ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Variable Pitch System for UAV proprotors

Carlos André Pinto Ramos Gonçalves Rijo

Thesis Research Plan of the Masterś degree in Critical Computing Systems Engineering

Supervisor: Ricardo Augusto Rodrigues Da Silva Severino
Co-Supervisor: José Renato Santos Machado

Porto, January 26, 2024

Abstract

TODO - abstract up to 200 words

Keywords: Keyword1, ..., Keyword6

Resumo

TODO - abstract up to 1000 words ???

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

FC	**F**light **C**ontroller
FPP	**F**ixed **P**itch **P**roprotor
MDU	**M**ain **D**evice **U**nit
SDU	**S**econdary **D**evice **U**nit
VPP	**V**ariable **P**itch **P**roprotor

List of Acronyms

COTS	Commercial Off-The Shelf.
CPU	Central Processing Units.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
eMMC	embedded MultiMedia Card.
FPGA	Field-Programmable Gate Arrays.
FRAM	Ferroelectric Random Access Memory.
GNSS	Global Navigation Satellite System.
HDD	Hard Disk Drives.
Li-ion	Lithium-ion.
Li-Po	Lithium Polymer.
MLC	Multi-Level Cell.
NiCd	Nickel-Cadmium.
NiMH	NickelMetal Hydride.
OBC	On Board Computer.
PCB	Pinted Circuit Board.
PWM	Pulse Width Modulation.
RAM	Random Access Memory.
ROS	Robot Operating System.
RPM	Revolutions Per Minute.
RTC	Real-Time Clock.
RTOS	Real-Time Operating System.
SD	Secure Digital.
SLC	Single-Level Cell.
SoC	State of Charge.
SSD	Solid State Drives.
TLC	Triple-Level Cell.
UAV	Unmanned Aerial Vehicle.

UVP	Under Voltage Protection.
VTOL	Vertical Take-Off and Landing.

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) have witnessed a surge in popularity and research attention. They have become indispensable in various applications, ranging from surveillance to reconnaissance, due to their versatility and efficiency in various applications [1].

This growing interest is evident in the numerous review papers exploring different aspects of UAV development, ranging from open-source hardware and software utilization [2], [3],[4], frame design and optimization [5], [6], control systems, including both conventional and modern communication modalities such as 5G networks [7], [8], [**UAV10**], [9], to efficient power management strategies, and alternative energy sources to extend UAV battery life [10].

Nowadays, most Vertical Take-Off and Landing (VTOL) UAVs, rely on proprotors with fixed pitch systems because of their simplicity and lack of better Commercial Off-The Shelf (COTS) reliable and efficient solutions but they impose limitations on the achievable flight performance [11]. Thrust generation is confined to a single direction, hindering the UAV's ability to produce upward thrust relative to the vehicle body. Additionally, the control bandwidth is restricted by the inertia of the motors and proprotors, constraining the UAV's agility and maneuverability [11].
As described in recent studies [11], these limitations become more pronounced as UAV size increases, impacting stability and control. Larger UAVs face challenges as the need for larger motors with higher inertia compromises rapid control through Revolutions Per Minute (RPM) adjustments alone.

The development of Variable Pitch Proprotor (VPP) systems plays a crucial role in overcoming the limitations of traditional UAV designs, such as those with fixed pitch proprotors [12], [11]. Several detailed descriptions of quadrotors, for instance, modeling and dynamics have been published [**FPP2**], [**FPP3**], [**FPP4**], [**FPP5**], emphasizing the need for dynamic control mechanisms.
The use of VPP systems, especially in VTOL UAVs, addresses challenges related to control instabilities and energy efficiency [12].
The incorporation of variable-pitch proprotors provides the necessary flexibility to enhance stability and enable larger UAVs to perform sophisticated maneuvers, overcoming the constraints inherent in fixed-pitch designs [12], [11].

TODO: MORE INFO

1.1 Problem Analyses

The utilization of fixed-pitch propellers in UAVs presents a set of limitations that significantly impact aircraft performance and efficiency. These issues are evident in various phases of flight, including hover and forward flight, and have repercussions for the UAV.

TODO: MORE INFO

1.1.1 Maneuverability and Response

Fixed-pitch propellers inherently constrain UAVs from adjusting the pitch angle during flight [11]. This limitation results in compromised maneuverability and response, restricting the range of aerobatic maneuvers that a UAV can execute [11]. Additionally, the inability to change the pitch angle impedes the optimization of lift, landing, and thrust during flight, leading to sub-optimal performance in various operational scenarios [11].

TODO: MORE INFO

1.1.2 Power Consumption

With fixed-pitch propellers, the power consumption will be higher. Without the ability to adjust the propeller angle, UAVs may be forced to operate at higher RPMs to compensate for this lack of adjustment [11]. This higher power consumption not only affects the efficiency of the UAV but also has implications for its endurance, limiting the time the vehicle can remain airborne.

TODO: MORE INFO

1.2 Motivation

TODO: MORE INFO

1.3 Objectives

TODO: MORE INFO

This thesis will focus on developing a stand-alone variable-pitch proprotor system that can, in real-time, change the propeller pitch according to each flight phase. The student will develop and implement the control system electronics and the underlying real-time wireless sensing and actuation system. Objectives include:

1. Understand the relevant fundamentals regarding VTOL flight performance and the impact of proprotors. Understand the fundamentals of short-range wireless communication protocols with particular emphasis on their reliability. Understand the fundamentals of electronic control systems with a focus on real-time sensing and actuation.

2. Create and present a consistent argument regarding the motivation for variable-pitch propeller systems and current challenges.

3. Survey current solutions, communication technologies, and electronic control strategies to address the problem. Compare the different approaches.

4. Design a systems architecture to carry out the real-time control of the variable-pitch system.

5. Implement the envisaged control system over a real mechanical prototype.

6. Evaluate the performance of the system and its limitations under different settings.

7. Write and publish a research paper featuring part of the results.

Chapter 2

State of the Art

2.1 UAV

TODO: MORE INFO TODO: ADD FIGURES

2.1.1 VTOL

TODO: MORE INFO TODO: ADD FIGURES

2.1.2 Fixed Pitch Proprotors

TODO: MORE INFO TODO: ADD FIGURES

2.1.3 Variable Pitch Proprotors

Historically, early aviation pioneers experimented with propellers that could only be adjusted on the ground. The first automatic variable pitch air screw was patented by L. E. Baines in 1919. The Gloster Hele-Shaw Beacham variable pitch propeller, developed in 1928, demonstrated practical controllable pitch capabilities. Over time, various designs and mechanisms, including hydraulic and pneumatic systems, were explored and refined. The development of constant-speed propellers marked a significant advancement in aviation technology, offering improved efficiency and performance [13].

A significant advantage of variable-pitch propellers is their ability to adapt to varying airspeeds. When an aircraft is stationary or moving slowly, the propeller blades can be set to a low angle of attack to reduce drag. As the aircraft gains speed, the pitch is increased to maintain optimal performance. This adaptability ensures efficient operation across a range of flight conditions.

The primary purpose of variable pitch propellers is to maintain the optimal angle of attack relative to the changing wind vector as the aircraft accelerates. Traditional fixed-pitch propellers face efficiency challenges in various flight conditions. Adjustable blade angles address this issue, allowing for improved efficiency during takeoff, climb, and cruise.[14].

Variable-pitch systems can adjust blade pitch to maintain a selected RPM enhancing overall performance, especially at high altitudes, by allowing the rotor to operate in its most economical speed range [13], [14].

Three methods change the pitch: Hydraulic, Centrifugal, and Electromechanical control [13].
TODO: ADD FIGURES

Hydraulic Method

This system involves the use of engine oil pressure to control the pitch-changing mechanism and consists of a pump, control valves, and cylinders that actuate the movement of the propeller blades. In an aircraft without a variable-pitch proprotor system, the pilot uses hydraulics to manually control the pitch of the propeller blades [13].

Hydraulic systems provide a precise means of adjusting the propeller pitch, allowing efficient performance under different flight conditions, and contributing to the overall safety and reliability of the system.

But Hydraulic systems add complexity and weight to the overall aircraft system. More components means more elements could potentially fail or require maintenance. There is also the risk of fluid leakage or fluid contamination that may lead to a reduction in hydraulic pressure, potentially affecting the pitch control mechanism. Hydraulic systems may have a slow response time due to the time it takes for hydraulic pressure changes to propagate through the system which might be a concern in situations where rapid adjustments are required.[13] TODO: ADD FIGURES

Centrifugal Method

In the centrifugal systems, centrifugal weights can be attached directly to the propellers. An eccentric weight is placed near or in the spinner and secured with a spring and, when the propeller reaches a certain RPM, centrifugal force swings the weights outward, driving a mechanism that twists the propeller to a steeper pitch. As the propeller slows down, the RPM drops and the spring pushes the weight back, readjusting the propeller pitch to a shallower pitch.

As advantages, centrifugal systems are simpler compared to hydraulic systems since they involve fewer components. The reliance on mechanical components driven by centrifugal force can enhance reliability because there are fewer points of failure. There is no need to use external power sources, such as an engine-driven pump. Also, centrifugal systems can operate automatically without direct pilot intervention. The system responds to changes in rotational speed without the need for continuous manual control.

However, centrifugal systems may provide less precise pitch control than more advanced hydraulic or electronic systems. This limitation can affect the ability to finely tune the propeller for optimal performance. The response time of centrifugal systems may be slower compared to more sophisticated systems. This limitation could be a factor in situations where rapid adjustments to the propeller pitch are necessary.[13]

TODO: ADD FIGURES

Electromechanical Method

These systems involve electric motors and mechanical linkages to control the pitch of the propeller blades.

