CHAPTER – 1

INTRODUCTION

Chapter – 1

INTRODUCTION

1.1 Introduction

Unmanned Aerial Vehicles (UAVs) are increasingly integral to applications such as surveillance, precision agriculture, and disaster management, requiring flexible and robust solutions for real-time monitoring and mission control. A Ground Control Station (GCS) forms the core of UAV operations, serving as the point from which operators issue commands, receive telemetry, and analyze mission data, either onsite or remotely via networked systems.

With the rise of the Internet of Things (IoT) and cloud-enabled services, IoT-based GCS architectures have emerged, offering extended accessibility, scalable management, and rich real-time data integration through web and mobile dashboards. Central to this innovation is the use of industry-standard MAVLink protocol, which enables reliable and interoperable communication between UAV autopilots and ground systems.

This work specifically utilizes the Seed Studio Xiao ESP32S3 microcontroller, interfacing directly with a Mini Pix flight controller, to establish a wireless bridge for telemetry and command using MAVLink. Integration with the Blynk IoT Cloud platform enables secure transmission and visualization of UAV status, live telemetry, and remote control functions via mobile and web-based interfaces. This approach demonstrates a cost-effective, scalable, and practical model for next-generation UAV ground control solutions applicable to a wide range of real-world scenarios.

Unmanned Aerial Vehicles (UAVs), commonly known as drones, have emerged as powerful tools in modern engineering applications due to their flexibility, mobility, and ability to access hard-to-reach areas. When combined with the Internet of Things (IoT), UAVs can collect, transmit, and analyze real-time data efficiently, enabling intelligent monitoring and control systems. An IoT-based Ground and Unmanned Aerial Vehicle (GUAV) system integrates both aerial and ground segments to enhance surveillance, data acquisition, and decision-making capabilities.

In traditional monitoring systems, data collection is often manual, time-consuming, and limited in coverage. These limitations reduce efficiency and may lead to delayed responses in critical applications such as environmental monitoring, agriculture, disaster management, and infrastructure inspection. The proposed IoT-based GUAV system overcomes these challenges by utilizing sensors,

wireless communication, and cloud platforms to provide real-time data visualization and remote operation.

This project focuses on the design and development of an IoT-enabled GUAV system that can autonomously or semi-autonomously monitor a given area, collect sensor data, and transmit it to a Ground Control Station (GCS) for analysis. The system aims to improve accuracy, reduce human effort, and enable faster decision-making through real-time monitoring

1.2 Problem Statement

Conventional monitoring and surveillance methods suffer from several limitations such as restricted coverage area, high manpower requirements, delayed data availability, and lack of real-time analysis. In many scenarios like agricultural field monitoring, environmental observation, disaster-affected regions, and remote infrastructure inspection, manual data collection becomes inefficient and sometimes unsafe.

Existing systems often rely on either ground-based monitoring or standalone UAV operations without proper integration of IoT technologies. This leads to issues such as limited data accessibility, poor communication reliability, lack of centralized monitoring, and inability to store and analyze data efficiently.

Therefore, there is a need for a smart, integrated system that combines UAV technology with IoT to enable real-time data acquisition, remote monitoring, efficient communication, and intelligent analysis. The problem addressed in this project is the design of an IoT-based GUAV system that can reliably collect sensor data, transmit it wirelessly, and provide meaningful insights through a centralized monitoring platform.

1.3 Objectives

The main objectives of the IoT-based GUAV system project are as follows:

1. To design and develop an integrated GUAV system using UAV and IoT technologies.

2. To collect real-time environmental and system parameters using onboard sensors.

3. To transmit sensor data wirelessly to a Ground Control Station or cloud platform.

4. To enable real-time monitoring and visualization of collected data.

5. To reduce human intervention and improve operational efficiency.

6. To enhance accuracy and reliability in data collection and analysis.

7. To demonstrate the practical application of IoT-enabled UAV systems in real-world scenarios

8. To ensure scalability and flexibility for future enhancements.

1.4 Methodology

The proposed UAV control system is designed using a layered operational approach, combining basic real-time control with advanced mission planning and secure communication. The methodology is divided into Basic Operation and Secondary Operation to ensure reliability, scalability, and safety.

1. Basic Operation

The ESP32-S3 microcontroller acts as the central processing unit of the UAV. It continuously reads essential flight parameters required for stable operation and maneuvering, such as:

- **Pitch** – forward and backward tilt of the UAV
- **Roll** – left and right tilt
- **Yaw** – rotational direction around the vertical axis
- **Throttle** – speed and altitude control.

These parameters are acquired from onboard sensors and control inputs and are processed in real time by the ESP32-S3. The processed data is transmitted to a real-time dashboard, which visually represents the UAV's current flight status using gauges, graphs, and indicators.

The dashboard allows users to:

- Monitor flight parameters in real time
- Manually control UAV movement and altitude
- Perform basic flight operations such as takeoff, landing, hovering, and directional movement

This operation mode ensures low-latency control, making it suitable for manual piloting and immediate response scenarios.

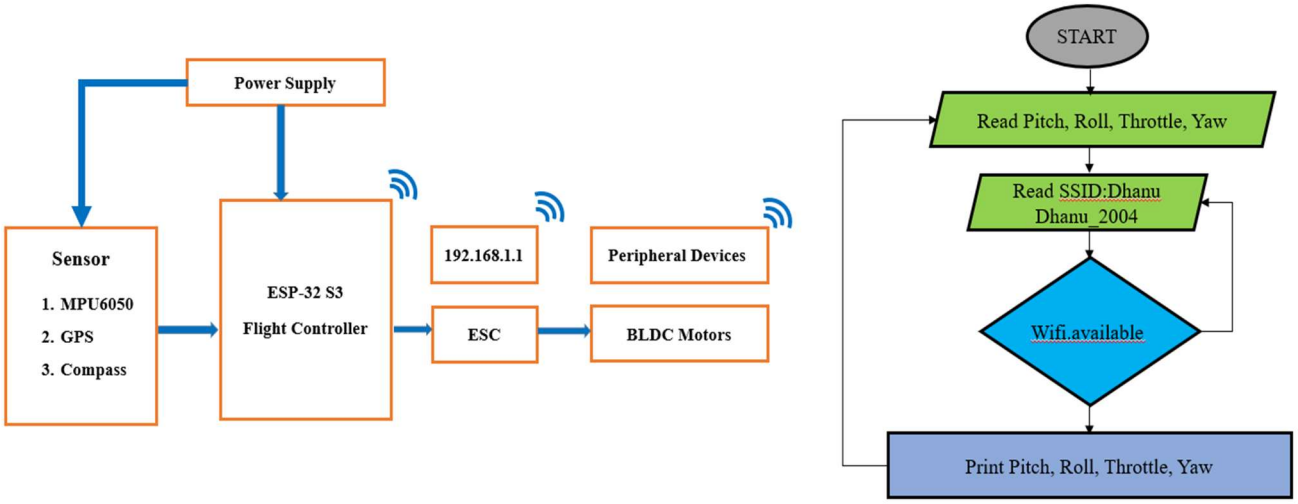


Fig 1 : Design of IOT GUAV Architecture for Basic Operations and Workflow

2. Secondary Operation

For advanced UAV functionality, the system integrates the MAVLink communication protocol, which is widely used in UAV and autopilot systems. MAVLink enables reliable and structured communication between the UAV and the Ground Control Station (GCS).

Through MAVLink, the system supports:

- Mission planning and execution
- Transmission of waypoints and predefined flight paths
- Setting altitude targets and speed constraints
- Telemetry data exchange such as battery status, GPS position, and system health

This secondary operation allows the UAV to perform autonomous or semi-autonomous missions, reducing the need for continuous manual control.

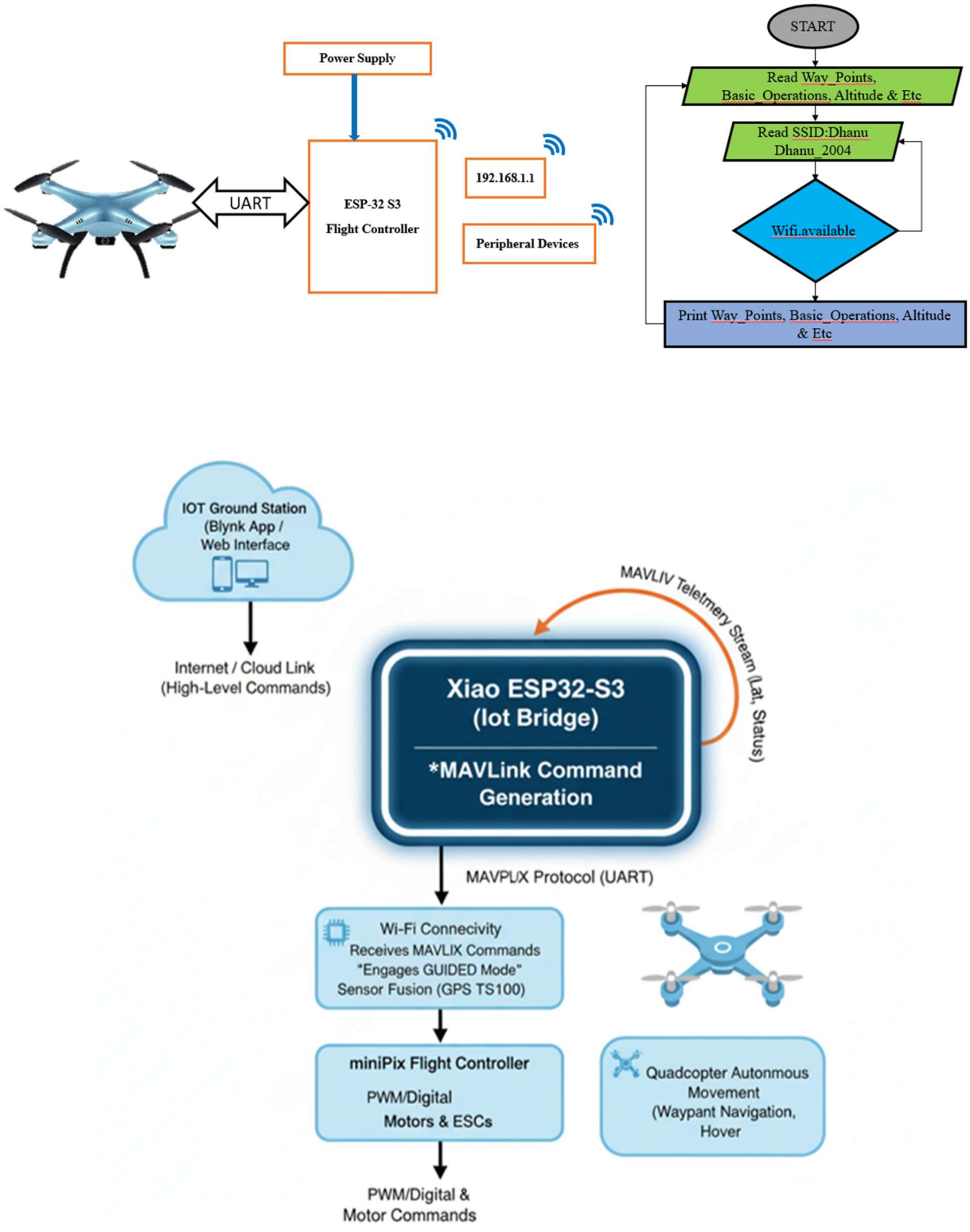


Fig 2: Design of IOT GUAV Architecture for Secondary Operation

1.5 Literature Survey

[1]. CloudStation": A Cloud-based Ground Control Station for Drones

Lyuyang Hu, Omkar Pathak (2020)

