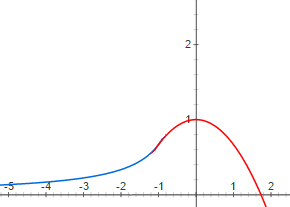
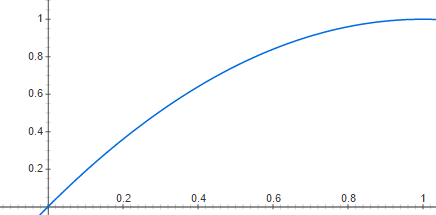
Software

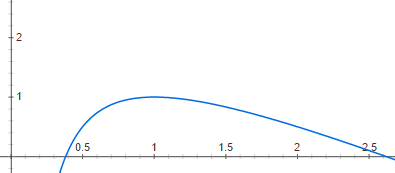
From the fore, the team knew that there was a number of different strands of software that were needed in order to control the can well, allowing it to become an autonomous drone. We decided, alongside the Electronics team, to keep the flying systems (through the main control board), the sensor systems and camera system separate, thus allowing us to ensure that in case of failure of one system, the others would continue to work. We also divided the team into a number of different strands, with some working on the website, alongside the outreach team and others with the electronics team. We also followed a large part of outreach, putting all of our code on GitHub (<http://github.com/cyclonecansat>) under the MIT license, thus allowing anyone else to use our code if they wanted to. We are also attempting to produce documentation to better explain how everything worked, for this to be even easier. In many ways, this process also works the other way, as we rely off some work others have completed, especially in the sensor system, learning from their examples. The flying system of the Can required little to no software, given that we were using an advanced control board which include auto stabilisation. Additionally, we decided that the autonomous aspect of movement would be achieved by altering the signals sent to the flying system, using a computer to calculate the signals to send to the RX of the control board. However, we were still required to write a very complex base station software to allow us to send the correct commands to the Can, so it could monitor the place of the Can at all times and issue commands to move it to a user-defined location.

# Algorithms

To design an equation for estimating agricultural viability of a particular site, we thought about how the different inputs (temperature, humidity and pressure) would affect the growth of crops. We came up with several mathematical equations to illustrate these relationships, plotting several graphs where the y-axis is a measure of agricultural viability with a maximum value of 1 and the x-axis is the input.

Temperature would control a plant’s enzymatic activity, meaning that temperatures near the optimal temperature for a given enzyme would favour plant growth. To show this representation graphically, the graphs of and were combined to produce the graph shown here. The lines cross at x=-1 where the two gradients are the same, ensuring the curve is continuous. When x is less than -1, is used, showing the increasing enzymatic activity as temperature increases and hence the increasing agricultural viability. For values of x greater than -1, is used, with the steep drop representing the effect of enzyme denaturing at high temperatures. For appropriate use by the can, the measured value for the temperature would be altered such that the optimal temperature (e.g. 27°C for rice) returns a value of 1 for agricultural viability and normalised such that a range of suitable temperatures still return relatively high values for agricultural viability. The graphs could also then be stretched to allow a sensible range of temperatures to yield high viability. For rice we will replace x with which creates a maximum temperature of T=42.5.

We decided that humidity could be a proxy for soil water content, with higher humidity resulting in more water available for crops and so a higher agricultural viability. Therefore, we decided that could be used. The humidity measured by the can would be divided by 100 so that 100% humidity returns a value of 1 for agricultural viability.

We assumed that for Earth-originating crops, a pressure of 1atm would be ideal for a plant, with a vacuum being more detrimental than higher pressures. Taking this into account, the graph of was chosen as it displayed the features we wanted. As it peaks when x=1, this graph would not require any further calibrating provided the pressure is in atmospheres. The three values for agricultural viability multiply together to give an overall value between 0 and 1.