Electromechanical methods provide precise control over the pitch of the propeller blades, can offer rapid response times to changes in flight conditions, are often versatile, and can be adapted for various aircraft configurations. Compared to certain hydraulic systems, electromechanical systems might require less maintenance. They often have fewer components prone to wear and can be more straightforward to service.

As disadvantages, electromechanical systems, including motors and associated components, can add weight to the aircraft, require electrical power to operate, and are more complex than purely mechanical systems, increasing the chance of failures.[13]

TODO: ADD FIGURES *VPP Video*

2.2 Control System

An On Board Computer (OBC) is a device capable of managing and/or controlling various functions such as:

- It can manage overall system operation.
- Implement safety mechanisms and respond to abnormal conditions.
- Execute algorithms and computations required for the system's functionality.
- Interface with external devices, sensors, actuators, or other embedded systems.
- Implement communication protocols for data exchange.
- Manage data storage and retrieval.
- Implement power-saving modes when appropriate.
- Manage and control peripherals such as communication interfaces, timers, and interrupt controllers.

There are, mainly, three types of control units: Microcontrollers, Microprocessors, and Field-Programmable Gate Arrayss (FPGAs).

Microcontrollers are integrated circuits that contain a processor core, memory, and programmable input/output peripherals. They are compact and cost-effective, have low power consumption, are designed for specific tasks, making them suitable for embedded systems, and often include integrated peripherals like timers, communication interfaces, and ADC. However, microcontrollers have more limited processing power compared to microprocessors and are less flexible for general-purpose computing.

Microprocessors or Central Processing Unitss (CPUs) focus on processing tasks and rely on external components for additional functionalities. As an advantage, they have high processing power (suitable for general-purpose computing), can run complex operating systems, and have greater flexibility in application design. However, microprocessors have higher power

consumption, may require additional components for specific applications, and have a larger form factor compared to microcontrollers.

FPGAs are integrated circuits that can be configured after manufacturing, allowing for custom digital logic circuits. They are customizable for specific applications, have parallel processing capabilities, and can be reprogrammed for different tasks.
But, they have a higher cost (compared to microcontrollers and microprocessors), have higher power consumption (compared to microcontrollers), and have a steeper learning curve for programming and design.

TODO: MORE INFO

2.3 Servo Motors

TODO: MORE INFO
TODO: ADD FIGURES

Chapter 3

Technologies

3.1 Battery Module

Batteries are a critical component in portable embedded systems, providing the necessary energy for the system to work properly.

This way, the battery technology selection must be carefully made to optimize the system performance, [15]. Different battery chemistries offer varying energy densities, voltages, sizes, weights, cycle lives and costs.

In table 3.1 it is possible to compare the most common battery technologies.

Technology	Energy Density	Voltage	Size/Weight	Cycle Life	Cost
Li-ion	High	3.7V	Compact/Light	Good	Moderate
Li-Po	High	3.7V	Flexible/Light	Good	Moderate
NiMH	Moderate	1.2V	Bulky/Heavy	Moderate	Moderate
NiCd	Moderate	1.2V	Bulky/Heavy	Good	Moderate
Lead-Acid	Low	2V (6V, 12V)	Bulky/Heavy	Moderate	Low
Alkaline	Moderate	1.5V	Standard Cylindrical	Poor	Moderate

Table 3.1: Comparison of Battery Technologies

Lithium-ion (Li-ion) and Lithium Polymer (Li-Po) batteries, shown in figures 3.1 and 3.2, can offer high energy density, rechargeability and moderate costs that make them suitable for portable devices, [16]. These batteries stand out as a prevalent choice for embedded systems.

Figure 3.1: Li-ion battery example [17]

Figure 3.2: Li-Po battery example [18]

NickelMetal Hydride (NiMH) and Nickel-Cadmium (NiCd) batteries (figures **??** and **??** respectably) can provide moderate energy density, rechargeability and moderate costs, but they can be heavier and NiCd batteries have a *memory effect* concern (where the battery, falsely, indicates full charge despite being only partially charged).

Figure 3.3: NiMH battery example [19]

Figure 3.4: NiCd battery example [20]

Lead-Acid batteries can be rechargeable and cost-effective but heavier and larger and with low energy density. These batteries are suitable for less portable applications, [15] as it is possible to see in figure 3.5.

Figure 3.5: Lead-Acid battery example [21]

Alkaline batteries in figure 3.6 are cost-effective but most of them are non-rechargeable, have a standard cylindrical format and have moderate energy density [15].

Figure 3.6: Alkaline battery example [22]

3.2 External Memory and Storage Units

Memory units can be classified as volatile memory and as non-volatile memory [23].

Volatile storage, like for example Random Access Memory (RAM) provides fast read and write speeds and is used for storing variables and managing application stacks. However, they require constant power to retain data, making them unsuitable for applications with strict power constraints, and have lower memory capacity [24].

Non-volatile storages are suitable for applications requiring frequent data read and write operations and have the capability of being electrically erased and reprogrammed. They have long-term data retention and low power consumption but have slow access speed compared to volatile storage [24]. These characteristics make non-volatile storages useful for storing configuration parameters and critical data that need to be retained during power cycles.

Often, in embedded systems, the system must be able to store data not only internally (main memory) but also externally (external memory). External memory units are normally used to expand storage capacity, store data and logs, facilitate data transfer, and backup critical information [23].

Since non-volatile storage can keep the data stored even when they are not powered, these storages are very common in embedded systems [25].

Secure Digital (SD) card, with its multiple formats and sizes, has moderate access speed and can keep data for a long term but it depends on the type (Single-Level Cell (SLC), Multi-Level Cell (MLC) and Triple-Level Cell (TLC)). Typically, the capacity ranges from a few megabytes to multiple terabytes, offering multiple choices for different use cases [26], [27]. However, the moderate power consumption and overall cost can, sometimes, be a setback to the system.

Electrically Erasable Programmable Read-Only Memory (EEPROM), commonly used for storing small amounts of data, has fast access speed and a moderate overall cost. Has long-term data retention and low power consumption but offers lower capacities (in the range of kilobytes to megabytes) making it only suitable for small data storage [24], [26].

Ferroelectric Random Access Memory (FRAM) combines the benefits of RAM and EEPROM and can be suitable for applications requiring fast and non-volatile memory. Has very fast access speed, long-term data retention, low capacity (in the range of kilobytes to megabytes), and very low power consumption. But has a relatively higher overall cost compared to other technologies [24], [26].

As for embedded MultiMedia Card (eMMC), normally found in smartphones, tablets, and other embedded systems, is characterized by its fast access speed, long-term data retention and high capacity (with ranges from megabytes to terabytes). Like SD cards it has moderate power consumption and moderate to high overall cost [26].

Hard Disk Drives (HDD) and Solid State Drives (SSD) memory units can have fast access speed, long-term data retention and high capacity (in the range of gigabytes to terabytes). But, in comparison to other memory units, it has a bigger size, higher power consumption and higher overall cost. These units are normally used as primary storage in computers and laptops for improved performance, [28].

Table 3.2 compares the described technologies in their access speed, overall cost, data retention, capacity and power consumption.

	Access Speed	Overall Cost	Data Retention	Capacity	Power Consumption
SD Cards	Moderate	Moderate	Long-term	High	Moderate
EEPROM	Fast	Moderate	Long-term	Low	Low
FRAM	Very Fast	Relatively Higher	Long-term	Low	Very Low
eMMC	Fast	Moderate	Long-term	High	Moderate
SSD	Very Fast	High	Long-term	Very High	High
HDD	Moderate	High	Long-term	very High	High

Table 3.2: Comparison of External Memory Technologies

3.3 Firmware

In embedded systems, the choice of programming languages and the use of a Real-Time Operating System (RTOS) in firmware development are critical decisions that can impact the performance, efficiency, and complexity of the embedded system.

3.3.1 Programming languages

Even though there are multiple programming languages, normally categorized as low-level or high-level, not all programming languages are optimized to be used in embedded systems. In this section, an overview and comparison were made on some examples of programming languages [29], [30].

C

C programming language has low-level features and is close to the hardware making it efficient and with high performance [31], [30].

It provides fine-grained control over memory, leading to efficient memory usage but can also lead to potential errors if not handled carefully [32]. Due to its low-level control and predictable performance, it is often used in real-time systems [33].

This programming language is generally portable and has a large and active community, with extensive support and numerous libraries, development tools and compilers available [33].

In terms of safety and reliability, it is a powerful language but lacks some safety features, like in memory management for example [33].

C can have a steeper learning curve, especially for beginners, due to manual memory management and low-level constructs [31].

Assembly

Assembly programming language provides direct control over hardware and is highly efficient, and useful for writing low-level code. It allows developers to directly manage memory, providing fine control over memory footprint [33].

Assembly can be used in real-time systems due to its precise control over hardware, predictable performance and high efficiency [32].

However, in *Assembly*, there is no safety net or restrictions, this way developers must handle all aspects of safety and reliability manually [33]. *Assembly* is also highly dependent on the architecture and is not inherently portable, has a niche community, and support is often architecture-specific and relies on specific tools provided by the hardware manufacturer. The learning curve is steep due to its low-level nature, and development is time-consuming compared to higher-level languages [32].

C++

C++ programming language is an object-oriented feature that can enhance code organization and reusability and can provide abstraction without sacrificing performance. It allows both manual and automatic memory management, providing flexibility [31], [30]. *C++* supports real-time programming, especially with the use of specific frameworks but is not as deterministic as low-level languages.

This programming language benefits from a robust community, with extensive support, it inherits development tools and compilers from *C* and has specific tools for features like object-oriented programming [33].

C++ introduces features like classes and objects, enhancing code organization and safety compared to C. However, it still allows low-level operations that may impact reliability [32].

