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, February 4, 2024

Abstract

Unmanned Aerial Vehicles (UAVs) have gained significant attention in diverse applications, prompting a surge in research and development. Current Vertical Take-Off and Landing (VTOL) UAVs rely on fixed-pitch proprotors which makes them face limitations in performance and maneuverability.

This thesis addresses these challenges by proposing the development of a stand-alone Variable Pitch Proprotor (VPP) system capable of real-time pitch adjustments during different flight phases.

The research delves into the fundamentals of VTOL flight performance, variable proprotor impact, control subsystems, power management, firmware strategies and short-range wireless communication protocols. Studying and analyzing existing solutions and technologies in these domains is conducted.

The subsequent stages involve designing the subsystem architecture, implementing it on a mechanical prototype, and evaluating the system's performance and limitations in various settings.

The goal, within this Thesis, is to enhance UAVs stability, maneuverability, and energy efficiency, contributing to the advancement of Urban Air Mobility (UAM).

Keywords: UAV, Variable-Pitch Proprotor System, RTOS, OBC, Wireless Communication

Resumo

Veículo Aéreo Não Tripulados (VANTs), ou Unmanned Aerial Vehicles UAVs em inglês, têm tido um aumento significativo na sua popularidade e atenção no estudo. Tornaram-se indispensáveis em diversas aplicações, desde vigilância até reconhecimento, devido à sua versatilidade e eficiência em várias áreas.

Atualmente, a maioria dos VANTs de decolagem e aterragem vertical (Vertical Take-O and Landing VTOL) depende de prorotores com sistemas de passo fixo devido à sua simplicidade e à falta de soluções comerciais prontas e eficientes. No entanto, esses prorotores impõem limitações ao desempenho de voo, restringindo a agilidade e a manobrabilidade. Essas limitações tornam-se mais evidentes à medida que o tamanho do VANT aumenta.

O desenvolvimento de sistemas de prorotores de passo variável (Variable Pitch Proprotor (VPP)) desempenha um papel importante na superação destas limitações. A inclusão de prorotores de passo variável proporciona a flexibilidade necessária para melhorar

Este trabalho tem como objetivo desenvolver um sistema independente de prorotores de passo variável que pode, em tempo real, alterar o passo da hélice de acordo com cada fase de voo.

A pesquisa aprofunda os fundamentos do desempenho de voo VTOL, o impacto de prorotores de passo variável, subsistemas de controlo, gerenciamento de energia, estratégias de firmware e protocolos de comunicação sem fio de curto alcance. Após a pesquisa, é realizado um estudo e análise de soluções e tecnologias existentes nestes domínios.

As etapas subsequentes envolvem desenhar a arquitetura do subsistema, implementar num protótipo mecânico e avaliar o desempenho do sistema e limitações em várias condições.

O objectivo, no âmbito desta tese, é melhorar a estabilidade, a manobrabilidade e a eficiência energética, contribuindo para o avanço da Mobilidade Aérea Urbana (UAM).

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

ADC	Analog Digital Converter.
BLE	Bluetooth Low Energy.
CAGR	Compound Annual Growth Rate.
CAN	Controller Area Network.
COTS	Commercial Off-The Shelf.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
eMMC	embedded MultiMedia Card.
FIFO	First In First Out.
FPGA	Field-Programmable Gate Arrays.
FRAM	Ferroelectric Random Access Memory.
GNSS	Global Navigation Satellite System.
HDD	Hard Disk Drives.
HDL	Hardware Description Languages.
I2C	Inter-Integrated Circuit.
IEEE	Institute of Electrical and Electronics Engineers.
IoT	Internet of Things.
Li-ion	Lithium-ion.
Li-Po	Lithium Polymer.
LoRaWAN	Long Range Wide Area Network.
MLC	Multi-Level Cell.
NFC	Near Field Communication.
NiCd	Nickel-Cadmium.
NiMH	NickelMetal Hydride.
OBC	On Board Computer.
PCB	Pinted Circuit Board.
PWM	Pulse Width Modulation.

RAM	Random Access Memory.
RFID	Radio Frequency Identification.
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.
SPI	Serial Peripheral Interface.
SSD	Solid State Drives.
TLC	Triple-Level Cell.
TLS	Transport Layer Security.
UART	Universal Asynchronous Receiver-Transmitter.
UAV	Unmanned Aerial Vehicle.
USB	Universal Serial Bus.
UVP	Under Voltage Protection.
VANT	Veículo Aéreo Não Tripulado.
VTOL	Vertical Take-Off and Landing.
Wifi	Wireless Fidelity.

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, possible to see in figure 1.1 [1], [2].

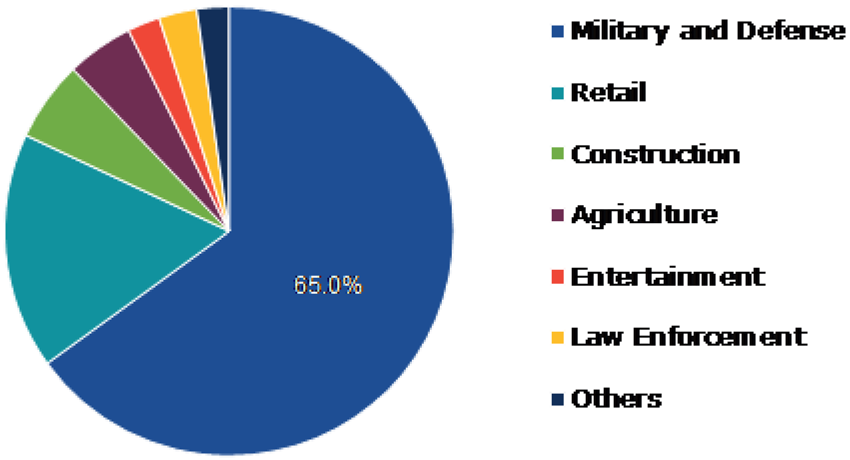


Figure 1.1: UAV Market Share, by End-Use, 2021[2]

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

It is estimated that UAV market will grow at a Compound Annual Growth Rate (CAGR) of 19.6% and generate a revenue of 95012,76 million by 2030 [2], [7]. The graph illustrated in figure 1.2, shows that this estimated growth is expected to occur worldwide [2].

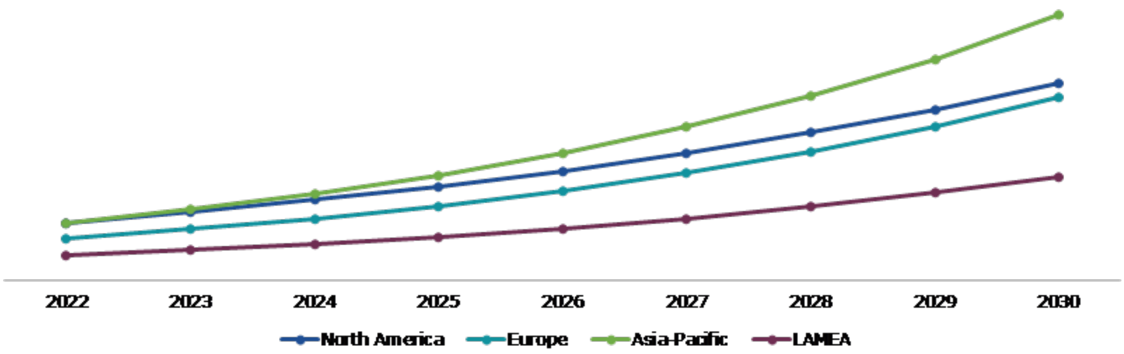


Figure 1.2: UAV Market Size and Forecast, by Region, 2021-2030 [2]

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 [8]. Thrust generation is confined to a single direction, hindering the UAV's ability to produce upward thrust relative to the vehicle body. Additionally, the UAVs control is restricted by the inertia of the motors and proprotors, constraining the UAV's agility and maneuverability [8]. As described in recent studies [8], 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 [8], [9]. Several detailed descriptions of quadrotors, for instance, modeling and dynamics have been published [10], [11], [12], [13], 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 [9].

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 [8], [9]. The graphs shown in figure 1.3 and 1.4, illustrate the differences between fixed-pitch and variable-pitch propellers (constant-speed in this graphs).

