

Yuri Winche Achermann

**Development of a Framework to Enable Human-Drone Interaction Through
Natural Gesture Control using Systems Engineering Methodology**

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Aeroespacial do Centro Tecnológico de Joinville da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Aeroespacial.
Supervisor:: Prof. Dr. Ricardo José Pfitscher

Joinville
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Achermann, Yuri Winche

A Framework to Enable Human-Drone Interaction Through
Natural Gesture Control / Yuri Winche Achermann ;
orientador, Ricardo José Pfitscher, 2024.

79 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Campus Joinville,
Graduação em Engenharia Aeroespacial, Joinville, 2024.

Inclui referências.

1. Engenharia Aeroespacial. 2. Drone Control Methods.
3. HCI. 4. Hand Gesture Recognition. 5. SITL Simulation.
I. Pfitscher, Ricardo José. II. Universidade Federal de
Santa Catarina. Graduação em Engenharia Aeroespacial. III.
Título.

Yuri Winche Achermann

**Development of a Framework to Enable Human-Drone Interaction Through
Natural Gesture Control using Systems Engineering Methodology**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Engenharia Aeroespacial” e aprovado em sua forma final pelo Curso de Graduação em Engenharia Aeroespacial.

Joinville, 12 de dezembro de 2024.

Banca Examinadora:

Prof. Dr. Ricardo José Pfitscher
Supervisor:

Prof. Benjamin Grando Moreira, Dr.
Avaliador
Universidade Federal de Santa Catarina

Prof. Dalton Luiz Rech Vidor, Dr.
Avaliador
Universidade Federal de Santa Catarina

Leandro Artur Custodio
Avaliador
Universidade Federal de Santa Catarina

ABSTRACT

Recent advances in drone technology and human-computer interaction have created new possibilities for controlling unmanned aerial vehicles (UAVs). Traditional drone control methods often require both hands and complex remote controllers, which can limit accessibility for users with physical limitations. This manuscript presents the development of a framework that enables drone control through natural hand gestures, using a Leap Motion Controller for gesture recognition and the PX4 flight stack for drone operation. Current drone control interfaces rely heavily on two-handed operation, creating barriers for users with limited mobility or those requiring simpler control methods. The challenge lies in creating an intuitive, reliable system that can accurately translate hand movements into precise drone commands. The development of gesture-based control systems for drones can increase accessibility and simplify human-drone interaction, potentially benefiting various fields such as search and rescue operations, industrial inspection, and recreational use. This research aims to document the decision processes in developing and validating a framework for gesture-based drone control that enables single-handed operation in a Software-In-The-Loop environment. The study follows an adapted version of NASA Systems Engineering Handbook procedures, focusing on requirements definition and preliminary design phases. The framework integrates the Leap Motion Controller for gesture capture, ROS 2 for system communication, and Gazebo for simulation validation. The development process includes systematic testing of gesture recognition accuracy and control precision. The resulting framework demonstrates the feasibility of controlling drones through natural hand gestures, offering an alternative to traditional control methods. Initial simulation results indicate that the system can effectively translate hand movements into drone commands while maintaining flight stability and control precision. This research contributes to the field of human-drone interaction by providing a documented approach to implementing gesture-based control systems, laying groundwork for future developments in accessible drone interfaces.

Keywords: Drone Control Methods. HCI. Hand Gesture Recognition. SITL Simulation.

LIST OF FIGURES

Figure 1 – Radio Transmitter Frsky Taranis Q X7 model.	20
Figure 2 – Myo Armband for muscle activity detection.	21
Figure 3 – Body gesture positions for drone control.	22
Figure 4 – Hand gesture mapping for flight commands.	23
Figure 5 – Multicopter Angular Rate Controller.	25
Figure 6 – Multicopter Control Architecture.	25
Figure 7 – Detailed overview of the building blocks and data flow of PX4.	27
Figure 8 – Holybro Pixhawk 6X model.	28
Figure 9 – How ROS 2 control interface modes interact with PX4.	29
Figure 10 – Communication schema for XRCE-DDS Agent.	30
Figure 11 – Gazebo in designed scenarios.	32
Figure 12 – Ultraleap Camera Position Visualizer Controls.	34
Figure 13 – Related Work analysis through multiple charts.	36
Figure 14 – Operational interaction flow defined in five steps.	40
Figure 15 – System Architecture Diagram.	44
Figure 16 – Hand Gestures Mapped to Drone Commands.	53
Figure 17 – Architecture uXRCE-DDS with ROS 2	55
Figure 18 – SITL communication environment using MAVLink	56
Figure 19 – Neutral hand position in the tracking software.	59
Figure 20 – Linux terminal with ROS command to start execution.	59
Figure 21 – Compilation of all the windows that open with SITL launch command.	60
Figure 22 – Drone path defined to assert navigability and sensitiveness.	60
Figure 23 – Drone Hardware Prototype.	61
Figure 24 – Gesture Controlled Drone System Stakeholders Hub-and-Spoke Map	77
Figure 25 – Gesture Controlled Drone System Stakeholders SVN Map	77

LIST OF TABLES

Table 2 – Multiple-Domain Matrix Function Correlations Table.	46
Table 3 – Morphological Matrix for Drone Software stacks.	47
Table 4 – Morphological Matrix for Gesture Recognition stacks.	47
Table 5 – Final Concepts Table for Drone Software stacks.	47
Table 6 – Final Concepts Table for Gesture Recognition stacks.	47
Table 7 – Pugh Matrix for Drone Software stacks.	48
Table 8 – Pugh Matrix for Gesture Recognition stacks.	48

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
CDR	Critical Design Review
ConOps	Concept of Operations
DoD	Department of Defense
DSM	Dependency Structure Matrices
EASA	European Union Aviation Safety Agency
EKF	Extended Kalman Filter
ESC	Electronic Speed Control
FAA	Federal Aviation Administration
FC	Flight Controller
FPV	First-Person View
FR	Functional Requirement
GCS	Ground Control Station
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HDI	Human-Drone Interaction
HMI	Human-Machine Interaction
IMU	Inertial Measurement Unit
MDM	Multiple-Domain Matrix
NASA	National Aeronautics and Space Administration
NFR	Non-Functional Requirement
PDB	Power Distribution Board
PDR	Preliminary Design Review
ROS	Robot Operating System
SITL	Software-in-the-Loop
SRR	System Requirements Review
UAV	Unmanned Aerial Vehicle
WSL	Windows Subsystem for Linux

CONTENTS

1	INTRODUCTION	15
1.1	RESEARCH OBJECTIVES	16
1.1.1	General objectives	16
1.1.2	Specific objectives	16
1.2	RESEARCH QUESTIONS	17
1.3	RESEARCH METHODOLOGY	17
2	THEORETICAL BACKGROUND	19
2.1	HCI DRONE CONTROL METHODS	19
2.1.1	Electromechanical	20
2.1.2	Bioelectrical	21
2.1.3	Vision-Based	22
2.1.4	Speech-Based	24
2.2	FLIGHT CONTROLLER AND AUTOPILOT SOFTWARE	24
2.2.1	PX4	26
2.2.2	ArduPilot	27
2.3	ROBOTIC MIDDLEWARE FRAMEWORKS	28
2.3.1	ROS 2	29
2.3.2	Micro XRCE-DDS	30
2.3.3	MAVLink	31
2.4	ROBOTICS SIMULATORS	31
2.4.1	Gazebo	32
2.4.2	AirSim	33
2.5	GESTURE RECOGNITION TECHNOLOGIES	33
2.5.1	Leap Motion Controller	34
2.5.2	Intel RealSense	35
2.5.3	Microsoft Kinect	35
2.6	RELATED WORK	36
2.7	CHAPTER FINDINGS	38
3	METHODOLOGY	39
3.1	SYSTEM REQUIREMENTS REVIEW	39
3.1.1	Mission and ConOps	39
3.1.2	Stakeholders	40
3.1.3	System Requirements	41
3.1.3.1	Functional Requirements	41
3.1.3.2	Non-Functional Requirements	42
3.1.4	Validation Criteria	43
3.2	PRELIMINARY DESIGN REVIEW	44

3.2.1	Architecture Diagram	44
3.2.2	MDM Function Correlations	45
3.2.3	Morphological Matrices	45
3.2.4	Pugh Matrices	48
3.2.5	Final Configuration	49
3.2.6	Risk Management	49
3.3	GESTURE RECOGNITION IMPLEMENTATION	50
3.3.1	Hand Tracking Data Structures	51
3.3.2	Initial Setup and Testing	51
3.3.3	Mapping Gesture to Drone Commands	52
3.3.4	Gesture Threshold Calibration	53
3.4	SIMULATION SETUP FOR SITL EXECUTION	54
3.4.1	Development Environment Configuration	54
3.4.2	Communication Architecture	55
3.5	CHAPTER REMARKS	56
4	RESULTS AND DISCUSSION	57
4.1	IMPLEMENTATION RESULTS	57
4.1.1	Optimal Input Threshold Values	57
4.1.2	ROS Package and SITL Execution	59
4.1.3	Hardware Prototype	61
4.1.4	Analysis of Requirements Validation	62
4.2	RESEARCH DISCUSSION	63
4.2.1	Comparison With Traditional Control Methods	63
4.2.2	Potential Applications and Use Cases	64
4.2.3	Limitations And Challenges Encountered	65
4.2.4	Future Improvements And Research Directions	65
5	CONCLUSION	67
5.1	SUMMARY OF FINDINGS AND CONTRIBUTIONS	67
5.2	IMPLICATIONS TO THE FIELD OF HCI FOR DRONES	68
	REFERENCES	69
	APPENDIX A – CONTROL LISTENER CLASS	73
	APPENDIX B – SETUP BASH COMMANDS	75
	ANNEX A – STAKEHOLDER MAPS	77

1 INTRODUCTION

The integration of unmanned aerial vehicles (UAVs) into various sectors of society has witnessed significant growth in recent years, transforming applications ranging from recreational use to professional services. According to (FEDERAL AVIATION ADMINISTRATION, 2023), the number of registered drones in the United States exceeded 1.5 million by 2023. This evolution in drone technology has brought forth new challenges in human-machine interaction, particularly in the development of intuitive and accessible control interfaces. Traditional drone control methods, primarily utilizing physical remote controllers, present limitations in terms of accessibility and user interaction, especially for individuals with limited mobility (FEUERRIEGEL et al., 2021).

The emergence of gesture-based control systems represents a paradigm shift in human-drone interaction (HDI), offering potential solutions to existing limitations in drone control interfaces. According to (SARKAR et al., 2016), natural user interfaces, particularly those based on hand gestures, can provide more intuitive interaction with autonomous systems. This approach aligns with broader trends in human-computer interaction, where the focus increasingly shifts toward developing more natural and accessible input methods.

Contemporary drone control systems predominantly rely on physical controllers requiring two-handed operation, which can present significant barriers for users with limited mobility or those requiring simplified control mechanisms. This limitation becomes particularly relevant as drone applications expand across various sectors, including search and rescue operations, industrial inspection, and recreational use. The challenge lies in developing control systems that maintain precise operation while improving accessibility and ease of use.

Recent advancements in gesture recognition technology, particularly through devices such as the Leap Motion Controller, have enabled new possibilities in human-drone interaction. Studies by (LEE; YU, 2023) demonstrate the potential of wearable drone controllers utilizing machine learning-based hand gesture recognition, indicating promising directions for development. Similarly, research by (ISKANDAR et al., 2023) has shown the viability of artificial intelligence-based human gesture tracking for quadrotor drone control, suggesting practical applications for these technologies.

The justification for developing gesture-based drone control systems extends beyond mere technological advancement. It addresses fundamental issues of accessibility and user experience in drone operation. As (ZHONG et al., 2024) note in their research on vision-based autonomous planning systems for quadrotor UAVs, the integration of advanced control mechanisms can significantly enhance safety and operational efficiency. This approach particularly benefits users with limited mobility, who may find traditional control methods challenging or impossible to use effectively.

Furthermore, the development of gesture-controlled drone systems aligns with

broader initiatives in assistive technology and universal design. Research by (BEGUM; HAQUE; KESELJ, 2020) on deep learning models for gesture-controlled drone operation demonstrates the potential for creating more inclusive control systems. This work suggests that gesture-based interfaces can provide equivalent or superior control precision compared to traditional methods while maintaining accessibility for a broader range of users.

The economic implications of developing more accessible drone control systems are also significant. As the global drone market continues to expand, the demand for intuitive and accessible control methods increases proportionally. The development of gesture-based control systems could potentially open new market segments and applications, particularly in sectors where traditional control methods present limitations.

1.1 RESEARCH OBJECTIVES

In order to establish a framework for gesture-based drone control, the following objectives are suggested in this work.

1.1.1 General objectives

The primary objective of this research is to develop and validate a framework for gesture-based drone control, operating in a Software-In-The-Loop (SITL) environment, that enables single-handed operation.

1.1.2 Specific objectives

- Design a modular system architecture that establishes communication between gesture recognition and drone flight control, incorporating suitable hardware and software.
- Document the development process following an adapted version of NASA Systems Engineering Handbook procedures, focusing on requirements definition and preliminary design phases.
- Implement gesture recognition algorithms that process hand movements and translate them into corresponding flight control commands for drone operation.
- Validate the framework's functionality through simulation, demonstrating the feasibility and accessibility with one-handed drone control using the developed interface.
- Provide a qualitative comparison between developed and traditional drone control methods, highlighting the differences in operational approach.

1.2 RESEARCH QUESTIONS

Alongside the stated objectives, the following research questions guide the investigation:

1. How can natural gestures captured be effectively translated into drone control commands?
2. What are the technical challenges in implementing a gesture-controlled drone system, and how can they be addressed through system architecture and software integration?
3. In what ways does gesture-based control enhance human-drone interaction compared to traditional control methods?
4. What are the potential applications for drones controlled via natural gestures, and what future research is needed to advance this technology?

1.3 RESEARCH METHODOLOGY

This research adopts a qualitative approach, with an applied nature and exploratory objectives, implemented through an experimental project methodology. The work consists of four chapters:

1. *Chapter 2 - Theoretical Background*: describes the foundational technologies needed for gesture-based drone control, that includes drone control architectures, gesture recognition systems, flight controllers, and robotic middleware frameworks.
2. *Chapter 3 - Methodology*: explains the work done applying systems engineering principles to develop the proposed framework. The chapter states system requirements and operational criteria in the SRR phase, as well as architecture design process and final configuration assessment in the PDR phase. It then details the gesture recognition implementation for drone control and establishes simulation protocols for system validation in SITL (Software in the Loop) environment.
3. *Chapter 4 - Results and Discussion*: presents implementation outcomes focusing on the framework's effectiveness against traditional control methods and evaluates its accessibility improvements.
4. *Chapter 5 - Conclusion*: synthesizes research findings, evaluates objective achievement, and proposes future development directions. It emphasizes the work's contribution to accessible drone control systems and human-robot interaction.

2 THEORETICAL BACKGROUND

This chapter presents the fundamental concepts required to understand the prototype designing process. The research procedure began with a structured literature review using specific search terms in Google Scholar, focusing on recent developments using the key-phrases as “natural gesture control for drones” and “human-machine interfaces in drone operations”. The sources were organized chronologically and by topic, allowing for analysis of research trends and technological evolution in this field. This structured approach revealed key areas that form the foundation of the current research: drone control methods, gesture recognition technologies, flight controller systems, and robotic frameworks.

The research draws significantly from recent works in gesture-controlled drone interfaces, particularly the studies by (LEE; YU, 2023) on wearable drone controllers and (ISKANDAR et al., 2023) on artificial intelligence-based gesture tracking. These works provide essential insights into the tracking integration with drone control systems. Additional guidance comes from (ZHONG et al., 2024), who present perspectives on vision-based autonomous planning systems for quadrotor UAVs. These key contributions are examined in detail in the Related Work section at the end of this chapter.

The chapter explores the different drone control methods, widely adopted flight control software, robotic middleware frameworks that enable communication between system components also mentioning their possible simulators and current gesture recognition hardware options. With the theoretical foundations established, the work itself, designing the system and implementing the framework, is better detailed in the methodology.

2.1 HCI DRONE CONTROL METHODS

Human-Computer Interaction (HCI) methods for drone control are systems and technologies that enable humans to communicate with and command unmanned aerial vehicles (UAVs). The main function of these methods is to translate human intentions into actionable commands for drones, ensuring effective and precise control. HCI drone control methods play a critical role in expanding the accessibility and usability of drones, especially in applications where conventional remote controllers or programming-based approaches are insufficient.

Traditional drone control typically relies on handheld remote controllers that require manual input through joysticks and buttons. However, advancements in technology have enabled alternative control methods that can enhance user experience, improve precision, and allow for more natural interactions. They rely on a variety of input modalities that interpret human gestures, speech, bioelectrical signals, or other physical interactions to provide seamless communication between the user and the drone.

The selection of control methods depends on several factors including the intended

application, environmental conditions, and user expertise. Professional applications typically prioritize precision and reliability, while consumer uses focus more on accessibility and ease of operation (Feuerriegel et al., 2021). But the current research state of drone control methods emphasizes intuitive interfaces and enhanced user experience.

This chapter explores the most significant categories of HCI methods for drone control on each subsection, examining in detail, discussing their mechanisms, their underlying technologies, and their advantages and challenges. Understanding these methods is crucial for developing effective HCI systems that can be tailored to specific applications and user needs.

2.1.1 Electromechanical

Electromechanical control represents the traditional and most widely used method for drone operation. This approach relies on physical input devices, primarily radio transmitters with joysticks and switches, to send control signals to the aircraft. According to (SARKAR et al., 2016), these systems translate mechanical movements into digital or analog signals that control the drone's flight parameters.

Figure 1 – Radio Transmitter Frsky Taranis Q X7 model.



Source: Frsky website (FRSKY..., n.d.).

As shown in Figure 1, modern radio transmitters feature dual joysticks that control four primary flight channels: throttle, yaw, pitch, and roll. Additional switches and knobs provide auxiliary functions like flight mode selection and camera control. These controllers typically operate in the 2.4 GHz frequency band, offering reliable communication range and interference resistance.

