traditional controllers like FLYSKY FSi6x with PX4-autopilot based drone. Result of comparison shows that Oculus Quest 2 gives a better experience, as it enhances user interaction. To further analyze the efficiency, the system was tested in simulation as well as in real life scenarios. These analyses show that the proposed system performs well. A noticeable delay of approx. 0.13 sec in real time control observed between input from the operator and the drone response which is negligible. To keep the communication smooth and without lag between the controllers and drone control, Message Queuing Telemetry Transport (MQTT) is used. The analysis thus verifies that Oculus Quest 2, with the ability of better user experience can replace traditional controllers in real life. The framework used in this work supports the extension for operating other types of robots using ROS for their manipulation.

Keywords: *Virtual Reality, Oculus Quest, Navigation, MQTT (Message Queuing Telemetry Transport), ROS (Robot Operating System).*

CHAPTER 1: INTRODUCTION

1.1 Background:

The rapid evolution of robotics and unmanned aerial vehicles (UAVs), commonly known as drones, has opened up a multitude of possibilities across various industries. From agriculture to defense, drones are now at the forefront of technological innovation, providing unprecedented capabilities in surveillance, delivery, disaster management, and environmental monitoring [1].

However, as drone technology continues to advance, so does the complexity of their control systems. Traditional methods of controlling drones, typically through handheld remotes or computer interfaces, present limitations that can hinder user experience, accuracy, and safety. These conventional systems require a significant amount of training and expertise, and even then, the risk of human error remains high.

This thesis is motivated by the need to enhance the user experience in drone operation by leveraging emerging technologies, specifically Virtual Reality (VR). VR has gained significant traction in recent years, not only in gaming and entertainment but also in training, education, and simulation. The immersive nature of VR provides an intuitive interface that can bridge the gap between human operators and complex robotic systems. By integrating VR into drone control, we can create an environment that allows users to interact with drones in a more natural and effective manner.

The Oculus Quest 2 (Figure 1), a cutting-edge VR headset, offers advanced capabilities that make it an ideal tool for this purpose. With its precise tracking, intuitive controls, and wireless freedom, the Oculus Quest 2 allows for a level of immersion that was previously unattainable. In Table 1, the main technical features of the Oculus Quest 2 VR system are reported. Using Oculus Quest 2 headset in this thesis helps to explore how these features can be harnessed to create a VR-based control system for drones, enabling users to practice and execute drone operations in both simulated and real-world environments.



Head-Mounted Display



Controller A



Controller B

Figure 1 - Oculus Quest 2. From the left, the head-mounted display, the left controller (Controller).

[2]

Table 1 - Main features of Oculus Quest 2.

[2]

Memory	6 GB
Storage	128 GB
Display	LCD 1832 × 1920 per eye @ 72–120 Hz
Graphics	Adreno 650 (~1.2 TFLOPS)
Sound	2 built in speaker/3.5 mm headphone jack
Input	6DOF Inside-out tracking through 4 built-in cameras and 2 controllers with accelerometers and gyroscopes Optional: QWERTY keyboard (via Bluetooth)
Controller Input	Oculus Touch
Camera	4 Infrared Cameras
Mass	503g

Robotics is a field that encompasses the design, construction, operation, and use of robots. It draws on multiple disciplines, including mechanical engineering, electrical engineering, computer science, and artificial intelligence, to develop machines that can perform tasks autonomously or semi-autonomously. Drones, as a specialized category of robotics, have evolved rapidly over the past decade. Initially used primarily for military purposes, drones are now employed in a wide range of civilian applications.

The evolution of drone technology has been driven by advancements in several key areas [3]:

1. **Sensor Technology:** Modern drones are equipped with a variety of sensors, including GPS, accelerometers, gyroscopes, cameras, LiDAR, and more. These sensors enable drones to navigate, stabilize, and perform complex tasks with high precision.
2. **Autonomous Navigation:** The development of algorithms that allow drones to operate autonomously has been a major breakthrough. These algorithms use sensor data to enable drones to navigate without human intervention, avoiding obstacles, and completing missions autonomously.
3. **Communication Systems:** Drones rely on robust communication systems to transmit data between the drone and the operator. Advances in wireless communication, including 5G technology, have significantly improved the reliability and speed of these data transmissions.
4. **Machine Learning and AI:** The integration of machine learning and artificial intelligence into drones has expanded their capabilities. Drones can now recognize objects, track targets, and make decisions based on real-time data analysis.

Despite these advancements, the human-machine interface (HMI) remains a critical component of drone operation. The effectiveness of an HMI of a drone can determine the success or failure of a mission. Traditional interfaces, such as handheld controllers and computer-based systems, can be cumbersome and difficult to master. They require extensive training and can be prone to errors, particularly in high-stress situations.

The integration of VR in the control and navigation of drones is a relatively new area of research, with studies emerging over the past few years. Previous research has focused on

various aspects of application of VR in robotics and UAVs, providing a foundation for the development of the VR-based control system proposed in this thesis.

1.2 Integration of VR with Robotics: Key Use Cases and Statistics

The integration of Virtual Reality (VR) with robotics has been increasingly recognized for its potential across various sectors, offering both immersive experiences and enhanced operational capabilities. Below are some notable use cases, accompanied by relevant statistics that underscore the impact of these technologies:

1. Training and Education:

VR provides immersive training simulations, allowing users to engage in realistic scenarios, while robotics offers hands-on practical experience, thus enhancing the overall learning process. According to a study by PwC, VR-based training programs have led to a 30% reduction in learning time and a 12% improvement in accuracy compared to traditional methods [4]. Additionally, the global market for educational robots is projected to reach \$1.7 billion by 2027.

2. Manufacturing and Industrial Automation:

VR is utilized for optimizing virtual assembly lines, enabling detailed planning and testing before implementation. Robotics, on the other hand, automates repetitive tasks, significantly improving efficiency and reducing human error. Recent data indicates that global sales of industrial robots reached 384,000 units in 2020, with the automotive sector accounting for approximately 30% of these installations [5].

3. Healthcare and Surgery:

VR is employed for surgical planning and pre-operative simulations, allowing surgeons to rehearse complex procedures. Robotics complements this by facilitating minimally invasive surgeries with enhanced precision. The global market for surgical robots is forecasted to reach \$13.1 billion by 2027, with a compound annual growth rate (CAGR) of 22.6% from 2020 to 2027 [6]. Moreover, a study published in JAMA Network Open reported that VR surgical simulations resulted in a 230% improvement in surgical performance compared to traditional training techniques [7].

4. **Remote Operation and Telepresence:**

VR enables the remote operation of robots in hazardous or geographically distant environments, offering real-time feedback and enhanced situational awareness, which is crucial for high-stakes operations. The market for telepresence robots is expected to grow at a CAGR of 16.3%, reaching \$477.8 million by 2027 [8].

5. **Entertainment and Gaming:**

VR offers deeply immersive gaming experiences, while robotics enhances these experiences through physical interactions and advanced haptic feedback, making the virtual world feel more tangible. The global VR gaming market is anticipated to grow at a CAGR of 30.2%, reaching \$70.57 billion by 2028 [9]. Additionally, sales of consumer robots, including entertainment robots, are projected to hit \$15.1 billion by 2023 [10].

These above-mentioned range of applications and scope illustrate the powerful synergy between VR and robotics, enhancing training methodologies, improving industrial efficiency, advancing surgical techniques, facilitating remote operations, and creating more immersive entertainment experiences. As the adoption of these technologies expands, further innovations and broader applications are expected across various industries.

1.3 Thesis Organization:

The remainder of this thesis is structured as follows:

- **Chapter 2: Literature Review** - Provides an overview of existing research on the use of VR in different fields.
- **Chapter 3: Research Objectives** – Details the objectives for the implementation of a VR-based control system that allows users to operate drones in both simulated and real environments.
- **Chapter 4: Methodology and Working** - Discusses the implementation of the proposed system, covering the tools and technologies used.
- **Chapter 5: Results and Discussion** - Presents the results of the system, and discusses the parameters such as accuracy, precision, time consumption etc.
- **Chapter 6: Conclusion and Future Work** - Summarizes the key findings of the research and outlines potential directions for future research.

CHAPTER 2: LITERATURE REVIEW

The integration of Virtual Reality (VR) technology with unmanned aerial vehicles (UAVs) has been greatly improved by user interaction and control precision in various applications, especially in drone piloting. For UAVs, the use of VR systems, like head-mounted displays (HMDs) has created more immersive and intuitive control environments. This literature review explores several studies that contribute to the use of VR-based drone control systems in both simulation and real-time environments, particularly in relation to the control of UAVs.

An important contribution to the field of VR interaction with drones is a VR-based training and assessment system for drone-assisted bridge inspection. This system has multiple modules, including a simulated environment in Unity to enable trainees to pilot a drone using a remote controller. The author highlights that real-time feedback and training is needed to assist inspectors to develop the necessary skills and confidence to fly drones as part of bridge inspection [11]. This is directly relevant to the use of Oculus Quest 2 controllers in drone flight.

In a relevant study, the use of MEMS sensor technology in combination with machine learning was considered for drone control in virtual environments. This technique uses sensors that are attached to the upper limb of the person controlling the system, thus allowing for fine control in such an environment [12].

The application of Head Mounted Displays as a means of control for the robotic system, for example the da Vinci Surgical System, provides an understanding of the relevance and efficiency of the translation of head movement to control commands. In the study [13], the HTC Vive headset was combined with the integration of Unity and ROS, suggesting that users' workload both physical and mental while controlling a robot is decreased. The work done in this study and the use of Oculus Quest 2 for drone control underscores the use of VR technology in terms of operator fatigue reduction and control precision.

VR sickness, being one of the important issues in VR based control system, has been addressed in research that proposed head-synced drone control to minimize sensory conflict [14]. A comparison between the joystick control with head control had been made in the study, showing that syncing head orientation with the drone's attitude could significantly reduce VR sickness. This shows that VR sickness is an important factor when using Oculus Quest 2 for extended drone operations.

In a study that involves the development of a P300-based brain-computer interface drone control system, that is compatible with VR and augmented reality offers a unique demonstration of how VR technology can be used [15]. Although the study was focused on the discussion of an innovative interface, it also indicates that using VR systems can guarantee users high reliability of the control which is considered when comparing the effectiveness of Oculus Quest 2 with traditional drone transmitters.

Another study that uses military applications, involves teleoperation of mobile robots, such as the TAROS system using VR headset, the Oculus Rift. The use of this headset for immersive control in a virtual space that represents the real-time data from the robot opens the opportunity for the same approach in the teleoperation of drones [16].

Furthermore, the teleoperation of UAVs for real-time mapping tasks shows the advancements in VR control systems [17]. This study utilized Unity3D to create virtual environments for UAVs flight that can be mirrored to the same real-world conditions. This demonstrates the practical application of VR in operational settings where real-time feedback and precise control are needed.