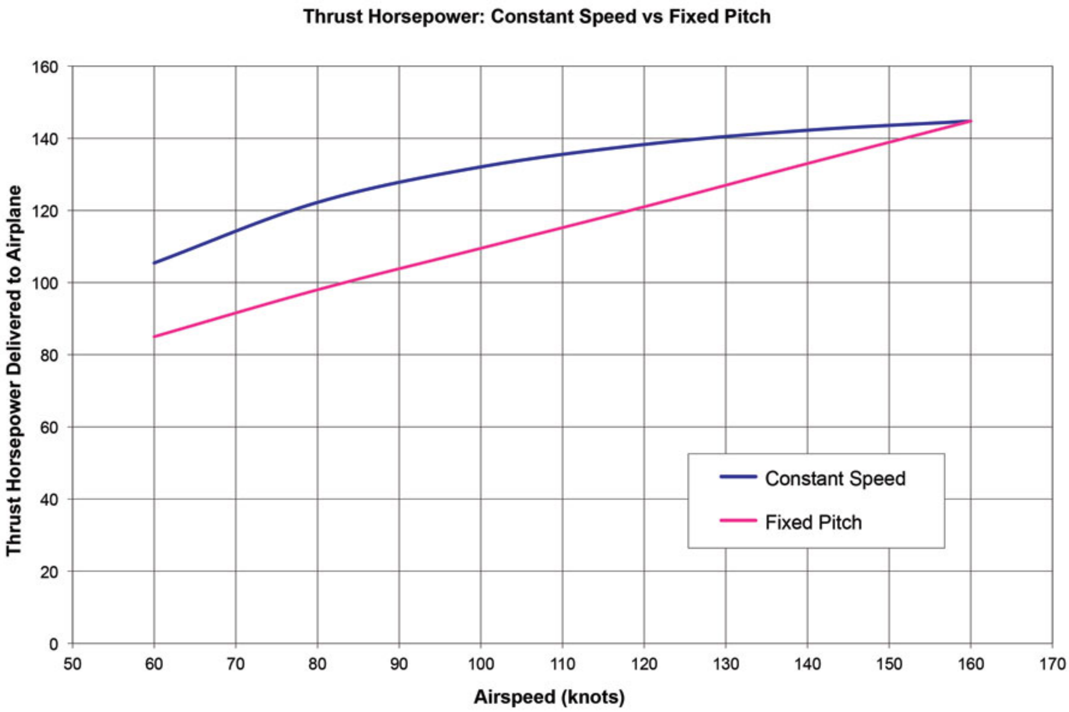


Figure 1.3: Thrust horsepower comparison between FPP and VPP [14]

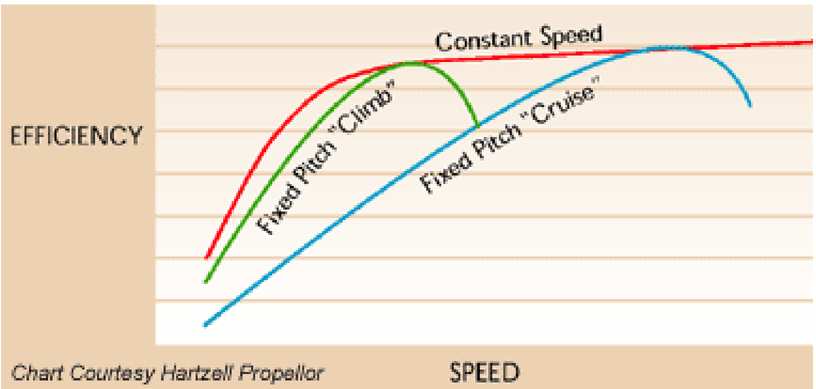


Figure 1.4: Efficiency comparison between FPP and VPP [15]

It is possible to see that when the propeller reaches its maximum airspeed design point, there is little to no difference between the performance of the fixed-pitch and variable-pitch propellers. The difference in performance and efficiency can be seen when the airspeed is below or above the maximum design point [15].

This Thesis aims to develop a stand-alone variable-pitch proprotor system that can, in real-time, change the propeller pitch according to each flight phase. This work will be integrated into a PhD Thesis that is currently being developed at Universidade da Beira Interior by Eng. Renato Machado about Urban Air Mobility.

1.1 Problem Analysis

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.

In terms of maneuverability and response, fixed-pitch propellers inherently constrain the UAVs from adjusting the pitch angle during flight [8]. This limitation results in compromised maneuverability and response, restricting the range of aerobatic maneuvers that a UAV can execute [8]. 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 [8].

As for power consumption in systems with fixed-pitch propellers, the power consumption will be higher. Without the ability to adjust the propeller angle, the UAVs may be forced to operate at higher RPMs to compensate for this lack of adjustment [8]. 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.

1.2 Objectives

As explained before, the focus will be on developing a stand-alone variable-pitch proprotor system.

The first goal will be the understanding of the relevant fundamentals regarding VTOL flight performance and the impact of variable proprotors.

Next, describe the fundamentals of control subsystems, power management and short-range wireless communication protocols with particular emphasis on their reliability and real-time sensing and actuation. There will also be made a survey about current solutions, communication technologies, electronic control and power management strategies.

After the research, it will be designed the subsystems architecture and then implemented the envisaged subsystem over a real mechanical prototype.

Finally, the performance of the system will be evaluated together with its limitations under different settings and environments.

Chapter 2

State of the Art

In this chapter, it will be presented an overview of UAVs, the fundamentals of propellers, fixed-pitch propellers and variable-pitch propellers.

2.1 Unmanned Aerial Vehicle

UAVs, commonly known as drones, have gathered significant attention in both civil and military operations due to their exceptional mobility, enhanced stability, cost-effectiveness, and endurance across various tasks. They can be found in applications of diverse fields such as logistics, forest monitoring, construction, freight transportation, communication, healthcare, post-disaster operations, search and rescue, remote sensing, precision agriculture, power-line inspection, traffic surveillance, as well as object detection and tracking [1], [4], [16].

In essence, UAVs are unmanned aerial vehicles, capable of autonomous and remote operation.. They rely on communication links to connect with ground control stations. Remote operation typically involves a human operator monitoring and/or controlling the UAV through a remote control [4], [16].

Given the widespread interest in drones, numerous UAVs of varying sizes and forms (illustrated in Figure 2.1) have been developed to fulfill a range of tasks [4], [17], [16].

Each type of UAV comes with its own set of advantages and disadvantages, guiding the selection based on the specific application requirements [4], [17], [16].

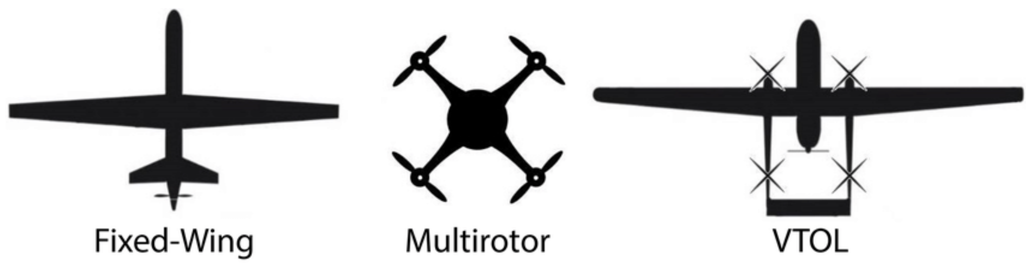


Figure 2.1: Examples of UAV types [17]

2.1.1 Propeller Fundamentals

The purpose of a propeller is to convert the rotational power produced by the engine into forward thrust during flight. This is achieved by accelerating a mass of air through the blades of the propeller as it spins, generating the necessary force to propel the aircraft forward at a specific airspeed [18]. Figure 2.2, illustrates the connection between the changing the propeller rotation speed and the speed and movement.

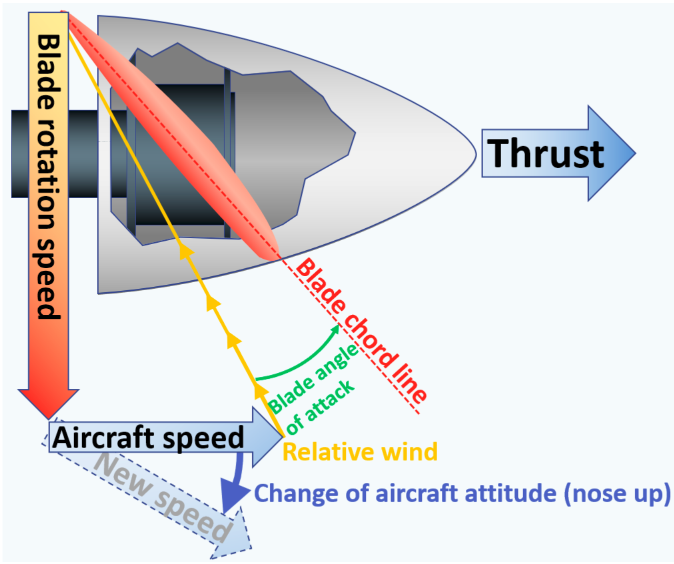


Figure 2.2: Propeller blade representation [19]

The propeller performance can be affected by [18]:

- Blade diameter
- Number of blades
- Blade pitch

While the propeller diameter affects lift efficiency (higher diameter, higher efficiency), the number of blades affects the thrust [18].

As for the propeller pitch, it is considered as the distance that a propeller moves forward in one revolution. A higher pitch allows the propeller to move the aircraft faster through the air and while a lower pitch results in less forward movement but may be more suitable for applications where lower speeds or higher thrust are required [18].

This means that each flight phase requires a different propeller pitch[18]:

- Take-off - Fine Pitch
- Hover - Finest Pitch
- Flight - Coarse pitch
- Landing - Fine Pitch

2.1.2 Fixed Pitch Proprotors

Fixed-pitch propellers, as the name suggests, have a predetermined blade pitch that remains constant during operation. The design, like the example in figure 2.3, involves selecting an ideal blade pitch based on the combination of motor angular speed and airspeed at which the propeller operates.

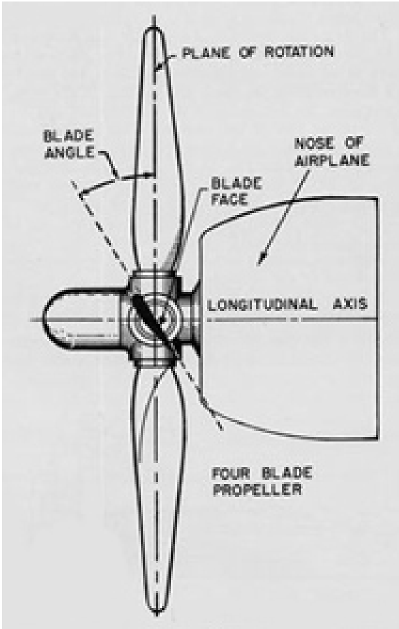


Figure 2.3: Example of Fixed Pitch [20]

To address varying thrust requirements at different airspeeds, it is possible to choose between a fine fixed-pitch propeller for take-off and climb and a coarse one for cruise.

