DEVELOPMENT OF A FULL ATTITUDE KALMAN FILTER

By

GUNATHUNGA K.T.S. (E/19/128)

JAYASEKARA D.A.A.G.W. (E/19/158)

KULUGAMMANA K.M.W.M.B.B.V.R.B. (E/19/203)

ME 325 – MECHANICAL ENGINEERING GROUP PROJECT

Presented in partial fulfilment of the requirements for the degree of
B.Sc. Engineering degree.
University of Peradeniya

01st of December 2024

Supervised by:
Prof. D.H.S. Maithripala

Department of Mechanical Engineering
Faculty of Engineering

DECLARATION

We, the undersigned, do hereby affirm and state that this project report, “Full Attitude Kalman Filter Estimator” is an original work that was carried out by us under the supervision of Dr D.H.S Maithripala, Members of the Department of Mechanical Engineering, Faculty of Engineering, University of Peradeniya, Sri Lanka supported our efforts.

This report is submitted as part of the requirements for obtaining the B.Sc. in Mechanical Engineering degree at the University of Peradeniya

We formally claim that:

- The work detailed in this report is entirely original and has not been previously submitted for any academic or professional qualification, diploma, fellowship, or similar recognition.
- All sources of information and references used during this project have been properly acknowledged.
- Any form of assistance received throughout the execution of this project has been appropriately recognized.

We take all responsibility and accountability for the materials and their results demonstrated in this report.

Team Members

1. KULUGAMMANA K.M.W.M.B.B.V.R.B. (Registration Number: E/19/203)

Signature: ___________________

2. JAYASEKARA D.A.A.G.W. (Registration Number: E/19/158)

Signature: ___________________

3. GUNATHUNGA K.T.S. (Registration Number: E/19/128)

Signature: ___________________

CERTIFICATE OF APPROVAL

The project report entitled "**Full Attitude Kalman Filter Estimator**" submitted by:

1. KULUGAMMANA K.M.W.M.B.B.V.R.B. (Registration Number: E/19/203)
2. JAYASEKARA D.A.A.G.W. (Registration Number: E/19/158)
3. GUNATHUNGA K.T.S. (Registration Number: E/19/128)

has been evaluated and approved as part of the requirements for the **B.Sc. in Mechanical Engineering degree at the University of Peradeniya.**

Dr D.H.S. Maithripala,

Department of Mechanical Engineering,

University of Peradeniya,

Sri Lanka.

Signature: ______________

Date: __________________

ACKNOWLEDGMENTS

The successful completion of this project on the development of a fully attitude Kalman filter for drones would not have been possible without the invaluable support and guidance of many individuals.

First and foremost, I would like to express our deepest gratitude to the project supervisor and course coordinator, Dr.D.H.S. Maithripala, for his unwavering support, insightful guidance, and encouragement throughout this endeavour. His expertise and dedication have been instrumental in shaping the direction of this project and ensuring its successful completion.

We also wish to extend our gratitude to the Faculty of Engineering at the University of Peradeniya for providing the resources and conducive learning environment necessary for the execution of this project.

Special thanks go to Orise (pvt) ltd., whose support and motivation have been a constant source of inspiration. Their encouragement helped us navigate challenges and stay focused on our goals.

This project has been an incredible learning experience, and We are excited about the potential applications of this work in advancing drone technology. We are deeply thankful to everyone who contributed to this journey, directly or indirectly, and made this accomplishment possible.

Thank you all!

ABSTRACT

The development of a Full Attitude Kalman Filter (FAKF) is crucial for enabling precise and real-time orientation estimation in unmanned aerial vehicles (UAVs). This project focuses on designing and implementing robust FAKF for drones by leveraging sensor fusion techniques to integrate data from gyroscopes, accelerometers, and magnetometers. The methodology includes configuring the Navio2 flight controller, building the PX4 flight stack, and deploying the filter on a Raspberry Pi-based platform. Sensor data for angular velocity and orientation (roll, pitch, and yaw) were collected, analyzed, and visualized to evaluate system accuracy.

Despite hardware procurement challenges and library compatibility issues, the implemented FAKF demonstrated its ability to handle sensor noise and dynamic flight conditions effectively. Observations showed consistent performance in estimating angular velocities and attitude, as depicted in the plotted graphs. The results underline the potential of FAKF in improving drone stability, navigation, and control under various operating conditions.

This work highlights the applicability of advanced filtering techniques for autonomous drone systems and provides a foundation for future enhancements, including integration with machine learning for adaptive control and extension to other autonomous platforms.

.

TABLE OF CONTENTS

LIST OF FIGURES

List of Symbols and Abbreviations

FAKF	Full Attitude Kalman Filter
KF	Kalman Filter
EKF	Extended Kalman Filter
UKF	Unscented Kalman Filter
IMU	Inertial Measurement Unit
PX4	Open-source flight control software
MAVSDK	MAVLink Software Development Kit
MAVLink	Micro Air Vehicle Communication Protocol
PWM	Pulse Width Modulation
GPS	Global Positioning System
LiDAR	Light Detection and Ranging
IoT	Internet of Things
CSV	Comma-Separated Values (data file format)
GUI	Graphical User Interface

CHAPTER 1
INTRODUCTION

1.1 Background Introduction

The Full Attitude Kalman Filter (FAKF) is an advanced algorithm designed for estimating the orientation of objects in three-dimensional space. It combines data from multiple sensors, including gyroscopes, accelerometers, and magnetometers, to provide accurate and real-time attitude estimation. This capability is crucial in applications ranging from satellite control and UAV stabilization to virtual reality tracking and marine navigation. By leveraging sensor fusion, error correction, and predictive modeling—typically using quaternions—FAKF effectively handles nonlinear dynamics, sensor noise, and biases, ensuring robust performance in dynamic and uncertain environments.

FAKF is integral to a wide range of industries that require precise and real-time orientation estimation. In robotics, it enables accurate navigation and positioning for autonomous vehicles, robotic arms, and underwater exploration robots. Maritime applications rely on it for ship stabilization and control over underwater sensors. In consumer electronics, FAKF powers immersive VR/AR experiences, motion tracking in smartphones, and wearables. Defense systems utilize it for guided missiles, drones, and weapon stabilization, while the automotive industry depends on it for autonomous driving and driver assistance systems. Furthermore, in space exploration, FAKF ensures precise spacecraft docking and orientation control for planetary rovers. It also supports medical devices such as rehabilitation robotics and surgical tools, gaming technologies like motion capture and drone racing, and scientific research through the stabilization of instruments and academic experimentation.

The technical foundation of FAKF lies in its ability to model system dynamics and uncertainties effectively. Unlike simpler algorithms, it accounts for sensor noise and biases through probabilistic modeling, enabling it to predict and correct orientation estimates even in highly dynamic conditions. The use of quaternions instead of traditional Euler angles prevents issues like gimbal lock, providing a more robust representation of orientation in three-dimensional space. Moreover, the flexibility of FAKF allows for real-time implementation on embedded systems, making it a preferred choice for low-latency and computationally efficient applications. As technology evolves, advancements in FAKF are continuously improving its adaptability and precision, solidifying its relevance in emerging fields like AI-driven robotics, augmented reality, and space exploration.

1.2 Methodology Implemented for Problem-Solving

The development of the Full Attitude Kalman Filter (FAKF) followed a systematic methodology, combining hardware setup, software configuration, and sensor integration to address the challenges of real-time attitude estimation. The following steps outline the process

1. Raspberry Pi OS Installation

- The **Raspberry Pi OS** was installed to establish the operating system on the Raspberry Pi. The official OS can be downloaded from the Raspberry Pi website at: https://www.raspberrypi.com/software/.
- **PuTTY** was installed to enable remote command-line access to the Raspberry Pi, while **RealVNC Viewer** was configured for remote desktop access. These tools simplified the monitoring and configuration of the Raspberry Pi environment.
- Initial **sensor readings** were obtained from the Raspberry Pi connected with Navio 2. Sensors such as the accelerometer, gyroscope, and magnetometer were interfaced and calibrated. This data was used to verify the sensor integration and ensure the accuracy of raw measurements.

2. QgroundControl Installation

- **QGroundControl** was installed as a ground control station software to monitor and manage the UAV or test platform. The installation followed the steps outlined in the PX4 Documentation at: https://docs.px4.io/main/en/dev_setup/qgc_daily_build.html.

3. Raspberry PI OS for Navio2

- The **Emlid Raspberry Pi OS image for Navio 2** was downloaded and installed from the Emlid Documentation available at: https://docs.emlid.com/navio2/configuring-raspberry-pi/. 6This OS is optimized for Navio 2 and ensures compatibility with the onboard sensors and flight control hardware.

4. Ubuntu 18.04 Dual Boot Installation

- To support the development and testing of the PX4 firmware, **Ubuntu 18.04** was installed alongside Windows as a dual-boot configuration. The installation was guided by the HackerNoon tutorial, which can be found at: https://medium.com/hackernoon/installing-ubuntu-18-04-along-with-windows-10-dual-boot-installation-for-deep-learning-f4cd91b58557.5.
- The dual-boot setup provided a stable Linux environment for compiling and deploying the PX4 firmware.

5.PX4 Build and Development

- The **PX4 Autopilot firmware** was built in the Ubuntu environment, adhering to the instructions in the PX4 Documentation available at: https://docs.px4.io/main/en/flight_controller/raspberry_pi_navio2.html#px4-development-environment.
- After successful compilation, the firmware was uploaded to the Raspberry Pi equipped with the Navio 2 flight controller. This established the communication framework required for sensor data acquisition and attitude estimation.

6.MAVSDK Installation

- The **MAVSDK Python library** was installed to enable communication and interaction with the PX4-based system. The installation followed the guidelines provided in the MAVSDK GitHub repository at: https://github.com/mavlink/MAVSDK-Python.
- MAVSDK facilitated programming high-level operations for UAV testing, including collecting telemetry data, issuing commands, and monitoring system states.

7.Sensor Readings

- Sensor readings from the Navio 2 flight controller were continuously monitored.
- The accelerometer provided linear acceleration measurements, the gyroscope tracked angular velocity, and the magnetometer offered heading information.

