Assessment Task 2:
Progress Report

HIVE AURC AVIONICS (Data Analytics and Verification)

RMIT University
Semester 1 – 2024

Supervisor: Dr. Glenn Matthews

Name	Student No.	Contribution
Matthew Ricci	s3785111	50%
Jeremy Timotius	s3779178	50%

Contents **1**

List of Equations

Summary

RMIT's high velocity rocket team (HIVE) this year is participating in the Australian Univercity Rocket Competition (AURC) under the 10,000ft category. As part of this competition HIVE proposed the completion of five rockets, given the name Aurora, with an iterative design approach working towards the fifth and final rocket that will fly as the competition entry vehicle. The Data Analytics and Verification team, a division of the avionics subsystem, was assembled for the purpose of producing the tools and implementation of both real-time and offline data processing systems that will run as a part of the rocket's avionics, as well as designing the verification routines to ensure proper testing procedure.

Key Milestones

- Summary of Progress to Date (Key milestones reached)
- Key Insights and Preliminary Findings
- Challenges faced and Mitigation Strategies
- Outline of Future Work? (Next steps and or focus points).

Aurora I launch outcomes

The Aurora I rocket launched successfully on the 14th of April. Aurora I launched with a custom Avionics bay which included two commercial flight computers (Blue Raven) and a custom electronics system which contained an Arduino Nano, sensor suite and flash storage. From a Data analytics and verification perspective, the following outlines key outcomes and findings:

- Implementation: Custom firmware and libraries were created for the for Arduino Nano.
- Data collection: Most data was successfully collected and stored by the system, however there was no available barometric data post-flight.
- Flash storage retrieval: Data was extracted from the on-board flash storage upon recovery.
- Preliminary data validation: Initial comparison of sensor data with the Blue Raven recordings indicated general agreement in acceleration, velocity, and attitude trends.

Aurora II launch outcomes

Aurora II launched a week prior to submission of this report, on Sunday the 26th of May. The rocket built for this launch was for the most part unchanged from Aurora I, with the original avionics system being re-flown with adjustments to ensure correct collection and storage of barometric data, in addition to ground communication hardware for preliminary telemetry testing from the Redundant Systems and Ground Communications team.

As of the date of submission, data collected by the avionics system during this flight has yet to be retrieved. Once the data is available it will be applied for verification of the developed analysis tools and processing algorithms that will be further discussed later in the report.

1 Introduction

1.1 Background and Significance

The Aurora V rocket, an L3 (Level 3) high-powered model rocket designed to reach an apogee of 10,000 feet, represents RMIT's entry in the 2024 Australian Universities Rocket Competition (AURC). To achieve a successful flight and meet the competition's requirements, real-time data collection, analysis, and verification is necessary. The Data Analytics and Verification team within the Avionics subsystem plays an important role in ensuring the rocket's performance, safety, and ultimately, the success of the mission.

Aurora V will be the fifth and final competition ready rocket as it represents the culmination of the Aurora project as a whole. Throughout the semesters, previous iterations of the

rocket (Aurora I to Aurora IV) will be developed and designed which will act a stepping stone towards the final design. With each iteration, the data gathered will feed into data analytics and verification processes in hopes to optimise algorithms and refine sensor configurations.

The RMIT AURC team is multidisciplinary made up of five major sub systems including, Aerostructures, Avionics, Payload, Recovery and Aerobrakes totaling a team of 30 students. The Data analytics team is subset of the Avionics sub system and will provide insights into the rocket's flight dynamics, enabling real-time decision-making for critical systems like the airbrakes and switching over to failover systems. By analysing sensor data such as inertial measurements, barometric pressure, and temperature and through sensor fusion and filtering, an estimate of the rocket's position, velocity, and attitude can be measured. This information is mission critical for maintaining a stable trajectory towards the target apogee. Additionally, data verification and state estimations will be used to identify and mitigate potential errors or anomalies in the sensor readings, ensuring the reliability of the information used for flight control.

The significance of this project extends beyond the immediate 2024 AURC competition as it will act as a resource for future RMIT model rockets. The development data analytics and verification techniques for resource constrained embedded systems has a broader application in the aerospace industry and other fields where data analysis is critical key. The work completed through this capstone will provide future recommendations contributing to future work.

1.2 Problem Statement

The primary challenge lies in designing and implementing firmware, data analytics and verification algorithms that can operate effectively within the limitations of the custom PCB board developed by the Aurora V's Avionics, Ground Communications and Redundant Systems team. This involves balancing the desire of captured sensor data and algorithms with the constraints of limited processing power, memory, and communication bandwidth. This subsystem's primary objective is to design, test, and validate data analytics processes for the avionics, ground communications, redundant, electronic systems, and payload systems.

1.2.1 Project Deliverables

Project deliverables and expected outcomes include:

- Data Capture and Logging Firmware Implementation: Defining sampling rates and intervals for data capture and establishing data logging and storing formats.
- Real-Time Processing: Research and develop algorithms for sensor fusion to integrate data from multiple sensors. Implement filtering techniques to remove noise. Provide accurate state estimation for the airbrake control system.
- Post-Processing: Perform statistical analysis to evaluate key performance metrics of avionics systems. Implement systems to detect deviations from expected behaviour when comparing raw sensor and fused data to the data captured by the blue raven.
- Data Visualization: Develop tools for generating informative graphs and charts for data interpretation. Explore the possibility of creating simulations based on collected data.
- Data Validation: Determine the validity of the captured sensor data.

1.2.2 Design/Research Questions

1.2.3 Project Scope

The Data analytics team is subset of the Avionics sub system and will operate within the project scope that encompasses the following activities:

- **Software Development**: Writing, testing, and optimizing firmware for data collection, logging, and real-time processing. Development post data visualization and analysis.

- **Algorithm Design**: Researching and implementing algorithms for sensor fusion, filtering (Kalman filters), and state estimation.
- **Hardware Integration**: Integrating sensors, microcontrollers, and communication modules into the avionics system.
- **Testing and Validation**: Conducting ground tests and flight tests (Aurora I-V) to validate the performance and reliability of the data analytics and verification system.
- **Iterative Refinement**: Continuously improving the system based on data and feedback from each launch, culminating in a competition-ready solution for Aurora V.