The main advantage of electromechanical control lies in its precise input capabilities and tactile feedback. However, this method presents limitations in terms of accessibility, as noted by (FEUERRIEGEL et al., 2021). The requirement for two-handed operation

and complex finger movements can create barriers for users with limited mobility or those requiring simplified control interfaces.

Despite these limitations, electromechanical control remains the standard for professional drone operations due to its reliability and precision. The method benefits from decades of development in radio control technology, resulting in robust systems with predictable performance characteristics.

2.1.2 Bioelectrical

Bioelectrical control methods utilize biological signals from the human body to direct drone movements. This approach primarily relies on electromyography (EMG) sensors that detect electrical activity produced by skeletal muscles during contraction and relaxation. Through signal processing and pattern recognition, these muscle signals are converted into drone control commands.

The implementation of bioelectrical control typically involves wearable devices equipped with EMG sensors. These devices capture the electrical signals generated by muscle movements, which are then processed and translated into control inputs. This technology offers a hands-free alternative to traditional control methods, potentially benefiting users with limited mobility.

Figure 2 – Myo Armband for muscle activity detection.



(a) Myo EMG Armband model.



(b) Armband being used to control a drone.

Source: Thalmic Labs video (LABS, 2015)

As illustrated in Figure 2a, the Myo armband represents a common implementation of EMG sensing technology. The device contains multiple sensors that detect muscle activity in the forearm, allowing users to control drones through subtle muscle movements. Research by (HU; WANG, 2020) demonstrates that these systems can achieve reliable control accuracy after proper calibration and user training.

However, bioelectrical control faces several technical challenges: signal noise and interference from other muscle movements, need for regular sensor calibration, limited

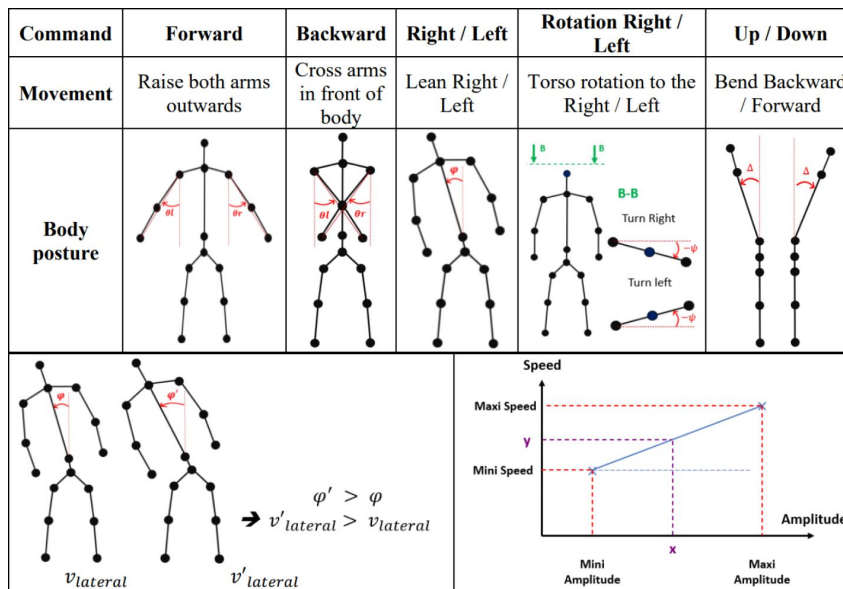
number of distinct commands compared to traditional controllers, variation in signal patterns between different users

Recent developments in signal processing and machine learning have improved the reliability of bioelectrical control systems. (LEE; YU, 2023) report successful integration of EMG-based control with haptic feedback systems, enhancing user control precision. These advances suggest potential applications in assistive technology and specialized drone operations where hands-free control provides significant advantages.

2.1.3 Vision-Based

Vision-based control methods interpret visual information from cameras or sensors to convert human movements into drone commands. This approach encompasses multiple input methods: full body gestures, hand movements, facial expressions, and eye tracking. According to (ISKANDAR et al., 2023), these systems use computer vision and machine learning algorithms to recognize and classify different types of visual inputs.

Figure 3 – Body gesture positions for drone control.



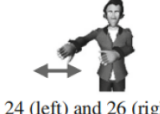
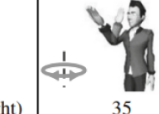









Source: adapted from (OBAID et al., 2016)

Body gesture recognition tracks the user's full body posture and movements. As shown in Figure 3, specific body positions correspond to different flight commands. (GIO; BRISCO; VULETIC, 2021) demonstrate that body gestures can provide intuitive control for basic flight maneuvers, though they require sufficient space for user movement.

Hand gesture recognition focuses on tracking hand positions and movements, offering more precise control in confined spaces. Research by (OBAID et al., 2016) established common hand gesture patterns for drone control, illustrated in Figure 4. This method achieves high accuracy through specialized tracking devices or standard cameras with computer vision algorithms.

Figure 4 – Hand gesture mapping for flight commands.

	<i>Up & Down</i>		<i>Left & Right</i>		<i>Rotate Left & Rotate Right</i>		<i>Forward & Backward</i>		<i>Takeoff & Land</i>	
Survey	up (81) upward (8)		left (82) west (6)		turn left (30) rotate left (26) rotate counter clockwise (17)		forward (58) go (13) straight (8) north (6)		up (28) takeoff (21) lift/start (17)	
	down (83) downward (4)		right (84) east (6)		turn right (31) rotate right (25) rotate clockwise (17)		back/backward (42) reverse/south (6)		land (56) down (23)	
Experiment	up (88) higher (12)		left (80) turn left (12)		turn (48) turn left (12)		forward (56) straight (32)		up (42) start (32)	
	down (100)		right (92) turn right (8)		turn (50) turn right (12)		back (76) backward (16)		land (40) stop (33) down (13)	
Gestures									-	
	31	58	24 (left) and 26 (right)		35		23	35		
Voice	7	9	8	10	10	9	7	7	9	15
Gesture	9	10	9	9	8	9	9	9	2	3
V+G	17	14	16	14	15	15	17	17	22	15
Interview									-	
	33	37	30	30	22		22	37		

Source: (PESHKOVA; HITZ; AHLSTRÖM, 2017).

Facial and eye tracking present alternative input methods for specialized applications. These approaches monitor facial expressions or eye movements to control drone behavior, particularly useful for users with limited mobility. However, (ZHONG et al., 2024) note that these methods typically offer fewer distinct commands compared to hand or body gestures.

The implementation of vision-based control systems involves several key components, Image capture devices (cameras, depth sensors), Computer vision algorithms for feature detection, Machine learning models for gesture classification, Signal processing for command translation. Recent advances in deep learning have significantly improved the accuracy of vision-based systems. Studies by (BEGUM; HAQUE; KESELJ, 2020) show that modern gesture recognition algorithms can achieve recognition rates above 95% under controlled conditions. These improvements make vision-based methods increasingly viable for practical drone control applications.

Challenges in vision-based control include Environmental factors affecting image quality, Processing latency in real-time applications, Variation in lighting conditions, Need for consistent background separation. Despite these challenges, vision-based methods continue to evolve, offering increasingly robust and natural interaction options for drone control. The flexibility to choose between different input methods makes this approach adaptable to various user needs and operational contexts.

2.1.4 Speech-Based

Speech-based control enables drone operation through voice commands, offering a hands-free approach to human-drone interaction. This method processes spoken instructions using natural language processing algorithms to generate corresponding flight commands. Research by (ABIOYE, 2023) demonstrates that speech control can complement other input methods, creating more flexible and accessible drone interfaces.

The implementation of speech-based control requires sophisticated audio processing systems that convert spoken words into machine-readable commands. These systems first capture audio through microphones, then process the signal to remove ambient noise and isolate speech patterns. Speech recognition algorithms analyze the processed audio to identify specific commands within the system's predefined vocabulary.

Modern speech control systems incorporate context awareness and command verification to enhance reliability. For example, the system might require specific command phrases or verbal confirmation before executing critical maneuvers. (ABIOYE, 2023) notes that this approach reduces accidental command execution while maintaining operational efficiency. Speech recognition for drone control faces particular challenges in outdoor environments. Wind noise, propeller sounds, and ambient conversations can interfere with command detection. Additionally, variations in accent, speech patterns, and environmental conditions may affect recognition accuracy. These limitations often restrict speech-based control to specific operational scenarios or require integration with other control methods.

Recent developments in machine learning have improved the robustness of speech recognition systems. Natural language processing models can now better handle different accents and speaking styles, making speech control more accessible to diverse user groups. However, the method still primarily serves as a supplementary control interface rather than a primary flight control system.

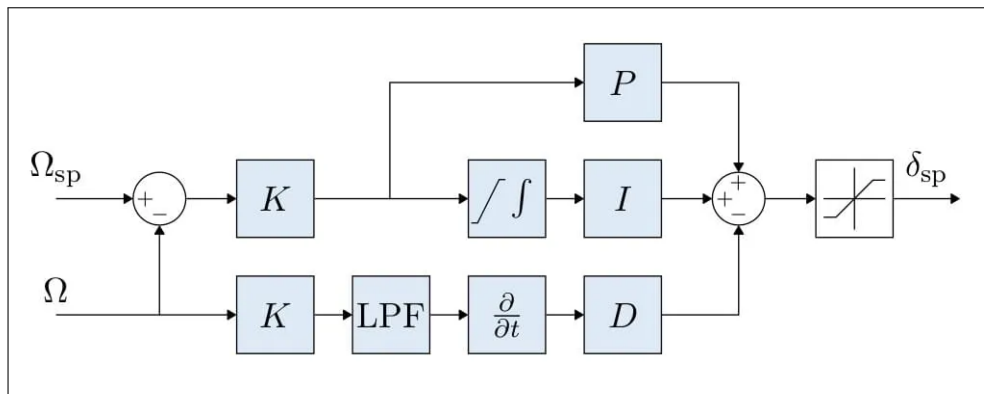
Applications of speech-based control prove particularly valuable in scenarios requiring hands-free operation, such as search and rescue missions or industrial inspections where operators must manipulate other equipment simultaneously. The technology also shows promise in educational settings, where verbal interaction can help new users learn drone operation concepts more intuitively.

2.2 FLIGHT CONTROLLER AND AUTOPILOT SOFTWARE

Flight Controllers (FC) and autopilot software form the core of drone control systems, managing the complex task of keeping the aircraft stable and responsive to user commands. These systems process sensor data and user inputs to control motor outputs, enabling stable flight through continuous adjustments of propeller speeds. Such controllers have evolved from early research prototypes into mature platforms widely used in both research and industry (BEARD; MCLAIN, 2012).

The primary function of flight control software involves maintaining aircraft stability through feedback control loops. At its core, a Flight Controller uses Proportional-Integral-Derivative (PID) controllers to process error signals—the difference between desired and actual aircraft states (SHIM; KIM; SASTRY, 2000). The PID controller calculates appropriate responses based on the current error (Proportional term), accumulated past errors (Integral term), and the rate of error change (Derivative term) (LEE; LEOK; MCCLAMROCH, 2010). The detailed implementation of this concept in a multicopter’s angular rate controller is illustrated in Figure 5.

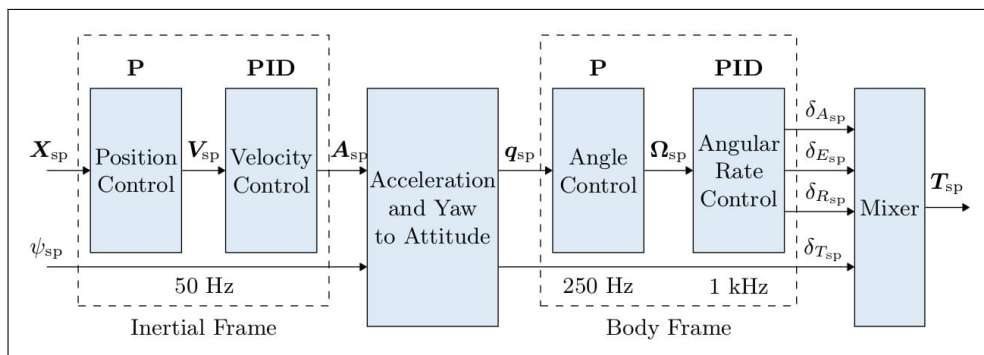
Figure 5 – Multicopter Angular Rate Controller.



Source: (PX4 DEVELOPMENT TEAM, 2024).

Flight rates, which determine how quickly the drone responds to control inputs, represent another key aspect of flight control. These rates affect the drone’s agility and stability, with higher rates providing more responsive but potentially less stable behavior, while lower rates result in smoother but slower responses (HOFFMANN et al., 2007). The multicopter control architecture, as illustrated in Figure 6, plays a crucial role in ensuring optimal performance.

Figure 6 – Multicopter Control Architecture.



Source: (PX4 DEVELOPMENT TEAM, 2024).

The architecture of flight control systems typically includes several interconnected modules: sensor processing for gathering flight data, attitude estimation for determining

aircraft orientation, position control for maintaining location, and motor mixing for converting control signals into motor commands (SHIM; KIM; SASTRY, 2000). State estimation often employs algorithms such as the Extended Kalman Filter (EKF) or complementary filters to fuse sensor data into reliable attitude and position information (RAJA, 2017). Modern flight controllers integrate these components through efficient software implementations, with PX4 and ArduPilot representing two leading open-source solutions in this field. A detailed overview of these components and their data flow within PX4 is shown in Figure 7.

2.2.1 PX4

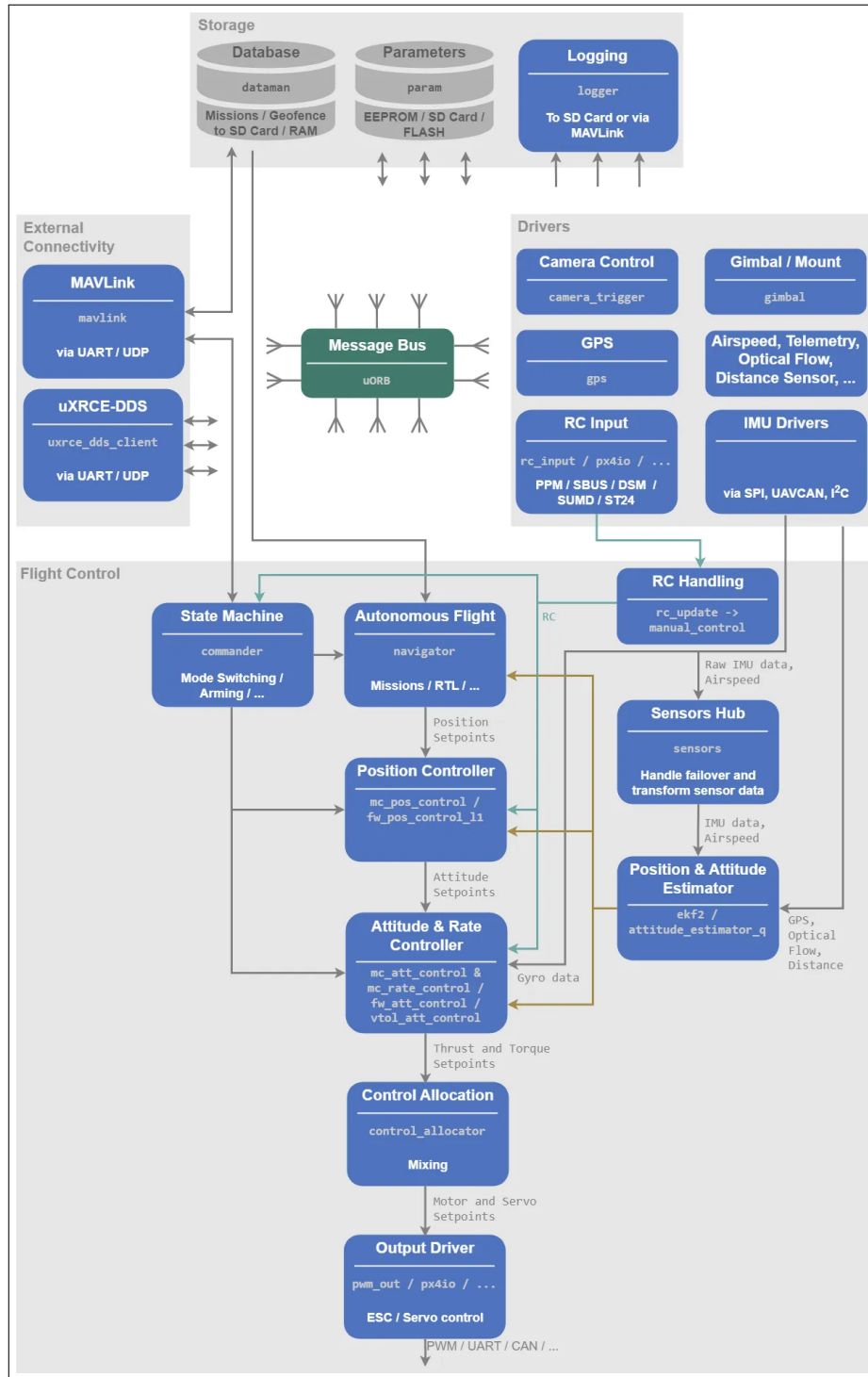
PX4 represents an open-source flight control software developed by the Dronecode Project, originating from the Swiss Federal Institute of Technology (ETH) in Zurich. This software provides comprehensive flight control capabilities for unmanned aerial vehicles through a modular architecture that enables both manual and autonomous operations (MEIER; HONEGGER; POLLEFEYS, 2015).

The technical framework of PX4 centers on its flight stack, which processes sensor data and control inputs through a series of specialized modules. These modules handle tasks from low-level motor control to high-level mission planning, all coordinated through the uORB (micro Object Request Broker) messaging system. The software implements advanced control algorithms for position estimation, navigation, and flight control, supporting various flight modes from manual control to fully autonomous missions (MEIER; HONEGGER; POLLEFEYS, 2015).