Since *C++* is similar to *C*, and since it introduces additional concepts, the learning curve can be slightly more complex. However, its object-oriented features can lead to more maintainable code [33].

Rust

Rust programming language, known for its focus on memory safety, is gaining popularity in embedded systems development. It offers performance similar to *C* and *C++* while providing memory safety features [31],[33] .

Rust's was designed with a strong focus on memory safety with an ownership system that helps prevent common memory-related errors without sacrificing performance. This results in a secure and efficient memory footprint [31].

As for portability, it aims to be highly portable, with a focus on minimizing platform-specific issues. With features like Cargo, it simplifies dependency management and project setup.

Rust has a growing community and is gaining popularity, with strong support. However, it can be challenging for beginners due to its ownership system [33].

MicroPython

MicroPython is a compact extension of Python designed for microcontrollers and IoT devices to emphasize efficiency. However, it sacrifices some features of the standard Python to fit within resource constraints [31].

It aims for a small memory footprint suitable for microcontrollers which enhances portability as well [33].

MicroPython can be used for real-time tasks on microcontrollers, but its capabilities may be limited since it is not as deterministic as low-level languages. The community focused on

supporting embedded systems for *MicroPython* is growing with resources specifically tailored to microcontroller development [31], [32].

Table 3.3 resumes and compares all programming languages described above.

Topic	C	C++	Rust	Assembly	MicroPython
Efficiency and Performance	High	High	High	Very High	Moderate
Memory Footprint	Low	Moderate	Moderate	Very Low	Low
Real-Time Capabilities	Limited	Limited	Developing	Yes	Limited
Portability	High	Moderate	Moderate	Low	High
Community and Support	Large	Large	Growing	Limited	Growing
Development Tools	Abundant	Abundant	Growing	Limited	Limited
Safety and Reliability	Moderate	Moderate	High	Low	Moderate
Learning and Development	Moderate	Moderate	Moderate	Difficult	Easy

Table 3.3: Comparison of Programming Languages in Embedded Systems

3.3.2 Real-time Operating Systems

RTOSs facilitates multitasking, allowing concurrent execution of multiple tasks, can provide task scheduling, priority management, and inter-process communication and is suitable for systems with real-time requirements. But this can add overhead, especially in terms of memory footprint, and the learning curve is steeper [34].
The following RTOS examples are open-source, well-documented, compact, and designed for resource-constrained systems, and they support various microcontroller architectures [35]. They also support microROS, an extension of the Robot Operating System (ROS), designed for microcontrollers and embedded systems that can be resource-constrained.

FreeRTOS

FreeRTOS is a popular open-source (with MIT license) real-time operating system.

It has a small footprint, making it suitable for resource-constrained embedded systems. Has a large and active community, which can be beneficial for support and finding solutions [36].

As for scheduling policies, it has priority-based, round-robin and rate monotonic schedulers and has semaphore/mutex management [37]. It supports I2C, SPI and UART wired protocols and BLE-Stack, TLS, Ethernet and Wifi network protocols [37].

ChibiOS

ChibiOS is a real-time operating system designed for embedded systems with GPL3/commercial license. It is designed to be modular, allowing the user to include only the necessary components. This way it can be used for both small and large systems [38].

Offers a rich set of features, including support for various architectures and a hardware abstraction layer.

NuttX

[39] [37]

NuttX is a real-time operating system with a focus on standards compliance (POSIX and ANSI). It uses the Apache 2.0 license, allowing for both open-source and commercial use [39].

NuttX can be scalable, providing a balance between small footprint for resource-constrained devices and support for larger systems.

In terms of scheduling Priority-based FIFO Round-Robin Sporadic Server Semaphore /Mutex management I2C SPI USB CAN Modbus

6LoWPAN Ethernet Wifi RFID

Zephyr

[35] [40] [37]

Zephyr is a real-time operating system for resource-constrained devices. It supports micro-ROS, allowing integration with ROS-based robotic systems. Micro XRCE-DDS is one of the DDS implementations that can be used with Zephyr.

Licensing: Released under the Apache 2.0 license, allowing for open-source and commercial use. Footprint: Designed to be scalable and can be configured to match the requirements of the target device. Community: Has a growing and active community. Supported by the Linux Foundation.

Critical Systems Perspective:

All of these RTOSs can be used in critical systems, but the choice might depend on specific requirements such as certification standards, safety features, and community support. Certification: Consider the certification standards required for critical systems, like DO-178C for avionics or ISO 26262 for automotive.

https://micro.ros.org/docs/concepts/rtos/comparison/

Table 3.4 [37].

Table 3.4: Comparison of Real-Time Operating Systems

Feature	FreeRTOS	ChibiOS	NuttX	Zephyr
Licensing	MIT	Mixed GPL3/Commercial	Apache 2.0	Apache 2.0
Memory Footprint	Small	Small to Large	Scalable	Scalable
Community and Support	Large	Active	Growing	Active
Architecture Support	Wide	Wide	Wide	Wide
Certification	Depends	Depends	Depends	Considered
POSIX/ANSI Compliance	Limited	Limited	Yes	Limited
Safety Features	Basic	Yes	Depends	Depends

3.3.3 Comparison

Programming Languages: Pros:

Portability: Higher-level languages like C or C++ provide better portability across different hardware platforms. Productivity: High-level languages can increase developer productivity by abstracting hardware details and providing more expressive syntax. Code Reusability: Modular code and libraries are more easily reusable, saving time and effort in development.

Cons:

Performance Overhead: High-level languages may introduce performance overhead compared to low-level languages like assembly or C, which can be critical in resource-constrained embedded systems. Memory Usage: Applications written in high-level languages might consume more memory compared to optimized low-level code. Limited Control: Developers may have less control over low-level hardware details, which could be essential for certain embedded applications.

Real-Time Operating Systems (RTOS): Pros:

Deterministic Timing: RTOS provides deterministic timing, crucial for systems where timing constraints are critical. Task Management: Efficient task scheduling and management enable the execution of multiple tasks simultaneously. Resource Management: RTOS can effectively manage resources, optimizing CPU and memory usage.

Cons:

Complexity: Integrating an RTOS can introduce complexity, especially for small-scale embedded systems where simplicity is essential. Overhead: Some RTOS overhead is incurred in terms of both processing time and memory usage. Learning Curve: Developers may need to invest time in learning and understanding the specific RTOS, adding to the development time.

3.4 Communication

TODO: MORE INFO

3.4.1 Wired

TODO: MORE INFO

3.4.2 Wireless

While researching wireless communication, multiple protocols can be studied. They can, mainly, be separated into two categories: short-range and long-range. In the context of UAVs, the focus will be on short-range wireless communication protocols [41], [42], [43].

Short-range protocols offer advantages such as lower power consumption, reduced interference, and efficient data transfer within confined spaces. Within this category, options like Bluetooth, Wi-Fi, Zigbee, Z-Wave, and LoRa for short distances emerge as noteworthy candidates. Each of these protocols addresses specific requirements, making them suitable for various aspects of UAV operations, from intra-component communication to data transfer between the UAV and ground control [42], [43].

- WiFi (802.11x): Can be used for high-speed data transfer over short ranges. It's suitable when you need to transmit large amounts of data between the UAV and a ground station.
  - Advantages

        * High Data Rates
        * Widespread Standard
        * Bi-Directional Communication
    - Disadvantages
        * High Power Consumption
        * Interference in 2.4 GHz and 5 GHz bands

- Bluetooth: Common short-range wireless technology with low power consumption. It's suitable for communication between components on a UAV.
- Advantages
    - Low Power Consumption
    - Ubiquity
- Disadvantages
    - Limited Range
    - Data Transfer Rates
- Zigbee: Low-power, low-data-rate wireless communication technology that is suitable for short-range communication in embedded systems.
- Advantages
    - Low Power Consumption
    - Mesh Networking
    - Low Latency
- Disadvantages
    - Limited Data Rate
    - Limited Range
- Z-Wave: Low-power wireless communication protocol often used in home automation. It's suitable for control and monitoring applications in UAVs.
- Advantages
    - Low Power Consumption
    - Interference Avoidance
- Disadvantages
    - Limited Data Rate
    - Less Common in Non-Home Automation Devices
- LoRa (Long Range): While designed for long-range communication, LoRa can also be used in short-range applications. It provides low-power, long-range communication suitable for certain UAV scenarios.
- Advantages

  - Long Range
  - Low Power Consumption
- Disadvantages
  - Low Data Rates
  - Unidirectional Communication

Chapter 4

Proposed Approach

4.1 Concept

Fixed-pitch proprotor (FPP) system limitations can be addressed by developing variable-pitch proprotors. Adjusting the pitch in both flight phases may be more complex and expensive but it offers more adaptability and a great positive impact on the overall propulsion system efficiency, increasing the endurance and range. The UAV will consume less in hover and the cruise speed will be much higher.

This way, the proposed solution for this problem is to develop a stand-alone variable-pitch proprotor system that can, in real-time, change the propeller pitch according to each flight phase.

4.2 Requirements

It is important to define requirements, for this system, to better understand the fixed-pitch propeller's limitations and to develop the necessary functionalities to achieve a variable-pitch proprotor system.
This way, the requirements should align with mechanical, control, communication, integration, and validation specifications.

- **Mechanical Requirements**

  (REQ_01): The system shall be designed to retrofit existing UAVs or integrate seamlessly into new UAV designs.

  (REQ_02): The variable-pitch mechanism shall be lightweight to minimize the impact on overall UAV weight and balance.

  (REQ_03): The system shall be able to withstand the operational stresses and environmental conditions encountered during UAV flights.

- **Control System Requirements**

  (REQ_04): The control system shall enable real-time adjustment of the proprotor pitch during different flight phases.