1.2.3.1 Limitations and Constraints

The project focuses on the software and algorithmic aspects of data analytics and verification. Hardware design, development and initialisation of firmware fall under the Avionics Ground Communications and Redundant Systems team.

The project timeline is subject to limitations and constraints, most notably the lead time required for procuring essential hardware components. This delay highlights the focus on software and algorithmic development while awaiting hardware delivery. During this waiting period, the team is using Nucleo-f439zi development board to prototype and test firmware and algorithms in a simulated environment. Additionally, sensor and integration verification tests are being designed to assess the performance and accuracy of firmware once the hardware becomes available. Furthermore, the team is actively analysing data captured during the Aurora I and II flights. This analysis aims to gain valuable insights into sensor behaviour, noise characteristics, and potential areas for algorithm optimisation. The findings from this analysis will directly inform the development of state estimation and Kalman filter implementations for Aurora III.

While real-time data transmission is a goal, the primary focus is on ensuring reliable data logging for post-flight analysis and iterative improvement. This project scope is deliberately flexible to accommodate the iterative nature of rocket development. As new challenges and opportunities arise with each launch, the scope may be adjusted accordingly to ensure the successful completion of the project.

2 Literature Review

During flight, the avionics subsystem is required to track various parameters of the rocket's state, both for the purposes of telemetry and for application in algorithms implemented by other onboard systems. In particular, it is imperative for the operation of the rocket's aerobrakes and recovery systems that accurate information concerning the upward velocity and altitude above ground are provided.

To determine these parameters, hardware sensors will be accessible for sampling both barometric and inertial data during flight. Importantly, accelerometers and gyroscopes provide information on the vehicle flight dynamics while a barometer samples pressure data that can be applied in calculation of vertical displacement. As the rocket's body axis during flight rotates off the global vertical, measurements from the accelerometer diverge from the true values. To account for this, attitude information that is calculated from the gyroscope is applied to rotate the data back to the global frame of reference.

2.1 State Estimation

A naive approach to state calculation would involve the numerical integration of sensor data to the required measurements. At a high level, MEMS (micro-electromechanical system) gyroscopes output axial data with units of degrees per second, and accelerometers provide acceleration measured in G's per second; Integrating these measurements, in theory, would result in the calculation of per-axis velocity and angular position. In reality, MEMS sensors are prone to drift from various noise sources [4, 11] that results in error offsets which propagate with the integral summation. In addition, the mathematical constructs and operations that are utilised

in these calculations can have a profound effect on the resulting output, particularly in the case of attitude determination from gyroscopic data.

Included amongst the most common representations of rotation in three dimensional space are Euler angles. These values are commonly referred to as *roll*, *pitch* and *yaw*, and provide a convenient mapping of rotation to three spacial dimensions that correlate to the physical world, allowing intuitive understanding of the resulting metrics. While pleasant for human parsing, Euler angles are susceptible to difficulties that limit their application in attitude determination. In particular, the issues of rotational singularity and the strictness of their sequential nature complicates their implementation in the necessary calculations [8, 5].

Alternatively, though complex in their mathematical structure, unit quaternions provide a more flexible representation of rotation that is not limited by these factors [5]. Furthermore, quaternion rotations are easily compatible with Euler angles and can be integrated just as easily, allowing for more accurate instantaneous rotational calculations. It is also documented that the Blue Raven altimeter, to which the Aurora avionics system in large part emulates, makes use of quaternions for its attitude estimation [1, 2]

Existing rocket designs such as Pioneer Rocketry's Skybreaker [7] and University of Calgary's Atlantis II [9] make use of Kalman Filter implementations when detecting apogee, fusing data both from barometric and inertial sensors.

2.2 Real-Time Operating Systems

The choice of an operating system (OS) for the avionics in Aurora V impacts system complexity, real-time performance, and resource utilisation. While bare-metal programming, where software directly interacts with hardware, offers simplicity and minimal overhead, it can become unmanageable as system complexity increases. As the number of processes increases, the harder it is to track timer and handle interrupts. This approach is suitable for resource constrained systems and has been employed in various amateur rocketry projects. Although Aurora I uses a bare-metal approach due to its simplistic hardware configuration, Aurora III and moving forward will introduce additional avionics processes such as communication between aerobrakes and payload and ground communications, outlined in section [].

The growing sophistication of model rocket avionics, as seen in projects like the Atlantis II, which employed FreeRTOS [9], has led to increased interest in Real-Time Operating Systems (RTOS) [3]. RTOS are specifically designed to handle real-time applications that require predictable timing behavior. An RTOS, such as FreeRTOS or VxWorks, provides features like prioritised thread scheduling, real-time clocks, and inter-process communication mechanisms that facilitate the development of reliable and responsive avionics software [6]. For instance, the use of RTOS in embedded control systems like quadrotor flight controllers has demonstrated significant improvements in managing event latencies [3]. RTOS can abstract much of the complexity involved in timing and resource management, which allows developers to focus on application-specific logic. Additionally, the use of RTOS in the Atlantis II project highlights the need for a more structured software architecture as avionics systems incorporate multiple sensors, actuators, and communication interfaces.

A Hybrid approaches may be considered which combine the benefits of both bare-metal and RTOS systems by integrating custom RTOS functionalities directly into hardware. This method reduces the overhead typically associated with full-blown RTOS, while still providing necessary real-time capabilities [3]. Such approaches have been shown to enhance performance by reducing interrupt lock times and memory usage, which is particularly beneficial in resource constrained environments such as the Aurora V.

3 Methodology and Engineering Design

This section primarily outlines the engineering designs for Aurora I and Aurora III, detailing the progression of the avionics system across these developmental stages. Aurora II, a re-flight of Aurora I, utilised the same avionics board to provide further data for analysis and testing of derived algorithms, with no major changes made to its design. As these designs

are implemented, testing and verification of hardware is to take place. This encompasses both individual component testing and sub system integration testing, which is outlined in Section 4.4.

3.1 Aurora I

Due to delays in hardware availability from vendors, the Aurora I avionics system resorted to "bare-bones" design, incorporating only essential components that were readily available. Although not ideal to the original Aurora I plan, the outcome still enabled the team to test fundamental functionalities including sensor data capture and writing data to flash storage. While limited in scope, the Aurora I design served as a stepping stone for subsequent iterations, providing initial data for post flight analysis.