Also, a study involving the immersive teleoperation of a robot in a search and rescue (SAR) operations using a head-mounted display provides further evidence of the benefits of VR-based control systems [18]. This research demonstrated the use of an HMD for robot control that results in the improvement of depth perception, situational awareness, and overall performance during SAR missions. These findings are directly applicable to the use of Oculus Quest 2 for drone control, particularly in scenarios where high levels of precision and situational awareness are required.

In a study related to biomedical that showcased the evaluation of the accuracy of the Oculus Quest 2 for shoulder rehabilitation by measuring translational and rotational displacements gives positive attitude towards the use of VR [19]. Although focused on rehabilitation, the study's findings can be linked with drone control where accurate and responsive input is crucial.

Furthermore, role of VR in training related to safety and hazard assessment, as explored in the work involving underground roof fall hazard assessment [20], illustrates the use of VR simulations in training environments. The ability to train users to deal with the hazardous conditions in a safe, controlled way is a key advantage of VR technology, which can be leveraged in the training and operation of drones using Oculus Quest 2.

Finally, the potential of VR in enhancing the decision-making process during UAV operation is discussed in a study that involves the generation and VR visualization of 3D point clouds for drone target validation. This study allows user to validate the target selection through visualizing simulated 3D data with the help of VR headset [21].

Recent advancements of the Virtual (VR), Augmented (AR) and Mixed Reality (MR), enable us to perform the teleoperation from either the real or the virtual world. The interface utilized in the study[22] enables users to execute a variety of articulated robot operations in a user-friendly way. Thus, virtual reality interfaces in robotics play a crucial role by enabling people without specialized expertise to remotely control robots.

In relation to virtual reality (VR), a study conducted by researchers in [23] examined the effectiveness of two distinct VR interfaces: location control, which is similar to navigating via waypoints, and trajectory control, which is analogous to the action of clicking and dragging. The objective was to operate a Baxter robot from a distance in order to accomplish a range of complex manipulation tasks. The study in [24] presented an innovative system architecture and control algorithms to improve the telepresence and maneuverability utilizing MR, which integrates real and virtual environments. Naceri et al. [25] introduced Vicarios, a virtual reality (VR) teleoperation interface designed to enhance the ease and immediacy of distant teleoperation in real-time. The study conducted in [26]

employed virtual reality to create a human-robot collaborative welding system that facilitated cooperation between people and robots in carrying out welding activities.

Recent work in [27] presented the use of VR headset, Oculus Rift to teleoperate NAO which is based on ROS. Inspired by this previous work done, we will be using Oculus Quest 2 for controlling drone based on ROS because of its major benefit of user familiar interface. In the Results and Discussion section, the suggested approach is verified.

In conclusion, all these studies collectively illustrate the advancements in VR-based control systems for drones and other robotic systems which can be used to integrate Oculus Quest 2 with platforms like Unity and ROS for drone control, both in simulation and real-time.

CHAPTER 3: RESEARCH OBJECTIVES

3.1 Objectives Overview:

From the above literature, the Oculus Quest 2 was found to be the most suitable VR headset with applied application. It offers several features that make it particularly well-suited for integration with drone control systems, such as:

1. **Wireless Freedom:** Unlike many other VR headsets, the Oculus Quest 2 does not require a tethered connection to a computer. This allows for greater freedom of movement, which is crucial in applications that involve physical interaction with the environment, such as drone operation.
2. **Precision Tracking:** The Oculus Quest 2 uses inside-out tracking, which relies on infrared sensors built into the headset to track the user's movements. This eliminates the need for external sensors and provides highly accurate tracking of head and hand movements.
3. **Intuitive Controllers:** The Oculus Quest 2 comes with ergonomic controllers that are designed to be easy to use. These controllers feature buttons, triggers, and thumbsticks that can be mapped to various drone controls, providing a familiar and intuitive interface for users.
4. **High-Resolution Display:** The headset features a high-resolution display that provides clear and detailed visuals. This is particularly important in drone operation, where visual feedback is essential for navigation and control.
5. **Expandable Ecosystem:** The Oculus Quest 2 is part of a larger ecosystem of VR applications and tools. This includes software development kits (SDKs) and APIs that allow developers to create custom applications and integrate the headset with other systems, such as drones.

The primary objective of this thesis is to develop and evaluate a VR-based control system for drones using Oculus Quest 2. This system aims to bridge the gap between simulation and real-world operation, providing users with a seamless transition from virtual to actual

drone control. To the best of the author's knowledge, the technique has not been developed and implemented before. The specific research objectives include:

1. **System Development:** Design and implement a VR-based control system that allows users to navigate and operate drones in both simulated and real environments. This involves integrating the Oculus Quest 2 with the control system of the drone using Unity and ROS and developing a user interface that is both intuitive and effective.
2. **User Experience Evaluation:** Conduct a series of experiments to evaluate the user experience of the VR-based control system. This includes assessing the ease of use, precision, and situational awareness provided by the system, as well as the overall user satisfaction.

The scope of this research is limited to the use of the Oculus Quest 2 in controlling quadcopter drones. While the principles and techniques developed in this thesis could be applied to other types of drones or robotic systems, the focus will remain on quadcopters due to their widespread use and versatility.

The integration of VR with drone control presents several challenges, both technical and operational. Some of the key challenges include:

1. **User Adaptation:** While VR provides an immersive experience, it can also be disorienting for some users, particularly those who are not familiar with the technology. Ensuring that the VR interface is intuitive and easy to use is critical to the success of the system.
2. **System Integration:** Integrating the Oculus Quest 2 with the drone's control system requires careful consideration of both hardware and software compatibility. This includes ensuring that the VR headset can communicate effectively with the flight controller of the drone via ROS.
3. **Safety:** Operating drones in real-world environments poses inherent risks, particularly when using a new control system.

Despite these challenges, the opportunities presented by VR-based drone control are significant. The ability to practice and refine drone operations in a simulated environment

has the potential to greatly reduce the risk of accidents and improve overall operational safety. Additionally, the intuitive nature of VR controls could make drone technology more accessible to a wider audience, including those with limited technical expertise.

3.2 Contributions

This thesis advances the field of drone control and navigation by integrating Virtual Reality (VR) technology, specifically using the Oculus Quest 2 VR headset. A comprehensive simulation environment was developed where users can control a PX4-autopilot-based drone through an immersive VR interface. This integration enhances user-drone interaction, making the control more intuitive for beginners while improving precision for experienced pilots.

A significant contribution of this work is the comparative analysis between the Oculus Quest 2 VR controllers and traditional controllers like the FLYSKY FSi6x. The findings reveal that the Oculus Quest 2 provides superior user experience, offering enhanced control accuracy and interaction. This is particularly beneficial for new pilots, as the VR system simplifies complex maneuvers, reducing the learning curve and the risk of operational errors.

The research also addresses the critical need for safe training environments. By enabling pilots to practice in a risk-free VR setting, this system reduces the potential dangers associated with real-world drone flights. Pilots can simulate various scenarios and refine their skills without the consequences of real-life crashes or damage, ultimately leading to better performance in actual drone operations.

Furthermore, the thesis introduces a framework that ensures a smooth transition from virtual simulation to real-world drone control. This seamless shift is facilitated by the use of the MQTT protocol, which ensures real-time, lag-free communication between the VR headset and the PX4-autopilot-based drone. The integration of this technology not only supports effective training but also demonstrates the practical applicability of VR-based control systems in real-world UAV operations.

Overall, this work establishes the Oculus Quest 2 as a viable alternative to traditional controllers for drone navigation and control, providing a foundation for further research and development in VR-based robotic systems. The framework developed in this study is adaptable for controlling other types of robots using the Robot Operating System (ROS), paving the way for broader applications in robotic manipulation and control.

CHAPTER 4: METHODOLOGY AND WORKING

This methodology uses the Oculus Quest 2 headset and its controllers to create an immersive VR-based drone control system. The system architecture considers the user-friendly experience with the use of Oculus controllers for minimizing error and safety.

Furthermore, the combination of Unity3D with ROS (Robot Operating System) is essential in creating an effective VR-based control system. The software Unity3D which enables developers to create realistic simulations where users can interact with virtual environments is used in this work [28]. ROS, on the other hand, is a flexible framework for writing robot software and is particularly useful for integrating various robotic systems, including drones, with VR interfaces. The combination of Unity and ROS allows for seamless communication between the VR headset and the drone, ensuring that user inputs are accurately translated into drone movements.

4.1 System Architecture

The system architecture is depicted in a block diagram that outlines the interaction and information flow between the various components: Human, VR Device, Unity, and the ROS-Based Environment. Each component plays a crucial role in ensuring that the drone can be controlled effectively through the VR interface.

Components and Their Roles

1. Human Operator:

- The human operator is the central element of the system, responsible for providing input through the VR controllers. The physical movements of operator, gestures, and commands drive the system, making their role pivotal in the successful operation of the drone.
- The operator interacts with the VR device, utilizing both the headset for visual feedback and the controllers for input. This interaction is critical as it dictates how the drone will navigate its environment, whether in a simulated or real-world setting.

2. **VR Device:**

- **Display:** The Oculus Quest 2 headset serves as the primary visual interface for the operator. It offers an immersive, first-person perspective of the surroundings of the drone, whether these are part of a real-world environment or a simulated scenario within Unity. This visual feedback is vital for precise navigation and situational awareness, as it allows the operator to perceive the position and movements of the drone in real-time.
- **Controllers:** The Oculus Quest 2 controllers capture the hand movements of the operator, gestures, and button presses. These inputs are crucial for controlling the drone, as they are translated into specific commands that direct the actions of the drone. The data from the controller is transmitted to Unity for further processing and conversion into commands that the ROS environment can understand.

3. **Unity:**

- Unity acts as the intermediary platform between the VR device and the ROS-based environment. It is responsible for processing the input from the VR controllers and translating these inputs into commands that can be executed by the drone. Unity also updates the visual display within the VR headset, providing the operator with real-time information based on feedback from the drone or the virtual environment.
- The role of unity is essential in creating a cohesive and interactive 3D environment where the operator can engage with the surroundings of the drone. It simulates real-world physics and scenarios, allowing for accurate control and navigation during both, the testing and live operations.

4. **ROS-Based Environment:**

- **Gazebo (Virtual Environment):** Gazebo is a simulation tool within the ROS framework that allows for the testing and development of drone control algorithms without needing to use a physical drone. Gazebo provides realistic physics, environmental conditions, and sensor data, which are crucial for creating a close-to-reality simulation experience. This allows for thorough testing and fine-tuning of the control system before deploying it in real-world scenarios.

- **Drone (Real-Time Environment):** The actual quadcopter drone executes the commands received from Unity in real-time. It sends feedback about its current state, including position, orientation, and sensor data, back to Unity. This feedback loop is essential for the operator to make real-time adjustments, ensuring the drone operates effectively and safely.