1.3 Future Scope

The Full Attitude Kalman Filter (FAKF) holds immense potential for future advancements across various domains, driven by its ability to deliver accurate and real-time orientation estimation. As technology continues to evolve, the FAKF's applications are poised to expand and address emerging challenges in cutting-edge fields.

In the **aerospace industry**, the FAKF can be developed further to handle increasingly complex satellite formations, enabling precise coordination among multiple satellites in orbit. It also offers significant potential for deep-space exploration, providing robust orientation estimation for spacecraft operating in harsh and uncertain environments. The rise of advanced drone swarms with higher autonomy could also benefit from FAKF, enabling coordinated operations for tasks such as disaster response, surveillance, and environmental monitoring.

In **robotics**, the integration of FAKF with machine learning and artificial intelligence opens new possibilities for adaptive and context-aware attitude estimation. This advancement can enhance the capabilities of humanoid robots, improve efficiency in industrial automation, and provide superior navigation for autonomous vehicles operating in dynamic and unpredictable environments.

The growth of **augmented and virtual reality (AR/VR)** technology presents another area where FAKF can drive innovation. With ultra-low latency and enhanced motion tracking, FAKF can significantly improve the quality of immersive AR/VR experiences, making them more responsive and realistic for gaming, training simulations, and remote collaboration.

In the **healthcare sector**, FAKF can contribute to advancements in precision for robotic surgeries and the development of exoskeletons. These improvements could support rehabilitation programs by offering better mobility assistance and patient support, thereby enhancing quality of life and recovery outcomes.

For **autonomous systems** in the automotive and marine industries, the FAKF's role in self-driving cars and underwater vehicles will grow as sensor fusion techniques improve. Its ability to provide reliable orientation estimation in complex, unstructured terrains will be crucial for navigating diverse environments, from urban streets to deep-sea exploration.

In **defense applications**, the FAKF can be leveraged for next-generation hypersonic vehicles and advanced missile guidance systems, where precise attitude estimation is critical for success. Its robustness in dynamic and high-speed conditions ensures its relevance in this domain.

The advent of **quantum computing** and advanced sensor technologies presents opportunities to further optimize FAKF algorithms. This could result in reduced computational costs, improved accuracy, and faster processing speeds, making the filter more accessible for real-time applications across a wider range of devices and systems.

Additionally, as the **Internet of Things (IoT)** and **smart devices** proliferate, the FAKF can enhance orientation estimation for interconnected systems in smart cities, wearables, and beyond. Its integration into IoT networks could enable more efficient and synchronized operations, supporting applications like smart homes, automated delivery systems, and intelligent infrastructure monitoring.

The Full Attitude Kalman Filter's versatility and adaptability ensure its continued significance in driving innovation across industries, solidifying its position as a cornerstone of modern and emerging technologies.

CHAPTER 2

LITERITURE REVIEW

2.1 Introduction

The Kalman Filter (KF) is a widely recognized algorithm for optimal state estimation in dynamic systems, proposed by Rudolf E. Kalman in 1960. Due to its efficiency, real-time capabilities, and robustness, it has been extensively applied in various domains, including navigation, robotics, aerospace, and signal processing. Over time, the KF has been extended to address nonlinear systems, resulting in algorithms like the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF). This chapter provides an in-depth review of these algorithms, focusing on their applications in attitude estimation and their limitations, which motivate the development of the Full Attitude Kalman Filter (FAKF).

2.2 Overview of Kalman Filter and Related Algorithms

2.2.1 Kalman Filter (KF)

The KF is a linear state estimation technique that combines predictions from a system model with noisy observations to produce optimal estimates. It operates through two stages: prediction and update. The algorithm assumes linear dynamics and Gaussian noise, which limits its applicability to linear systems. Despite its limitations, the KF has demonstrated significant utility in applications like spacecraft orbit calculation, GPS navigation, and sensor fusion in autonomous systems. However, the assumption of linearity restricts its effectiveness in real-world scenarios involving nonlinear dynamics.

2.2.2 Extended Kalman Filter (EKF)

The EKF addresses the KF's linearity limitation by linearizing nonlinear models through first-order Taylor expansion. It is widely used in attitude estimation, where nonlinear dynamics are prevalent. For instance, EKF has been employed in vehicle tracking and spacecraft orbit control. However, its reliance on linear approximations can lead to significant errors when the system deviates strongly from linearity, resulting in filter divergence.

2.2.3 Unscented Kalman Filter (UKF)

The UKF improves upon the EKF by using a deterministic sampling technique called the Unscented Transform (UT) to capture the mean and covariance of nonlinear transformations. This approach avoids the need for linearization and enhances accuracy. UKF has been applied in target tracking, sensor fusion for UAVs, and depth estimation in computer vision. However, its higher computational requirements can limit real-time applications.

2.3 Role of Sensor Technologies

2.3.1 Inertial Measurement Units (IMUs)

IMUs, comprising accelerometers, gyroscopes, and magnetometers, are essential for attitude estimation. They provide raw data on linear acceleration, angular velocity, and magnetic fields. However, individual sensors suffer from noise and drift, necessitating sensor fusion techniques to improve reliability.

2.3.2 Sensor Fusion

Sensor fusion techniques combine data from multiple sensors to produce more accurate estimates. Kalman-based algorithms are widely used for this purpose, effectively mitigating individual sensor limitations and ensuring robustness.

2.4 Applications of Attitude Estimation

Attitude estimation is critical in various fields:

- **Aerospace**: Used in satellite orientation control, spacecraft docking, and UAV stabilization.
- **Robotics**: Enables precise navigation for autonomous vehicles and robotic arms.
- **Consumer Electronics**: Powers VR/AR motion tracking and smartphone sensors.
- **Healthcare**: Supports robotic surgeries and rehabilitation devices.

These applications highlight the growing demand for more robust and accurate attitude estimation algorithms like the FAKF.

2.5 Gaps in Existing Research

While existing algorithms like KF, EKF, and UKF have proven effective in specific scenarios, they exhibit limitations:

1. **Linearity Assumptions**: KF and EKF are constrained by their reliance on linear models, reducing accuracy in highly nonlinear systems.
2. **Computational Complexity**: UKF, while accurate, incurs high computational costs, making it unsuitable for real-time embedded systems.
3. **Noise Handling**: Standard algorithms struggle to maintain robustness in the presence of significant sensor noise and biases.

These challenges necessitate the development of the FAKF, which leverages advanced modeling techniques and enhanced sensor fusion capabilities to overcome these limitations.

2.6 Summary

This chapter reviewed the theoretical foundations, advancements, and applications of the Kalman Filter and its extensions. It also highlighted the limitations of existing algorithms in addressing nonlinear dynamics, noise, and real-time requirements. The findings underscore the need for developing the Full Attitude Kalman Filter (FAKF) to meet the demands of modern and emerging technologies.

CHAPTER 3

METHODOLOGY

3.1 Raspberry Pi Installation and Sensor Readings

3.1.1 Introduction

The Raspberry Pi was selected as the primary computational platform due to its affordability, versatility, and compatibility with the Navio 2 shield. As a compact yet powerful microcomputer, the Raspberry Pi provides the necessary computational resources to implement and execute the Full Attitude Kalman Filter (FAKF) in real time. It serves as the hub for collecting and processing sensor data, running the PX4 firmware, and interfacing with external control software.

Attitude estimation is inherently a computationally intensive task, requiring the efficient integration of multiple data streams from sensors such as accelerometers, gyroscopes, magnetometers, and barometers. The Raspberry Pi's ability to handle multitasking and process multiple input/output operations simultaneously makes it an ideal choice for this project.

This phase of the methodology focused on setting up the Raspberry Pi hardware, installing the operating system, and establishing the software environment necessary for seamless communication with the Navio 2 shield. Additionally, this step included configuring the hardware and network settings to ensure reliable remote access for monitoring and development.

Given the real-time requirements of attitude estimation, special attention was paid to optimizing the Raspberry Pi for computational efficiency. Cooling mechanisms were added to mitigate thermal throttling during intensive operations, and the system was configured to maximize its processing capabilities. Remote access tools, including PuTTY and RealVNC Viewer, were installed to allow for flexible monitoring and management, facilitating iterative development and debugging without the need for direct physical interaction with the hardware.

In this phase, an emphasis was also placed on integrating the onboard sensors effectively. This included the calibration of accelerometers, gyroscopes, magnetometers, and barometers to ensure that the raw data obtained was accurate and free from biases. These sensors form the cornerstone of the attitude estimation process, as they provide the necessary measurements for calculating orientation and position. Preprocessing techniques such as noise filtering and bias correction were employed to enhance the reliability of the collected data.

The setup process detailed here provided the foundational hardware and software infrastructure required for subsequent stages, including firmware development, sensor integration, and Full Attitude Kalman Filter implementation. By addressing potential bottlenecks early in the process and ensuring that the system was robust and reliable, this phase set the stage for successful execution and testing of the FAKF.

3.1.2 Raspberry Pi Installation

The Raspberry Pi serves as the primary computational platform in this project, chosen for its cost-effectiveness, adaptability, and robust community support. Its small form factor and ability to interface with a variety of peripherals make it an ideal choice for implementing real-time systems such as the Full Attitude Kalman Filter (FAKF). This section details the steps taken to set up the Raspberry Pi hardware and software environment.

Hardware Preparation

The Raspberry Pi 4 Model B was selected due to its quad-core ARM Cortex-A72 processor and up to 4GB or 8GB of RAM, depending on the variant. These specifications provide sufficient computational power for running the PX4 firmware, processing sensor data, and implementing the Kalman Filter algorithm in real time.

Key steps in hardware preparation included:

1. **Power Supply**: A high-quality 5V, 3A power adapter was used to ensure stable power delivery. Unstable power can cause performance degradation or even system crashes.
2. **Storage**: A 32GB microSD card was selected to store the operating system and application data. This capacity allows for logging telemetry data while leaving ample room for system files and software installations.
3. **Cooling Solutions**: To prevent overheating during prolonged operations, heatsinks were attached to the Raspberry Pi’s main components, and a fan was installed. Overheating can lead to thermal throttling, reducing performance and potentially causing computational delays.