3.1.1 Avionics Design Overview

The Aurora I avionics system served as the initial prototype for data collection and recovery system testing. It consisted of two main components:

- Commercial Off-The-Shelf (COTS) Blue Raven Flight Computers: These flight computers were responsible for activating the recovery systems (drogue and main parachutes) and recording flight data for post-flight analysis.
- Custom Electronics Board: This board, built on a perfboard, housed an Arduino Nano microcontroller, a sensor suite (accelerometer, barometer, gyroscope, magnetometer), and a flash storage module. The Arduino Nano controlled the sensors and data storage, while the flash module provided onboard data storage.

The primary objectives of the Aurora I avionics system were to:

- Gain experience and understanding of the COTS Blue Raven flight computers.
- Test various components of the avionics system, including battery types and sensor data collection.
- Gather flight data for post processing and analysis

3.1.2 Software Architecture

The above high-level flow chart in Figure 1 depicts how the Aurora I avionics system captures and transmits. The primary goal is to facilitate post-flight analysis and verification, aligning with the data collection intervals of the Blue Raven system (500 Hz high-resolution and 50 Hz low-resolution). Data logging is initiated by a launch event detected from the Blue Raven. During flight, raw sensor data is stored onboard in flash memory and transmitted via LoRa to ground communications, ensuring redundancy. While flash memory analysis is limited to post-flight, LoRa provides real-time monitoring capabilities.

High-Resolution Data Capture The high-resolution interval (500 Hz) focuses on data from an Inertial Measurement Unit (IMU), comprising accelerometer, gyroscope, and magnetometer sensors. This high sampling rate is used to accurately capture subtle changes in aircraft motion and orientation. Raw data from sensor registers is combined with headers into data frames, stored in a buffer, and then written to flash memory.

Low-Resolution Data Capture and Transmission The low-resolution interval (50 Hz) reads data from a barometer (pressure and temperature sensor) to determine altitude. Similar to the high-resolution process, raw data is framed and temporarily stored in a buffer before writing to flash. In this interval, both flash writing and LoRa transmission of the data occur.

Design Considerations and Implementation Alternative design approaches were considered, including a separate logging interval. However, the chosen design simplifies the system by consolidating flash writing and LoRa transmission into the low-resolution interval. This streamlines data handling tasks while still meeting the core requirements. Although LoRa communications was included in the initial Aurora I software design, the hardware to support this was unavailable at the time due to delays in hardware delivery. As a result, testing for LoRa communications organised to be implemented in Aurora II.

3.1.2.1 Data Storage and Handling

To support post-flight data processing, the sensor data is logged in a structured format using dataframes. This structured approach ensures that the host system can easily recognise and interpret the different types of data being read.

Data frame Structure Each dataframe consists of a two-byte header and a payload. The header identifies the type of data recorded (e.g., high-resolution, low-resolution, or payload data), specifies the number of bytes in the payload, and includes a synchronisation byte for post-processing alignment. The payload contains the raw sensor data, organised in a specific order depending on the data type.

To optimise storage efficiency and simplify implementation, the dataframes are stored in a `uint8_t` buffer, where each byte represents an 8-bit unsigned integer. This choice is driven by the fact that some sensor data is 24-bits per sample and using 8-bit nibbles allows for efficient space utilization while maintaining simplicity.

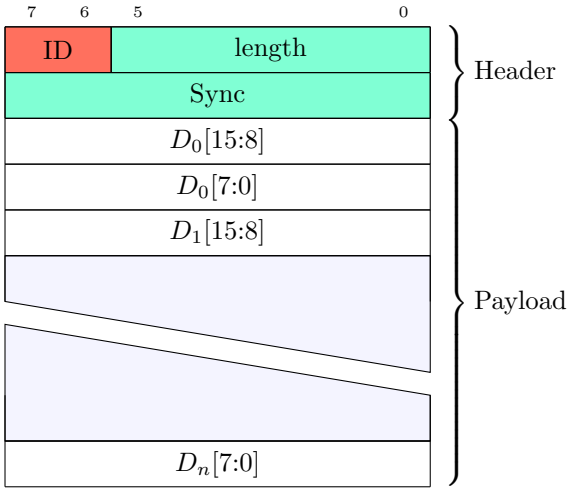

Figure 1: Dataframe structure for avionics

Data Typing The data stored by the avionics system retains the raw structure as output by the onboard sensors, written as 8-bit unsigned blocks. This minimises storage requirements and avoids computational overhead during flight. Post-flight analysis can then convert this raw data into meaningful values by applying sensor-specific scaling factors and unit conversions.

3.1.2.2 Synchronisation

Synchronisation between high- and low-resolution data, as well as with the Blue Raven data, is achieved using a "sync code". Based off the Blue Raven data user guide, this code is a counter that increments every millisecond, wrapping around at 250. By including this sync code in the header of each dataframe, the system ensures that data from different sources can be accurately aligned and compared during post-flight analysis. As detailed in Section 4.1.3.2, this sync code

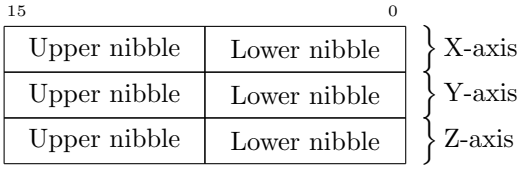

Figure 2: Example structure for a single sensor payload

is included as a full byte in the frame header, following the ID and frame length. This byte will be added into all saved data to ensure synchronisation consistency across all datasets

3.2 Aurora III

3.2.1 Avionics Design Overview

The Aurora III avionics system is designed to manage power, acquire flight and payload data, facilitate internal vehicle communication, and enable vehicle-to-ground communication. However, when compared to Aurora I and Aurora II, it also plays an additional role in supporting the Aerobrakes subsystem by providing essential flight data which includes altitude and velocity. Ground communication through LoRa protocol will also be included within the systems, supporting live state estimates during flight. From a high-level perspective, the avionics system comprises two SRAD (Student-Research and Developed) flight computers, each equipped with its own sensor suite for redundancy.

