ENGR 301 Project 12 Architectural Design and Prototype

Mohammad Al-Rubayee, Ciaran King, Nicholas Lauder, Ikram Singh, Angus Tan, Angus Weich, Finn Welsford-Ackroyd

Table of Contents

1. Introduction

Amateur rockets are flown regularly worldwide. These rockets often exceed the speed of sound and altitudes above 30 km are not unheard of. These rockets are almost never controlled, they are stable due to passive aerodynamic features.

While passively stable rockets are reasonably simple and reliable if well designed, they are susceptible to a variety of disturbances, particularly early in flight. Unexpected winds can cause the rocket to weathercock; flexibility in the launch tower/rail can cause railwhip, imparting a random launch angle to the rocket; the thrust from the rocket motor is also never perfectly symmetrical.

The stabilising response afforded by active control provided changes depending on the measured state of the rocket. While it requires a significant overhead of weight and complexity; active control can provide course correction that passive stabilisation is not able to achieve.

Client

Andre Genldenhuis is the client for this project. He is a rocket enthusiast and is a member of the New Zealand Rocketry Association. He has had previous experience with building and launching rockets. Contact email: Andre.Geldenhuis@vuw.ac.nz

1.1 Purpose

The purpose of the avionics package is to provide active control for aerodynamic launch vehicles to launch without the aid of a launch rail.

1.2 Scope

This avionics package:

- Shall provide telemetry broadcasting over a wireless channel.
- Shall provide location broadcasting over a wireless channel.
- Shall provide data logging for diagnostics and analysis.
- Is to be used on actively controlled rocket vehicles which use gimballed solid fuel motors.

2. References

[1] Civil Aviation Authority of New Zealand, Civil Aviation Rules Part 101 Gyrogliders and Parasails, Unmanned Aircraft (including Balloons), Kites, and Rockets - Operating Rules, 10 March 2017, [Online]. Available: https://www.caa.govt.nz/rules/Rule_Consolidations/Part_101_Consolidation.pdf. [Accessed May. 12, 2018].

[2] Draw.io, free online diagram software for making flowcharts, process diagrams, org charts, UML, ER and network diagrams, [Online]. Available: https://www.draw.io/. [Accessed May. 12, 2018].

[3] A. Venkataraman & K. K. Jagadeesha, "Evaluation of Inter-Process Communication Mechanisms", [Online]. Available: http://pages.cs.wisc.edu/~adityav/Evaluation_of_Inter_Process_Communication_Mechanisms.pdf [Accessed May. 12, 2018]

3. Architecture

3.1 Stakeholders

Stakeholder Requirements

Client

- Avionics package shall fit entirely inside a 29mm airframe rocket vehicle.
- Avionics package shall interface with the gimbal.
- Avionics package shall guide the rocket on course.
- Avionics package shall stabilise the rocket so that it stands in an upright position while preparing to launch.
- Avionics package blueprints and information shall be published on the client's GitHub as open-source.
- The avionics package logical database will be human readable in the form of a text file.

Software Developers / Testers

- Avionics package development environment shall allow rapid deployment to the avionics package.
- Avionics package shall have means to feed back information about it's performance.
- Avionics package shall have organised modules with defined responsibilities.

New Zealand Student Space Association (NZSSA)

- That the NZSSA is kept up to date on the development of the avionics package for a small rocket

School of Engineering and Computer Science (ECS)

- ECS requires that all aspects of the project must be correctly licensed.
- It is vitally important that the project also follows all guidelines set out by ECS.

Wider Rocketry Community

- To benefit the wider rocketry community, the project is required to be open source so that it can be used and edited by other licensed users under the open source license. If the package is not open source the potential uses by the community are reduced as it cannot be changed to suit different projects.

Residents of Wellington Region

- It is essential for the residents of Wellington that the avionics package is reliable and conforms to all relevant safety regulations and guidelines from relevant organisations and ruling bodies.

Local Council

- The local council (Wellington City Council) requires that the rocket operates without violating any laws or regulations governing the use of model rockets in the area.
- It is also required that during the operation of the rocket there is no damage to property or persons linked to the use of the rocket.

Civil Aviation Authority (CAA)

- The CAA requires that any rockets conform to all regulations set out in CAA Regulations Part 101 [1].

3.2 Architectural Viewpoints

Logical

The logical viewpoint describes the functionality of the system it provides to end-users; the viewpoint breaks the system down into key concepts, which are represented using class and state diagrams.

Development

The development viewpoint describes the architecture that supports the processes involved in the software development cycle aims to address concerns regarding the management of the software.

Process

The process viewpoint deals with the dynamic and non-functional requirements of the system, such as performance and availability, addressing concurrency and distribution of the system's integrators and how main abstractions from the logical view fit within the process architecture.

Physical

The physical viewpoint details how the software fits within the given hardware, mapping software to different parts of the hardware for single and multi-system architectures; in particular, it deals with hardware availability, reliability, performance, and scalability of the system.

Circuit Architecture

The circuit architecture viewpoint describes how the physical system will be put together and addresses concerns regarding construction and any physical phenomenon that may interfere with processes of the physical system.

Hardware Architecture

The hardware viewpoint details means and purposes for which the software and systems interact with the hardware fundamentally, concerning how the hardware enables the overall system to function.

Scenarios

The scenario viewpoint is a collection of use cases that describe the interactions between users or processes, used as a qualitative review of a given architecture and form the basis for the tests of the architecture.

4. Architectural Views

4.1 Logical

The logical view is the structural breakdown of computational, communicational and behavioural responsibilities of the system. Multiple styles of diagrams are used to portray the logical view; one example is

a box and arrow diagram. These show a box which indicates the functional components of the system and arrows connecting them, indicating dependencies between the boxes. Box and arrow diagrams have the benefits of being intuitive and can convey a large volume of information about the high-level architecture in a single slide. However, they have the disadvantage of being imprecise, for example, the ambiguity of the meaning behind the arrows.

A class diagram shows a set of classes and their logical relationships: association, usage, composition, inheritance, and so forth. Sets of related classes can be grouped into class categories. Class templates focus on individual classes; they emphasise the main class operations and identify essential object characteristics.

The notation used for the logical view follows the notation of Unified Modeling Language (UML) case diagrams. In figure 4.1.1 is shown a class diagram for the avionics package; it shows the relationship between the modules, the main function, and the states.

