UK CanSat Competition
Team Athena
Sutton Grammar School
Final Design Review
Date: 15/04/24

Part 1: Introduction

1.1 The team

We are Project Athena, a 6-man Year 10 team participating in the CanSat Competition from Sutton Grammar School.

1.1.1 Our roles

Team Lead/Coordinator: Responsible for scheduling deadlines and meetings, task lists and organisation and facilitating and leading team meetings.
Mission/Technical Researcher: Conducts research of mission and equipment feasibility, as well as measurement research.
Outreach Coordinator: Directs the outreach program- assigns members roles for social media engagement and leads the team with outreach initiatives.
Software Programmer: Develops algorithms in code and manages data transmission/reception.
Model Designer: Creates CAD models, designs and 3D prints components of the CanSat.
Electronics Engineer: Selects components and designs/solders the circuits, as well as assisting with coding algorithms running on the CanSat and receiver.
Antenna Designer: Design, tests and makes the antenna.
Ground Station Engineer: Responsible for designing the ground station.

Note: Some members had multiple roles, such as our team lead also being the electronics engineer.

1.2 Mission objectives

1.2.1 Primary mission

Our goals for the primary mission:

- To take temperature and pressure readings once per second
- To send these readings down to ground via the antenna

1.2.2 Secondary mission: Testing suitability of planet for renewable power production

Our goals for the secondary mission:

- To take humidity, light intensity, wind, acceleration and GPS location readings once per second
- To send these readings down to ground via the antenna
- By analysing the data, to test if a planet is suitable for renewable power production and if so, test which types are possible and feasible

The four sources we are testing for are solar, wind, hydroelectric and geothermal. We selected this mission because currently, humanity is on the precipice of revolutionising energy production and use, so we are addressing the explorative requirement for extraterrestrial energy sources.

The example which sparked this idea was when we read an article about Mars being a viable source of geothermal energy and discovered other planets were more suitable for renewable energy than Earth.

Preluding the mission, we had the idea of developing a machine-learning algorithm which would have detected space debris and send images down to ground where they would pass through a deconvolution algorithm to eliminate motion blur. However, during parts researching, we discovered that the four cameras we would require to get a 360 view to detect the space debris were too much for the microcontroller to handle so we abandoned the idea. Luckily, we discovered this limitation early on, which let us minimise time and resource expenditure.

1.2.3 The process of the mission:

1. The leg system (specifics detailed in section 3.1.1) ensures the CanSat's upright landing.

2. CanSat will begin the process of taking measurements.
3. Sensor readings are transmitted back to the ground station and are converted.
4. Our custom-made graph maker allows us to make graphs of this data and draw conclusions.

1.2.2 Measurements, rationale and methodology

Our sensors will take multiple measurements and transmit them down to ground once per second, these readings will also be stored on the CanSat itself in case we lose signal.

Measurements and their uses:

- Light (light-dependent resistor):
  - The light intensity is transmitted back to ground and helps determine solar power viability and effectiveness.
  - It also shows the day/night cycle which also helps determine viability and effectiveness.
- Humidity (DHT20):
  - Can detect if it rains and therefore the availability of water
  - Water is needed to measure if hydroelectric energy is viable.
- Air Pressure (BME680):
  - This is required in the primary mission.
  - We will analyse the effects of air pressure on temperature and see possible correlations.
  - Calculate altitude (formulae detailed later)
- Wind (custom-made anemometer):
  - Calculate wind velocity
  - See if wind power is viable as wind turbines require wind speed within a certain range.
- Temperature (DHT20 & BME680):
  - We can record heat patterns to help scientists make informed decisions about feasibility of power plant production and maintenance.
  - We can see if it is the optimum temperature for solar power.
  - This is required for the primary mission.
- GPS (PA1616D):
  - Allows tracking/monitoring of the CanSat and can be used for tracking the flight path.
  - It will also help us find the CanSat post-flight.
  - Get altitude readings to measure the accuracy of the Barometric Pressure formula
- 9-axis Inertial Motion Unit (ICM-20948):
  - Measures acceleration, rotational velocity and orientation relative to Earth.
  - Gives all around useful information such as if the CanSat is descending correctly.
  - Can monitor the CanSat's orientation.
  - Can monitor the acceleration forces on the CanSat during different launch stages.
  - Will be used to determine the feasibility of geothermal energy as we can detect earthquakes.
- Volatile Organic Compounds (BME680):
  - Acts as extra data that we can draw conclusions on.

A plan of our data analysis can be found in section 5.4.

Part 2: Project Planning

2.1 Project timescale

2.1.1 Meeting schedules

Weekly meetings took place on Monday lunchtime to establish a plan for the upcoming week, with multiple flexible drop-in and online sessions throughout the week during and after school. In those meetings, we discussed role assignments for particular sections, CanSat design changes and proposals, and sometimes even built and designed parts, such as our group effort of parachute sewing and assembly and regular antenna testing.

2.1.2 Task arrangement

We grouped our tasks into work cycles, established at the beginning of each new cycle.
Appendix I shows a high-level Gantt Chart that displays our task schedules between the CDR and the FDR.

2.1.3 Git and GitHub

For an efficient coding process, we used Git for source control. Our code is split into three repositories, one for code on the CanSat, one for the receiver code and another for ground station code. This also ensured easier progress tracking and, through the use of GitHub Projects, allowed us to plan ahead easily. Our software engineers regularly committed code with informative comments (for example, there are currently over 110 commits in the CanSat code repository) and most of our code includes comments to give viewers easier understanding. By using Git, it was easy for multiple people to work on the same project.

Our GitHub: https://github.com/CanSat-Athena

2.2 Team and external support

2.2.1 Team skills

We are a highly dedicated team of 6, and the only team representing Sutton Grammar School for the finals. The majority of us have previous experience from working together on the Big Bang competition, where we were nominated for the National Finals last year and we have a strong team culture and bond between our members. We are very adaptable to sudden changes and we demonstrated this at the start of the project, where we had a thought–out secondary mission idea that would mean we had to spend about a week learning necessary skills for the mission. However, roughly four days in, when running feasibility studies, we found that no equipment advanced enough was compatible with the Raspberry Pi Pico we were using. The team lead quickly coordinated multiple emergency meetings and we successfully managed to repurpose our secondary mission within 24 hours. More recently, the team had a tight timescale to work with producing functional antenna and leg extension systems and testing these due to the deadline for the FDR and the finals, and a large number of parts arriving late. The team worked around it by quickly changing our testing plans to accommodate the late arrivals, including a lot of partial testing. Through these incidents, the team has repeatedly displayed its adaptiveness in urgent scenarios and quick thinking.

2.2.2 Technical skills

The majority of the team have had a strong foundation in STEM-related projects with the 2022-23 Big Bang Competition requiring some of our team to learn an entirely new programming language for the project. This has proved a massive help to achieve our objectives of the secondary mission, which requires processing received data and determining whether the launch field is suitable. Multiple team members can program in C/C++, Java and Python and are always keen on learning new skills. Our lead electronics engineer is highly experienced with the Raspberry Pi Pico and its workings, having done many projects previously.

2.2.3 Limitations

Due to the financial limitations of our school, we did not have access to many resources that some of the other teams did such as a CNC machine suited for PCBs or paid industrial software such as Solidworks. To learn the concepts of engineering our CanSat and mission programming/coding, the team members needed time to learn and master the concepts, as many are A-Level content and all our members are still doing GCSEs. However, the team is brimming with passion and are keen to learn, so this has not proven to be too much of a limitation.

2.2.4 External support

Our team has benefitted from an expansive external network, including many of the team members' family friends being engineers. One of them is a mechanical engineer, and they helped to guide us to the idea of our exterior design of the CanSat and introduced us to the roly-poly mechanics behind the design, though we ended up abandoning the idea after failure during testing. Another, a meteorologist, helped us come up with the methodology for our wind and solar measurements, which we are taking as part of our secondary mission. Along with this, our peers have significantly helped us in the project as well. As one of our team's weaknesses is graphic design, we outsourced to a classmate to help

design our logo (which can be seen on the cover). Also, we worked with an older team from our school who had entered the competition and we traded skills. One of their members was experienced at drawing so she drew our sketches for us and in return, one of our members knew how to use LaTex so he coded their maths equations into their document. One of our friends was also a brilliant article writer and was a part of a young journalist program so he agreed to write an article in a local newspaper and updated our blog with articles about our progress.

2.3 Risk assessment

2.3.1 Physical

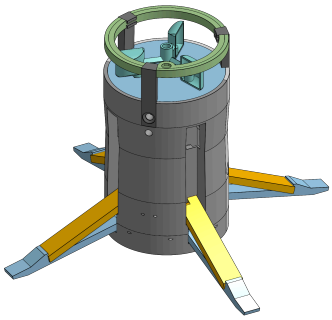
Short circuiting: Exposed wires can contact other parts of the circuit, causing high current draw, breaking the affected components and/or causing unintended side-effects. To mitigate these risks, we securely mounted each component, using JST connectors for connections where possible and using heat shrink tubes/hot glue to isolate exposed electrical connections. In addition, the battery management circuitry on the Pico LiPo has over-current protection.
Soldering/3D printing fumes: Soldering and 3D printing can release toxic fumes that are harmful to the lungs. To mitigate this risk, we soldered in a well-ventilated area, using a fume extractor and lead-free solder. When 3D printing, we printed in a ventilated area and avoided using filaments that produce an excessive amount of fumes.
Soldering/3D printing burns: Solder and 3D printing filament melt at 200+°C, which can cause burns if touched. We avoided contact with the 3D printer nozzle while the machine was in operation and took care when soldering by keeping a perimeter around the solderer where no-one else was allowed.

2.3.2 Technical

Time constraints: We decided to take part in this competition in mid September and we hadn't anticipated the amount of school work we would be faced with. This is because our team has just started their GCSEs and most are taking 11, with some taking 12. This has led to an unforeseen rise in work and this rise will only increase as we approach our mocks. To mitigate this risk we have done two things. Firstly we chose our secondary mission as something which is achievable without the need to dedicate an extraordinary amount of time. Secondly we have also planned our tasks with a generous amount of buffer time which greatly helped when our work suddenly increased.
Anemometer not working: A quarter of our secondary mission relies on detecting wind speed but there is a significant chance that our anemometer will either not detect velocity accurately due to its size or breaking on impact due to its intricate design. This is probably one of the greatest potential risks so we started making the anemometer in mid-December and got a working prototype done by the end of December. Then we carried out extensive tests (see section 3.6.1).
CanSat is prone to breaking: There is a risk that the 3D printed parts may be damaged during the launch. To mitigate this risk, we are using polycarbonate printed at a high infill level and 3+ perimeters. We also designed the parts in a way that allowed them to be printed in an orientation where the stress applied is perpendicular to the layers of the 3D print, lowering the chance of layer separation.
Defective Components: There is always a risk that a part we order comes with an issue, therefore we will have spare sensors on the launch day and have designed the electronics in a way that lets them be replaced quickly by using JST-PH connectors where possible (see section 3.2.3).
Antenna Signal disruptions/transmission failures: This is perhaps one of the most concerning risks of all, as transmission failures would hugely hamper the success of the missions. As such, we have done thorough testing in order to make sure the antennas can successfully transmit signals to and from the CanSat and ground station (see 3.6.5). Furthermore, we are storing the readings on the CanSat to avoid losing data if the signal is disrupted (see 3.3.1.3).

Part 3: CanSat design

3.1 Mechanical design

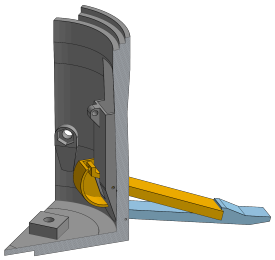


CAD model of the CanSat

3D printed model

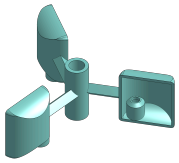
Note: the CAD model is available to view in 3D at https://cansatathena.com/cad-model.

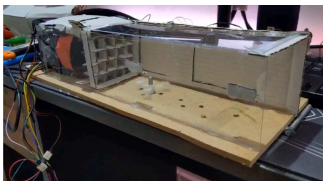
3.1.1 Leg system



Our leg system is a system powered by rubber band elasticity. Our initial design used motors to extend the legs but this was replaced as the team believed there is a higher chance of success using the elastic bands. We also made the change from 3 legs in our initial design to 4 legs in our final design due to 3 not being enough to land the CanSat upright with a high success rate. The legs extend as soon as the CanSat leaves the rocket due to the walls of the rocket not holding it together anymore.

3.1.2 Anemometer





A key part of our secondary mission is that we will be measuring the wind speed. To do that, we are using an anemometer. It is 15 mm tall (we have increased it by 3 mm since CDR) and has rectangular cups to make the most of the available wind power. We had to 3D print ~15 different anemometers and measure using a custom-made mini wind tunnel (see image) to get the cup width and depth just right to get the maximum possible RPM. To be able to measure the rotational speed of the anemometer we have small cylindrical magnets on the bottom of each cup and a hall effect sensor to sense the magnets (see section 3.2.1.7).