This paper demonstrates an open source cloud based ground control station that allows remote piloting of drones (aerial, land, or sea based vehicle) from anywhere in the world with just a web browser. A standard client-server architecture is used for pilot-server and drone-server communication and it is designed to be scalable for the control of millions of drones simultaneously. CloudStation, a proof of concept web app, is built with Django and hosted on a cloud based service provider with global availability and scalability.

[2]. Micro Air Vehicle Link (MAVLink) in a Nutshell

Yasir Javed (2019)

The Micro Air Vehicle Link (MAVLink in short) is a communication protocol for unmanned systems (e.g., drones, robots). It specifies a comprehensive set of messages exchanged between unmanned systems and ground stations. This protocol is used in major autopilot systems, mainly ArduPilot and PX4, and provides powerful features not only for monitoring and control ling unmanned systems missions, but also for their integration into the Internet.

[3]. Integration of Artificial Intelligence and IoT with UAVs for Precision Agriculture

Ashfakul Karim Kausik, Anika Khandoker (2025)

This review paper explores the dynamic landscape of precision agriculture and the integration of Unmanned Aerial Vehicles (UAVs) with Artificial Intelligence (AI) and the Internet of Things (IoT). Precision agriculture has emerged as a transformative approach to farming, enabling data-driven decisions, resource optimization, and improved yields. UAVs have become essential tools in agriculture, providing real-time data on crop health, pest infestations, and irrigation needs. Integrating AI and IoT further enhances precision agriculture by enabling data analytics, predictive modeling, and remote monitoring. This paper delves into the concepts and techniques of precision agriculture, the role of UAVs in farming, the applications of AI in crop management, and the IoT de vices and connectivity protocols used in agriculture

[4]. MAVLink (Micro Air Vehicle Link) communication protocol

Lorenz Meier et al. (2009)

The MAVLink (Micro Air Vehicle Link) communication protocol, which provides a lightweight and efficient messaging system for communication between unmanned aerial vehicles and ground control stations. The paper explains how MAVLink supports telemetry data exchange, mission commands, and waypoint navigation using minimal bandwidth. Due to its reliability and low latency, MAVLink has become a standard protocol in many open-source UAV platforms. This work forms the basis for using MAVLink in the proposed system to transmit advanced mission data such as waypoints and altitude targets.

[5]. Microcontroller-Based UAV flight control system

J. Kim and S. Park (2016)

They presented the design and implementation of a low-cost microcontroller-based UAV flight control system. The study demonstrates how embedded controllers process essential flight parameters including pitch, roll, yaw, and throttle in real time to maintain stable flight. The results show that microcontroller-based systems can effectively manage real-time control tasks while remaining power-efficient and economical. This research supports the use of the ESP32-S3 for basic UAV operation and real-time control in the proposed system.

[6]. IoT-based real-time monitoring and control system for UAVs.

A. Sharma and R. Verma (2018)

It proposes an IoT-based real-time monitoring and control system for UAVs. The paper describes how UAV telemetry data can be transmitted to cloud platforms and visualized using web or mobile dashboards. This approach enables remote monitoring, improved accessibility, and scalable UAV deployment. The findings validate the integration of real-time dashboards and IoT platforms such as Node.js or Blynk IoT in the proposed system.

CHAPTER – 2

WORKING MODEL

CHAPTER – 2

WORKING MODEL

2.1 Working Model

The IoT-Based GUAV System integrates a Ground Unmanned Vehicle (GUV) with an Unmanned Aerial Vehicle (UAV) to perform intelligent monitoring, surveillance, and data collection tasks using IoT technologies.

Working Principle: Data Collection

Sensors mounted on the UAV and GUV collect real-time data such as:

- Temperature
- Humidity
- Gas/air quality
- Camera feed (image/video)

UAV Operation

- UAV flies autonomously or semi-autonomously using GPS waypoints.
- Captures aerial images/videos and sensor data.
- Transmits data to the Ground Control Station (GCS) via wireless communication.

GUV Operation

- GUV moves on land to reach inaccessible or risky areas.
- Equipped with obstacle detection sensors.
- Sends environmental and positional data to the GCS.

IoT Connectivity

- All collected data is uploaded to a cloud server using Wi-Fi/4G/5G.
- Users can monitor live data using a web or mobile dashboard.

Decision Making

- The system analyzes sensor data.
- Alerts are generated if abnormal conditions are detected (fire, gas leakage, intrusion).

Remote Monitoring

- Operators control UAV/GUV using a GCS or mobile application.
- Live telemetry and video streaming are displayed.

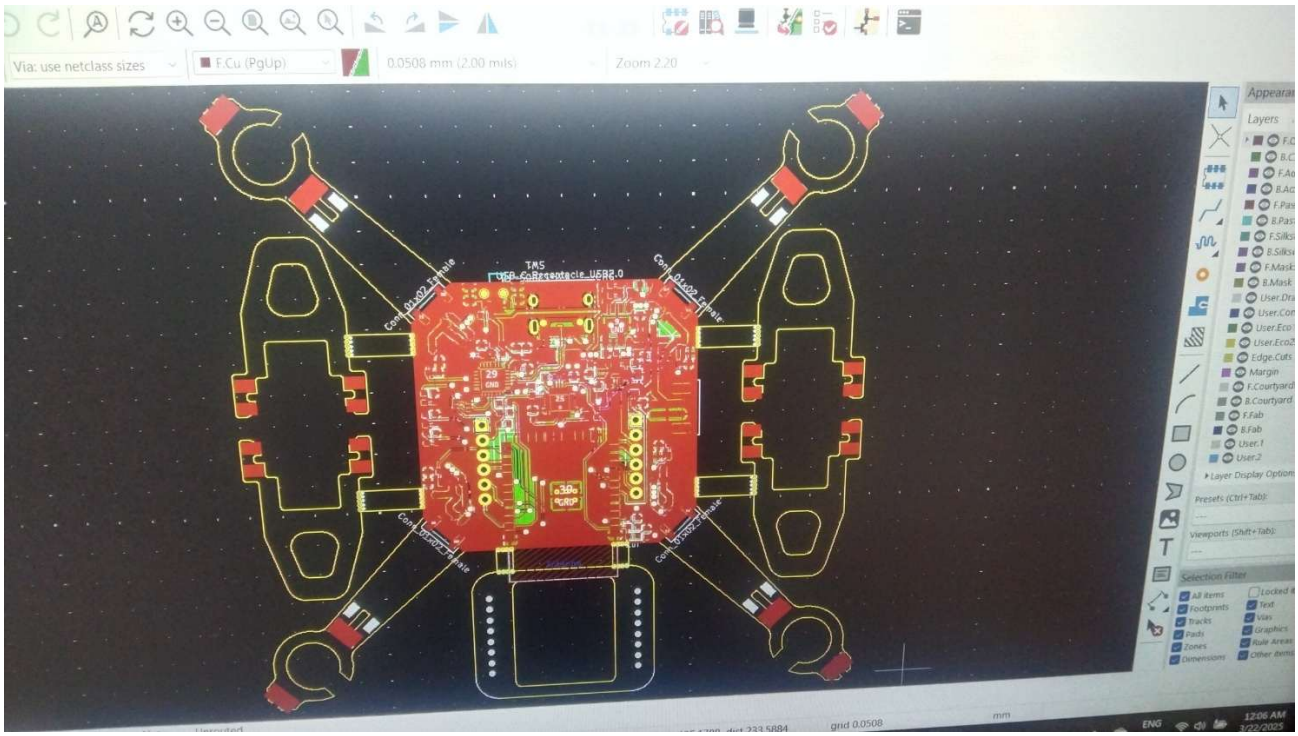
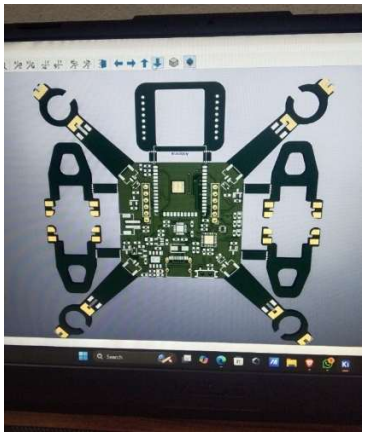
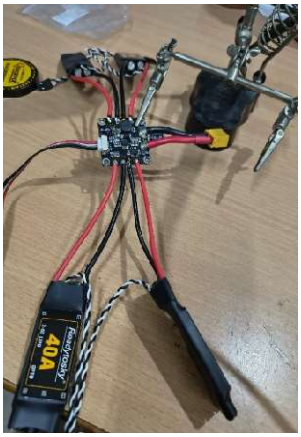


Fig 3: Schematic Diagram of IOT GUAV

"The waypoint navigation system utilizes an IoT-centric architecture to bridge the communication gap between the remote node device and the UAV's flight controller. The process initiates at the user-operated node device, where target latitude and longitude coordinates are input via the Blynk IoT interface. These geospatial data points are instantaneously transmitted to the Blynk cloud server, which serves as the centralized command relay. Onboard the UAV, the ESP32-S3 microcontroller functions as a telemetry bridge, actively polling or receiving these coordinate updates from the cloud over a Wi-Fi network. Upon reception, the ESP32-S3 parses the raw coordinate data and encapsulates it into standard MAVLink mission protocol packets (specifically utilizing MISSION_ITEM or MAV_CMD_NAV_WAYPOINT messages). These serialized MAVLink commands are subsequently transmitted via UART to the MiniPix flight controller, which validates the data and updates its autonomous flight plan in real-time, thereby enabling Over-the-Air (OTA) mission planning beyond standard radio frequency ranges."