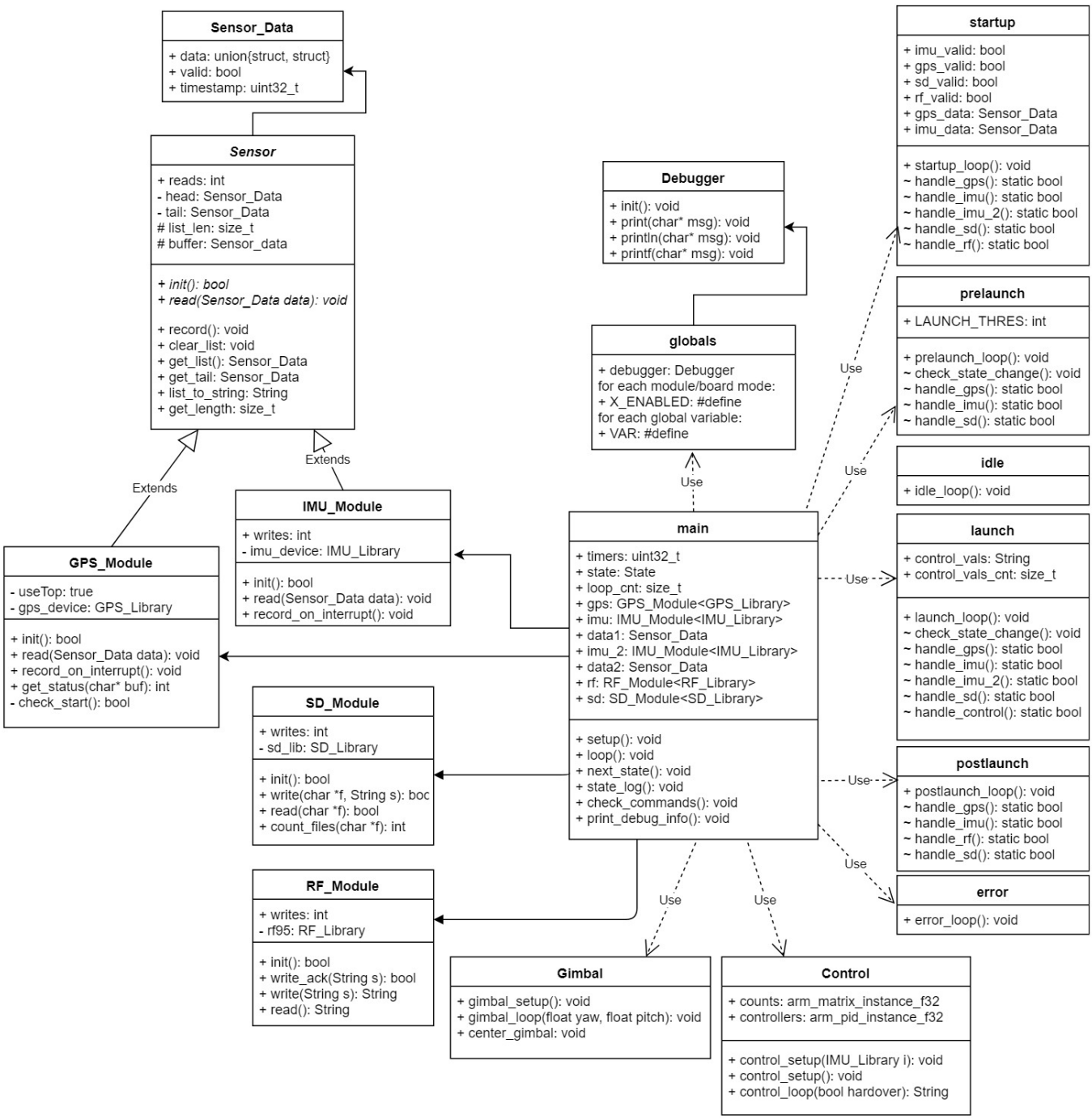

Figure 1: Class Diagram for the avionics package.

Mapping of classes to tasks:

- The GPS_Module class - This class is where all primary GPS methods are procured. The responsibilities of this class are to read and record the data received from the GPS module to the main function. The primary data that will be recorded will contain the physical location of the avionics package based on the satellite positioning system. This class has an association with the GPS_Data class which contains the value of the continuously updating location of the avionics package.
- The IMU_Module class - This class is where all primary inertial measurement unit (IMU) methods are obtained. The responsibilities of this class are to read and record the data received from the IMU to the main function. It is associated with the IMU_Data class which contains all the initialised variables for the accelerometer, magnetometer, and gyroscope.
- The Logger class - This class sends all the data that the inertial measurement unit and satellite positioning system send to the main function to the logger, where it holds the data and formats the data sets. This class has an association with the RF_Module class and SD_Module class. When data formatting has completed, the logger will send the formatted data to either the Radio Frequency board or the SD card, respectively.

4.2 Development

The development viewpoints concerns include the module organisation, standardisation of the design, standardisation of the testing, performance monitoring and branching organisation. The main stakeholders of the development viewpoint are the Software developers and the testers. This viewpoint applies to all systems that require significant software development. There are a range of problems that can occur within the development viewpoint such as too much detail in the planning, an uneven focus of the development team, lack of focus from developers, lack of precision while implementing components and problems with the development environment.

There is a well-founded requirement for clear, distinct module organisation within the software system. This is achieved by separating each component into its own module and ensuring that the modules are not dependant on one another. Ensuring transparent organisation allows for easy addition or removal of modules without disrupting other parts of the system.

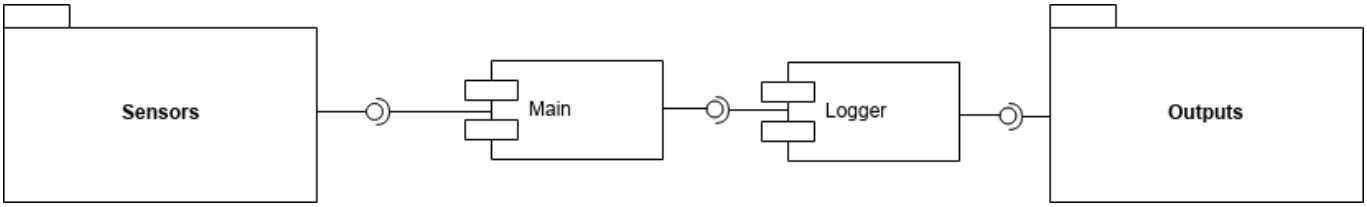

Figure 2: Component diagram of the system

Common processing ensures a linear pathway for data transferred within the system. The software to be implemented is fairly small but with specific goals. These goals involve moving sensor data from the sensor modules to the logging modules. With common processing of how this can be achieved, the system can be well defined in the steps that are taken to achieve these goals. Furthermore, with common processing in place, sensor or logging modules can be easily added or removed to the system without disrupting the tasks of other modules. This is represented visually in the process section.

Standardisation of design is achieved with a mixture of different views. For example, to standardise design, the development team must make use of common processing and module organisation; these are both key design points of a system. The design can be standardised by roughly planning each component within the team; discussing how it affects the existing system, what it will bring to the new system, what existing components/functionality it makes use of, and where the new component(s) fits into the model. Keeping similar modules, like sensors, in the same space and talking, in the same way, allows the developers to have a clear visual understanding of how the system fits together while avoiding too much abstraction.