PX4 provides broad hardware compatibility, functioning with numerous flight controller boards including the Pixhawk series, Matek Systems, and Holybro controllers, as shown in Figure 8. This compatibility extends to both hobbyist and professional-grade hardware, making it suitable for applications ranging from recreational drones to commercial unmanned aircraft systems (BEARD; MCLAIN, 2012; PX4 DEVELOPMENT TEAM, 2024). The hardware support encompasses different sensor configurations, enabling integration with GPS modules, optical flow sensors, and various other peripheral devices.

The ecosystem surrounding PX4 includes development tools like QGroundControl for mission planning and SITL (Software In The Loop) simulation capabilities, facilitating both development and testing (MEIER; HONEGGER; POLLEFEYS, 2015; PX4 DEVELOPMENT TEAM, 2024). Commercial adopters can utilize the MAVLink protocol for custom implementations, while maintaining compliance with the software's BSD license terms. Through its robust architecture and extensive toolset, PX4 provides a foundation for developing advanced drone applications and research projects.

Figure 7 – Detailed overview of the building blocks and data flow of PX4.



Source: (PX4 DEVELOPMENT TEAM, 2024).

2.2.2 ArduPilot

ArduPilot constitutes an open-source autopilot software system with origins in the DIY Drones community, developing into a comprehensive platform for autonomous vehicle control. The software maintains active development through a global community of contributors, supporting various vehicle types beyond just multicopters (TEAM, 2024).

Figure 8 – Holybro Pixhawk 6X model.



Source: (HOLYBRO, 2024).

The technical architecture of ArduPilot features a robust flight control system built on the APM (ArduPilot Mega) codebase. Its core functionality implements nested PID loops for attitude and position control, with support for multiple flight modes including stabilize, altitude hold, and various autonomous operations (TEAM, 2024). The software utilizes an EKF for state estimation, combining data from multiple sensors to achieve precise vehicle control (RAJA, 2017).

Hardware compatibility spans across numerous flight controller boards, with primary support for ChibiOS-based platforms. Common compatible hardware includes Cube autopilot series, Pixhawk variants, and various commercially available flight controllers from manufacturers like Holybro and Mateksys. ArduPilot demonstrates flexibility in sensor integration, supporting standard peripherals such as GPS, optical flow sensors, and rangefinders (TEAM, 2024).

ArduPilot's development environment includes Mission Planner and MAVProxy ground control stations, along with SITL simulation capabilities for testing and development. The platform provides extensive documentation and development tools, enabling both educational use and commercial applications under the GNU GPL license. Its modular architecture allows for customization while maintaining core stability and reliability features required for autonomous operations (TEAM, 2024).

2.3 ROBOTIC MIDDLEWARE FRAMEWORKS

Robotic middleware frameworks provide the communication infrastructure that enables different components of a robotic system to interact. In drone applications, these frameworks manage data flow between the gesture recognition system, flight controller, and ground control station. They handle message passing, data distribution, and service calls while abstracting the complexity of network protocols and hardware interfaces (QUIGLEY et al., 2009).

The selection of an appropriate middleware framework impacts system latency, reliability, and scalability. For gesture-controlled drones, the middleware must support real-

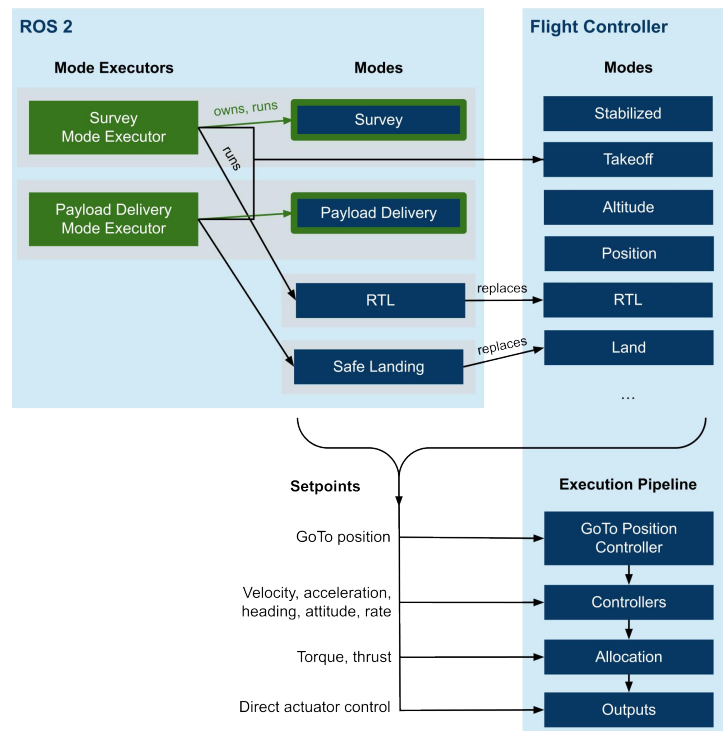
time communication to translate hand movements into flight commands with minimal delay. Three main frameworks emerge as candidates for this application: ROS 2 with its extensive robotics ecosystem, Micro XRCE-DDS for resource-constrained systems, and MAVLink as the de facto standard in drone communications (MAVLINK, n.d.; EPROSIMA, n.d.).

2.3.1 ROS 2

The Robot Operating System (ROS) originated at the Stanford Artificial Intelligence Laboratory and later developed at Willow Garage, represents an open-source framework for robot software development. ROS provides libraries and tools for building robot applications, managing hardware abstraction, and implementing common robotics functionality (QUIGLEY et al., 2009).

ROS version 1, released in 2007, established fundamental concepts like the publisher-subscriber architecture, parameter services, and the ROS master node for centralized communication. ROS 2 constraints through Data Distribution Service (DDS) middleware integration, improved real-time capabilities, and multi-robot support. For gesture-controlled drone applications, ROS 2 provides essential features including message transport and compatibility with simulation environments like Gazebo (GHASEMI et al., 2020). The architecture of ROS 2 and its interaction with the PX4 flight stack is depicted in Figure 9.

Figure 9 – How ROS 2 control interface modes interact with PX4.



Source: (PX4 DEVELOPMENT TEAM, 2024).

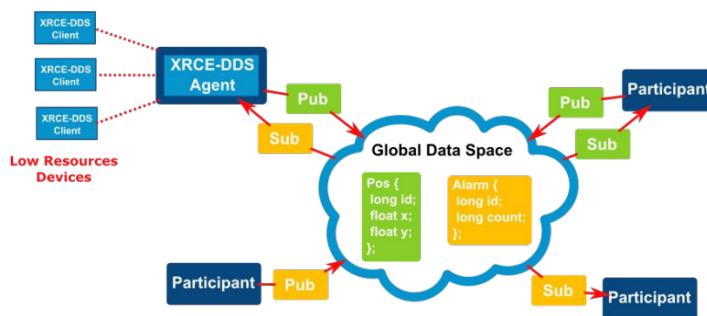
The framework supports multiple programming languages through client libraries, primarily C++ and Python, enabling flexible development of robotic applications. ROS 2's

modular architecture allows independent development of system components, facilitating the integration of gesture recognition modules with drone control systems (LATIF; BUCKLEY; SECCO, 2022).

2.3.2 Micro XRCE-DDS

eProsima Micro XRCE-DDS is an open-source wire protocol that implements the OMG DDS for eXtremely Resource Constrained Environment standard (DDS-XRCE). The aim of the DDS-XRCE protocol is to provide access to the DDS Global-Data-Space from resource-constrained devices. This is achieved thanks to a client-server architecture, where low resource devices, called XRCE Clients, are connected to a server, called XRCE Agent, which acts on behalf of its clients in the DDS Global-Data-Space (see Figure 10).

Figure 10 – Communication schema for XRCE-DDS Agent.



Source: (PX4 DEVELOPMENT TEAM, 2024).

The uXRCE-DDS middleware consists of a client running on PX4 and an agent running on the companion computer, with bi-directional data exchange between them over a serial, UDP, TCP or custom link. The agent acts as a proxy for the client to publish and subscribe to topics in the global DDS data space.

The PX4 `uxrce_dds_client` is generated at build time and included in PX4 firmware by default. It includes both the "generic" micro XRCE-DDS client code, and PX4-specific translation code that it uses to publish to-from uORB topics. The subset of uORB messages that are generated into the client are listed in `PX4-Autopilot/src/modules/uxrce_dds_client/dds_topics.yaml`. The generator uses the uORB message definitions in the source tree: `PX4-Autopilot/msg` to create the code for sending ROS 2 messages.

ROS 2 applications need to be built in a workspace that has the same message definitions that were used to create the uXRCE-DDS client module in the PX4 Firmware. One can include these by cloning the interface package `PX4/px4_msgs` into their ROS 2 workspace (branches in the repository correspond to the messages for different PX4 releases).

Note that the micro XRCE-DDS agent itself has no dependency on client-side code. It can be built from source either standalone or as part of a ROS build. It will normally require to start both the client and agent when using ROS 2. Note that the uXRCE-DDS client is built into firmware by default but not started automatically except for simulator builds.

2.3.3 MAVLink

MAVLink (Micro Air Vehicle Link) provides a communication protocol designed specifically for unmanned vehicles. Created by Lorenz Meier in 2009 at ETH Zurich, this lightweight messaging protocol enables data exchange between drones and ground control stations (NATARAJAN; NGUYEN; METE, 2018).

The protocol uses a binary serialization format to maintain efficient bandwidth usage while ensuring message integrity through checksum verification. MAVLink messages contain essential flight data including vehicle telemetry, command instructions, and system parameters. When implemented in gesture-controlled systems, MAVLink translates high-level gesture commands into standardized drone control messages.

MAVLink's architecture supports both point-to-point and multi-node network topologies, facilitating communication between multiple system components. The protocol defines message sets for various operations, from basic flight control to advanced mission planning. Version 2.0, released in 2017, introduced features such as message signing for security and support for larger payloads (HU; WANG, 2020).

In gesture recognition applications, MAVLink handles the transmission of processed gesture commands from the ground station to the flight controller, while simultaneously managing the return flow of vehicle state information. This bidirectional communication enables real-time feedback for gesture control systems (BEGUM; HAQUE; KESELJ, 2020).

2.4 ROBOTICS SIMULATORS

Robotics simulators provide virtual environments for testing robot and drone behavior before physical deployment (SICILIANO; KHATIB, 2016). These platforms enable researchers and developers to validate control algorithms, test system responses, and train artificial intelligence models, all without risking hardware damage.

For drone applications, simulators recreate flight dynamics and environmental conditions, allowing developers to assess control and navigation algorithms in diverse and challenging scenarios. This approach reduces development costs and accelerates the testing process by enabling rapid prototyping and validation of new functionalities.

The selection of an appropriate simulation platform depends on project requirements, including physics fidelity, sensor simulation capabilities, and integration with robotic frameworks. Two platforms that have gained prominence in drone development

are Gazebo and AirSim, each offering distinct advantages and features suited to different applications (FURRER et al., 2016).

2.4.1 Gazebo

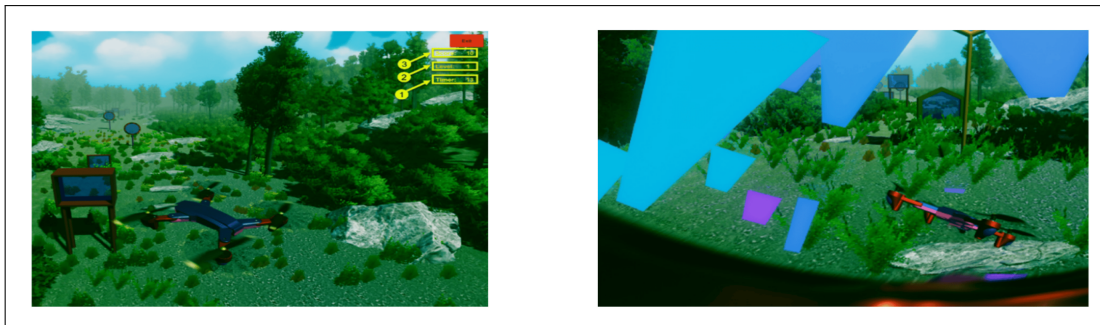
Gazebo emerged as an open-source robotics simulator developed at the University of Southern California in 2002, later maintained by the Open Source Robotics Foundation (KOENIG; HOWARD, 2004). Over time, it has become closely integrated with the Robot Operating System (ROS), providing a standard platform for testing robotic systems and drone controls (FURRER et al., 2016).

The simulation environment in Gazebo leverages physics engines like the Open Dynamics Engine (ODE) and Bullet Physics for realistic motion calculations. These engines provide accurate rigid body dynamics, essential for simulating drone flight characteristics, collisions, and complex environmental interactions (KOENIG; HOWARD, 2004).

Gazebo also excels in sensor modeling capabilities, supporting simulations of cameras, GPS receivers, and Inertial Measurement Units (IMUs). By generating synthetic sensor data that closely mimics real-world conditions, developers can effectively test perception algorithms and refine control systems prior to field testing (FURRER et al., 2016). This is particularly valuable when validating drone navigation and obstacle avoidance strategies.

Environments in Gazebo are described using the Simulation Description Format (SDF), enabling the creation of elaborate worlds with buildings, varied terrain, and dynamic obstacles. Users can leverage both community-generated models and custom designs, resulting in highly specialized simulated scenarios (KOENIG; HOWARD, 2004). As shown in Figure 11, this flexibility permits detailed scenario configurations and intuitive visualizations.

Figure 11 – Gazebo in designed scenarios.



Source: (COVACIU; IORDAN, 2022).

The integration with ROS ensures easy communication between simulated components using standardized message protocols. This compatibility allows for transferring control algorithms directly from simulation to physical drones with minimal adjustments,

ensuring consistent behavior across both virtual and real-world environments (FURRER et al., 2016).

2.4.2 AirSim

AirSim, an open-source simulator developed by Microsoft Research, focuses on drones and autonomous vehicles, providing photorealistic environments rendered through the Unreal Engine (SHAH et al., 2018). Leveraging the NVIDIA PhysX engine, AirSim offers high-fidelity physics simulations that accurately represent aerodynamics and environmental interactions.

Its sensor simulation includes cameras, GPS, barometers, and IMUs, making it suited for computer vision research applications. By connecting with external flight controllers and supporting APIs for platforms like ROS and PX4, AirSim simplifies integration into existing robotics workflows.

A distinctive feature of AirSim is the capability to simulate different weather conditions and lighting scenarios. This allows developers to test drone systems under challenging environmental factors, such as rain, fog, or changing wind patterns, further enhancing the reliability and robustness of control solutions (SHAH et al., 2018).

2.5 GESTURE RECOGNITION TECHNOLOGIES

Gesture recognition technologies represent specialized hardware devices and software systems that capture, track, and interpret human hand or body movements, converting them into meaningful digital commands. These systems use various sensing methods including infrared cameras, depth sensors, and computer vision algorithms to detect and process human gestures in real-time (LEE; YU, 2023).

The integration of gesture recognition in drone control systems addresses a fundamental challenge in Human-Drone Interaction (HDI): creating natural, intuitive control interfaces that reduce the learning curve for operators while maintaining precise control capabilities (ISKANDAR et al., 2023). This technology enables users to control drones through predefined hand movements instead of traditional physical controllers, potentially increasing accessibility and operational efficiency.

Modern gesture recognition hardware typically employs one or more sensors working together to achieve reliable hand tracking. These sensors capture spatial data about hand position, orientation, and movement patterns, which software algorithms then process to identify specific gestures. The following subsections examine three prominent gesture recognition devices: the Leap Motion Controller, Intel RealSense, and Microsoft Kinect, each offering distinct approaches to hand tracking and gesture recognition

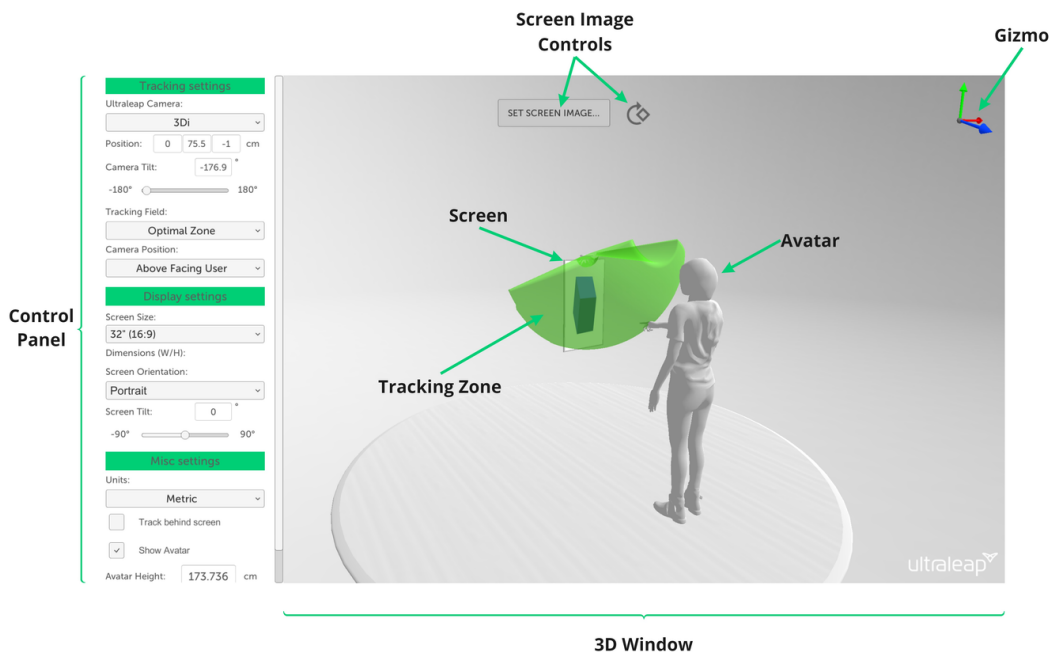
2.5.1 Leap Motion Controller

The Leap Motion Controller, developed by UltraLeap, is a compact USB peripheral device designed for high-precision hand and finger tracking (MUTALIB, 2020). The device utilizes two monochromatic infrared cameras and three infrared LEDs to generate a three-dimensional interactive zone extending approximately 60 centimeters above the device (LEE; YU, 2023).