  (REQ_05): It shall incorporate failsafe mechanisms to respond to unexpected malfunctions or loss of communication and revert to a fixed-pitch state in case of critical failures.

(REQ_06): The system shall provide precise control over the pitch angle, allowing for fine adjustments to optimize performance.

- **Wireless Communication Requirements**

  (REQ_07): The wireless communication system shall be reliable, with minimal latency to ensure quick response times.

  (REQ_08): It shall operate within designated frequency bands and comply with relevant aviation communication standards.

  (REQ_09): Security measures shall be implemented to prevent unauthorized access or interference with the control signals.

- **Integration Requirements**

  (REQ_10): The system shall be designed for easy integration with common UAV autopilot systems.

  (REQ_11): It shall have compatibility with existing UAV avionics and navigation systems.

  (REQ_12): The variable-pitch system shall not interfere with other onboard sensors or communication systems.

- **Testing and Validation Requirements**

  (REQ_13): The system shall undergo rigorous testing under various operational scenarios, including weather conditions and flight profiles.

  (REQ_14): Validation shall include simulated and real-world flights to assess performance and reliability.

  (REQ_15): The system shall comply with relevant aviation regulations and standards.

4.3 System Architecture

As it is possible to see, in the proposed implementation of the System Architecture diagram (figure 4.1), the system will be composed of two subsystems: the Main Device Unit (MDU) and (multiple) Secondary Device Unit (SDU).

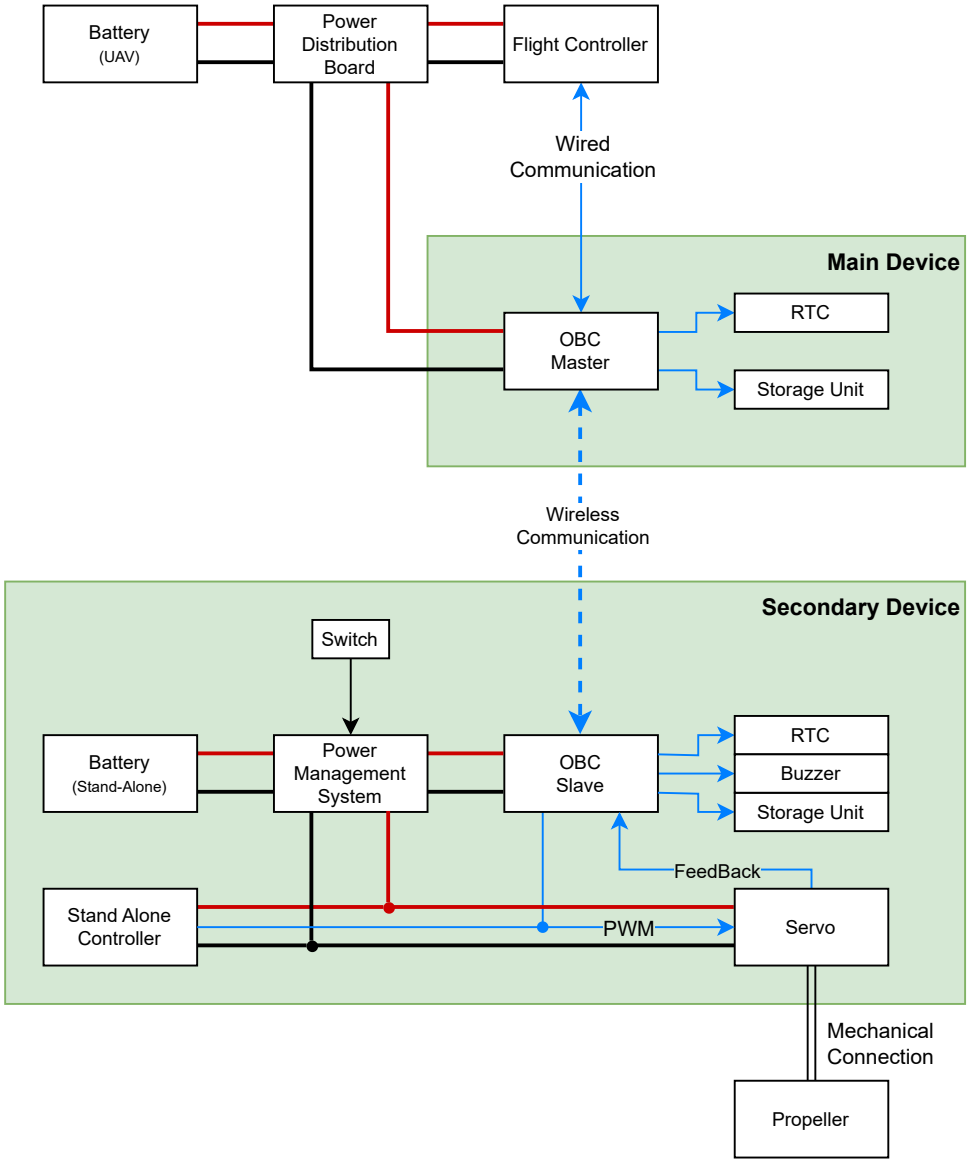


Figure 4.1: Proposed System Architecture High Level Diagram

The subsystems can be considered as stand-alone since the main UAV can work without the subsystems. These subsystems, described in the next subchapters, are responsible for monitoring the flight phase and controlling the propeller pitch angle.

4.3.1 Main Device

This subsystem will be, mainly, composed by an OBC, a Real-Time Clock (RTC), a Storage Unit, and a wireless communication module.
The MDU will communicate with the Flight Controller and the SDUs by receiving and transmitting information.

With the Flight Controller, through wired communication, the MDU will:

- Receive
  - Flight Phase message

  - Maneuver control message
  - Global Navigation Satellite System (GNSS) epoch time
  - Heartbeat signal
- Transmit
  - Heartbeat signal
  - System Status message

The Flight Phase message and the Maneuver Control message will inform the MDU about the current flight phase and the need to make additional adjustments to the propeller pitch. After interpreting the message, the OBC will send a control command. This control command will be explained further in this chapter

The GNSS epoch time will update the OBC date and time (periodically or on startup). With the help of the RTC, the MDU system will be able to maintain the date and time even when the UAV system is powered off. The GNSS epoch time will be helpful when storing system logs (in the Storage Unit) and will help to calculate the latency of the communication between devices.

Lastly, the received heartbeat signal will work as a *keep alive* mechanism informing, this way, the OBC if the system is powered on. This will help save power since the OBC can shut down when the Flight Controller turns off.

The transmitted heartbeat, which also works as a *keep alive* mechanism, will inform the Flight Controller that the MDU is working correctly. This function will be crucial because if the MDU is not working (powered off or unresponsive) the UAV system will need to enter a failsafe mode and land, as soon as possible, since it can no longer control the pitch of the blades.

Since the MDU is responsible for managing all the SDUs, it must, periodically, inform the Flight Controller about the overall status of the system, so that, in case of any failure, the Flight Controller may enter in failsafe.

With the SDUs, through wireless communication, the MDU will:

- Receive
  - Heartbeat signal
  - SDU Status message
- Transmit
  - Heartbeat signal
  - Control Command
  - Epoch Time

The received and transmitted heartbeat signals will have the same functionality as explained previously. The MDU and SDUs will inform each other if they are working correctly.

The SDU Status message will help the MDU keep track of the status of all Secondary devices. In case of malfunction or if one or more SDUs can't change the propeller pitch, the MDU must be noticed so that it can communicate to the Flight Controller about the failure.

The MDU will send a Control Command, containing the desired pitch, to all the SDUs according to the phase of flight message received previously.

By sending the epoch time to all the SDUs, it is possible to keep the whole system updated and with the same date and time reference.

4.3.2 Secondary Device

This subsystem will be, composed of an OBC, a RTC, a Storage Unit, a battery (with a power management system), an on/off switch, a buzzer, a stand-alone Pulse Width Modulation (PWM) controller, a servo, and a wireless communication module.

The on/off switch and the buzzer will work as human-machine interfaces to help the user interact with the system.
Since the SDU will be designed to be stand-alone (with a dedicated power supply) the system needs to be powered on manually and the buzzer can notice the user that the system is powered on.

The servo, mechanically connected to the propeller, will be responsible for changing the propeller pitch according to the state of flight. It will be equipped with feedback functionality so that the system can control, more precisely, the pitch and know if the propeller has reached its goal.

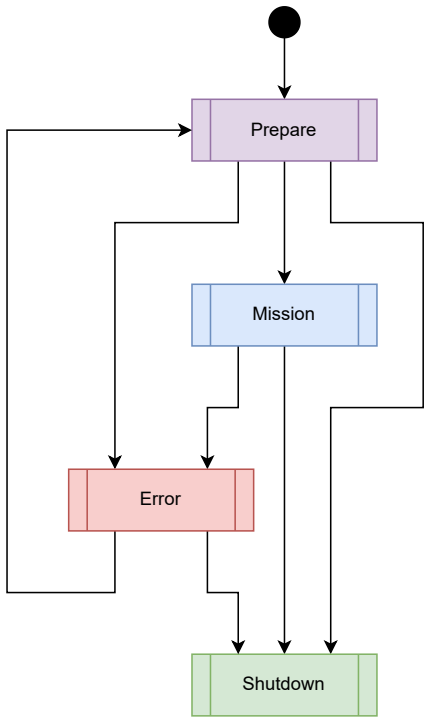
There will also be implemented a stand-alone PWM controller, able to generate a fixed PWM signal, to control the servo in case of failure from the OBC. As a failsafe mechanism, if one or more SDUs fail, the PWM controller will generate a PWM signal fixing the pitch of the propellers to a designated angle.
This action will transform the UAV system into a fixed-pitch proprotor but will help avoid having a system with a single point of failure that can cause the UAV to crash and possibly hurt people.