3.1.3 Materials

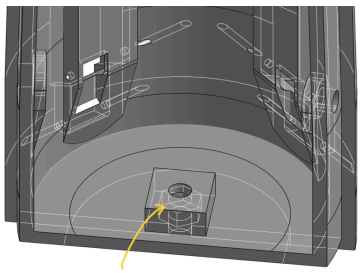
We chose polycarbonate filament to 3D print the main body and leg system due to its high strength compared to other filaments; it is used in bulletproof glass. The polycarbonate filament has high impact resistance which will be helpful in maintaining the structural integrity of the CanSat when it lands and help it survive the high acceleration forces during launch. It is also relatively cheap compared to other filaments at £22.00 per kilogram. Furthermore, it has strong layer adhesion, which means that it has high tensile strength perpendicular to the layer lines as well as parallel. On the other hand, there are some disadvantages to polycarbonate filament too. It has a tendency to warp and clog the nozzle while printing, making it difficult to 3D print. The warping greatly reduces the dimensional accuracy of the print and results in a deformed structure; we even had to re-print some parts again. Finally, it needs a very high printing temperature (270+°C hotend and 100°C bed temperature) and the 3D printer we were using could only go up to 255°C, therefore our electronics engineer had to upgrade its hotend, allowing us to print at up to 300°C. To prevent warping, we built an

insulating enclosure to keep the print warm. As the warping is caused by rapid cooling, when it is kept warm it takes longer to cool down, meaning less warping. To avoid nozzle clogs, we printed the model at 275°C.

3.1.4 Structural strength

We are using an M4 threaded rod which runs directly through the centre of the CanSat as a 'spine' to provide strength. As metal has much higher tensile strength than plastic, having the parachute connected to this instead means that the metal will take most of the force instead of the plastic. This ensures that the CanSat doesn't break in half due to the high acceleration forces when the parachute opens.

When we 3D printed parts, we also used higher extrusion width, meaning that more filament is pushed through when extruding a line and as a result the line is thicker. This means that the extruded plastic is pressed down with more force, improving layer bonding.

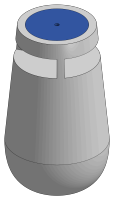


3.1.5 Embedded prints

In addition to previous choices, we have adopted a special printing method that would allow us to embed nuts and other metal parts into the 3D print. During printing, the 3D printing process will pause halfway in order for a nut to be placed in. Then, the 3D print will resume and print over the nut. This would secure our CanSat more effectively by locking small parts into place and also increases our CanSat strength as a whole. We have used this technique in the bottom part of the main body, to hold a nut securely in place.

3.1.6 Design changes

Our design first started off as a cylinder with a hemispherical bottom. We planned to have this bottom section weighted so it would roll up right like a roly-poly toy. This idea came from a mechanical engineer who told us to stick with a principle of minimising the moving/mechanical parts of the model to minimise the chance of failure. This is why we went with something that just relies on gravity instead of motors to extend legs. The model then had slightly sloped sides which converged into a circle. There were connectors creating a gap near the top to allow space for the anemometer.

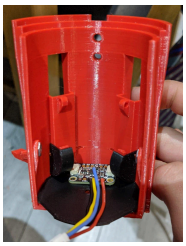


However after printing this out we realised the roly-poly system was not reliable enough and the hemispherical bottom took up too much space. That's why we switched from that design to our current leg design but we still stuck with the principle that we wanted to minimise the amount of mechanical parts hence our leg system is based on elastics not motors. Another key change which we did was the top blue section has been removed so it is hollow from the top. This means the anemometer has more exposure to the air and is more accurate. However we still kept the outer part of the top section to stop the paracord getting caught in the anemometer, preventing it from spinning.

Our design also slightly changed afterwards as we realised the legs kept getting stuck onto the internals, hence we added a leg protector to solve this issue. We then added a plate at the bottom to hold the weights as we needed to make sure our CanSat weighs enough.

3.1.7 IMU Plate

We have also had to add a "plate" for the IMU to sit on as it needs to be horizontal and aligned with the X, Y and Z axes of the CanSat to measure properly. It is printed in PLA as it is not a structural part of the CanSat and sits on top of the bottom embedded nut of the main body. The gap under the IMU plate can then be used to place weights since we determined that the CanSat is under the weight limit.

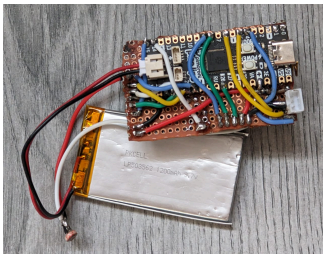


3.2 Electrical design

The main schematics are available at https://cansatathena.com/schematics/. The receiver schematics are available at https://cansatathena.com/receiver-schematics/.

We decided to use I^2C sensors where possible over others, meaning we could use a single I^2C bus to connect to them. This allowed us to reduce the number of wires and, as a result, reduce weight and space used.

3.2.1 Components



A list of the components and prices is available in Appendix A.

3.2.1.1 Pico LiPo (16MB)

We are using the Pimoroni Pico LiPo as our microcontroller. This is for a few reasons. It is small and lightweight which is crucial in a space and weight limited mission like this. The size also gives more room to play around with when deciding where internals go. For its size, it also has a lot of pins which is necessary as we are using plenty of sensors. We are using this instead of the regular Pico as it has a 8x greater flash memory which will allow us to store many more readings. It also allows for the use of a LiPo battery without a separate battery charging circuit, which would take up valuable space and we are already space-constrained due to the anemometer.

3.2.1.2 BME680

We had a choice between this sensor and BMP280 as they both measure pressure. Initially we were going to choose the BMP280 as it was already included in the pack however the BME680 also measures VOCs which opens a lot of doors for analysis in the Primary Mission. This swap also doesn't lose any accuracy, both measure pressure with ±1 hPa accuracy and the dimensions are identical so it doesn't affect our arrangement. This sensor also measures humidity and temperature too, so we can use it as a backup if the DHT20 fails. It uses I^2C for communication.

3.2.1.3 Adafruit PA1616D GPS

There were many options for the GPS sensor but we chose the PA1616D for multiple reasons. Firstly it has a coin cell slot which eliminates "cold starts" which saves us roughly 30 seconds every time we start the GPS. However, in the end we decided against using the coin cell as we are space-constrained. The PA1616D module also has a very high maximum velocity, 515m/s, and has 3V logic level making it compatible with the Pico. Its accuracy is quite high with it being accurate to ± 3m. This is one of the most energy demanding sensors we used but that is common with all GPS sensors and this has one of the lowest current draws with it being only ~29mA during navigation. Like most GPS sensors, it uses UART for communication, with a default baud rate of 9600.

3.2.1.4 DHT20

This is an I^2C version of the popular DHT22 sensor, based on the AHT20 chip. Even though we already have a temperature and humidity sensor (the BME680 also senses humidity and temperature in addition to pressure and VOCs) this has higher accuracy and precision, so given that we have the budget we decided the extra space it takes is worth it. However, we had to remove the plastic cover around it to reduce its space usage.

3.2.1.5 ICM-20948

The main reason we chose this as our Inertial Motion Unit (IMU) was that it is very cost effective, especially for a 9-axis IMU at only £14.70. It is also very small, accurate, uses I^2C for communication and can measure up to 16G which is plenty for the mission. This allows us to measure the acceleration, rotational speed and exact orientation in all 3 axes.

3.2.1.6 Light dependent resistor

It was between this or a photodiode and we chose the LDR because it was cheaper, lighter and smaller. However this comes at the slight cost of speed (photodiodes take mere nanoseconds to adapt to change in light intensity whereas LDRs take up to 50 ms to update) which isn't that significant as we don't need to take readings at that speed.

3.2.1.7 Hall effect sensor

As explained in section 3.1.2, we will have magnets on the bottom of the anemometer. Therefore, we need a sensor to sense the changes in the magnetic field. For this, we have two options: reed switches and hall effect sensors. However, hall effect sensors are the clear winner here for three reasons. Firstly, reed switches are made of glass, which makes

them extremely fragile and prone to breaking, especially since it will be shot up 300m in the air and will experience upwards of 20G. Secondly, they are mechanical and mechanical sensors usually have a much lower lifespan than solid state ones as the metal inside can wear out over time. Thirdly, reed switches are slower to switch which can be an issue if the wind intensity is high. To measure the number of times the magnets pass over the sensor, we are using interrupts that are triggered on the rising edge of the signal from the sensor. Once the interrupt is triggered, an integer variable is incremented and every second, this variable is reset to zero and the old value is recorded as the 'third of a revolutions' (as the anemometer has 3 cups) per second.

3.2.1.8 RFM96W

We are using the RFM96W LoRa radio module which uses SPI for communication. We are doing this for a multitude of reasons. Firstly its range is up to 2km with basic antennas if there is a direct line of sight and this is plenty for our purpose. It also has the added advantage of being included in the starter kit so we didn't need to deal with placing an order and accounting for its price. However it did have the setback of only being able to send 252 bytes per packet and our data being sent per second was around 628 bytes per packet. However, by using compression techniques, we managed to reduce the size down to 148 bytes (see section 3.3.1.5).

3.2.3 Modularity

If a component breaks on the launch day, we will have at least one replacement ready. However, replacing components still requires desoldering/resoldering wires correctly, testing that everything is connected correctly and there aren't any short circuits - it takes a lot of time. This is why we decided to use modular connectors - we have soldered JST-PH connectors to components where possible which will make swapping them effortless and easy in the case that they break. This means we also will not have to check the connections and check for shorts, making the process extremely fast. In the CDR we had stated that we would solder connectors onto every component; we were not able to do this due to space constraints and instead opted to solder the components onto a piece of perfboard to minimise space usage.

3.2.4 Receiver

The ground station receiver schematics is the same as the CanSat, except without any of the sensors. We have soldered this onto a piece of perfboard, similarly to the CanSat, to avoid wires getting loose.

3.3 Software design

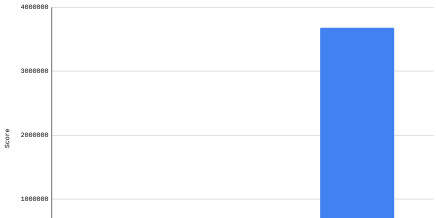
Note: All of our code is open source and is available here: https://github.com/CanSat-Athena/CanSat, here: https://github.com/CanSat-Athena/receiver and here: https://github.com/CanSat-Athena/data-processing. The diagrams used (in .drawio format) are available at https://github.com/CanSat-Athena/Diagrams.

3.3.1 Onboard the CanSat

Readings stored on the Pico	Readings sent over radio
Timestamp (milliseconds since startup) DHT20 - temperature and humidity BME680 - temperature, humidity, air pressure, gas resistance (VOC levels) ICM20948 - (5x) accelerometer XYZ, gyroscope XYZ, magnetometer XYZ LDR - light intensity Anemometer - wind speed PA1616D - latitude, longitude, altitude, time, fix status	Everything in the other column Battery voltage Battery percentage Filesystem usage

3.3.1.1 Language choice

For our language, we had two options: C++ or MicroPython (or its easier-to-use fork, CircuitPython). In the end we decided to choose C++ for a few reasons. Firstly, it runs at a significantly higher speed (up to 46x faster than MicroPython and 69x faster than CircuitPython – see bar chart – which is very useful when taking real time measurements as we do not want to waste any processor cycles. Secondly, C++ is a compiled language compared to MicroPython being interpreted. This means that nearly all errors in C++ are caught at compile time; this is vital in a mission where errors causing the code to crash could be catastrophic. Due to being constrained to a



microcontroller, memory is a constraint with us only having access to 256 KB of RAM. This is where again C++ comes ahead of MicroPython as it is memory efficient since it gives control over the memory to the user with the use of features such as pointers. However, this is also a double-edged sword as it is much easier to create memory leaks in C++ – with great power comes great responsibility. To combat this, we have carefully crafted our code to mitigate this issue and peer-reviewed the parts where we used pointers. Our programming and electronics lead has written a comprehensive article about the points above on our blog here: https://cansatathena.com/c-vs-py.

Finally, due to the need of having to schedule different parts of the code at precise intervals to take precise readings, a tool which helps with that is greatly beneficial. This is where C++ opens the door for us to use an RTOS (Real-Time Operating System). There aren't very many practical real-time operating systems in the MicroPython ecosystem, therefore we must use C++ to be able to take advantage of such tools. This allows us to multitask easily even with just the 2 cores of the RP2040 chip and have a different, separate task for each component of the code, all running concurrently with FreeRTOS' powerful task scheduling system. Real time operating systems are used in NASA missions and practically all space missions which speaks to their usefulness.