From a technical perspective, the controller operates by projecting infrared light patterns and capturing the reflected data through its cameras at a rate exceeding 200 frames per second. This high sampling rate enables sub-millimeter accuracy in tracking hand movements and finger positions. The device features a wide field of view of 150 degrees, allowing it to maintain tracking consistency even during rapid hand movements (ZHAO et al., 2018).

As depicted in Figure 12, the controller's software processes the captured infrared images to construct a detailed skeletal model of the user's hands. This model includes precise positioning data for palm orientation, finger joints, and overall hand pose. The tracking system can maintain accuracy even when fingers overlap or hands cross paths, making it particularly suitable for natural gesture interactions in drone control applications (NATARAJAN; NGUYEN; METE, 2018).

Figure 12 – Ultraleap Camera Position Visualizer Controls.



Source: (ULTRALEAP, 2024).

Integration with various development environments occurs through the UltraLeap SDK, which provides APIs for accessing hand tracking data and implementing gesture recognition algorithms. The SDK supports multiple programming languages, facilitating

implementation across different platforms and applications (BEGUM; HAQUE; KESELJ, 2020).

2.5.2 Intel RealSense

The Intel RealSense represents a family of depth cameras designed for spatial computing applications, with the D435i model specifically optimized for robotics and drone applications (GHASEMI et al., 2020). This technology combines depth sensors, RGB cameras, and an inertial measurement unit (IMU) to provide accurate environmental mapping and gesture recognition capabilities.

The D435i depth camera, a key component in drone control applications, utilizes global shutter sensors and structured light stereo technology to generate depth maps with a resolution up to 1280 x 720 pixels at 90 frames per second (FEUERRIEGEL et al., 2021). The camera's active infrared stereo system projects a pattern onto surfaces, enabling reliable depth perception even in challenging lighting conditions.

The device's technical specifications include a depth field of view of $87^\circ \times 58^\circ$ and an RGB field of view of $69^\circ \times 42^\circ$. Operating at distances between 0.1 and 10 meters, the D435i provides depth resolution accuracy up to 1% of the distance from the target. The integrated IMU enhances tracking stability by providing motion data, crucial for maintaining gesture recognition accuracy during dynamic interactions (COVACIU; IORDAN, 2022).

RealSense technology implements computer vision algorithms directly in hardware, reducing processing latency and enabling real-time gesture recognition. The SDK provides developers with access to calibrated sensor data streams, facilitating integration with robotic control systems through standard programming interfaces.

2.5.3 Microsoft Kinect

The Microsoft Kinect, initially developed as a motion sensing device for the Xbox gaming console, evolved into a significant tool for gesture recognition applications (FERNANDEZ et al., 2016). The device combines an RGB camera, depth sensor, and multi-array microphone to enable full-body tracking and gesture recognition capabilities.

From a technical perspective, the Kinect utilizes structured light technology, projecting an infrared pattern to create depth maps with a resolution of 512 x 424 pixels at 30 frames per second. The system's depth sensing range extends from 0.5 to 4.5 meters, making it suitable for room-scale gesture recognition applications (PESHKOVA; HITZ; AHLSTRÖM, 2017). The RGB camera operates at 1920 x 1080 pixels, providing high-resolution color imaging alongside depth data.

Microsoft's SDK provides developers with access to raw sensor data and pre-built recognition algorithms, facilitating the implementation of gesture-based interfaces. While Microsoft discontinued Kinect production for gaming, its technology continues to influence modern gesture recognition systems, particularly in research and development applications.

2.6 RELATED WORK

The literature review for this research involved analyzing academic publications from 2016 to 2024, with particular attention to works published after 2020 due to significant technological advances in gesture recognition and drone control systems during this period. A structured search was conducted using Google Scholar, focusing on key phrases related to natural gesture control for drones and human-machine interfaces in drone operations.

Publication data was organized chronologically, categorized into main research areas (performance aspects, control methods, gesture recognition technologies, flight control software, robotics frameworks, and simulation platforms) and comprised into 6 charts distributing each topic as is possible to see in Figure 13. Analysis of publication trends revealed an increasing academic interest in gesture-based drone control, with a notable surge in research focusing on vision-based control methods and advanced gesture recognition techniques after 2020.

Performance aspects received consistent attention throughout the analyzed period, with recent works focusing more on control accuracy and user experience. Among control methods, vision-based approaches dominated the research landscape, followed by electromechanical solutions. The distribution analysis highlighted that bioelectrical and speech-based methods, while present in the literature, represented a smaller portion of the research focus.

Regarding technology stacks, the Leap Motion Controller emerged as a frequently used tool for gesture recognition, appearing in approximately 30% of the analyzed works. ROS and Gazebo simulator also showed significant presence in recent publications. However, it is important to note that some works might have used these or other technology stacks without explicitly mentioning them in their abstracts or keywords, potentially affecting the distribution analysis.

Figure 13 – Related Work analysis through multiple charts.



Source: Created by the author.

It is worth noting that while many studies may have utilized specific flight control software, robotics frameworks, or simulation platforms, not all publications explicitly mention their technical implementation details. This analysis therefore reflects only explicitly documented technology stacks, acknowledging that actual usage might be more widespread than represented in the literature.

Among the reviewed literature, the following four mentioned studies stand out for their uniqueness and direct relevance to the current research, providing key insights into gesture-based drone control implementation.

The research by (SEIDU; LAWAL, 2024) presents a comprehensive approach to personalized drone interaction by combining facial authentication with hand gesture control. Their system uses the DJI Tello drone's onboard camera with computer vision techniques, implementing the Histogram of Oriented Gradients method and FaceNet model for user verification. The hand gesture recognition integrates MediaPipe with a custom convolutional neural network, achieving real-time processing capabilities. This work stands out for its focus on security through user authentication while maintaining intuitive control mechanisms.

(LEE; YU, 2023) developed a wearable drone controller that utilizes an inertial measurement unit (IMU) placed on the back of the user's hand. Their approach includes a detailed analysis of different machine learning models for gesture classification and introduces haptic feedback through a wrist-mounted vibration motor that alerts users to obstacles. The research provides valuable insights into user experience through simulation experiments and practical validation with real drone flights, offering quantitative data on control effectiveness.

The work of (GIO; BRISCO; VULETIC, 2021) explores body gesture control using the Microsoft Kinect sensor. Their system processes full-body movements to generate drone commands, implementing a graphical user interface for real-time feedback. The research demonstrates advantages in natural interaction compared to traditional controllers, particularly for novice users. Their implementation includes additional functionality beyond basic flight control, such as photo and video capture through gesture commands.

(MUTALIB, 2020) investigated drone control using the Leap Motion sensor, focusing on translating hand positions into flight commands. Their system integrates hardware components including an Arduino microcontroller and digital-to-analog converter to interface with commercial drone hardware. The research examines three primary control aspects - throttle, pitch, and roll - and provides temporal analysis comparing drone response to hand movements. This work offers practical insights into implementing gesture control with existing commercial drone platforms.

2.7 CHAPTER FINDINGS

Based on the literature review presented in this chapter, the exploration of gesture-based drone control systems reveals the dependence of technologies from across multiple technical domains. Recent research demonstrates an interest towards more natural interaction methods, with vision-based approaches emerging as the predominant technology. The analysis of control methods, flight software, and recognition technologies indicates that combining the Leap Motion Controller with PX4 flight stack and ROS 2 middleware presents a viable configuration for implementing gesture-controlled drone systems.

The literature findings suggest that while gesture recognition accuracy has improved through advanced algorithms and hardware, challenges remain in system integration and real-time performance. Studies from 2020 onwards show increased focus on accessibility and user experience, though precision control still poses technical hurdles. Recent publications emphasize the importance of standardized testing protocols and validation methods for gesture-based control systems, particularly in safety-critical applications. These findings provide direction for the methodology and system development phases of this research.

3 METHODOLOGY

In this chapter, the Systems Engineering approach employed in this research work follows a structured development process, where the System Requirements Review (SRR) serves as the initial technical review milestone followed by the Preliminary Design Review (PDR), which is the final preparatory stage here, evaluating design alternatives and selecting the system architecture. After these engineering reviews, the gesture recognition implementation phase details how hand movements are captured and translated into drone commands. Finally, the simulation setup describes the testing arrangement used to validate system performance in a controlled virtual environment before physical deployment.

3.1 SYSTEM REQUIREMENTS REVIEW

The SRR is a step that establishes the foundation for the development process by ensuring that the system requirements are properly defined and aligned with the stakeholder needs before proceeding with the preliminary design phase. According to the NASA Systems Engineering Handbook (HIRSHORN; VOSS; BROMLEY, 2017), the SRR “examines the functional and performance requirements defined for the system and the preliminary program or project plan and ensures that the requirements and the selected concept will satisfy the mission”. And the International Council on Systems Engineering (INCOSE, 2023) emphasizes that the SRR is crucial for “ensuring that the set of requirements is complete, consistent, clearly stated and feasible”.

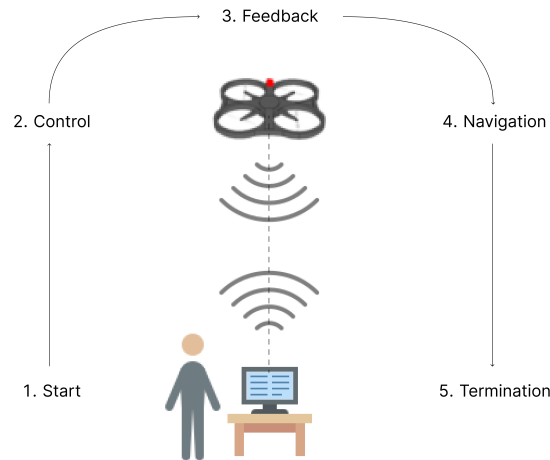
In the context of this research work, the SRR plays a role in defining the basic concept of operations and system requirements. The following subsections present detailed analysis and documentation for each of these SRR components. This systematic approach helps ensure that the final framework will effectively meet its intended goals properly documented before proceeding to the design phase.

3.1.1 Mission and ConOps

The Mission of this project is to develop a gesture-based drone control system that allows users to fly drones with hand movements instead of traditional controllers. The main goal is to make drone control easier and more accessible, especially for people with limited hand mobility. Following systems engineering methods, the framework integrates hand tracking technology to recognize gestures and convert them into precise drone commands. The system goal is to be adaptable for various applications, from personal use to professional tasks like search and rescue. The framework will be validated in simulation environments before real-world implementation to ensure safe operation.

The Concept of Operations (ConOps) outlines the operational interaction flow, describing how the system should work from a user perspective, as illustrated in Figure 14.

Figure 14 – Operational interaction flow defined in five steps.



Source: Created by the author.

This plan consists of five main steps:

1. **Start:** User setup the Ground Control Station (GCS) app, connects hand tracker, and powers on drone.
2. **Control:** User initiates the takeoff and performs hand gestures to control drone movements and operations.
3. **Feedback:** GCS shows real-time feedback status including altitude, speed, battery life, and possible errors.
4. **Navigation:** User modifies gestures based on visual feedback to achieve desired drone behavior.
5. **Termination:** User lands drone with appropriate command and shuts down system.

By dividing the operational flow into these distinct stages, the system maintains a clear interface between human commands and drone movements. But it is important to state that this ConOps only considers normal flight operational scenario, excluding emergency situations and different environmental conditions, regardless of real or simulated applications. It also does not take into account the need for user training, system maintenance, and potential future expansions of the system's capabilities.

3.1.2 Stakeholders

The development of the proposed framework involves several groups with different interests in the project's success. Understanding these stakeholders helps ensure that the system meets their needs and requirements, as stated in (INCOSE, 2023). The adopted organization for the key stakeholder groups is as follows:

- End-users (individuals with disabilities, drone operators, researchers)
- Industry (drone manufacturers, technology companies, rescue organizations)
- Academic institutions (universities, research labs, students)
- Regulatory bodies (aviation authorities, regulatory compliance organizations)
- Community (disability advocacy groups, general public, policy makers)

The relationships and influence levels between these stakeholders are shown in the Stakeholders Hub-and-Spoke Map and Stakeholders SVN Map (see A). These visualizations help understand how different groups interact and impact the project's development. This simplified understanding of stakeholders helps guide the development process while ensuring the final system serves its intended purpose and meets necessary requirements.

3.1.3 System Requirements

This section presents the core requirements for the framework. These requirements ensure safe and efficient operation while still meeting project goals. The base methodology from NASA Systems Engineering Handbook (HIRSHORN; VOSS; BROMLEY, 2017) suggests multiple requirement categories (e.g. functional, performance, interface, environmental, design, verification, safety), but this work will follow a separation based on simplified INCOSE standards (INCOSE, 2023), dividing requirements into functional and non-functional categories.

3.1.3.1 Functional Requirements

Functional requirements define specific behaviors and features the system must provide. They describe the core operations, inputs, outputs, and interactions between system components. According to INCOSE standards (INCOSE, 2023), these requirements focus on what actions the system performs, specifying tasks, services, and functions that users can directly observe and measure.

- **FR1** Gesture Recognition
 - **FR1.1** The system must detect and interpret hand gestures.
 - **FR1.2** The gesture recognition system should track hand positions within a defined three-dimensional workspace.
 - **FR1.3** The system should support the recognition of at least four distinct channel commands, including but not limited to pitch, roll, yaw and throttle.
- **FR2** Communication

- **FR2.1** The receiver must communicate the received commands to the flight controller.
- **FR2.2** The system must handle communication between the GCS and the drone receiver.
- **FR3** Flight Control
 - **FR3.1** The drone must execute all recognized commands.
 - **FR3.2** The flight control system must continuously monitor and adjust the drone's control.
- **FR4** User Feedback
 - **FR4.1** The system must provide real-time feedback about the status of command execution, potential errors, or safety alerts (e.g. signal loss).
 - **FR4.2** Visual confirmation should be shown in the GCS interface, related to gesture recognition and command transmission.

3.1.3.2 Non-Functional Requirements

Non-functional requirements define the system's operational qualities and constraints rather than specific behaviors. They address aspects such as performance metrics, reliability standards, safety measures, and environmental adaptability. These requirements shape how well the system performs its functions and often serve as key quality indicators (INCOSE, 2023).

- **NFR1** Performance
 - **NFR1.1** The system must detect and interpret hand gestures with a minimum accuracy of 95%, aligning with recent research results (LEE; YU, 2023; BEGUM; HAQUE; KESELJ, 2020; HU; WANG, 2020).
 - **NFR1.2** The total system response time from gesture to drone action must have minimal latency.
 - **NFR1.3** Must support continuous operation for typical drone mission of extended durations.
- **NFR2** Hardware
 - **NFR2.1** The system must operate within the constraints of the onboard processing unit without overloading.
 - **NFR2.2** If using a visual-based hand-tracking device, it must have a detection range ensuring reliability between 10 and 60 centimeters (ULTRALEAP, 2024).

- **NFR3** Safety
 - **NFR3.1** The system must incorporate safety protocols, including a fail-safe mode where the drone returns to a hover or lands in the event of signal loss.
 - **NFR3.2** The system must provide a manual override option for users to regain control through an alternate interface.

- **NFR4** Adaptability
 - **NFR4.1** The system architecture must be scalable for future enhancements, such as adding more complex gestures.
 - **NFR4.2** The design must support adaptability for different drone sizes and flight capabilities across various applications.

- **NFR5** Environment
 - **NFR5.1** The system must operate both indoors and outdoors under varying light levels and background conditions while maintaining gesture recognition accuracy and drone stability.

By adhering to these functional and nonfunctional requirements, the system will ensure a high level of performance, safety, and usability, establishing an overall foundation for the development and for compliance regulations.

3.1.4 Validation Criteria

The validation process focuses on verifying that the system meets all defined functional and non-functional requirements through tests in simulated and real environments. These tests examine five key areas: gesture recognition, simulation performance, system response time, accuracy, and comparison with traditional controls.

The gesture recognition system must fulfill the core functional requirements by accurately identifying and mapping hand gestures to drone commands. Tests will verify that the system can recognize at least five distinct gestures consistently across different users, as specified in FR1. The communication and response time requirements outlined in FR2 and FR3 will be validated through experiments. Real-world testing will examine the system's performance under actual operating conditions.

The final validation phase compares the gesture control system with traditional remote controls. This comparison evaluates whether the system meets non-functional requirements related to performance (NFR1), hardware compatibility (NFR2), safety features (NFR3), and environmental adaptability (NFR5). Success requires the gesture system to perform at least as well as traditional controls but offering improved accessibility.

Meeting these criteria will confirm that the system satisfies both functional and non-functional requirements while providing an effective platform for gesture-based drone control.