A block diagram illustrating the high-level design of the avionics system is shown in Figure X

The system utilizes an STM32F439 microcontroller as its core processing unit. This microcontroller was chosen by the boarder team for its familiarity, sufficient clock speed, ample I/O pins, and built-in SPI/I2C and CAN bus functionality. The addition of LoRa communication enables real-time data transmission to the ground station, enhancing monitoring and analysis capabilities. The system also incorporates the ability to receive and send data to the payload and aerobrakes subsystems via the CAN 2.0A protocol, facilitating integrated control and decision-making during flight.

3.2.2 Software Architecture

The Aurora III avionics software will undergo a significant architectural shift with the introduction of a Real-Time Operating System (RTOS). This decision was driven by the increased complexity of the avionics system compared to Aurora I. The addition of communication with the Aerobrakes and Payload subsystems, along with the need for precise timing and task prioritization, highlights the need for a more capable software framework. The engineering rationale for RTOS Implementation are as follows:

- **Increased System Complexity**: The Aurora III avionics system handles a wider range of tasks, including sensor data capture, state estimation, LoRa communication, inter-subsystem communication, flash storage management, and failover monitoring. An RTOS provides the necessary framework to manage these diverse tasks efficiently.
- **Real-Time Requirements**: Many avionics operations, such as sensor data sampling and state estimation calculations, have strict timing requirements. Timing is particularly critical for when the motor has burnt out and for when apogee has been reached as these events trigger critical actions in other subsystems. The detection of motor burnout signals the Aerobrakes subsystem to initiate its control sequence, while apogee detection triggers the deployment of recovery systems. Any delays in these actions could lead to mission failure or even jeopardize the safety of the rocket and its surroundings. An RTOS ensures that these time-critical tasks are executed within their deadlines, preventing potential failures due to missed timing constraints.

- **Task Prioritization**: An RTOS allows for the prioritization of tasks, ensuring that critical operations like sensor data acquisition and failover management take precedence over less urgent tasks like writing to flash and LoRa ground communications.
- **Efficient Task Communication**: The RTOS facilitates communication and synchronization between different tasks, enabling seamless data exchange and coordination between various components of the avionics system.
- **Feasibility**: The selected microcontroller, the STM32F439, has sufficient processing power and memory to support the overhead of an RTOS. Additionally, FreeRTOS, the chosen RTOS, is known for its lightweight and efficient, making it well-suited for embedded systems like the Aurora III avionics.

To implement RTOS, Aurora III will use FreeRTOS, an open-source RTOS known for its effectiveness and real-time capabilities. FreeRTOS provides a multitasking kernel that enables task scheduling, priority-based execution, and efficient task communication. This allows the avionics system to handle time-critical operations effectively while maintaining overall system responsiveness. The following flowchart provides a high-level overview of the tasks executed by the RTOS sorted by their priority.

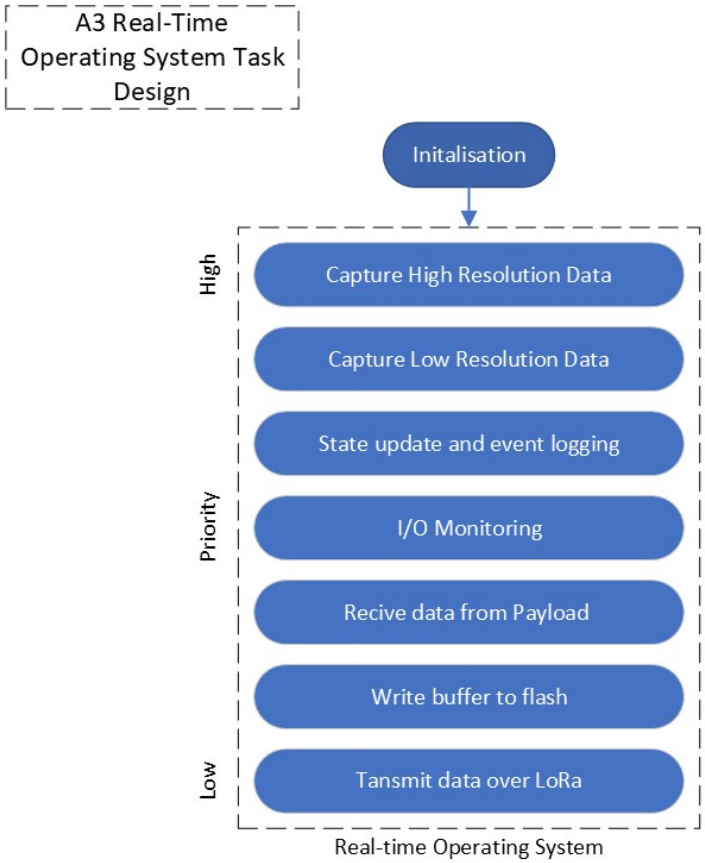

Figure 3: Aurora III RTOS task architecture

3.2.2.1 High Resolution Data Acquisition

The High-Resolution Data Acquisition task is a high priority process within the Aurora III avionics system, responsible for capturing raw sensor data from the gyroscope, accelerometer,