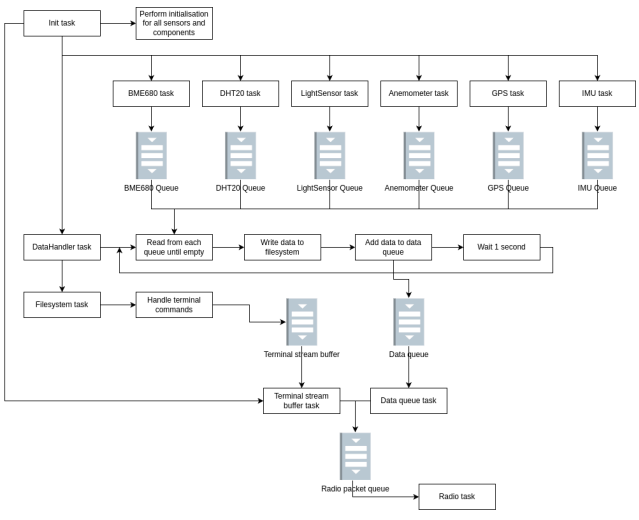
The fact we are using C++, an OOP (Object Oriented Programming) language, also gives us many advantages. Firstly, given our mission requires accurate data readings, the data corruption must be kept to a minimum. This is where one of the fundamentals of an OOP comes to play, encapsulation. By encapsulating, it makes it less prone to bugs. However the biggest advantage is that it makes the code organised and modular. Through the use of classes (and sometimes structs) the code is broken down into independent sections allowing for easy troubleshooting. It also makes the code flexible and allows us to add code very easily and not have to worry too much about breaking everything. By using inheritance, a lot of time is saved as code can be reused. An example of this is when our software engineer wrote just one function for reading data on the I^2C bus and a parameter with the sensor's details instead of having a separate function for each sensor. Even though Python has classes, C++ has a better implementation of them.

3.3.1.2 FreeRTOS design

We have decided to use an RTOS due to their ability to multitask easily, allowing us to run multiple tasks concurrently. Out of all available RTOSes such as Zephyr and embOS, we chose the FreeRTOS kernel due to its stability, simplicity, relative ease of use and popularity, meaning it is easier to find documentation.

As we are using an RTOS, we have designed every component of the code as a separate task, using queues to interface between them. Firstly, when the CanSat is turned on, the initialisation task runs. Inside this task, the `init()` method of each component's class is called and the tasks of the components are started.

As each sensor task performs a similar task - read data, wait for an amount of time, repeat - we have made a single task function to handle all the sensors, called `sensorReadTask`. Each sensor still has its own task, however as the FreeRTOS method `xTaskCreate` allows a pointer to be passed as an argument to the task (`void* pvParameters`), when we are creating the task we can use the same function for all sensors and instead pass a pointer to a struct (another reason to use C++) that contains data about the individual sensor. The sensor data struct has data such as the instance of the Sensor class, the FreeRTOS queue to send the sensor data to and the measurement delay period. In the sensor read task, there is an infinite loop; inside the loop the read function of the sensor class (as said, passed as an argument) is called, the returned data is added to the FreeRTOS queue (again, passed as an argument) and the task sleeps (`vTaskDelayUntil`) for the sensor's delay period. Implementing it as such instead of having a separate task function for each sensor means that it requires relatively little change to the code to add or remove a sensor, making the code modular. Another advantage of this is that if we want to make a change to the function because of, for example, a bug, we only need to make changes in a single place instead of six.



Each sensor's task adds its data to its queue and we need to send and store that data. Therefore, we need a task to handle that data, which is exactly what the data handler task is for. It is responsible for reading the data from the queues, storing it in the filesystem and sending it over radio to the ground station.

A major change we have made since the CDR is that we have switched to static allocation for FreeRTOS where possible, for example by using `xTaskCreateStatic` instead of `xTaskCreate`. This has a major advantage – memory is allocated at compile time, not runtime. This means that, whilst dynamic allocation can fail if the Pico is out of memory, static allocation simply cannot, by design. Even though it is unlikely that we will run into a memory issue, this is still important to have.

An enlarged version of the above diagram showing program structure is at https://cansatathena.com/freertos-fdr/.

3.3.1.3 Filesystem and data storage

On the somewhat likely occasion that the signal is briefly lost, in order to avoid losing any readings the data is stored onboard the CanSat, allowing us to retrieve and analyse it after recovery. We had briefly considered using an SD card but concluded that it would add additional cost, space and complexity to an already very complex system and that the 16 MB of onboard flash storage of the Pico LiPo is more than enough, assuming we can make use of it efficiently.

When we were implementing data storage on the CanSat, we considered storing the data on the raw flash of the Pico, however as that would be difficult to manage, provide no flash wear levelling and had little to no power loss resilience we decided to instead use a filesystem. We did have a few options such as FatFS and SPIFFS but in the end we decided on LittleFS due to its excellent dynamic wear levelling, unparalleled power loss resilience and small RAM and ROM usage, making it excellent for this type of embedded applications.

To store the data efficiently on the Pico, we had to think outside the box. Our first plan was to store the data in a CSV-like format – here is a sample line:

```
[360502],[27.342033,38.079166],[26.861551,36.603718,100091.320312,105015.664062],[[0.003784,0.922485,-0.351501,-0.213740,0.290076,-0.290076,-17.250000,54.750000,-5.250000][0.004822,0.918884,-0.353760,-0.290076,0.221374,-0.366412,-17.400002,55.949997,-6.150000][0.004578,0.920715,-0.355896,-0.167939,0.251908,-0.305344,-17.099998,53.699997,-6.450000][0.007812,0.916138,-0.353027,-0.267176,0.213740,-0.312977,-17.849998,55.949997,-6.900000][0.006836,0.918274,-0.354492,-0.274809,0.267176,-0.335878,-18.299999,56.250000,-5.550000]],[1050,],[0,],[0.000000,0.000000,0.000000,18,20,47,0]
```

Every second, this amount of data would be appended to a file on the Pico's filesystem. However, if we include the newline character at the end, this is 582 characters long. This means that it would be using over 2 million characters, or 2 MB of data every hour. This means it would fill up within 8 hours, although if we also considered the filesystem overhead it would be closer to 5 hours. This is usable, however it is not great.

A much better way to do this is to use a data structure and store the binary form of it. For example, assume we need to store four values: the timestamp, GPS latitude, longitude and altitude. The GPS values have a float data type and the timestamp is an unsigned 32-bit integer. This is how a struct that holds those items of data can be created in C:

```
typedef struct GPSData {
    uint32_t timestamp;
    float latitude;
    float longitude;
    float altitude;
} GPSData_t;
```

In the CSV-like format we were using previously, this would be formatted as:

```
4294967295,27.342033,-90.594452,91.320312
```

This is 42 bytes long if we include the newline character at the end. However, the struct above holds the same data and is only 16 bytes long. Furthermore, it does not require a new line character and is much easier to parse – when the data needs to be read from the disk, the nth reading can be read by simply moving the read cursor to $16(n - 1)$ and reading 16 bytes into a GPSData_t data structure. There is no additional parsing required.

We calculated that, when using this new method, the amount of data stored every second is 148 bytes, allowing us to calculate how long it would take before running out of space. Excluding filesystem overhead, 533 KB is used per hour, which would mean it would take just over 30 hours of operation to fill the disk up. Taking the filesystem overhead into account, we calculated that this number would be closer to 20 - still much more than required.

In order to avoid overwriting old data if the Pico resets due to an unlikely momentary loss of power or a crash, the microcontroller creates a new data file on each reboot, thus avoiding overwriting the old one. When the Pico is switched on, it reads, then increments the value of a file called boot_count. This file contains the number of times the microcontroller has been booted. The boot count is then used to create the data file. The data file is called data_[boot count]_[file count]. For example, the file created on the 8th boot would be called data_8_0. The file count is used as a simple pagination system. Every time the file size exceeds 32 KB (approx every 220 readings), the file count is incremented and a new file is created. For example, the file created after data_8_0 would be data_8_1. This lets us avoid having to read hours of data when we only require a small part.

3.3.1.4 Fail-safes

- We are using the Pico's built-in watchdog feature (as described in section 4.7 of the RP2040 Datasheet here: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf). A watchdog is a countdown timer that resets the RP2040 when the value reaches zero. During normal execution, the watchdog is updated periodically (by calling `watchdog_update()`), however in the case of a rare event such as a FreeRTOS kernel crash, the Pico will reset. This is vital to ensure the CanSat will always remain operational, even if there is an unrecoverable crash.
- No data is overwritten on a reset because a new data file is created on each boot, as explained above.
- Vital tasks have a higher FreeRTOS priority than less important ones, for example the data handler task (handles ground communications and storage) has a priority of 3, whereas sensor read tasks have a priority of 2. This ensures that if one of the sensor read tasks is stuck in an infinite loop, the more important data handler task will not be stuck in a blocked state.

3.3.1.5 Data transmission

In our code, there are two types of packets: data and terminal. The data packets contain the data, formatted in struct form (see section 3.3.1.3) to save bandwidth as LoRa has extremely limited bandwidth, especially when parameters such as the coding rate and spreading factor are tuned to maximise range. This also has the advantage of letting us change the format it is printed to the terminal in without changing the amount of data transmitted. The other type of packet is a terminal packet. It contains logs from the CanSat in plaintext form. We have also added a 3-byte team header of "ATH" to the start of the packet to avoid mixing up packets with other teams.

The packets are structured as follows:

- Team header, 3 byte array of characters, "ATH"
- Packet type, 1-byte character, 't' for terminal (logs) or 'd' for data packet
- Packet body length, 8-bit unsigned integer, 0 to 255
- Packet body, array of characters

3.3.2 Ground station receiver

As described above, there are two types of packet: data and terminal. As we wanted to keep these two separate, we decided that the best way to do so was to use two serial ports; one for data only and the other for terminal logs. However, by default, the Pico only has one serial port. To get around this, we had to use TinyUSB. More specifically, we had to use its Communication Device Class (CDC) stack. This allows the Pico to send a dual CDC device descriptor, meaning it creates two virtual serial ports.

When a packet is received, the G0 pin of the radio module pulses high, triggering an interrupt and causing the Pico to read and process the packet. When processing, if the packet is a data packet, the packet body is cast into a data struct. Afterwards, the fields of the data struct are printed to the second serial port to be processed by the ground station. If the packet is a terminal packet, it is printed in the form received to the first serial port.

3.3.3 Ground Station

3.3.3.1 Language Choice

For our ground station language, we had two main choices, C++ or Python. In the end we decided to go with Python for a few reasons. The main uses of the ground station were to parse data, store it into a database, show a live updating GUI and create a graph maker. This is where Python's massive ecosystem becomes useful as it has libraries for every

use listed. To parse the CSV sent it had a csv library, storing it in a database was made easy with the sqlite3 library, Tkinter made making a live GUI seamless and MatPlotLib streamlined the making of the graph maker. This ease of use allowed our ground station engineer to focus more on the logical parts of the ground station instead of language-specific features. These features would have been possible to implement in C++ however there would be unnecessary complexity added. One advantage of C++ that was mentioned earlier in 3.3.1.1 was its speed compared to Python. We considered this advantage but decided it was not enough to switch to C++ as the ground station does not have to have the speed our CanSat itself does because all the data is stored in a text file before we parse so even if it is a lot slower, our mission does not get impacted.

3.3.3.2 Library Choices

The main thought process we had in our head when selecting the libraries was that they should be easy to use because our ground station is quite small in terms of features so we wanted to limit the complexity that big, powerful libraries brought and instead go with something simpler that our ground station engineer can easily pick up.

As mentioned earlier, we decided to use SQLite3 as our database. It was chosen over something like mySQL for multiple reasons. Primarily because it is light-weight and easy to use with Python. As our system only had one user (the team), the complexity that mySQL brought was unnecessary. This paired with the fact that SQLite3 required practically no setup and was ready to use almost immediately made it the obvious choice.

For our graph maker, we chose MatPlotLib. This was again because of its ease to use and its comprehensive documentation, allowing our ground station engineer to pick it up really quickly and not spend too much time having to deal with syntax.

Our live GUI was made with Tkinter as it is recommended for people who have not used GUI software before like our ground station engineer. As our GUI was small scale and only meant for us to see, Tkinter was more than enough to handle our needs.

3.3.3.3 Data Parser

To parse the CSV sent, our software engineers have written a simple script which reads through every row of the csv and stores the data in a global list. The first step in this process is the script jumps to the last parsed line in the CSV to stop rows from being parsed twice. Then it carries out a preliminary misprinted line check (see 3.3.3.5). If that check has been passed, it reads through every single element in the row and since it is a CSV, elements are separated by a comma. If the element contains a '[', then a variable called inSection is made true as we now know we are parsing a new section of data, each section contains all the data from one sensor. Then a variable called dataCounter is made 0 and for every element inside the section, it is increased by one. As mentioned before each sensor's data is enclosed by square brackets and the different data inside is stored in a periodic pattern. For example, the DHT20 data is stored as [dhtTemp, dhtHum, dhtTemp, dhtHum...]. To see what specific reading the data is, we do the data counter modulo the number of distinct items in that section (a hard coded value). Using the section counter and the data counter modulo the number of distinct items, that data element is placed in a specific place in the global list mentioned before. Then once an element contains ']', the data counter is set to 0 and the process repeats.