3.2 PRELIMINARY DESIGN REVIEW

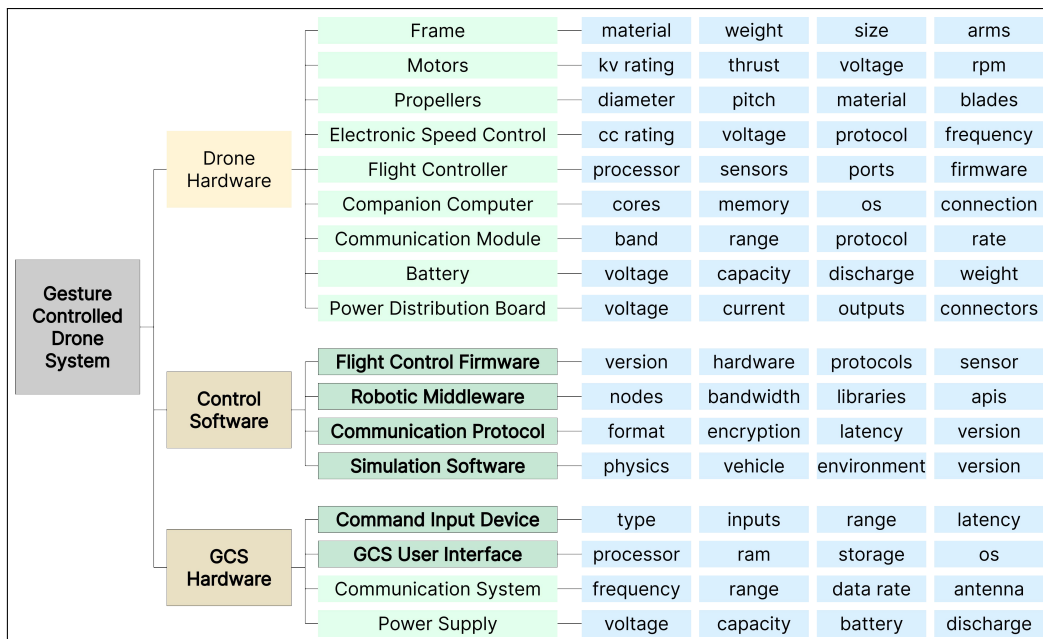
In this research work, the PDR methodology follows an adapted approach from the IEEE 15288.2 standard (IEEE, 2014; ISO, 2015) used on US Department of Defense (DoD). But here omitting cost estimates and program planning documentation, typically used for big-scale systems. It encompasses here six review products: Architecture Diagram, Multiple-Domain Matrix (MDM) Function Correlations, Morphological Matrices, Pugh Matrices, Final Prototype Configuration and Risk Management

The following subsections present an examination of each PDR component. This structured approach ensures an evaluation of design alternatives and selection of the most appropriate configuration to meet the established requirements.

3.2.1 Architecture Diagram

The architecture diagram in Figure 15 shows a structured breakdown of the system, each part contains its components with key attributes. Although the number of attributes is not restricted to four, that limit was set as sufficient to represent the main characteristics without over complicating the diagram.

Figure 15 – System Architecture Diagram.



Source: Created by the author.

The three main parts are the Drone Hardware (with Frame, Motors, Propellers, Electronic Speed Control, Flight Controller, Companion Computer, Communication Module,

Battery, and PDB), Control Software (with Flight Control Firmware, Robotic Middleware, Communication Protocol, and Simulation), and GCS Hardware (with Command Input Device, User Interface, Communication, and Power Supply).

This architectural decomposition serves as the foundation for analyzing component relationships in the MDM Function Correlations. The matrix utilizes the mapped components and their attributes to identify dependencies between different parts of the system. By tracking specific attributes like battery capacity, communication protocols, or processor specifications, the architecture ensures coverage of most important technical aspects that influence system performance.

3.2.2 MDM Function Correlations

The Multiple-Domain Matrix, introduced by (MAURER, 2007) in his doctoral thesis, combines several Dependency Structure Matrices (DSM) into one matrix, or as stated in his work “it represents a DSM on a higher level of abstraction”. It basically describes how different elements of a system interact and affect each other in the same and different domains. This matrix helps identify important interface connections between hardware, software, ground control systems (GCS), environmental factors, and most importantly the evaluation metrics used. The matrix reads from left to right, showing how items in the left column impact or relate to items across the top.

The matrix (see Table 2) uses "X" marks to show where components interact. For example, in the hardware domain, the drone’s motors connects to propellers and ESCs. The flight controller connects with both hardware (motors, sensors) and software (firmware, middleware) components, making it a key integration point. Environmental factors like weather and regulations impact several system components and evaluation metrics. The MDM reveals critical interfaces that need careful attention and understanding these relationships helps predict potential issues and plan appropriate testing procedures.

3.2.3 Morphological Matrices

A morphological matrix is a method used to generate a comprehensive set of possible solutions by systematically combining options for each component. To develop an effective SITL simulation prototype, the most relevant components from the architecture diagram were selected, which are the highlighted ones in Figure 15. Morphological matrices (see Tables 3 and 4) were created to explore all feasible combinations of these components.

Based on recent studies on multimodal drone control (ABIOYE, 2023), initial options were identified for each element. In the Drone Software stack, options included flight control firmware (PX4, ArduPilot, Betaflight), robotic middleware (ROS 1, ROS 2, DroneKit), communication protocols (DDS, Micro XRCE-DDS, MQTT), and simulation software (Gazebo, AirSim, jMAVSim), resulting in 81 potential combinations. For the Gesture Recognition stack, recognition methods (hand image, body image, wrist EMG)

Table 2 – Multiple-Domain Matrix Function Correlations Table.

		Drone								Software			GCS			Env.				Evaluation								
		Frame	Motors	Propellers	Electronic Speed Control	Flight Controller	Companion Computer	Communication Module	Battery	Power Distribution Board	Flight Control Firmware	Robotic Middleware	Communication Protocol	Simulation Software	Command Input Device	GCS User Interface	Communication System	Power Supply	Weather Conditions	Obstacle in Between	Geographical Terrain	Regulatory Constraints	Flight Time	Response Latency	Payload Capacity	Communication Range	Flight Stability	
Drone Hardware	Frame	X			X					X																	X	
	Motors		X																				X				X	
	Propellers			X																			X		X		X	
	Electronic Speed Control		X		X					X																		
	Flight Controller				X	X	X	X	X		X													X			X	
	Companion Computer				X	X	X	X			X	X	X	X														
	Communication Module				X		X	X				X			X	X										X		
	Battery				X	X			X	X													X					
	Power Distribution Board				X	X			X	X																		
Software	Flight Control Firmware				X					X													X				X	
	Robotic Middleware					X				X	X		X															
	Communication Protocol						X																		X			
	Simulation Software									X	X		X															
GCS	Command Input Device				X									X									X				X	
	GCS User Interface														X													
	Communication System						X				X					X	X						X	X	X	X		
	Power Supply														X	X		X					X					
Env.	Weather Conditions		X		X									X					X				X				X	
	Obstacle in Between						X												X						X	X		
	Geographical Terrain						X												X				X			X		
	Regulatory Constraints	X	X		X	X	X				X	X	X		X	X			X					X	X			
Evaluation	Flight Time		X		X	X	X							X	X	X				X			X					
	Response Latency				X	X				X	X	X	X		X	X	X		X				X				X	
	Payload Capacity	X	X																	X			X				X	
	Communication Range						X				X					X			X	X	X		X				X	
	Flight Stability				X					X	X	X	X		X				X									

Source: Created by the author.

and hardware (Leap Motion, RealSense, Kinect, Myo band) were considered, leading to 7 initial combinations.

However, not all options were suitable for the project objectives. To streamline the development process, certain options were promptly discarded:

- Gesture Recognition stack discarded:
 - Betaflight was removed because it is optimized for high-speed racing and lacks advanced autonomous features required for the simulation.
 - ROS 1 and DroneKit were excluded due to being older versions and having limitations in real-time performance and scalability.
 - DDS is resource-intensive and complex, while MQTT lacks necessary quality of service and drone protocols compatibility.

Table 3 – Morphological Matrix for Drone Software stacks.

Drone Software	Characteristc	Alternatives			Total
	Flight Control Firmware	PX4	ArduPilot	Betaflight	3
	Robotic Middleware	ROS 1	ROS 2	DroneKit	3
	Communication Protocol	DDS	Micro XRCE-DDS	MQTT	3
	Simulation Software	Gazebo	AirSim	jMAVSim	3

Source: Created by the author.

Table 4 – Morphological Matrix for Gesture Recognition stacks.

Recognition	Characteristc	Alternatives			Total	
	Recognition Method	Visual Hand		Visual Body	Bioelectrical wristband	3
	Recognition Hardware	Leap Motion Controller	Intel RealSense	Microsoft Kinect	Myo EMG Armband	4

Source: Created by the author.

- jMAVSim was removed because it lacks environmental complexity and sensor simulation fidelity.
- Gesture Recognition stack discarded:
 - The wrist EMG method with the Myo band was excluded due to its discontinuation and potential inconsistency in EMG signals across users.
 - Some combinations such as body image with Leap Motion or RealSense, and hand image with Kinect were also removed due to their specific application scope.

Table 5 – Final Concepts Table for Drone Software stacks.

Drone Software	Concept	Flight Control Firmware	Simulation Software
	DS-C1	PX4	Gazebo
	DS-C2	PX4	AirSim
	DS-C3	ArduPilot	Gazebo
	DS-C4	ArduPilot	AirSim

Source: Created by the author.

Table 6 – Final Concepts Table for Gesture Recognition stacks.

Recognition	Concept	Recognition Method	Recognition Hardware
	R-C1	Visual Hand	Leap Motion Controller
	R-C2	Visual Hand	Intel RealSense
	R-C3	Visual Body	Microsoft Kinect

Source: Created by the author.

By refining the options, the Drone Software stack combinations were reduced to four viable concepts (refer to Table 5). These configurations pair either PX4 or ArduPilot

with ROS 2, Micro XRCE-DDS, and either Gazebo or AirSim. Similarly, the Gesture Recognition stack was narrowed down to three concepts (see Table 6), focusing on hand image using Leap Motion, RealSense, or body image with Kinect.

By eliminating less suitable options, the choices were streamlined to the most promising configurations. This focused approach concentrates on combinations that are feasible and optimal for achieving a high-fidelity SITL simulation. The selected configurations will undergo further evaluation using Pugh matrices to determine the best setup for the prototype.

3.2.4 Pugh Matrices

The Pugh matrix, developed by Stuart Pugh (STUART, 1990), is a tool used to compare multiple design options against a set of criteria. It helps in decision-making by quantifying the strengths and weaknesses of each option relative to a baseline concept. For this project, Pugh matrices (Tables 7 and 8) were constructed for both the concept tables (Tables 5 and 6). The matrices compare the design concepts against key criteria, with assigned weights reflecting their importance on the MDM (Table 2).

Table 7 – Pugh Matrix for Drone Software stacks.

			Reference		Options	
	Objectives	Weight	DS-C1	DS-C2	DS-C3	DS-C4
Drone Software	Performance	4,0	0	-1	-1	-1
	Community	2,0	0	-1	0	-1
	Compatibility	3,0	0	0	1	0
	Fidelity	1,0	0	1	0	1
	Stability	3,0	0	-1	0	-1
Weighed Positives			0	1	3	1
Weighed Negatives			0	-6	-4	-6
Overall			0	<0	<0	<0

Source: Created by the author.

Table 8 – Pugh Matrix for Gesture Recognition stacks.

			Reference		Options	
	Objectives	Weight	R-C1	R-C2	R-C3	
Recognition	Accuracy	4,0	0	1	-1	
	User Comfort	4,0	0	0	-2	
	API	5,0	0	-1	-1	
	Cost	2,0	0	-1	1	
	Future-Proofing	3,0	0	0	-2	
Weighed Positives			0	4	2	
Weighed Negatives			0	-7	-17	
Overall			0	<0	<0	

Source: Created by the author.

In the Drone Software stack Pugh matrix, the criteria included Performance (4), Community (2), Hardware Compatibility (3), Fidelity (1), and Learning Curve (3). After

the grades placed based on author's experience, the first concept, DS-C1, emerged as the most favorable option, consistently performing well across critical criteria. Its high performance, strong community support, good hardware compatibility, and moderate learning curve contributed to its overall superiority. While alternatives offered advantages in certain areas, none surpassed DS-C1 when considering all weighted criteria.

Similarly, the Gesture Recognition stack Pugh matrix utilized criteria such as Accuracy (4), User Comfort (4), API (5), Cost (2), and Future-Proofing (3). The first concept, R-C1, stood out due to its high accuracy and user comfort, which are crucial for intuitive gesture control. Its robust API support facilitated easier integration and development. Although other options had certain benefits, R-C1's overall performance across all criteria made it the optimal choice. By systematically comparing the options using the Pugh matrices, the most suitable configurations were identified, ensuring an effective SITL simulation prototype.

3.2.5 Final Configuration

After evaluating the design options using the Pugh matrices, the optimal configurations for both the Drone Software stack and the Gesture Recognition stack were determined. The chosen Drone Software consists of PX4 as the flight control firmware, ROS 2 as the robotic middleware, Micro XRCE-DDS as the communication protocol, and Gazebo as the simulation software. And for the Gesture Recognition visual hand recognition using the Leap Motion Controller.

These final configurations ensure an effective SITL simulation prototype, leveraging the strengths of each component to meet the project's objectives. This dual-stack approach creates a modular system where components can be modified or upgraded independently. Research from (LEE; YU, 2023) supports the effectiveness of this architectural approach for gesture-based drone control systems.

3.2.6 Risk Management

The development of a gesture-based drone control system requires careful management of potential risks to ensure project success. This section presents key risks identified across technical, operational, and integration domains, along with strategies to address them. The risks are classified by severity based on their potential impact on system functionality and safety: high (H), medium (M), or low (L).

Three main categories of risks were identified. Technical risks include gesture recognition accuracy problems, hardware failures, and communication issues between system components. Operational risks involve user interaction challenges and environmental factors that could affect system performance. Integration risks encompass compatibility issues between components and limitations of the simulation environment.

- **Technical Risks**

- Gesture Recognition Inaccuracies (H): The accuracy of hand gesture interpretation affects system safety. To mitigate this, testing and calibration procedures should be performed in varied conditions and with different users.
- Hardware Component Failures (M): Critical components like tracking devices or flight controllers may fail during operation. System implements safe protocols that automatically hover or land the drone until repairs are made.
- Communication Delays (H): Delays between components can compromise real-time control. Implementation of optimized transmission protocols and error checking mechanisms help minimize latency.

- **Operational Risks**

- User Gesture Misinterpretation (M): Users may perform gestures incorrectly leading to unintended commands. User training programs and real-time visual feedback help ensure proper gesture execution.
- Environmental Interference (L): Poor lighting, heat or visual background conditions could affect tracking accuracy. Alternative control methods serve as backup when environmental conditions degrade system performance.

- **Integration Risks**

- Component Compatibility (M): Integration between hardware and software components may cause conflicts. A modular design approach allows incremental testing and easier component replacement if needed.
- Simulation Limitations (L): The simulation environment may not fully replicate real conditions. Additional real-world testing validates system performance beyond simulation constraints.

Risk management involves continuous monitoring and periodic assessment of these identified risks. High-severity risks receive priority attention and resource allocation. The project implements regular testing and validation procedures to identify new risks and evaluate the effectiveness of mitigation strategies, ensuring system safety and reliability throughout development.

3.3 GESTURE RECOGNITION IMPLEMENTATION

The implementation of gesture recognition capabilities in this research began with a validation phase, focusing on establishing a reliable foundation for hand tracking before integration with the drone control system. This methodical approach allowed an isolated

testing and validation of the gesture recognition component, checking its robustness before its implementation within the broader system architecture.

The gesture recognition algorithm was developed utilizing the Ultraleap LeapC Python bindings, accessed through their official GitHub repository (leapc-python-bindings), to interface with the Leap Motion Controller. This choice was made because the LeapC Python bindings provide low-level access to the Leap Motion Controller hardware, enabling precise control over data acquisition and processing.

3.3.1 Hand Tracking Data Structures

Among the various hand tracking properties available through the Ultraleap API, palm position and orientation were selected as the primary data structure metrics for the recognition. These fundamental data structures are defined as follows:

1. **Palm Position** (based on LEAP_VECTOR struct)

Describes the palm's center position using three components vector $\mathbf{v}[3]$: x , y , z (coordinates portion measured in millimeters relative to camera's origin).

2. **Palm Orientation** (based on LEAP_QUATERNION struct)

Describes the palm's orientation using four components vector $\mathbf{v}[4]$: x , y , z (vector portion measured in radians relative to the normal vector) and w (scalar portion).

The palm data is encapsulated in the LEAP_PALM structure, which provides additional properties such as stabilized position for smoother tracking, velocity measurements, normal vector (pointing outward from the palm surface), direction vector (pointing from palm toward fingers) and palm width measurements.

The LEAP_HAND object contains the LEAP_PALM, which includes the position and orientation properties. These combined data enable precise tracking of hand movements and gesture interpretation in three-dimensional space, forming the foundation for the gesture-to-command mapping system described in following section.

3.3.2 Initial Setup and Testing

The initial setup involved creating a standalone testing environment to verify hand tracking capabilities without connecting to ROS. This included installing the Leap Motion Controller drivers, setting up the Python development environment, and implementing basic hand tracking functionality. The tracking mode was set to desktop for appropriate camera positioning. The sample code snippet in Listing 3.1 illustrates the setup.

Listing 3.1 – Initial Setup Code

```
1 canvas = Canvas()  
2
```

```
3 tracking_listener = TrackingListener(canvas)
4
5 connection = leap.Connection()
6 connection.add_listener(tracking_listener)
7
8 with connection.open():
9     connection.set_tracking_mode(leap.TrackingMode.Desktop)
10    canvas.set_tracking_mode(leap.TrackingMode.Desktop)
11
12    cv2.imshow(canvas.name, canvas.output_image)
```

This preliminary testing focused on tracking precision, update rate, and system stability, confirming the feasibility of using the Leap Motion Controller as the primary input device.

3.3.3 Mapping Gesture to Drone Commands

Mapping hand gestures to drone control commands involved creating intuitive correlations between hand movements and drone actions. The implementation directly associates specific components of the palm's position and orientation with the four primary drone control channels: Throttle, Pitch, Roll, and Yaw.

- **Throttle Control** (Up/Down Movement)

Mapped to the palm's position along the Y-axis (`palm.position.y`).

Vertical movement of the hand controls the drone's altitude:

- Raising the hand increases altitude (positive values).
- Lowering the hand decreases altitude (negative values).

- **Pitch Control** (Forward/Backward Movement)

Mapped to the palm's orientation around the X-axis (`palm.orientation.x`).

Tilting the hand controls the drone's forward or backward movement:

- Tilting the hand forward moves the drone forward.
- Tilting the hand backward moves the drone backward.