Summary:

where;

|  |  |  |  |
| --- | --- | --- | --- |
|  | Calculation | Condition | Notes |
| = Viability of Temperature |  | For | Where *T* = temperature in degrees Celsius. |
|  | For |
| = Viability of Humidity |  | For 00 | Where *H* is relative humidity in % |
| = Viability of Pressure |  | For | Where *P* = pressure in atmospheres |

# Website

Over the summer, the Software Team has worked hard to design a responsive website, now available at: <http://teamcycl.one> which is both very attractive aesthetically, as well as containing a lot of information, allowing visitors to easily find out more about the project and indeed us. The website even has an embedded blog where each team has so far written an article, including the Software and Electronics team. We hope to continue doing this and even make them more frequent as the launch draws nearer. This website has been extensively promoted using our social networks, and so far has encountered over 2000 page views, thus allowing this many people to learn more about our project. The website proved an interesting challenge for us, as we attempted to use all aspects of web development to our advantage, using JavaScript to allow the website to be responsive, CSS for stylistic aspects, and HTML(5) for the core programming. Additionally, a mobile website was designed, as this responds to the idea that more and more of the visits we would receive would be from mobile devices. The mobile website was designed to be as simple as possible, while still very usable. This has proved very successful, as the average time spent on our website (approximately 2 minutes on other devices) is closer to 4 for mobile (cite Google Analytics). Finally, we decided to produce mobile apps for iOS, Android and Windows Phone, in order to reach a greater audience even more easily. To do this, we employed the abilities of the Software Teams, producing apps in Swift, Java and C# respectively. To date, Android and Windows Phone apps have been launched to their respective App Stores, and been successful, with the Android app having more than 400 downloads, however, we are having a few problems with the iOS application. Though it has been designed, the app is currently being rejected by Apple as it’s market is too ‘niche’. However, we hope to make the app more general for engineering and especially St Paul’s in order to counter this problem.

# Can Code

The majority of progress in the can code has been on the libraries that govern how we can read from the various sensors. This has largely been delayed as the majority of the coders, such as Ashwin and William, have been highly involved in the Electronics Design and choice of components, where software actually played a large part in why specific components were chosen. Ultimately, most of the components chosen are going to use the relatively simple I2C protocol, apart from the GPS sensor which will make use of TTL Serial, another well documented protocol. Unfortunately, many of the components do not have recognised libraries, so we are producing our own libraries. We will produce two main libraries, one for the sensors, and one for communications. The Sensor library will manage the reading of data from all sensors which will in fact make use of other libraries for each sensor. While the MS5637 and HYT271 libraries are being made in-house, the library for the IMU has already been produced by Sparkfun, so we are making use of this. For the GPS sensor, we are going to use the TinyGPS++ (an open source GPS library) to parse the NMEA statements that the breakout produces. This library is very well acknowledged and the team has lots of experience having used it in the past (though with other GPS units). Additionally, we are also planning to produce a communications library, to simplify the use of the Hope RF98W in the actual program. However, for this, we are working off the base of the Team Impulse Library, which was produced by William as part of last year’s competition (under the MIT licence), which makes the job much simpler. However, it still must be adapted in order for it to work most efficiently for us. To date, much of the MS5637 library has been written, but much work must still occur.

Additionally, a basic plan, illustrated by the flowchart below, for the main loop (since we are using Arduino, a language which has a cyclic program structure), where the data is collected and then transmitted.

Open arms of can upon command

Void Loop()

Void Setup()

Communications, using RF98W library

Include Libraries

Initialise Sensors and motor (using Sensor library)

Read from sensors, using Sensor library

Though there is a lot of work to do, we believe that with much of the software team freed from electronics duty, this could happen fairly quickly, allowing us to have a working prototype of the completed code by Christmas, the original aim.

# Base Station

Little work has been completed on the Base Station software, beyond deciding what the base station software entailed. However, we have now decided to write the software in C#, probably using the very popular GTK+ framework, due to the familiarity of the main coder on the project with this, with in fact, Team Impulse’s Base Station Software having been written using this. However, beyond receiving the data, the base station has many complex objectives.