While simple to operate, fixed-pitch propellers require alignment of factors such as motor RPM, airspeed, relative airflow, blade pitch, diameter, number of blades, blade chord and length. This design proves inefficient for various flight modes [18], [21], [22].

2.1.3 Variable Pitch Proprotors

Historically, early aviation pioneers experimented with propellers that could only be adjusted on the ground. The Gloster Hele-Shaw Beacham Variable Pitch Propeller (VPP), 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 [23].

A significant advantage of VPP is its 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 [18], [9].

The primary purpose of VPPs 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 and cruise [23], [24].

Variable-pitch systems, like in figure 2.4, 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 [23], [24].

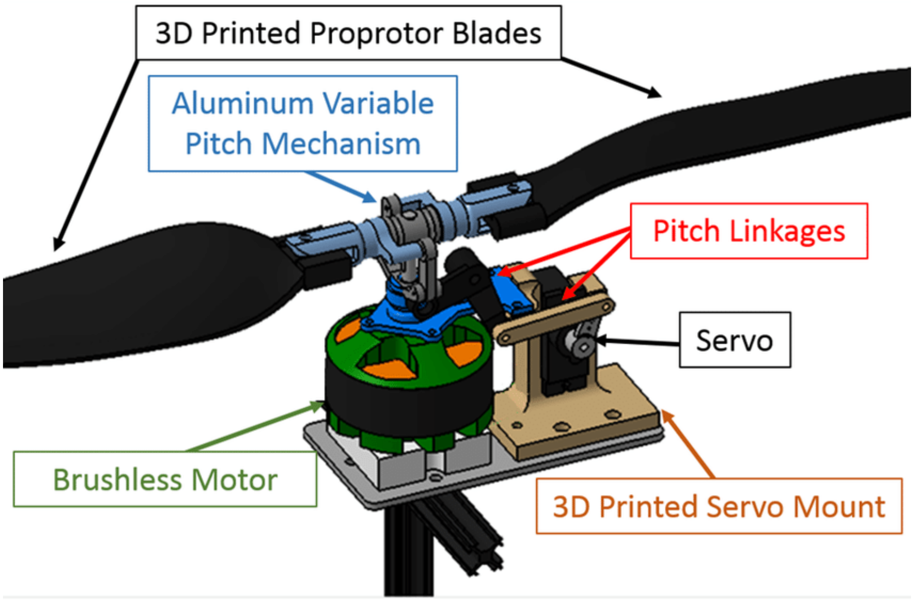


Figure 2.4: Example of Variable Pitch Mechanism [25]

There are three methods to change the pitch: Hydraulic, Centrifugal, and Electromechanical control [23].

Hydraulic Method

This system involves the use of engine oil pressure to control the pitch-changing mechanism, as it is possible to see in figures 2.5 and 2.6. It consists of a pump, control valves, and cylinders that actuate the movement of the propeller blades [23], [26]..

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 [27].

But Hydraulic systems add complexity and weight to the overall aircraft system. More components means more elements could potentially fail or require maintenance. Hydraulic systems also have a slow response time due to the time it takes for hydraulic pressure changes to propagate through the system [23], [26]..

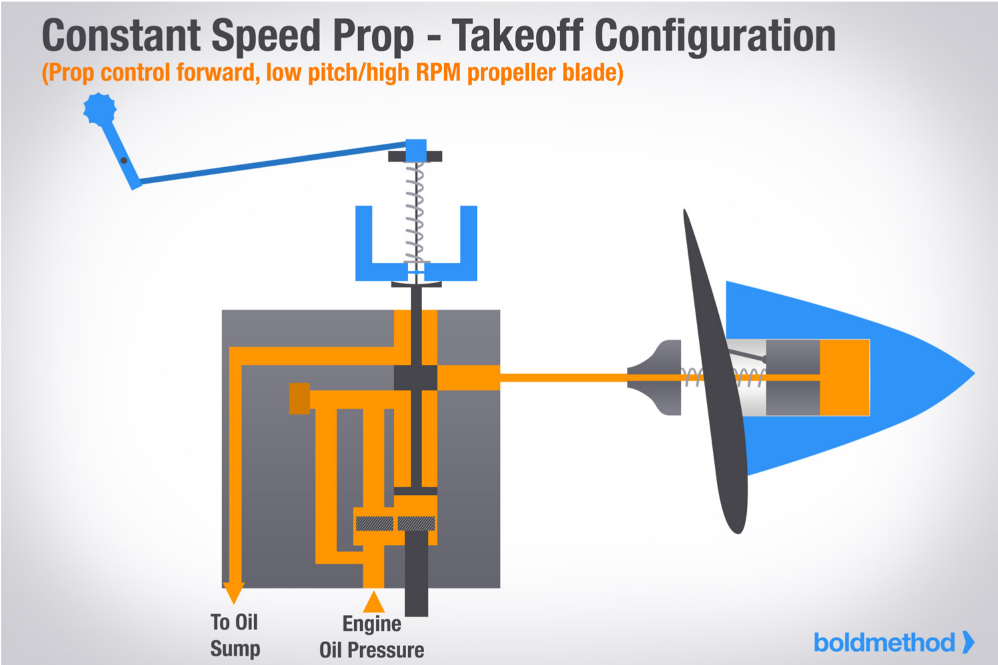


Figure 2.5: VPP Hydraulic Method - Takeoff [27]

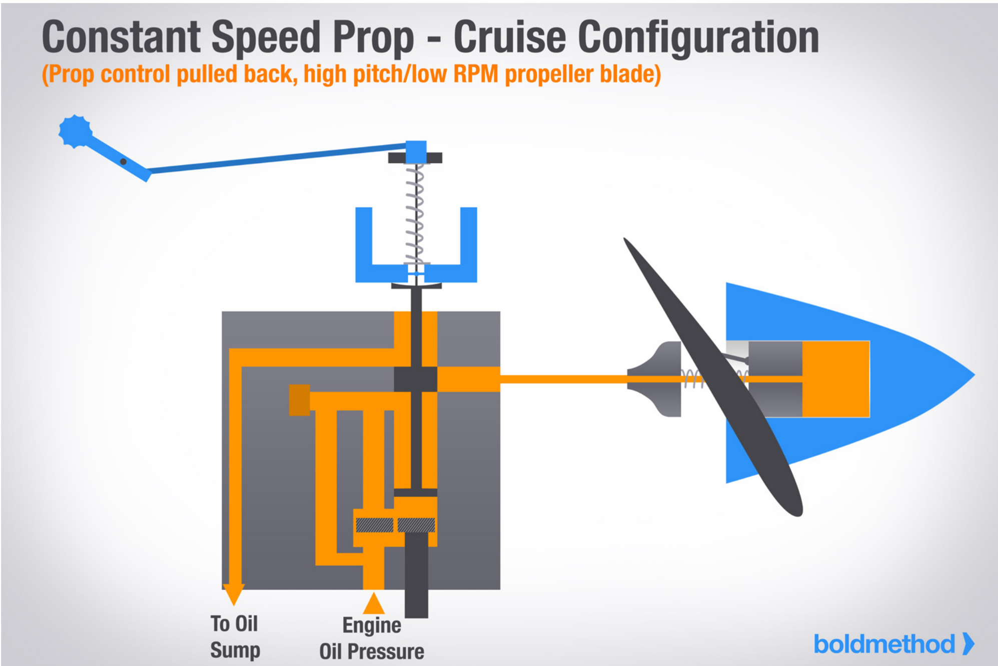


Figure 2.6: VPP Hydraulic Method - Cruise [27]

Centrifugal Method

In the centrifugal systems, centrifugal weights are 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. 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.

As disadvantages, centrifugal systems may provide less precise pitch control than more advanced hydraulic or electronic systems. The response time of centrifugal systems may be slower compared to more sophisticated systems [23].

Electromechanical Method

These systems involve electric motors and mechanical linkages to control the pitch of the propeller blades [26]. Figures 2.4 and 2.7 and this *VPP Electromechanical Method Video* illustrates the components and functionality of this method.

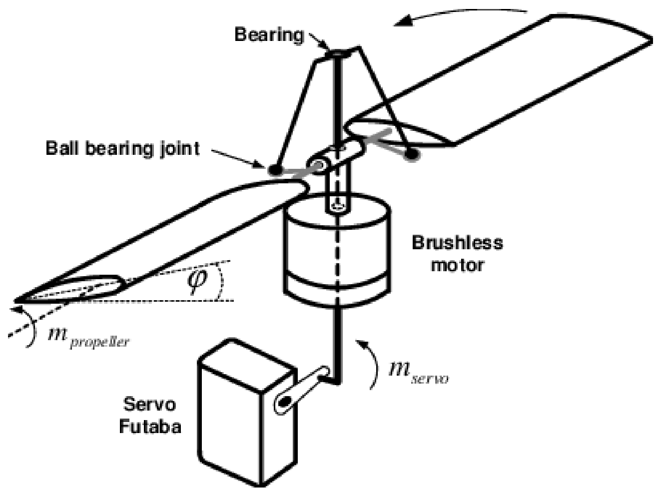


Figure 2.7: VPP Electromechanical Method [28]

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. They often have fewer components prone to wear and can be more straightforward [26].

However 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 [23], [26].