The Power Management System will be responsible for monitoring the State of Charge (SoC) of the battery, for converting the voltage from the battery to the needed voltage levels and for distributing to all components.
An Under Voltage Protection (UVP) will be also implemented to ensure that the SDU subsystem shuts down when the battery is at a critical level.

4.4 System Behavior

In order to properly design the system behavior it was developed a system flow chart. The flow chart, represented in algorithm 4.1, describes the expected high-level behavior of both MDU and SDU.

Algorithm 4.1 Proposed System Behavior - High-Level Flow Chart



In this flow chart, there are four main tasks:

- Prepare
- Mission
- Error
- Shutdown

4.4.1 Main Device Unit Behavior

In this chapter, it is described all the tasks referring to the Main Device Unit Behavior.

Prepare Task

The **Prepare** task is responsible for preparing the subsystem after activation and checking if the system is ready for the mission.

Firstly it will check the connection and the number of SDUs (this also represents the number of proprotors). In case of an insufficient number of SDUs, the subsystem will enter a state of Error and enter the **ERROR** task.

After this, the subsystem checks the connection with the Flight Controller (FC). If there is no response from the FC (meaning, for example, that the FC is turned off), the MDU subsystem will turn off by entering in the task **Shutdown**.

The next step is to publish an heartbeat to the SDUs (using topic *mdu/heartbeat*) and to acquire Epoch time to update its own date and time (if not possible use stored date and time) and update all the connected SDUs by publishing in *mdu/date_time* topic.

Finally, in this task, the MDU will subscribe to *sdu/heartbeat* and *sdu/status*. If all the SDUs are ready, the subsystem is ready for the mission and, in this case, enter **Mission** task. Otherwise, if any or all the SDUs have any problem or can not be reached, the MDU will enter the **ERROR** task.

The MDU **Prepare** task flow is represented in figure A.1 in Appendix A.

Mission Task

In the **Mission** task the MDU will monitor the flight phase (given by the FC) and publish, accordingly, to *mdu/pitch_cmd* topic. Since each phase requires a different pitch angle, it was defined three flight phases:

- Take-Off
- Landing
- Forward Flight

And while Take-Off and Landing phase, the pitch angles are defined, fixed and equal between all proprotors, in the Forward Flight phase each maneuver will require a different pitch angle and may require different pitches between each proprotor.

In parallel, the subsystem will also be publishing an heartbeat to the SDUs, checking the heartbeat from the FC and monitoring the heartbeat and status of all SDUs. By constantly monitoring the heartbeat and status of all the SDUs it is possible to recognize errors in the system and try to find solutions (in **Error** task) to avoid mission failures.

The MDU **Mission** task flow is represented in algorithm B.1 in Appendix B.

Error Task

Error task will be responsible for analyzing the error type and trying to resolve the error before mission failure. If the error is resolved, the system will re-enter the **Prepare** task otherwise it will enter **Shutdown** task.

In case one or more SDUs are unreachable, the MDU will send a reboot command to the unreachable SDUs to try to resolve the communication problem. And, in case one or more SDUs can not change the pitch angle to the desired one, the MDU will force the value again before shutting down.

The MDU Error task flow is represented in algorithm C.1 in Appendix C

Shutdown Task

When the subsystem enters **Shutdown** task, it will start by publishing the shutdown command to *mdu/shutdown*, so that all SDUs can turn off. Next, it will close the communication with the storage unit to avoid corrupted data or files. And finally, enter a low-power sleep mode until the user disconnects the UAV battery.

The MDU Shutdown task flow is represented in algorithm D.1 in Appendix D.

4.4.2 Secondary Device Unit Behavior

In the same way, this chapter will describe all the tasks referring to the Secondary Device Unit Behavior.

Prepare Task

In the SDU **Prepare** task, the subsystem will be prepared after activation to ensure it is ready for the mission.

The first step will be subscribing to *mdu/heartbeat* and, if there is no response from the MDU, the SDU subsystem will turn off by entering in the task **Shutdown**.

The next step is to subscribe to *mdu/date_time* to acquire Epoch time and update its date and time.

After publishing to *sdu/heartbeat*, the SDU subsystem will perform a self-check test. In this test, the subsystem will vary the value of the pitch angle and check the feedback signal from the servo. If every movement is performed successfully, the SDU will publish an OK status to *sdu/status*. But in case the feedback is different, the SDU will enter in **Error** task.

Finally, in this task, the MDU will subscribe to *mdu/heartbeat*. If MDU is ready, the subsystem is ready for the mission and, in this case, enter **Mission** task, otherwise, the SDU will enter the **ERROR** task.

The SDU Prepare task flow is represented in algorithm E.1 in Appendix E.

Mission Task

The **Mission** task will monitor regularly the *mdu/pitch_cmd* topic. When it receives a new pitch command from the MDU, the SDU will change the pitch according to the command. Then it will check the servo feedback to ensure the propeller has the correct pitch and publish its status in *sdu/status*.

In parallel, the subsystem will also be publishing an heartbeat to the MDU, checking the heartbeat from the MDU and monitoring the shutdown command from the MDU.

The SDU Mission task flow is represented in algorithm F.1 in Appendix F.

Error Task

Error task will be responsible for analyzing the error type and trying to resolve the error before mission failure. If the error is resolved, the system will re-enter the **Prepare** task otherwise it will enter **Shutdown** task.

If the SDU can not change the propeller pitch angle correctly, it will force another try at changing the pitch angle. And, if there is no heartbeat from the MDU, the SDU will try to obtain the heartbeat again.

In case of unsuccess, in both cases, the SDU will activate the stand-alone PWM generator to fix the propeller pitch angle.

The SDU Error task flow is represented in algorithm G.1 in Appendix G.

Shutdown Task

When the SDU subsystem enters **Shutdown** task, it will start by setting the propeller pitch angle to a default value. Next, it will close the communication with the storage unit to avoid corrupted data or files. And finally, enter a low-power sleep mode until the user disconnects the UAV battery.

The SDU Shutdown task flow is represented in algorithm H.1 in Appendix H.

TODO: CHANGE "TOPICS" TO "NODES" ??

Chapter 5

Development Plan

5.1 Research Approach

To achieve the desired objectives and system requirements, the development approach will be composed of three phases: Dissertation Development, System Development, and Implementation.

During the first part of dissertation development, it will be analyzed the problems with fixed-pitch propeller systems, described in the previous chapters, in UAVs to be able to find the best approach to solve the issue at hand, to define the new system requirements, determine the objectives of the proposed solution and to design the system architecture. And, to gain a better understanding of the present status of the subject, a study of the literature related to the problem, will be conducted. In this phase, it will be also written all the steps and considerations taken during the development and implementation of the solution and an analysis of the results obtained.

As we go on to the System Development phase, there will be a detailed procurement to find the most adequate components, according to the system architecture and requirements, since it is necessary to design and manufacture Pinted Circuit Boards (PCBs) for the final prototype. In this phase, the system flow charts, shown in the appendix, will be modeled and validated using model checker tools like NuSMV. This step will increase the confidence in the designed firmware and validate the expected behavior of the system.

In the implementation phase, the Main and Secondary Devices will be soldered and assembled, to, later on, perform, bench and ground tests, and evaluate the developed system in comparison to predetermined goals and requirements. By documenting and analyzing the results it will be possible to make any necessary refinements to enhance performance.

5.2 Evaluation

In order to evaluate the system's performance, in comparison to the requirements, the analyses will be divided into three categories.

In the Communication category, it will be analyzed the stability and the latency of the chosen communication technology.

Another category is the Pitch Angle Control in which the precision and stability of the control over the pitch angle will be evaluated. It will also be analyzed if the system has a quick response to changes in flight phase and Maneuver, a quick response in error scenarios and if the fail-safe mechanism can set a fixed pitch angle.

The last category to be evaluated is the Firmware. In this category, the model of the system flow will be validated with mathematical tools as explained before.

This way, the system will be evaluated in each subsystem and as a whole.

5.3 Timeline

In the Gantt chart (figure 5.1) all phases, described previously in the Research Approach section, were added together with multiple tasks and subtasks each with a given duration and dependencies.

The first task started is the Research Plan, part of the Dissertation Development phase, and will be the starting point of future work. All the other tasks, in this phase, will be done in parallel until the end of the dissertation.

Next will be the System Development phase where hardware and firmware tasks will be made. These tasks will start in mid-January 2024 and end in early April 2024.

The last tasks will be from the Implementation phase with tests, evaluations and refinement tasks. They will be carried out from mid-March 2024 to mid-July 2024.