3.3.3.4 Conversions

Once the data is in a list, a script carries out many conversions on this data.

Firstly, it converts the air pressure into altitude using the *Barometric Pressure Formula*.
This can be seen below:

$$h = 44330(1.0 - (\frac{p_m}{p_r})^{0.1903})$$

Where:

h: Height at measurement
p_m : Measured Pressure
p_r: Reference pressure

This gives us the estimate height relative to the height of p_r so what we have implemented is that once readings are being sent, there is a calibrate readings button where once pressed, records and stores multiple of the current readings, one of which being the current pressure and that value is then use as p_r so we get an altitude relative to the ground.

It then converts the hall effect sensor trigger readings to wind speed. This is done by firstly dividing the triggers by 3 since there are three magnets on the anemometer. We then multiply that value by the circumference of the anemometer to get the distance. This is then divided by the time between this reading and the last reading to get the speed of the anemometer. However this is not the exact wind speed but it is proportional to the wind speed. So we used an anemometer at school and used a hair dryer at different settings to make our anemometer and the school anemometer experience the same speed. We then plotted this data and then found the line of best to get:

$$V = 1.27(v) + 5.4$$

Where V is the actual wind velocity and v is the anemometer speed.

This is the wind speed at the height of the CanSat but for analysis, we need the wind speed at roughly 90m up. For this we are using the *Logarithmic Sheer Formula (*as the height we want is less than a 100m*):*

$$v_1 = v_2\left(\frac{\ln \frac{h_2}{z_0}}{\ln \frac{h_1}{z_0}}\right)$$

Where:

- v_1: Velocity at new height
- v_2: Velocity at measured height
- h_2: The new height we want
- z_o: The roughness value (detailed later)
- h_1: Measured height (the distance from the anemometer to the base)

The roughness value is taken from a table Appendix B and will be decided on the launch day after quickly analysing our surroundings.

From our IMU readings, we will calculate the orientation of the CanSat relative to earth. This is achieved by the ImuFusion library which uses a modified Altitude and Heading Reference System (AHRS) algorithm and calculates the quaternion, linear acceleration and Earth acceleration of the CanSat. This algorithm is based on part of Sebastian Madgwick's PhD thesis and is a modified version of the Madgwick Filter. This data can be used to determine whether the CanSat has landed upright and other such useful stuff however due to time limitations we have not been able to implement this.

A link to the Fusion GitHub repository can be found here: https://github.com/xioTechnologies/Fusion

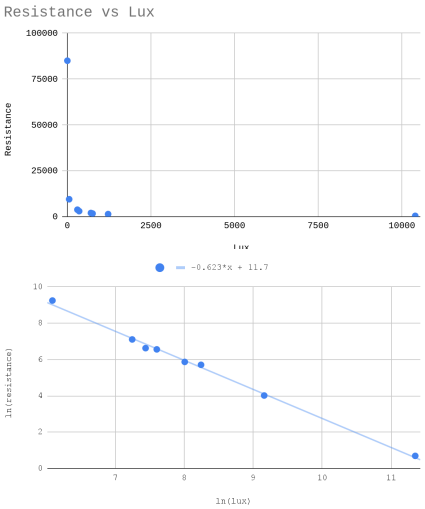
One of our largest challenges was calibrating our LDR value to get an output in Lux.. This required many different steps to carry out:
Firstly, the LDR transmitted a value from 0 to 4095 (the Pico has a 12-bit ADC). This corresponds with the voltage (0 being 0V and 4095 being 3.3V) so we converted this into voltage by dividing the value by 4095 and multiplying by 3.3 (Pico VSYS voltage) to get a value in volts. We then converted the voltage into resistance using the voltage divider formula and rearranging:

$$V_{out} = V_{in}\left(\frac{R_1}{R_1 + R_2}\right) \Rightarrow V_{out}(R_1 + R_2) = V_{in}(R_2) \Rightarrow V_{out}(R_1) + V_{out}(R_2) = V_{in}(R_2) \Rightarrow V_{out}(R_1) = V_{in}(R_2) - V_{out}(R_2) \Rightarrow R_1 = \frac{V_{in}(R_2) - V_{out}(R_2)}{V_{out}}$$

In our case:

- V_{out}: The voltage calculated from the LDR value
- V_{in}: Pico VSYS voltage of 3.3
- R_1: The resistance we want to calculate
- R_2: The potentiometer resistance (measured to be 7450 ohms)

Now we have the resistance, we need to transform this into a lux value. To do this we had to use a lux meter borrowed from our school. We exposed the lux meter and the LDR to the same level of light and recorded the lux from the lux meter and resistance from the LDR.
To figure out a rough estimate for the equation which mapped Lux onto Resistance, we took the natural log of both sides and plotted a graph.



As this is a straight line, we can now easily figure out the function as seen as below:

$$\ln(Resistance) = -0.6230 \ln(Lux) + 11.7 \Rightarrow \ln(Resistance) = \ln(Lux^{-0.623}) + \ln e^{11.7} \Rightarrow \ln(Resistance) = \ln((e^{11.7})Lux^{-0.623}) \Rightarrow Resistance = (e^{11.7})(Lux^{-0.0623}) \Rightarrow \frac{e^{11.7}}{Resistance} = Lux^{0.0623} \Rightarrow \frac{e^{11.7}}{Resistance}^{0.623} = Lux$$

This formula is then used to calculate the lux. This lux is then converted into solar radiation by multiplying by 0.0079 (assuming the light is from the sun).

To calibrate our IMU readings, we have a button called 'Calibrate Readings' and it takes and stores the average of the different accelerometer and gyroscope over a 1 second period. This button will be pressed when the CanSat is in an upright position and still. It then subtracts these values every time from new accelerometer and gyroscope readings.

3.3.3.5 Error Handling

To handle errors in the transmitted CSV, the data parser carries out three different checks.
The first check is a preliminary check where it counts the number of square brackets in each row and if it is not 20 (for radio parsing) and 14 (for file parsing) it rejects the row, this accounts for misprint errors.
The second check is a data amount check where it sums the amount of elements in each section of data and if it's not divisible by the amount of distinct items in the sections (hard-coded values) the row is rejected.
Finally, a range check is carried out by making sure the temperature readings are between -40 and 80 and the humidity readings are between 0 and 100, if they are not the readings are not added to the database.

There is also the low possibility that the Pico crashes mid-mission, and that will reset the "milliseconds since boot" reading which will throw off our data. To combat this, the script always stores the last milliseconds since boot and if it is greater than the new milliseconds since boot, there is a global variable called offset and it sets offset equal to the last millisecond since boot and adds offset to every new timestamp reading.

3.3.3.6 Database Design

We have two databases, one for radio readings and another for file readings to avoid duplicate readings. In the radio readings database, there are 6 tables: IMU readings, Regular Readings, Running Average Readings, GPS Readings, Flight Checks, Calibration Readings
The list of readings in each table can be seen below:

IMU Readings	Regular Readings	Running Average Readings	GPS Readings	Flight Checks	Calibration Readings
accelX, accelY, accelZ, magX, magY, magZ, gyrX, gyrY, gyrZ	dhtTemp, dhtHum, bmeTemp, bmeHum, bmePres, bmeGasRes, ldrValue, ldrLux, ldrSolarRadiation, hallEffTrigs, windSpeed, windSpeed90, batVolt, batPerc, fsUsage, fsSize, RSSI, alt_calc	dhtTemp, dhtHum, bmeTemp, bmeHum, bmePres, bmeGasRe, ldrVoltage, ldrLux, ldrSolarRad, windTriggers, windSpeed, windSpeed90	lat, long, alt, fix	hasLanded, hasDeployed, hasLaunched	accelXOffset, accelYOffset, accelZOffset, gyrXOffset, gyrYOffset, gyrZOffset, baseAlt, basePre

The format of each table can be seen below:
IMU Readings: name, value, timestamp
Regular Readings: name, value, timestamp
Running Average Readings: name, value, timestamp
GPS Readings: name, value, timestamp
Flight Checks: name, hasHappened, timestamp
Calibration Readings: name, value, timestamp

The Running Average Readings sums up all the previous data of a certain reading in the Regular Readings table and then finds its average and then updates it. The Flight Checks store the time when a certain check has been satisfied (section 3.3.3.8) and the Calibration Readings stores the base altitude, base pressure and IMU offsets and is updated when the 'Calibrate Reading' button is pressed.

3.3.3.7 GUIs

On our ground station, there are two main GUIs that we have made. One is a live updating GUI which we will be using during the mission and the other is a graph maker which makes graphs for us using data stored.

There are three main components to our live GUI: live updating tables, location of the CanSat on a map and a flight check display. The live updating table shows the last five temperatures, altitudes, wind speeds, pressures, humidities, lux values and timestamps. As this is live data, to reduce the impact of anomalies or misread readings, we will be displaying the exponential moving average of each data set. For the map, we are using Tkinter Mapview and it takes in the last latitude and longitude reading and sets a red marker's position to that readings and keeps a set marker on our location. This allows us to easily track the position of the CanSat. The flight checks shows us if the CanSat has landed, deployed and launched, the way these checks are carried out will be detailed later.

The graph maker GUI firstly shows a form to fill. In this form, you first choose whether you want a reading-time graph or a reading-reading graph. Then it shows two (or one if you chose reading-time) selection boxes where you can pick the x-axis data and y-axis data. Afterwards three entry boxes allow you to label the axis and the title. Finally, once you press enter it makes the graph and saves it into a file as a JPEG. To make the reading-time graph, it selects all the values from the table where the name is the data the user wants and then it selects all the timestamps for the same set of data. It then plots them using Matplotlib as a scatter graph. To make the reading-reading graph, it carries out the same process as before for the x-axis data. Then it iterates through every time gotten from selecting the timestamps and then selects the value from the table where the name is y-axis data the user wanted and the timestamp is the current time it is iterating through. Then again it plots this as a scatter graph using Matplotlib. Every graph also shows vertical lines marking the three checks mentioned before, hasLaunched, hasDeployed and hasLanded, allowing us to differentiate data from different stages of the mission.

A picture of our main GUI and graph maker can be seen in Appendix F.

3.3.3.8 Flight Checks

During the mission, three checks will be carried out, a hasLaunched check, a hasDeployed check and a hasLanded check These checks will allow us to keep track of our CanSat and its status and later on differentiate the data from the different stages.

To check if it hasLaunched, it sees if the current altitude is 50 metres or more greater than the base altitude. The buffer is 50m to account for inaccuracies of the GPS data due to noise. The base altitude is manually stored in the calibration Data table by pressing the 'Calibrate Readings'.

To check if it hasDeployed, it takes the last 30 altitude readings and if at least three of those readings are greater than the current reading, it increases a global variable called altitude_decrease_counter by 1 and the process keeps repeating until that variable reaches 3 then hasDeployed is made true. By having the process repeat at least 3 times, it mitigates the effect of bad readings which give us a false positive.

To check if it hasLanded, it takes the last altitude readings and if the difference between atleast 20 of those readings and the current reading is less than or equal to 10, a global variable called altitude_similair_counter is increased by 1. If this variable is 3, then hasLanded is made true.

Sometimes, we won't have a GPS fix so we can't use its altitude therefore if there is no fix, it uses the calculated altitude.

Every time one of these checks are made true, an entry is made into the flightChecks table where it enters the name of the variable into the name field and the timestamp of the variable was made true into the timestamp field.

We removed the hasParachute deployed check and hasReached terminal velocity check due to their being not much use to these checks as the nature of the readings recorded do not change between these checks unlike the previous ones.

A flowchart of this can be found in Appendix C

3.4 Parachute design and recovery system

3.4.1 Material

For our parachute material, we have decided to go with Nylon Ripstop Fabric. Firstly because it is highly durable against tearing and creasing meaning that the chance it rips mid flight is very low. It is also windproof and waterproof which decreases its chance of sustaining damages due to flying conditions and provides the CanSat a partial protection from the elements. Finally, it is lightweight for its strength which is crucial in a mass-limited mission.
For our paracord we are using a 2mm Nylon Rope Paracord. It has been tested (externally) up to 90 kg or around 882N which is more than enough for our purposes. We switched from 3mm paracord because after speaking with the people at Regional Launches, we found out our paracord was too thick for the size of our project and added unnecessary weight for strength which will not be used.

3.4.2 Design

3.4.2.1 Shape

For the shape of our parachute, we have stuck with a flatpack hexagon shape. This is for two main reasons, firstly it is one of the easiest to make and since our mission is not primarily focused on landing, we thought it would be best to divert our resources to more important places. Secondly it has a decently high drag coefficient of around 0.78 which means it slows down the parachute more for less material compared to a hemispherical parachute.