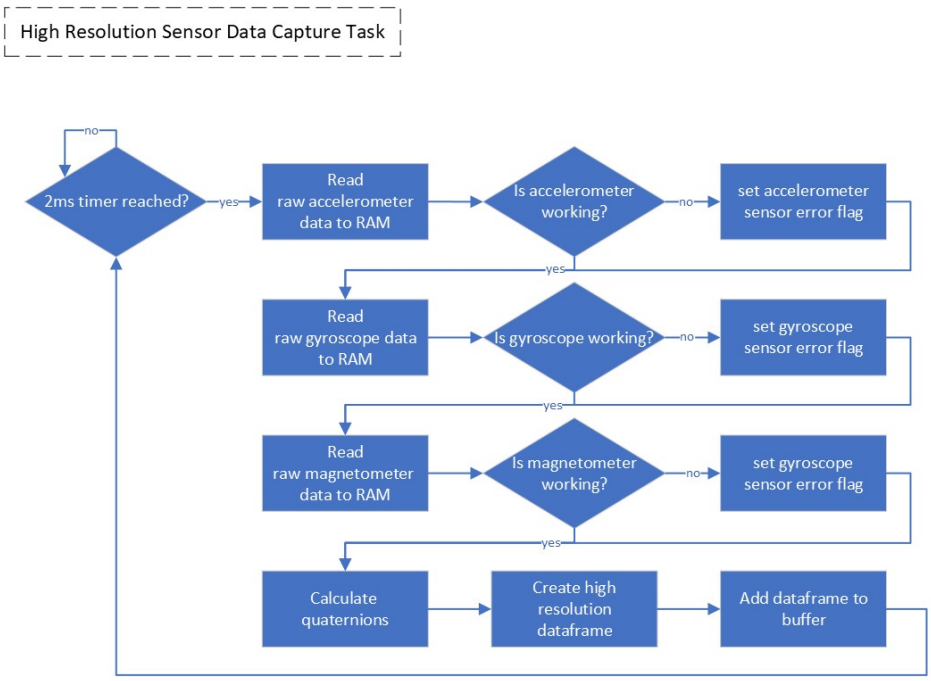

Figure 4: High resolution task flowchart

and magnetometer at a rate of 250 Hz. This high sampling rate enables the tracking of rapid changes in inertial measurements to determine precise attitude and movement estimation.

The tasks read raw data from the gyroscope and magnetometer, storing it in RAM and sets a sensor error flag if any sensors are not operational. These flags trigger the microcontroller to switch to redundant sensors in the I/O monitor task. The task then calculates the rocket's orientation using quaternions, derived from the raw gyroscope and accelerometer data. These quaternions, along with the raw sensor data, are packaged into a high-resolution data frame, complete with a header containing the data type and synchronisation information.

Finally, this data frame is added to a circular buffer, serving as temporary storage before being written to flash memory.

3.2.2.2 Low Resolution Data Acquisition

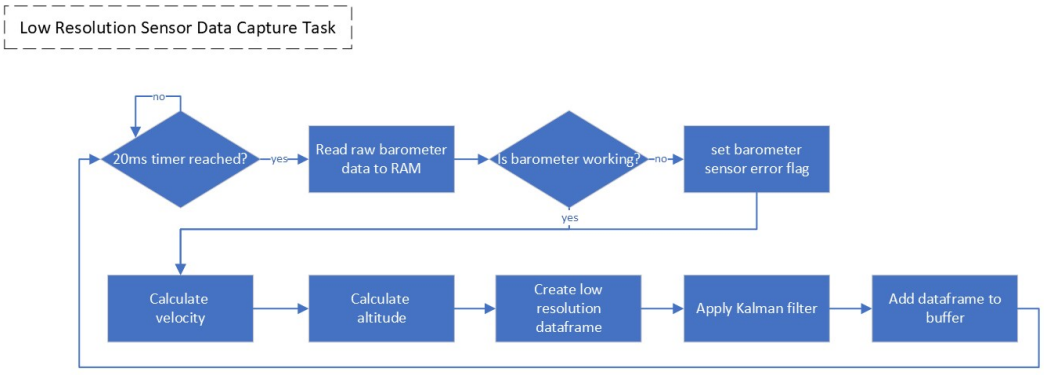

Figure 5: Low resolution task flowchart

The Low Resolution Data Acquisition task is responsible for collecting data from the barometer at a rate of 50 Hz. This task reads the raw barometer data, calculates altitude and velocity, applies a Kalman filter to smooth the data, and then creates a low-resolution data frame. Altitude and velocity are be used to inform aerobrakes and recovery. The data frame is then added to a buffer for subsequent writing to flash memory. Similar to the high resolution task, a sensor error flag is to be set if the barometer has malfunctioned.

3.2.2.3 State Update and Event Logging

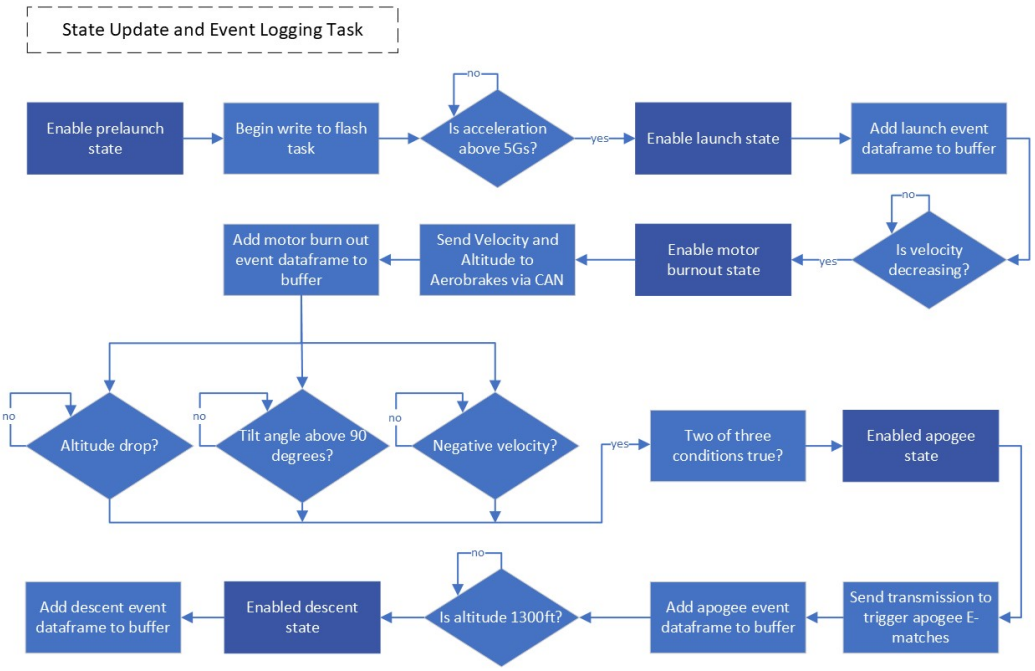

Figure 6: State and event system flowchart

The State Update and Event Logging task is responsible for tracking the rocket's state throughout the flight and logging significant events. It begins by enabling the pre-launch state and then transitions to the launch state when acceleration exceeds 5Gs. It then sends velocity and altitude data to the Aerobrakes subsystem via CAN if the measured velocity is decreasing. The task then continuously monitors velocity and altitude to detect apogee, which is defined as a decrease in velocity, or a combination of altitude drop, tilt angle above 90°, and negative velocity. Upon reaching apogee, the task sends a signal to trigger E-matches. Finally, the task transitions to the descent state when the altitude drops below 1300ft. Throughout the flight, the task adds event data frames to the buffer to record the occurrence of key events for post flight analysis.