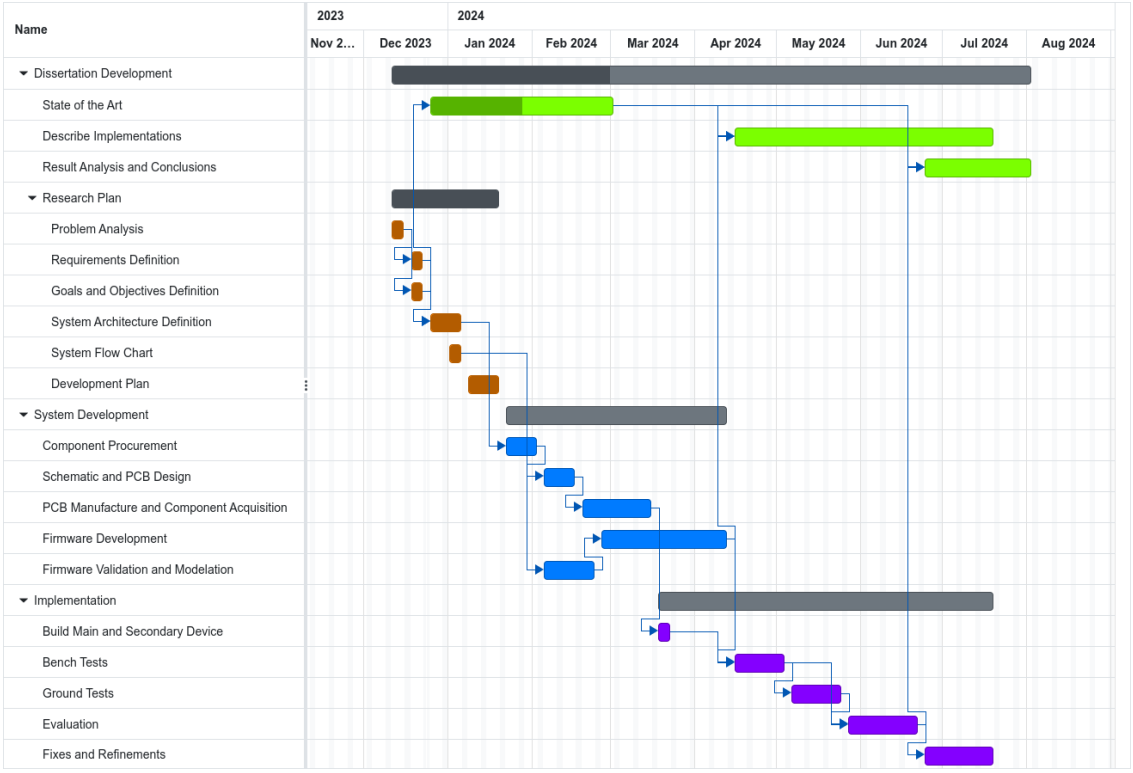


Figure 5.1: Project timeline Gantt chart

The workload will be the following:

Table 5.1: Development Work Load

Type	**Name**	**Duration (days)**
Main Task	Dissertation Development	170
Sub Task	Research Plan	30
Main Task	System Development	60
Sub Task	Hardware	40
Sub Task	Firmware	50
Main Task	Implementation	90
Sub Task	Tests	30

This timeline will help to ensure that all necessary activities are completed in the correct order and on time.

Bibliography

[1] Khaled Telli et al. "A Comprehensive Review of Recent Research Trends on UAVs". In: (Aug. 2023).

[2] János Mészáros. "AERIAL SURVEYING UAV BASED ON OPEN-SOURCE HARDWARE AND SOFTWARE". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 38 (Jan. 2011). doi: 10.5194/isprsarchives-XXXVIII-1-C22-155-2011.

[3] Emad Samuel Malki Ebeid et al. "A Survey of Open-Source UAV Flight Controllers and Flight Simulators". In: *Microprocessors and Microsystems* 61 (May 2018). doi: 10.1016/j.micpro.2018.05.002.

[4] L. Oyuki Rojas-Perez and J. Martinez-Carranza. "On-board processing for autonomous drone racing: An overview". In: *Integration* 80 (Sept. 2021), pp. 46–59. issn: 0167-9260. doi: 10.1016/j.vlsi.2021.04.007.

[5] Abdul Aabid et al. "Reviews on Design and Development of Unmanned Aerial Vehicle (Drone) for Different Applications". In: *Journal of Mechanical Engineering Research and Developments* 45 (Apr. 2022), pp. 53–69.

[6] Musa Galimov, Roman Fedorenko, and Alexandr Klimchik. "UAV Positioning Mechanisms in Landing Stations: Classification and Engineering Design Review". In: *Sensors* 20 (June 2020), p. 3648. doi: 10.3390/s20133648.

[7] Rooh Amin, Li Aijun, and Shahab Band. "A Review of Quadrotor UAV: Control Methodologies and Performance Evaluation". In: *International Journal of Automation and Control* 10 (Dec. 2015). doi: 10.1504/IJAAC.2016.076453.

[8] Mitchell Campion, Ranganathan Prakash, and Saleh Faruque. "UAV Swarm Communication and Control Architectures: A Review". In: *Journal of Unmanned Vehicle Systems* 7 (Nov. 2018). doi: 10.1139/juvs-2018-0009.

[9] Radheshyam Singh et al. "Overview of Drone Communication Requirements in 5G". In: Jan. 2023, pp. 3–16. isbn: 978-3-031-20935-2. doi: 10.1007/978-3-031-20936-91.

[10] Nashwan Othman and Ilhan Aydin. "Development of a Novel Lightweight CNN Model for Classification of Human Actions in UAV-Captured Videos". In: *Drones* 7 (Feb. 2023), p. 148. doi: 10.3390/drones7030148.

[11] Mark Cutler. "Design and Control of an Autonomous Variable-Pitch Quadrotor Helicopter". PhD thesis. Aug. 2012.

[12] Ching-Wei Chang et al. "An Actuator Allocation Method for a Variable-Pitch Propeller System of Quadrotor-Based UAVs". In: *Sensors (Basel, Switzerland)* 20 (Oct. 2020). doi: 10.3390/s20195651.

[13] Academic Accelerator. *Variable Pitch Propeller Aeronautics: Most Up-to-Date Encyclopedia, News & Reviews*. en. url: https://academic-accelerator.com/encyclopedia/variable-pitch-propeller-aeronautics.

[14] Vishnu S. Chipade et al. "Systematic design methodology for development and flight testing of a variable pitch quadrotor biplane VTOL UAV for payload delivery". en. In: *Mechatronics* 55 (Nov. 2018), pp. 94–114. issn: 09574158. doi: 10.1016/j.mechatronics.2018.08.008.

[15] EDN. *Selecting the Best Battery for Embedded-System Applications*. en-US. Nov. 2010. url: https://www.edn.com/selecting-the-best-battery-for-embedded-system-applications/.
[16] en-US. url: https://www.toradex.com/blog/using-lithium-ion-batteries-in-embedded-systems.
[17] url: https://www.nkon.nl/pt/samsung-inr-18650-30q-3000mah.html.
[18] en. url: https://thepihut.com/products/1200mah-3-7v-lipo-battery.
[19] url: https://www.nkon.nl/pt/rechargeable/nimh/gp-ni-mh-d-11000mah.html.
[20] admin. *NiCd Battery Charger Circuit*. en-US. July 2019. url: https://theorycircuit.com/nicd-battery-charger-circuit/.
[21] url: https://www.nkon.nl/pt/rechargeable/lead-acid-batteries/6v-vrla/yuasa-6v-1-2ah-loodaccu.html.
[22] url: https://www.nkon.nl/pt/c-duracell-industrial-1-5v.html.
[23] Fikret Basic, Christian Steger, and Robert Kofler. "Embedded Platform Patterns for Distributed and Secure Logging". In: *26th European Conference on Pattern Languages of Programs*. EuroPLoP21. New York, NY, USA: Association for Computing Machinery, Jan. 2022, pp. 1–9. isbn: 978-1-4503-8997-6. doi: 10.1145/3489449.3490004. url: https://dl.acm.org/doi/10.1145/3489449.3490004.
[24] Staff. *Selecting the Correct Memory Type for Embedded Applications*. Jan. 2007. url: https://www.electronicdesign.com/technologies/embedded/digital-ics/memory/article/21787261/selecting-the-correct-memory-type-for-embedded-applications.
[25] Risto Avila. *How to Select a Memory Configuration for Embedded Systems*. en. Dec. 2021. url: https://www.qt.io/embedded-development-talk/memory-options-for-embedded-systems-how-to-select-the-right-memory-configuration.
[26] en-pt. url: https://www.digikey.pt/en/articles/the-fundamentals-of-embedded-memory.
[27] en. Jan. 2021. url: https://www.kingston.com/en/blog/personal-storage/microsd-sd-memory-card-guide.
[28] en. Dec. 2023. url: https://www.kingston.com/en/blog/pc-performance/hdd-vs-external-ssd.
[29] J. E. Cooling. "Languages for the programming of real-time embedded systems a survey and comparison". In: *Microprocessors and Microsystems* 20.2 (Apr. 1996), pp. 67–77. issn: 0141-9331. doi: 10.1016/0141-9331(95)01067-X.
[30] L. Prechelt. "An empirical comparison of seven programming languages". In: *Computer* 33.10 (Oct. 2000), pp. 23–29. issn: 1558-0814. doi: 10.1109/2.876288.
[31] Risto Avila. *Embedded Software Programming Languages: Pros, Cons, and Comparisons of Popular Languages*. en. Mar. 2022. url: https://www.qt.io/embedded-development-talk/embedded-software-programming-languages-pros-cons-and-comparisons-of-popular-languages.
[32] Dmitry Koshkin. *All You Need To Know About Embedded Systems Programming*. en-US. May 2020. url: https://sam-solutions.us/all-you-need-to-know-about-embedded-systems-programming/.
[33] en. Aug. 2023. url: https://www.linkedin.com/pulse/5-best-embedded-systems-programming-languages-pros.
[34] S Ramanarayana Reddy. "Selection of RTOS for an Efficient Design of Embedded Systems". en. In: (2006).

[35] Kristoffer Skoien. *RTOS: Real-Time Operating Systems for Embedded Developers*. en-gb. June 2021. url: https://blog.nordicsemi.com/getconnected/what-is-rtos-real-time-operating-systems-for-embedded-developers.

[36] About The Author Carsten Rhod Gregersen. *FreeRTOS vs ThreadX vs Zephyr: The Fight For True Open Source RTOS*. en-US. Jan. 2023. url: https://www.iiot-world.com/industrial-iot/connected-industry/freertos-vs-threadx-vs-zephyr-the-fight-for-true-open-source-rtos/.

[37] en-US. Nov. 2023. url: https://micro.ros.org/docs/concepts/rtos/comparison/.