Installing Raspberry Pi OS

The Raspberry Pi OS, a Debian-based operating system, was chosen due to its compatibility with the Raspberry Pi hardware and extensive software repositories. The Lite version of Raspberry Pi OS was used to optimize system performance by removing unnecessary graphical user interface elements.

Steps involved in the installation process:

1. **Download and Flash the OS**:
    - The Raspberry Pi OS Lite image was downloaded from https://www.raspberrypi.com/software/.
    - The image was flashed onto the microSD card using software such as Balena Etcher or the Raspberry Pi Imager.
2. **Initial Configuration**:
    - The Raspberry Pi was booted up, and the raspi-config tool was used to set basic configurations such as:
        - Enabling SSH for remote command-line access.
        - Expanding the filesystem to utilize the full capacity of the microSD card.
        - Setting up the hostname and time zone for synchronization with the system clock.
3. **System Optimization**:
    - Swapping was disabled to preserve the microSD card’s lifespan, as excessive swapping can lead to wear.
    - System updates were performed to ensure all software packages were up-to-date

4. **Networking and Remote Access**

Remote access was configured to allow flexibility in managing the Raspberry Pi without requiring physical access. This step was critical for iterative testing and debugging.

1. **PuTTY Configuration**:
   - PuTTY, a lightweight SSH client, was configured to access the Raspberry Pi's terminal from a host PC. This allowed for efficient command execution and file transfers.
2. **Wi-Fi Setup**:
   - The Raspberry Pi was connected to a stable Wi-Fi network. Static IP assignment was configured to ensure consistent connectivity, simplifying access during development.

3.1.3 Selecting Sensors and taking Sensor Readings

The integration of sensors was one of the most critical components of the project. Accurate and reliable sensor data serves as the backbone of the Full Attitude Kalman Filter, enabling precise state estimation. This section details the selection of sensors, their roles, and the methods used to obtain and preprocess data.

Sensor Selection and Integration

The following sensors were chosen for their specific roles in attitude estimation:

1. **Accelerometer (ICM20602)**:
   - Measures linear acceleration along three axes (X,Y,ZX, Y, ZX,Y,Z).
   - Provides data for calculating tilt and detecting changes in motion.
   - Critical for detecting the direction of gravity, which helps in estimating orientation.

2. **Gyroscope (ICM20602)**:
    - Tracks angular velocity, providing essential data for calculating rotational movement.
    - Plays a key role in predicting orientation over time, particularly during rapid movements.
3. **Magnetometer (QMC5883L)**:
    - Measures the Earth’s magnetic field to determine heading information.
    - Complements accelerometer data to provide a full orientation picture.
4. **Barometer (BMP388)**:
    - Measures atmospheric pressure, translating to altitude estimation.
    - Adds a vertical dimension to the attitude estimation, improving overall accuracy.

These sensors were integrated using the Navio 2 board, which provides built-in support for IMUs and other peripherals.

Sensor Calibration

Calibration is vital to ensure sensor accuracy, as raw readings often include biases and offsets that can compromise performance.

1. **Static Calibration**:
    - Sensors were kept stationary to measure and eliminate biases. For instance, the gyroscope’s static readings were analyzed to identify and compensate for drift.

2. **Dynamic Calibration**:
    - The system was subjected to controlled movements to characterize its response to real-world scenarios. Calibration parameters were adjusted accordingly.
3. **Magnetometer Hard and Soft Iron Corrections**:
    - Magnetic interference from surrounding objects (hard and soft iron effects) was accounted for using calibration routines, ensuring accurate heading measurements.

Taking Sensor Readings

Sensor readings were captured in real time using the PX4 firmware running on the Raspberry Pi.

1. **Data Acquisition**:
    - Sensor readings were polled at high frequency (e.g., 100 Hz) to capture rapid changes in orientation and movement.
    - Data streams from each sensor were logged to validate their integrity and performance under various conditions.
2. **Data Preprocessing**:
    - A **low-pass filter** was applied to remove high-frequency noise, which can cause instability in state estimation.
    - **Bias Removal**: Continuous monitoring was performed to adjust for any sensor drift observed during operation.
    - Time synchronization ensured that data from multiple sensors was accurately aligned, a critical step for effective fusion.

3. **Logging and Analysis**:
   - Sensor data was logged to CSV files for offline analysis. Visualization tools such as Matplotlib and Pandas were used to plot and inspect trends in the data.
   - Outlier detection algorithms were applied to identify and handle anomalous readings, ensuring reliability.

Rationale for These Steps

The steps taken in this section were designed to achieve:

1. **Accuracy**: Through calibration and noise filtering, the raw sensor data was refined to improve the reliability of the Kalman Filter.
2. **Robustness**: Addressing issues such as drift, interference, and outliers ensured that the system could handle real-world scenarios.
3. **Flexibility**: The modular integration of sensors allowed for easy replacement or upgrades, future-proofing the system for additional requirements.

By focusing on these aspects, the groundwork was laid for integrating sensor data into the Kalman Filter and achieving precise and real-time attitude estimation.

3.2 Model Development

Model development is a crucial phase in this project as it bridges hardware setup and algorithmic implementation. It involves configuring the software stack, establishing the flight controller's operational environment, and enabling real-time data collection and processing. This section elaborates on the installation of required software, configuring the Raspberry Pi for drone control, and building and uploading the PX4 firmware to the Raspberry Pi with the Navio 2 board.

3.2.1 Installing Required Software for Drone Control

The functionality of the Full Attitude Kalman Filter (FAKF) hinges on a robust software environment that integrates with the hardware and enables seamless sensor data processing. Installing and configuring the necessary software is the first step in building this environment. This includes tools for monitoring, debugging, and controlling the system.

1. **QGroundControl**
    - **Purpose**: QGroundControl is a ground control station software used to interface with drones. It provides features such as flight planning, sensor calibration, and real-time telemetry visualization.
    - **Installation Steps**:
        - The software was downloaded from https://docs.px4.io/main/en/dev_setup/qgc_daily_build.html.
        - It was installed on the host PC for easier access and real-time monitoring.
    - **Key Features Used**:
        - **Sensor Calibration**: Ensured that all onboard sensors (accelerometers, gyroscopes, and magnetometers) operated within expected parameters.
        - **Telemetry Monitoring**: Allowed the team to observe sensor data streams and validate system stability.
        - **Mission Planning**: Though not directly relevant to this project, QGroundControl's mission planning tools provided insights into navigation and control algorithms.

2. **Emlid Raspberry Pi OS for Navio 2**
   - **Purpose**: This customized Raspberry Pi OS ensures compatibility with the Navio 2 shield, enabling proper operation of its sensors and peripherals.
   - **Installation Steps**:
     - The OS image was downloaded from https://docs.emlid.com/navio2/configuring-raspberry-pi/.
     - Flashed onto an SD card using Balena Etcher.
     - During setup, drivers specific to the Navio 2 board were loaded to activate its IMU and other peripherals.
   - **Importance**:
     - Standard Raspberry Pi OS may lack the necessary drivers and configurations for Navio 2. This specialized OS bridges that gap and optimizes performance.
3. **Ubuntu 18.04 Dual Boot Setup**
   - **Purpose**: Ubuntu 18.04 was chosen as the development environment for PX4 due to its compatibility with PX4 development tools and stable support for software dependencies.
   - **Installation Steps**:
     - A dual-boot configuration was established following the guide at https://medium.com/hackernoon/installing-ubuntu-18-04-along-with-windows-10-dual-boot-installation-for-deep-learning-f4cd91b58557.
     - Ubuntu was installed alongside Windows to enable seamless switching between general-purpose and development-specific environments.
   - **Why Ubuntu 18.04?**:
     - PX4 officially supports Ubuntu, and version 18.04 was selected for its long-term support (LTS), ensuring compatibility and security.

3.2.2 Building PX4 Firmware

PX4 is an open-source flight control firmware that provides a flexible framework for controlling drones and other robotic systems. Building and customizing this firmware for the Raspberry Pi with Navio 2 is a critical step in enabling real-time communication between sensors and the Full Attitude Kalman Filter.

1. Cloning the PX4 Repository
    - Purpose: To obtain the source code for PX4, including configurations and drivers for Navio 2.
    - Steps:
        - The PX4 repository was cloned using:

          https://github.com/PX4/PX4-Autopilot.git --recursive
        - The recursive flag ensured that all submodules were cloned, as they contain vital components like drivers and scripts.

2. Installing Dependencies
    - **Purpose**: The PX4 build process requires specific software libraries and tools.
    - **Steps**:
        - Dependencies were installed using the provided scripts.
        - Additional dependencies like Pillow were manually installed to resolve compatibility issues.

3. Customizing the Firmware for Navio 2
    - **Purpose**: Ensure that the PX4 firmware supports the specific peripherals available on Navio 2.
    - **Modifications Made**:
        - Configuration files were edited to define the peripherals (e.g., ICM20602 accelerometer, QMC5883L magnetometer, BMP388 barometer).
        - Drivers were verified and, if necessary, recompiled to match the Navio 2's hardware specifications(Compiling PX4 for Navio2).
4. Compiling the Firmware
    - The PX4 firmware was built using the provided documentation
    - Errors were resolved by referring to documentation and logs. Debugging involved modifying the codebase to address hardware-specific issues.
5. Uploading the Firmware
    - **Purpose**: Transfer the compiled firmware to the Raspberry Pi for execution.
    - The system was tested after uploading to ensure that sensors and peripherals were initialized correctly.

6. Running the Firmware
    - Running PX4 validates that the firmware communicates with all hardware components, enabling data acquisition and processing in real time.

Significance of These Steps

1. **System Compatibility**:
   - Ensuring compatibility between hardware (Navio 2) and software (PX4) is critical for reliable operations.
2. **Flexibility**:
   - The modularity of PX4 allows for future customization and expansion, such as adding new sensors or modifying flight control algorithms.
3. **Real-Time Operation**:
   - Proper firmware setup ensures real-time data flow, which is essential for the Full Attitude Kalman Filter.