- **Roll Control** (Left/Right Movement)

Mapped to the palm's orientation around the Z-axis (`palm.orientation.z`).

Tilting the hand controls the drone's lateral movement:

- Tilting the hand left moves the drone to the left.
- Tilting the hand right moves the drone to the right.

- **Yaw Control** (Rotational Movement)

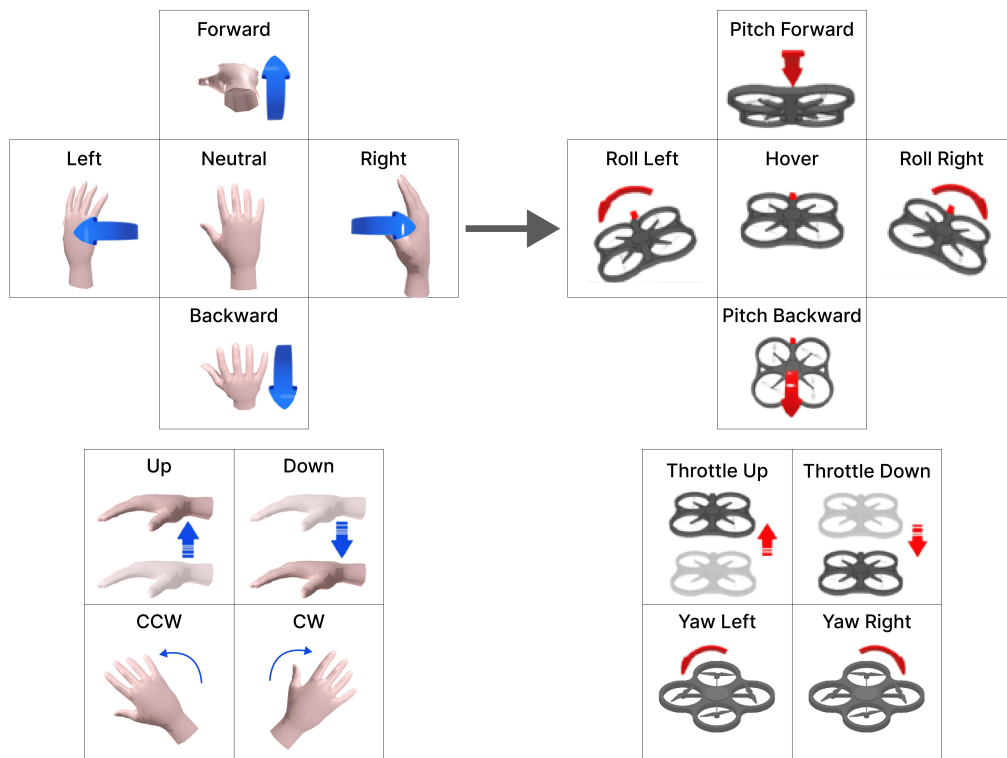
Mapped to the palm's orientation around the Y-axis (`palm.orientation.y`).

Rotating the hand controls the drone's yaw:

- Rotating the hand clockwise yaws the drone to the right.
- Rotating the hand counterclockwise yaws the drone to the left.

Figure 16 illustrates the mapping of hand gestures to the corresponding drone commands, providing a visual representation of the implemented controls.

Figure 16 – Hand Gestures Mapped to Drone Commands.



Source: Created by the author.

3.3.4 Gesture Threshold Calibration

The conversion of continuous gesture input data into discrete drone commands required evaluation of different control strategies. Two main approaches were considered for converting continuous gesture inputs into drone commands: stepped functions and linear mapping.

The stepped function approach divides the input range into distinct zones, with each zone corresponding to a specific command output value. When gesture measurements reach predetermined thresholds, discrete command values are triggered. For example, hand

movements exceeding 150mm could trigger maximum speed commands, while movements between 50mm and 150mm would generate moderate speed commands.

In contrast, a linear mapping approach would create a direct proportional relationship between input gestures and output commands. This method would translate hand position or orientation directly into command values using scaling factors. However, testing revealed that linear mapping made precise control difficult due to its sensitivity to small input variations.

3.4 SIMULATION SETUP FOR SITL EXECUTION

The implementation and validation of the gesture-based drone control system requires a configured simulation environment that integrates multiple software components and frameworks. This section details the system integration process and the development of a simulation framework based on the previously defined robotics middleware and flight control software. The implementation builds upon previous work from the ETH Zurich's research community, specifically adapting the offboard control framework developed by (LIM, 2023) for PX4 autopilot systems.

3.4.1 Development Environment Configuration

The simulation environment implementation began with the installation of Ubuntu 22.04 as the operating system foundation. This choice was driven by its compatibility with both ROS 2 Humble and PX4 version 1.14.

The first step involved preparing the operating system and installing ROS 2 Humble. This robotics middleware provides the communication framework for integrating gesture recognition with drone control. The process begins with system preparation through package repository updates, where commands like `sudo apt update` ensure system currency.

The next phase focused on establishing the PX4 Autopilot environment, which handles flight control simulation. PX4 implementation required specific Python dependencies for proper functionality, such as `empy` and `pyros-genmsg` packages.

The development workspace creation followed a structured approach using standardized ROS 2 workspace organization. This organization facilitates package management and system integration testing. The complete installation process involves multiple configuration steps, with the full setup instructions provided in Appendix B.

Gemini LeapC API installation provided the interface for hand gesture tracking, requiring Python bindings compilation from source code. The installation process used Python build tools, with commands such as `python -m build leapc-cffi`, followed by package installation through `pip`.

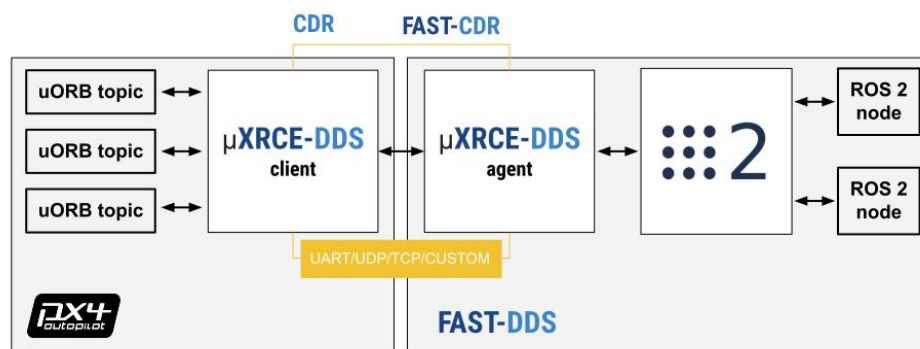
The Micro XRCE-DDS Agent installation bridges ROS 2 with the PX4 autopilot. The process involves building the agent using `CMAKE` and configuring system libraries.

The final component, Gazebo simulation engine, completed the environment setup. The complete setup instructions are provided in Appendix B.

3.4.2 Communication Architecture

The communication architecture implements a publish-subscribe pattern through ROS 2 nodes. This design allows for modular component interaction while maintaining system reliability. The architecture consists of three primary communication channels: gesture data transmission, flight control commands, and system state feedback.

Figure 17 – Architecture uXRCE-DDS with ROS 2



Source: (PX4 DEVELOPMENT TEAM, 2024).

The gesture recognition node publishes processed hand movement data to dedicated ROS 2 topics. These topics use custom message types defined in the `px4_msgs` package, ensuring compatibility with the PX4 flight stack. The following code snippet in Listing 3.2 illustrates the basic structure of a ROS 2 node for handling gesture commands:

Listing 3.2 – GestureControlNode class defined using ROS 2 node structure

```

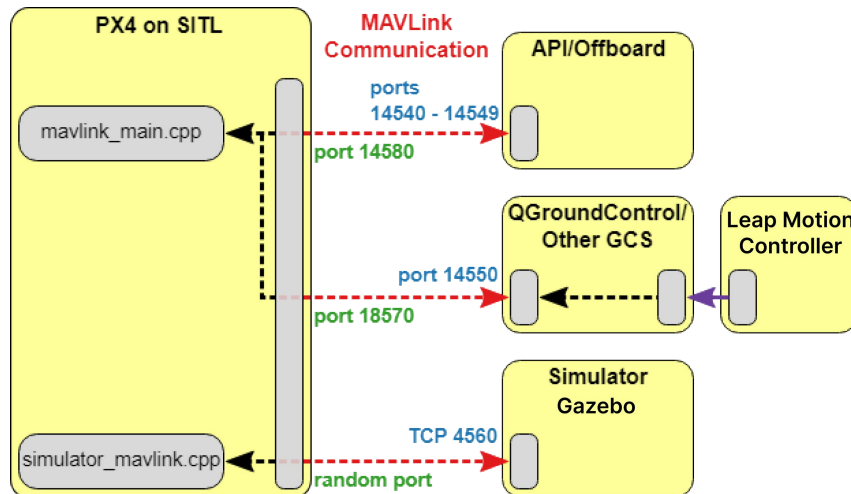
1 class GestureControlNode(Node):
2     def __init__(self):
3         super().__init__('gesture_control_node')
4
5         # Publishers for PX4 control
6         self.offboard_control_mode_publisher = self.create_publisher(
7             OffboardControlMode,
8             '/fmu/in/offboard_control_mode',
9             10)
10        self.trajectory_setpoint_publisher = self.create_publisher(
11            TrajectorySetpoint,
12            '/fmu/in/trajectory_setpoint',
13            10)
14
15        # Timer for control loop
16        self.timer = self.create_timer(0.1, self.control_loop)

```

The MAVLink communication architecture defines how messages are exchanged between various components in the simulation and control ecosystem. As shown in Fig-

Figure 18, MAVLink streams are set up using specific ports dedicated to particular endpoints. For instance, the SITL simulation environment runs PX4 and uses a defined port (e.g., 14550) to exchange telemetry and command messages with GCS or other offboard APIs. Additional ports are reserved to connect to external tools, like Gazebo simulator.

Figure 18 – SITL communication environment using MAVLink



Source: Adapted from (PX4 DEVELOPMENT TEAM, 2024).

3.5 CHAPTER REMARKS

This chapter detailed the methodological approach for developing a gesture-based drone control framework, following systems engineering principles through the SRR and PDR phases. The SRR established core requirements and validation criteria, while the PDR determined optimal system configurations through systematic analysis tools.

The implementation phase focused on two key components: gesture recognition using the Leap Motion Controller and SITL simulation setup using ROS 2 with PX4. The gesture recognition implementation defined threshold-based mapping between hand movements and drone commands, while the simulation environment integrated multiple software frameworks to enable system testing. This dual focus on both control interface and testing environment provides a foundation for validating the framework's effectiveness in Chapter 4.

4 RESULTS AND DISCUSSION

This chapter presents the outcomes and discusses their implications. The work completed represents the progress achieved during the ongoing PDR stage of the project development cycle. It is important to note that the primary focus was not to close the full cycle validate the requirements established during the SRR. Instead, this research aimed to address the core research objectives defined in the Chapter 1, laying the groundwork for future development phases.

The research serves as a step toward developing a complete system with a fully implemented life cycle. But a complete verification and validation of the requirements would only be possible after reaching later stages, which include hardware field testing, usability assessment with different user groups, measurement of system performance metrics under various conditions, analysis of stakeholders opinions, and thorough evaluation of safety protocols in real-world scenarios. These additional validation steps would provide comprehensive data about the system's effectiveness and reliability in practical applications.

The following sections examine the results obtained from the gesture input calibration, SITL implementation and a brief overview on a hardware prototype version built. The discussion includes an analysis of the chosen technical approach, with a comparison showing advantages and limitations, possible use case scenarios and potential areas for improvement. By examining these aspects, we can better understand the system's current capabilities and identify paths for future development.

4.1 IMPLEMENTATION RESULTS

The implementation produced three main outcomes from the development process described in Chapter 3. The results include optimal threshold values for gesture recognition, SITL simulation testing culmination, and a basic hardware prototype option proposal statement. These tests were performed on a PC equipped with an Intel Core i5-12450H 2GHz 8-core processor, 64GB of 4800MHz SODIMM RAM, and running a Windows Subsystem for Linux (WSL) environment. This multi-faceted evaluation helps assess the framework's current capabilities and highlights paths for future development. The following sections examine these outcomes in detail.

4.1.1 Optimal Input Threshold Values

The system uses threshold values to convert hand positions into drone commands. These values were set through testing to find the best balance between control sensitivity and stability. The control system uses two threshold levels for most movements, creating five possible command outputs (-2.0, -1.0, 0.0, 1.0, 2.0). These outputs correspond to maximum movement, medium movement, neutral position, medium reverse movement,

and maximum reverse movement. A agnostic version of this mapping is shown in the code snippet shown in Listing 4.1.

Listing 4.1 – Example of stepped function for agnostic channel control

```
1 if position > threshold_2:
2     command = 2.0 # Maximum positive
3 elif position > threshold_1:
4     command = 1.0 # Medium positive
5 elif position > -threshold_1:
6     command = 0.0 # Neutral position
7 elif position > -threshold_2:
8     command = -1.0 # Medium negative
9 else:
10    command = -2.0 # Maximum negative
```

The final threshold values were determined through testing:

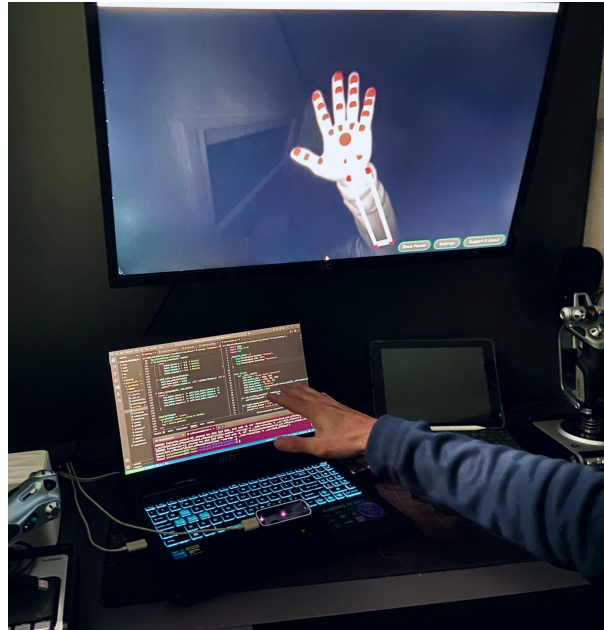
- Initial neutral position:
 - Height (Z-axis): 250.0 mm
 - Roll angle: -5.0 degrees
 - Pitch angle: 5.0 degrees
 - Yaw angle: 0.0 degrees
- Movement thresholds:
 - Higher linear movement threshold: 150.0 mm
 - Medium linear movement threshold: 50.0 mm
 - Higher angular movement threshold: 10.0 degrees
 - Medium angular movement threshold: 20.0 degrees

The height of 250 mm was chosen as the neutral position because through iterations it was found to be comfortable to maintain. The slight adjustments in roll (-5 degrees) and pitch (5 degrees) angles account for the natural resting position of the wrist (see Figure 19 for neutral hand position in Ultraleap Gemini execution).

For vertical control (throttle), movement thresholds of 50 mm and 150 mm allowed comfortable hand motions. For rotation controls (pitch, roll, yaw), the 10-degree and 20-degree thresholds matched natural wrist movement limits. The system only processes right-hand inputs, ignoring any left-hand movements. For safety, the throttle down command (negative Z-axis) uses only one threshold level to prevent rapid descent.

These thresholds create three control zones: neutral (no movement), moderate speed (between thresholds), and maximum speed (beyond second threshold). The system processes only right-hand gestures, with linear thresholds controlling altitude and angular thresholds managing orientation. This configuration balances control precision with natural hand movements.

Figure 19 – Neutral hand position in the tracking software.



Source: Created by the author.

4.1.2 ROS Package and SITL Execution

The developed framework was organized into a ROS 2 package structure and made publicly available through GitHub as an open-source project (ACHERMANN, 2024). The package includes all necessary dependencies, launch files, and configuration parameters required for the package. It also includes a `README.md` file with installation and execution documented instructions. The SITL execution environment launches through a single ROS command in a Linux terminal as shown in Figure 20, which initializes multiple components:

Figure 20 – Linux terminal with ROS command to start execution.

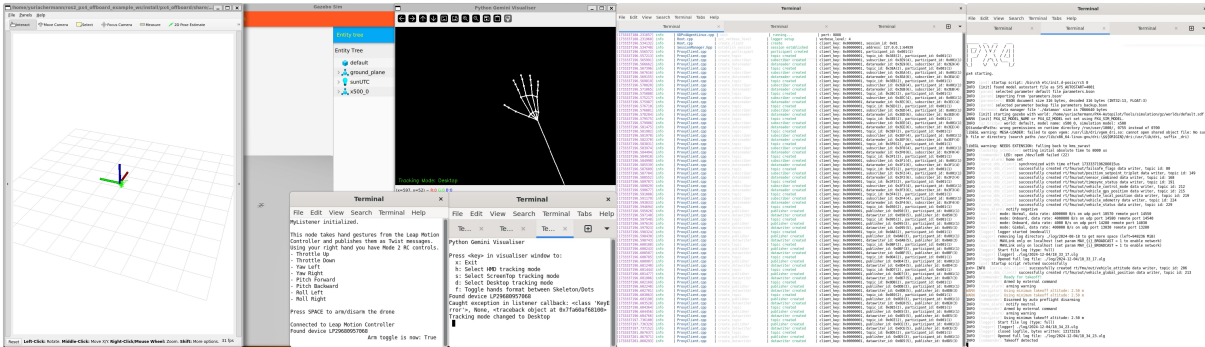
```
Windows PowerShell x yuriachermann@MSI-Yuri: ~/ x + v
(base) yuriachermann@MSI-Yuri: $ cd ros2_px4_offboard_example_ws/
(base) yuriachermann@MSI-Yuri:~/ros2_px4_offboard_example_ws$ source install/setup.bash
(base) yuriachermann@MSI-Yuri:~/ros2_px4_offboard_example_ws$ ros2 launch px4_offboard offboard_velocity_control.launch.py
```

Source: Created by the author.