3.2.2.4 I/O Monitoring

The I/O Monitor Task is a low-priority task responsible for overseeing the status and functionality of the avionics system's input/output operations. Its primary function is to detect and manage potential errors or malfunctions in the sensors and the slave microcontroller.

The task continuously checks if the slave microcontroller is operational. If the slave microcontroller is not responding, the I/O Monitor Task takes corrective action by resetting it to restore normal operation. Additionally, the task monitors error flags set by the sensor data capture tasks. If any sensor error is detected, the I/O Monitor Task initiates a switch to the redundant sensor.

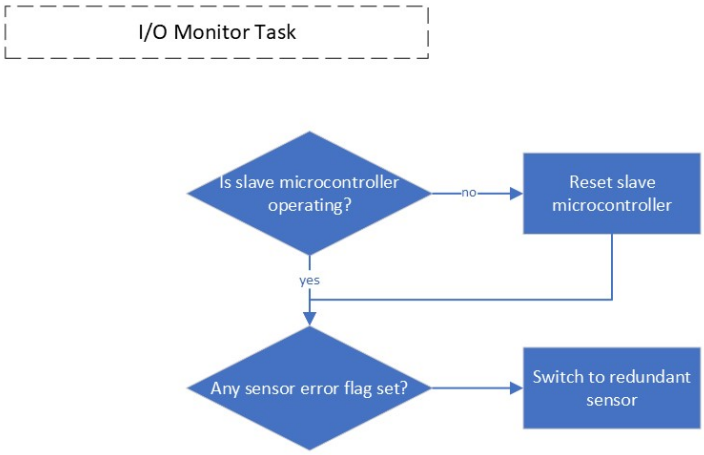

Figure 7: I/O monitoring flowchart

3.2.2.5 LoRa Communications

LoRa communications work is to be conducted by the ground stations team. However, this task is a lower priority task, responsible for transmitting data from the rocket to the ground station. It waits for a signal from the State Update and Event Logging task to begin transmission.

Once triggered, it sends data packets over the LoRa link, including sensor data, state information, and event logs. This real-time data transmission allows for ground-based monitoring and analysis of the rocket's performance during flight.

3.2.2.6 Memory Flashing

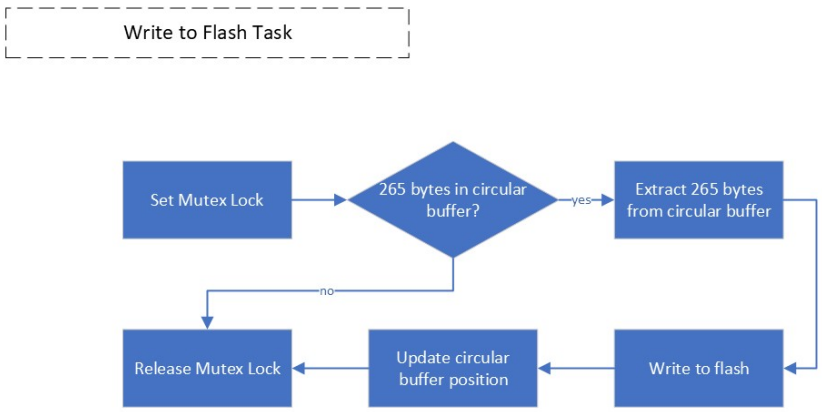

Figure 8: Data storage flowchart

The Write to Flash Memory task is also low priority process that manages the storage of data frames in the onboard flash memory. It uses a mutex lock to ensure exclusive access to the flash memory, preventing conflicts with other tasks. The task checks if there are enough data frames in the circular buffer to fill a page (256 bytes). If so, it extracts the data frames from the buffer, writes them to flash memory, and updates the buffer position. This process is to repeat in the background unless a higher priority task requires the CPU time or until the flight ends.

3.3 Testing Plan

3.3.1 Verification

Components	Test
Accelerometer (High G)	Shake table: • Subject the accelerometer to known vibration profiles (frequencies, amplitudes from A1 data) • Verify output readings accurately reflect the applied vibrations.
Accelerometer (Low G)	Shake table: • Subject the accelerometer to known vibration profiles (frequencies, amplitudes from A1 data) • Verify output readings accurately reflect the applied vibrations. Saturation test: • Subject the accelerometer to vibration greater than the peak measurable by the sensor. • Observe the behaviour of the sensor after saturation – how long it remains saturated, accuracy of the data after leaving saturation. Tilt test: • Slowly rotate the accelerometer across its measurement axes.
Gyroscope	Rate table: • Mount gyroscope on a rate table that can rotate at different angular velocities. • Verify output accurately measures the applied rotation rates in each axis. Bias stability: • Leave gyroscope sampling data intermittently over a long period of time (overnight) • Measure average readings within a window of time at each interval, averaging the samples. • Measure resulting drift over time from initial values
Barometer	Vacuum chamber / temperature test: • Placing Barometer in a vacuum chamber and apply different pressure and temperature within a range relevant to expected flight altitudes. • Verify readings accurately reflect the known pressure values.

Magnetometer	Outdoor testing: • Record data outdoors, with known changes in orientation. • Compare against expected magnetic field values for your location.
GPS	Outdoor testing: • Test in open locations under various conditions including stationary and while moving. • Evaluate accuracy against known locations or compare against high-precision reference GPS.
LoRa Communications	Transmission range: • Test in different environments (clear line-of-sight and obstructed) to determine effective range. Data integrity: • Transmit known data packets and verify correct transmission.
Flash Storage	Retention test: • Write data, power off for certain amount of time and verify data remains readable. Flash write timing: • Write different test data vectors to flash and measure the time it takes. • Perform tests with both full- and partial-page writes. Flash read timing: • Read data back in from flash chip and measure the time it takes.
Microcontroller	Code execution test: • Test program functions. Continuity test: • Monitoring signals and pin behaviour using logic analysers and oscilloscopes.