4.2 System Workflow

The flow of information within this system is critical to its seamless operation. The following steps detail how information travels through the system, ensuring that the drone responds accurately to the commands of the operator:

1. Human to VR Device:

- The human operator initiates the process by providing input via the VR controllers. These inputs can include movements, gestures, and button presses, which are captured by the VR device. This initial step is crucial as it sets the stage for how the drone will be controlled.

2. VR Device to Unity:

The VR device then sends the captured inputs to Unity. Unity processes these inputs, converting them into commands suitable for the ROS-based environment. Simultaneously, the VR headset provides visual feedback to the operator, enhancing the immersive experience and ensuring the operator remains engaged and aware of the environment of the drone.

3. Unity to ROS-Based Environment:

- Unity acts as a bridge, forwarding the processed commands to the ROS-based environment. In simulation mode, these commands are directed to Gazebo, where they are executed in a virtual setting. During real-time operations, the commands are sent directly to the drone, which carries them out in the physical world.

4.3 Implementation Details

1. **Hardware Setup:**

- The hardware setup involves equipping the quadcopter drone with a Raspberry Pi, which is connected to the flight controller. The Raspberry Pi runs ROS, which enables communication with Unity and manages the physical movements of the drone based on the commands it receives. The Oculus Quest 2 headset and controllers serve as the VR interface, providing immersive control capabilities.
- The choice of Raspberry Pi as the central processing unit is strategic, given its ability to handle the computational demands of ROS while remaining lightweight and cost-effective. This setup ensures that the system is not only powerful but also portable, making it suitable for a wide range of applications.

2. **Software Integration:**

- Unity and ROS are the cornerstone of software integration, working together to facilitate communication between the VR device and the drone. Unity is responsible for creating the virtual environment and processing inputs from the VR controllers. The ROS-TCP-ENDPOINT is a critical component that establishes a communication link between Unity and ROS, ensuring smooth and reliable data transmission.

The integration of Unity and ROS is meticulously designed to ensure that the system can handle real-time data processing and command execution without lag or disruption. This is particularly important in scenarios where the drone needs to respond immediately to changes in its environment, such as in search and rescue missions or industrial inspections.

3. **Calibration and Testing:**

- Precise calibration is required to ensure that the VR environment accurately reflects the movements of the drone, and that the system responds as expected to the inputs from the operator. Initial testing is conducted in the Gazebo simulation environment, where the control algorithms can be fine-tuned, and the stability of the system can be ensured.
- The use of Gazebo as a testing platform is advantageous because it allows for the safe testing of the system in a controlled environment before deploying it in the real world.

This step is crucial in identifying any potential issues with the control system and making necessary adjustments to improve performance.

- Once the system is calibrated and thoroughly tested in the simulation environment, it is then ready for real-time drone operations. This transition from simulation to real-world application is carefully managed to ensure that the system remains stable and responsive, minimizing the risk of errors during live operations.

The methodology employed in this research is built on a foundation of advanced technological integration and innovative application of VR technology. Each component of the system is chosen and configured with the goal of creating a seamless and immersive control experience that enhances the ability of the operator to manage the drone effectively.

- **VR Device and Human Interaction:** The Oculus Quest 2 headset and controllers are at the heart of this system, providing the interface through which the human operator interacts with the drone. The choice of Oculus Quest 2 is particularly significant due to its high-quality display, precise tracking capabilities, and intuitive controller design. These features are crucial in creating an immersive experience where the operator feels fully engaged with the environment of the drone.
- **Unity and ROS Integration:** The role of Unity as the intermediary platform is critical in translating the inputs from the operator into actionable commands. The use of the XR Interaction Toolkit within Unity allows for the capture of complex hand movements and gestures, which are then processed and sent to ROS. ROS, running on the Raspberry Pi, interprets these commands and manages the flight operations of the drone. This integration ensures that the system can handle real-time control tasks with the precision and responsiveness required for dynamic environments.
- **MQTT integration as a communication protocol:** The Message Queuing Telemetry Transport (MQTT) protocol plays a pivotal role in ensuring reliable and efficient communication between the VR interface and the drone. In this system, MQTT is used to transmit control commands from the Oculus Quest 2 headset, processed through Unity and ROS, to the PX4-autopilot-based drone. The choice of MQTT is driven by its lightweight nature, low bandwidth consumption, and minimal latency, which are essential for maintaining real-time responsiveness in drone operations. This protocol

ensures that even in scenarios with limited network resources, communication remains robust and uninterrupted. By using MQTT, the system is capable of managing the bidirectional flow of data—such as telemetry from the drone and control inputs from the user—without lag or delay, which is crucial for applications requiring precise control and quick response times. This integration not only facilitates seamless interaction between the various system components but also enables scalability, allowing for potential expansion of the system to include multiple drones or additional control interfaces.

- **Simulation and Real-time Testing:** The use of Gazebo for simulation is a strategic choice that allows for extensive testing of the control system without the risks associated with real-world operations. Gazebo’s realistic physics engine provides a close approximation of real-world conditions, enabling the development and refinement of control algorithms in a safe and controlled setting. This simulation phase is essential for ensuring that the system is robust and capable of handling the complexities of real-time drone operations.
- **Calibration and Real-time Operations:** Calibration is a critical step in the methodology, ensuring that the VR environment is accurately aligned with the real-world movements of the drone. This process involves fine-tuning the system to ensure that the inputs from the operator are precisely reflected in the actions of the drone. The transition from simulation to real-time operations was carefully managed to maintain the stability and responsiveness of the system, which is crucial for applications where timing and accuracy are critical.

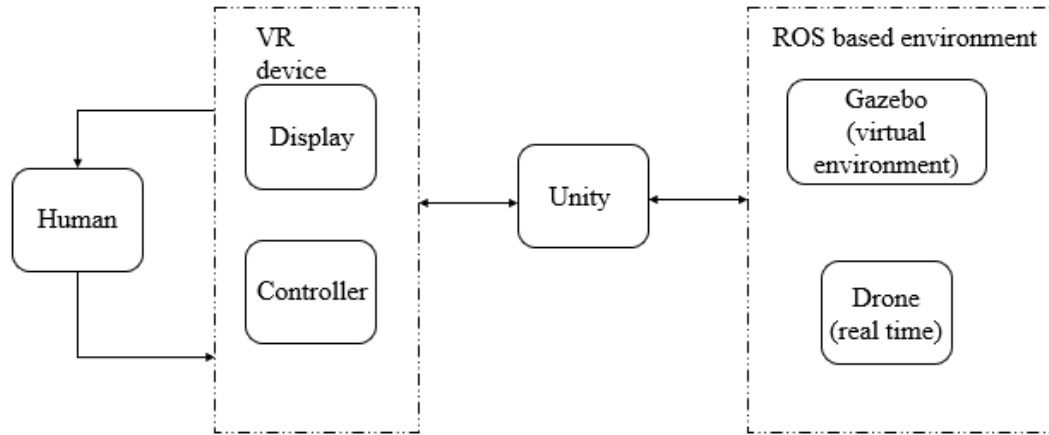


Figure 2 - Block Diagram representing overall system.

The system architecture is shown in the figure below:

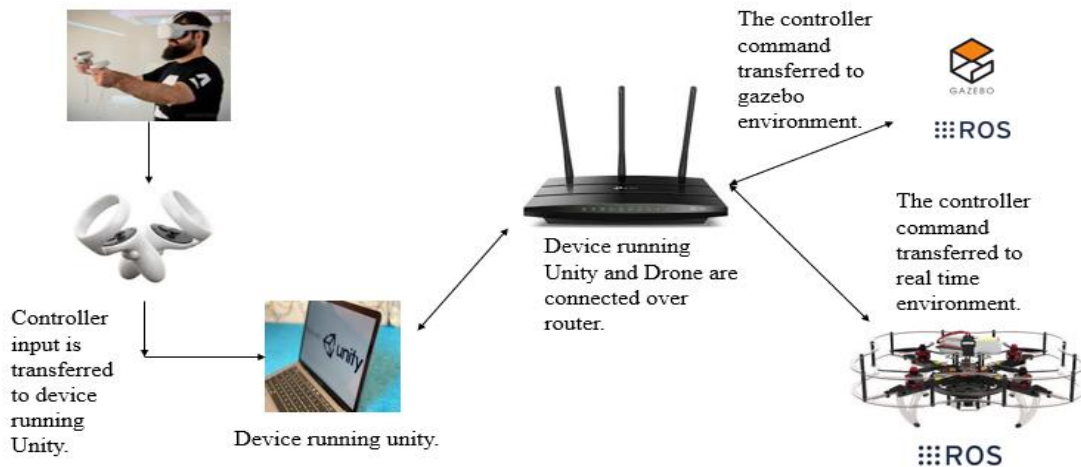


Figure 3 - Overall System Architecture.

4.4 Key Components in Unity and ROS Integration

In the development of the Virtual Reality (VR) drone control system, several key components were integral to the successful integration of Unity with the Robot Operating System (ROS). These components include the **XR Interaction Toolkit**, **Oculus XR Plugin**, **ROS-TCP-CONNECTOR** on the Unity side, and **ROS-TCP-ENDPOINT** on the

ROS side. Each of these tools played a crucial role in ensuring seamless communication between the VR environment and the control system of drone. Below is a detailed explanation of these components and their respective functions in the overall system architecture.

XR Interaction Toolkit

The XR Interaction Toolkit is a Unity package designed to simplify the development of VR and augmented reality (AR) applications. It provides a set of pre-built components and interaction models that allow developers to create intuitive and immersive VR experiences. In the context of this project, the XR Interaction Toolkit was utilized to handle user input from the Oculus Quest 2 controllers, enabling the operator to interact with the virtual environment and control the drone.

The toolkit includes various interaction components, such as direct and ray interactors, grab interactable, and interactable objects, which facilitate the mapping of physical gestures to virtual actions. For example, when the user presses a button or moves the controller, the XR Interaction Toolkit captures these inputs and processes them to trigger corresponding actions within Unity. This input is then translated into drone control commands that are sent to ROS.

One of the key advantages of using the XR Interaction Toolkit is its ability to handle complex interactions with minimal coding effort. It abstracts much of the complexity involved in VR development, allowing the focus to remain on creating a smooth and responsive control interface. The toolkit also supports various input devices, making it a versatile choice for VR development, particularly when integrating with other systems like ROS.

Oculus XR Plugin

The Oculus XR Plugin is a crucial component in this project, providing the necessary tools to integrate Oculus devices, such as the Oculus Quest 2 with Unity. This plugin is part of the broader Unity XR ecosystem, which supports a wide range of VR and AR devices. The

Oculus XR Plugin enables Unity to communicate with the Oculus hardware, facilitating the capture of head and hand movements and rendering the VR environment.

In this system, the Oculus XR Plugin is responsible for ensuring that the VR environment responds accurately to the physical movements of the user. It handles the low-level input from Oculus Quest 2, such as tracking the position and orientation of the headset and controllers. This data is essential for creating an immersive experience where the interactions of the user with the virtual world directly influence the behavior of the drone.