Chapter 3

Proposed Approach

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.

3.1 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.

Thus, 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.

3.2 System Architecture

As it is possible to see, in the proposed implementation of the System Architecture diagram (figure 3.1), the system will be composed of two subsystems: the Main Device Unit (MDU) and (multiple) Secondary Device Unit (SDU).

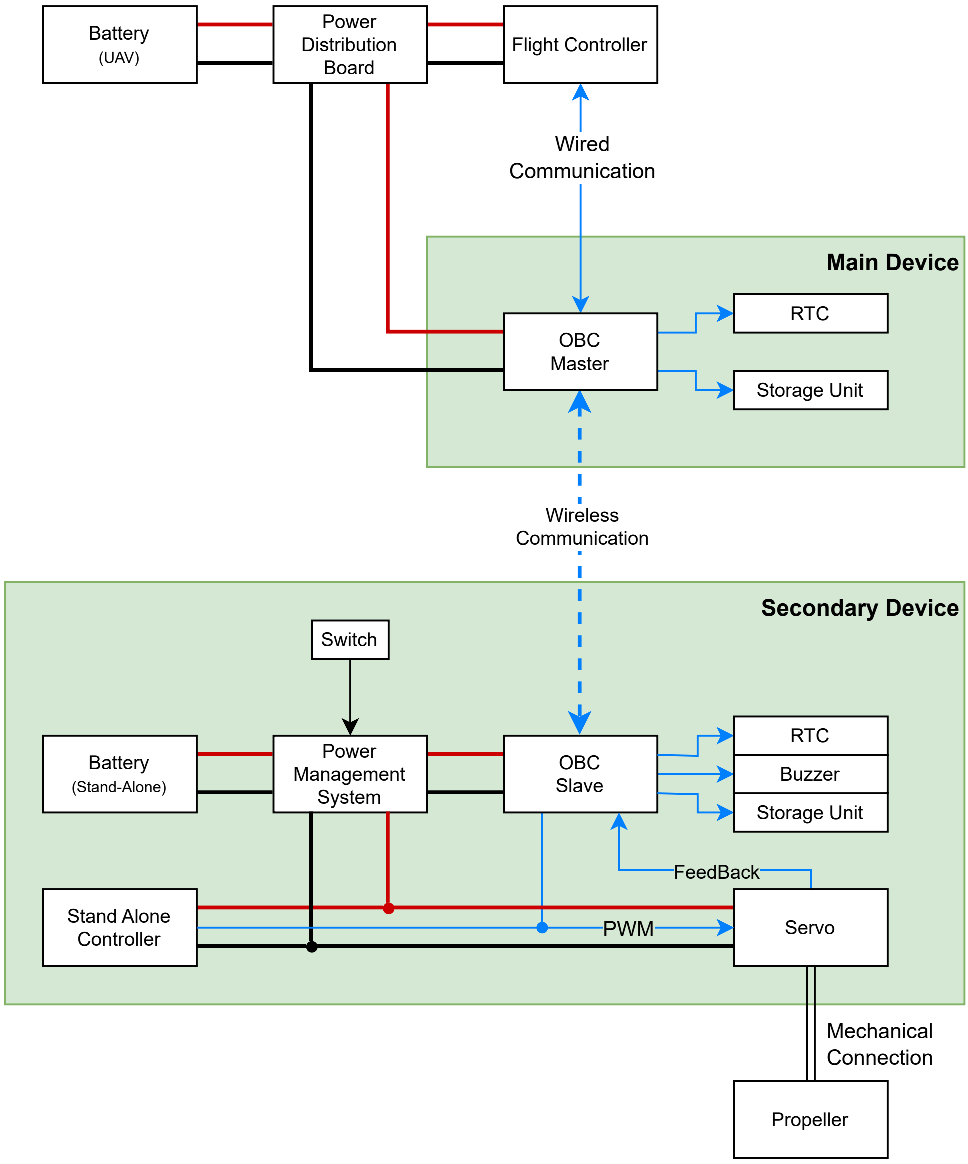


Figure 3.1: Proposed System Architecture High-Level Diagram

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.

3.2.1 Main Device

This subsystem will be, mainly, composed by an On Board Computer (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.

3.2.2 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.

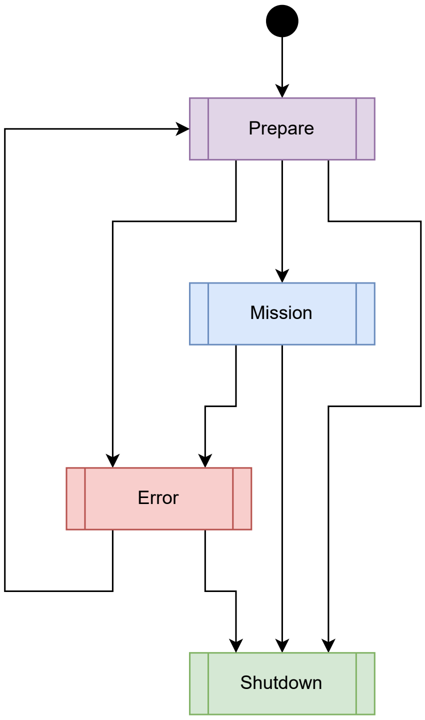
This action will transform the UAV system into a fixed-pitch proprotor but will help avoid having a system with a single point of failure that can cause the UAV to crash and possibly hurt people.

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.

3.3 System Behavior

In order to properly design the system behavior it was developed a system flow chart. The flow chart, represented in algorithm 3.1, describes the expected high-level behavior of both MDU and SDU.

Algorithm 3.1 Proposed System Behavior - High-Level Flow Chart



In this flow chart, there are four main tasks:

- Prepare
- Mission
- Error
- Shutdown

3.3.1 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.

3.3.2 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.

Chapter 4

System Development

This chapter addresses the most relevant technologies to be considered in this Thesis, according to its objectives. It highlights different battery technologies, types of external memories and storage units, computing systems, firmware strategies and wireless communication technologies.

4.1 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, [29]. Different battery chemistries offer varying energy densities, voltages, sizes, weights, cycle lives and costs.

Lithium-ion (Li-ion) and Lithium Polymer (Li-Po) batteries, shown in figures 4.1 and 4.2, can offer high energy density, rechargeability and moderate costs that make them suitable for portable devices, [30].

These batteries stand out as a prevalent choice for embedded systems.

Figure 4.1: Li-ion battery example [31]

Figure 4.2: Li-Po battery example [32]

NickelMetal Hydride (NiMH) and Nickel-Cadmium (NiCd) batteries (figures 4.3 and 4.4 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 4.3: NiMH battery example [33]

Figure 4.4: NiCd battery example [34]

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, [29] as it is possible to see in figure 4.5.

Figure 4.5: Lead-Acid battery example [35]

Alkaline batteries, in figure 4.6, are cost-effective but most of them are non-rechargeable, have a standard cylindrical format and have moderate energy density [29].

Figure 4.6: Alkaline battery example [36]

4.1.1 Battery Technologies Comparison

In table 4.1 it is possible to see the resume and comparison between the battery technologies examples described previously.

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 4.1: Comparison of Battery Technologies

In this Thesis, the energy density of the battery will be a key characteristic when choosing a battery module. To avoid adding too much weight to the UAV and to be able to integrate this solution easily the battery should be small but with high capacity.

4.2 External Memory and Storage Units

Memory units can be classified as volatile memory and as non-volatile memory [37].

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 [38].

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 [38]. 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 [37].

Since non-volatile storage can keep the data stored even when they are not powered, these storages are very common in embedded systems [39].

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 [40], [41]. 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 [38], [40].

Ferroelectric Random Access Memory (FRAM) combines the benefits of RAM and EEP-ROM 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 [38], [40].

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 [40].

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, [42].

4.2.1 External Memory Technologies Comparison

Table 4.2 resumes and 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**	**Ease of Access**
SD Cards	Moderate	Moderate	Long-term	High	Moderate	Easy
EEPROM	Fast	Moderate	Long-term	Low	Low	Difficult
FRAM	Very Fast	Relatively Higher	Long-term	Low	Very Low	Difficult
eMMC	Fast	Moderate	Long-term	High	Moderate	Easy
SSD	Very Fast	High	Long-term	Very High	High	Easy
HDD	Moderate	High	Long-term	Very High	High	Easy

Table 4.2: Comparison of External Memory Technologies

The external memory, or storage unit, will play an important role in this system. The ability to store system logs and other types of data will not only help to debug during tests but it will also help to evaluate the system performance.

Thus for this Thesis, the external memory should be able to retain data for long periods, have a high capacity, have a moderate to low power consumption and have easy access to the stored data.

4.3 Computing Systems and Firmware

4.3.1 Computing Systems

An OBC is a device capable of managing and/or controlling various functions such as:

- 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, mainly, contain a processor core, memory, and programmable input/output peripherals. Since they are compact and have low power consumption, they can be designed for specific tasks which makes them suitable for embedded systems. They often include integrated peripherals like timers, communication interfaces (for example Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), Controller Area Network (CAN) and Universal Asynchronous Receiver-Transmitter (UART)), Analog Digital Converter (ADC) and RTC [43].

Microcontrollers are programmable using low-level programming languages like C, C++ and Assembly [43].

However, microcontrollers have more limited processing power and are less flexible for general-purpose computing [43] and [44].

Microprocessors 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. They can also be programmed by low-level or high-level programming languages which facilitates the development of firmware [45].

However, microprocessors have higher power consumption, may require additional components for specific applications, and have a larger form factor compared to microcontrollers [45].