- Four terminal windows displaying:
 - ROS topic messages and node communication logs
 - PX4 flight stack connection status
 - Python Gemini Visualizer interface for system status
 - MyListener class output for control verification
- A visualization window showing real-time hand tracking wireframe

- RViz2 for ROS environment visualization
- Gazebo simulator for drone physics and environment rendering

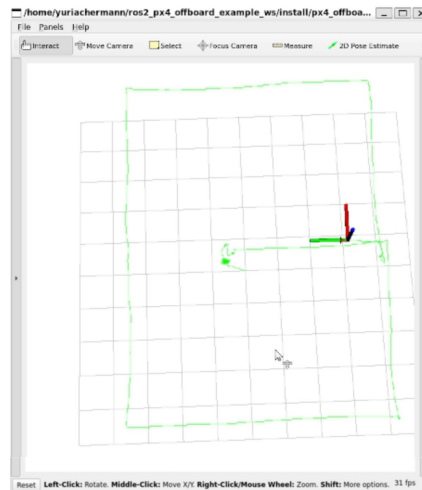
Figure 21 – Compilation of all the windows that open with SITL launch command.



Source: Created by the author.

Figure 21 shows the complete SITL execution environment with all components running simultaneously. This integration enables comprehensive system testing and validation. To validate the control system's precision, a rectangular path following test was conducted. As illustrated in Figure 22, the drone was commanded to follow a rectangular trajectory using only the hand gestures.

Figure 22 – Drone path defined to assert navigability and sensitiveness.



Source: Created by the author.

The test results demonstrate the system's ability to maintain stable control while following predefined paths, supporting the framework's potential for practical applications. The implementation maintains a modular structure, separating the gesture recognition, command processing, and drone control components. This design choice, facilitates system testing and future modifications. The open-source nature of the project aligns with the importance of accessible and reproducible research in HCI.

4.1.3 Hardware Prototype

A hardware prototype was assembled to demonstrate the potential for real-world implementation of the framework, though no physical testing was conducted during this research period. Figure 23 shows the assembled quadcopter drone. The prototype uses a Holybro Pixhawk 6X flight controller, which can be integrated with the developed control framework as any board that runs PX4 firmware. The complete drone configuration includes:

- **Quadcopter frame:** F450
- **FC:** Holybro Pixhawk 6X
- **Motors:** EMAX RS2205 RaceSpec
- **ESCs:** RaceStar RS30A V2
- **GPS module:** Holybro M10
- **Power module:** Holybro PM06D
- **FPV Camera:** RunCam Swift 2
- **Depth Camera:** Intel RealSense D435i
- **Companion Computer:** ASUS NUC 14 Pro Revel Canyon

Figure 23 – Drone Hardware Prototype.



Source: Created by the author.

This hardware setup represents one possible configuration for testing the gesture control framework. The framework should work with any PX4-compatible flight controller board, making it adaptable to different drone platforms. Future work will involve testing the gesture control system on this physical prototype to validate the simulation results.

4.1.4 Analysis of Requirements Validation

This section evaluates how the implemented system fulfilled the functional requirements established during the SRR phase. While the SRR defined both functional and non-functional requirements, this analysis focuses only on functional requirements. That is because the evaluation of non-functional requirements would not provide meaningful results at the current development stage, as it would require extensive real-world testing beyond the scope of the current SITL simulation. This analysis examines each requirement group based on the system implementation outcomes, focusing on the four main functional requirement categories.

- **FR1 - Gesture Recognition:** The Leap Motion Controller implementation fulfilled the core gesture recognition requirements. The system detected hand movements during testing. The system successfully implemented all four primary control channels (pitch, roll, yaw, throttle), with hand position data mapped to corresponding flight commands.
- **FR2 - Communication:** The communication requirements showed mixed results. The ROS 2 implementation established reliable command transmission between components, with the system maintaining stable connections during testing sessions. However, occasional usability latency was felt during high-frequency command updates, suggesting room for optimization in the communication pipeline.
- **FR3 - Flight Control:** Flight control requirements achieved partial fulfillment in the SITL environment. The PX4 flight stack executed recognized commands consistently, demonstrating proper integration with the gesture recognition system. The parameter monitoring system maintained stable flight navigation.
- **FR4 - User Feedback:** The Ground Control Station interface met basic feedback requirements. While the system provided status updates through the terminal logs and command confirmation, using visual feedback implemented in a GUI (Graphical User Interface) to improve user understanding of gesture recognition states. The error reporting system could successfully detected communication issues, though the alert presentation may require further refinement for clarity.

The framework implementation demonstrated core functionality across all requirement categories. Areas needing additional development include communication latency, edge case handling in flight control and enhanced visual feedback systems. The results suggest that while the basic framework meets fundamental requirements, further refinement could improve system robustness and user experience.

4.2 RESEARCH DISCUSSION

This section provides a critical analysis focused on four main aspects: a comparison with traditional control approaches, potential use cases for the technology, current limitations, challenges, and opportunities for future development. The discussion synthesizes findings from both the software implementation and preliminary hardware testing phases to understand the system's real-world potential.

4.2.1 Comparison With Traditional Control Methods

The comparison between gesture-based control and traditional radio control (RC) systems reveals several key differences in operation, accessibility, and user interaction. Traditional RC controllers, such as the FrSky Taranis Q X7 model mentioned in the Chapter 2, require both hands for operation and include multiple switches, knobs, and sticks. The gesture control system developed in this research presents an alternative approach using single-hand movements for drone control.

- **Physical Interface Differences:** Traditional RC systems use two main control sticks, with the left stick managing throttle and yaw while the right stick controls pitch and roll. This control method requires constant two-handed operation and finger dexterity for precise adjustments. In contrast, the gesture control system allows users to control all four channels using one hand's position and orientation in space, as tracked by the Leap Motion Controller.
- **Learning Curve and Intuitiveness:** The research by (FERNANDEZ et al., 2016) indicates that gesture-based interfaces often prove more intuitive for new users, as they mirror natural hand movements. Traditional RC controllers demand significant practice to develop muscle memory for coordinated stick movements. However, experienced RC pilots often achieve higher precision with traditional controllers due to their familiarity with the physical feedback from control sticks.
- **Control Precision:** Tests conducted in the SITL environment showed that traditional RC systems currently offer more precise control due to their physical resistance and defined movement limits. The gesture system's dual-threshold approach, while functional, did not match the fine-grained control possible with physical sticks. This finding aligns with research by (HU; WANG, 2020), who also noted precision limitations in gesture-based systems.
- **Accessibility and Mobility:** The primary advantage of the gesture control system lies in its accessibility. Users with limited mobility in one hand can operate the drone effectively, whereas traditional RC controllers require both hands. This improvement in accessibility supports findings from (FEUERRIEGEL et al., 2021),

who emphasized the importance of developing more inclusive control interfaces for drone operations.

- **Technical Limitations:** Both systems face different technical challenges. Traditional RC systems may experience signal interference but maintain consistent control feel. The gesture system depends on the tracking device's field of view and can be affected by lighting conditions or hand positioning. These limitations reflect observations made by (LEE; YU, 2023) regarding environmental factors in gesture recognition systems.

This comparison demonstrates that while gesture control offers improved accessibility and intuitive operation, traditional RC systems currently maintain advantages in control precision and reliability. These findings suggest that gesture control systems may serve as a complementary rather than replacement technology, particularly beneficial for applications prioritizing accessibility over maximum control precision.

4.2.2 Potential Applications and Use Cases

The comparison between gesture-based and traditional radio control methods reveals several key differences in operational characteristics and user interaction patterns. Traditional radio controllers, such as the widely used dual-stick transmitters, have been the standard method for drone operation. These controllers provide precise input with each stick controlling specific flight parameters. In contrast, gesture-based control systems translate hand movements into flight commands.

The physical design of traditional controllers presents both advantages and limitations. Radio controllers offer tactile feedback, helping pilots maintain awareness of input levels without visual confirmation. (SARKAR et al., 2016) note that this tactile feedback becomes particularly important during first-person view (FPV) flight, where visual attention focuses on the drone's camera feed. Gesture control systems lack this inherent physical feedback, though some implementations incorporate visual or haptic feedback mechanisms to compensate.

Accessibility represents a significant difference between the two control methods. Traditional controllers require both hands for operation, limiting their use by individuals with disabilities or those who need to perform other tasks while controlling the drone. The gesture control system developed in this research enables single-handed operation, making drone control more accessible to a broader range of users.

System reliability differs between the two approaches. Traditional radio controllers have proven reliability in various environmental conditions, with established fail-safe mechanisms and predictable performance limitations. Gesture recognition systems face additional challenges from environmental factors such as lighting conditions and background interference. However, recent developments in tracking technology, as demonstrated

by (LEE; YU, 2023), show improved reliability in gesture recognition systems through machine learning implementations.

Cost considerations also differ between the two methods. Traditional radio controllers represent a mature technology with established manufacturing processes and relatively stable costs. Gesture control systems often require specialized tracking hardware and more complex processing capabilities, potentially increasing initial system costs. However, the declining cost of tracking technology and processing hardware suggests this gap may narrow over time.

This comparison indicates that while traditional radio controllers maintain advantages in certain operational scenarios, gesture-based control systems offer unique benefits in accessibility and user interaction. The choice between control methods ultimately depends on specific use cases and operator requirements, with each approach showing distinct strengths in different operational contexts.

4.2.3 Limitations And Challenges Encountered

The development and implementation of the gesture-based drone control system encountered several technical and practical limitations. The most significant challenge emerged from hardware constraints when running the simulation environment.

The use of WSL created notable performance limitations for system testing. WSL lacks proper GPU support, forcing all computations to run on the CPU. This hardware constraint made it impossible to execute the Gazebo simulator effectively, as Gazebo requires substantial graphics processing capabilities for real-time 3D rendering and physics calculations.

The inability to conduct full simulation testing affected the validation process of the gesture recognition system. Without access to a complete simulation environment, the evaluation of the system's real-time performance and response characteristics remained partial. This limitation aligns with findings from (BEGUM; HAQUE; KESELJ, 2020), who note that testing in a simulated environment forms a crucial step before real-world implementation of gesture-controlled systems.

Additional technical challenges included processing delays in gesture recognition due to computational limitations, reduced frame rates affecting the smoothness of hand tracking and limited ability to test complex flight scenarios. These limitations suggest that future implementations should prioritize native Linux installations or alternative simulation platforms that can function effectively without GPU acceleration.

4.2.4 Future Improvements And Research Directions

The development of gesture-based drone control systems presents several paths for future research and technical improvements. These directions emerge from both the

limitations encountered in current implementations and the evolving capabilities of gesture recognition and drone technologies.

Simulation environments require expansion to better represent real-world flying conditions. Future work should implement various virtual testing scenarios, including obstacle courses and defined flight paths. This testing would help measure how well users can navigate drones through complex environments using hand gestures. Creating standardized test circuits would allow researchers to compare different control methods by measuring completion times and precision metrics.

The relationship between hand movements and drone responses needs further investigation. The current implementation uses fixed thresholds, but future research could explore progressive output mapping. This could involve testing both proportional and exponential relationships between hand position and drone movement speed. These alternative mapping approaches might provide more natural control, especially for precise movements requiring fine adjustments.

Control system optimization presents another important research direction. Different PID (Proportional-Integral-Derivative) controller combinations could be tested to find the best balance between responsiveness and stability. This testing would help determine optimal control parameters for various flight conditions and user skill levels.

Quantitative assessment of hand recognition accuracy remains crucial for system validation. Future work should develop standardized testing protocols to measure recognition accuracy across different users and environmental conditions. This data would help identify areas where gesture recognition needs improvement and guide the development of more robust tracking algorithms.

The broader adoption of gesture-controlled drones depends on advances in both gesture recognition and drone technology. As noted by (LEE; YU, 2023), improvements in sensor technology and machine learning algorithms will enhance the reliability of gesture recognition. Similarly, developments in drone flight controllers and safety systems will make gesture-based control more practical for everyday use.

Integration with emerging technologies also deserves exploration. For example, (ZHONG et al., 2024) discuss the potential of combining gesture control with autonomous planning systems. This integration could create hybrid control schemes that combine the intuitiveness of gesture control with the safety benefits of autonomous systems.

These research directions aim to address current limitations while preparing for future applications of gesture-based drone control. Each improvement path contributes to the overall goal of making drone control more accessible and reliable for various user groups.

5 CONCLUSION

This research focused on developing a framework for gesture-based drone control, addressing the need for more accessible and intuitive drone interfaces. It also documented the systematic development of a control system that enables single-handed drone operation through natural hand movements, validated in a SITL environment.

The work produced key findings about gesture recognition, system integration, and human-drone interaction that advance our understanding of natural control interfaces for unmanned aerial vehicles. The following summary addresses the four research questions established at the beginning of this work, examining how the results contribute to our understanding of gesture-based drone control systems.

5.1 SUMMARY OF FINDINGS AND CONTRIBUTIONS

The following summary examines how the results contribute to this field of knowledge addressing the four research questions established at the beginning of this work:

1. The first research question explored how hand gestures could be effectively translated into drone control commands. The work demonstrated that through elaborated mapping of hand position and orientation data, combined with a dual-threshold approach, precise drone control could be achieved. The implementation used palm position for altitude control and hand orientation angles for directional movements, creating an intuitive correlation between gesture and drone response.
2. The second research question investigated the technical challenges in implementing a gesture-controlled drone system. The findings revealed that the primary challenges centered around system integration and real-time performance. Communication structure between components presented a significant hurdle, which was addressed through the integration of ROS 2, Micro XRCE-DDS and px4_msgs middleware.
3. The third research question examined how gesture-based control enhances human-drone interaction compared to traditional methods. The final analysis showed that gesture control offers improved accessibility through single-handed operation and provides a more intuitive learning experience for new users. However, we also found that the system required careful calibration of recognition thresholds to balance control sensitivity with stability, highlighting both the advantages and limitations of this approach.
4. The fourth research question focused on potential applications and future research directions for gesture-controlled drones. The research identified several promising applications, ranging from professional settings, in industrial inspections that could benefit from intuitive control for improved response times, while search and rescue

operations gain advantages from single-handed operation. In the entertainment industry, gesture control creates an appealing visual performance experience for audiences, making drone spectacles more engaging when operators control flight patterns through hand movements.

Future research opportunities include developing adaptive gesture recognition algorithms, implementing haptic feedback systems, and expanding the framework to support more complex flight maneuvers.

5.2 IMPLICATIONS TO THE FIELD OF HCI FOR DRONES

This research advances the field of human-drone interaction in several ways. First, it provides a documented approach to implementing gesture recognition for drone control, offering future researchers a foundation for developing similar systems. Second, it demonstrates the viability of using natural interfaces for precise drone control, challenging traditional assumptions about the necessity of physical controllers.

The findings suggest that future human-drone interaction systems should consider:

- The importance of intuitive mapping between user movements and drone responses
- The need for robust communication architectures in real-time control systems
- The value of accessibility in control interface design

Several areas warrant further investigation:

1. Enhancement of simulation environments with obstacle course scenarios
2. Investigation of progressive output mapping between gesture and drone response
3. Optimization of control parameters through PID controller rates tuning
4. Development of standardized protocols for gesture recognition accuracy testing
5. Integration with autonomous planning systems for hybrid control schemes

While this research focused on basic flight control through hand gestures, future work could explore more complex interactions. This might include gesture-based programming of autonomous flight paths or collaborative control of multiple drones. These developments would further expand the possibilities of natural interfaces in drone operation.

The framework presented here serves as a starting point for more accessible drone control systems. As the applications continue to expand across industries, the need for intuitive and inclusive control interfaces will grow. This research demonstrates that gesture-based control offers a viable path toward meeting some of these needs while maintaining the precision required for effective operations.

REFERENCES

- ABIOYE, Ayodeji. **Multimodal speech and visual gesture control interface technique for small unmanned multirotor aircraft**. 2023. PhD thesis – University of Southampton.
- ACHERMANN, Yuri. **Hand Controlled Drone framework for PX4 ROS2 package**. [S.l.: s.n.], Sept. 2024. Software available at GitHub. Disponível em: <https://github.com/yuriachermann/Hand-Controlled-PX4-Drone>.
- BEARD, Randal W; MCLAIN, Timothy W. **Small unmanned aircraft: Theory and practice**. [S.l.]: Princeton university press, 2012.
- BEGUM, Tahajjat; HAQUE, Israat; KESELJ, Vlado. Deep learning models for gesture-controlled drone operation. In: IEEE. 2020 16th International Conference on Network and Service Management (CNSM). [S.l.: s.n.], 2020. P. 1–7.
- COVACIU, Florin; IORDAN, Anca-Elena. Control of a drone in virtual reality using MEMS sensor technology and machine learning. **Micromachines**, MDPI, v. 13, n. 4, p. 521, 2022.
- EPROSIMA. **Micro XRCE-DDS Documentation**. [S.l.: s.n.]. <https://docs.eprosima.com/en/latest/>. Accessed: 2024-12-06.
- FEDERAL AVIATION ADMINISTRATION. **FAA Aerospace Forecast Fiscal Years 2023–2043: Unmanned Aircraft System and Advanced Air Mobility**. [S.l.], 2023. Accessed: 2024-11-07. Disponível em: https://www.faa.gov/sites/faa.gov/files/2023-Unmanned%20Aircraft%20Systems%20and%20Advance%20Air%20Mobility_0.pdf.
- FERNANDEZ, Ramon A Suarez et al. Natural user interfaces for human-drone multi-modal interaction. In: IEEE. 2016 International Conference on Unmanned Aircraft Systems (ICUAS). [S.l.: s.n.], 2016. P. 1013–1022.
- FEUERRIEGEL, Stefan et al. Interface design for human-machine collaborations in drone management. **IEEE Access**, IEEE, v. 9, p. 107462–107475, 2021.
- FRSKY Official Website. Accessed: 2024-11-05. Disponível em: <https://www.frsky-rc.com>.
- FURRER, Fadri et al. Rotors—a modular gazebo mav simulator framework. **Robot Operating System (ROS) The Complete Reference (Volume 1)**, Springer, p. 595–625, 2016.