3.4.2.2 Dimensions

For finals, we have cut and sewn 6 parachutes of lengths 15, 20, 25, 30, 35, 40 cm. This is because we went to Regional Launches with a 30cm parachute however they informed us due to the weather and the fact we were in the middle of a military base, the descent had to be around 15 ms^{-1} so our parachute was not allowed as it gave us a descent speed of around 5 ms^{-1} (however in the end we did not get to launch due to adverse weather conditions). The staff working there then informed us that descent speed allowed will change depending on the weather and surroundings as too slow of a speed means that it will drift far away and might go into dangerous areas. By having 6 parachutes, we can adjust to any condition and get the CanSat to fall as slow as possible.

To calculate the terminal velocities of these 6 parachutes, we first need to calculate their surface areas. This can be done using the area of a hexagon formula as seen below:

$$A = \frac{3\sqrt{3}}{2}(a^2)$$

Where:

A: Area of the parachute
a: Side Length

Using this formula we get the areas:
15cm: 584.57 cm^2
20cm: 1039.23 cm^2
25cm: 1623.8 cm^2
30cm: 2338.27 cm^2
35cm: 3182.64 cm^2
40cm: 4156.92 cm^2

Terminal velocity is achieved when the drag force is equal to the gravity force acting on the CanSat so we can set these equations equal to each other and rearrange:

$$mg = \frac{C_d p A v^2}{2} \Rightarrow 2mg = C_d p A v^2 \Rightarrow v^2 = \frac{2mg}{C_d p A} \Rightarrow v = \sqrt{\frac{2mg}{C_d p A}}$$

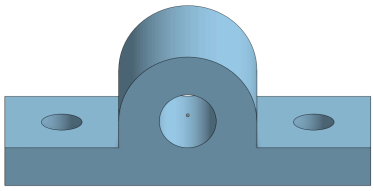
Where:

m: Mass (0.35kg)
g: Gravity (9.81 m^{-2})
C_d: Drag Coefficient (0.75)

p: Local Air Density (1.225 kgm^3)
A = Surface Area(m^2)

By plugging in the surface area values from before we get the velocities below:
15 cm: 11.3 ms^{-1}
20 cm: 8.48 ms^{-1}
25 cm: 6.78 ms^{-1}
30 cm: 5.65 ms^{-1}
35 cm : 4.85 ms^{-1}
40 cm: 4.24ms^{-1}

3.4.3 Parachute Connections



To connect our paracord to the parachute, our model designer created and printed a connector which we have sewn onto the parachute and then put the paracord through.

The two side holes allow us to stitch the connector onto each corner of the parachute and the centre hole is where we slide the paracord into and then tie off. This part was printed with a tighter tolerance so the paracord would have to be forced to get through the centre hole to decrease the chance of it slipping out. To slightly increase the cross sectional surface area, the rope is put in from the outside in, instead of inside out so the parachute is pulled outwards instead of inwards

These paracord attach opposite corners of the hexagon so there are three ropes in total and they all pass through a ring nut which is firmly screwed into the top of the bolt passing through our model.

3.4.4 Spill Hole

As our mission requires the CanSat landing upwards, we added spill holes to our parachutes. From research, the area of the spill hole needed to be around 4% of the parachute area. From this we get the spill-hole areas:

15 cm: 23.38 cm^2
20 cm: 41.57 cm^2
25 cm: 64.95 cm^2
30 cm: 93.53 cm^2
35 cm: 127.3 cm^2
40 cm: 166.3 cm^2

Since our spill hole must a circle, we can use the equation of a circle formula and rearrange for r to get:

$$\pi r^2 = A \Rightarrow r^2 = \frac{A}{\pi} \Rightarrow r = \sqrt{\frac{A}{\pi}}$$

Where:
r: Radius of spill hole
A: Area of spill hole

That gives us the radii below:
15 cm: 2.7 cm
20 cm: 3.6 cm
25 cm: 4.5 cm
30 cm: 5.5 cm
35 cm : 6.4 cm
40 cm : 7.3 cm
So we cut circles in the centre with these radii.

3.4.5 Recovery Methods

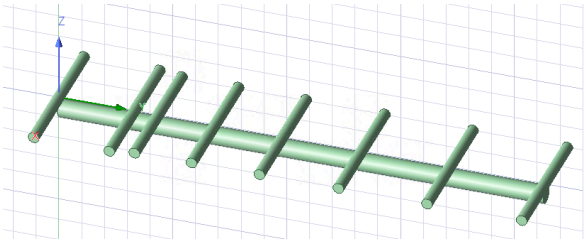
To ensure the highest possibility our CanSat is retrieved, 3 different procedures are in place.

Firstly the parachute fabric is neon orange and the paracord is red. These colours are not typically found in nature and contrast really well with the blue sky and green grass/foliage. This allows us to easily spot our CanSat while it's falling and makes it easier to track when falling. To add to this contrasting effect, our CanSat itself is red.

We realised that the weather/light conditions are not always suitable so there is a possibility that it gets a bit dark during the mission. Hence our paracord is reflective which makes it easier to locate with a torch.

Finally, we have a GPS sensor onboard our CanSat which will tell us the latitude, longitude and altitude and our live GUI will show the position of the CanSat at all times. However the GPS is not always accurate and sometimes we might lose a fix hence we have the two other procedures in place.

3.5 Ground station support and antenna design



A simulation model of the antenna

A real life model of the antenna

One of the main highlights of our team is our custom-built Yagi-Uda antenna, which we built and designed over the course of 4 months. The ground station equipment is one of the most important bits of the project, as it allows communication between the CanSat and the ground station over a considerable distance.

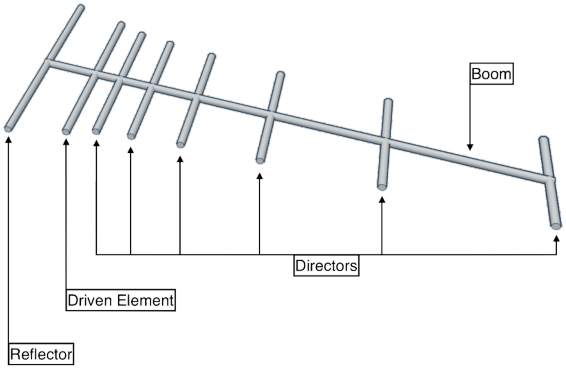
Our Yagi-Uda antenna is engineered around our frequency requirement — 433 MHz, the operational frequency for our CanSat components. From this, we can work out the wavelength as 69.236 cm (to 3dp) from the formula $c = f\lambda$.

3.5.1 Antenna Design and Development

3.5.1.1 Concept

The Yagi-Uda antenna has a lot of functions, including for household Internet or television signals, often seen on top of roofs. Typical designs have operational frequencies ranging from 30 MHz to 3 GHz, within our target frequency. This antenna is classified as a half-length antenna, meaning that the driven element (the part that carries current) length should be half of the wavelength.

As seen right, the Yagi-Uda antenna has multiple components, and has both active (connected to current) and parasitic (not connected to current) elements. From left to right:



Component	Use	Theoretical Length (cm)
Reflector	Reflects any waves going backwards and helps with direction. Parasitic.	0.55λ
Driven Element/Dipole	The main active element, which provides the current required to radiate signals to our CanSat.	0.5λ (34.64)
Directors	These help direct the signal precisely. The more directors are added, the more precise (enhancing the antenna's gain), however if the number of directors exceeds a certain amount	0.45λ, decrease by 0.05λ every director (0.4λ,

	it would have a negative effect on the impedance of the antenna. We will have between 4 and 6 directors (see below & testing plan) Parasitic.	0.35λ…)
Boom	The only non-radiating part of an antenna. Usually made of non-conductive material and is used to support the weight of components.	Mentioned below

3.5.1.1.1 Distances between components (combined will make the length of boom)

Reflector & Dipole: 0.35λ
Dipole and Director 1: 0.125λ
Between Directors: 0.2λ
Therefore, the total length of the boom (4 directors, lower bound):
$0.35\lambda + 0.125\lambda + 0.2\lambda * 3 = 1.075\lambda = 74.476cm$
(6 directors, upper bound):
$0.35\lambda + 0.125\lambda + 0.2\lambda * 5 = 1.475\lambda = 102.188cm$
We decided to use 6 directors.

3.5.1.2 Design process

3.5.1.2.1 Choosing Materials

Radiating Components: As a team we considered a plethora of different possibilities of conductors like copper which was our initial decision as it is a very common material, which we are using for our internal CanSat systems. However, we ultimately decided to use aluminium, due to its cheap price and that it is easy to obtain yet still conductive enough for our antenna design.
We chose a consistent diameter for all radiating components of 12mm, as this would be able to accommodate our chosen frequency while also not being too wide which could interfere with the signal.
Boom: As the boom is a non-radiating component (i.e. no current/conduction of any sort), theoretically we could use any non-metal/insulator for it. There are a couple of benefits to this:

- Using plastic is lightweight and in the case of PVC, more durable.
- Using an insulator would not affect the current as much as using a conductor as the boom. This is important as our calculations might be modified and we would need to also consider the boom's impact on the antenna, which would be further complicated if we had a conductor.

However, PVC plastic is one of the most harmful plastics in terms of environmental impact. To combat this, we will not dispose of our spare PVC plastic, instead we will hand it to our school's DT or physics departments, as it might be useful for product design in their future projects.
We chose PVC tubes with a diameter of 20mm, as this width is optimal to support the antenna's radiating components, while also being durable.

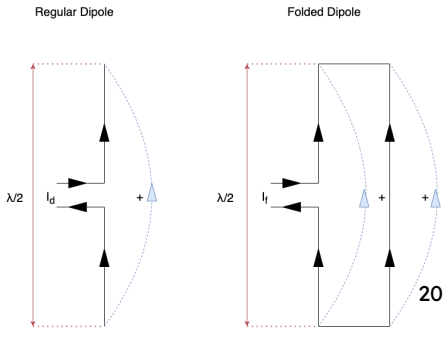
All our components are hollow inside. We decided this because cutting hollow components to length would be easier than cutting non-hollow components, and that it made little to no difference whether we used hollow or non-hollow elements in terms of signal capability.

3.5.1.2.2 Designing the driven element

The driven element is the most important component in the antenna, as it receives the signal from the source and sends it to our CanSat. The team researched multiple ways in which we could design this component, the two main ones being the regular dipole and the folded dipole.

Using the folded dipole means the circuit wires would be able to connect to the antenna more easily, mainly because of its design. It also has a larger gain, meaning it can transmit signals more effectively, and a more directed radiation pattern, allowing it to transmit signals more precisely. However, the impedance is far greater than using a regular dipole, so matching up with the impedance of other components was more of a challenge.

On the other hand, although using a regular dipole is often less convenient than using a folded dipole, we chose this design. This is mainly because of the massive difference in impedance— the team had worked out a way to combat the difference on a regular dipole, not a folded dipole. The



precision of the antenna has been addressed by our decision to include 6 directors, which ensures a successful signal transmission.

3.5.1.3 Prototype & details

The dimensions of our Yagi antenna have been updated due to new testing results.

3.5.1.3.1 Dimensions

Component	Component Length (mm)	Distance from Reflector (mm)	Distance between previous component (mm)
Reflector	339	0	N/A
Driven Element/Dipole	319	138	138
Director 1	292	190	52
Director 2	288	315	125
Director 3	283	464	149
Director 4	279	637	173
Director 5	275	831	194
Director 6	272	1039	208

Boom Diameter: 20mm
Radiating Component Diameter: 12mm

Frequency f: 433 MHz
Wavelength λ: 692 mm
Element diameter d: 12 mm
Boom diameter D: 20 mm
Total number of elements: 8
Boom length: 1063mm
Gain: 12.7 dBi (approx.)

The antenna should be able to send signals for up to half a kilometre, because the theoretical gain is 12.65 dBi, and a stronger gain means the signal can propagate further. The beamwidth (angle in which signal would be received) should be about 20-30 degrees.

Note: dBi is the hypothetical unit for measuring the gain of an isotropic antenna that radiates signal to all of its surroundings, and dBd is the hypothetical unit for dipoles.

3.5.2 Antenna Production and Testing

3.5.2.1 Wiring the antenna

Initially, the team wanted to use a U.FL (IPEX) cable to connect the radio sensor and the antenna. However, the team decided that using a balun did not have any obvious benefits on any aspect of the antenna, least of all the gain, and so we merely wired it to the radio sensor using coaxial wire.

3.5.2.2 Free Space Path Loss (FSPL)

One thing we considered about our CanSat is the fact that radio wave signals weaken over distance. This can be achieved by using the FSPL formula below:

$$\mathrm{FSPL} = 20 * \log_{10} \frac{4\pi df}{c}$$

Where:

- d is distance between transmitter and receiver (km)
- c is the speed of light
- f is the frequency (Hz)
- FSPL is the Free Space Path Loss (measured in dB)

Bounds:
Frequency is 433MHz, or $4.33*10^8$ Hz,
Vertical distance (altitude) is 350m
Horizontal distance is 100m

Firstly, use Pythagoras' Theorem to calculate the maximum distance:
$\sqrt{100^2 + 350^2} \approx 364.0055m$
This can be rounded to 365m.