FPGAs are integrated circuits that can be configured after manufacturing, allowing for custom circuits. The FPGA architecture is composed of configurable logic blocks that a developer can program making them highly customizable for specific applications. It has parallel processing capabilities and can be reprogrammed for different tasks [43], [46] and [45].

But, they have a higher cost and higher power consumption (compared to microcontrollers). Since programming FPGA is done by programming the hardware (with Hardware Description Languages (HDL)), the method is different from programming a microcontroller (with specific software) the learning curve can be steeper [43] [46].

Computing System Technologies Comparison

The table 4.3 compares the programming languages, flexibility, processing power, development complexity, cost, real-time performance and power consumption of these three main types of control units [45].

Topic	Microcontrollers	Microprocessors	FPGAs
Programming Languages	Low-level languages	High-level languages	HDL
Flexibility	Low	Moderate	High
Processing Power	Limited	High	High
Development Complexity	Low	Moderate	High
Cost Considerations	Low	Moderate	High
Real-time Performance	Moderate	Low	High
Power Consumption	Low	Moderate to high	Moderate

Table 4.3: Comparison of Microcontrollers, Microprocessors, and FPGAs.

When choosing the OBC for this Thesis, the first characteristic to be analyzed should be the availability to be programmed easily (being by a low-level language, an high-level language or a Real-Time Operating System (RTOS)). This selection will dictate the firmware strategy.

Other characteristics must also be present like power consumption, available interfaces (both wired and wireless) and real-time performance.

4.3.2 Firmware

In embedded systems, the choice of programming languages and the use of a RTOS in firmware development are critical decisions that can impact the performance, efficiency, and complexity of the embedded system.

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 [47], [48].

C programming language has low-level features and is close to the hardware making it efficient and with high performance [49], [48].

It provides fine-grained control over memory, leading to efficient memory usage but can also lead to potential errors if not handled carefully [50]. Due to its low-level control and predictable performance, it is often used in real-time systems [51].

This programming language is generally portable and has a large and active community, with extensive support and numerous libraries, development tools and compilers available [51].

In terms of safety and reliability, it is a powerful language but lacks some safety features, like in memory management for example [51].

C can have a steeper learning curve, especially for beginners, due to manual memory management and low-level constructs [49].

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 [51].

Assembly can be used in real-time systems due to its precise control over hardware, predictable performance and high efficiency [50].

However, in Assembly, there is no safety net or restrictions, this way developers must handle all aspects of safety and reliability manually [51]. 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 [50].

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 [49], [48]. 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 [51].

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 [50].

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 [51].

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 [49],[51].

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 [49].

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 [51].

MicroPython is a compact extension of Python designed for microcontrollers and Internet of Things (IoT) devices to emphasize efficiency. However, it sacrifices some features of the standard Python to fit within resource constraints [49].

It aims for a small memory footprint suitable for microcontrollers which enhances portability as well [51].

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 [49], [50].

Table 4.4 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 4.4: Comparison of Programming Languages

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 [52].
The following RTOS examples are open-source, well-documented, compact, and designed for resource-constrained systems, and they support various microcontroller architectures [53]. They also support microROS, an extension of the Robot Operating System (ROS), designed for microcontrollers and embedded systems that can be resource-constrained.

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 [54].

As for scheduling policies, it supports priority-based, round-robin and rate monotonic schedulers and has semaphore/mutex management [55]. It supports I2C, SPI and UART wired protocols and Bluetooth Low Energy (BLE)-Stack, Transport Layer Security (TLS), Ethernet and Wireless Fidelity (Wifi) network protocols [55].

It is worth mentioning that FreeRTOS has Software Development Process DO178B Level A and Functional Safety IEC-61508 certifications.

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 [56].

NuttX can be scalable, providing a balance between a small footprint for resource-constrained devices and support for larger systems [56].

In terms of scheduling it supports priority-based (First In First Out (FIFO)), Round-Robin and Sporadic Server schedulers and has semaphore/mutex management For wired protocols, it has support over I2C, SPI, Universal Serial Bus (USB), CAN and Modbus. And for network protocols, NuttX, supports 6LoWPAN, Ethernet, Wifi and Radio Frequency Identification (RFID) [55].

Zephyr is a real-time operating system for resource-constrained devices. It also uses Apache 2.0 license that allows open-source and commercial development [57].

Designed to be modular and can be configured to match the requirements of the target device. Zephyr is supported by the Linux Foundation and has a growing and active community [57] and [53].

As for scheduling policies, it supports priority-based and rate monotonic schedulers and has semaphore/mutex management [55]. It supports I2C, SPI, USB, CAN and UART wired protocols and BLE-Stack, TLS, 6LoWPAN, Ethernet, Near Field Communication (NFC), Wifi and RFID network protocols [55] [55].

In table 4.5 is possible to see a comparison between some RTOSs examples [55].

Table 4.5: Comparison of Real-Time Operating Systems

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

Overall Analyses

In this subsection, it was made a resume and overall analyses of programming languages and RTOSs, describing the pros and cons of each solution.

Programming languages can provide better portability across different hardware platforms. They can have modular code and/or libraries making them more reusable. Since programming languages are more common, the time dedicated to development is smaller and more efficient. In the case of high-level languages productivity is increased since this languages abstracts some hardware details. But, programming languages may introduce performance overhead which can be critical in resource-constrained embedded systems. And may have less control over low-level hardware details, which could be essential for certain embedded applications.

RTOSs provides deterministic timing and has task scheduling and management that can enable the execution of multiple tasks simultaneously. And can manage resources more effectively, optimizing performance and memory usage. However, integrating a RTOSs can be complex and more time-consuming since the developers may need to invest time to learn and understand the specific RTOSs.

4.4 Wireless Communication

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 communication between devices inside an UAVs, the focus will be on short-range wireless communication protocols [58], [59], [60].

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 and Long Range Wide Area Network (LoRaWAN) 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 [59], [60].

Wifi, with 802.11 series Institute of Electrical and Electronics Engineers (IEEE) standard, is a communication technology that enables devices to connect to the same network without the need for physical cables [61], [62] and [63]. It can be used for bi-directional high-speed data transfer (ranging from Mbps to Gbps) over short ranges, using 2.4 GHz and 5 GHz frequency bands [61], [62] and [63]. Typically, its topology is star[1] or mesh[2] format [61].

However, it has high power consumption and has interference in 2.4 GHz and 5 GHz bands [61].

This technology is normally used in homes, offices, public spaces, and industrial settings.

Bluetooth, based on IEEE 802.5.1 standard, is a common short-range wireless technology with low power consumption [61], [62] and [63]. It has bi-directional data transfer operating within the 2.4 GHz band [61], [62] and [63]. Bluetooth topology is, normally, point-to-point[3] or mesh [61].

But has a limited range and lower data rates (in the Mbps range) [61].

It is commonly used in personal devices, audio accessories and IoT applications.

In Bluetooth version 4.0, was introduced BLE standard providing even less power consumption and profiles and services that define the functions and characteristics of devices [61].

Zigbee, with IEEE 802.15.4 standard, is a wireless communication standard designed for short-range communication with low power consumption [61], [62] and [63]. Working in 2.4 GHz frequency band, it has bi-directional data transfer and low latency [61], [62] and [63]. Zigbee uses mesh network topology [61].

However, the limited data rate (in the Kbps range) and the limited range can be a compromise to the system, [62] and [63].

Zigbee is normally used in home automation, industrial control and sensor networks.

LoRaWAN[4] is designed for long-range communication but can also be used in short-range. It provides low-power, long-range wireless communication suitable for various scenarios and applications [61].

The bi-directional communication, resilient to interference, works in frequency ranges of 400 MHz, 868 MHz and 900 MHz. And uses Star-of-Stars[5] [64] and mesh topologies [65].

But has low data rates in the Kbps range [61].

Normally it is used in IoT applications, smart agriculture and smart cities.

[1]devices are connected to a central hub, with all communication flowing through this central point
[2]devices are interconnected, allowing for multiple communication paths between nodes
[3]device connects with one or more slave devices (point-to-multipoint)
[4]protocol for the MAC layer and network layer being LoRa the physical layer technology
[5]multiple star networks are interconnected through a central hub

4.4.1 Wireless Communication Technologies Comparison

The comparison of the wireless communication technologies explained, in this section, can be seen in Table 4.6 [62] and [63].

Technology	**Wi-Fi**	**Bluetooth**	**Zigbee**	**LoRaWAN**
Data Rate	High	Moderate	Moderate	Low
Widespread	Yes	Yes	Yes	Limited
Direction	Bi-Directional	Bi-Directional	Bi-Directional	Bi-Directional
Power Consumption	High	Low	Low	Low
Frequency	2.4 GHz	2.4 GHz	2.4 GHz	Sub-1 GHz
Range (in meters)	30-100	5-100	10-100	2000-10000
Topology	Star, Mesh	Star, Mesh, Piconet	Mesh	Star-of-Stars, Mesh

Table 4.6: Comparison of Communication Technologies

The wireless technology to be used in this system should be in a different band than the network already installed on the UAV. This is important since this system can not interfere with the normal function during a mission.

It should also support multiple devices, have a range bigger than the size of the UAV and have low power consumption.

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 chosen 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.