An issue with testing embedded systems is that there are a number of challenges that need to be overcome in order to test effectively. These primarily concern the fact that physical changes need to take place in order to test the whole system properly. With this range of different aspects in place, there are multiple ways testing can be done. E.g. writing defined tests in the internal software system, writing runtime tests in the internal software system, writing an externally applied test suite that can communicate with the board over a serial or radio frequency connection, or even having a number of physical actions and ensuring that the outputs are within acceptable bounds. Different developers will have different preferences to how the system will be tested and some may try to test their modules in a way that only they see fit. Having a well-defined testing standard within the development team ensures that there is a clear view of what is being tested and to what level of scrutiny it is being tested too. Testing standards also encompass the way in which tests are written. By standardising test structure, any member of the team can quickly recognise the functionality as a specific test and easily make modifications as necessary.

As the avionics package requires real-time updates to control the direction of the rocket, proper instrumentation is vital to ensure that the software is optimised as necessary. Using the Arduino framework presents advantages in that the process of monitoring clock cycles by using the micros() method is well documented. This can be placed throughout the software package surrounded by precompiler conditions, designed to only run when necessary. This type of instrumentation will allow the recording of how fast a particular method, or combination of methods is running. This will be used to monitor performance changes between updates and optimisation of the software system

Branching the project allows features to be developed into a separate environment to the master branch. This means that developers can safely push to the remote repository, allowing all to see, but also keep potentially buggy code out of the master branch. When the feature has been finalised, other developers can review the changes and either approve a merge request or give specific reasons as to why to reject it. This method of branch organisation ensures the safety of new code, as it is not only stored locally, and significantly benefits others in reviewing the changes as they can see exactly what changes are to be made before confirming the implementation. Branching is built into Git, which the is the version control system used by this project.

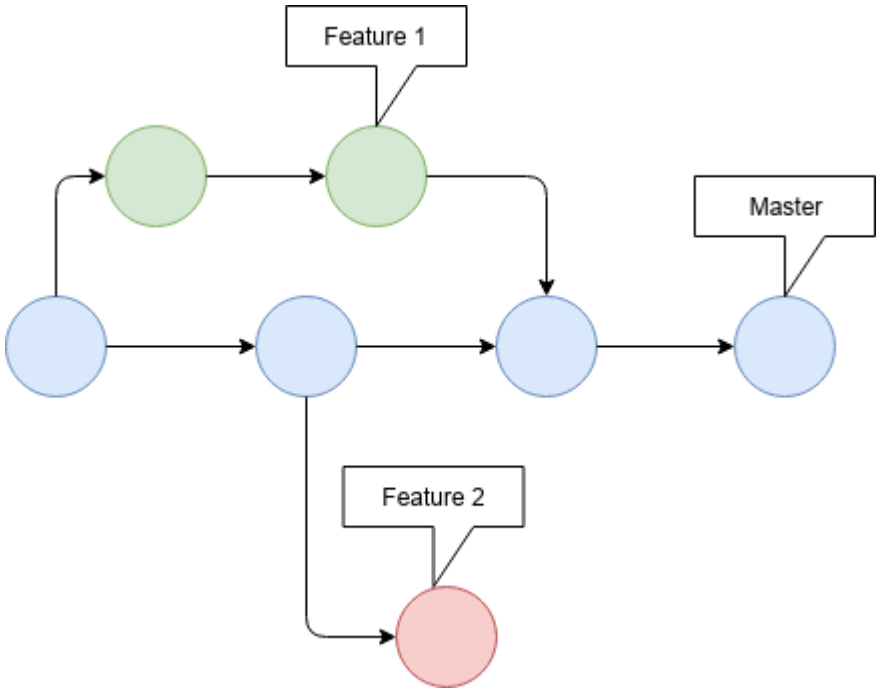

Figure 3: Branching Example

Certain scenarios must be avoided to ensure that the software development process is smooth and predictable. While designing the architecture, it is important to avoid adding too much detail to plans. By adding large amounts of detail, the team restricts itself in terms of manoeuvrability when unexpected scenarios arise. It is vital that when faced with an issue, any developer should be able to make the changes necessary to make a fix. Specific details of an approach to a module should not be made until there is sufficient knowledge about the implementation and environment that the module is in.

Agile and scrum development strategies are an effective way of managing team focus during project development. By using these processes, the team can monitor the activities of others, not in a watchful and negative way, but in a way that can be used to support one another and provide solutions to problems. Having regular meetings, such as standups, the team can be held accountable for what they are working on and how quickly they are working on it. This can be correlated to the git commits to show if any given member is working on what the entire team has decided on. When the team does have sprint planning session, they create and assign tasks that the team has agreed on and that the team is expected to stick to within this sprint.

4.3 Process

A process is a series of actions or steps taken to achieve a particular goal. Processes represent a level of the process architecture that can strategically control the starting, recovery, reconfiguration and shut down of the system. Additionally, processes can be replicated for increased distribution of the processing load, or for improved availability. This is particularly useful if the system is implementing or automating a business process and can be used to highlight aspects such as parallelism, concurrency, where processes fork or join, etc.

The concurrency present in the system can be associated with the reading of the data. It is vital for the system to be perpetually reading the data of the GPS, IMU, and gimbal. The concurrency of this system will be coordinated and controlled via a multitasker function that will contain the necessary components that will

allow the reading and writing of the continuous data flow into main function without the risk of crashing. This will also be aided by a debugging process.

At the highest level, the process architecture will show how the system works as a whole. It demonstrates the independently executing logical networks of communicating programs distributed across a set of hardware resources connected by a local area network (LAN) or a wide area network (WAN).

The notation used for the process view follows the notation of UML activity diagrams. In figure 4.3.1, the UML activity diagram for the avionics system shows the process of which the system will flow through.