The plugin also optimizes performance for Oculus devices, ensuring smooth rendering and low latency in the VR environment. This is particularly important in a real-time control system, where any delay or jitter could negatively impact the operator's ability to control the drone accurately. The Oculus XR Plugin, therefore, plays a critical role in maintaining the responsiveness and realism of the VR control system.

ROS-TCP-CONNECTOR

The ROS-TCP-CONNECTOR is a Unity package designed to facilitate communication between Unity and ROS. It acts as a bridge, enabling Unity to send and receive messages over TCP/IP to a ROS environment. This connector is essential for integrating the VR control system with the flight controller of the drone, as it allows Unity to transmit user input data from the Oculus controllers to ROS, where it can be processed and executed.

The ROS-TCP-CONNECTOR is responsible for establishing a reliable connection between Unity and ROS, ensuring that data flows seamlessly between the two systems. It handles the serialization and deserialization of messages, converting them into a format that can be understood by ROS. For instance, when a user moves the VR controller to command the drone to move forward, the ROS-TCP-CONNECTOR packages this command into a ROS-compatible message and sends it to the ROS environment for execution.

Furthermore, the ROS-TCP-CONNECTOR supports bi-directional communication, meaning that it can also receive messages from ROS and relay them back to Unity. This capability is vital for providing real-time feedback to the operator, as it allows Unity to

update the VR environment based on the current state of the drone, such as its position, orientation, and sensor readings.

ROS-TCP-ENDPOINT

The ROS-TCP-ENDPOINT is the corresponding component on the ROS side, working in tandem with the ROS-TCP-CONNECTOR to facilitate communication between ROS and Unity. This endpoint listens for incoming TCP/IP connections from Unity and handles the reception and processing of messages. It plays a crucial role in ensuring that the commands generated in the Unity environment are accurately interpreted and executed by the ROS-based drone control system.

The ROS-TCP-ENDPOINT is configured to handle various types of messages, such as command inputs from Unity or status updates from the drone. When it receives a message from Unity, it processes the data and routes it to the appropriate ROS nodes responsible for controlling the movements of the drone or processing sensor data. This ensures that the drone responds correctly to the inputs from the operator, whether it involves changing direction, adjusting altitude, or performing complex maneuvers.

The endpoint also manages the flow of feedback data from ROS back to Unity. For example, as the drone moves, the ROS-TCP-ENDPOINT collects data on its position and sends this information back to Unity, where it is used to update the VR environment. This feedback loop is essential for providing the operator with real-time situational awareness, enabling them to make informed decisions during flight.

The ROS-TCP-ENDPOINT is designed to be highly reliable, ensuring that communication between Unity and ROS remains stable even in dynamic and unpredictable environments. Its ability to handle multiple types of messages and maintain a continuous connection with Unity is critical for the smooth operation of the VR control system.

Integration and Workflow

The integration of these components forms the backbone of the VR drone control system. The workflow begins with the XR Interaction Toolkit and Oculus XR Plugin in Unity,

capturing the inputs from the operator and rendering the VR environment. These inputs are then passed through the ROS-TCP-CONNECTOR, which packages them into ROS-compatible messages and sends them to the ROS-TCP-ENDPOINT.

Once the ROS-TCP-ENDPOINT receives these messages, it processes them and triggers the appropriate actions in the ROS environment, such as adjusting the flight path of the drone or altering its speed. The feedback from the drone, including its current state and sensor data, is then sent back to Unity via the same communication pathway. This data is used to update the VR environment, ensuring that the operator has an accurate and up-to-date view of the surroundings of the drone.

This continuous loop of input, processing, and feedback enables the operator to control the drone in real-time using Oculus Quest 2. The combination of these components ensures that the system is both responsive and reliable, providing a seamless and immersive control experience that can be applied to a wide range of applications, from industrial inspections to search and rescue missions.

4.5 Unity integration with Drone:

This section presents the integration of Virtual Reality (VR) technology with a drone control system, utilizing Unity and the Robot Operating System (ROS) with Gazebo for simulation. The information given below highlights various stages of system operation, from the initial setup and environment configuration to the simulation and real-time control of the drone by a user equipped with VR controllers.

Unity Scene Before Running Simulation

The initial stage of the operation of the system is captured in the Unity environment before the simulation is initiated. In this stage, the virtual scene is meticulously set up to provide a visual and interactive representation that facilitates user input. Unity is responsible for capturing input from the Oculus Quest 2 controllers, which will later be translated into commands for the drone. The setup includes all necessary elements for an immersive user

experience, ensuring that the VR controllers' function correctly and that the system is ready to interface with ROS.

At this point, the connection between Unity and ROS has not yet been established. The purpose of this stage is to prepare the Unity environment to receive input from the VR controllers. The inputs gathered here will be processed and sent to ROS, where the actual drone control occurs within the Gazebo simulation environment. The accurate setup in Unity is crucial because it determines how effectively the user's inputs will be translated into drone commands once the system is fully operational.

The pre-simulation setup also serves as a baseline, confirming that Unity is ready to capture and transmit the necessary data to ROS. Ensuring that all interactive elements in Unity respond correctly to user inputs is essential for the system's subsequent stages, where these inputs will be used to control the drone in the ROS environment.

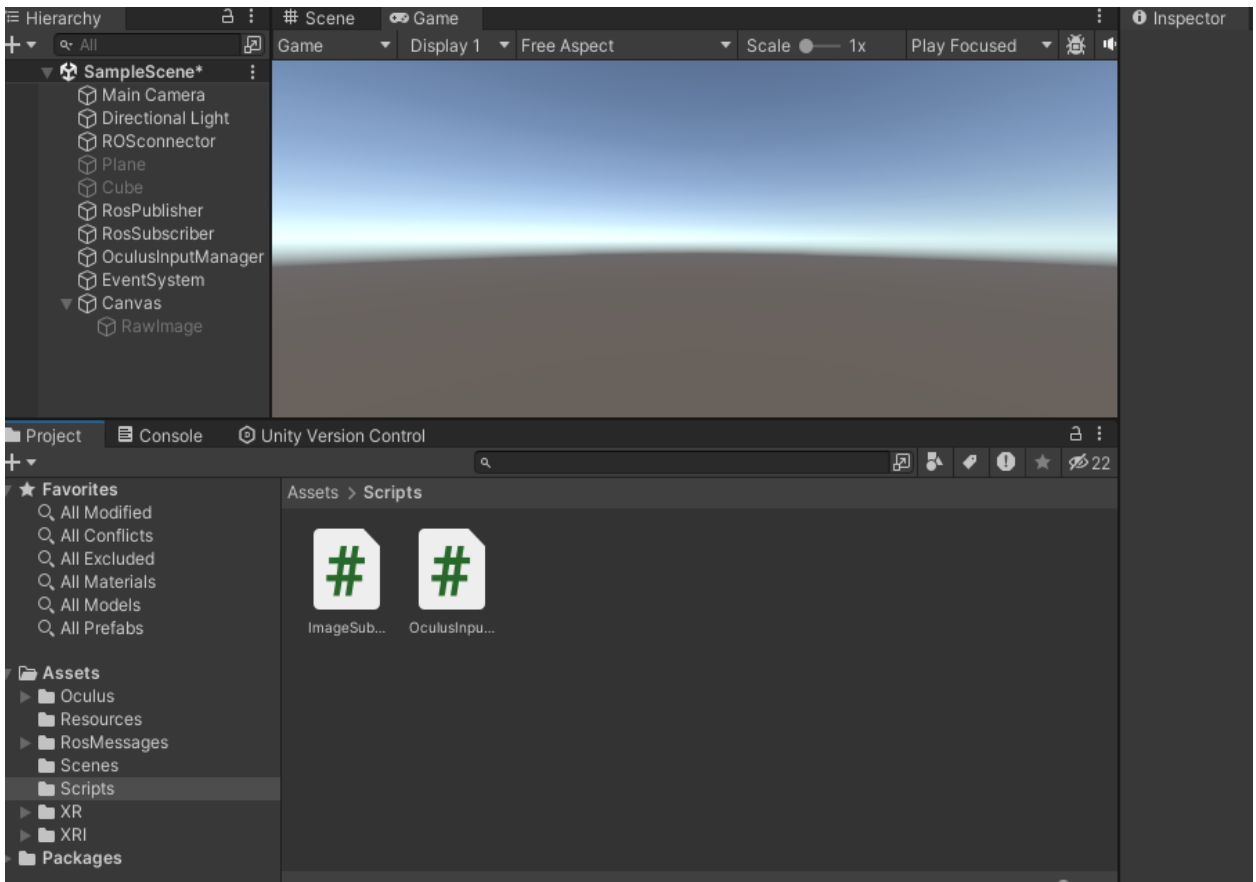


Figure 4 - Unity side before running simulation.

ROS Environment with Drone in Gazebo

The next stage of the process involves the ROS environment, where the drone is placed within the Gazebo simulation environment. This step is illustrated by an image showing the drone situated in the virtual environment provided by Gazebo. Gazebo is an integral part of the ROS framework, offering a high-fidelity simulation environment that includes realistic physics, sensor data, and environmental variables.

In this stage, the drone is positioned in a scenario that closely resembles real-world conditions, allowing for the testing and development of control algorithms without the risks associated with physical drone operations. The Gazebo environment is designed to simulate various factors that the drone might encounter during actual flights, such as wind, gravity, and obstacles. This realistic simulation is critical for ensuring that the control system will perform reliably when deployed in real-world applications.

The placement of the drone within Gazebo serves as the starting point for all subsequent testing and interaction. The accuracy of the simulation in Gazebo is essential for validating the effectiveness of the system before it is applied to a physical drone. By using Gazebo, the research ensures that any potential issues or errors in the control algorithms can be identified and corrected in a controlled environment, thereby reducing the risk of failure during live operations.

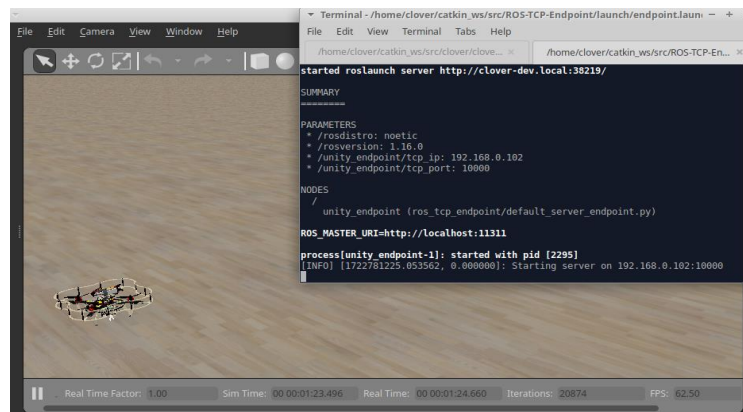


Figure 5 - ROS side before running simulation.