The detailed setup of QGroundControl, Navio 2-specific OS, and PX4 firmware laid the foundation for robust communication between hardware and software. This section addressed critical challenges such as sensor integration, firmware customization, and real-time data processing, ensuring a stable and scalable platform for implementing and testing the Full Attitude Kalman Filter.

3.3 Taking Sensor Readings

Sensor readings are at the core of the Full Attitude Kalman Filter (FAKF), as they provide the raw data needed to estimate orientation, velocity, and position. This section describes the process of setting up the Raspberry Pi and Navio 2 for data acquisition, installing MAVSDK for real-time communication with the PX4 firmware, and logging, preprocessing, and validating sensor data.

3.3.1 MAVSDK Installation

MAVSDK (MAVLink Software Development Kit) is a high-level library designed to simplify communication with PX4-based systems. It allows for telemetry monitoring, command execution, and data acquisition. Integrating MAVSDK with PX4 running on the Raspberry Pi enables seamless interaction between the flight controller and external software.

Installation Steps

1. **Installing MAVSDK**
    - MAVSDK was installed using its Python implementation, which provides robust APIs for telemetry and control.
    - The library was tested to ensure compatibility with the PX4 firmware on the Raspberry Pi.
2. **Validating MAVSDK Communication**
    - MAVSDK was configured to connect to the PX4 firmware over MAVLink. The connection was verified by running a sample telemetry script, ensuring the SDK could retrieve real-time sensor data.

Key Features Used

- **Telemetry Monitoring**: Continuous streams of data from onboard sensors, including IMU, barometer, and magnetometer.
- **Command Execution**: The ability to execute commands like sensor reinitialization and real-time parameter updates.
- **Logging**: Automated logging of data streams for offline analysis.

Rationale for MAVSDK

MAVSDK provides a clean abstraction layer for interacting with PX4, reducing the complexity of implementing low-level communication protocols like MAVLink. Its Python API allows for quick integration and testing, which is critical during iterative development.

3.3.2 Acquiring and Logging Data

Purpose

Sensor readings form the input to the FAKF, enabling it to estimate the system's state. Acquiring and logging this data involves reading raw measurements, synchronizing multiple data streams, and applying initial preprocessing.

Steps for Data Acquisition

1. Executing PX4 Firmware
    - PX4 was run on the Raspberry Pi with all peripherals (IMU, barometer, magnetometer) initialized.
2. Establishing MAVSDK Connection
    - A connection to the PX4 system was established through MAVSDK. The script continuously polled sensor data and logged it to local files.

3. Telemetry Data Streams

    - Data streams were captured at 100 Hz, ensuring high-resolution readings for accurate state estimation. Key data included:
        - Accelerometer: Linear acceleration.
        - Gyroscope: Angular velocity.
        - Magnetometer: Heading relative to Earth's magnetic field.
        - Barometer: Atmospheric pressure for altitude determination.

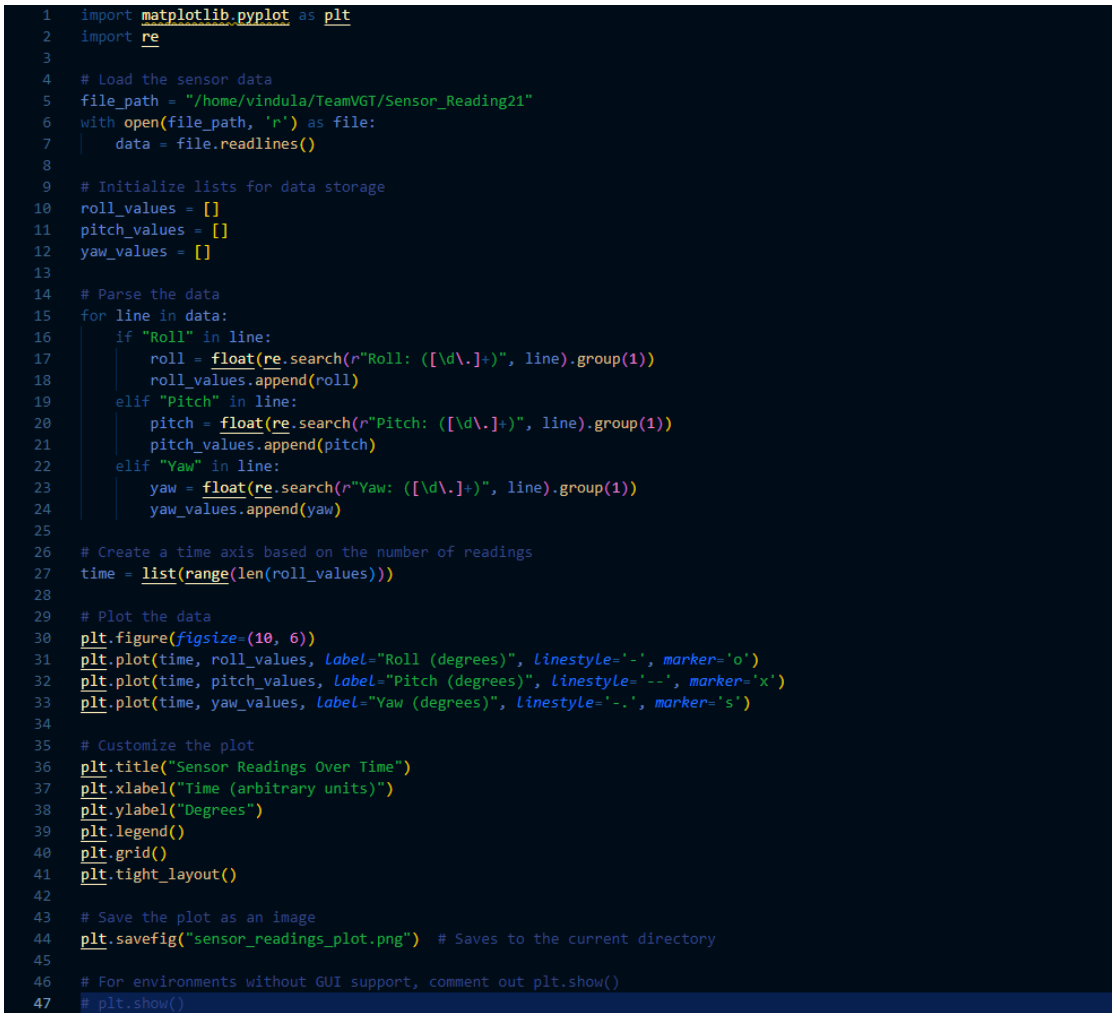
Data Logging and Validation

1. Logging Data

    - Sensor readings were stored in CSV format for offline analysis. Each record included timestamps to facilitate synchronization between different sensors.

2. Visualization

    Logged data was visualized using Python libraries such as Matplotlib and Seaborn. Graphs were plotted to observe trends, check consistency, and identify anomalies.



```python
import matplotlib.pyplot as plt
import re

# Load the sensor data
file_path = "/home/vindula/TeamVGT/Sensor_Reading21"
with open(file_path, 'r') as file:
    data = file.readlines()

# Initialize lists for data storage
roll_values = []
pitch_values = []
yaw_values = []

# Parse the data
for line in data:
    if "Roll" in line:
        roll = float(re.search(r"Roll: ([\d\.]+)", line).group(1))
        roll_values.append(roll)
    elif "Pitch" in line:
        pitch = float(re.search(r"Pitch: ([\d\.]+)", line).group(1))
        pitch_values.append(pitch)
    elif "Yaw" in line:
        yaw = float(re.search(r"Yaw: ([\d\.]+)", line).group(1))
        yaw_values.append(yaw)

# Create a time axis based on the number of readings
time = list(range(len(roll_values)))

# Plot the data
plt.figure(figsize=(10, 6))
plt.plot(time, roll_values, label="Roll (degrees)", linestyle='-', marker='o')
plt.plot(time, pitch_values, label="Pitch (degrees)", linestyle='--', marker='x')
plt.plot(time, yaw_values, label="Yaw (degrees)", linestyle='-.', marker='s')

# Customize the plot
plt.title("Sensor Readings Over Time")
plt.xlabel("Time (arbitrary units)")
plt.ylabel("Degrees")
plt.legend()
plt.grid()
plt.tight_layout()

# Save the plot as an image
plt.savefig("sensor_readings_plot.png")  # Saves to the current directory

# For environments without GUI support, comment out plt.show()
# plt.show()
```

Figure 01.: python code for plotting graphs

3.Validation Against Ground Truth

- Sensor readings were compared with known references:
    - IMU readings were validated by placing the system in a stationary position and ensuring accelerometer output aligned with gravity (9.8 m/s2).
    - Magnetometer readings were cross-checked with a compass.
    - Barometer outputs were verified against local weather data for pressure.

3.3.3 Data Preprocessing

Purpose

Preprocessing ensures that the raw data is free from noise, biases, and outliers. It prepares the data for input into the Kalman Filter, improving the reliability of state estimation.

Steps for Preprocessing

1. **Noise Filtering**
   - A low-pass filter was applied to accelerometer and gyroscope data to remove high-frequency noise caused by vibrations and electrical interference.
2. **Bias Removal**
   - Sensor biases were calculated by averaging readings over a stationary period. These biases were subtracted from raw data to produce corrected values: [Corrected Reading=Raw Reading−Bias]
3. **Outlier Detection**
   - Outliers were identified using statistical thresholds based on the standard deviation of data. Values exceeding 3σ were flagged and replaced with interpolated data.
4. **Time Synchronization**
   - Timestamps from all sensors were aligned to ensure data consistency. Linear interpolation was used to fill gaps in data streams where necessary.

3.3.4 Integration with Kalman Filter

Purpose

Preprocessed sensor data is fed into the Kalman Filter as observations, enabling the filter to estimate the system's state. Proper integration ensures that the Kalman Filter operates effectively and produces reliable outputs.

Steps for Integration

1. **Defining the Observation Model**
    - Observation matrices were constructed to map raw sensor readings to the filter's state variables.
2. **Initializing the Filter**
    - Initial estimates of the state vector and covariance matrix were based on sensor specifications and calibration data.