Next, plug that value and the frequency into the formula in order to get the result:
$$\text{FSPL} = 20 * \log_{10} \frac{365\pi * (17.32 * 10^8)}{3 * 10^8} \approx 76.417dB$$
This data helped us identify how powerful our antennas should be and the dimensions of them. This was just a rough measurement to identify the antenna as there are many variables such as the transmitter and receiver gains, however for a CanSat this would be precise enough.

The gain of the transmitting and receiving antennas are a factor in helping to overcome FSPL, and so we decided to also calculate our antenna gains. Since we purchased a spring antenna for our CanSat, the product specifications listed its gain at 2.5 dBi.

3.5.2.3 Calculating Theoretical Antenna Gain

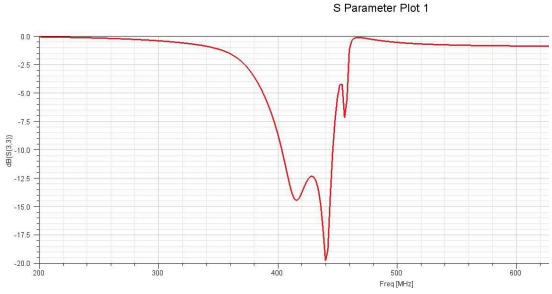
From an antenna builder, we calculated that the ground antenna gain is 10.5 dB, but we can calculate maximum antenna gain by using this formula:
$$G_{dB} = 10log_{10}\left(\frac{4\pi\eta A}{\lambda^2}\right)$$
Where:
- GdB is the antenna gain (dB)
- η is the efficiency (multiply by 100)
- A is the physical aperture area (m^2)
- λ is the wavelength of the signal (m)

Ground Antenna Gain (Note that this is a hypothetical calculation but most antennas have an efficiency of near 100%):
$$10\log\left(4\pi \cdot 100 \cdot \frac{0.702192}{0.69236}\right) = 31.05333772 \approx 31.053dB$$
CanSat Antenna Gain:
$$10\log\left(4\pi \cdot 100 \cdot \frac{0.067835}{0.69236}\right) = 20.90331724 \approx 20.903dB$$
This gain makes it easier to transmit signals with success to the antenna, as this value would send strong signals for a considerable distance.

Our radio modules' (on the ground and CanSat) gain is +20 dBm.

3.5.2.4 Antenna Testing Graphs

As not all testing for the antenna is executable in real life, we used HFSS software on Ansys to build the antenna and test it. The charts and graphs below show the relationship between frequency and gain, the 2D radiation pattern and the 3D radiation pattern.

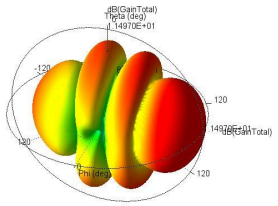


3.5.2.4.1 Antenna Frequency Gain Graph

This graph shows the frequency (X axis) in relation to the gain (Y axis) of the antenna. The antenna's projected maximum gain is 15.6 dB at about 440 MHz, but at 433 MHz the antenna still has a gain of about 12 dB.

3.5.2.4.2 3D Radiation Pattern Graph

This graph shows the 3D radiation pattern of the antenna and which parts around the antenna have the largest gain. Blue means it has the most negative gain, followed by cyan, green, and yellow, with red representing a high positive gain. The antenna's reflector is on the left and the last director(director 6) on the right. The gain, as seen above, has a negative range between -12.5 and -25 dB close to the dipole/driven element (green), and has the largest positive gain in the red areas, the most predominant one located at the front of the antenna (director 6), with a gain of up to 12.5 dB (red). These results were possibly different from the calculations in section 3.5.2.3 for a couple of reasons, for example how we wired the antenna on the simulation.



3.5.2.5 Challenges, Resolutions and Reflections

We faced three main problems regarding the entire process of making the antenna:

1. **Only one person could work on our simulation software, Ansys HFSS, at a time** as we had a student(free) version. We solved this problem by utilising applications such as Microsoft Remote Desktop and Teamviewer to allow remote desktop control.
2. **Antenna Frequency was not in the right range, and the spreading factor(frequency range) was too narrow** according to our simulations. We solved this problem by modifying the distance between directors, dipole and reflector, as the components had to go from long(reflector) to short(director 6)
3. **Insufficient gain/range for the antenna.** We solved this by a couple of methods, including decreasing signal bandwidth for more range(see section 3.6.5.2 for full details)

3.6 Testing

3.6.1 Mechanical testing

Thing to Test	How we tested	Results	Improvements
Model Durability	By using a SUVAT equation, we calculated how far up to drop the model so it hits the ground at 8m/s (our terminal velocity) and calculated it to be 3.3m (ignoring air resistance). We then dropped our model at 0.5m and then repeated while increasing the height by 0.5m all the way to 3.5m. After each drop we took a picture and noted down any cracks or damage. The first time we did this, our surface was grass so we repeated with a new model on hard cement.	For the grass test, no cracks or damages showed up all the way until the 3.5m where some of the layers near the top were being pulled apart. For the cement test, a crack appeared at 2.5m along the base of the model	To combat the layers being pulled apart, our model designer decided to print the layers vertically so the stress is perpendicular to the printing direction. To increase the general strength of the model, we decided to commit to using polycarbonate instead of PLA and print with a high infill. After carrying out these modifications, we repeated the test and on both concrete and at grass, there was no damage until 5.5m.
Anemometer Section Columns	We printed only the top part of the model and repeated the model strength test mentioned before	At only 2m, on both grass and concrete, the connector snapped at the base	We widened the columns and combined with the improvements detailed in the whole model test, the columns only started to break at 5m.
Bottom to Top Connection	We attached a rope to the top of our model and started swinging it around as hard as we could, trying to simulate the G force the CanSat will face during launch/descent.	Once the CanSat was swinging as hard as the person could swing it, the top section of the CanSat disconnected from the bottom due to layer separation in the 3D prints.	To combat this weakness, we have a threaded rod running through the centre of the model, holding both sections together.
Anemometer Optimisation	Using a home-made mini wind tunnel, we tested different anemometers with differing cup widths and cup depths and recorded it in slow-motion. We then counted the rotations per minute.	We found the optimal cup width and cup depth to be 17.5mm and 5.8mm respectively. These gave us the highest RPM which should help us measure the speed accurately at lower wind speeds.	Removed bearings as they were increasing the weight and slowed the anemometer down slightly.
Anemometer Durability	Placed an anemometer in its place in the top section of the CanSat. Started dropping it onto grass and concrete from 0.5m height and then increased the height by 0.25m every time	The anemometer and the top part of the CanSat showed no cracks on grass until we threw it up as high as we could, which is around 8 - 9m where it cracked slightly. On	As the cracks appeared on the connection, our model designer increased the width of these connections

		concrete a small crack appeared at 5.25m.	
Internals Durability	Fitted all the internals into the CanSat along with the anemometer. Threw it off a balcony 8 metres high with a parachute and checked the state of the internals afterwards	No damage was found even after multiple throws	N/A

3.6.2 Electrical testing

Thing Being Tested	How we tested	Result	Success?
GPS Accuracy	At a team members house, we took GPS readings and stored them. We then took the exact latitude and longitude from Google Maps of the house and calculated the offset.	The offset was relatively small and the graph of our data can be found here: https://cansatathena.com/GPS-3d-graph Where the red line is the origin line.	Yes
LDR	Using a lux meter, we exposed the LDR and lux meter to the same light level.	The lux calculated by the LDR and the lux on the lux meter were within 5% of each other.	Yes
Battery life	Left the CanSat running with all sensors on.	Lasted for 15 hours	Yes

3.6.3 Software testing

To test the electronics code, we printed all the data onto a text file and checked to make sure it was reasonable looking data and printed in the correct order. We also then made sure the code could handle things like crashes and failed initialisations by turning off the Pico mid data transmission and disconnecting certain sensors to see how it handled that.

To test the ground station data, first we fed made up data into our parser and made sure it was able to firstly, correctly enter data into the database and then handle things such as crashes, misprinted data and misread data. Then we made sure the graph maker was working with the database and could plot the data. Finally, we used the actual data from the sensors so we had live data to test on and made sure the main GUI updated correctly.

As an overall check, the ground station engineer looked over the electronics code and the electronic engineer looked over ground station code to try to find any logic errors in the code.

3.6.4 Landing & recovery testing

Thing to Test	How we tested	Results	Improvements
Parachute cord strength	We cut a 30 cm section of the rope and tied one end to a high up bar and the other end attached to weights.	The cord showed no signs of tear even after team members hung on it.	N/A
Parachute connection	Attached the parachute cord to the parachute and the other end to the top part of the model (not printing the rest) and then tied two 20 kg weights and a 10 kg weight to the bottom and help up the parachute	Parachute stayed attached to the CanSat and both the ring bolt and printed parachute connection held strong. The top part of the CanSat snapped after 7 seconds however in the actual mission that part will not face anywhere near that amount of force	N/A
Parachute deployment	Dropped an empty model with the parachute folded up from the top of our school building.	Parachute deployed quickly and caught wind	N/A
Parachute (35cm)	We placed nuts and bolts into the CanSat until it weighed 300g. Afterwards, we dropped it off the top of a balcony 8 stories up and measured the descent rate.	CanSat fell at a speed of 4m/s.	Decreased parachute size but later made 6 parachutes of differing sizes

3.6.5 Ground station & antenna testing

3.6.5.1 Simulations ran on HFSS:

Thing Being Tested	Success Criteria	Success?	Improvements
Gain Frequency relationship	Gain is about 12 dB at 433 MHz	Yes	Moved boom distance from the antenna
Radiation Pattern - 2D	N/A: just a simulation.	Yes	N/A
Radiation Pattern - 3D	N/A: just a simulation, but the shape of the radiation pattern should look like a ghost/octopus.	Yes	Changed the elements' placings, especially the driven element/the port.

3.6.5.2 Testing conducted in real life

Thing Being Tested	How we tested	Success Criteria	Success?	Improvements
Signal Travel Capability (Distance)	We placed the antenna at a certain distance to the CanSat. We then ran the code to receive signals from the CanSat for 5 seconds, then moved away slowly until the signal could not be transmitted anymore.	The antenna should be able to receive signals from further than 500m, which is the theoretical maximum distance our CanSat would be from the ground station, if not further (preferably up to 1km) Estimates put the gain at 10.5 dBi (or 12.65 dBd).	Yes(after some software, but not design improvements) The antenna has a range of beyond 525m.	Decreased signal bandwidth for more range. Increased spreading factor(chirp spread or frequency range) Set coding rate to 4/8 Increased transmission power to +20dBm We would have increased the number of directors for a larger gain/ introduction of a mesh to prevent signal backflow.

3.6.6 Requirements Check

1. **All components of the CanSat must fit within a cylinder with height 115mm and 66mm diameter:**
   The model has been printed to a diameter of 65mm and height of 114mm
2. **No parts of the CanSat can extend beyond the diameter until it has left rocket:**
   Designed leg system so it can only extend when it has space to extend
3. **Mass of the CanSat must be between 300 and 350g:**
   Our CanSat with a parachute weighs 290g so two 10 g weights were added to the bottom
4. **Materials must be safe to the personnel, equipment and environment:**
   PLA, Polycarbonate and the metal we use are not actively dangerous for the environment
5. **Must be powered by a battery/solar panels and run continuously for at least 3 hours:**
   CanSat is powered by a lithium battery and can run for 15 hours.
6. **Battery must be easily accessible**
   Battery location is accessible quickly after unscrewing the top lid
7. **CanSat must have an easily accessible power switch**
   Placed the Pico in such a way the power button is easily accessible behind the leg
8. **CanSat must have a recovery system**
   Have a neon orange coloured parachute
9. **Parachute connection must withstand 500N of force**
   Connection can withstand 500N of force as tested in 3.6.4
10. **Descent Speed is recommended to be 8ms^{-1} and 11ms^{-1}**
    Have made 6 parachutes, two of them giving a descent speed between 8 and 11ms^{-1}
11. **The recovery of the CanSat is not guaranteed after launch**
    The data will be transmitted to ground via antenna and internals are financially disposable
12. **Budget must not exceed £400**
    As seen in Appendix B we still have around £120 remaining
13. **CanSat must have ability to alter frequency for European Finals**
    Not applicable this year as there are no European Finals

3.7 Overall testing for launch

3.7.1 Launch simulation

To simulate launch conditions, we did our overall tests in parks with open fields. This helped us get a more accurate picture of the situation we will be faced with in the finals, which is especially crucial for antenna testing.

3.7.2 Cross-Departmental Testing

In addition to the checks mentioned above, we also need to conduct a few cross-departmental tests before our CanSat would be ready for launch.