The images together represent the design, assembly, and working of an IoT-based G-UAV system. First, the quadcopter structure is designed using PCB/CAD software in an X-configuration to ensure proper balance, equal thrust distribution, and stable flight. This design integrates mounting points for motors, flight controller, power distribution, and communication modules. After fabrication, the PCB is assembled with four BLDC motors mounted on each arm and propellers attached, while the flight controller is placed at the center to minimize vibration and improve sensor accuracy. Power from the Li-Po battery is supplied through Electronic Speed Controllers (ESCs), which regulate motor speed based on control signals from the flight controller. The flight controller processes data from onboard sensors such as the gyroscope and accelerometer to maintain stability and executes flight commands received from the Ground Control Station (GCS) via MAVLink communication. Through this coordinated operation, the power system, control electronics, communication protocol, and mechanical structure work together to enable controlled, stable, and autonomous UAV flight suitable for IoT-based monitoring and applications.





The components shown in the figure collectively form the working model of the IoT-based G-UAV system by integrating control, communication, power, and propulsion units. The Pixhawk Mini flight controller acts as the central processing unit, where sensor data such as orientation and motion are processed to maintain stable flight. It receives control commands either from the radio receiver or from the Ground Control Station (GCS) through MAVLink communication. The radio receiver enables manual control and mode switching, while telemetry modules provide real-time data such as position, altitude, and battery status to the GCS. Power from the Li-Po battery is regulated and distributed through Electronic Speed Controllers (ESCs), which convert low-power control signals from the flight controller into high-power signals required to drive the BLDC motors. These motors generate thrust through propellers, enabling lift, maneuvering, and directional control of the UAV.

All wiring, connectors, and mounting hardware ensure reliable electrical connections and mechanical stability, allowing the UAV to operate as a coordinated IoT-enabled aerial platform for monitoring, data collection, and autonomous missions.

Fig 4: Working Model of IOT GUAV

2.2 HARDWARE & SOFTWARE COMPONENTS

(A) Xiao ESP32 S3

A tiny and powerful microcontroller board used to add Wi-Fi, Bluetooth, and smart processing to electronic prototypes.

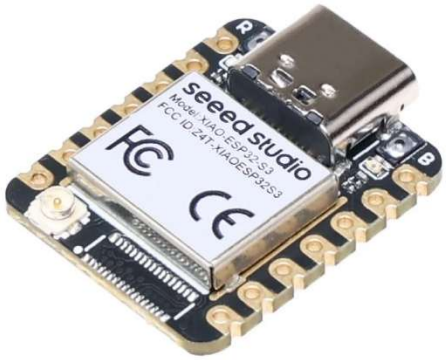


Fig 5Xiao ESP32 S3

- Very small size, fits easily in drones and compact devices.
- Has Wi-Fi + BLE for IoT communication.
- Connects to the flight controller using MAVLink.
- Sends drone data (GPS, battery, altitude) to cloud or ground station.
- Acts as the IoT companion computer inside the UAV.

(B) MiniPix Flight Controller:

A compact and reliable flight controller used to control and stabilize drones.

Fig 6: MiniPix Flight Controller

- Keeps the drone stable during flight.
- Controls pitch, roll, yaw, and throttle.
- Manages motors, ESCs, and sensors.
- Acts as the main brain of the drone.
- Works with GPS TS100 for accurate positioning.
- Communicates with Xiao ESP32-S3 through MAVLink.
- Enables both manual (RC) and automated guided operations.

(C) TS 100:

A high-precision GPS module used to give accurate location and altitude data to the drone.

Fig 7: TS 100

- Provides latitude, longitude, altitude information.
- Helps the drone follow waypoints and autonomous missions.
- Improves flight stability using position hold (Loiter).
- Sends GPS data to the MiniPix flight controller.
- Enables guided flights, navigation, and return-to-launch (RTL).

(D) Quad-Copter Frame

- The frame is lightweight and strong to handle tough disaster conditions.
- It has four arms that keep the drone balanced and stable in the air.

Fig 8: Quad-Copter Frame

- The design allows easy mounting of cameras and sensors for rescue work.
- It protects the drone's parts from shocks and vibrations during flights.
- The open structure helps keep the motors cool with good airflow.
- It can carry extra equipment needed for disaster management tasks.
- It is usually made from carbon fiber, plastic, or aluminum for strength and light weight.
- It provides a balanced and stable base for smooth flight.

(E) Brushless DC Motor

Fig 9: Brushless Motor

- Brushless dc motors give strong and stable power for smooth drone flying.
- They have no brushes, so they last longer and work more reliably.
- These motors are very efficient, helping the drone stay in the air for longer time.

- They generate less heat, making them safe for long disaster-response missions.
- Brushless motors offer fast response, allowing precise and quick drone movements.
- They create less noise, which helps in quiet search-and-rescue operations.
- They work well with ESCs to deliver consistent speed and performance in the air.

(F) Electronic Speed Controller (ESC)

Fig 10: Electronic Speed Controller

- The ESC controls the speed of the drone's brushless motors for smooth flight.
- It quickly adjusts motor speed to help the drone stay stable during missions.
- ESCs convert battery power into the right signals for the motors.
- Converts battery power into 3-phase power needed by brushless motors.
- They help the drone respond instantly to pilot commands or autopilot changes.
- ESCs protect the motors by preventing overloads and overheating
- They ensure balanced motor rotation, which is important in disaster areas.
- ESCs support safe takeoff, landing, and hovering during rescue operations.
- Helps the drone stay stable by adjusting motor speeds quickly.
- They work with the flight controller to provide accurate and controlled movement.

(G) Propellers

- Propellers create the lift needed to make the drone fly and stay in the air.

Fig 11: Propellers

- They are lightweight, usually made of plastic or carbon fiber for better performance.
- Balanced propellers reduce vibrations and help the drone fly smoothly.
- They come in different sizes and shapes, which affect speed, stability, & lifting power.
- Strong propellers can handle windy and rough disaster-area conditions.
- They work with brushless motors to provide stable and controlled movement.
- Efficient propellers help increase flight time during long rescue missions.
- Each pair spins in opposite directions (CW & CCW) to keep the drone balanced.

(H) Transmitter & Receiver

Fig 12: Transmitter & Receiver

- The transmitter sends control signals from the pilot to the drone during missions.
- The receiver on the drone receives these signals to guide movement and actions.

- They provide long-range communication, useful for large disaster-area coverage.
- Both use radio frequencies to ensure fast and reliable control.
- The transmitter controls functions like speed, direction, or movement.
- The receiver ensures each command reaches the device without delay.
- They have multiple channels to manage different controls at the same time.
- They allow the user to control the drone in real time.

2.3 Software Implementation

(A) MavLink

MAVLink is a communication language that allows your drone parts to talk to each other.

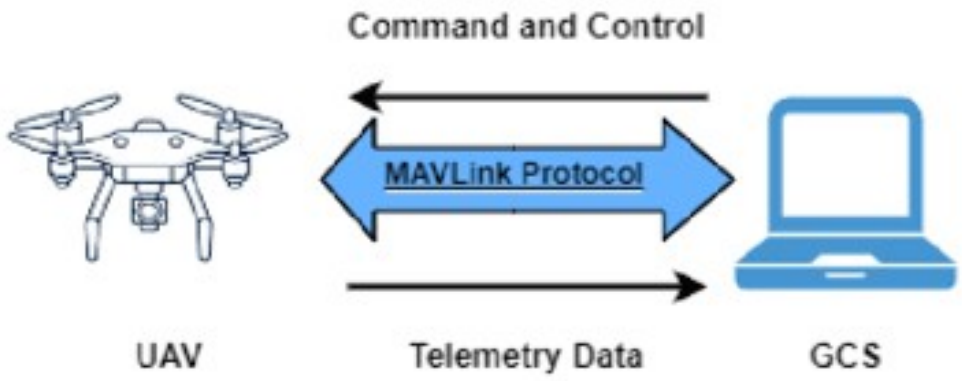


Fig 13: MavLink

- The MAVLink communication shown in the figure enables effective command and control between the UAV and the Ground Control Station (GCS).
- Using the MAVLink protocol, the GCS sends control commands such as takeoff, landing, mode changes, speed, and navigation instructions to the UAV.
- At the same time, the UAV continuously transmits telemetry data back to the GCS, including altitude, position, attitude (pitch, roll, yaw), battery status, and system health.
- This bidirectional communication allows the operator to monitor the drone's real-time performance and make immediate decisions during flight.
- MAVLink acts as a common communication language, ensuring reliable, low-latency data exchange between onboard flight controllers and ground systems, which is essential for stable operation, autonomous missions, and IoT-based monitoring applications

(B) Mission Planner

Mission Planner is a ground control software used to plan, monitor, and control unmanned aerial vehicles (UAVs) such as drones. It provides an intuitive interface where users can create flight paths, set waypoints, and define mission parameters like altitude, speed, and route. The software communicates with the drone's flight controller to upload the planned mission and monitor the drone's status in real-time. Mission Planner also supports live telemetry, allowing users to track GPS coordinates, battery levels, and sensor data during flight, which is essential for safe and efficient drone operations.