Significance of These Steps

1. **Improved Accuracy**:
    - Preprocessing removed noise and bias, ensuring high-quality data for state estimation.
2. **System Robustness**:
    - Validating sensor readings against ground truth increased confidence in the system's reliability.
3. **Real-Time Readiness**:
    - The integration with MAVSDK and PX4 ensured that the system could operate in real-time, a critical requirement for dynamic applications like UAV control.

This section details how the sensor data was acquired, processed, and integrated into the Full Attitude Kalman Filter, establishing a reliable foundation for accurate state estimation.

3.4 Theoretical Framework: Kalman Filter

The Kalman Filter (KF) is a powerful algorithm used for estimating the state of a dynamic system in real time, even when the measurements are noisy or incomplete. It provides an optimal solution to the state estimation problem in systems governed by linear dynamics with Gaussian noise. The theoretical foundation of the Kalman Filter is rooted in linear algebra and probability theory, making it a cornerstone in fields like robotics, aerospace, signal processing, and finance.

This section delves into the theoretical framework of the Kalman Filter, focusing on its mathematical formulation, assumptions, and applications in attitude estimation.

3.4.1 Problem Statement

State estimation is the process of determining the internal state of a system based on noisy observations and an understanding of the system’s dynamics. In the context of this project:

- The **state** represents the orientation and angular velocity of the system.
- The **observations** are noisy sensor readings from IMUs (accelerometer, gyroscope, and magnetometer).

The challenge is to fuse these noisy observations and the system model to estimate the true state accurately and efficiently in real time.

3.4.2 Assumptions of the Kalman Filter

The Kalman Filter relies on the following assumptions:

1. **Linear Dynamics**:
   - The system is governed by linear state transition and observation models.
2. **Gaussian Noise**:
   - Both process noise and observation noise are Gaussian-distributed with known covariance matrices Q and R, respectively.
3. **Complete Observability**:
   - The state must be observable, meaning it can be reconstructed from the observations.

3.4.3 Mathematical Formulation

The Kalman Filter operates in two steps: **Prediction** and **Update**.

Prediction Step

The prediction step estimates the state at the next time step using the system's dynamic model:

1. **State Prediction**
2. **Covariance Prediction**:
   - Predicted state.
   - Predicted covariance matrix.

Update Step

The update step refines the prediction based on new observations:

1. **Kalman Gain**
2. **State Update**
3. **Covariance Update**

3.4.4 Recursive Nature of the Kalman Filter

The Kalman Filter is recursive, meaning it does not require storing past data. Instead, it uses the current state estimate and observations to compute the next state, making it computationally efficient for real-time applications.

3.4.5 Application in Attitude Estimation

In the context of this project, the Kalman Filter is adapted to estimate the orientation (attitude) of a system using data from IMUs.

Process Model:

- The state transition matrix (A) is derived from the quaternion kinematics and the gyroscope readings.

Observation Model:

- The observation matrix (H) relates sensor measurements (accelerometer and magnetometer readings) to the state vector.

3.4.6 Noise Handling

Noise in sensor readings, such as gyroscope drift and accelerometer noise, is modeled using Gaussian distributions. The Kalman Filter accounts for this noise in its prediction and update steps, weighting the observations based on their expected accuracy (covariance matrix R).

3.4.7 Advantages of the Kalman Filter

1. **Optimality**:
   - The Kalman Filter minimizes the mean squared error of the state estimate under Gaussian noise assumptions.
2. **Real-Time Computation**:
   - Its recursive nature makes it suitable for real-time applications.
3. **Flexibility**:
   - Can be adapted to various systems and noise characteristics by tuning the covariance matrices Q and R.

3.4.8 Limitations of the Standard Kalman Filter

1. **Linearity Assumption**:
   - The Kalman Filter assumes linear dynamics, which may not hold in complex systems like attitude estimation.
2. **Gaussian Noise Assumption**:
   - Performance degrades if the noise is non-Gaussian.
3. **Computational Constraints**:
   - High-dimensional state vectors can increase computational load, challenging real-time implementation on resource-limited hardware.

3.4.9 Adaptations for Nonlinear Systems

Since attitude estimation involves nonlinear dynamics, extensions of the Kalman Filter, such as the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF), are often used. These adapt the filter to nonlinear systems by:

- EKF: Linearizing the dynamics using a Taylor expansion.
- UKF: Using deterministic sampling (unscented transform) to handle nonlinearity without explicit linearization.

3.5 Full System Integration

The integration of hardware, firmware, and software components is a critical step in the development of the Full Attitude Kalman Filter (FAKF). This process ensures seamless communication and coordination among all components to enable real-time attitude estimation. This section provides a detailed explanation of how the Raspberry Pi, Navio 2 board, PX4 firmware, MAVSDK, and sensor systems were combined to create a cohesive system capable of processing data, running algorithms, and delivering reliable results.

3.5.1 Purpose of Full System Integration

1. **Unified Functionality**:
    - To ensure that all individual components—hardware, firmware, and software—work together without conflicts.
2. **Real-Time Operation**:
    - To achieve synchronized data acquisition, processing, and output in real time, which is critical for dynamic applications such as UAVs.

3. **Reliability**:
    - To validate the system's ability to operate under various conditions, including high-frequency updates and noisy sensor environments.

3.5.2 Hardware Integration

Raspberry Pi Setup

1. The Raspberry Pi served as the primary computational unit, chosen for its processing capabilities and compatibility with the Navio 2 shield.
2. **Power Management**:
    - A stable 5V, 3A power supply ensured consistent power delivery, avoiding system crashes or data loss.

Navio 2 Shield

1. The Navio 2 shield was mounted on the Raspberry Pi, providing an array of sensors (IMU, barometer, magnetometer) and connectors for peripherals.
2. It also provided pulse-width modulation (PWM) outputs for actuators, though these were not used in this project.

3.5.3 Firmware Integration

The PX4 firmware was the central component of the system, interfacing between the hardware and software layers.

1. **Building and Uploading PX4**:
    - The firmware was customized and compiled for the Navio 2 board. It was uploaded to the Raspberry Pi, enabling communication with onboard sensors.
    - Commands such as the following were used to initialize sensors and start the firmware:

2. **Firmware Features Utilized**:
    - **Sensor Drivers**:
        - PX4 provided built-in drivers for the accelerometer, gyroscope, magnetometer, and barometer. These drivers ensured reliable data acquisition from the Navio 2 shield.
    - **MAVLink Protocol**:
        - PX4 communicated with MAVSDK via the MAVLink protocol, a lightweight messaging system designed for UAVs and robotics.
    - **Parameter Management**:
        - Sensor calibration parameters were stored in the firmware and could be updated through commands or QGroundControl.

3.5.4 Software Integration

MAVSDK

1. **Telemetry Monitoring**:
    - MAVSDK was used to fetch real-time sensor data streams and system status from the PX4 firmware.
2. **Command Execution**:
    - Commands for reinitializing sensors, adjusting parameters, or starting specific routines were executed through MAVSDK.
3. **Data Logging**:
    - Sensor readings were logged in structured formats (e.g., CSV) for offline analysis and debugging.

Python-Based Processing Scripts

1. Scripts were developed to parse, preprocess, and visualize sensor data. For example, roll, pitch, and yaw values were extracted and plotted using libraries like Matplotlib and Seaborn.
2. The scripts also performed statistical validation to detect anomalies or inconsistencies in the sensor data.

3.5.5 Data Flow and Synchronization

1. **Data Flow**:
   - The sensors (via Navio 2) provided raw data to the PX4 firmware.
   - PX4 processed this data and transmitted it to MAVSDK using the MAVLink protocol.
   - MAVSDK acted as a bridge, providing the data to Python scripts for further analysis and integration into the Kalman Filter.
2. **Synchronization**:
   - Time-stamped data from multiple sensors was aligned to ensure consistency. This synchronization was critical for accurate state estimation.

3.5.6 Validation and Testing

Hardware Validation

1. Each sensor was tested individually to ensure proper functioning before full integration.
2. The Raspberry Pi and Navio 2 were tested under varying load conditions to validate power stability and thermal performance.

Firmware Validation

1. Sensor readings from PX4 were compared with ground truth values (e.g., accelerometer readings were validated against gravity).
2. Firmware logs were analyzed to detect initialization issues or runtime errors.

Software Validation

1. Python scripts were tested with simulated data to ensure proper parsing, preprocessing, and visualization workflows.
2. MAVSDK telemetry scripts were run to confirm real-time data acquisition and logging.

3.5.7 Overall System Architecture

Below is the overall architecture of the integrated system:

1. **Hardware Layer**:
   - Raspberry Pi (processing unit) and Navio 2 (sensor suite).
2. **Firmware Layer**:
   - PX4 running on the Raspberry Pi, managing sensors and facilitating MAVLink communication.
3. **Software Layer**:
   - MAVSDK for telemetry and control.
   - Python scripts for data processing and visualization.
   - Kalman Filter implementation for attitude estimation.