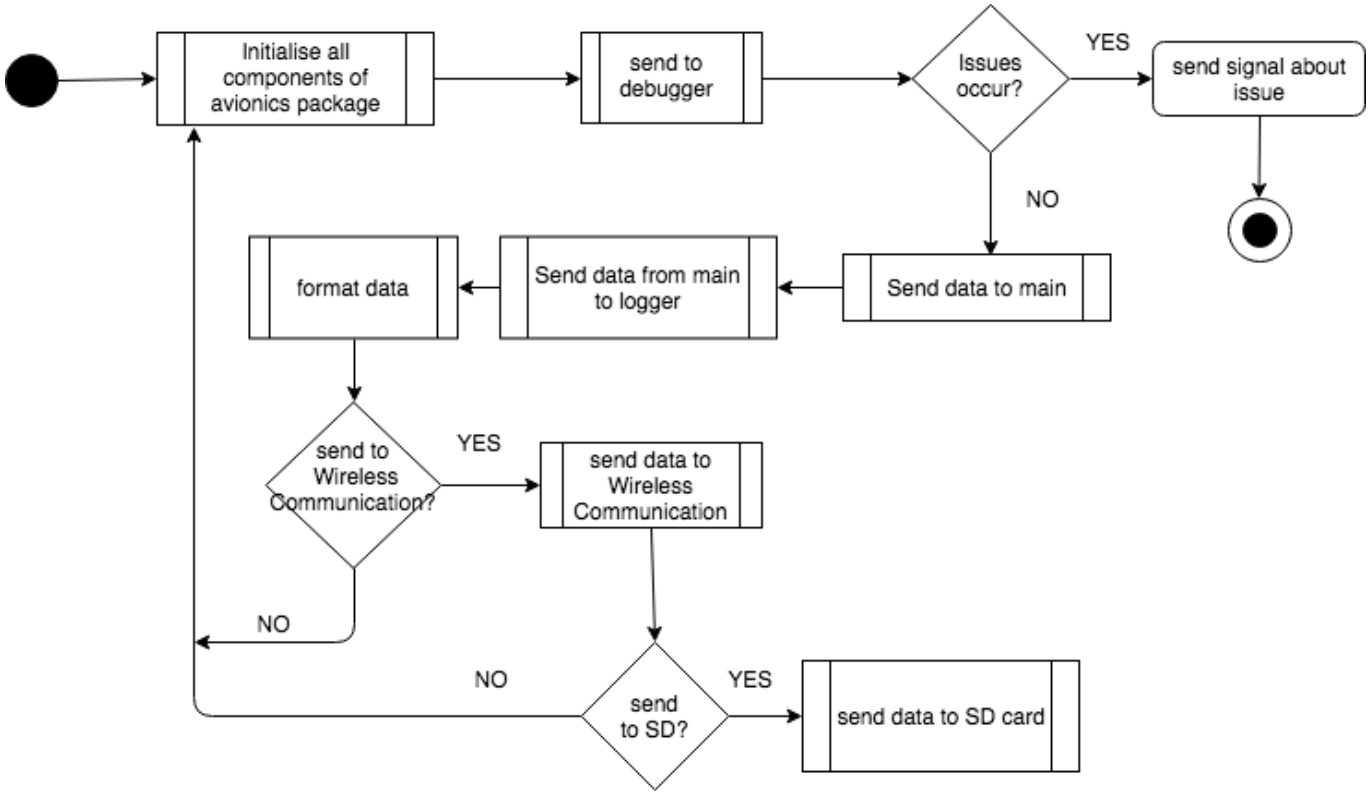

Figure 4: UML activity diagram for the avionics system.

There are concerns that need to be contemplated such as:

- Task structure - This is a major concern to this system. For example, consider a variable that is reading or writing data; for that to occur with no problems there needs to be a receipt that gives the variable the green light to follow through the reading or writing of data. The reason for this is because, as concurrency occurs, there may be a dangerous overlapping of two variables reading or writing from the same function and they may overwrite each other. The task must be structured in a way that allows for continuous reading and writing of data while keeping a linear motion of such actions.

Mapping of functional elements to tasks:

- Interprocess communication - Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. This allows a program to handle multiple user requests at the same time. The primary concern regarding this is that the IPC used for this system is not fast enough or has limitations to its performance. There are three main interprocess communication mechanisms: pipes, sockets and shared memory.

A pipe is a unidirectional communication device that permits serial transfer of bytes from the writing process to the reading process [3]. A socket is a bi-directional communication mechanism that can be used to communicate with another process on the same machine or on a different machine [3]. Shared memory allows two or more processes to access the same memory region, which is mapped onto the address space of all participating processes [3]. In accordance to the article 'Evaluation of Inter-Process Communication Mechanisms' authored by A. Venkataraman and K. K. Jagadeesha, shared memory is the fastest form of IPC due to all processes sharing the same piece of memory, avoiding the copying of data.

- State management - This concern refers to the management of the state of one or more user interface controls. The states of the system are: pre-launch, during launch and post-launch. In pre-launch, the user is required to send the configuration to the system, this requires an undisturbed connection. During a launch, the control code will follow through its commands in collecting data, writing it and sending it through wireless communication. In post-launch, the gimbal is locked to one side and the GPS will broadcast its location.

- Synchronisation and integrity - This concern refers to synchronisation of processes and synchronisation of data. Data synchronisation refers to the idea of keeping multiple copies of a dataset in coherence with one another, or to maintain data integrity. Some challenges that the user may encounter in data synchronisation are data formats complexity; where data formats tend to grow more complex with time as the organisation grows and evolves. Real-timeliness; wherein real-time systems users want to see the current status of the processes. Data quality; where it is common practice to store the data at one location and share with different systems and/or applications from different locations, (this assists in preventing inconsistencies in the data).

- Startup and shutdown - The concerns for the startup is the initialising process. If the components do not initialise properly, for example, if cables are damaged, incorrect configuration of module libraries, insufficient power supply to the components etc. the entire system would be compromised and be unable to read or write any data.

  The concerns for the shutdown is not necessarily the shutting down of the system as requested by the user but rather that the system shuts down without the required user input. For example, in the case that a component breaks down in some way and causes the whole system to shut down; what is required for this concern to be invalidated is that if such a scenario is to occur, a message must be sent to the user about the location of the issue while ensuring the system is kept turned on. The user will then request the shutdown of the system as to go mitigating the problem. Fundamentally, the shutdown of the system should only occur with the request and confirmation of the user.

- Task failure - If a task fails, the concern is that it could affect other tasks and cause failure in other parts of the system. For example, if a variable fails to read data from a particular component; it is of high importance to make sure that such a failure does not inhibit other variables from reading the components.

  A similar example is the process of sending the data from the main function to the logger. It is of extreme importance to introduce a fault checking system to account for the case that the process may be unreliable; furthermore, the fault checking system can attempt to send that data again should the transmission process fail. The main concern in this is that a task failure must not propagate throughout the system; failures need to be isolated.

4.4 Physical

The physical view shows how the final system will be spread out across different machines. The avionics package has a simple physical view as the avionics package must act with near autonomy. The software works on the avionics package managing the data gathering, control system and data logging which includes logging the data to an SD card and transmitting it to the base station via radio. The base station will run its own process to receive the data from the avionics package and store the data in a file within the base station. This view addresses concerns of reliability by having both the base station and avionics package logging data, so that if one of the processes fails in some manner the other would still record the data. Scalability can be handled by improving the processing power of the avionics package or increasing the number of base stations receiving the data and spreading them out so that if some physical phenomenon interferes with the radio transmission to one base station, another base station will likely be able to receive and log the data.