[38] url: https://www.chibios.org/dokuwiki/doku.php?id=chibios:documentation:books:rt:intro.

[39] url: https://nuttx.apache.org/docs/latest/introduction/about.html.

[40] url: https://docs.zephyrproject.org/latest/introduction/index.html.

[41] Yong Zeng, Rui Zhang, and Teng Joon Lim. "Wireless communications with unmanned aerial vehicles: opportunities and challenges". In: *IEEE Communications Magazine* 54.5 (May 2016), pp. 36–42. issn: 1558-1896. doi: 10.1109/MCOM.2016.7470933.

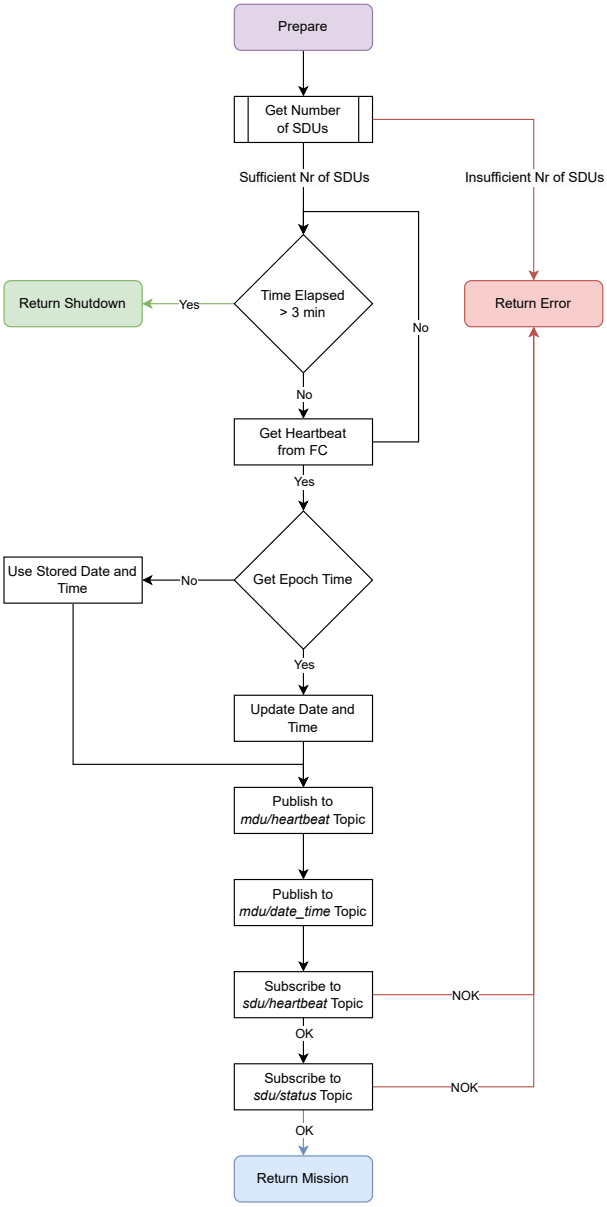
[42] Federico Montori et al. "Machine-to-machine wireless communication technologies for the Internet of Things: Taxonomy, comparison and open issues". In: *Pervasive and Mobile Computing* 50 (Oct. 2018), pp. 56–81. issn: 1574-1192. doi: 10.1016/j.pmcj.2018.08.002.

[43] E. Ferro and F. Potorti. "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison". In: *IEEE Wireless Communications* 12.1 (Feb. 2005), pp. 12–26. issn: 1558-0687. doi: 10.1109/MWC.2005.1404569.

Appendix A

Main Device Flow Chart - Prepare Task

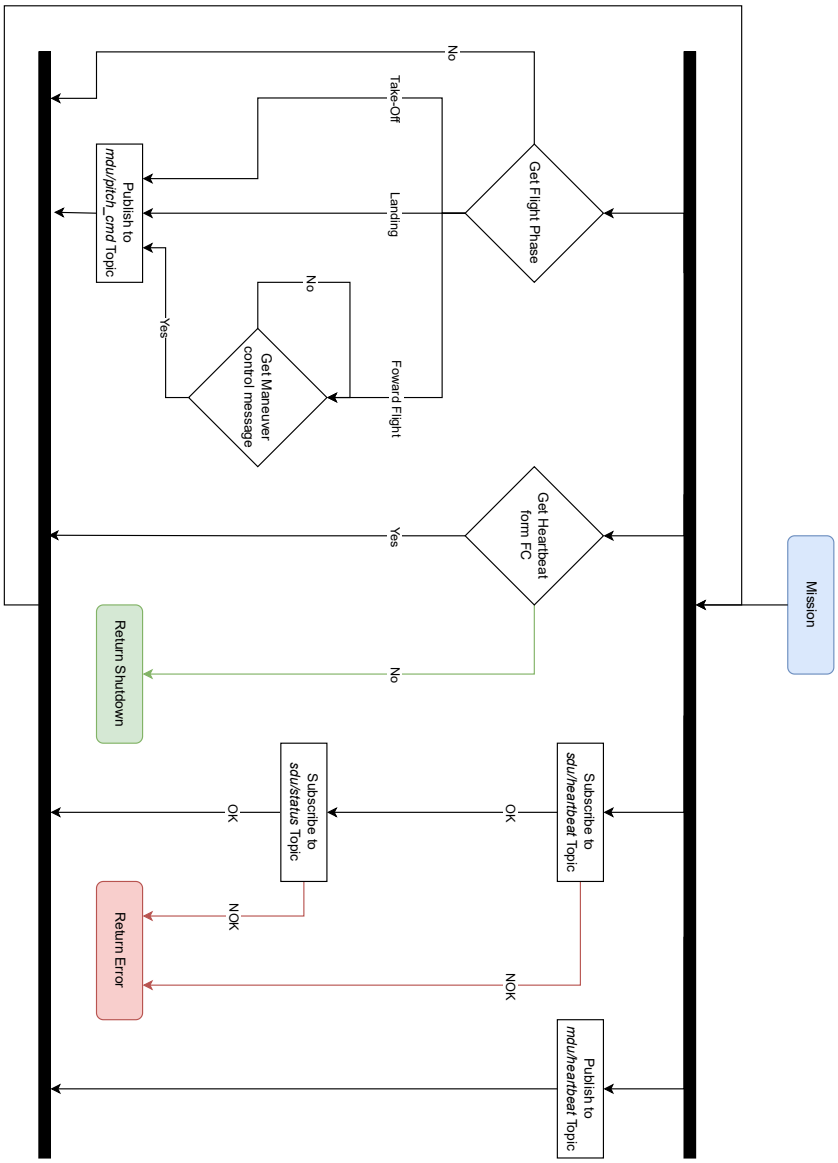
Algorithm A.1 Proposed System Behavior - Prepare Task Flow Chart (MDU)



Appendix B

Main Device Flow Chart - Mission Task

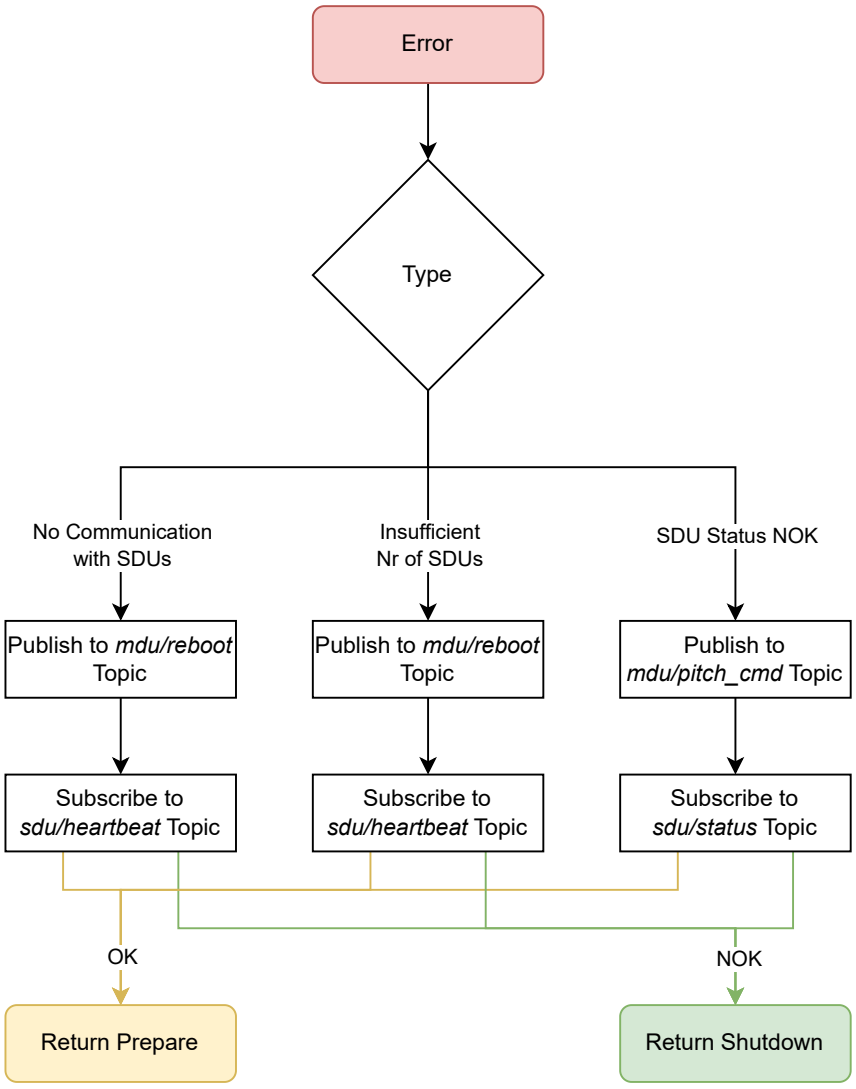
Algorithm B.1 Proposed System Behavior - Mission Task Flow Chart (MDU)