The specification is as follows:

1. The software should parse and store sensor data from the Can (n a sensible file format)
2. The software should show the live camera feed from the Can
3. The software should display the current rover location, preferably on a map (making use of Google Maps or OpenStreetMaps)
4. The software should use the incoming data to calculate altitude (using barometric pressure) and the agricultural viability (using the predefined algorithm)
5. The software should allow you to open the arms of the can, by sending a command over the RF98W
6. The software should be able to send commands to the control board of the quadcopter
7. The software should be able to autonomously monitor the position of the quadcopter and move it to a user-defined location, using (6)

Electronics

The software and electronics were largely split into three parts, which would not be interconnected except for sharing the same power source, as we wanted to ensure a fail-safe that if one system failed, the others would continue to work. Additionally, large parts of the flight system were bought in, as attempting to produce them would be useless, as the probability of things working would have been relatively low. Additionally, to provide more flexibility, the system that would be used in order to open the arms, with the single motor will be connected to the sensor system. Additionally, this means we could trigger the system to work with changes in altitude if we wanted (though this may be too risky).

# Flight System

For the quadcopter system, the first thing we decided was the exact parts we would choose. The actual types of parts we required rather decided themselves, since we did not have the space to add any unnecessary parts. Thus only vital parts were chosen. The first main decision that we made was that we would buy as many parts as possible for this system given the high complexity required to keep a quadcopter in flight, and the lack of space and time for ineffective parts that we had. Additionally, parts such as ESCs and the Control Board, which we could theoretically replicate were very good and cheap, and since we were sponsored by HobbyKing, became free. Below, is the list of the exact parts we chose and why we decided that they were the right choices.

|  |  |
| --- | --- |
| Battery – Turnigy Nano-Tech 3s 850mAh | The power supply system is vital to the entire project, since any failure in the system could prevent the operation of the entire CanSat. We need to thus ensure that all components of all systems receive a safe voltage. Additionally, the battery needs to be very small, given the lack of space that we have in the can. However, the smaller the battery the lower the capacity of the battery. In order to maximise the capacity per unit area, the use of a LiPo battery is the best choice. Though it is a very powerful battery, there are a few issues related to the danger aspect of the battery, since LiPos are liable to explosions if they are overcharged, dis-charged or indeed short-circuited. However, certain batteries, such as the battery that we have chosen to use has specific protection built into the system to ensure that there is no excess in voltage or indeed current, which could be damaging. Though this could have been achieved using a Zener Diode and a collection of transistors, or indeed specific Integrated Circuits, it would have taken up some space and have probably been less effective than the system integrated in the battery.  Given that the motors have to lift 370g of the can, the majority of motors we could use appeared to require the use of a 3s (11.1V) battery, so we chose to use this one, since it was the one which was the most rounded, thus best fitting into the existing mechanics of the can. Additionally, using our existing expertise and extensive online research we have found that this should be able to provide around 5 minutes of flight time, our aim, while also powering the sensor system. However, also through testing, we have found that the camera system takes a lot of power, so we hope to include a high-current logic-level MOSFET to control the camera remotely, so that we could keep the camera off when the can is idle.  In order to distribute all the power according to the wiring chart (see Figure ), we have designed a power distribution PCB, (see Figure ) in order to allow power to go to the correct places. Additionally, on board we have a 5V voltage regulator, which will provide the correct voltage for the motor which opens the arms of the quadcopter. We have chosen to rely off the voltage regulator of our microcontroller (the Teensy 3.2) to supply the 3.3V required for the rest of the system.  C:\Users\Ashwin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture31.png  *Figure : Power Distribution PCB (designed using DesignSpark PCB)* |
| Motors – Turnigy Outrunner v2 | For the motor choices, we were forced to consider two important aspects, the size of the motors and their power. In order to provide a little leeway, we wanted to ensure that the motors we chose had at least a thrust of 400g, thus it could lift 400g. This meant that the four motors together had a thrust of over 1.6kg, more than enough to comfortably lift the 370g of the CanSat. In order to calculate the thrust, we used online calculators. We soon found out that we would not struggle with thrust, with the majority meeting our specification, but would with size. The Turnigy Outrunner v2 motors were selected because they were the smallest motors we could find but one of the most powerful. Though normally very expensive, the sponsorship by HobbyKing (from whom we purchased the motors) meant that they were the best choice. |
| Control Board – HobbyKing i86 | This control board was chosen as it is very simple, with little to go wrong, while having sufficient features, such as auto-stabilisation built in, using a barometer. Additionally, a member of our team (Daniel) had used the board before and had found it very easy to use. Additionally, we could not purchase a more complex control board because of space concerns which meant the 40x40mm size of the board was very helpful. |
| ESCs – Turnigy Nano Tech 20A | Given our motor choice, we knew we needed a 20A ESC. By working together with the Mechanics team, we chose this ESC, as it was long and thin, thus would fit in the arm, as required most easily. Additionally, the ESC was highly recommended with most users online having been able to use it with few issues. |
| TX / RX system – Orange Nano | This TX / RX system was chosen because it was very compact, thus most easily fitting in the little space we had in the Can. and very highly rated by many other users, who had found it easy to set up. Additionally, Orange, the manufacturer is a high end manufacturer, and normally produce reliable products. |