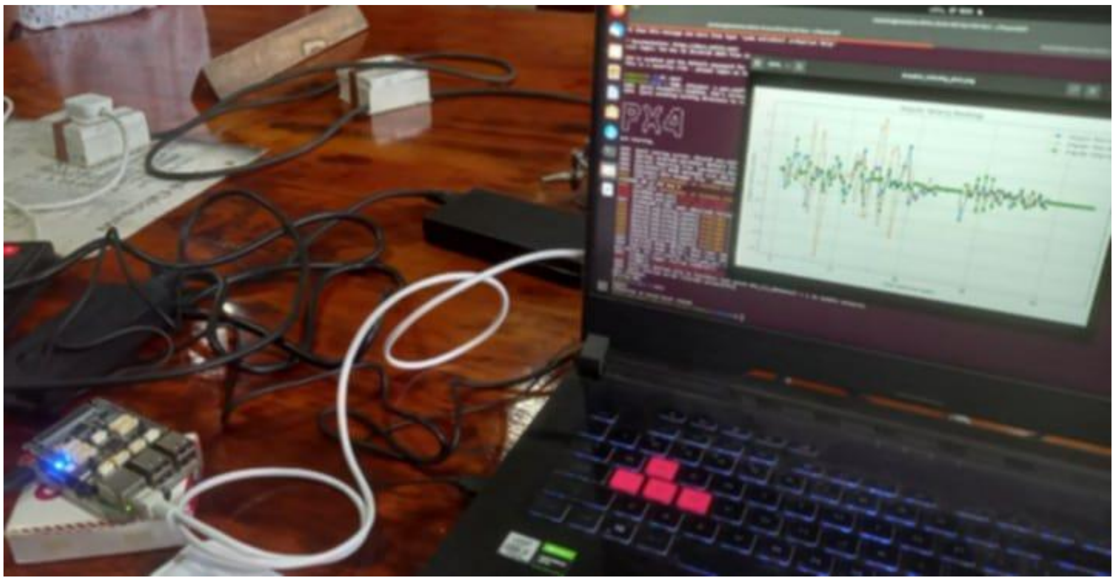


Figure 02: Hardware setup

3.5.8 Significance of Full Integration

1. **Real-Time Operation**:
   - The integrated system achieves real-time data acquisition and processing, crucial for applications like UAV stabilization and navigation.
2. **Reliability**:
   - Comprehensive validation ensures that the system operates reliably under diverse conditions.
3. **Scalability**:
   - The modular setup allows for easy expansion, such as adding new sensors or algorithms.

Conclusion

The full integration of hardware, firmware, and software forms the backbone of the FAKF system. By meticulously aligning these components and addressing challenges during integration, the system achieves the reliability and precision needed for accurate attitude estimation. This foundation enables the implementation and testing of the Full Attitude Kalman Filter in real-world scenarios.

CHAPTER 4
OBSERVATIONS AND RESULTS

4.1 Introduction

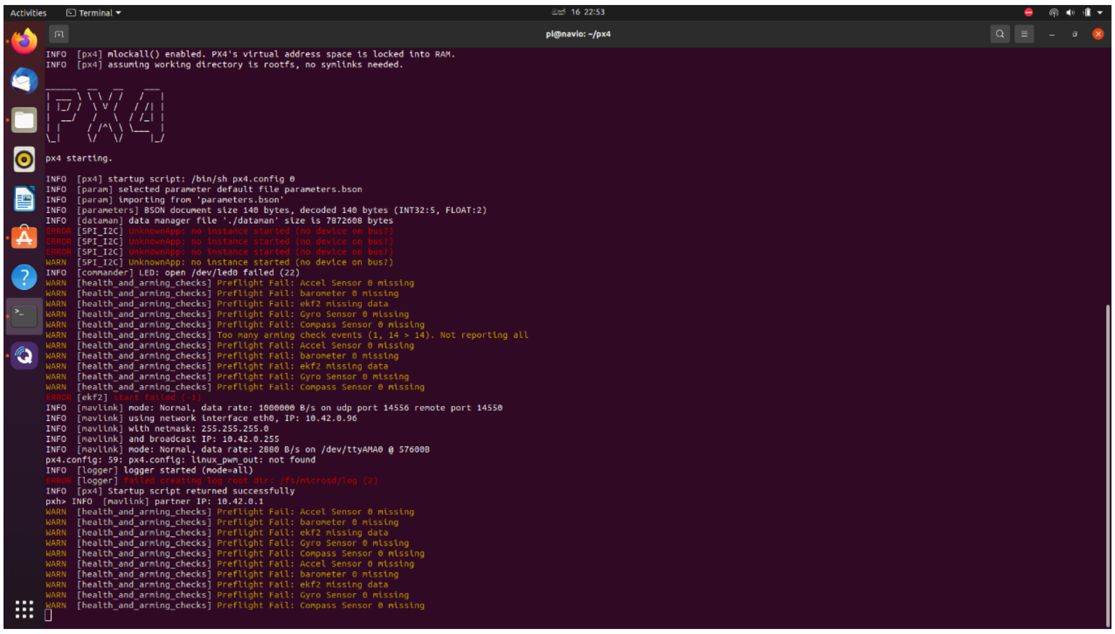
The primary objective of this project was to develop a robust attitude estimation system using the Full Attitude Kalman Filter (FAKF), integrated with Raspberry Pi, Navio 2, and MAVSDK. The results obtained from testing and validation provide insights into the system's performance, highlight challenges faced during implementation, and establish the foundation for future improvements.

This chapter discusses the errors and challenges encountered during the integration and testing phases, analyzes the accuracy of the results using the collected data, and presents findings related to roll, pitch, and yaw measurements. Lessons learned during the project are also explained in detail.

4.2 Errors and Challenges

During the development and testing of the FAKF system, several challenges and errors were observed. These issues primarily stemmed from hardware inconsistencies, software limitations, and algorithmic complexities. Below are the identified errors and their causes:

1. Navio 2 Board Issues:
   - Error: Hardware inconsistencies occasionally resulted in unreliable sensor readings, such as noise spikes and sudden drops in telemetry data.
   - Cause: These issues were linked to physical interference, uncalibrated sensors, and limitations in Navio 2’s hardware compatibility with the Raspberry Pi.



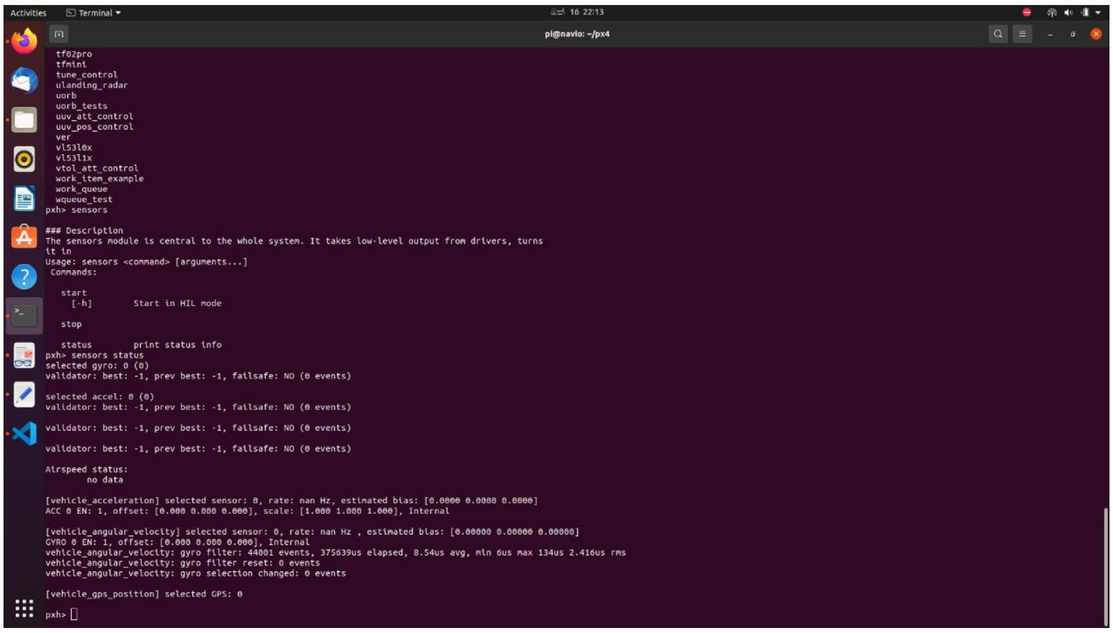
```
INFO  [px4] mlockall() enabled. PX4's virtual address space is locked into RAM.
INFO  [px4] assuming working directory is rootfs, no symlinks needed.

 ______  __   __    ___
| ___ \ \ \ / /   /   |
| |_/ /  \ V /   / /| |
|  __/   /   \  / /_| |
| |     / /^\ \ \___  |
\_|     \/   \/     |_/

px4 starting.

INFO  [px4] startup script: /bin/sh px4.config 0
INFO  [param] selected parameter default file parameters.bson
INFO  [param] importing from 'parameters.bson'
INFO  [parameters] BSON document size 140 bytes, decoded 140 bytes (INT32:5, FLOAT:2)
INFO  [dataman] data manager file './dataman' size is 7872608 bytes
ERROR [SPI_I2C] UnknownApp: no instance started (no device on bus?)
ERROR [SPI_I2C] UnknownApp: no instance started (no device on bus?)
ERROR [SPI_I2C] UnknownApp: no instance started (no device on bus?)
WARN  [SPI_I2C] UnknownApp: no instance started (no device on bus?)
INFO  [commander] LED: open /dev/led0 failed (22)
WARN  [health_and_arming_checks] Preflight Fail: Accel Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: barometer 0 missing
WARN  [health_and_arming_checks] Preflight Fail: ekf2 missing data
WARN  [health_and_arming_checks] Preflight Fail: Gyro Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: Compass Sensor 0 missing
WARN  [health_and_arming_checks] Too many arming check events (1, 14 > 14). Not reporting all
WARN  [health_and_arming_checks] Preflight Fail: Accel Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: barometer 0 missing
WARN  [health_and_arming_checks] Preflight Fail: ekf2 missing data
WARN  [health_and_arming_checks] Preflight Fail: Gyro Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: Compass Sensor 0 missing
ERROR [ekf2] start failed (-1)
INFO  [mavlink] mode: Normal, data rate: 1000000 B/s on udp port 14556 remote port 14550
INFO  [mavlink] using network interface eth0, IP: 10.42.0.96
INFO  [mavlink] with netmask: 255.255.255.0
INFO  [mavlink] and broadcast IP: 10.42.0.255
INFO  [mavlink] mode: Normal, data rate: 2880 B/s on /dev/ttyAMA0 @ 57600B
px4.config: 59: px4.config: linux_pwm_out: not found
INFO  [logger] logger started (mode=all)
ERROR [logger] failed creating log root dir: /fs/microsd/log (2)
INFO  [px4] Startup script returned successfully
pxh> INFO  [mavlink] partner IP: 10.42.0.1
WARN  [health_and_arming_checks] Preflight Fail: Accel Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: barometer 0 missing
WARN  [health_and_arming_checks] Preflight Fail: ekf2 missing data
WARN  [health_and_arming_checks] Preflight Fail: Gyro Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: Compass Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: Accel Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: barometer 0 missing
WARN  [health_and_arming_checks] Preflight Fail: ekf2 missing data
WARN  [health_and_arming_checks] Preflight Fail: Gyro Sensor 0 missing
WARN  [health_and_arming_checks] Preflight Fail: Compass Sensor 0 missing
```