CHAPTER 5: RESULT AND DISCUSSION

This section shows the fusion of drone control using Oculus Quest 2 headset, focusing on the integration of Unity and the Robot Operating System (ROS). The images provided in this section showcase the successful integration and operation of all components.

5.1 Drone Control in Simulation:

The successful connection between Unity and ROS in the VR-based drone control system is demonstrated in figure 6: the left window shows the Unity environment where the connection to ROS has been established, and the right window shows the Gazebo simulation with the drone successfully controlled. This confirms that the commands generated through the VR controllers are transferred from Unity to ROS environment, that helps precise control of the drone within the simulation (figure 7). The seamless communication between Unity and ROS as seen in figures below highlights the effectiveness of the ROS-TCP-CONNECTOR and ROS-TCP-ENDPOINT.

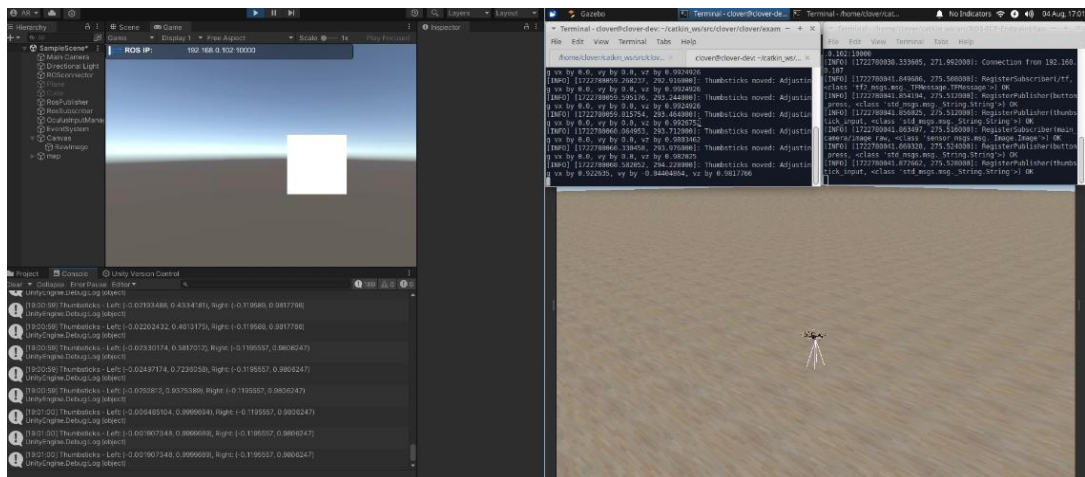


Figure 6 - Left side window shows Unity scene and right-side window shows ROS side after simulation runs.



Figure 7 - User using controller to control drone in simulation.

5.2 Drone Control in Real Time:

As shown in figure 8, in real-time drone control, the system is able to effectively translate the user commands provided by the Oculus Quest 2 controllers into corresponding actions of the drone. This indicates that the connection between Unity and ROS that has been established successfully and allows the commands to be relayed, ensuring that the drone can respond to the operator's control.

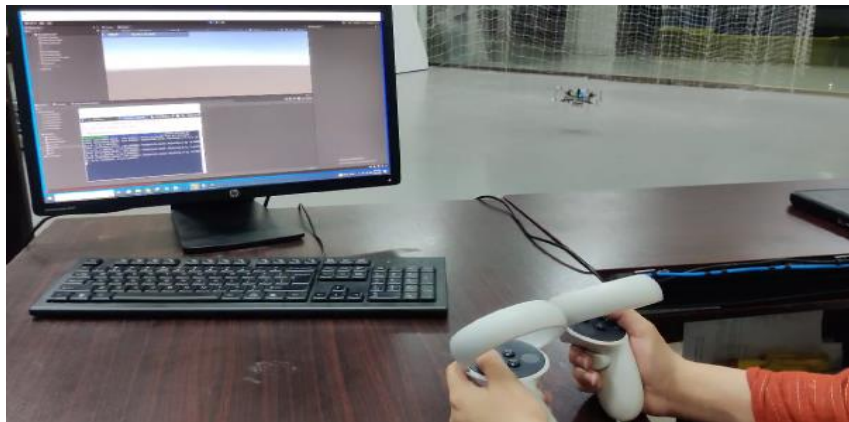


Figure 8 - User using controller to control drone in real time

To evaluate the system's performance in different real-world conditions, the authors generated velocity plots (figure 9) to check the drone response in real time.

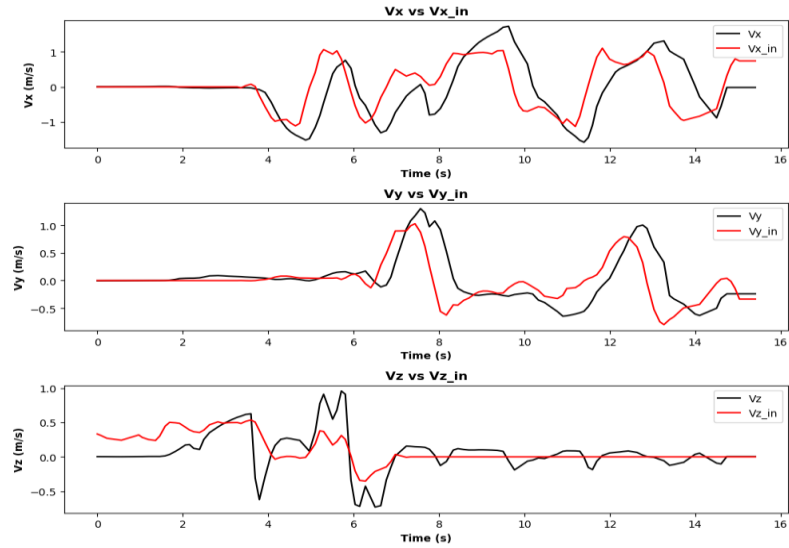


Figure 9 - Plot showing user input velocities and drone velocities in real time

To evaluate the system's responsiveness and precision, the velocities (v_x , v_y , and v_z) of the drone were plotted against the corresponding user inputs from the VR controllers. These plots (figure 8) provide a visual representation of how the drone in the Gazebo simulation follows instructions given by user. The plotted graphs show that the velocities of the drone (v_x , v_y , and v_z) closely follow the velocity commands given by the user through the VR controllers. A lag of approx. 0.13 sec has been seen in between the velocity of drone and velocity input given by user which is negligible.

It has been observed from the plots that there is a lag of 0.13 sec in between the drone response and input from the user. As the system running Unity and a system in which simulation is running are connected through router. The commands sent from Unity side pass through router and then received on ROS side. This causes a delay because of the lag in wireless communication since the network is operating at 2.4 GHz with a data transfer speed of 400Mbps. Communication can further be improved, and the lag can be decreased with the use of router with more bandwidth capability and also with the use of better communication protocol.

5.3 User Feedback on VR-Based Drone Control System:

To evaluate the effectiveness and user experience of the VR-based drone control system, feedback was collected from 15 users. Each user was asked to provide their responses based on a 5-point Likert scale, where 1 represents "Strongly Agree," 2 represents "Agree," 3 represents "Neutral," 4 represents "Disagree," and 5 represents "Strongly Disagree."

The feedback focused on key aspects such as Ease of Use, Immersion, Learning Curve, Confidence, Mental Demand, and Physical Demand. Two sets of feedback were collected—one during simulation and the other during real-time drone control.

- **Simulation Feedback:** The responses as shown in figure 10 indicated a generally positive experience in the simulated environment, with most users agreeing that the system was easy to use and provided a high level of immersion. The majority of users felt confident operating the drone in simulation, with lower reported physical and mental demand.
- **Real-time Feedback:** In real-world scenarios, users still reported good levels of confidence and immersion but noted a slight increase in physical and mental demand. Despite this, the overall user experience remained positive, suggesting that the system effectively transitions from simulation to real-world applications. The feedback is shown in figure 11 below.

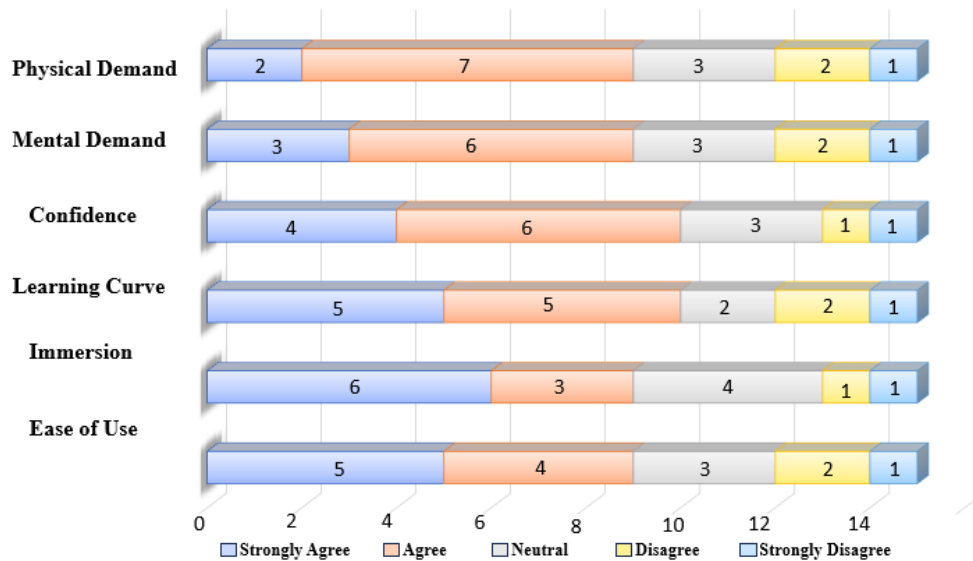


Figure 10 - Feedback of user interacting with simulation.

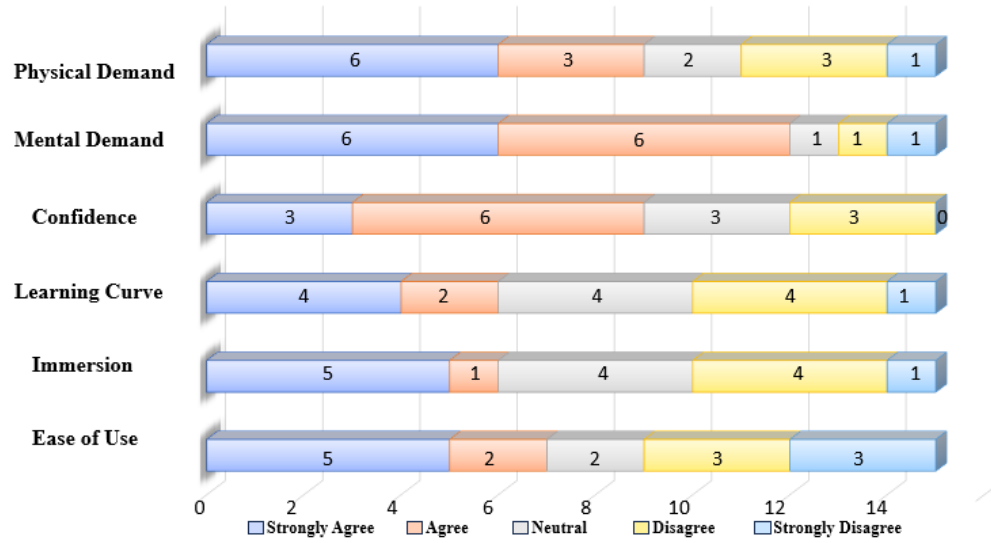


Figure 11 - Feedback of user interacting with real-time.