# Sensor System

## Components

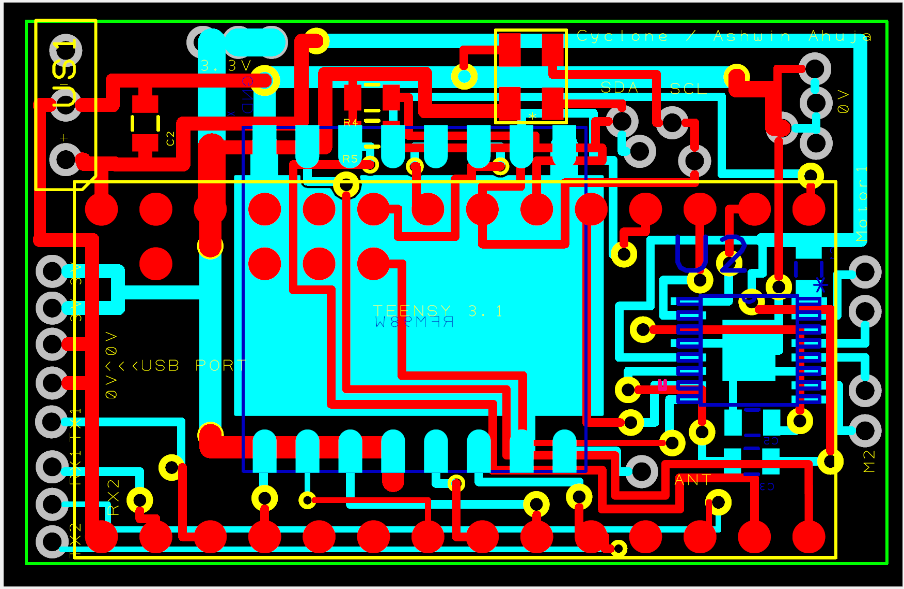
However, the first choice we made was to use an Arduino microcontroller, an obvious one, given the team’s familiarity with it, and the vast availability of parts and examples. We chose to use a microcontroller board rather than a microcontroller, given the significant amount of supporting circuitry required for regular operation and further circuitry required for reprogramming. We have chosen to use the ‘**Teensy 3.2**’ a very small board, which includes all the equipment required to reprogram the Can. Additionally, it is very powerful, containing an Arm Cortex M4, far superior to the Atmel chips on other Arduino boards. It also contains a 3.3V voltage regulator, rated for 500mA which we could rely off. For many of the components, we chose to work off many of the choices made by Team Impulse, of whom a couple of members (including their Team Leader (and Head of Software and Electronics) – William Eustace) we had inherited. Thus we immediately chose to use the **MS5637** and **HYT271**, the pressure sensor and relative humidity sensor that they had used, since they had been used very effectively. Though we had considered using the BME280, a Bosch sensor which included temperature, pressure and humidity sensing, we felt that it was too inaccurate, and too small to feasibly be soldered by hand. Additionally, we chose to harness both the MS5637 and HYT271’s abilities to sense temperature to find a more accurate temperature of the surroundings by averaging their results. We also chose to use the **Hope RF98W**, since by using Spread Spectrum Technology, the sensor is able to more accurately send all the information, as well over a longer distance. In fact, even without the use of a Yagi, the sensor has found to be able to send data with little error over 3km, a larger distance than we would ever encounter over CanSat. Additionally, the RF98W can be used to perform cyclic redundancy checks, ensuring the amount of data received is equal to the amount expect, which would allow us to ensure that errors in receipt of data can be ignored. Finally, we will be using the **DRV8833-PWR** as a Motor Driver for the motor that opens the arms, since it was found to be simple and effective by Team Impulse. Though the even more simple L2D93D was being considered, it was discovered that we do not have the space on the PCB for this chip.