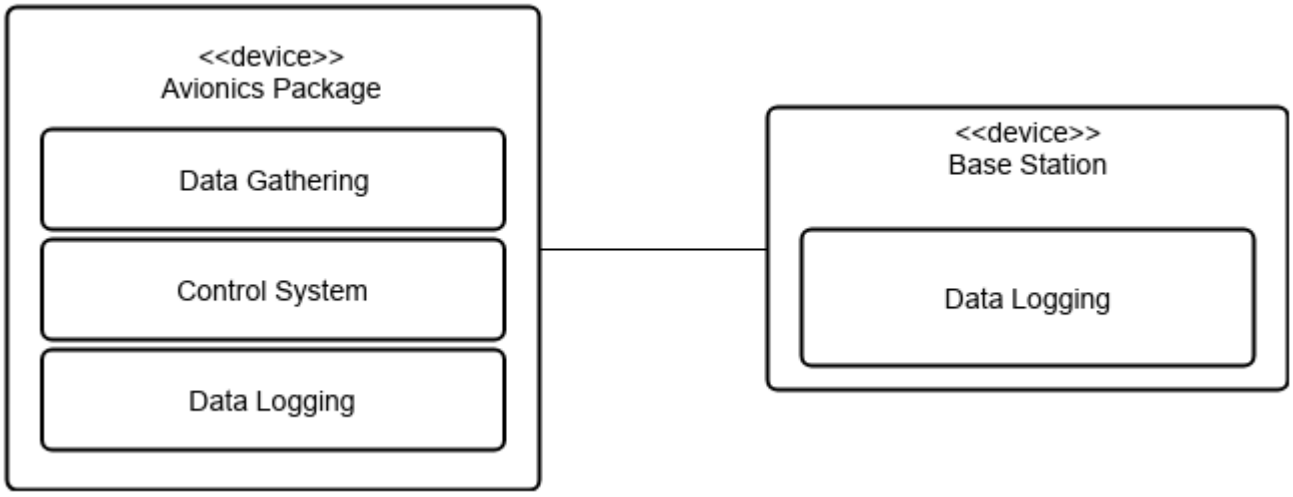

Figure 5: Deployment Diagram for the Avionics Package.

4.5 Circuit

The circuit architecture describes the physical layout of the components of the system. This allows the systems engineer to more easily lay out the circuit by abstracting the components into general blocks.

The two main circuit blocks are the power and microcontroller blocks, described as such as they connect most of the other blocks. The microcontroller is the foremost central block as it receives data from the IMU and GPS and performs any analytics needed for the control system. Post control system, the microcontroller takes the output of the control system and sends instructions to the servos to allow the system to sustain vertical flight by means of adjusting the orientation of the thrust vector. The data is also sent to the SD card reader to be stored on on-board SD card for later debugging. Some of the data is also sent to the radio to be transmitted back to the base station, including but not limited to, data from the IMU and GPS. The microcontroller also can receive data from the radio to instruct it to perform specific tasks such as ejecting a parachute or igniting the motor.

The power supply is also a vital block as it facilitates power to most of the other blocks except the antennas and supplies a ground reference for the overall system. The power supply block is comprised of connector pins connected to a lithium-ion polymer (LiPo) battery which supplies the power and ground reference. A safety diode will be included on the input rail of the power block to ensure that, in the case that the battery is mistakenly plugged in backward, reverse current flow does not damage the system. Finally, a voltage regulator must be used to ensure the system receives a constant and reliable source of power.

The IMU must be attached to the airframe, with wires to facilitate its connection to the microcontroller. This is essential as the orientation of the IMU must remain constant with respect to the airframe; a critical requirement for the control system to be accurate. It is therefore vital that the IMU can be mounted separately from the rest of the avionics package which may not be able to be mounted vertically within a rocket and which may move around within the airframe during flight.

The radio block must be connected to the antennas so that the radio can able to transmit data in all directions. This will ensure that regardless of the orientation of the package, data can still be received by the base station(s).

This architecture allows for scalability by including extra ports on the board for later additions such as a remote parachute. Another concern addressed by this architecture is the ability to bug-fix the system, this is dealt with by having multiple test points on the circuit that allows a developer to quickly check the status of each circuit track.

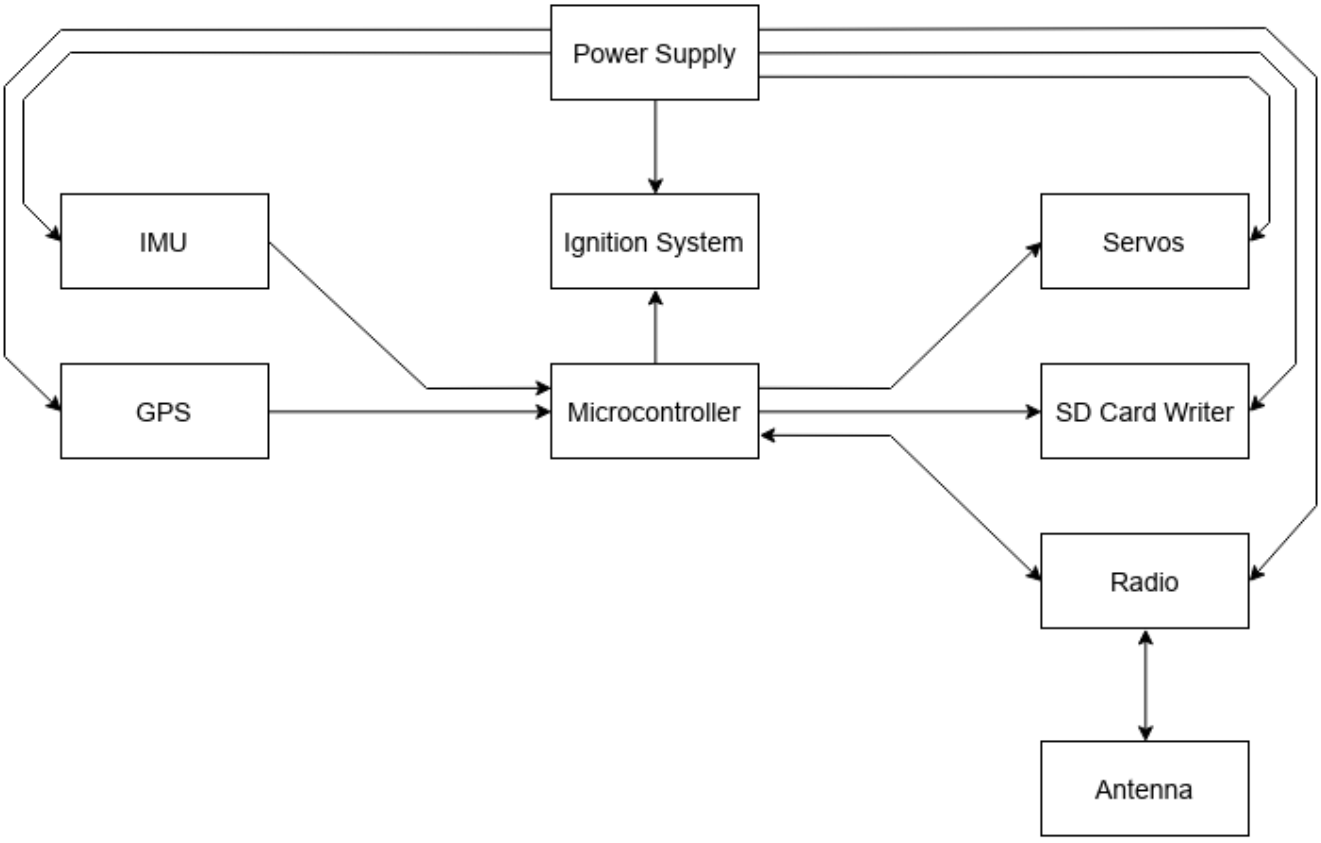

Figure 6: Circuit Layout of the Avionics Package.