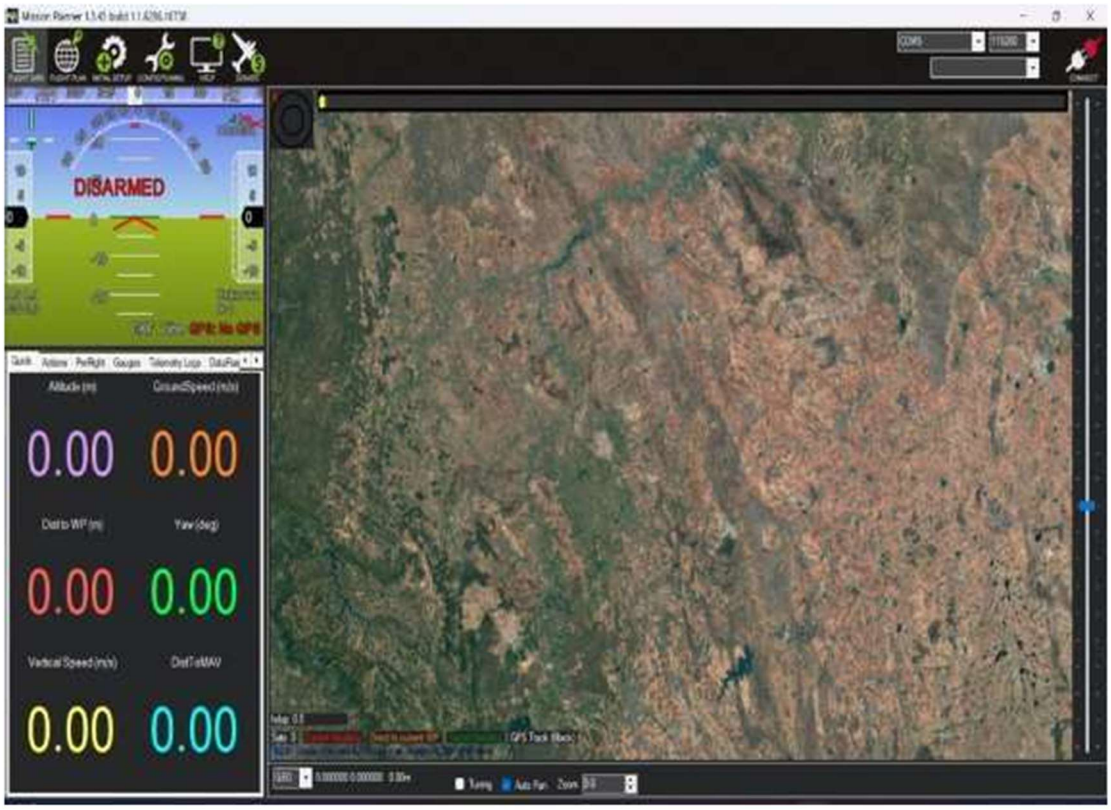


Fig 14: Mission Planner Software

Mission Planner is a ground control software used to plan, monitor, and control unmanned aerial vehicles (UAVs) such as drones. It provides an intuitive interface where users can create flight paths, set waypoints, and define mission parameters like altitude, speed, and route. The software communicates with the drone's flight controller to upload the planned mission and monitor the drone's status in real-time. Mission Planner also supports live telemetry, allowing users to track GPS coordinates, battery levels, and sensor data during flight, which is essential for safe and efficient drone operations.

CHAPTER – 3

RESULTS

CHAPTER – 3

RESULTS

3.1 Results

The proposed IoT-Based GUAV System was successfully designed, implemented, and tested under real-time conditions. The system demonstrated effective coordination between the UAV and GUV with reliable IoT communication.

Key Results Achieved:

- Successful UAV Flight Operation UAV achieved stable take-off, hovering, navigation, and landing.
- GPS-based waypoint navigation worked accurately.
- Live telemetry data (altitude, speed, battery level) was received at the GCS.

Efficient GUV Ground Movement:

- GUV moved smoothly in forward, reverse, left, and right directions.
- Obstacle detection using ultrasonic sensors was accurate.
- Automatic stopping and path correction were observed when obstacles were detected.

Real-Time Sensor Data Monitoring:

- Temperature, humidity, and gas sensor data were collected continuously.
- Sensor values were uploaded to the cloud platform in real time.
- Data visualization through graphs and dashboards was accurate and stable.

Reliable Wireless Communication:

- Stable communication between UAV, GUV, and Ground Control Station.
- Minimal data loss during telemetry transmission.
- Wi-Fi/telemetry modules showed acceptable range and latency.

Live Surveillance Capability:

- Camera module provided clear live video feed.
- Useful for real-time monitoring and post-mission analysis.

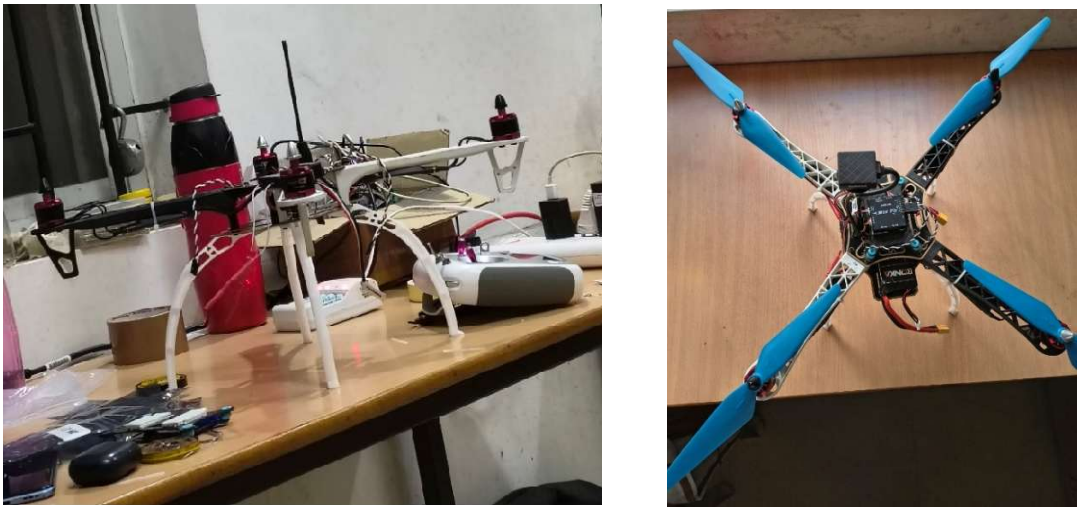
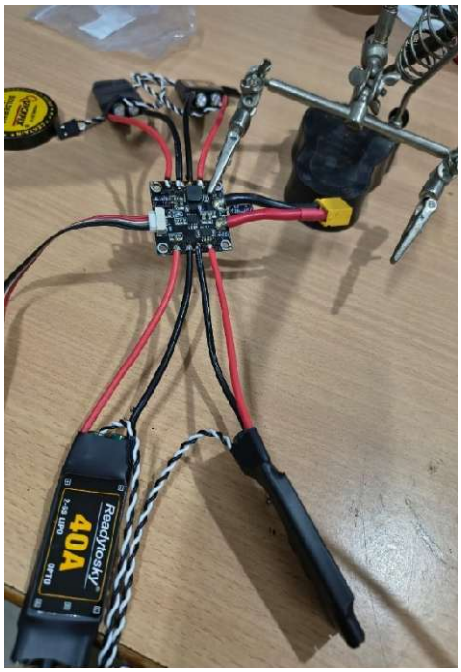


Fig 15: Final Results



The figure illustrates the final working model and results of the IoT-based G-UAV system after successful integration and testing of all components. In this stage, the UAV is fully assembled with the frame, BLDC motors, propellers, flight controller, power system, and communication modules securely mounted. When powered on, the Li-Po battery supplies energy to the flight controller and ESCs, which drive the motors to generate lift. The flight controller continuously processes sensor data to stabilize the UAV and maintain controlled flight. Commands from the Ground Control Station are received through MAVLink communication, allowing the UAV to perform actions such as takeoff, hovering, navigation, and landing. Simultaneously, real-time telemetry data including

position, altitude, and battery status is transmitted back to the GCS for monitoring. The successful flight and stable operation shown in the results demonstrate that the mechanical structure, electronic components, and communication system worked together effectively, validating the proposed IoT G-UAV system for practical aerial monitoring and autonomous applications.

Fig 16: Demonstration of UAV

Drones have emerged as a highly effective tool in modern disaster management, providing fast and accurate situational awareness during emergencies. Their ability to fly over inaccessible or hazardous areas allows authorities to quickly assess damage, identify risks, and understand the overall impact of disasters such as floods, earthquakes, landslides, and fires. With real- time imaging and sensor data, decision-makers can prioritize critical zones and deploy rescue teams more efficiently. In rescue operations, drones significantly enhance the chances of saving lives by locating stranded or injured individuals using thermal cameras, night-vision systems, and high- resolution live video. They can navigate areas where traditional vehicles cannot reach, making them invaluable in searching collapsed structures, dense forests, or flooded regions. Additional capabilities such as delivering essential supplies—medical kits, food, water, and communication devices—ensure timely support to victims when ground access is delayed. Beyond immediate relief, drones continue to contribute during the recovery phase by generating detailed maps, monitoring reconstruction activities, and assessing environmental changes. These insights help governments and disaster-response agencies

plan long-term rehabilitation, allocate resources wisely, and restore affected communities efficiently. Overall, drone technology has transformed disaster management by improving response speed, operational safety, and the effectiveness of rescue and recovery efforts.

4.2 Advantages of IoT-Based G-UAV System

- **Real-Time Data Monitoring**
  - The system provides real-time transmission of sensor data and live video, enabling faster and more accurate decision-making.
- **Remote Accessibility**
  - G-UAV operations can be controlled and monitored remotely through IoT platforms, reducing the need for human presence in hazardous areas.
- **Improved Safety**
  - Minimizes human risk by performing surveillance, inspection, and monitoring tasks in dangerous or inaccessible locations.
- **High Mobility and Coverage**
  - UAVs can easily access remote, elevated, or difficult terrains and cover large areas efficiently compared to ground-based systems alone.
- **Cost-Effective Solution**
  - Reduces operational costs by minimizing manpower, time, and the need for expensive traditional monitoring systems.
- **Integration of Multiple Sensors**
  - Supports various IoT sensors such as GPS, temperature, gas, humidity, and cameras, enabling multi-purpose applications.
- **Accurate Data Collection**
  - Provides precise and reliable data for analysis due to automated data acquisition and reduced human error.
- **Scalability and Flexibility**
  - The system can be easily upgraded by adding new sensors, communication modules, or software features.
- **Faster Response Time**
  - Enables quick detection of issues such as disasters, security breaches, or environmental changes, improving response efficiency.