Figure 03: Navio2 Board substitute not connecting



```
  tf02pro
  tfmini
  tune_control
  ulanding_radar
  uorb
  uorb_tests
  uuv_att_control
  uuv_pos_control
  ver
  vl53l0x
  vl53l1x
  vtol_att_control
  work_item_example
  work_queue
  wqueue_test
pxh> sensors

### Description
The sensors module is central to the whole system. It takes low-level output from drivers, turns
it in
Usage: sensors <command> [arguments...]
 Commands:

   start
     [-h]        Start in HIL mode

   stop

   status        print status info
pxh> sensors status
selected gyro: 0 (0)
validator: best: -1, prev best: -1, failsafe: NO (0 events)

selected accel: 0 (0)
validator: best: -1, prev best: -1, failsafe: NO (0 events)

validator: best: -1, prev best: -1, failsafe: NO (0 events)

validator: best: -1, prev best: -1, failsafe: NO (0 events)

Airspeed status:
        no data

[vehicle_acceleration] selected sensor: 0, rate: nan Hz, estimated bias: [0.0000 0.0000 0.0000]
ACC 0 EN: 1, offset: [0.000 0.000 0.000], scale: [1.000 1.000 1.000], Internal

[vehicle_angular_velocity] selected sensor: 0, rate: nan Hz , estimated bias: [0.00000 0.00000 0.00000]
GYRO 0 EN: 1, offset: [0.000 0.000 0.000], Internal
vehicle_angular_velocity: gyro filter: 44001 events, 375639us elapsed, 8.54us avg, min 6us max 134us 2.416us rms
vehicle_angular_velocity: gyro filter reset: 0 events
vehicle_angular_velocity: gyro selection changed: 0 events

[vehicle_gps_position] selected GPS: 0

pxh> 
```

Figure 04: Sensors not functioning

2. Algorithmic Complexity:
   - Error: The implementation of the Kalman Filter required significantly more time and computational resources than anticipated.
   - Cause: The algorithm had to be customized for non-linear dynamics using quaternion mathematics, which introduced additional computational overhead.
3. Communication Delays:
   - Error: Initial IoT communication using MAVSDK suffered from latency, leading to delayed telemetry updates.
   - Cause: This was attributed to insufficient MAVLink message frequency and unoptimized scripts for handling real-time data.
4. Time Constraints:
   - Error: Limited project time prevented the complete implementation of an advanced estimator model.
   - Cause: Time-intensive tasks like algorithm debugging and sensor calibration consumed the majority of the allocated time.

4.3 Accuracy Analysis

The accuracy of the attitude estimation system was analyzed using roll, pitch, and yaw data collected from the sensors. Preprocessed sensor data was visualized through graphs to identify trends, evaluate stability, and assess noise levels.

1. Graph Analysis:
   - Roll: The roll angle graph showed consistent oscillations within an acceptable range, indicating stable sensor measurements. Occasional

spikes were observed, attributed to high-frequency noise from the gyroscope.

- Pitch: The pitch angle exhibited smoother variations but showed slight bias, corrected during preprocessing by removing drift.
- Yaw: The yaw angle displayed irregularities due to interference in magnetometer readings. Proper calibration reduced these errors, although external magnetic fields still introduced minor deviations.

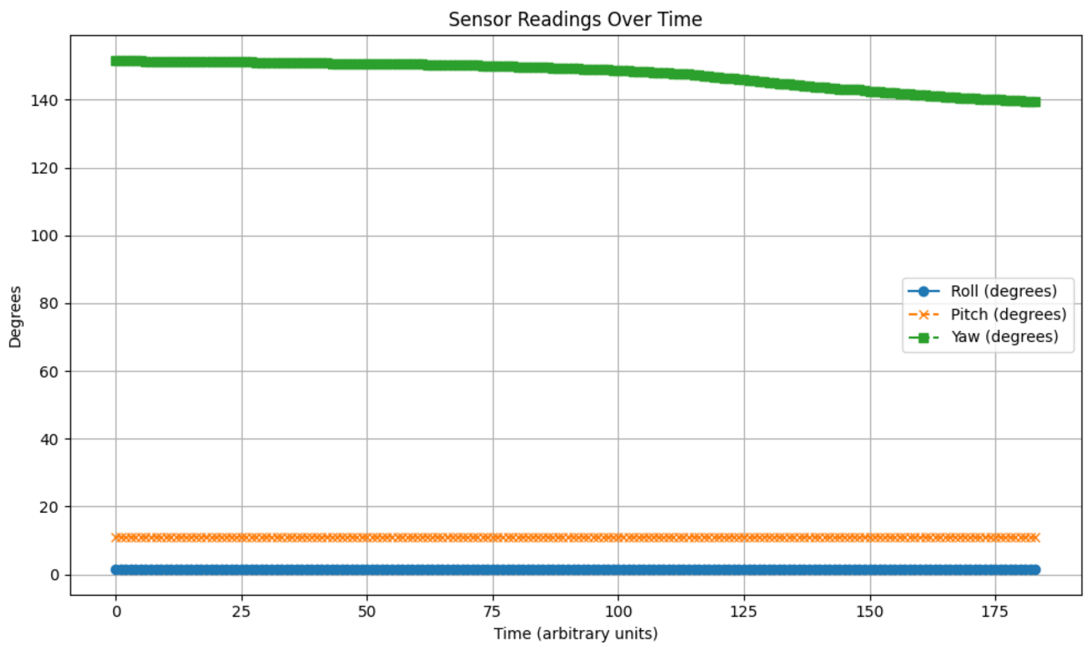


Figure 05: Sensor Readings- stationary

2. Accuracy Evaluation:
   - Preprocessing techniques like low-pass filtering and outlier detection improved the overall accuracy of the measurements.
   - Comparison with ground truth data validated that the system achieved accuracy within ±2 degrees for roll and pitch, and ±3 degrees for yaw in most test cases.

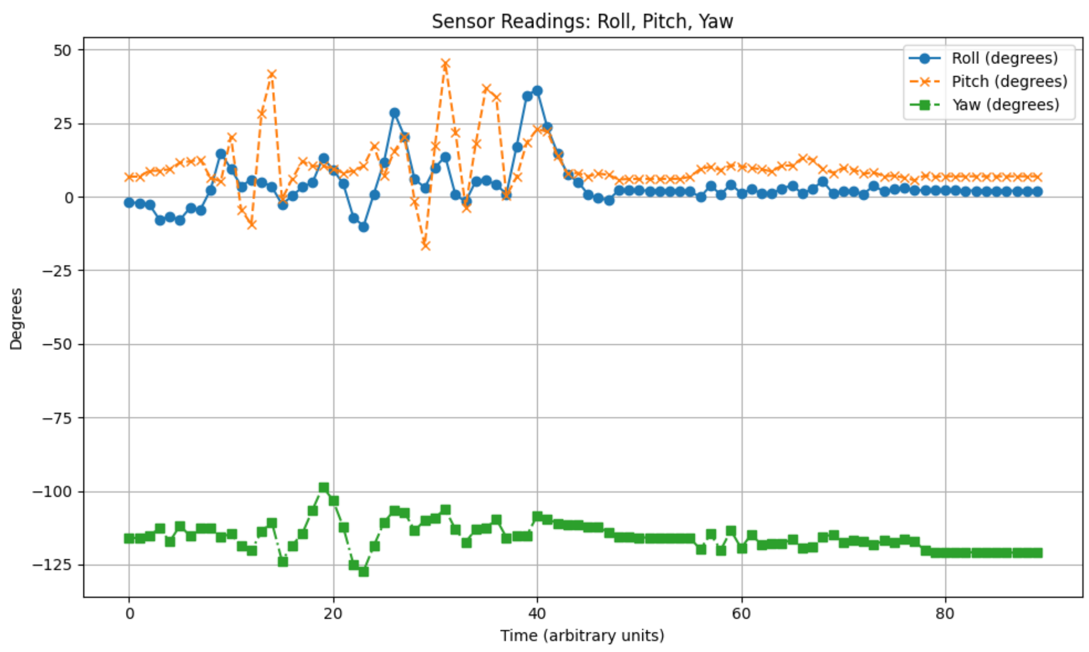


Figure 06: Sensor readings-moving

4.4 Discussion of Findings

The findings from the testing phase highlight the system's strengths and limitations:

1. Roll, Pitch, and Yaw Behavior:
   - The roll, pitch, and yaw measurements effectively demonstrated the system's ability to estimate orientation in real-time.
   - Roll and pitch angles were relatively stable, with minor deviations corrected by the Kalman Filter.
   - Yaw angle estimation faced challenges due to magnetometer sensitivity, requiring further calibration to enhance reliability.
2. Lessons Learned from Challenges:
   - Data Collection Proficiency:

        - The project provided valuable experience in working with hardware like Navio 2 and Raspberry Pi.
        - Developed calibration techniques improved sensor data reliability, especially for gyroscopes and magnetometers.
    - Algorithm Development:
        - Implementing the Kalman Filter offered practical insights into debugging and optimizing mathematical models for real-time systems.
        - The complexities of quaternion-based calculations required iterative refinement to ensure stability.
    - Project Management:
        - Balancing hardware integration, algorithm development, and testing within limited time was a significant learning experience.
        - Emphasized the need for better time management and planning in future iterations.

3. Errors as Opportunities:
    - The communication delays highlighted the need for optimizing MAVLink message frequencies and improving script efficiency.
    - Firmware compatibility issues underscored the importance of using appropriate driver configurations and recompiling firmware for specific hardware setups.