CHAPTER 6: CONCLUSION

This study explored the use of Virtual Reality (VR) technology in combination with unmanned aerial vehicles (UAVs) for drone control and navigation, utilizing platforms such as Unity and the Robot Operating System (ROS). By integrating the Oculus Quest 2 VR headset and its controllers, the research demonstrated an enhanced user experience, providing a more immersive and intuitive interface for operating drones in both simulated and real-world environments.

The system was implemented using Unity to process VR controller inputs and ROS for drone communication and control, with a PX4-autopilot-based drone as the primary UAV platform. The ROS-TCP-CONNECTOR on the Unity side and ROS-TCP-ENDPOINT on the ROS side facilitated seamless communication between the platforms, enabling effective control in both simulation and real-time scenarios.

A comparative analysis between the Oculus Quest 2 and traditional controllers, such as the FLYSKY FSi6x, revealed that the Oculus Quest 2 offers superior control accuracy and user interaction, making it more suitable for drone piloting. This system simplifies complex maneuvers, making it easier for new pilots to learn and reducing the risk of operational errors.

Further evaluation of system performance in real-time environments using the PX4-autopilot-based drone showed a minimal lag of approximately 0.13 seconds between user input and drone response, which is negligible for effective control.

While this thesis has laid a strong foundation for VR-based drone control, several areas offer opportunities for further research and development. Future work could explore the following directions:

- **Multi-Drone Coordination:**

Future research could focus on developing a VR interface for controlling and coordinating multiple drones in a swarm. This could involve creating a more sophisticated user interface

in Unity that allows operators to manage drone formations, assign tasks, and monitor the status of each drone in real-time. The ability to control a swarm of drones using VR could revolutionize applications such as precision agriculture, environmental monitoring, and military operations.

- **Haptic Feedback Integration:**

Incorporating haptic feedback into the VR system could further enhance the operator's immersion and control precision. Haptic devices could provide tactile feedback when the drone encounters obstacles or when specific actions are performed, such as landing or object manipulation. This would give operators a more tangible sense of the drone's interaction with its environment, potentially improving the accuracy and safety of operations.

- **Advanced Simulation and Training Environments:**

Developing more advanced simulation environments in Unity and Gazebo could improve training for drone operators. These environments could include dynamic weather conditions, complex terrains, and simulated emergencies, offering a more comprehensive training experience. Additionally, integrating VR with machine learning could enable the system to adapt to the operator's skill level, providing customized training scenarios.

- **Cross-Platform and Device Compatibility:**

Future work could explore the use of different VR devices and platforms, comparing their effectiveness in various control scenarios. Expanding compatibility to include other VR headsets and controllers could make the system more accessible to a broader audience. Additionally, research could investigate the potential of augmented reality (AR) in drone control, where operators could interact with a combination of real-world and virtual elements.

- **Integration with Augmented Reality (AR):**

Exploring the integration of AR with VR for drone control could offer a hybrid approach where operators can see both the real world and virtual overlays, enhancing situational awareness. This would be particularly useful in scenarios where the operator needs to

interact with both real and simulated elements simultaneously, such as in industrial inspections or rescue operations.

- **Real-World Deployment and Case Studies:**

Finally, deploying the VR-based drone control system in real-world scenarios and conducting case studies would provide valuable insights into its practical applications and limitations. Collaborating with industries such as agriculture, logistics, or emergency services could help refine the system and demonstrate its effectiveness in various operational contexts.

APPENDIX A

7.1 Unity Side

The following code is a Unity script designed to capture user input from an Oculus VR controller and publish these inputs to a ROS (Robot Operating System) environment.

```
using UnityEngine;

using Unity.Robotics.ROSTCPConnector;

using RosMessageTypes.Std;

using UnityEngine.XR;

public class OculusDroneController : MonoBehaviour
{
    private ROSConnection ros;

    private string topicName = "button_press";

    private string thumbstickTopicName = "thumbstick_input";

    private bool buttonAPressed = false;

    private bool buttonBPressed = false;

    private Vector2 rightThumbstick;

    private Vector2 leftThumbstick;
```

```

private bool buttonAState = false;

private bool buttonBState = false;

private const float thumbstickThreshold = 0.1f; // Adjust this threshold as needed

void Start()

{

    ros = ROSConnection.instance;

    ros.RegisterPublisher<StringMsg>(topicName);

    ros.RegisterPublisher<StringMsg>(thumbstickTopicName);

}

void Update()

{

    var rightHandDevice = InputDevices.GetDeviceAtXRNode(XRNode.RightHand);

    var leftHandDevice = InputDevices.GetDeviceAtXRNode(XRNode.LeftHand);

    if (rightHandDevice.isValid)

    {

        rightHandDevice.TryGetFeatureValue(CommonUsages.primaryButton,    out
buttonAPressed);

        rightHandDevice.TryGetFeatureValue(CommonUsages.secondaryButton,    out
buttonBPressed);

```

```

        rightHandDevice.TryGetFeatureValue(CommonUsages.primary2DAxis,    out
rightThumbstick);

    if (buttonAPressed && !buttonAState)
    {
        StringMsg buttonAMsg = new StringMsg("A");

        ros.Publish(topicName, buttonAMsg);

        Debug.Log("Button A Pressed");

        buttonAState = true;
    }
    else if (!buttonAPressed)
    {
        buttonAState = false;
    }
    if (buttonBPressed && !buttonBState)
    {
        StringMsg buttonBMsg = new StringMsg("B");

        ros.Publish(topicName, buttonBMsg);

        Debug.Log("Button B Pressed");

        buttonBState = true;
    }
}

```

```

else if (!buttonBPressed)

{

    buttonBState = false;

}

}

if (leftHandDevice.isValid)

{

    leftHandDevice.TryGetFeatureValue(CommonUsages.primary2DAxis, out
leftThumbstick);

}

if (rightThumbstick.magnitude > thumbstickThreshold ||
leftThumbstick.magnitude > thumbstickThreshold)

{

    StringMsg thumbstickMsg = new StringMsg($"thumbsticks {leftThumbstick.x}
{leftThumbstick.y} {rightThumbstick.x} {rightThumbstick.y}");

    ros.Publish(thumbstickTopicName, thumbstickMsg);

    Debug.Log($"Thumbsticks - Left: ({leftThumbstick.x}, {leftThumbstick.y}),
Right: ({rightThumbstick.x}, {rightThumbstick.y}");

}

}

}

```

The following code is a Unity script designed to capture image feed coming from ROS (Robot Operating System) environment.

```
using UnityEngine;

using UnityEngine.UI;

using Unity.Robotics.ROSTCPConnector;

using RosMessageTypes.Sensor;

using System.Linq;

public class ImageSubscriber : MonoBehaviour

{

    public RawImage rawImage;

    private ROSConnection ros;

    private Texture2D texture;

    void Start()

    {

        ros = ROSConnection.instance;

        ros.Subscribe<CompressedImageMsg>("main_camera/image_raw/compressed",

DisplayImage);

    }

    void DisplayImage(CompressedImageMsg compressedImageMessage)
```

```
{

    // Decode the compressed image data

    byte[] imageData = compressedImageMessage.data.ToArray();

    // Load the image data into a texture

    if (texture == null)

    {

        texture = new Texture2D(2, 2); // Texture will be resized automatically by
LoadImage

    }

    if (texture.LoadImage(imageData)) // LoadImage returns true if loading is
successful

    {

        // Set the texture to the RawImage component

        rawImage.texture = texture;

    }

    else

    {

        Debug.LogError("Failed to load image data.");

    }

}
```

```
}
```

7.2 ROS Side

The provided ROS launch file is used to configure and start the ROS-TCP-ENDPOINT, which facilitates communication between Unity and the ROS environment.

```
<launch>

  <arg name="tcp_ip" default="192.168.0.106"/>

  <arg name="tcp_port" default="10000"/>

  <node          name="unity_endpoint"          pkg="ros_tcp_endpoint"
type="default_server_endpoint.py" output="screen">

    <param name="tcp_ip" type="string" value="$(arg tcp_ip)"/>

    <param name="tcp_port" type="int" value="$(arg tcp_port)"/>

  </node>

</launch>
```

The provided launch file is used to set up a simulation environment for a drone using Gazebo, which are popular simulators in the ROS ecosystem.

```
<launch>

  <arg name="type" default="gazebo"/> <!-- gazebo, jmavsim, none (only clover
packages) -->
```

```

<arg name="mav_id" default="0"/>

<arg name="est" default="ekf2"/> <!-- PX4 estimator: lpe, ekf2 -->

<arg name="vehicle" default="clover"/> <!-- PX4 vehicle configuration: clover,
clover_vpe -->

<arg name="main_camera" default="true"/> <!-- Simulated vision position
estimation camera (optical flow, ArUco) -->

<arg name="maintain_camera_rate" default="false"/> <!-- Slow simulation down to
maintain camera rate -->

<arg name="rangefinder" default="true"/> <!-- Simulated downward-facing
rangefinder, vl5311x-like -->

<arg name="led" default="true"/> <!-- Simulated LED strip, ws281x-like -->

<arg name="gps" default="false"/> <!-- Simulated GPS module -->

<arg name="use_clover_physics" default="false"/> <!-- Use inertial parameters from
CAD models -->

<arg name="gui" default="true"/> <!-- Run Gazebo with GUI -->

<!-- Gazebo instance -->

<include file="$(find gazebo_ros)/launch/empty_world.launch" if="$(eval type ==
'gazebo')">

  <!-- Workaround for crashes in VMware -->

  <env name="SVGA_VGPU10" value="0"/>

  <arg name="gui" value="$(arg gui)"/>

```



```

    <arg name="world_name" value="$(find
clover_simulation)/resources/worlds/clover_aruco.world"/>

    <arg name="verbose" value="true"/>

</include>

<!-- PX4 instance -->

<node name="sitl_$(arg mav_id)" pkg="px4" type="px4" output="screen"
required="true" args="$(find px4)/ROMFS/px4fmu_common -s etc/init.d-posix/rcS -i
$(arg mav_id)" unless="$(eval type == 'none')">

    <env name="PX4_SIM_MODEL" value="$(arg vehicle)"/>

    <env name="PX4_ESTIMATOR" value="$(arg est)"/>

</node>

<!-- Clover model -->

<include file="$(find clover_description)/launch/spawn_drone.launch" if="$(eval
type == 'gazebo')">

    <arg name="main_camera" value="$(arg main_camera)"/>

    <arg name="maintain_camera_rate" value="$(arg maintain_camera_rate)"/>

    <arg name="rangefinder" value="$(arg rangefinder)"/>

    <arg name="led" value="$(arg led)"/>

    <arg name="gps" value="$(arg gps)"/>

    <arg name="use_clover_physics" value="$(arg use_clover_physics)"/>

</include>