3.3.2 Validation

Sub-Assemblies	Test

Recovery Assembly (Sensors, Mirco and GPS)	Data injection tests • Place system in a testing mode, connecting recovery outputs to some readable output (i.e. buzzer, LED). • Feed in data collected from previous flights and determine if recovery system "fires" at the known apogee.
Gound Communications	Telemetry reception: • Test reception of data from Avionics Bay, ensuring that data is correctly being transmitted.
Storage	Data logging: • Validate that sensor data is correctly being stored in flash.
Internal Communications (CAN bus)	Message integrity • Test transmission and reception of all expected message types over the CAN bus.
State Estimation	Simulation: • Use a simulator to input realistic sensor data. Verify state estimation algorithms correctly calculate state estimation.
SRAD Redundancy	Error injected testing: • Testing for failover between primary and secondary systems. • Place system in a testing mode, inject errata on communication lines (payload/aerobrakes CAN, SPI, etc.) • Verify system correctly identifies sources of error and appropriately swaps to redundant system.

3.3.3 Operation

- Full Avionics integration testing
- Final Integration (Operational Testing)

Prior to launch, a final ground test will be conducted using flight-like sensor data taken from A1 and A2 profiles. This test will run the Avionics system in "test mode" where simulated packets are transmitted, successful SRAD deployment is indicated through an LED, and aerobrake deployment is verified. This final check verifies correct data flow, decision-making logic, and functionality of both the recovery and aerobrakes systems.

3.4 Data Processing and Analysis

Development of data processing tools and algorithms was in large part a significant challenge within the progression of the project. These systems were developed offline first, mostly through the implementation of scripts built with Python, through analysis of the data that was collected after the flight of Aurora I.

This section outlines these systems as they were designed, and how they were and will continue to be applied for the analysis of data throughout future Aurora rockets.

3.4.1 Attitude Estimation

As touched on previously, it is important that the rocket maintains some representation of its orientation relative to a global frame of reference during the flight. Since the accelerometers included within the available sensor suite are only able to perform inertial measurement relative to the body axis of the vehicle, the rotational information is applied to rotate and scale these values to determine a true vertical acceleration to apply in further calculations.

The instantaneous angular rate measurements output by the gyroscopic sensors can be considered as time derivatives of the Euler angle representation of the rocket's attitude. As such, by constructing rotation quaternions from these instantaneous rates and applying the rotation to the previously determined attitude (simply the last calculated rotation quaternion), effectively a numerical integration process is applied:

First the angular rates in °/s are converted to Euler angles (radians):

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \frac{\pi}{180}\cdot\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix}$$

A unit quaternion is then initialised through the half Euler angles (per equation 66 [5])

$$\dot{q} = \begin{bmatrix} c_{\phi/2}c_{\theta/2}c_{\psi/2} - s_{\phi/2}c_{\theta/2}s_{\psi/2} \\ c_{\phi/2}c_{\theta/2}c_{\psi/2} + s_{\phi/2}s_{\theta/2}s_{\psi/2} \\ c_{\phi/2}s_{\theta/2}s_{\psi/2} - s_{\phi/2}c_{\theta/2}s_{\psi/2} \\ c_{\phi/2}c_{\theta/2}s_{\psi/2} + c_{\phi/2}c_{\theta/2}s_{\psi/2} \end{bmatrix}$$

Where $c_x = \cos x$, $s_x = \sin x$. Then from equation 102 [5]:

$$\begin{aligned} q_k &= q_{k-1}\cdot\dot{q} \\ &= \begin{bmatrix} (q_{k-1_0}\dot{q}_0) + (-q_{k-1_1}\dot{q}_1) + (-q_{k-1_2}\dot{q}_2) + (-q_{k-1_3}\dot{q}_3) \\ (q_{k-1_0}\dot{q}_1) + (q_{k-1_1}\dot{q}_0) + (q_{k-1_2}\dot{q}_3) + (-q_{k-1_3}\dot{q}_2) \\ (q_{k-1_0}\dot{q}_2) + (-q_{k-1_1}\dot{q}_3) + (q_{k-1_2}\dot{q}_0) + (q_{k-1_3}\dot{q}_1) \\ (q_{k-1_0}\dot{q}_3) + (q_{k-1_1}\dot{q}_2) + (-q_{k-1_2}\dot{q}_1) + (q_{k-1_3}\dot{q}_0) \end{bmatrix} \end{aligned}$$

Finally, normalise the quaternion to $[-1, 1]$:

$$q_{normal} = q_k / \sqrt{q_{k_0}^2 + q_{k_1}^2 + q_{k_2}^2 + q_{k_3}^2}$$

The resulting quaternion as calculated through the integration of angular velocities then represents the rotation of the rocket body from its initial pose.

3.4.2 State Estimation (Velocity and Altitude)

In order to determine information on the rocket's displacement and upward velocity, a Kalman Filter is applied for greater accuracy in estimation. The filter operates in two steps, first predicting future state values and then updating the predictions through measurement and calculated errors.

"An introduction to the Kalman filter" describes the following equations for a linear Kalman filter:

Time update (predict):

(1) Project the state ahead

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$$

(2) Project the error covariance ahead

$$\bar{P}_k = AP_{k-1}A^T + Q$$

Measurement update (correct):

(1) Compute the Kalman gain

$$K_k = \bar{P}_k H^T (H\bar{P}_k H^T + R)^{-1}$$

(2) Update estimate with measurement z_k

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

(3) Update the error covariance

$$P_k = (I - K_k H)\bar{P}_k$$

The state is defined as a tri-cell column vector $x = \{s, v, a\}^T$ for the rocket's altitude, upward velocity, and upward acceleration. The system is modelled with constant acceleration, where v and s are simply integrated from a. This results in a state transition matrix A:

$$A = \begin{bmatrix} 1 & dt & \frac{1}{2}dt^2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Since s and a are measured directly through the barometer and accelerometer respectively, the corresponding measurement model is then:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Assuming no control input to the system, B and u are both omitted. The measurement of s is simply the altitude as calculated directly from the barometric data, while a must be processed as part of the measurement update.