GHASEMI, Hamed et al. Control a drone using hand movement in ROS based on single shot detector approach. In: IEEE. 2020 28th Iranian Conference on Electrical Engineering (ICEE). [S.l.: s.n.], 2020. P. 1–5.

GIO, Nicolas; BRISCO, Ross; VULETIC, Tijana. Control of a drone with body gestures. **Proceedings of the Design Society**, Cambridge University Press, v. 1, p. 761–770, 2021.

HIRSHORN, Steven R; VOSS, Linda D; BROMLEY, Linda K. **NASA systems engineering handbook**. [S.l.], 2017.

HOFFMANN, Gabriel et al. Quadrotor helicopter flight dynamics and control: Theory and experiment. In: AIAA guidance, navigation and control conference and exhibit. [S.l.: s.n.], 2007. P. 6461.

HOLYBRO. **Holybro Official Website**. Accessed: 2024-11-05. 2024. Disponível em: <https://holybro.com>.

HU, Bin; WANG, Jiacun. Deep learning based hand gesture recognition and UAV flight controls. **International Journal of Automation and Computing**, Springer, v. 17, n. 1, p. 17–29, 2020.

IEEE. **IEEE Standard for Technical Reviews and Audits on Defense Programs**. [S.l.: s.n.], June 2014. P. 1–219. IEEE Std 15288.2-2014. DOI: 10.1109/IEEESTD.2014.6835334.

INCOSE. **INCOSE systems engineering handbook**. [S.l.]: John Wiley & Sons, 2023.

ISKANDAR, Mohd et al. Artificial intelligence-based human gesture tracking control techniques of Tello EDU Quadrotor Drone. In: IET. INTERNATIONAL Conference on Green Energy, Computing and Intelligent Technology (Gen-CITY 2023). [S.l.: s.n.], 2023. P. 123–128.

ISO. **Systems and software engineering – System life cycle processes**. Geneva, Switzerland: [s.n.], 2015. ISO/IEC/IEEE 15288:2015.

KOENIG, Nathan; HOWARD, Andrew. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IEEE. 2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566). [S.l.: s.n.], 2004. P. 2149–2154.

LABS, Thalmic. **Myo - Real Life Applications of the Myo Armband**. First Available: 2015; Accessed: 2024-11-19. 2015. Disponível em: <https://www.youtube.com/watch?v=te1RBQQ1Hz4>.

- LATIF, Bilawal; BUCKLEY, Neil; SECCO, Emanuele Lindo. Hand gesture and human-drone interaction. In: SPRINGER. PROCEEDINGS of SAI Intelligent Systems Conference. [S.l.: s.n.], 2022. P. 299–308.
- LEE, Taeyoung; LEOK, Melvin; MCCLAMROCH, N Harris. Geometric tracking control of a quadrotor UAV on SE (3). In: IEEE. 49TH IEEE conference on decision and control (CDC). [S.l.: s.n.], 2010. P. 5420–5425.
- LEE, Ji-Won; YU, Kee-Ho. Wearable Drone Controller: Machine Learning-Based Hand Gesture Recognition and Vibrotactile Feedback. **Sensors**, MDPI, v. 23, n. 5, p. 2666, 2023.
- LIM, Jaeyoung. **px4-offboard**. [S.l.: s.n.], Apr. 2023. Software available at GitHub. Disponível em: <https://github.com/Jaeyoung-Lim/px4-offboard>.
- MAURER, Maik. **Structural awareness in complex product design**. 2007. PhD thesis – Technische Universität München.
- MAVLINK. **MAVLink Micro Air Vehicle Communication Protocol**. [S.l.: s.n.]. <https://mavlink.io/en/>. Accessed: 2024-12-06.
- MEIER, Lorenz; HONEGGER, Dominik; POLLEFEYS, Marc. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In: IEEE. 2015 IEEE international conference on robotics and automation (ICRA). [S.l.: s.n.], 2015. P. 6235–6240.
- MUTALIB, Muhammad. Flying Drone Controller by Hand Gesture Using Leap Motion. **International Journal of Advanced Trends in Computer Science and Engineering**, v. 9, p. 111–116, Sept. 2020. DOI: 10.30534/ijatcse/2020/1791.42020.
- NATARAJAN, Kathiravan; NGUYEN, Truong-Huy D; METE, Mutlu. Hand gesture controlled drones: An open source library. In: IEEE. 2018 1st International Conference on Data Intelligence and Security (ICDIS). [S.l.: s.n.], 2018. P. 168–175.
- OBAID, Mohammad et al. How would you gesture navigate a drone? a user-centered approach to control a drone. In: PROCEEDINGS of the 20th International Academic Mindtrek Conference. [S.l.: s.n.], 2016. P. 113–121.
- PESHKOVA, Ekaterina; HITZ, Martin; AHLSTRÖM, David. Exploring user-defined gestures and voice commands to control an unmanned aerial vehicle. In: SPRINGER. INTELLIGENT Technologies for Interactive Entertainment: 8th International Conference, INTETAIN 2016, Utrecht, The Netherlands, June 28–30, 2016, Revised Selected Papers. [S.l.: s.n.], 2017. P. 47–62.
- PX4 DEVELOPMENT TEAM. **PX4 User Guide**. Accessed: 2024-11-05. 2024. Disponível em: <https://docs.px4.io>.

- QUIGLEY, Morgan et al. ROS: an open-source Robot Operating System. In: KOBE, JAPAN, 3.2. ICRA workshop on open source software. [S.l.: s.n.], 2009. P. 5.
- RAJA, Muneeb Masood. Extended Kalman Filter and LQR controller design for quadrotor UAVs, 2017.
- SARKAR, Ayanava et al. Gesture control of drone using a motion controller. In: IEEE. 2016 international conference on industrial informatics and computer systems (ciics). [S.l.: s.n.], 2016. P. 1–5.
- SEIDU, Idris; LAWAL, Jafaar Olasunkanmi. Personalized Drone Interaction: Adaptive Hand Gesture Control with Facial Authentication. **Int J Sci Res Sci Eng Technol**, v. 11, p. 43–60, 2024.
- SHAH, Shital et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: SPRINGER. FIELD and Service Robotics: Results of the 11th International Conference. [S.l.: s.n.], 2018. P. 621–635.
- SHIM, David; KIM, Hyoun; SASTRY, Shankar. Hierarchical control system synthesis for rotorcraft-based unmanned aerial vehicles. In: AIAA guidance, navigation, and control conference and exhibit. [S.l.: s.n.], 2000. P. 4057.
- SICILIANO, Bruno; KHATIB, Oussama. Robotics and the Handbook. In: SPRINGER Handbook of Robotics. [S.l.]: Springer, 2016. P. 1–6.
- STUART, Pugh. Total Design, Addison-Wesley. **Reading, MA**, 1990.
- TEAM, ArduPilot Development. **ArduPilot Documentation**. [S.l.], 2024. Accessed: 2024-11-04. Disponível em: <https://ardupilot.org/ardupilot/index.html>.
- ULTRALEAP. **Tracking Technical Specifications**. [S.l.: s.n.], 2024. Accessed: 2024-11-06. Disponível em: <https://docs.ultraleap.com/tracking/specifications/>.
- ZHAO, Zhenfei et al. Web-based interactive drone control using hand gesture. **Review of Scientific Instruments**, AIP Publishing, v. 89, n. 1, 2018.
- ZHONG, Jiageng et al. A safer vision-based autonomous planning system for quadrotor uavs with dynamic obstacle trajectory prediction and its application with llms. In: PROCEEDINGS of the IEEE/CVF Winter Conference on Applications of Computer Vision. [S.l.: s.n.], 2024. P. 920–929.


```

53         elif (orientation_x < self.angular_threshold_1):
54             twist.linear.y = 0.0 # Neutral
55         elif (orientation_x < self.angular_threshold_2):
56             twist.linear.y = -1.0 # Backward x1
57         else:
58             twist.linear.y = -2.0 # Backward x2
59         # Roll Left/Right movements
60         if (orientation_z > self.angular_threshold_2):
61             twist.linear.x = +2.0 # Left x2
62         elif (orientation_z > self.angular_threshold_1):
63             twist.linear.x = +1.0 # Left x1
64         elif (orientation_z > -self.angular_threshold_1):
65             twist.linear.x = 0.0 # Neutral
66         elif (orientation_z > -self.angular_threshold_2):
67             twist.linear.x = -1.0 # Right x1
68         else:
69             twist.linear.x = -2.0 # Right x2
70         # Yaw CW/CCW movements
71         if (orientation_y < -self.angular_threshold_2):
72             twist.angular.z = +1.0 # Rotate CW x2
73         elif (orientation_y < -self.angular_threshold_1):
74             twist.angular.z = +0.5 # Rotate CW x1
75         elif (orientation_y < self.angular_threshold_1):
76             twist.angular.z = 0.0 # Neutral
77         elif (orientation_y < self.angular_threshold_2):
78             twist.angular.z = -0.5 # Rotate CCW x1
79         else:
80             twist.angular.z = -1.0 # Rotate CCW x2
81
82     elif str(hand.type) == "HandType.Left":
83         print(f"Left hand detected. Ignoring.")
84
85     else:
86         twist.linear.z = 0.0 # Neutral
87         twist.linear.y = 0.0 # Neutral
88         twist.linear.x = 0.0 # Neutral
89         twist.angular.z = 0.0 # Neutral
90
91     if self.twist_changed(twist):
92         self.pub.publish(twist)
93         self.last_twist = twist
94         print(f"X:{twist.linear.x: .1f} \
95             Y:{twist.linear.y: .1f} \
96             Z:{twist.linear.z: .1f} \
97             Yaw:{twist.angular.z: .1f}")
98
99     def twist_changed(self, new_twist):
100         return (
101             new_twist.linear.x != self.last_twist.linear.x or
102             new_twist.linear.y != self.last_twist.linear.y or
103             new_twist.linear.z != self.last_twist.linear.z or
104             new_twist.angular.z != self.last_twist.angular.z
105         )

```

APPENDIX B – SETUP BASH COMMANDS

Listing B.1 – Bash commands defining framework requirements installation steps

```

# 1. Ubuntu 22.04 System Update
sudo apt update && sudo apt full-upgrade -y
sudo reboot

# After rebooting, run this script again starting from step 2.

# 2. Install ROS2 Humble
# Set locale
locale # check for UTF-8
sudo apt update && sudo apt install locales -y
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale # verify settings
# Setup Sources
# Ensure Ubuntu Universe repository is enabled
sudo apt install software-properties-common -y
sudo add-apt-repository universe -y
# Add the ROS 2 GPG key
sudo apt update && sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros
.key -o /usr/share/keyrings/ros-archive-keyring.gpg
# Add the ROS 2 repository to your sources list
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
keyrings/ros-archive-keyring.gpg] \
    http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo \
$UBUNTU_CODENAME) main" | \
    sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
# Update your apt repository caches
sudo apt update
sudo apt upgrade -y # Ensure your system is up to date
# Install ROS 2 Desktop
sudo apt install ros-humble-desktop -y
# Source ROS 2 environment
source /opt/ros/humble/setup.bash

# 3. Install Python 3.10
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install python3.10 python3.10-venv python3.10-dev -y

# 4. Create and activate a Python virtual environment
python3.10 -m venv ~/venv
source ~/venv/bin/activate

# 5. Install Gemini LeapC API
cd ~
git clone https://github.com/ultraleap/leapc-python-bindings.git
cd leapc-python-bindings
pip install -r requirements.txt
python -m build leapc-cffi

```

```
pip install leapc-cffi/dist/leapc_cffi-0.0.1.tar.gz
pip install -e leapc-python-api

# 6. Install PX4 Autopilot
cd ~
git clone https://github.com/PX4/PX4-Autopilot.git --recursive -b release/1.14
bash ./PX4-Autopilot/Tools/setup/ubuntu.sh

# Restart your computer before continuing.
sudo reboot

# After rebooting, reactivate the Python virtual environment:
source ~/venv/bin/activate

# 7. Install PX4 Python Dependencies
pip3 install --user -U empy pyros-genmsg setuptools
pip3 install --user kconfiglib
pip install --user jsonschema
pip install --user jinja2

# 8. Build Micro XRCE-DDS
cd ~
git clone https://github.com/eProsima/Micro-XRCE-DDS-Agent.git
cd Micro-XRCE-DDS-Agent
mkdir build && cd build
cmake ..
make
sudo make install
sudo ldconfig /usr/local/lib/

# 9. Setup Workspace
mkdir -p ~/hand_controlled_drone_ws/src
cd ~/hand_controlled_drone_ws/src

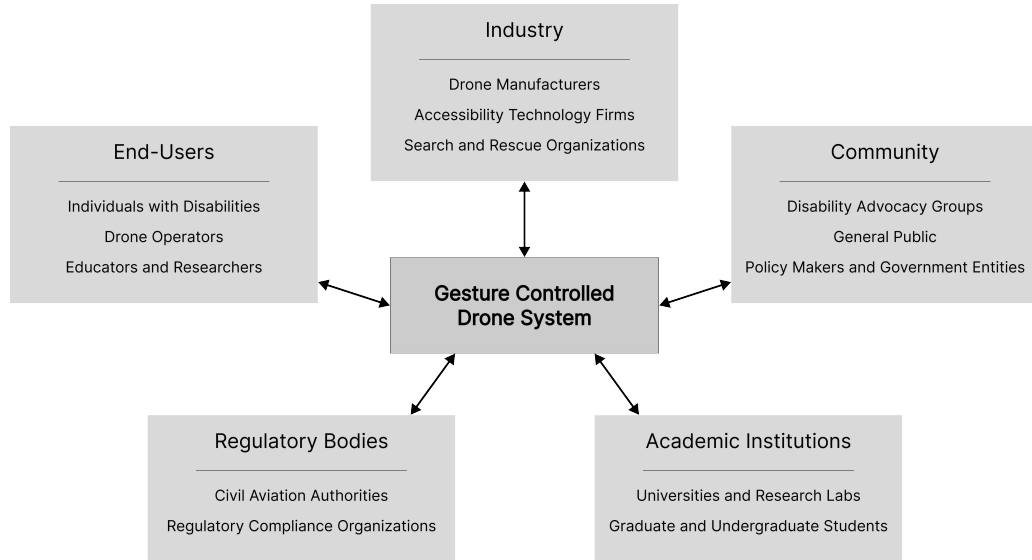
# 10. Clone in Packages
git clone https://github.com/PX4/px4_msgs.git -b release/1.14
git clone https://github.com/yuriachermann/Hand-Controlled-PX4-Drone.git

# 11. Building the Workspace
source /opt/ros/humble/setup.bash
cd ~/hand_controlled_drone_ws
colcon build
source install/setup.bash

# 12. Launch the simulation
ros2 launch px4_offboard hand_gesture_control.launch.py
```

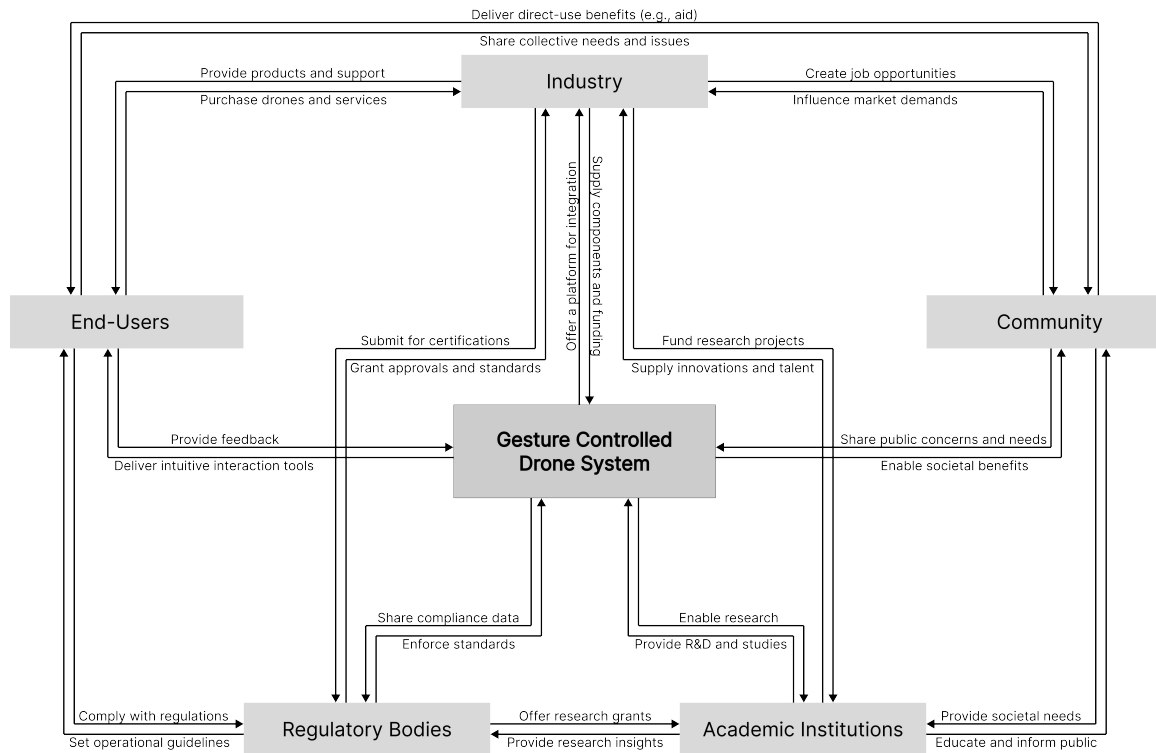
ANNEX A – STAKEHOLDER MAPS

Figure 24 – Gesture Controlled Drone System Stakeholders Hub-and-Spoke Map



Source: created by the author.

Figure 25 – Gesture Controlled Drone System Stakeholders SVN Map



Source: created by the author.