4.6 Hardware

The hardware architecture is related to the mapping of hardware to the software and system components of the rocket. This allows hardware and software designers to integrate their independent designs to the hardware aspects. The hardware architecture may seem to be similar to the physical architecture but has details regarding the underlying link between hardware, software and the system. The hardware architecture can be portrayed as shown in the figure below.

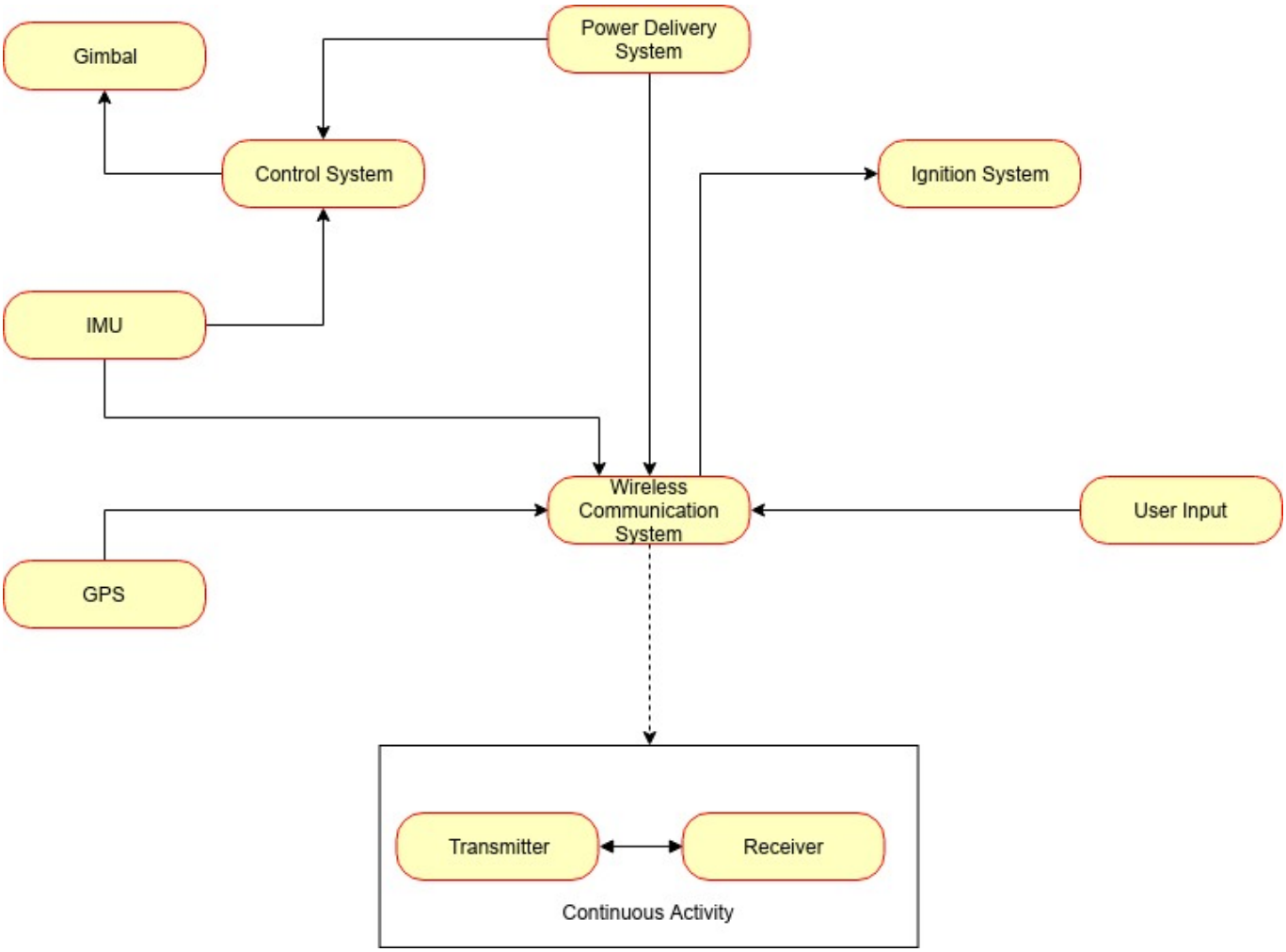

Figure 7: Hardware Architecture layout

The gimbal is used for stability control of the rocket during flight. As such, computations will be performed by the on-board Teensy 3.6 microcontroller as well which will be created by a software designer. The gimbal is part of the controlling system, such that any vectoring correction computations performed by the microcontroller will be sent to the gimbal regarding the rocket's stability.

Power delivery is the central aspect of the hardware architecture as means the difference between the rocket functioning and not functioning. Furthermore, any software design(s) will not function if the power delivery system doesn't exist. Power delivery can be facilitated by a straightforward hardware design or a very complex one. In the case of this system, it is relatively simple as the power delivery aspect is separate from all of the other systems (i.e. Teensy 3.6 microcontroller, gimbal, telemetry). There will be a total of 3 separate lithium-ion polymer batteries supplying power to these systems. The power delivery system is connected to the control and wireless communication systems as well as the microcontroller.

The wireless communication system will have continuous background activity, transmitting information between the transmitter and the receiver. The satellite positioning system and the inertial measurement unit provide information (data) to the communication system which is sent to the base station(s). The home base thus receives the information regarding the rockets flight characteristics. An on-board SD card will be included on the avionics package which also retains this information. The user input is a signal sent to the wireless communication system regarding any actions which are to be only triggered by the user, such as the ignition system. The power being continuously sent to all of the systems before any launch is to be triggered by the user.

Concerns regarding the hardware architecture involve all of the systems. The power delivery and ignition systems should not interact with each other in any way as this will lead to physical damage to the other systems in place. The power delivery system will provide sufficient power to all of the systems (excluding the ignition system) so that there is no failure in the operation of any system. Having three lithium-ion polymer batteries as mentioned previously ensures continuous power to all systems.