- **Enhanced Monitoring and Control**
    - Combines ground station intelligence with aerial surveillance, offering better coordination and system control.

4.3 Applications

- **Disaster Management & Rescue Operations**
    - Used to locate victims, assess damage, and send real-time data from disaster-affected areas where human access is risky.
- **Agricultural Monitoring (Smart Farming)**
    - Helps in crop health monitoring, soil analysis, irrigation planning, and detecting pest-affected areas using aerial data.
- **Surveillance & Security Monitoring**
    - Applied in border surveillance, campus security, traffic monitoring, and crowd management with real-time video streaming.
- **Environmental Monitoring**
    - Used to monitor air pollution, water quality, forest conditions, and climate data using onboard IoT sensors.
- **Smart City Applications**
    - Supports traffic analysis, infrastructure inspection, waste management monitoring, and public safety surveillance.
- **Military & Defense Operations**
    - Used for reconnaissance, enemy area monitoring, and mission planning without risking human lives.
- **Search and Mapping (GIS Applications)**
    - Helps in land surveying, mapping inaccessible areas, and creating 3D terrain models.
- **Industrial Inspection.**
    - Used for inspection of pipelines, power lines, towers, and large industrial plants to detect faults or damages.
- **Healthcare & Emergency Support**
    - Can deliver medical supplies, monitor remote health camps, and provide live video feed during emergencies.
- **Forest & Wildlife Monitoring**

- Helps track animal movement, prevent poaching, and monitor forest fires in real time.

4.4 Challenges of IoT-Based G-UAV System

- **Limited Battery Life**
  - UAVs have limited flight time due to battery constraints, which affects long-duration monitoring and missions.
- **Network Connectivity Issues**
  - Reliable internet or wireless communication is difficult in remote or disaster-affected areas, impacting real-time data transfer.
- **Data Security and Privacy**
  - IoT-based communication is vulnerable to cyber threats, unauthorized access, and data interception.
- **Weather Dependency**
  - Strong winds, rain, fog, and extreme temperatures can affect UAV stability and sensor performance.
- **High Initial Cost**
  - Initial setup cost for UAV hardware, sensors, controllers, and IoT platforms can be relatively high.
- **Payload Limitations**
  - UAVs have restricted payload capacity, limiting the number and type of sensors that can be mounted.
- **Complex System Integration**
  - Integrating hardware components, sensors, communication modules, and software platforms requires technical expertise.
- **Latency and Data Loss**
  - Network delays and packet loss can reduce the effectiveness of real-time monitoring and control.
- **Regulatory and Legal Constraints**
  - UAV operations are subject to aviation regulations, airspace restrictions, and permissions from authorities.
- **Maintenance and Calibration Issues**

- Regular maintenance, calibration of sensors, and system updates are required for accurate performance.

4.5 Conclusion

This project successfully demonstrated the design and implementation of an IoT-Based GUAV System that integrates aerial and ground robotic platforms for intelligent monitoring and surveillance applications.

The system effectively combines sensor data acquisition, wireless communication, cloud computing, and remote control, enabling real-time monitoring and decision support. The coordinated operation of UAV and GUV enhances area coverage, improves data accuracy, and reduces human risk in hazardous environments.

The results prove that the proposed system is reliable, scalable, and cost-effective, making it suitable for applications such as disaster management, smart agriculture, border surveillance, and industrial inspection. With further advancements in automation, AI, and communication technologies, the system can be expanded into a fully autonomous and intelligent monitoring solution.

4.6 Future Scope

The IoT-Based GUAV System has wide potential for future enhancements and real-world deployment.

Possible Improvements:

- AI & Machine Learning Integration Image processing for object detection and tracking.
- Predictive analytics for environmental hazards.

Fully Autonomous Operation

- Advanced path planning algorithms.
- Automatic decision-making without human intervention.

Extended Communication Range

- Use of 5G, LoRaWAN, or satellite communication.
- Improved real-time control over long distances.

Swarm Technology

- Multiple UAVs and GUVs working cooperatively.

- Faster area coverage and redundancy.

Enhanced Power Management

- Solar-powered charging stations.
- Smart battery health monitoring.

Improved Security

- End-to-end encryption for IoT data.
- Secure authentication mechanisms.

Advanced Sensors

- Thermal cameras for night surveillance.
- LiDAR for precise mapping and obstacle avoidance.

REFERNECES

[1] J. Gao, Y. Zhang, Z. Wu, and L. N. Yu, “Optimized collaborative scheduling of unmanned aerial vehicles for emergency material distribution in flood disaster management,” Mechatron. Intell Transp. Syst., vol. 4, no. 1, pp. 1–15, 2025. https://doi.org/10.56578/mits040101.

[2] Grando,L.;Jaramillo, J.F.G.; Leite, J.R.E.; Ursini, E.L. Systematic Literature Review Methodology for Drone Recharging Processesin Agriculture and Disaster Management. Drones 2025, 9, 40. https://doi.org/ 10.3390/drones9010040.

[3] M. Fujisawa et al., “Flight Models Considering Wind Effects on Drone Based Networks During Large-Scale Disasters,” IEICE Communications Express, 2025.

[4] S. Alqefari and M. E. B. Menai, “Multi-UAV Task Assignment in Dynamic Environments: Current Trends and Future Directions,” Drones, 2025.

[5] G. Ariante and G. Del Core, “UAS: Current State, Emerging Technologies, and Future Trends,” Drones, 2025.

[6] D. E. Martin et al., “Spray Deposition and Drift ... RPAAS,” Drones, 2025.

[7] Z. Tang et al., “UAV Detection with Passive Radar,” Drones, 2025.

[8] Y. Kawahara, A. Kamal, and M. Fujisawa, “Lightweight information- sharing system with access control in disaster management with se- curity against dishonest users,” Journal of Signal Processing, vol. 28, no. 4, pp. 161–164, 2024. DOI: 10.2299/jsp.28.161.

[9] Abdolazimi, O., and J. Ma. 2024. “An Integrative Production-Delivery System with Mobile Additive Manufacturing and Truck-Drone Delivery Considering Optimal Printing Sequence and Preferred Delivery Time Window.” International Journal of Production Research 1–26.

[10] Bogyrbayeva, A., T. Yoon, H. Ko, S. Lim, H. Yun, and C. Kwon. 2023. “A Deep Reinforcement Learning Approach for Solving the Traveling Salesman Problem with Drone.” Transportation Research Part C: Emerging Technologies 148: 103981.

[11] Cha, H., D. Kim, J. Eun, and T. Cheong. 2023. “Collaborative Traveling Salesman Problem with Ground Vehicle as a Charger for Unmanned Aerial Vehicle.” Transportation Letters 15 (7): 707–721. https://doi.org/10.1080/19427867.2022.2082006.

[12] Mohsan, S. A. H., N. Q. H. Othman, Y. Li, M. H. Alsharif, and M. A. Khan. 2023. “Unmanned Aerial Vehicles (UAVs): Practical Aspects, Applications, Open Challenges, Security Issues, and Future Trends.” Intelligent Service Robotics 16 (1): 109–137.

[13] Pachayappan, M., and B. Sundarakani. 2023. “Drone Delivery Logistics Model for on-Demand Hyperlocal Market.” International Journal of Logistics Research and Applications 26 (12): 1728–1760. https://doi.org/10.1080/13675567.2022.2107189.

[14] Ghelichi, Z., M. Gentili, and P. B. Mirchandani. 2022. “Drone Logistics for Uncertain Demand of Disaster-Impacted Populations.” Transportation Research Part C: Emerging Technologies 141: 103735. https://doi.org/10.1016/j.trc.2022.103735.

APPENDIX

Appendix A: List of Hardware Components

- UAV Frame (Quadcopter)
- Brushless DC (BLDC) Motors
- Electronic Speed Controllers (ESCs)
- Flight Controller (Pixhawk / equivalent)
- GPS Module
- IoT Sensors (Temperature, Humidity, Gas, etc.)
- Camera Module
- Battery (Li-Po)
- Power Distribution Board
- Communication Module (Wi-Fi / LTE / Telemetry)
- Ground Control Station (Laptop / PC)

Appendix B: List of Software Tools

- Mission Planner / QGroundControl
- Arduino IDE
- IoT Cloud Platform (ThingSpeak / Firebase / Blynk)
- Embedded C / Python
- MAVLink Protocol
- Operating System: Windows / Linux

Appendix C: Block Diagram Description

The IoT-based G-UAV system consists of a UAV integrated with multiple sensors and a camera. Sensor data is collected through the onboard controller and transmitted via IoT communication modules to a cloud server. The Ground Control Station monitors real-time data, flight parameters, and video feed for analysis and control.

Appendix D: Algorithm (Basic Workflow)

- Initialize UAV and IoT sensors
- Establish communication with Ground Control Station
- Collect sensor and GPS data

- Transmit data to IoT cloud platform
- Display real-time data on dashboard
- Monitor UAV flight and environmental conditions
- Store data for future analysis

Appendix E: Applications Snapshot

Disaster management, Agricultural monitoring, Surveillance and security, Environmental monitoring, Smart city applications

Appendix F: Future Enhancements

- AI-based object detection
- Extended battery life using solar charging
- 5G-enabled communication
- Autonomous mission planning