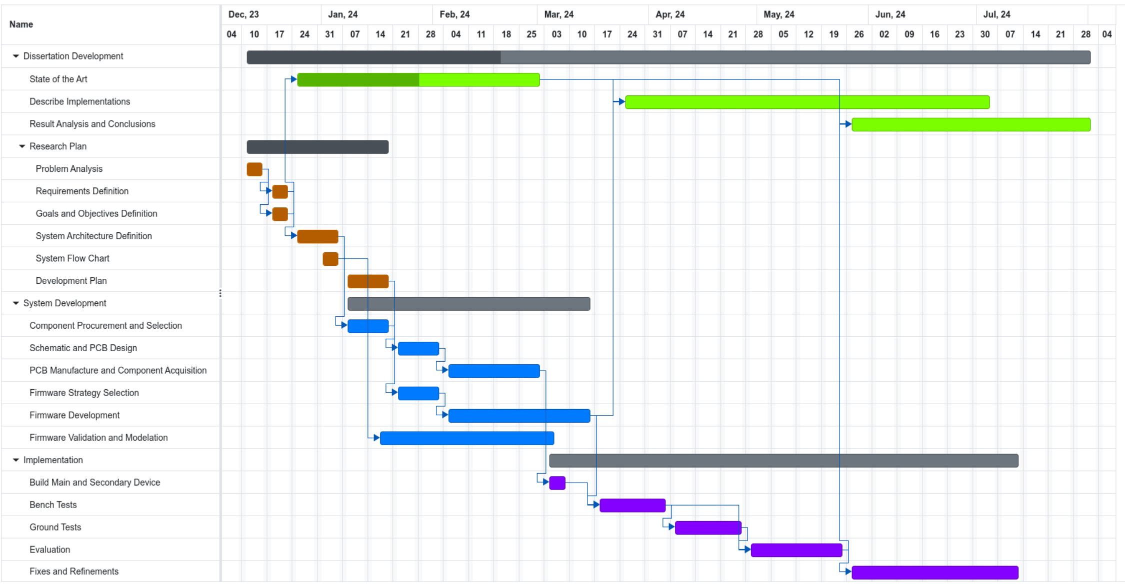


Figure 5.1: Project timeline Gantt chart

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.

5.4 Final Remarks

Even if detailed planning has been defined, unforeseen events may occur that delay the development of this Thesis. The manufacture of the PCBs and the purchase of the components are examples where the delay in the arrival of the hardware will postpone the start of initial tests.

Even so, designing the development plan helped structure the main tasks needed to finish the development of this Thesis. The comprehensive analysis and literature review, done in the Dissertation Development phase, will set the foundation for defining system requirements, outlining objectives, and designing the system architecture. This will be helpful during the selection of hardware components and firmware strategies, done in the System Development phase.

Both these phases lay the groundwork for the subsequent implementation of the developed solution, where the main prototype will be built and tested. Through these evaluations, the developed system will be assessed against the goals and requirements described in previous chapters.

By analyzing the results it will be possible to determine if necessary refinements are needed.

Bibliography

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

[2] en. url: https://www.researchdive.com/8348/unmanned-aerial-vehicle-uav-drones-market.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Abhishek Paliwal. *The Global UAV Drones Market Growth is highly anticipated due to its Wide Scale Industrial Applications Likewise in Oil and Gas, Sea Ports, etc*. Eng. url: https://www.researchdive.com/press-release/the-global-uav-drone-market.html.

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

[9] 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.

[10] M. Yasir Amir and Valiuddin Abbass. “Modeling of Quadrotor Helicopter Dynamics”. In: *2008 International Conference on Smart Manufacturing Application*. Apr. 2008, pp. 100–105. doi: 10.1109/ICSMA.2008.4505621. url: https://ieeexplore.ieee.org/document/4505621.

[11] Bora Erginer and Erdinc Altug. “Modeling and PD Control of a Quadrotor VTOL Vehicle”. In: *2007 IEEE Intelligent Vehicles Symposium*. June 2007, pp. 894–899. doi: 10.1109/IVS.2007.4290230. url: https://ieeexplore.ieee.org/document/4290230.

[12] M. Alpen, K. Frick, and J. Horn. “Nonlinear modeling and position control of an industrial quadrotor with on-board attitude control”. In: *2009 IEEE International Conference on Control and Automation*. Dec. 2009, pp. 2329–2334. doi: 10.1109/ICCA.2009.5410536. url: https://ieeexplore.ieee.org/document/5410536.

[13] Jinhyun Kim, Min-Sung Kang, and Sangdeok Park. “Accurate Modeling and Robust Hovering Control for a Quadrotor VTOL Aircraft”. In: *Journal of Intelligent and*

Robotic Systems 57 (Jan. 2010), pp. 9–26. issn: 978-90-481-8763-8. doi: 10.1007/978-90-481-8764-5_2.

[14] Barnaby Wainfan. *Fixed Pitch vs. Constant Speed*. en-US. May 2016. url: https://www.kitplanes.com/wind-tunnel-56/.

[15] Robin May. *Variable Pitch Propellers*. English. Dec. 2004. url: http://www.fly13.co.uk/Tug/VP%20Prop/Prop.htm.

[16] Syed Agha Hassnain Mohsan et al. "Role of Drone Technology Helping in Alleviating the COVID-19 Pandemic". In: *Micromachines* 13 (Sept. 2022), p. 1593. doi: 10.3390/mi13101593.

[17] Yousef Alghamdi, Arslan Munir, and Hung La. "Architecture, Classification, and Applications of Contemporary Unmanned Aerial Vehicles". In: *IEEE Consumer Electronics Magazine* PP (Mar. 2021), pp. 1–1. doi: 10.1109/MCE.2021.3063945.

[18] Marcos Filipe Andrade Nunes Rosa. "Preliminary Design of an eVTOL Aircraft". eng. Accepted: 2023-02-20T09:26:43Z. masterThesis. July 2022. url: https://ubibliorum.ubi.pt/handle/10400.6/13031.

[19] url: https://skybrary.aero/articles/p-factor.

[20] url: https://www.thaitechnics.com/propeller/prop_intro.html.

[21] url: https://skybrary.aero/articles/fixed-pitch-propeller.

[22] Tony Bingelis. *The Fixed Pitch Propeller Dilemma*. en. Aug. 1986. url: https://www.eaa.org/eaa/aircraft-building/builderresources/while-youre-building/building-articles/propellers-and-spinners/the-fixed-pitch-propeller-dilemma.

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

[24] 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.

[25] Brandyn Phillips, Vikram Hrishikeshavan, and Inderjit Chopra. "Enhancing the Performance of a Quadrotor Biplane Tail-sitter (QBiT)". In: May 2018.

[26] Sean Walsh. *Types of Aircraft Propellers: Exploring Fixed Pitch, Variable Pitch, and Constant Speed*. en-US. June 2023. url: https://pilotpassion.com/types-of-aircraft-propellers/.

[27] Colin Cutler. *How A Constant Speed Propeller Works*. Dec. 2017. url: https://www.boldmethod.com/learn-to-fly/aircraft-systems/how-a-constant-speed-prop-works/.

[28] Jose Guerrero et al. "Robust Control Design based on Sliding Mode Control for Hover Flight of a Mini Tail-Sitter Unmmanned Aerial Vehicle". In: Dec. 2009, pp. 2342–2347. doi: 10.1109/IECON.2009.5415267.

[29] 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/.

[30] en-US. url: https://www.toradex.com/blog/using-lithium-ion-batteries-in-embedded-systems.

[31] url: https://www.nkon.nl/pt/samsung-inr-18650-30q-3000mah.html.

[32] en. url: https://thepihut.com/products/1200mah-3-7v-lipo-battery.

[33] url: https://www.nkon.nl/pt/rechargeable/nimh/gp-ni-mh-d-11000mah.html.

[34] admin. *NiCd Battery Charger Circuit*. en-US. July 2019. url: https://theorycircuit.com/nicd-battery-charger-circuit/.

[35] url: https://www.nkon.nl/pt/rechargeable/lead-acid-batteries/6v-vrla/yuasa-6v-1-2ah-loodaccu.html.

[36] url: https://www.nkon.nl/pt/c-duracell-industrial-1-5v.html.

[37] 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.

[38] 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.

[39] 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.

[40] en-pt. url: https://www.digikey.pt/en/articles/the-fundamentals-of-embedded-memory.

[41] en. Jan. 2021. url: https://www.kingston.com/en/blog/personal-storage/microsd-sd-memory-card-guide.

[42] en. Dec. 2023. url: https://www.kingston.com/en/blog/pc-performance/hdd-vs-external-ssd.

[43] Abiola Ayodele. *FPGA vs. Microcontroller: Understanding the Key Differences*. url: https://www.wevolver.com/article/fpga-vs-microcontroller-understanding-the-key-differences.

[44] Tyler Ross Lambert. "An introduction to microcontrollers and embedded systems". en. In: ().

[45] en. Feb. 2018. url: https://community.element14.com/technologies/fpga-group/b/blog/posts/comparing-an-fpga-to-a-microcontroller-microprocessor-or-an-asic.

[46] Karen Parnell and Roger Bryner. "Comparing and Contrasting FPGA and Microprocessor System Design and Development". en. In: (2004).

[47] 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.

[48] 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.

[49] 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.

[50] 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/.

[51] en. Aug. 2023. url: https://www.linkedin.com/pulse/5-best-embedded-systems-programming-languages-pros.

[52] S Ramanarayana Reddy. "Selection of RTOS for an Efficient Design of Embedded Systems". en. In: (2006).

[53] 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.

[54] 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/.

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

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

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

[58] 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.

[59] 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.

[60] 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.

[61] Fiona Kuan. *Short range wireless communication technology vs long range wireless communication technology*. en-US. Sept. 2022. url: https://www.mokosmart.com/short-range-wireless-communication-technology-vs-long-range-wireless-communication-technology/.