The wireless communication system needs to have accurate information sent between the transmitter and receiver. This prevents a loss of information in communications, for example, the information sent is not what was received. Possible causes for this loss of information can be attributed to interference from external communication systems and faulty information being sent from the IMU, GPS, etc.

4.7 Scenarios

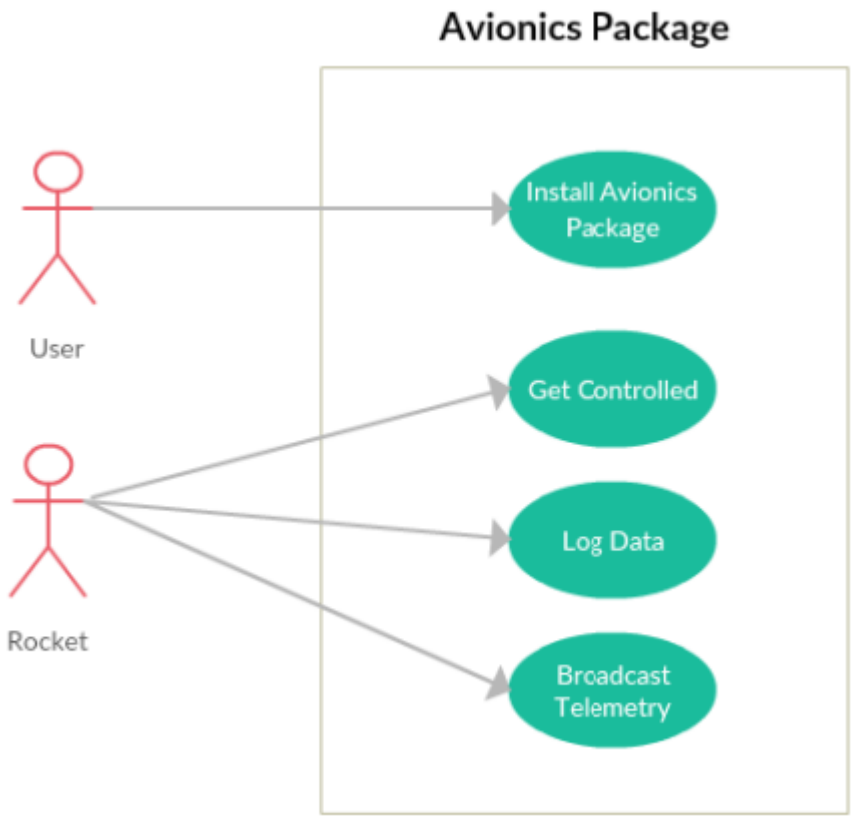

Figure 8: Use Case Scenarios

The primary use case of the avionics package is to provide in flight avionics for a rocket launch vehicle. In this scenario, the system can be viewed as a function which takes sensor data and outputs logging and control. In this usecase, performance of the system is influenced by the rate that the function can execute. This influenced the architectural decision to execute the loop as rapidly as the hardware is reliably able. Once the main class connects to the inertial measurement unit and GPS chip, it starts the loop. Within the loop, data acquired for the IMU and GPS is sent to the data logger which firstly sends data to the SD card writer onboard the avionics package; and secondly sends selected data to the radio module which broadcasts the data over the two antennas onboard the package. This signal is then received by the radio transceiver attached to the base station which will read the data and store it on the base station machine for post launch analysis.

The acquired data is also given to the control system within the main class takes in data and outputs servos commands. The servos move the gimbal and the attached rocket motor. The rocket motor movement adjusts

the thrust vector of the rocket, changing the direction and thus the telemetry data measured by the inertial measurement unit and GPS allowing the loop to repeat. The control system is being treated as a black box because its architecture has not been decided yet.

These effects of executing the loop as rapidly as the hardware is reliably able, mean that everything is run in a single thread rather than multithreading with queues.

A supporting use case of the avionics package is installation on the launch vehicle. The avionics system's physical hardware packaging greatly influences its performance in this use case. Having the fewest number of interconnected pieces of hardware was a concern that influenced the hardware architectural choice to include the teensy, IMU, GPS, and radio on a single PCB.

5. Development schedule

5.1 Schedule

18th of May 2018 - On this date, a launch will commence with the aim to log data and be sure that the software is functioning as required. This will be after the completion of the architectural prototype and thus will be testing the functionality of the current software base of the avionics package.

1st of June 2018 - This launch date will follow the analysis of the 18th of May launch and will start implementing a wireless return communication with the rocket i.e. the user will send a command and log data from the package. A new rocket cone will be printed off and a new PCB design would have been implemented.

8th of June 2018 - on this launch date, the aim is to test the Minimal Viable Product and determine whether or not the requirements have been met.

Architectural prototype

The architectural prototype will be completed by the 14th of May 2018.

Minimum viable product

The minimum viable product will be completed by the 15th of June 2018

Further releases required or specified by the client

The final product will be completed by the 3rd of September 2018

5.2 Budget

As all of the necessary items are already at hand (as well as spares), there will be no need to incur more items. If any of the items were to be damaged or more is needed to be incurred, then the budget is as shown below.

Item	Purpose of Item	Cost per unit

Item	Purpose of Item	Cost per unit
Teensy 3.6 Microcontroller	The Teensy 3.6 is a microcontroller that will be on board the rocket. It will have the ability to record IMU, GPS etc measurements onto the onboard SD card as well as the algorithm required for controlling the gimbal.	As of 5th April 2018, the cost is 40.02 NZD based on a 1USD: 1.37NZD conversion
RFM9x LoRa Radio	The RFM9x LoRa Radio is a 433MHz radio used to send data over long distances. This will allow the avionics package to communicate with the home base station.	As of 5th April 2018, the cost is 27.30 NZD based on a 1USD: 1.37NZD conversion
NCP1117ST50T3G. Voltage Regulator	The NCP1117ST50T3G. is a 5V, 1A Surface Mount Device (SMD). It function is a voltage regulator, capable of having an input voltage range of 7V-25V and capable of having an output current of up to 1.5A. This is used to provide a stable DC supply to the components/devices on board the PCB	0.726 NZD
HC2422T 8.5kg servo	This servo will be used to operate the gimbal based on the gimbal computations performed in the Teensy 3.6 microcontroller. It requires 2.2A@6V - 2.5A@7.4V for a torque range of 7.5kg@6V - 8.5kg@7.4V.	As of 7th April 2018, the cost is 33.25 NZD based on a 1USD: 1.38NZD conversion
Turnigy 1000mAh 2S 30C Lipo Pack	LiPo battery required to power the necessary components of the avionics package such as the servos and the onboard PCB. 3 of these LiPo batteries will be enough for the job and will be able to supply enough power for a reasonable time.	As of 10th May 2018, the cost is 10.45 NZD based on a 1USD: 1.44NZD conversion
MPA-154-ANT L1 GPS	A high-performance GPD embedded antenna. This antenna has a very high efficiency of and can operate in extreme environments. This is required as the rocket in some circumstances may be far from the observer as well as being able to provide positional data.	10.12NZD
FUNcube Dongle	A dongle that connects to the PC and behaves as a receiver for the Radio. Capable of accessing a range of frequencies (64Hz to 1,700MHz).	As of 7th April 2018, the cost is 290.83 NZD based on a 1GBp: 1.94NZD conversion