Appendix C

Main Device Flow Chart - Error Task

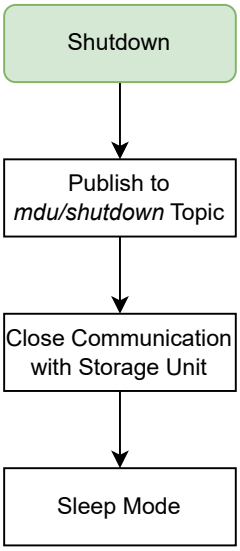
Algorithm C.1 Proposed System Behavior - Error Task Flow Chart (MDU)



Appendix D

Main Device Flow Chart - Shutdown Task

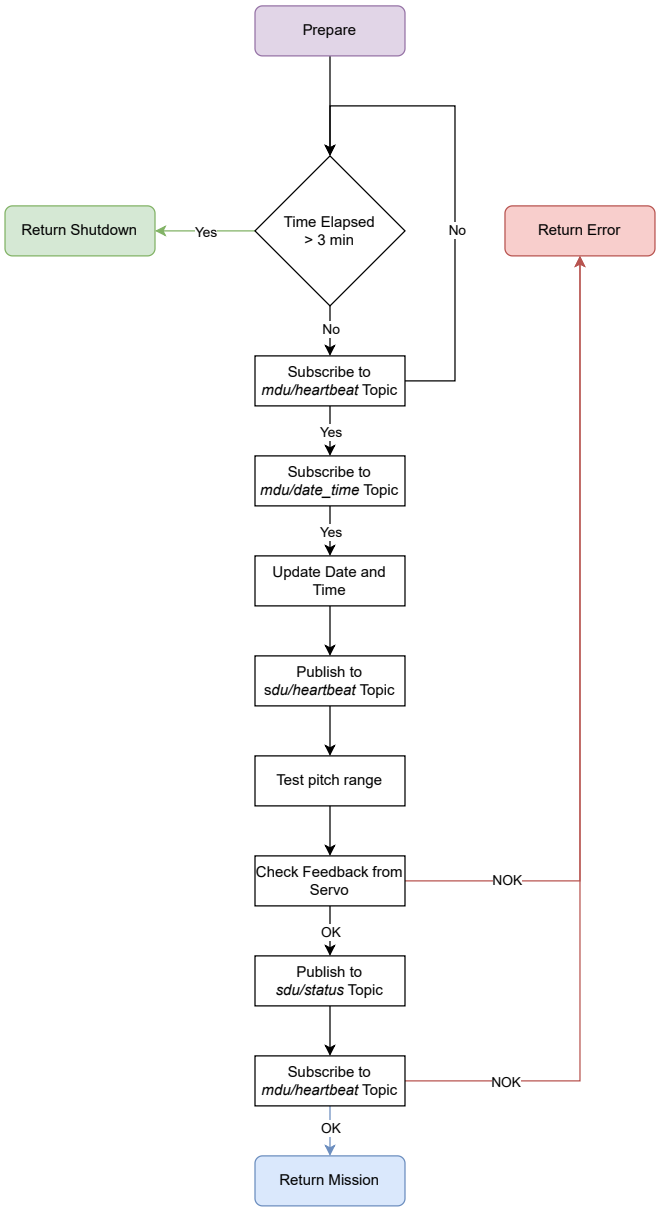
Algorithm D.1 Proposed System Behavior - Shutdown Task Flow Chart (MDU)



Appendix E

Secondary Devices Flow Chart - Prepare Task

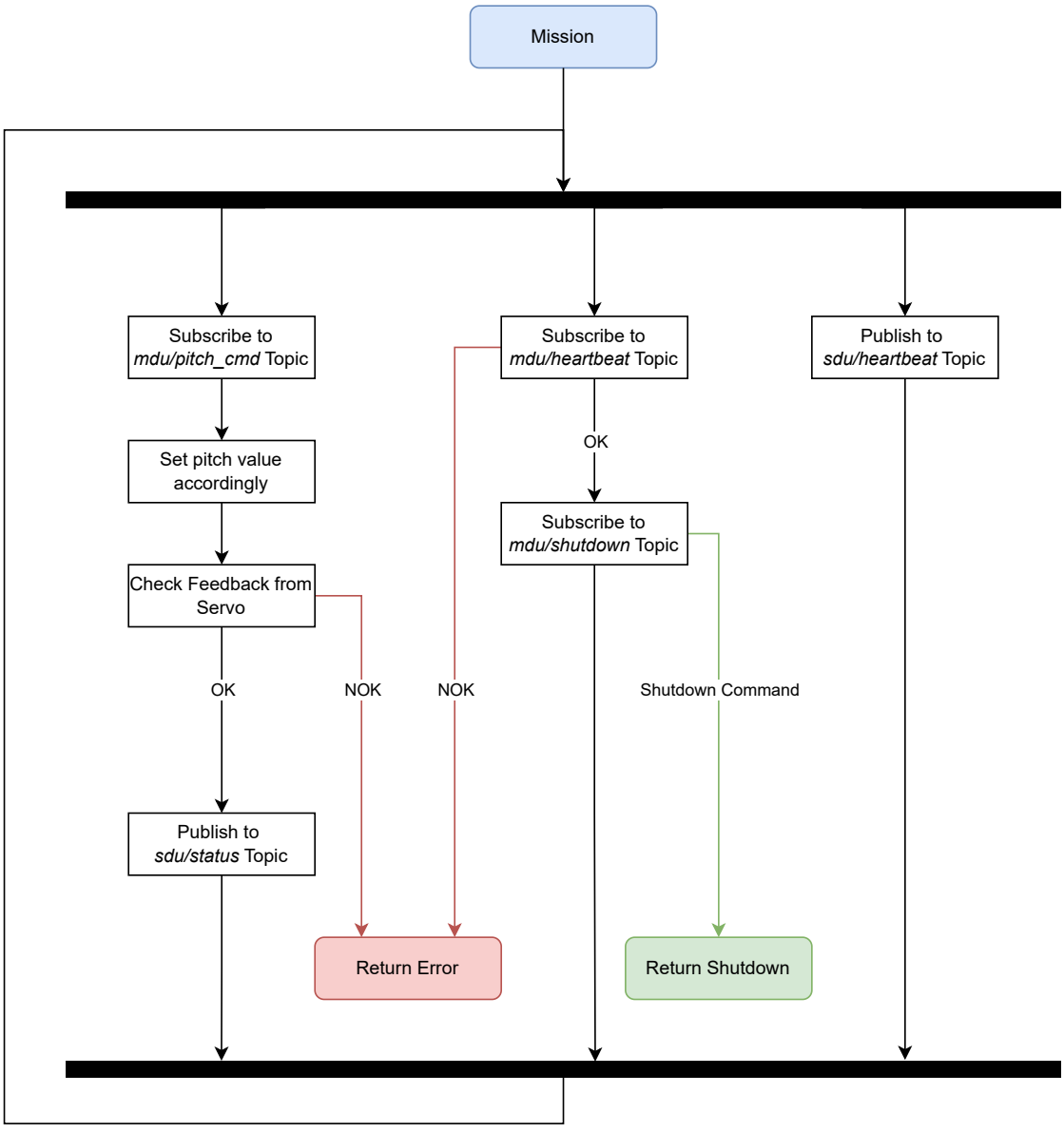
Algorithm E.1 Proposed System Behavior - Prepare Task Flow Chart (SDU)



Appendix F

Secondary Devices Flow Chart - Mission Task

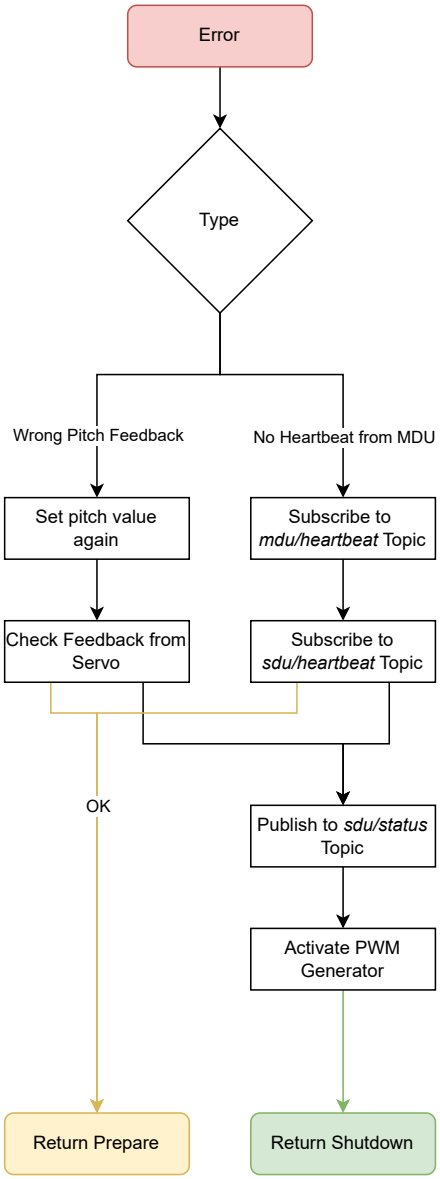
Algorithm F.1 Proposed System Behavior - Mission Task Flow Chart (SDU)



Appendix G

Secondary Devices Flow Chart - Error Task

Algorithm G.1 Proposed System Behavior - Error Task Flow Chart (SDU)



Appendix H

Secondary Devices Flow Chart - Shutdown Task

Algorithm H.1 Proposed System Behavior - Shutdown Task Flow Chart (SDU)