[62] M. Deseada Gutierrez Pascual. "Indoor Location Systems based on ZigBee networks". PhD thesis. May 2012. doi: 10.13140/2.1.4300.1126.

[63] Muhammad Khan, Ijaz Qureshi, and Fahim Khanzada. "A Hybrid Communication Scheme for Efficient and Low-Cost Deployment of Future Flying Ad-Hoc Network (FANET)". In: 3 (Feb. 2019), p. 22. doi: 10.3390/drones3010016.

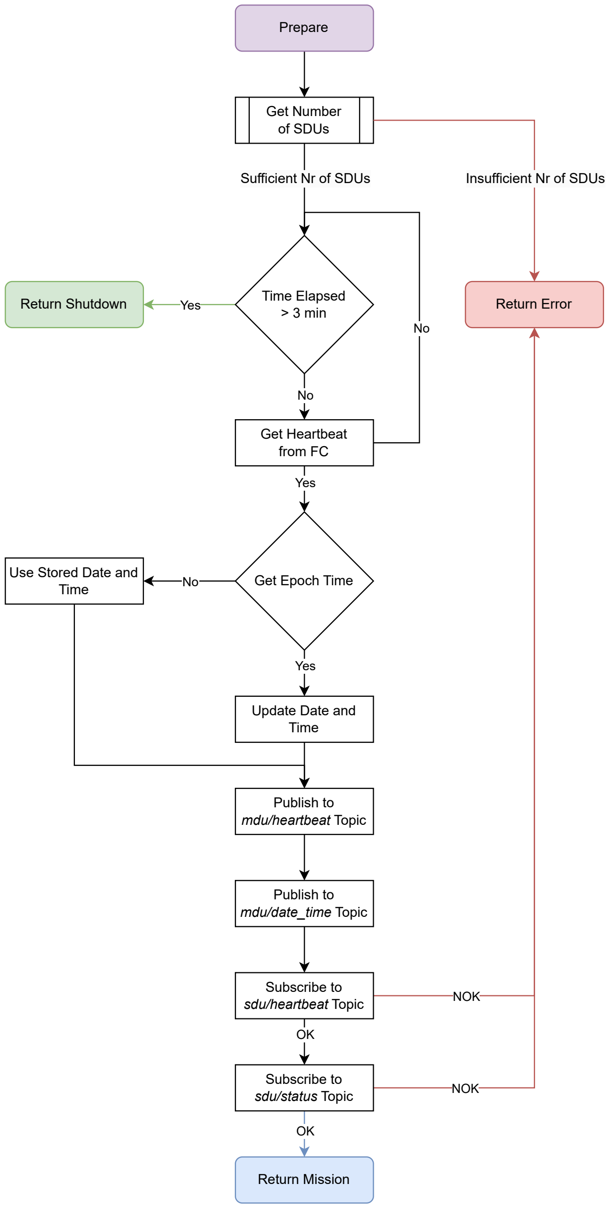
[64] en-us. url: https://www.thethingsnetwork.org/docs/lorawan/architecture/.

[65] Jeferson Rodrigues Cotrim and João Henrique Kleinschmidt. "LoRaWAN Mesh Networks: A Review and Classification of Multihop Communication". In: *Sensors (Basel, Switzerland)* 20.15 (July 2020), p. 4273. issn: 1424-8220. doi: 10.3390/s20154273.

Appendix A

Main Device Flow Chart - Prepare Task

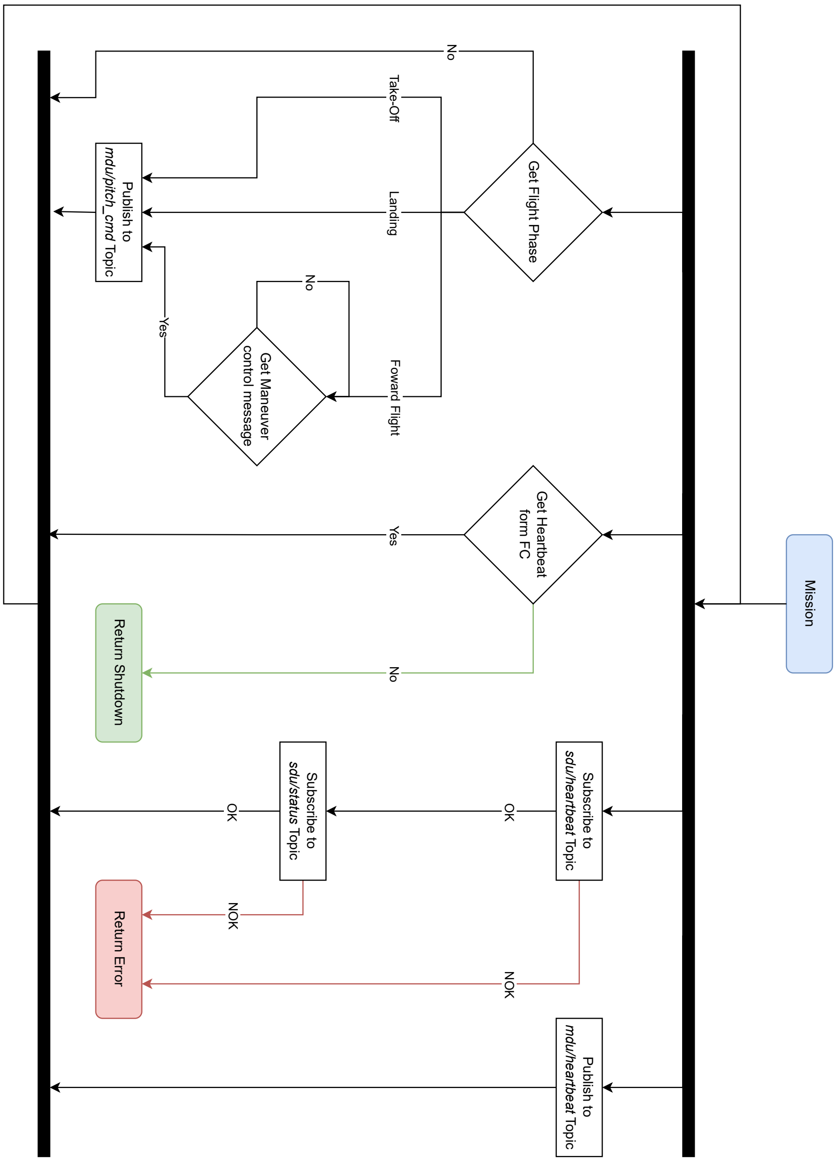
Algorithm A.1 Proposed System Behavior - Prepare Task Flow Chart (MDU)



Appendix B

Main Device Flow Chart - Mission Task

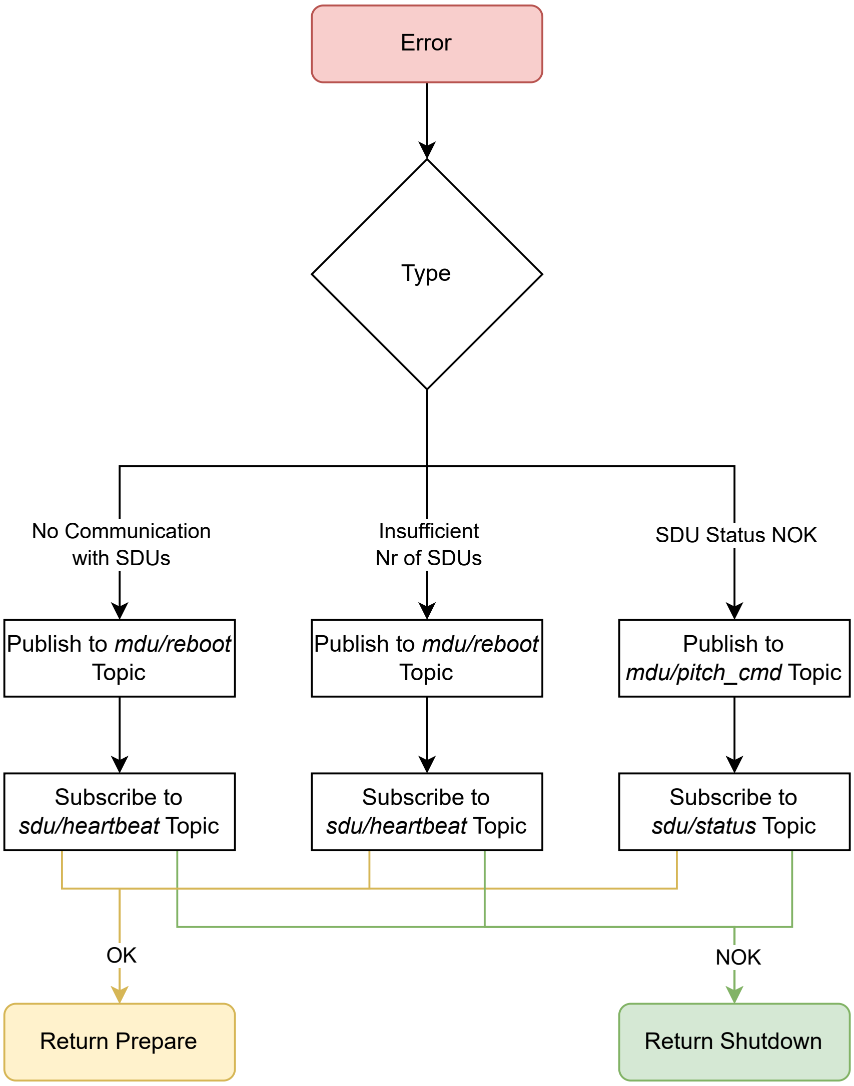
Algorithm B.1 Proposed System Behavior - Mission Task Flow Chart (MDU)