CODE

1. **Code of Haversine**

```
#include <TinyGPSPlus.h>
#include <HardwareSerial.h>

// Define GPS Serial (Use UART2 on ESP32)
HardwareSerial GPS(2);
TinyGPSPlus gps;

// Define GPS Pins
#define RXD2 16  // GPS TX to ESP32 RX
#define TXD2 17  // GPS RX to ESP32 TX

// Previous GPS position
double prevLat = 0.0;
double prevLon = 0.0;
bool firstFix = true;
double totalDistance = 0.0; // in meters

// Haversine formula to calculate distance between two lat/lon points
double haversine(double lat1, double lon1, double lat2, double lon2) {
  const double R = 6371000; // Earth radius in meters
  double dLat = radians(lat2 - lat1);
  double dLon = radians(lon2 - lon1);
  lat1 = radians(lat1);
  lat2 = radians(lat2);

  double a = sin(dLat/2) * sin(dLat/2) +
             cos(lat1) * cos(lat2) *
             sin(dLon/2) * sin(dLon/2);
  double c = 2 * atan2(sqrt(a), sqrt(1 - a));
  return R * c;
```

```
}

void setup() {
  Serial.begin(115200);
  GPS.begin(9600, SERIAL_8N1, RXD2, TXD2);
  Serial.println("GPS Distance Tracker Initialized");
}

void loop() {
  while (GPS.available()) {
    gps.encode(GPS.read());

    if (gps.location.isUpdated()) {
      double currentLat = gps.location.lat();
      double currentLon = gps.location.lng();

      Serial.print("Lat: ");
      Serial.print(currentLat, 6);
      Serial.print(" | Lon: ");
      Serial.print(currentLon, 6);

      if (firstFix) {
        prevLat = currentLat;
        prevLon = currentLon;
        firstFix = false;
        Serial.println(" | First fix.");
      } else {
        double distance = haversine(prevLat, prevLon, currentLat, currentLon);

        // Optional filter: ignore very small movements (<1 meter) to reduce GPS jitter
        if (distance > 1.0) {
          totalDistance += distance;
```

```
        prevLat = currentLat;
        prevLon = currentLon;
      }

      Serial.print(" | Step Distance: ");
      Serial.print(distance, 2);
      Serial.print(" m | Total Distance: ");
      Serial.print(totalDistance, 2);
      Serial.println(" m");
    }
  }
 }
}
```

2. UAV Calibration_Secondary Operations

```
#define BLYNK_TEMPLATE_ID "TMPL33nZf9y-7"
#define BLYNK_TEMPLATE_NAME "IoT_UAV_Calibration"
#define BLYNK_AUTH_TOKEN "ohKAlPN-ps5APpwG62tmVXBg8RuQhGHz"
#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <ESP32Servo.h>

char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Dhanu";
char pass[] = "Itzzmee_Dhanu";
```

```
Servo esc;
Servo esc1;
Servo esc2;
Servo esc3;
void setup() {
  esc.attach(D1,   1000, 2500);
  esc.write(1500);

  esc1.attach(D2,  1000, 2500);
  esc1.write(1500);

  esc2.attach(D5,  1000, 2500);
  esc2.write(910);

  esc3.attach(D6,  1000, 2500);
  esc3.write(1500);

  delay(2000);
  Serial.begin(115200);
  Blynk.begin(auth, ssid, pass);

}


BLYNK_WRITE(V0)
{
  esc.write(param.asInt());
}

BLYNK_WRITE(V1)
{
  esc1.write(param.asInt());
```

```
}

BLYNK_WRITE(V2)
{
  esc2.write(param.asInt());
}
BLYNK_WRITE(V3)
{
  esc3.write(param.asInt());
}


void loop() {

  Blynk.run();


}
```

3.Gyro_caliberation

```
#include<Wire.h>
#include <ESP32WiFi.h>
#include <WiFiUdp.h>
WiFiUDP UDP;
char packet[7];
boolean recvState;
//----------------------------------------------------------------------//
int ESCout_1 ,ESCout_2 ,ESCout_3 ,ESCout_4;
int input_PITCH = 50;
```

```
int input_ROLL = 50;
int input_YAW;
int input_THROTTLE=1100;
int state1,state2,state3,state4;
//-------------------------------------------------------------------//
int16_t gyro_x, gyro_y, gyro_z, acc_x, acc_y, acc_z, temperature, acc_total_vector;
float angle_pitch, angle_roll, angle_yaw;
boolean set_gyro_angles;
float angle_roll_acc, angle_pitch_acc;
float angle_pitch_output, angle_roll_output;
float elapsedTime;
long Time, timePrev, time2;
long gyro_x_cal, gyro_y_cal, gyro_z_cal;
//-------------------------------------------------------------------//
float pitch_PID,roll_PID,yaw_PID;
float roll_error, roll_previous_error, pitch_error, pitch_previous_error, yaw_error;
float roll_pid_p, roll_pid_d, roll_pid_i, pitch_pid_p, pitch_pid_i, pitch_pid_d, yaw_pid_p,
yaw_pid_i;
float roll_desired_angle, pitch_desired_angle, yaw_desired_angle;
double twoX_kp=5;
double twoX_ki=0.003;
double twoX_kd=2;
double yaw_kp=3;
double yaw_ki=0.002;
//-------------------------------------------------------------------//
```

```
void setup() {
pinMode(D2,OUTPUT);pinMode(D5,OUTPUT);pinMode(D6,OUTPUT);pinMode(D8,OUTPUT);
delay(1300);
Serial.begin(115200);
WiFi.mode(WIFI_STA);
WiFi.begin("DRR", "D R-N K-K-L");
while (WiFi.status() != WL_CONNECTED){
  GPOS = (1 << 14);GPOS = (1 << 12);GPOS = (1 << 13);GPOS = (1 << 15);
  delayMicroseconds(1000);
  GPOC = (1 << 14);GPOC = (1 << 12);GPOC = (1 << 13);GPOC = (1 << 15);
  delayMicroseconds(1000);
  yield();
}
Serial.println(WiFi.localIP());
UDP.begin(9999);
//-----------------------------------------------------------------------//
  Wire.begin();
   Wire.setClock(400000);
  Wire.beginTransmission(0x68);
  Wire.write(0x6B);
  Wire.write(0x00);
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x1C);
  Wire.write(0x10);
```

```
  Wire.endTransmission();
  Wire.beginTransmission(0x68);
  Wire.write(0x1B);
  Wire.write(0x08);
  Wire.endTransmission();
  delay(1000);
for (int cal_int = 0; cal_int < 2000 ; cal_int ++){
  if(cal_int % 125 == 0)Serial.print(".");
  Wire.beginTransmission(0x68);
  Wire.write(0x3B);
  Wire.endTransmission();
  Wire.requestFrom(0x68,14);
  while(Wire.available() < 14);
  acc_x = Wire.read()<<8|Wire.read();
  acc_y = Wire.read()<<8|Wire.read();
  acc_z = Wire.read()<<8|Wire.read();
  temperature = Wire.read()<<8|Wire.read();
  gyro_x = Wire.read()<<8|Wire.read();
  gyro_y = Wire.read()<<8|Wire.read();
  gyro_z = Wire.read()<<8|Wire.read();
  gyro_x_cal += gyro_x;
  gyro_y_cal += gyro_y;
  gyro_z_cal += gyro_z;
  GPOS = (1 << 14);GPOS = (1 << 12);GPOS = (1 << 13);GPOS = (1 << 15);
  delayMicroseconds(1000);
```

```
  GPOC = (1 << 14);GPOC = (1 << 12);GPOC = (1 << 13);GPOC = (1 << 15);
  delayMicroseconds(1000);
  yield();
}
gyro_x_cal /= 2000;
gyro_y_cal /= 2000;
gyro_z_cal /= 2000;
//---------------------------------------------------------------------//
Time = micros();
}

void loop(){
GPOS = (1 << 14);GPOS = (1 << 12);GPOS = (1 << 13);GPOS = (1 << 15);
  timePrev = Time;
  Time = micros();
  elapsedTime = (float)(Time - timePrev) / (float)1000000;
  Wire.beginTransmission(0x68);
  Wire.write(0x3B);
  Wire.endTransmission();
  Wire.requestFrom(0x68,14);
  while(Wire.available() < 14);
  acc_x = Wire.read()<<8|Wire.read();
  acc_y = Wire.read()<<8|Wire.read();
  acc_z = Wire.read()<<8|Wire.read();
  temperature = Wire.read()<<8|Wire.read();
```

```
  gyro_x = Wire.read()<<8|Wire.read();
  gyro_y = Wire.read()<<8|Wire.read();
  gyro_z = Wire.read()<<8|Wire.read();
gyro_x -= gyro_x_cal;
gyro_y -= gyro_y_cal;
gyro_z -= gyro_z_cal;
  angle_pitch += gyro_x * elapsedTime * 0.01526717557;
  angle_roll += gyro_y * elapsedTime * 0.01526717557;
  angle_yaw += gyro_z * elapsedTime * 0.01526717557;
  angle_pitch += angle_roll * sin(gyro_z * 0.000001066);
  angle_roll -= angle_pitch * sin(gyro_z * 0.000001066);
  acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));
  angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296;
  angle_roll_acc = asin((float)acc_x/acc_total_vector)* -57.296;
  angle_pitch_acc += 0;
  angle_roll_acc += 0;
  if(set_gyro_angles){
    angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;
    angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;
  }
  else{
    angle_pitch = angle_pitch_acc;
    angle_roll = angle_roll_acc;
    set_gyro_angles = true;
  }
```

```
  angle_pitch_output = angle_pitch_output * 0.9 + angle_pitch * 0.1;
  angle_roll_output = angle_roll_output * 0.9 + angle_roll * 0.1;
//-------------------------------------------------------------------//
roll_desired_angle = 3*((float)input_ROLL/(float)10 - (float)5);
pitch_desired_angle =3*((float)input_PITCH/(float)10 - (float)5);
//yaw_desired_angle =0;

roll_error =  angle_roll_output - roll_desired_angle;
pitch_error = angle_pitch_output - pitch_desired_angle;
yaw_error = angle_yaw - yaw_desired_angle;

roll_pid_p = twoX_kp*roll_error;
pitch_pid_p = twoX_kp*pitch_error;
yaw_pid_p = yaw_kp*yaw_error;

if(-3 < roll_error <3){roll_pid_i = roll_pid_i+(twoX_ki*roll_error);}
if(-3 < pitch_error <3){pitch_pid_i = pitch_pid_i+(twoX_ki*pitch_error);}
if(-3 < yaw_error <3){yaw_pid_i = yaw_pid_i+(yaw_ki*yaw_error);}

roll_pid_d = twoX_kd*((roll_error - roll_previous_error)/elapsedTime);
pitch_pid_d = twoX_kd*((pitch_error - pitch_previous_error)/elapsedTime);
roll_PID = roll_pid_p + roll_pid_i + roll_pid_d;
pitch_PID = pitch_pid_p + pitch_pid_i + pitch_pid_d;
yaw_PID = yaw_pid_p + yaw_pid_i;
```

```
if(roll_PID < -400){roll_PID=-400;}
else if(roll_PID > 400) {roll_PID=400;}
if(pitch_PID < -400){pitch_PID=-400;}
else if(pitch_PID > 400) {pitch_PID=400;}
if(yaw_PID < -400){yaw_PID=-400;}
else if(yaw_PID > 400) {yaw_PID=400;}

ESCout_1 = input_THROTTLE - roll_PID - pitch_PID - yaw_PID;
ESCout_2 = input_THROTTLE + roll_PID - pitch_PID + yaw_PID;
ESCout_3 = input_THROTTLE + roll_PID + pitch_PID - yaw_PID;
ESCout_4 = input_THROTTLE - roll_PID + pitch_PID + yaw_PID;

if(ESCout_1>2000) ESCout_1=2000;
else if(ESCout_1<1100) ESCout_1=1100;
if(ESCout_2>2000) ESCout_2=2000;
else if(ESCout_2<1100) ESCout_2=1100;
if(ESCout_3>2000) ESCout_3=2000;
else if(ESCout_3<1100) ESCout_3=1100;
if(ESCout_4>2000) ESCout_4=2000;
else if(ESCout_4<1100) ESCout_4=1100;

roll_previous_error = roll_error;
pitch_previous_error = pitch_error;
//-----------------------------------------------------------------------//
while((micros() - Time) < 1000);
```

```
state1 = 1; state2 = 1; state3 = 1; state4 = 1;
while(state1 == 1 || state2 == 1 || state3 == 1 || state4 == 1){
  time2 = micros();
  if((time2 - Time) >= ESCout_1 && state1 == 1){ GPOC = (1 << 14); state1=0;}
  if((time2 - Time) >= ESCout_2 && state2 == 1){ GPOC = (1 << 12);state2=0;}
  if((time2 - Time) >= ESCout_3 && state3 == 1){ GPOC = (1 << 13);state3=0;}
  if((time2 - Time) >= ESCout_4 && state4 == 1){ GPOC = (1 << 15);state4=0;}
}
//--------------------------------------------------------------------//
if(!recvState){
  int packetSize = UDP.parsePacket();
  if (packetSize) {
    int len = UDP.read(packet, 6);
    packet[len] = '\0';
if(String(packet[0]) == "a"){
input_ROLL = int(packet[1]);
input_PITCH = int(packet[2]);
input_THROTTLE = 1000 + int(packet[3]);
input_YAW = int(packet[4]);
}
else if(String(packet[0]) == "b"){
input_ROLL = int(packet[1]);
input_PITCH = int(packet[2]);
input_THROTTLE = 1000 + int(packet[3])*100 + int(packet[4]);
input_YAW = int(packet[5]);
```

```
}
if(String(packet[0]) == "1"){
twoX_kp = (float)int(packet[1])/(float)100;
twoX_ki = (float)int(packet[2])/(float)1000;
twoX_kd = (float)int(packet[3])/(float)100;
}
else if(String(packet[0]) == "2"){
twoX_kp = (float)(float)(int(packet[1])*100 + int(packet[2]))/(float)100;
twoX_ki = (float)(float)(int(packet[3])*100 + int(packet[4]))/(float)1000;
twoX_kd = (float)(float)(int(packet[5])*100 + int(packet[6]))/(float)100;
}
else if(String(packet[0]) == "3"){
twoX_kp = (float)(int(packet[1])*100 + int(packet[2]))/(float)100;
twoX_ki = (float)int(packet[3])/(float)1000;
twoX_kd = (float)int(packet[4])/(float)100;
}
else if(String(packet[0]) == "4"){
twoX_kp = (float)int(packet[1])/(float)100;
twoX_ki = (float)(int(packet[2])*100 + int(packet[3]))/(float)1000;
twoX_kd = (float)int(packet[4])/(float)100;
}
else if(String(packet[0]) == "5"){
twoX_kp = (float)int(packet[1])/(float)100;
twoX_ki = (float)int(packet[2])/(float)1000;
twoX_kd = (float)(int(packet[3])*100 + int(packet[4]))/(float)100;
```

```
}
else if(String(packet[0]) == "6"){
twoX_kp = (float)(int(packet[1])*100 + int(packet[2]))/(float)100;
twoX_ki = (float)(int(packet[3])*100 + int(packet[4]))/(float)1000;
twoX_kd = (float)int(packet[5])/(float)100;
}
else if(String(packet[0]) == "7"){
twoX_kp = (float)int(packet[1])/(float)100;
twoX_ki = (float)(int(packet[2])*100 + int(packet[3]))/(float)1000;
twoX_kd = (float)(int(packet[4])*100 + int(packet[5]))/(float)100;
}
else if(String(packet[0]) == "8"){
twoX_kp = (float)(int(packet[1])*100 + int(packet[2]))/(float)100;
twoX_ki = (float)int(packet[3])/(float)1000;
twoX_kd = (float)(int(packet[4])*100 + int(packet[5]))/(float)100;
}
Serial.print(input_ROLL);Serial.print(" ");
Serial.print(input_THROTTLE);Serial.print(" ");
Serial.print(twoX_kp);Serial.print(" ");
Serial.print(twoX_ki,3);Serial.print(" ");
Serial.print(twoX_kd);Serial.println();
}
}
else if(recvState){
UDP.beginPacket(UDP.remoteIP(), UDP.remotePort());
```

```
UDP.print(Time - timePrev);
UDP.endPacket();
}
recvState = !recvState;
//---------------------------------------------------------------------//
Serial.print(angle_roll_output);Serial.print("  ");
Serial.print(angle_pitch_output);Serial.print(" | ");
Serial.print(roll_desired_angle);Serial.print("  ");
Serial.print(pitch_desired_angle);
Serial.println();
}
```