Thing being tested	How we tested	Success criteria	Success?	Improvements
Wind effect on CanSat	On a windy day, we did the drop test—we deployed the parachute mid-air, and investigated how far the parachute deviated from the initial predicted landing position. We then calculated the predicted deviating distance for the altitude of launch day.	N/A, just a measure. Ideal scenario would be minimal wind effect on CanSat (i.e. The CanSat would not be too off-course due to wind)	Yes(after adding a spill hole)	We added a spill hole to our CanSat. If the deviation distance was too significant, we would have possibly considered changing the design of the parachute.
Leg system & parachute opening manoeuvre	We dropped the CanSat from a height, automatically deploying the parachute, extending the legs and landing.	Legs and parachute successfully deployed and CanSat lands upright.	Yes(after improvements)	Added a spill hole for the CanSat Added one more leg(from 3 to 4)
Sending and receiving real data from the CanSat(done in conjunction with antenna range test)	We re-enacted part of our secondary mission (after the CanSat landing), transmitting data to the ground station every second. We used verifiable data from other external sensors next to the CanSat to ensure correct data(see section 5.4.3)	The data of the external sensors matches up to the CanSat reading; the CanSat successfully transmits signals every second/otherwise instructed.	Yes(after improvements)	Decreased signal bandwidth for more range. Increased spreading factor(chirp spread or frequency range) Set coding rate to 4/8 Increased transmission power to +20dBm

Part 4: Outreach

We have also got a new article on thisLocalLondon written about us:
https://www.thisislocallondon.co.uk/news/24095861.young-reporter-tiny-team-cosmic-triumphs-luca-lo-presti-sgs/

	Reddit	Website	Instagram	Youtube	School
Links	https://www.reddit.com/user/Project_Athena/	https://cansatathena.com	https://www.instagram.com/athena_cansat	https://www.youtube.com/@CanSatAthena	
Actions	Have engaged in discussions with engaged followers, building up our follower base with conversations about the academics behind the Cansat, as well as memes to engage different demographics.	Have released 13 articles since the CDR, have created a new blog – "State of Cansat". Have updated the website format to look more modernised.	Have released 3 shorts and 2 posts about the CanSat	Have released 3 videos with Youtube Shorts about our Cansat's development.	We gave a formal assembly to the Year 7 pupils about CanSar, we also put up posters to spread publicity throughout the school, in every school building.
Engagement	Hundreds of thousands of views and hundreds of replies and comments from users.	Thousands of views from over 20 different countries and 5 different continents.	Over a thousand views from multiple countries.	Over a thousand views, with interactions in the comments sections.	Gave the pupils a survey, and over 90% of students would want to engage in Cansat competitions of the future, and nearly all of the students agreed that they had gained a strong understanding about what

					Cansat was after the presentation.
Summary	We began to engage users by posting on GCSE subreddits, and replied to all comments, as this gave us a much stronger following, with the Cansat memes posted to give off the fun vibe about Cansat. We also posted on programming and 3D printing subreddits as that is part of our target demographic. We have now developed a focus on posting updates and also memes to keep followers engaged.	In the last couple of months, there have been articles about the progress we have made since the CDR, like the regional launch and our parachute progression, we have also posted about our code and our choices we made for the code. This wide variety of topics gave us a broader range of viewer demographic.	In the last couple of months, we have been producing shorts on Instagram promoting different aspects of the launch, including parachute testing and data transmission. We have also posted some interior designs of our CanSat and antenna.	We posted three shorts, combined getting a massive reach of over 1000 views. These shorts showed the evolution of our model which catered to people interested in our story, we then posted a 3D-printing timelapse and a unboxing video which are trendy videos and engaged people who did not previously know about our project	We also had many interactions with the school's DT and Physics departments, who helped us with both the construction of the CanSat, and also the equations and theory behind our concepts throughout the project. Giving the assembly allowed us to distribute our knowledge with the lower years, who enjoyed the presentation we had prepared and were engaged with the questions we asked.
Stats	95.25% average upvote rate 250+ share 175k+ views 150+ comments	Over 1500 views 5 different continents From over 25 different countries 50+ views generated from social media	Over 1700 views across multiple videos and shorts Global reach including in Spain, the UK, Bulgaria and more.	1000+ views over 3 shorts Global Reach	N/A
Changes	We began by making content specifically targeted to teenagers, pre-teens. We have changed our strategy to target students in general and younger adults who have an interest in STEM, this change has seen a significant increase in viewer engagement.	We created the blog to make it seem like the target audience are on the journey with us, and this has been successful in gaining more attraction to the website, with all comments being positive about viewers' own experiences. with STEM projects.	Similar to Reddit, we began making content for teens, but our focus was for teens who did not have much interest in STEM. Although this meant we had to choose cool bits and simplify a lot of technical things, this strategy has led to significant increase in viewership.	Our YouTube strategy is centred around making shorts for a general audience. This included a diverse variety of people. This general target meant we needed to make generalised and simple content, not too dissimilar to Instagram, and this strategy has led to quite some success.	We have undertaken a much more active role in publicising our project to the rest of the school in the last couple of months. This has created a school community where many students have expressed interests in helping the project.

Part 5: Launch Day Preparation

5.1 Launch Checklist

Task	Time	Duration	Members Involved
Safety Briefing	9:00 begin	15-30 minutes	Whole team
Ground Laptop Configuration	9:30	15-30 minutes	Software Programmer
Antenna Wiring & Positioning	9:30	15-30 minutes	Antenna Team
Turn on CanSat and attach parachute	9:30	10 minutes (+5 more for error interval)	Model Designer & Electronics Engineer
Run final diagnostics & tests	10:00	30 minutes	Software Programmer

5.2 Post-Mission Checklist

Task	Time	Duration	Members Involved
Data Saving	N/A	10 minutes	Automatic
CanSat Retrieval	10 minutes after mission complete	30 minutes	All
Results Compilation: Primary Mission	30 minutes after mission completed	1 hour	Software Programmer
Results Compilation: Secondary Mission	30 minutes after mission completed	1 hour	Software Programmer

5.3 Risk Assessment

5.3.1 Safety Hazards

Hazard	Risk	Risk Grading	Solution	Risk Grading after
High velocity projectiles in powered flight	Serious injury or death from impact	High	Safety distance precautions should be taken. Warning when launch happens.	Low
Falling rocket or components after launch	Serious injury or death from impact	Medium	Safety distance precautions should be taken. Warning when launch happens. Flight paths should aim away from personnel	Low
Burns from rocket motors during or after firing	Physical harm, burns	Medium	Safety briefing	Low
Noise from launch	Damage to hearing	Medium	Ear defenders, safety briefing	Medium
Fire Risk	Property and personal risk from fire started by the activity	Medium	Checking for dry grass and other flammable objects. Safety Briefing	Low

5.3.2 Project Launch Risks

Risk	Risk Grading	Solution	Risk grading after	Responsible member
Parachute not opening	Low	Make sure the parachute has been tested prior to the launch. Parachute must be fitted correctly.	Low	Parachute Engineer
Sensors not working from the CanSat	Medium	Make sure everything is soldered and calibrated properly before launch, do a final test.	Low	Electronic Engineer
Loss of signal transmission	High	Store data on CanSat Keep Antenna pointed at CanSat at all times	Medium - High	Antenna Engineer & Team
Leg system not operating	Medium	Ensure elastic bands are in place and test prior to launch	Low	Model Designer
Anemometer breaking	Medium	Inspect the anemometer before launch for any	Low	Model Designer

		damage.		

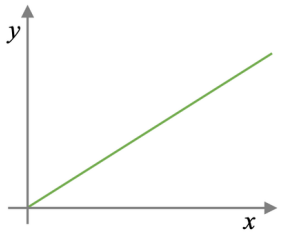
5.4 Results Analysis Procedure

5.4.1 Primary Mission

For the primary mission, we are analysing in the the following way:

1. **Correlation between temperature and altitude:** We will convert the pressure readings into altitude readings and then plot the graph between the temperature and altitude readings. We expect to see as altitude increases the temperature decreases. If this happens, it confirms the fact that the air expands higher up as it's under less pressure and therefore the temperature decreases. This is called adiabatic cooling.
2. **Barometric Pressure Formula Accuracy:** We will convert the pressure readings into altitude readings as seen before and then plot the graph of these calculated readings compared to the GPS altitude readings. We hope to see a linear correlation of the form y = x as this shows our barometric pressure formula is accurate but if we find there is a significant difference between the two readings, we will try to change the constants in our formula to try to get a better correlation.

Our second procedure should show the linear correlation on the right:



5.4.2 Secondary Mission

The purpose of secondary mission analysis is to see whether wind, solar, geothermal and hydroelectric energy are viable at that launch field and if they are efficient.

To see whether wind energy is viable, we will use the Hall Effect sensor triggers. These triggers will be converted into distance and then divided by the time between readings to measure speed. This will then be multiplied by 1.27 to get the wind speed. However since this is only the wind speed at the height of the anemometer, we need to use the logarithmic sheer formula to get the wind speed higher up. A wind turbine is around 90 m high normally so we will use 90 as the height in said formula. From research we have found that a wind turbine needs a speed of at least 5 m/s so we will check to see if the wind speed is above that. A turbine starts to break at around 25 m/s so we will also check the wind speed is below that. The peak efficiency of a turbine is around 15 m/s so if the place is suitable for wind energy, the speed will be close to that.

To see whether solar energy is viable, we will be using the LDR. From the LDR value we can estimate the lux value the CanSat is exposed to. This value can be converted into Solar Radiation by multiplying by 0.0079 and that gives us a value in Wm^{-2}. This is useful as we know how much energy that area will be getting. Averaged over a year, the best solar farms receive 250 - 300 Wm^{-2} so we will be looking out for that. The optimum temperature for a solar panel is around 25 degrees celsius so that's another thing we will be checking for.

To see whether hydroelectric energy is viable, we need to check if there is rain. This can be achieved by using the humidity sensor and if the humidity is higher than 90% for a significant time, there is a significant chance there will be rain and therefore rivers/lakes which can be used for the hydroelectric energy.

To see whether geothermal energy is viable, two things must be there, water and magma. We can see if there will be water using the same method mentioned before for hydroelectric energy, to see if there's magma, we can instead measure for continental drift (as that implies the existence of magma) and that can be done using the IMU data. We will look for any significant changes in the IMU data as that shows it has faced an earthquake and therefore there is continental drift.

One problem with our secondary mission analysis is that it requires data over a long time as renewable energy sources need to be running almost continuously and we can't guarantee that with data taken over a 10 minute time-span.

All the conversions mentioned above can be found in detail in 3.3.3 Conversions and source for nearly all data can be found in Appendix E

5.4.3 Data Verification

To ensure our data is accurately recorded and transmitted, we are using sensors from the school to record ground level data (a lux meter, an anemometer and a temperature sensor), and compare it with data recorded with our CanSat sensors to verify our results as correct.

Part 6: Lessons Learned

Member 1:
For me, CanSat was a huge step out of my normal comfort zone. There are a lot of things that I would not have done, including looking into antenna production and learning all about communications engineering. CanSat also allowed me to spend quality time with my friends and get to know them much better. Ultimately it has been a really fun process and I particularly enjoyed all our testing sessions. Although we were one of many teams that could not launch their CanSat due to terrible weather, I still learned a lot from the launch event, including the rockets we were going to use for launch, which will be useful for the National Final. My only regret is not being able to expand on primary mission data analysis like I wanted — perhaps my initial expectations were unrealistic. All in all, CanSat has brought a massive challenge to me — in a good way — and given the choice I would definitely recommend it or go for it again in future.
Member 2:
Personally, I have enjoyed the team camaraderie that has kept us going, especially when we had long online meetings to meet our team deadlines. CanSat has helped me gain communication skills that I would have otherwise not gained. For example, as Outreach Coordinator, I had a firm role in leading the team's social media macro/micro approach and taking the lead on presentations that we have given as part of our Outreach programme. These presentations have advanced my presentation and communication skills and allowed me to grow in confidence when speaking to an unfamiliar audience. Looking back on the project, I feel like I could have focused more on the outreach section which was dedicated to the school community, as we could have spread even more awareness in the forms of more assemblies and more posters.
Member 3:
Due to CanSat I have learned and picked up many new skills. Firstly, I have added a new language to my repertoire,Python, as I now feel comfortable with its syntax and using external libraries. Being a parachute designer as well, I have learnt a vast amount of parachute physics and the different steps in making one. My biggest challenge was researching data analysis as I had not realised the amount of effort required for accurate research and making sure the sources are reputable. For me I would say the best part of CanSat was when we conducted a parachute test and threw our CanSat from around 30m high. I think the nerves of the moment combined with finally seeing our product work made an unforgettable moment. Though I put in nearly all the effort I could, I would say one area I could improve on was setting realistic expectations as I had visions for the ground station which I tried to recreate but I realised later we were not feasible and ended up with me wasting a decent amount of time.
Member 4:
CanSat was an extremely rewarding experience for me, and I really gained a lot of insight just by participating in this project. For me, I learned a lot from my friends on the team, including a lot of communications engineering, model design and even how to sow a parachute. I particularly enjoyed our meetings in which we would joke about and also discuss and finish designs, reports and plans. This team has an excellent dynamic and atmosphere and I am delighted to be a CanSat national finalist in Project Athena. Given the choice I would definitely take part in this competition, or similar, again.
Member 5:
One of the major things I learned from CanSat was FreeRTOS and how to implement it as before the competition. I was slightly learning it but after writing the code for the electronics I can confidently say that I am very knowledgeable about FreeRTOS and can use it frequently in my future project. That being said, I would still say learning how to implement FreeRTOS was my biggest challenge as it is quite an advanced undertaking but with the help of my teammates I achieved it. I would say that the best part of our project was Regional Launches as even though we did not get to launch, it was a great time going out with my friends and making new ones at the launch. One thing I think I would improve on is adding a sleep mode to extend our battery life however as it wasn't required I do not feel too bad about skipping it.
Member 6:
As testing coordinator, I had a unique job in the team dealing with all the departments of the project. This put me in an excellent position to learn a bit of everything. Such a wide range of different tests were also very unique, and I am honoured to have been in this role. One of the best moments was when I coordinated the parachute drop test, where we could drop our CanSat down from a height — that was funny to watch and I remember it almost getting caught in a tree. One thing I could have improved on, however, would be coordinating cross-departmental testing more efficiently.