Appendix C

Main Device Flow Chart - Error Task

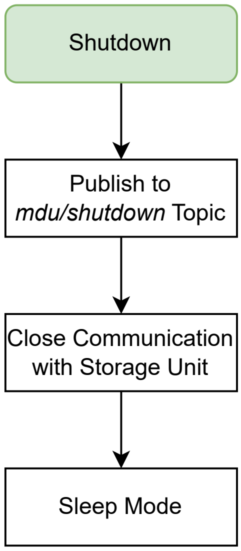
Algorithm C.1 Proposed System Behavior - Error Task Flow Chart (MDU)



Appendix D

Main Device Flow Chart - Shutdown Task

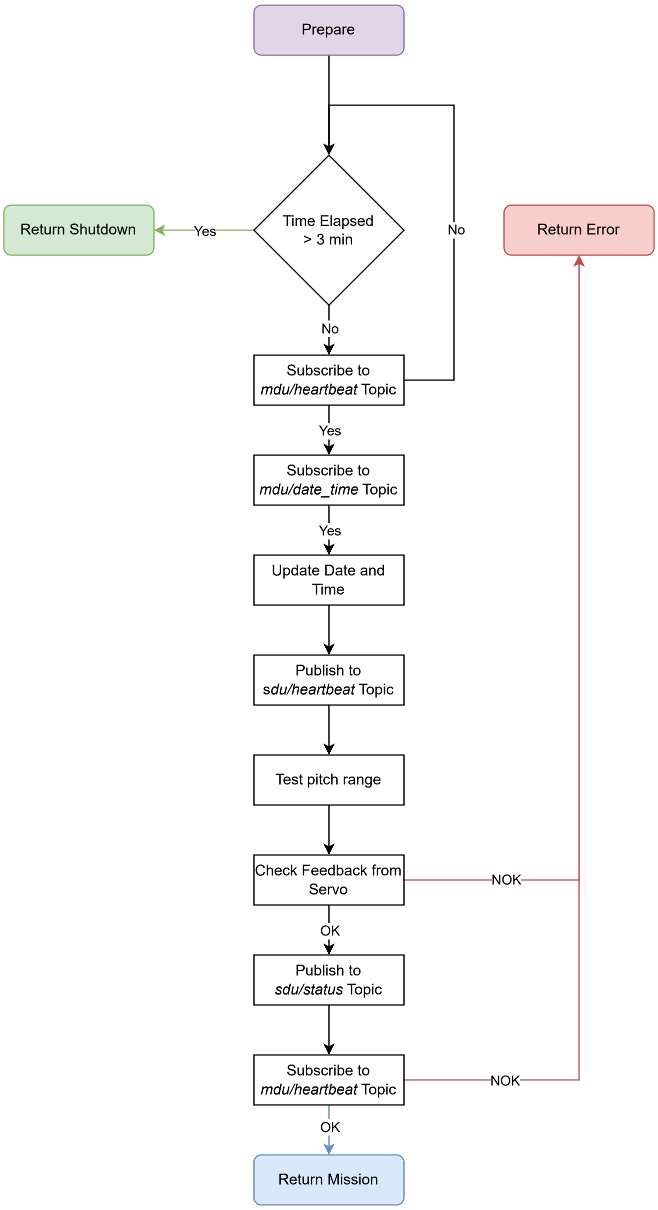
Algorithm D.1 Proposed System Behavior - Shutdown Task Flow Chart (MDU)



Appendix E

Secondary Devices Flow Chart - Prepare Task

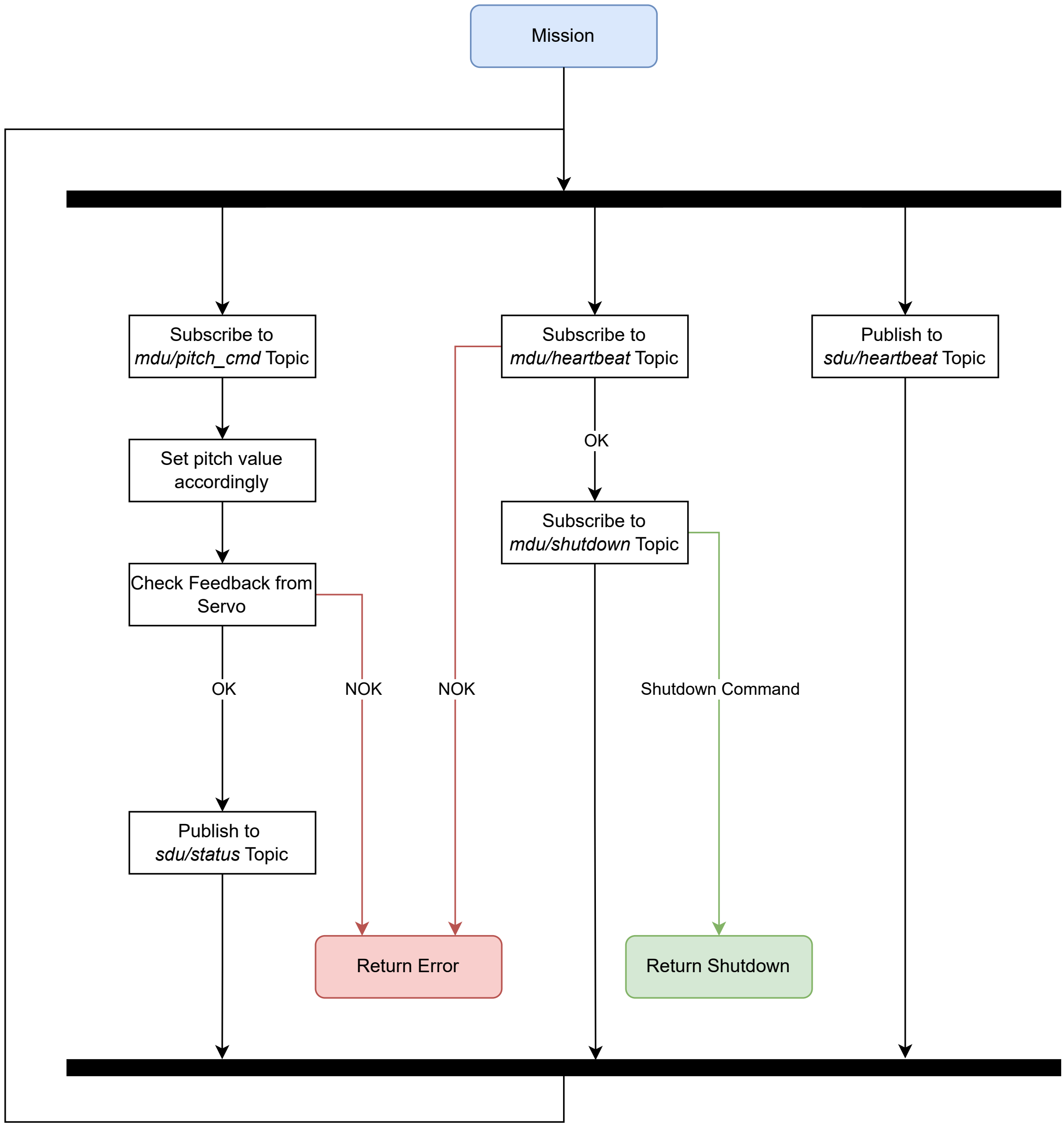
Algorithm E.1 Proposed System Behavior - Prepare Task Flow Chart (SDU)



Appendix F

Secondary Devices Flow Chart - Mission Task

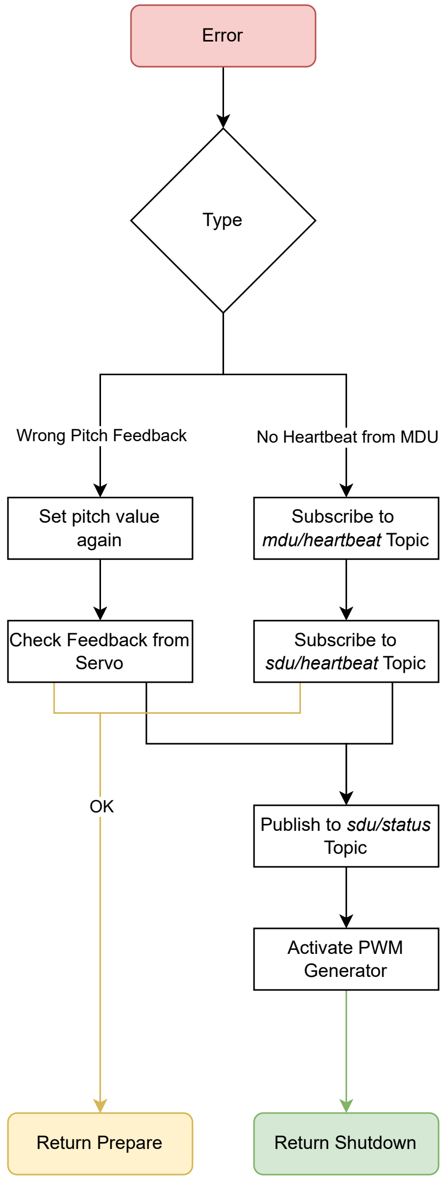
Algorithm F.1 Proposed System Behavior - Mission Task Flow Chart (SDU)



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)