```

```

<node name="jmavsim" pkg="px4" required="true" type="jmavsim_run.sh"
output="screen" if="$(eval type == 'jmavsim')">

  <env name="HEADLESS" value="1" if="$(eval not gui)"/>

</node>

<param name="use_sim_time" value="false" if="$(eval type == 'jmavsim')"/>

<!-- Clover services -->

<include file="$(find clover)/launch/clover.launch">

  <arg name="simulator" value="true"/>

  <arg name="fcu_conn" value="sitl"/>

  <arg name="fcu_ip" value="127.0.0.1"/>

  <arg name="gcs_bridge" value=""/>

  <arg name="web_video_server" default="false" if="$(eval type == 'jmavsim')"/>

  <arg name="main_camera" default="false" if="$(eval type == 'jmavsim')"/>

  <arg name="aruco" default="false" if="$(eval type == 'jmavsim')"/>

  <arg name="optical_flow" default="false" if="$(eval type == 'jmavsim')"/>

  <arg name="led" default="false" if="$(eval type == 'jmavsim')"/>

</include>

</launch>

```

The below provided launch file is for setting up and configuring the Clover drone, whether in simulation or real-world scenarios.

```
<launch>

  <arg name="fcu_conn" default="usb"/>

  <arg name="fcu_ip" default="127.0.0.1"/>

  <arg name="fcu_sys_id" default="1"/>

  <arg name="gcs_bridge" default="tcp"/>

  <arg name="web_video_server" default="true"/>

  <arg name="rosbridge" default="true"/>

  <arg name="main_camera" default="true"/>

  <arg name="optical_flow" default="true"/>

  <arg name="aruco" default="false"/>

  <arg name="rangefinder_vl5311x" default="true"/>

  <arg name="led" default="true"/>

  <arg name="blocks" default="false"/>

  <arg name="rc" default="false"/>

  <arg name="force_init" default="true"/> <!-- force estimator to init by publishing
zero pose -->

  <arg name="simulator" default="false"/> <!-- flag that we are operating on a
simulated drone -->
```

```

<!-- log formatting -->

<env name="ROSCONSOLE_FORMAT" value="[${severity}] [${time}]:
${logger}: ${message}"/>

<!-- mavros -->

<include file="$(find clover)/launch/mavros.launch">

  <arg name="fcu_conn" value="$(arg fcu_conn)"/>

  <arg name="fcu_ip" value="$(arg fcu_ip)"/>

  <arg name="fcu_sys_id" value="$(arg fcu_sys_id)"/>

  <arg name="gcs_bridge" value="$(arg gcs_bridge)"/>

</include>

<!-- web video server -->

<node name="web_video_server" pkg="web_video_server"
type="web_video_server" if="$(arg web_video_server)" required="false"
respawn="true" respawn_delay="5">

  <param name="default_stream_type" value="ros_compressed"/>

  <param name="publish_rate" value="1.0"/>

</node>

<!-- aruco markers -->

<include file="$(find clover)/launch/aruco.launch" if="$(eval aruco or force_init)">

  <arg name="force_init" value="$(arg force_init)"/>

```

```

    <arg name="disable" value="\$(eval not aruco)"/>

</include>

<!-- optical flow -->

<node pkg="nodelet" type="nodelet" name="optical_flow" args="load
clover/optical_flow main_camera_nodelet_manager" if="\$(arg optical_flow)"
clear_params="true" output="screen" respawn="true">

    <remap from="image_raw" to="main_camera/image_raw"/>

    <remap from="camera_info" to="main_camera/camera_info"/>

    <param name="calc_flow_gyro" value="true"/>

    <param name="roi_rad" value="0.8"/>

    <param name="disable_on_vpe" value="true"/>

</node>

<!-- simplified offboard control -->

<node name="simple_offboard" pkg="clover" type="simple_offboard"
output="screen" clear_params="true">

    <param name="reference_frames/main_camera_optical" value="map"/>

    <param name="terrain_frame_mode" value="range"/>

</node>

<!-- main camera -->

<include file="\$(find clover)/launch/main_camera.launch" if="\$(arg
main_camera)">

```

```

    <arg name="simulator" value="$(arg simulator)"/>

</include>

<!-- rosbridge -->

<include file="$(find rosbridge_server)/launch/rosbridge_websocket.launch"
if="$(eval rosbridge or rc)"/>

<!-- tf2 republisher for web visualization -->

<node name="tf2_web_republisher" pkg="tf2_web_republisher"
type="tf2_web_republisher" output="screen" if="$(arg rosbridge)"/>

<!-- v15311x ToF rangefinder -->

<node name="rangefinder" pkg="v15311x" type="v15311x_node" output="screen"
if="$(eval rangefinder_v15311x and not simulator)"/>

    <param name="frame_id" value="rangefinder"/>

    <param name="min_signal" value="0.4"/>

    <param name="pass_statuses" type="yaml" value="[0, 6, 7, 11]"/>

</node>

<!-- rangefinder's frame -->

<node pkg="tf2_ros" type="static_transform_publisher" name="rangefinder_frame"
args="0 0 -0.05 0 1.5707963268 0 base_link rangefinder" if="$(arg
rangefinder_v15311x)"/>

<!-- led strip -->

<include file="$(find clover)/launch/led.launch" if="$(arg led)"/>

```

```

    <arg name="simulator" value="$(arg simulator)"/>

</include>

<!-- Clover Blocks -->

<node name="clover_blocks" pkg="clover_blocks" type="clover_blocks"
output="screen" if="$(arg blocks)"/>

<!-- rc backend -->

<node name="rc" pkg="clover" type="rc" output="screen" if="$(arg rc)"
clear_params="true">

    <!-- Send fake GCS heartbeats. Set to "true" for upstream PX4 -->

    <param name="use_fake_gcs" value="false"/>

</node>

</launch>

```

The provided launch file is responsible for setting up and configuring the MAVROS node, which acts as a bridge between the Robot Operating System (ROS) and the flight control unit (FCU) of a drone.

```

<launch>

    <arg name="fcu_conn" default="usb"/> <!-- options: usb, uart, tcp, udp, sitl, hitl -->

    <arg name="fcu_ip" default="127.0.0.1"/>

    <arg name="fcu_sys_id" default="1"/>

    <arg name="gcs_bridge" default="tcp"/>

```

```

<arg name="viz" default="true"/>

<arg name="respawn" default="true"/>

<arg name="distance_sensor_remap" default="rangefinder/range"/>

<arg name="usb_device" default="/dev/px4fmu"/>

<arg name="prefix" default="" unless="$(eval fcu_conn == 'usb')"/>

<arg name="prefix" default="roslaunch clover waitfile $(arg usb_device)" if="$(eval fcu_conn == 'usb')"/>

<node pkg="mavros" type="mavros_node" name="mavros" launch-prefix="$(arg prefix)" required="false" clear_params="true" respawn="$(arg respawn)" unless="$(eval fcu_conn == 'none')" respawn_delay="1" output="screen">

  <!-- UART connection -->

  <param name="fcu_url" value="/dev/ttyAMA0:921600" if="$(eval fcu_conn is None or fcu_conn == 'uart')"/>

  <!-- USB connection -->

  <param name="fcu_url" value="$(arg usb_device)" if="$(eval fcu_conn == 'usb')"/>

  <!-- sitl before PX4 1.9.0 -->

  <param name="fcu_url" value="udp://@$(arg fcu_ip):14557" if="$(eval fcu_conn == 'udp')"/>

  <!-- sitl since PX4 1.9.0 -->

  <param name="fcu_url" value="udp://@$(arg fcu_ip):14580" if="$(eval fcu_conn == 'sitl')"/>

```



```

<!-- hitl connection (to gazebo_mavlink_interface plugin) -->

<param name="fcu_url" value="udp://$(arg fcu_ip):14540@" if="$(eval fcu_conn
== 'hitl')"/>

<!-- set target_system_id -->

<param name="target_system_id" value="$(arg fcu_sys_id)" />

<!-- gcs bridge -->

<param name="gcs_url" value="tcp-l://0.0.0.0:5760" if="$(eval gcs_bridge ==
'tcp')"/>

<param name="gcs_url" value="udp://0.0.0.0:14550@14550" if="$(eval
gcs_bridge == 'udp')"/>

<param name="gcs_url" value="udp-b://$(env
ROS_HOSTNAME):14550@14550" if="$(eval gcs_bridge == 'udp-b')"/>

<param name="gcs_url" value="udp-pb://$(env
ROS_HOSTNAME):14550@14550" if="$(eval gcs_bridge == 'udp-pb')"/>

<param name="gcs_url" value="" if="$(eval not gcs_bridge)"/>

<param name="gcs_quiet_mode" value="true"/>

<param name="conn/timeout" value="8"/>

<!-- basic params -->

<roscpp command="load" file="$(find clover)/launch/mavros_config.yaml"/>

<!-- remap rangefinder -->

```

```
<remap      from="mavros/distance_sensor/rangefinder_sub"      to="$(arg
distance_sensor_remap)" if="$(eval bool(distance_sensor_remap))"/>
```

```
<roscparam param="plugin_whitelist">
```

```
- altitude
```

```
- command
```

```
- distance_sensor
```

```
- ftp
```

```
- global_position
```

```
- imu
```

```
- local_position
```

```
- manual_control
```

```
# - mocap_pose_estimate
```

```
- param
```

```
- px4flow
```

```
- rc_io
```

```
- setpoint_attitude
```

```
- setpoint_position
```

```
- setpoint_raw
```

```
- setpoint_velocity
```

```
- sys_status
```

```

- sys_time

- vision_pose_estimate

# - vision_speed_estimate

# - waypoint

</rosparam>

</node>

<!-- remapped distance_sensor config -->

<rosparam      param="$(arg      distance_sensor_remap)"      if="$(eval
bool(distance_sensor_remap))">

  subscriber: true

  id: 1

  orientation: PITCH_270

  covariance: 1 # cm

</rosparam>

<!-- Copter visualization -->

<node name="visualization" pkg="mavros_extras" type="visualization" if="$(arg
viz)">

  <remap to="mavros/local_position/pose" from="local_position"/>

  <remap to="mavros/setpoint_position/local" from="local_setpoint"/>

  <param name="fixed_frame_id" value="map"/>

```

```
<param name="child_frame_id" value="base_link"/>

<param name="marker_scale" value="1"/>

<param name="max_track_size" value="20"/>

<param name="num_rotors" value="4"/>

</node>

</launch>
```