Appendix A

Component	Type	Use	Dimensions (mm)	Included?	Price
Pico LiPo (16MB)	Microcontroller	Controls everything, variant of Raspberry Pi Pico with more flash + built-in LiPo controller	51 x 21 x 8	No	£13.50
Adafruit BME680	Sensor	Will be used for sensing pressure and VOCs	17.8 x 20.3 x 3.0	No	£18.90
1200mAh 3.7V LiPo Battery	Battery	Powers everything	62 x 35 x 5	No	£8.00
Adafruit PA1616D	Sensor	GPS sensor to allow us to locate the CanSat and track its trajectory	25 x 35 x 6.5	No	£29.80
RFM96W LoRa 433MHz (Transceiver)	Radio	Transmits signals back	29 x 25 x 4	Yes	£19.50
RFM96W LoRa 433MHz (Transceiver)	Radio	Same as previous, but this time will be used to receive signals	29 x 25 x 4	No	£19.50
Potentiometer	Resistor	To create a voltage divider in series with the Light Dependent Resistor	-	No	£0.80
JST PH Connector Kit	Other	Better than soldering/using header pins because less space, more secure + easier to swap out	-	No	£4.30
DHT-20	Sensor	Measures the temperature and humidity (higher accuracy than BME680)	5.8 x 12.5 x 16	No	£4.50
ICM-20948	Sensor	Accelerometer/Gyroscope/Magnetometer	25 x 17.8 x 2	No	£14.70
Light dependent resistor	Sensor	Used for measuring light intensity	-	No	£2.00
Hall Effect Sensor	Sensor	Used for checking if a magnet has passed it	-	No	£5.65

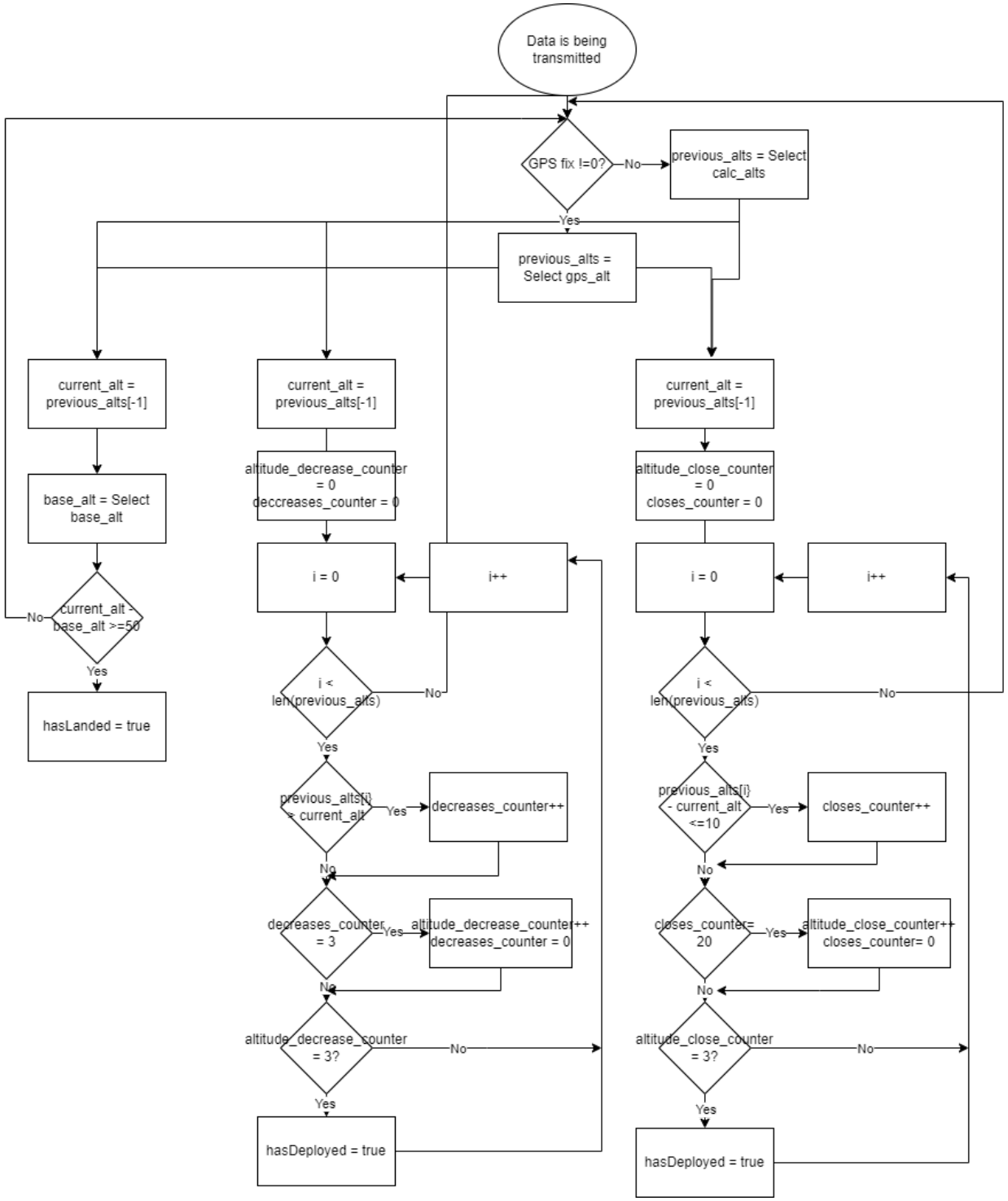
TOTAL	£121.65	**Remaining Budget**	£228.35

Appendix B

Financial Calculations

Description	Price
Pico LiPo (16MB)	£13.50
Adafruit BME680	£18.90
1200mAh 3.7V LiPo Battery	£8.00
Adafruit PA1616D	£29.80
Potentiometer	£0.80
JST PH Connector Kit	£4.30
DHT-20	£4.50
ICM-20948	£14.70
Light dependent resistor	£2.00
Hall Effect Sensor	£5.65
Female headers	£1.08
Pico LiPo (spare)	£13.50
1200mAh LiPo (spare)	£6.96
PA1616D GPS (spare)	£29.80
BME680 (spare)	£18.90
ICM-20948 (spare)	£14.70
DHT-20 (spare)	£4.80
Polycarbonate filament	£22.00
Parachute rope (30m)	£11.00
Eye bolts	£8.00
Total:	**£232.89**
Budget remaining:	£117.11

Appendix C


Data is being transmitted
GPS fix !=0?
No
previous_alts = Select calc_alts
Yes
previous_alts = Select gps_alt
current_alt = previous_alts[-1]
base_alt = Select base_alt
current_alt - base_alt >=50
No
Yes
hasLanded = true
current_alt = previous_alts[-1]
altitude_decrease_counter = 0
deccreases_counter = 0
i = 0
i++
i < len(previous_alts)
No
Yes
previous_alts[i] = current_alt
Yes
decreases_counter++
No
decreases_counter = 3
Yes
altitude_decrease_counter++
decreases_counter = 0
No
altitude_decrease_counter = 3?
No
Yes
hasDeployed = true
current_alt = previous_alts[-1]
altitude_close_counter = 0
closes_counter = 0
i = 0
i++
i < len(previous_alts)
No
Yes
previous_alts[i] - current_alt <=10
Yes
closes_counter++
No
closes_counter= 20
Yes
altitude_close_counter++
closes_counter= 0
No
altitude_close_counter = 3?
No
Yes
hasDeployed = true

Appendix D

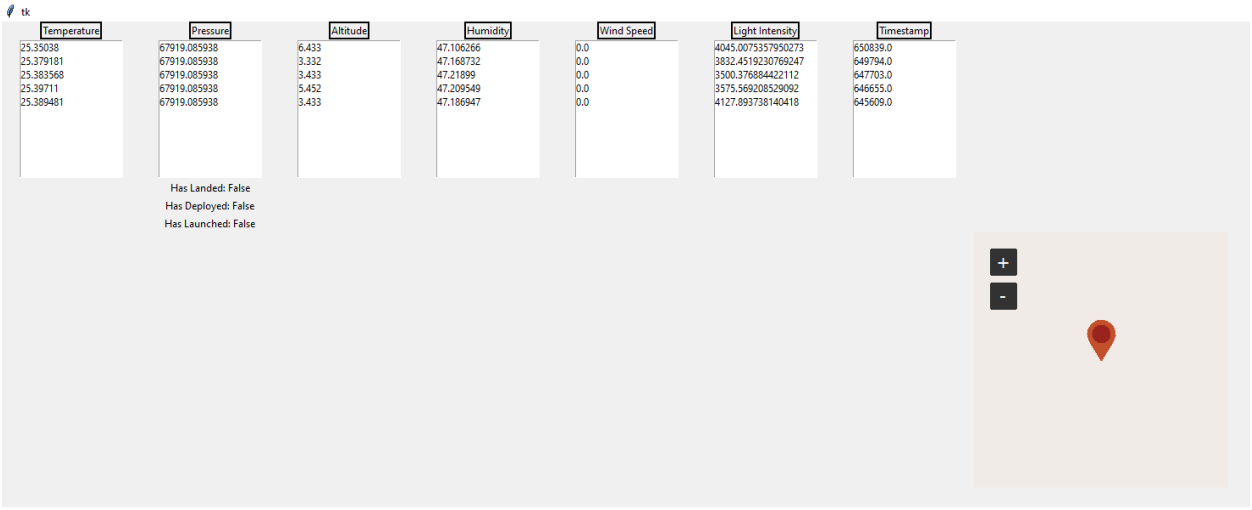
Roughness class	Roughness length z_0	Types of terrain surfaces
0	0.0002m	Water areas: sea and lakes
0.5	0.0024 m	Open terrain with a smooth surface, e.g. concrete, airport runways, mown grass, etc.
1	0.03m	Open agricultural land without fences or hedges, possibly with widely scattered buildings and very rolling hills
1.5	0.055m	Agricultural area with a few houses and 8 m high hedges more than 1 km apart
2	0.1m	Agricultural area with a few houses and 8 meter high hedges approximately 500 m apart
2.5	0.2m	Agricultural area with many houses, bushes and plants, or 8 m high hedges at a distance of approx. 250 m
3	0.4m	Villages, small towns, agricultural areas with many or high hedges, forests and very rough and uneven terrain
3.5	0.6m	Larger cities with tall buildings
4	1.6m	Big cities with tall buildings and skyscrapers

Appendix E

Source for Sheer Formula and Roughness Table:
https://wind-data.ch/tools/profile.php?h=0.1&v=5&z0=0.1&abfrage=Refresh
Source for turbine speed range and optimum speed:
https://www.ewea.org/wind-energy-basics/faq/#:~:text=Wind%20turbines%20start%20operating%20at,second%2C%20wind%20turbines%20shut%20down
Source for average turbine height:
https://css.umich.edu/publications/factsheets/energy/wind-energy-factsheet#:~:text=The%20average%20hub%20height%20of%20modern%20wind%20turbines%20is%2094%20meters.&text=Global%20onshore%20and%20offshore%20wind,872%2C000%20TWh%20of%20electricity%20annually
Source for solar radiation in the top solar farms:
https://www.alternative-energy-tutorials.com/solar-power/solar-irradiance.html#:~:text=Irradiation%20through%20the%20Atmosphere&text=When%20dealing%20with%20photovoltaic%20solar,or%20commonly%20%E2%80%9CPeak%20Sun%E2%80%9D
Source for optimum solar panel temperature:
https://blog.ecoflow.com/us/effects-of-temperature-on-solar-panel-efficiency/#:~:text=The%20optimal%20temperature%20for%20solar%20panels%20is%20around%2025%C2%B0,%25%2C%20affecting%20overall%20energy%20production

Appendix F

Main Gui



Graph Maker