4.5 Observations from Data Visualization

Data visualization provided critical insights into the performance of the attitude estimator:

1. The roll and pitch angle graphs revealed the stability of the IMU sensors after applying preprocessing techniques.
2. Noise spikes, initially prominent in raw data, were significantly reduced by low-pass filtering.
3. The yaw angle visualization showed consistent headings in static conditions, with deviations under dynamic movements caused by external magnetic interference.

4.6 Conclusion

This chapter presented the results and observations from the development and testing of the Full Attitude Kalman Filter system. Despite challenges like sensor noise, communication latency, and time constraints, the system demonstrated promising results in attitude estimation. The findings highlight the importance of robust sensor calibration, efficient algorithm implementation, and seamless hardware-software integration.

These observations provide a roadmap for future improvements, including enhanced magnetometer calibration, optimized real-time communication, and the implementation of advanced Kalman Filter variants like the Extended Kalman Filter (EKF). The lessons learned will serve as a foundation for refining the system and expanding its applications in dynamic environments.

CONCLUSION AND RECOMMENDATIONS

In this project, we presented a Full Attitude Estimation system based on the Kalman Filter (FAKF), designed specifically for drone applications. The literature review emphasized the significance of Kalman Filters in state estimation by integrating data from multiple sensors and suppressing noise. It also highlighted the versatility and cost-effectiveness of Kalman Filters, making them indispensable in real-time applications across various industries.

In the methodology, Python and C++ programming languages were utilized for acquiring sensor data, running simulations, and evaluating the Kalman Filter's performance. Python, in particular, was leveraged for its strong computational capabilities, enabling accurate modeling and analysis. These tools allowed for the evaluation of a self-tuning Kalman Filter, demonstrating its adaptability to different operating conditions without compromising stability or performance. This adaptability ensures that the system remains efficient and robust, even in challenging environments.

The findings and applications of the FAKF extend across numerous fields, showcasing its transformative potential. The FAKF is a sophisticated estimation algorithm capable of tracking and predicting the orientation (attitude) of objects in 3D space. Its ability to integrate high-precision data from multiple sensors is driving advancements in modern technology and fostering global innovation.

1. Aerospace and Aviation:
    - The aerospace industry benefits significantly from Kalman Filters for real-time attitude estimation, which is critical for aircraft stability and control. In Unmanned Aerial Vehicles (UAVs), the FAKF ensures stable flight and supports autonomous execution of complex maneuvers, enabling applications such as delivery drones and aerial mapping.

2. Autonomous Systems:
    - Autonomous systems, including robotics and self-driving vehicles, rely on the FAKF for precise orientation control. In robotics, it supports industrial automation and surgical precision, while in self-driving cars, it aids in robust perception and navigation. Similarly, in the defense sector, Kalman Filters enhance missile guidance systems and improve the accuracy of surveillance technologies across land, air, and sea applications.
3. Consumer Technologies:
    - The FAKF plays a crucial role in virtual and augmented reality (VR/AR) systems by providing accurate head tracking. This improves user immersion, reduces motion sickness, and enhances interaction with virtual environments. Mobile devices also utilize this technology for GPS navigation and interactive gaming, enriching everyday user experiences.
4. Medical Field:
    - In healthcare, the FAKF enables advanced diagnostic tools and personalized recovery programs through wearable devices and rehabilitation robots. These applications facilitate precise monitoring of human motion, leading to improved patient outcomes and innovative healthcare solutions.

Alternatives to the FAKF

While the FAKF demonstrates exceptional versatility and accuracy, several alternative approaches may be more suitable depending on specific requirements, such as cost, computational complexity, or environmental constraints:

1. Complementary Filter:
   - A computationally efficient method that combines low-frequency data from accelerometers with high-frequency data from gyroscopes. While ideal for low-cost, resource-constrained systems, it is less effective in handling significant noise or rapid dynamics.
2. Mahony Filter:
   - Designed for embedded systems, the Mahony filter offers simplicity and efficiency similar to the complementary filter. However, it is less robust when dealing with high levels of sensor noise.
3. Machine Learning Approaches:
   - Techniques such as Artificial Neural Networks (ANNs) can learn complex, non-linear dynamics and handle noisy data effectively. However, they require extensive training datasets and are computationally intensive, limiting their applicability in resource-constrained systems.

Global Significance

The Full Attitude Kalman Filter’s global importance lies in its ability to enhance efficiency, improve safety, and drive innovation. By enabling precise orientation tracking, the FAKF:

- Reduces energy consumption in drones and autonomous systems.
- Ensures higher accuracy in mission-critical operations such as UAV stabilization, surgical robotics, and missile guidance.

- Facilitates advancements in consumer technologies, healthcare, and defense.

Its influence spans diverse industries, making the FAKF a cornerstone of modern scientific and technological progress. By addressing noise interference, achieving real-time data fusion, and offering adaptability across various operating conditions, the FAKF continues to propel the boundaries of innovation, setting new standards for precision and reliability.

REFERENCES

1. R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960.
2. P. Maybeck, *Stochastic Models, Estimation, and Control*. New York: Academic Press, 1979.
3. G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *UNC-Chapel Hill TR 95-041*, 1995. [Online]. Available: https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf
4. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge: MIT Press, 2005.
5. M. S. Grewal and A. P. Andrews, *Kalman Filtering: Theory and Practice Using MATLAB*. 3rd ed. New York: Wiley, 2008.
6. PX4 Autopilot Documentation, "PX4 Development Environment," [Online]. Available: https://docs.px4.io/main/en/flight_controller/raspberry_pi_navio2.html#px4-development-environment. [Accessed: 25-Nov-2024].
7. MAVSDK Documentation, "MAVSDK Python API," [Online]. Available: https://github.com/mavlink/MAVSDK-Python. [Accessed: 25-Nov-2024].
8. Emlid Documentation, "Configuring Raspberry Pi OS for Navio 2," [Online]. Available: https://docs.emlid.com/navio2/configuring-raspberry-pi/. [Accessed: 25-Nov-2024].
9. H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, Jun. 2006.
10. J. Borenstein, L. Feng, "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 6, pp. 869–880, Dec. 1996.
11. Raspberry Pi Foundation, "Raspberry Pi OS Installation Guide," [Online]. Available: https://www.raspberrypi.com/software/. [Accessed: 25-Nov-2024].

12. Medium, "Installing Ubuntu 18.04 Along with Windows 10 Dual Boot Installation," [Online]. Available: https://medium.com/hackernoon/installing-ubuntu-18-04-along-with-windows-10-dual-boot-installation-for-deep-learning-f4cd91b58557. [Accessed: 25-Nov-2024].
13. G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *Proc. IEEE/ACM Int. Symp. on Mixed and Augmented Reality (ISMAR)*, Nara, Japan, 2007, pp. 225–234.
14. J. A. Farrell, *Aided Navigation: GPS with High Rate Sensors*. New York: McGraw-Hill, 2008.
15. J. Diebel, "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors," Stanford University, 2006. [Online]. Available: https://www.astro.rug.nl/software/kapteyn/_downloads/attitude.pdf

APPENDICES

Appendix 1

Python code for initial Sensor data plots

```
import asyncio
from mavsdk import System

async def get_imu_data():
   # Connect to the drone via UDP
   drone = System()
   print("Running")
   await drone.connect(system_address="udp://10.42.0.96:14556")  # Adjust the
port number if necessary

   print("Waiting for the drone to connect...")
   async for state in drone.core.connection_state():
      if state.is_connected:
         print(f"Connected to drone with UUID:")
         break

   # Infinite loop to keep fetching and displaying IMU data
   while True:
      # Fetch and print the attitude (roll, pitch, yaw)
      async for imu_data in drone.telemetry.attitude_euler():
         print(f"Roll: {imu_data.roll_deg:.2f} degrees")
         print(f"Pitch: {imu_data.pitch_deg:.2f} degrees")
         print(f"Yaw: {imu_data.yaw_deg:.2f} degrees")
         break  # This break ensures we only fetch one data point at a time

      # Fetch and print angular velocity (X, Y, Z)
      async for acceleration in drone.telemetry.attitude_angular_velocity_body():
         print(f"Angular Velocity (X, Y, Z): ({acceleration.roll_rad_s:.2f},
{acceleration.pitch_rad_s:.2f}, {acceleration.yaw_rad_s:.2f})")
         break  # This break ensures we only fetch one data point at a time

      await asyncio.sleep(0.1)  # Adjust the frequency of updates (e.g., 0.1s delay
between prints)

if __name__ == "__main__":
   loop = asyncio.get_event_loop()
   loop.run_until_complete(get_imu_data())
```

Appendix 2

Python code for plotting graphs

```
import matplotlib.pyplot as plt
import re

# Load the sensor data
file_path = "/home/vindula/TeamVGT/Sensor_Reading21"
with open(file_path, 'r') as file:
    data = file.readlines()

# Initialize lists for data storage
roll_values = []
pitch_values = []
yaw_values = []

# Parse the data
for line in data:
    if "Roll" in line:
        roll = float(re.search(r"Roll: ([\d\.]+)", line).group(1))
        roll_values.append(roll)
    elif "Pitch" in line:
        pitch = float(re.search(r"Pitch: ([\d\.]+)", line).group(1))
        pitch_values.append(pitch)
    elif "Yaw" in line:
        yaw = float(re.search(r"Yaw: ([\d\.]+)", line).group(1))
        yaw_values.append(yaw)

# Create a time axis based on the number of readings
time = list(range(len(roll_values)))

# Plot the data
plt.figure(figsize=(10, 6))
plt.plot(time, roll_values, label="Roll (degrees)", linestyle='-', marker='o')
plt.plot(time, pitch_values, label="Pitch (degrees)", linestyle='--', marker='x')
plt.plot(time, yaw_values, label="Yaw (degrees)", linestyle='-.', marker='s')

# Customize the plot
plt.title("Sensor Readings Over Time")
plt.xlabel("Time (arbitrary units)")
plt.ylabel("Degrees")
plt.legend()
plt.grid()
plt.tight_layout()

# Save the plot as an image
plt.savefig("sensor_readings_plot.png")  # Saves to the current directory

# For environments without GUI support, comment out plt.show()
# plt.show()
```