5.3 Procurement

5.3.1 Hardware

Type	Item	Source	Related Project Goals
Hardware	Printed Circuit Board	PCBWay, PCB Zone or other PCB fabrication company	In order for the avionics package to meet the MVP, a PCB is needed to enable the software to communicate with the hardware and achieve the goals of reading telemetry and writing to an SD card and controlling the gimbal
Hardware	Electrical Components	element14	Power components such as the voltage regulator and reservoir capacitors are vital to the operation of the package.
Hardware	Soldering Equipment	ECS Technicians	Access to soldering irons, solder and solder wick are required for the construction of the PCB
Hardware	3D Printing	ECS Technicians	Any new chassis, gimbals or mounts will need to be printed to be used in the package
Hardware	Bench Power Supply	ECS Technicians	A bench power supply is required in order to test the avionics package

5.3.2 Software

Type	Item	Source	Related Project Goals
Software	KiCAD	KiCad	KiCAD is open source PCB design software that is needed to design the PCB and export Gerber files to be fabricated into final circuit boards.
Software	OnShape	OnShape	Onshape is open source 3D modelling software that is needed to design the chassis and gimbal to be 3d printed
Software	VSCode	Microsoft	VSCode is a free, open source IDE that supports PlatformIO
Software	PlatformIO Ecosystem	PlatformIO	PlatformIO is a free, open source ecosystem for IoT development. Allowing the programming of the Teensy3.6 used in the package
Software	Teensy Arduino Framework	Arduino	Using the Arduino framework sets up a suitable platform for the avionics community to pick up the package and develop it in an open source environment
Software	TinyGPSPlus	Mikal Hart	Provides the use of the GPS Module required to achieve location data from the rocket
Software	SparkFun MPU-9250 9 DOF IMU Breakout	SparkFun Electronics	Provides the use of the MPU9250 IMU module required to add active stabilisation to the rocket

Type	Item	Source	Related Project Goals
Software	Google Test/Mock	Google - GitHub Repo	An open-source software package which provides the testing/mocking framework for Continuous Integration.

5.4 Risks

Risk	Risk Type	Likelihood	Impact	Mitigation Strategies
Power supply incorrectly attached damaging components	Hardware	3	5	A diode will be placed in the circuit by the power pins to short-circuit the power pins if incorrectly inserted
A component or board is damaged to the point where it no longer functions or is lost	Hardware	3	5	Backups of all components will be purchased in case of failure, when boards are manufactured at least two will be made and if possible wired up as a backup
Bad Weather that prevents a launch	Environmental	4	4	All launch plans will be submitted in doubles, the intended date and a backup date within the next few days
No launch can be made in the weeks leading up to the final deadline	Environmental	2	4	A launch plan will be followed to the team's best ability so that if a launch cannot happen leading up to a deadline, the previous launch should not have been long ago so that the data is still relevant
A member of the team cannot attend a lab	Management	5	3	All jobs within the team will have redundancy so that if someone cannot attend a lab the workflow will not be blocked. If a member continues to miss labs the team will confront the member and ask if they could make an effort to attend more labs and if the behaviour does not improve the senior manager will be notified

Risk	Risk Type	Likelihood	Impact	Mitigation Strategies
The member of the team that has the key cannot make a lab or is late to a lab, blocking work	Management	4	3	Before a lab is concluded and the hardware returned to the locker a brief chat will be had regarding who will need the hardware next and the key will be left with them
Files needed for lab work are locally saved to a members computer or H drive and the member cannot make it to a lab	Management	4	3	All files worked on during a lab or otherwise will be uploaded to GitLab as soon as possible to allow all members of the team to view and access them
Bugs in the software cause the package to fail the requirements	Software	3	5	All code will be reviewed by another member of the team before it is committed to the main code base. Regular tests on hardware (not necessarily during a launch) will be used to find bugs before project completion
Other course deadlines prevent a member or members from contributing during a given lab session	Management	5	3	Other course requirements will be discussed with the team as early as possible to give the team warning and other coursework will be done as early as possible so that as much lab time as possible can be dedicated to the project
Control system operates too slowly to properly control the rocket	Software/Hardware	2	4	Regular launches will help in recognising this risk while code reviews and tune-ups will ensure the control system operates at the highest possible efficiency

6. Appendices

6.1 Assumptions and dependencies

- That all circuit board components have correctly labelled values, if the assembly is erroneous, telemetry may be affected.
- All components used are available.
- The servos are without defect, else control signals will be affected.

- The micocontroller must have sufficient processing power, therefore it is assumed that a Teensy 3.6 Microcontroller will be used.
- The GPS can be correctly interfaced with the Teensy 3.6 microcontroller.
- Required Arduino libraries will remain available.
- Communications between components remain reliable.

6.2 Acronyms and abbreviations

- CAA (Civil Aviation Authority): The New Zealand government agency responsible for the rules that govern managing the risk of aviation systems.
- IMU (Inertial Measurement Unit): Electronic device that is used to measure and report the body's force, angular rates.
- UML (Unified Modelling Language): A UML diagram is used to visually represent a system along with its main roles, actions, and artefacts.
- SMD (Surface Mounted Devices): Components used with surface mount boards where the components are soldered to small metal pads on the surface of the board.
- LAN (Local Area Network): A computer network comprised of computers within a limited area.
- WAN (Wide Area Network): A computer network comprised of computers that extends over a large geographical distance.
- GPS (Global Positioning System): A satellite-based global navigation system.
- LiPo (lithium-ion polymer): A rechargeable lithium-ion battery which uses polymer electrolyte, features low weight for an equivalent capacity due to its high specific energy

7. Contributions

Name	**Contributions/Sections**
Mohammad Al-Rubayee	3.2, 4.1. 4.3, 5.1, Table of contents
Ciaran King	4.7, 3.1
Nicholas Lauder	3.2, 4.2, 5.3.2, git fixing
Ikram Singh	4.6, 5.2
Yee Young Angus Tan	1, 3.1, 3.2, 6.1, 6.2, grammar, formatting, rewriting sections of S4, general editing
Angus Weich	3.2, 4.4, 4.5, 4.7, 5.3.1, 5.4
Finn Welsford-Ackroyd	5.3, 4.1