As the measured acceleration is in reference to the rocket's body axis, the tilt angle is applied to compensate. The previously determined attitude quaternion rotates a unit vector $\hat{\mathbf{z}}$, representing the global vertical axis, to the rocket's body frame axis $\hat{\mathbf{z}}_{\boldsymbol{B}}$. The dot product of these two vectors provides the cosine of the tilt angle $\cos\theta = \hat{\mathbf{z}}_{\boldsymbol{B}} \cdot \hat{\mathbf{z}}$, which is multiplied by the measured acceleration to determine the upwards acceleration in the global frame of reference.

Given the inertial measurement of the accelerometer includes the reactionary force opposing acceleration due to gravity, this must also be subtracted from the measurement in order to purely determine the upward acceleration of the rocket. The resulting measurement vector follows:

$$z = \begin{bmatrix} s \\ (\cos\theta \cdot a) - g \end{bmatrix} \tag{3}$$

Fortunately, the Blue Raven flight computers provide flight data post-recovery against which the performance of these calculations can be measured. A particular advantage of this is that it provides greater flexibility in tuning the noise parameters Q and R of the filter. These were determined offline through a technique provided by the *BayesianOptimization* [10] Python module.

The routine calculating state with the implemented filter was fed into the optimisation module, returning a root mean square error between the calculated estimate and the measurements from the Blue Raven. Over several iterations this then determines the noise parameters corresponding to best performance, which can then be applied within the filter.

4 Aurora I&II Results and Analysis

4.1 Key Findings

4.2 Verification Results

4.3 Challenges and Refinements

5 Project Management and Timeline

5.1 Project Timeline

5.2 Risk Assessment and Mitigation

Although the most risks were outlined in our proposal and risk assessment, it is worth noting that there will additional risk primarily from conducting verification and validation of hardware. To validate the avionics senor suite, a range of tools will be required which include, shake tables, vacuum chambers and rate tables. Such tools will be accessed through RMIT physics department and laboratory.

Currently, the team is working closely with technical officer Peter McGlynn to assist with providing access to the relevant tools. Although most tests will be relatively straight forward, the use of a vacuum chamber to test the barometer poses a safety concern. In addition to controlling a vacuum, it is planned to have a heat source to within the chamber to adjust the temperature, simulating the temperate increase in the rocket's environment during flight.

It was suggested that toaster wires can be fed through the camber to do so. Creating and using such mechanism will be challenging, however all activity and equipment risk assessments will be completed before any validation testing is conducted.

6 Discussion and Recommendations

6.1 Critical Evaluation of Progress

The project is progressing well towards meeting its objectives and deliverables. The successful launch of Aurora I and the resulting data analysis have provided valuable insights into the performance of the avionics system and identified areas for improvement. The implementation of the RTOS in Aurora III will significantly enhance the system's ability to handle complex tasks and meet real-time requirements.

The data collection and logging processes are functioning as intended, and the initial validation tests have shown promising results. However, there are areas that require further attention. Although the RTOS implementation is feasible, this is the first time any team member has used an RTOS. Extensive testing and validation are required, ensuring that software design is operating in a flight environment as intended.

Additionally, the integration of the avionics system with the aerobrakes and payload subsystems also requires thorough testing and validation to ensuring aerobrake processes are triggered at the right time.

6.2 Future Recommendations

The tight deadlines associated with rocket launches highlight the need for a more proactive approach to testing and validation. A more detailed plan regarding how and when testing of both individual components and integrated subsystems will need to outlined, which includes

ground tests and simulations to identify and address potential issues before the rocket takes flight.

Furthermore, the delays in hardware highlight the importance of utilising simulation environments as testing platforms during waiting periods. Although some simulation was conducted, it is recommended to priorities tests, especially when using new techniques such as RTOS and real-time data processing.

7 Conclusions

Bibliography

[1] Adrian Adamson. *Blue Raven User's Guide*. https://www.apogeerockets.com/downloads/PDFs/09170-blue_raven_users_manual_september_15.pdf. Sept. 2023. (Visited on 04/04/2024).

[2] Adrian Adamson. *Featherweight Blue Raven Development Thread: Recorded time-series data*. Rocketry Forum - Model Rocketry Forums, Mar. 2023. URL: https://www.rocketryforum.com/threads/featherweight-blue-raven-development-thread-recorded-time-series-data.179129/ (visited on 05/31/2024).

[3] Stefano Di Cairano and Ilya V Kolmanovsky. "Real-time optimization and model predictive control for aerospace and automotive applications". In: *2018 annual American control conference (ACC)*. IEEE. 2018, pp. 2392–2409.

[4] Zhanlin Diao et al. "Analysis and compensation of MEMS gyroscope drift". In: Dec. 2013, pp. 592–596. ISBN: 978-1-4673-5222-2. DOI: 10.1109/ICSensT.2013.6727722.

[5] James Diebel et al. "Representing attitude: Euler angles, unit quaternions, and rotation vectors". In: *Matrix* 58.15-16 (2006), pp. 1–35.

[6] Christian Dietrich and Daniel Lohmann. "OSEK-V: application-specific RTOS instantiation in hardware". In: *ACM SIGPLAN Notices* 52.5 (2017), pp. 111–120.

[7] Jake Ellenberger. "Pioneer Rocketry-2016 Midwest Rocket Competition". In: *Proceedings of the Wisconsin Space Conference*. 2016.

[8] Evan G Hemingway and Oliver M O'Reilly. "Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments". In: *Multibody system dynamics* 44 (2018), pp. 31–56.

[9] J Martens et al. "Student organization for aerospace research Atlantis II. Sounding Rocket". In: *SA CUP project technical report* (2018).

[10] Fernando Nogueira. *Bayesian Optimization: Open source constrained global optimization tool for Python*. 2014–. URL: https://github.com/bayesian-optimization/BayesianOptimization.

[11] Grantham Pang and Hugh Liu. "Evaluation of a low-cost MEMS accelerometer for distance measurement". In: *Journal of Intelligent and Robotic Systems* 30 (2001), pp. 249–265.

[12] Greg Welch, Gary Bishop, et al. "An introduction to the Kalman filter". In: (1995).

Appendix A