The provided Python script is a ROS (Robot Operating System) node that interfaces with a Clover drone to control its movement based on input from an Oculus VR controller.

```
import rospy

from clover import srv

from std_srvs.srv import Trigger

from std_msgs.msg import String

import matplotlib.pyplot as plt

import time

# Initialize the ROS node

rospy.init_node('flight')
```

```

# Define service proxies to communicate with the drone's services

get_telemetry = rospy.ServiceProxy('get_telemetry', srv.GetTelemetry)

navigate = rospy.ServiceProxy('navigate', srv.Navigate)

land = rospy.ServiceProxy('land', Trigger)

set_velocity = rospy.ServiceProxy('set_velocity', srv.SetVelocity)

# Initialize velocity variables

vx = 0.0

vy = 0.0

vz = 0.0

speed = 0.5

last_command_time = time.time()

command_cooldown = 0.1 # 100 ms cooldown to prevent rapid command execution

# Lists to store telemetry and controller input data

time_data = []

vx_data = []

vy_data = []

```

```
vz_data = []

vx_input_data = []

vy_input_data = []

vz_input_data = []

def log_data():

    telemetry = get_telemetry()

    current_time = time.time() - flight_start_time

    # Log the telemetry data

    time_data.append(current_time)

    vx_data.append(telemetry.vx)

    vy_data.append(telemetry.vy)

    vz_data.append(telemetry.vz)

    # Log the controller inputs

    vx_input_data.append(vx)

    vy_input_data.append(vy)
```

```
    vz_input_data.append(vz)

def button_press_callback(data):

    global vx, vy, vz

    if data.data == "A":

        rospy.loginfo("Button A pressed: Take off and hover")

        navigate(frame_id='body', auto_arm=True)

        vx = 0.0

        vy = 0.0

        vz = 0.0

    elif data.data == "B":

        rospy.loginfo("Button B pressed: Perform landing")

        land()

def thumbstick_callback(data):

    global vx, vy, vz, last_command_time

    if time.time() - last_command_time < command_cooldown:

        return
```

```
last_command_time = time.time()

input_data = data.data.split()

left_x_val = float(input_data[1])

left_y_val = float(input_data[2])

right_x_val = float(input_data[3])

right_y_val = float(input_data[4])

vx = left_y_val * 2 # Adjust the factor as needed for sensitivity

vy = left_x_val * 2 # Adjust the factor as needed for sensitivity

vz = right_y_val # Adjust the factor as needed for sensitivity

rospy.loginfo("Thumbsticks moved: Adjusting vx by {}, vy by {}, vz by
{}".format(vx, vy, vz))

set_velocity(vx=vx, vy=vy, vz=vz, yaw=0.0, frame_id='body')

# Log the data each time the thumbstick is moved

log_data()
```



```
# Subscribe to the topics published by Unity for Oculus controller button presses and
thumbstick inputs

rospy.Subscriber('button_press', String, button_press_callback)

rospy.Subscriber('thumbstick_input', String, thumbstick_callback)

# Record the start time of the flight

flight_start_time = time.time()

# Keep the node running and waiting for inputs

rospy.spin()

# After the flight, plot the data

plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)

plt.plot(time_data, vx_data, label='vx (m/s)')

plt.plot(time_data, vx_input_data, label='vx input', linestyle='--')

plt.xlabel('Time (s)')

plt.ylabel('vx (m/s)')

plt.title('Velocity in x direction and controller input')
```

```
plt.legend()

plt.subplot(3, 1, 2)

plt.plot(time_data, vy_data, label='vy (m/s)')

plt.plot(time_data, vy_input_data, label='vy input', linestyle='--')

plt.xlabel('Time (s)')

plt.ylabel('vy (m/s)')

plt.title('Velocity in y direction and controller input')

plt.legend()

plt.subplot(3, 1, 3)

plt.plot(time_data, vz_data, label='vz (m/s)')

plt.plot(time_data, vz_input_data, label='vz input', linestyle='--')

plt.xlabel('Time (s)')

plt.ylabel('vz (m/s)')

plt.title('Velocity in z direction and controller input')

plt.legend()

plt.tight_layout()

plt.show()
```

REFERENCES

- [1]. Mohsan, S.A.H., Othman, N.Q.H., Li, Y. *et al.* Unmanned aerial vehicles (UAVs): Mohsan, S.A.H., Othman, N.Q.H., Li, Y. *et al.* Unmanned aerial vehicles (UAVs): practical aspects, applications, open challenges, security issues, and future trends. *Intel Serv Robotics* **16**, 109–137 (2023). <https://doi.org/10.1007/s11370-022-00452-4>
- [2]. Carnevale, A., Mannocchi, I., Sassi, M. S. H., Carli, M., De Luca, G., Longo, U. G., Schena, E. (2022). Virtual Reality for Shoulder Rehabilitation: Accuracy Evaluation of Oculus Quest 2. *Sensors (Basel)*, 22(15). doi:10.3390/s22155511.
- [3]. <https://aeronautics-sys.com/exploring-the-latest-drone-technology-innovations-in-uas/>
- [4]. Grand View Research. (2021). *Educational Robots Market Size, Share & Trends Analysis Report By Product (Pre-programmed Robots, Humanoid Robots), By Application (Primary Education, Secondary Education), By Region, And Segment Forecasts, 2020 - 2027.*
- [5]. IFR (International Federation of Robotics). (2021). *World Robotics Report 2021: Industrial Robots.*
- [6]. Grand View Research. (2021). *Surgical Robots Market Size, Share & Trends Analysis Report By Product, By Application, By Region, And Segment Forecasts, 2020 - 2027.*
- [7]. JAMA Network Open. (2019). *Study on the impact of VR surgical simulations on surgical performance.*
- [8]. Grand View Research. (2021). *Telepresence Robots Market Size, Share & Trends Analysis Report By Component, By Application, By End Use, By Region, And Segment Forecasts, 2020 - 2027.*
- [9]. Fortune Business Insights. (2021). *Virtual Reality in Gaming Market Size, Share & Trends Analysis Report By Component, By Device, By Application, By Region, And Segment Forecasts, 2021 - 2028.*
- [10]. Statista. (2020). *Consumer Robotics Market Report.*

- [11]. Berge, L.-P., Aouf, N., Duval, T., & Coppin, G. (2016). *Generation and VR visualization of 3D point clouds for drone target validation assisted by an operator*. Paper presented at the 2016 8th Computer Science and Electronic Engineering (CEECE).
- [12]. Carnevale, A., Mannocchi, I., Sassi, M. S. H., Carli, M., De Luca, G., Longo, U. G., Schena, E. (2022). Virtual Reality for Shoulder Rehabilitation: Accuracy Evaluation of Oculus Quest 2. *Sensors (Basel)*, 22(15). doi:10.3390/s22155511
- [13]. Covaciu, F., & Iordan, A. E. (2022). Control of a Drone in Virtual Reality Using MEMS Sensor Technology and Machine Learning. *Micromachines (Basel)*, 13(4). doi:10.3390/mi13040521
- [14]. Dardona, T., Eslamian, S., Reisner, L. A., & Pandya, A. (2019). Remote Presence: Development and Usability Evaluation of a Head-Mounted Display for Camera Control on the da Vinci Surgical System. *Robotics*, 8(2). doi:10.3390/robotics8020031
- [15]. Isleyen, E., & Duzgun, H. S. (2019). Use of virtual reality in underground roof fall hazard assessment and risk mitigation. *International Journal of Mining Science and Technology*, 29(4), 603-607. doi:10.1016/j.ijmst.2019.06.003
- [16]. Kim, S., Lee, S., Kang, H., Kim, S., & Ahn, M. (2021). P300 Brain-Computer Interface-Based Drone Control in Virtual and Augmented Reality. *Sensors (Basel)*, 21(17). doi:10.3390/s21175765
- [17]. Kot, T., & Novák, P. (2018). Application of virtual reality in teleoperation of the military mobile robotic system TAROS. *International Journal of Advanced Robotic Systems*, 15(1). doi:10.1177/1729881417751545
- [18]. H. Martins and R. Ventura, "Immersive 3-D teleoperation of a search and rescue robot using a head-mounted display," *2009 IEEE Conference on Emerging Technologies & Factory Automation*, Palma de Mallorca, Spain, 2009, pp. 1-8, doi: 10.1109/ETFA.2009.5347014.
- [19]. Li, Y., Karim, M. M., & Qin, R. (2022). A Virtual-Reality-Based Training and Assessment System for Bridge Inspectors With an Assistant Drone. *IEEE Transactions on Human-Machine Systems*, 52(4), 591-601. doi:10.1109/thms.2022.3155373

- [20]. Ramírez, G., Verdín, R., & Flores, G. (2024). *Real-Time Mapping for Teleoperation Systems in VR of Unmanned Aerial Vehicles*. Paper presented at the 2024 International Conference on Unmanned Aircraft Systems (ICUAS).
- [21]. Watanabe, K., & Takahashi, M. (2019). Head-synced Drone Control for Reducing Virtual Reality Sickness. *Journal of Intelligent & Robotic Systems*, 97(3-4), 733-744. doi:10.1007/s10846-019-01054-6
- [22]. I. Rodriguez, A. Astigarraga, E. Jauregi, T. Ruiz and E. Lazkano, "Humanizing NAO robot teleoperation using ROS," 2014 IEEE-RAS International Conference on Humanoid Robots, Madrid, Spain, 2014, pp. 179-186, doi: 10.1109/HUMANOIDS.2014.7041357.
- [23]. Hetrick, R., et al., *Comparing Virtual Reality Interfaces for the Teleoperation of Robots*, in 2020 Systems and Information Engineering Design Symposium (SIEDS). 2020. p. 1-7.
- [24]. Sun, D., et al., *A New Mixed-Reality-Based Teleoperation System for Telepresence and Maneuverability Enhancement*. IEEE Transactions on Human-Machine Systems, 2020. **50**(1): p. 55-67.
- [25]. Naceri, A., et al., *The Vicarios Virtual Reality Interface for Remote Robotic Teleoperation*. Journal of Intelligent & Robotic Systems, 2021. **101**(4).
- [26]. Wang, Q., et al., *Virtual reality human-robot collaborative welding: A case study of weaving gas tungsten arc welding*. Journal of Manufacturing Processes, 2019. **48**: p. 210-217.
- [27]. Alonso, R., et al., *Exploiting virtual reality and the robot operating system to remote-control a humanoid robot*. Multimedia Tools and Applications, 2022. **81**(11): p. 15565-15592.
- [28]. Zhang, P., Carey, J., Te'eni, D., & Tremaine, M. (2005). Integrating Human-Computer Interaction Development into the Systems Development Life Cycle: A Methodology. Communications of the Association for Information Systems, 15, pp-pp. <https://doi.org/10.17705/1CAIS.01529>