4.Testing Phase

```
#include <ArduinoJson.h>
 DynamicJsonDocument doc(1024);
 long yaw=1500,roll=1500,pitch=1500,throttle=910;
#include <ArduinoWebsockets.h>
#include <ESP32WiFi.h>

#include<Servo.h>

const char* ssid = "Dhanu"; //Enter SSID
const char* password = "Itzzmeebow_Dhanu"; //Enter Password
const char* websockets_server_host = "atrover.hexitronics.in"; //Enter server adress
const uint16_t websockets_server_port = 80; // Enter server port
```

```
Servo servo1,servo2,servo3,servo4;
using namespace websockets;
WebsocketsClient client;
void setup() {
  Serial.begin(115200);
  // Connect to wifi
  WiFi.begin(ssid, password);
  servo1.attach(D1);//roll
  servo2.attach(D2);//pitch
  servo3.attach(D5);//yaw
  servo4.attach(D6);//throttle
  pinMode(D0,OUTPUT);

  servo1.write(1500);
  servo2.write(1500);
  servo3.write(1500);
  servo4.write(910);
  // Wait some time to connect to wifi
  for(int i = 0; i < 10 && WiFi.status() != WL_CONNECTED; i++) {
    Serial.print(".");
    delay(1000);
  }

  // Check if connected to wifi
  if(WiFi.status() != WL_CONNECTED) {
```

```
    Serial.println("No Wifi!");
    return;
  }

  Serial.println("Connected to Wifi, Connecting to server.");
  // try to connect to Websockets server
  bool connected = client.connect(websockets_server_host, websockets_server_port, "/");
  if(connected) {
    Serial.println("Connecetd!");
    client.send("Hello Server");
    digitalWrite(D0,1);
  } else {
    Serial.println("Not Connected!");
     digitalWrite(D0,0);

  }

  // run callback when messages are received
  client.onMessage([&](WebsocketsMessage message) {
    Serial.print("Got Message: ");
    Serial.println(message.data());

      String input = message.data();

    Serial.println(input);
```

```
  deserializeJson(doc, input);


  if(doc[String("yaw")])
  {
    yaw = doc[String("yaw")];
  }
  if(doc[String("roll")])
  {
    roll = doc[String("roll")];
  }
  if(doc[String("pitch")])
  {
    pitch = doc[String("pitch")];
  }
  if(doc[String("throttle")])
  {
    throttle = doc[String("throttle")];
  }



    });
}


void loop() {
    // let the websockets client check for incoming messages
```

```
    if(client.available()) {
        client.poll();
    }



    servo1.write(roll+6);
    servo2.write(pitch+5);
    servo3.write(yaw+5);
    servo4.write(throttle);
Serial.println(roll);
Serial.println(pitch);
    Serial.println(yaw);
    Serial.println(throttle);
Serial.println("-------");




switch (WiFi.status()){
    case WL_NO_SSID_AVAIL:
        Serial.println("Configured SSID cannot be reached");
        for(int i=throttle;i>=910;i-=10){
          servo4.write(throttle);
          delay(500);
        }
        break;
```

```
    case WL_CONNECTED:
      Serial.println("Connection successfully established");
      break;
    case WL_CONNECT_FAILED:
      Serial.println("Connection failed");
      for(int i=throttle;i>=910;i-=10){
        servo4.write(throttle);
        delay(500);
      }
      break;
}}
```

5. Working on Basic Operations

```
 #include <WiFi.h>
#include <ESPAsyncWebSrv.h>
#include <Wire.h>
#include <MPU6050.h>
#include <ESP32Servo.h>

const char* ssid = "Dhanu";
const char* password = "Itzzmeebow_Dhanu";

AsyncWebServer server(80);

// Variables for flight control
```

```
float throttle = 0.0;
float roll = 0.0;
float pitch = 0.0;
float yaw = 0.0;

// Servo objects for motor control
Servo esc;
Servo esc1;
Servo esc2;
Servo esc3;

// Motor signal ranges
const int minSignal = 910;
const int maxSignal = 2400;

// MPU6050 object
MPU6050 mpu;

// PID constants
const float kp = 2.0;
const float ki = 0.5;
const float kd = 1.0;

// PID variables
float rollError = 0.0;
```

```
float rollIntegral = 0.0;
float rollDerivative = 0.0;
float rollPreviousError = 0.0;

float pitchError = 0.0;
float pitchIntegral = 0.0;
float pitchDerivative = 0.0;
float pitchPreviousError = 0.0;

float yawError = 0.0;
float yawIntegral = 0.0;
float yawDerivative = 0.0;
float yawPreviousError = 0.0;

// Function to map a value from one range to another
int mapRange(float value, float inMin, float inMax, int outMin, int outMax) {
  return (value - inMin) * (outMax - outMin) / (inMax - inMin) + outMin;
}

void setupWiFi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
```

```
  Serial.println("Connected to WiFi");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void setup() {
  Serial.begin(115200);

  setupWiFi();

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    String html = "<html><body>";
    html += "<h1> Vade Mahantha's Quadcopter Configuration</h1>";
    // Add your configuration options as HTML inputs/buttons/etc.
    html += "<form method='post' action='/save'>";
    html += "Throttle: <input type='text' name='throttle'><br>";
    html += "Roll: <input type='text' name='roll'><br>";
    html += "Pitch: <input type='text' name='pitch'><br>";
    html += "Yaw: <input type='text' name='yaw'><br>";
    html += "<input type='submit' value='Save'>";
    html += "</form>";
    html += "</body></html>";
    request->send(200, "text/html", html);
  });
```

```
 server.on("/save", HTTP_POST, [](AsyncWebServerRequest *request){
   String strThrottle = request->arg("throttle");
   String strRoll = request->arg("roll");
   String strPitch = request->arg("pitch");
   String strYaw = request->arg("yaw");

   throttle = strThrottle.toFloat();
   roll = strRoll.toFloat();
   pitch = strPitch.toFloat();
   yaw = strYaw.toFloat();

   request->send(200);
 });

 server.begin();

 // Attach servo objects to motor control pins
 esc.attach(14,   0, 2500);
 esc1.attach(25,  0, 2500);
 esc2.attach(26,  0, 2500);
 esc3.attach(27,  0, 2500);

 // Initialize MPU6050 sensor
 Wire.begin();
 mpu.initialize();
```

```
  // Calibrate MPU6050 sensor
  mpu.CalibrateGyro();
  mpu.setDLPFMode(3); // Set gyroscope low pass filter to 42Hz

  // Enable the sensor
  mpu.setSleepEnabled(false);

  // Wait for stabilization
  delay(1000);
}

void loop() {
  // Read sensor data
  int16_t gyroX = mpu.getRotationX();
  int16_t gyroY = mpu.getRotationY();
  int16_t gyroZ = mpu.getRotationZ();

  // Quadcopter flight control logic
  // Use throttle, roll, pitch, yaw, and sensor data for stability and control

  // Calculate roll PID control
  rollError = roll - gyroX;
  rollIntegral += rollError;
  rollDerivative = rollError - rollPreviousError;
```

```
  float rollOutput = kp * rollError + ki * rollIntegral + kd * rollDerivative;
  rollPreviousError = rollError;

  // Calculate pitch PID control
  pitchError = pitch - gyroY;
  pitchIntegral += pitchError;
  pitchDerivative = pitchError - pitchPreviousError;
  float pitchOutput = kp * pitchError + ki * pitchIntegral + kd * pitchDerivative;
  pitchPreviousError = pitchError;

  // Calculate yaw PID control
  yawError = yaw - gyroZ;
  yawIntegral += yawError;
  yawDerivative = yawError - yawPreviousError;
  float yawOutput = kp * yawError + ki * yawIntegral + kd * yawDerivative;
  yawPreviousError = yawError;

  // Map the flight control values and PID outputs to motor signal ranges
  int motor1Signal = mapRange(throttle + rollOutput - pitchOutput - yawOutput, -1.0, 1.0,
minSignal, maxSignal);
  int motor2Signal = mapRange(throttle - rollOutput - pitchOutput + yawOutput, -1.0, 1.0,
minSignal, maxSignal);
  int motor3Signal = mapRange(throttle - rollOutput + pitchOutput - yawOutput, -1.0, 1.0,
minSignal, maxSignal);
  int motor4Signal = mapRange(throttle + rollOutput + pitchOutput + yawOutput, -1.0, 1.0,
minSignal, maxSignal);
```

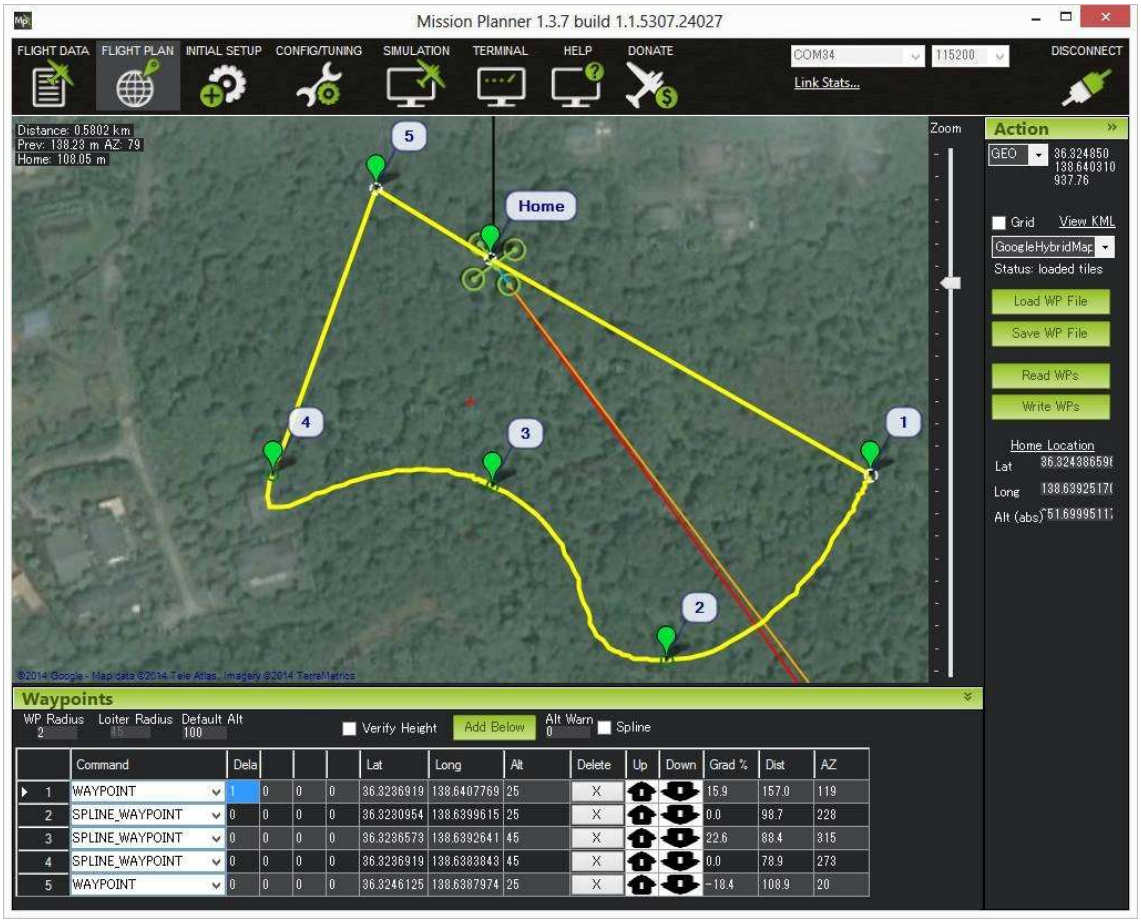
```
// Send motor signals using Servo library
esc.write(motor1Signal);
esc1.write(motor2Signal);
esc2.write(motor3Signal);
esc3.write(motor4Signal);

// Sample code to print the mapped motor signals, sensor data, and PID outputs
Serial.print("Motor 1 Signal: ");
Serial.println(motor1Signal);
Serial.print("Motor 2 Signal: ");
Serial.println(motor2Signal);
Serial.print("Motor 3 Signal: ");
Serial.println(motor3Signal);
Serial.print("Motor 4 Signal: ");
Serial.println(motor4Signal);
Serial.print("Gyro X: ");
Serial.println(gyroX);
Serial.print("Gyro Y: ");
Serial.println(gyroY);
Serial.print("Gyro Z: ");
Serial.println(gyroZ);
Serial.print("Roll Output: ");
Serial.println(rollOutput);
Serial.print("Pitch Output: ");
```

```
  Serial.println(pitchOutput);
  Serial.print("Yaw Output: ");
  Serial.println(yawOutput);

  delay(1000); // Adjust the delay as per your flight control requirements
}
```

GRAPH


Mission Planner 1.3.7 build 1.1.5307.24027
FLIGHT DATA
FLIGHT PLAN
INITIAL SETUP
CONFIG/TUNING
SIMULATION
TERMINAL
HELP
DONATE
COM34
115200
DISCONNECT
Link Stats...
Distance: 0.5802 km
Prev: 138.23 m AZ: 79
Home: 108.05 m
Home
Zoom
Action
GEO
Grid
View KML
GoogleHybridMap
Status: loaded tiles
Load WP File
Save WP File
Read WPs
Write WPs
Home Location
Lat
Long
Alt (abs)
Waypoints
WP Radius
Loiter Radius
Default Alt
Verify Height
Add Below
Alt Warn
Spline
Command
Dela
Lat
Long
Alt
Delete
Up
Down
Grad %
Dist
AZ
1 WAYPOINT 1 0 0 0 36.3236919 138.6407769 25 X 15.9 157.0 119
2 SPLINE_WAYPOINT 0 0 0 0 36.3230954 138.6399615 25 X 0.0 98.7 228
3 SPLINE_WAYPOINT 0 0 0 0 36.3236573 138.6392641 45 X 22.6 88.4 315
4 SPLINE_WAYPOINT 0 0 0 0 36.3236919 138.6383843 45 X 0.0 78.9 273
5 WAYPOINT 0 0 0 0 36.3246125 138.6387974 25 X -18.4 108.9 20


Flight Data
Flight Planner
Configuration
Simulation
Firmware
Terminal
Help
COM50
115200
Connect
Distance: 0.8993 km
Prev: 257.36 m
Home: 391.27 m
Home
Zoom
Action
Mouse Location
Lat
Long
Alt
GoogleSatellite
Status: loaded tiles
Read WPs
Write WPs
Home Location
Lat
Long
Alt (abs)
Waypoints
WP Radius
Loiter Radius
Default Alt
Hold default Alt
Verify Height
Add Below
Command
Dela
Hit Rad
Yaw Ang
Lat
Long
Alt
Delete
Up
Down
1 WAYPOINT 0 0 0 0 40.1312555 -105.1109326 100 X
2 WAYPOINT 0 0 0 0 40.1314442 -105.1090014 100 X
3 WAYPOINT 0 0 0 0 40.1309684 -105.1076925 100 X
4 WAYPOINT 0 0 0 0 40.1297133 -105.1081109 100 X
5 WAYPOINT 0 0 0 0 40.1294180 -105.1104605 100 X