However, from here the similarities with Team Impulse’s electronics end, as we chose different parts. Firstly, we have chosen to use the **GP-2106** GPS module as it has been very effective in testing, and has a very small footprint, much smaller than the GPS module used by either Team Colossus or Team Impulse (last year’s teams). Additionally, we have chosen to use the Evaluation Breakout produced by Sparkfun, to help use the module, which has proprietary connectors, and makes use of 1.8V logic. Finally, we are going to use the **Sparkfun 9DOF Breakout**, as an IMU, containing a 3-axis accelerometer, 3-axis gyro and 3-axis magnetometer. This is because it is very small, very accurate, and is well supported by Sparkfun, with them having produced a very good Arduino library for the device.

## PCBs

The main sensor system PCB is shown below:

*It was designed in RS / Allied DesignSpark PCB 6. All PCB files are available on the GitHub repository, under the Electronics Design section.*



The board shows how the Teensy 3.2 is mounted on the top layer of the board, using Surface Mount Soldering. This was done so that we may conserve space, and could be achieved by ensuring that some insulation is placed between the board and the Teensy, and then soldering of the pads of the Teensy directly to the pads on the board. Also on the top layer is the MS5637 – a 4 pin QFN package, as well as resistors required for the I2C line. On the bottom side, there is the Hope RF98W, breakout which will be surface mounted to the PCB. There is a ground plane under this, as recommended by the manufacturers, in order to reduce noise. Also, under the Teensy, (labelled U2) there is the Motor Driver (DRV8833-PWR), using the HTSSOP-16 package, and associated resistors and capacitors. Finally, dotted around the board, there are a number of connectors for the many breakouts and sensors which must be on flying wires. Given this PCB is at the bottom of the Can, the Humidity Sensor will be just below, so that it may have exposure to the air. Additionally, the IMU and GPS evaluation board will be on the same layer, attached with short wires. The GP-2106 module, however has to be at the very top of the device, to ensure that it can get a fix, and find the Can’s position.

# Camera System

For the camera system, we have chosen to use a pre-made FPV system, with the Mini FPV Camera combined with the HobbyKing FPV Transmitter. Though we had considered using an Intel Edison and WiFi transmission with a NTSC webcam, it transpired that it would take up more space, as well as being a higher quality. Thus, we had the choice of 5.8GHz or 2.4GHz FPV. Though there were some problems regarding legality of using certain powers of 5.8GHz transmission, through more research, it is clear that one can use devices with a transmit power of under 25mW without the necessity of an Amateur Radio License. Though this severely limits the quality of image we can use, it is still superior to the quality of products one receives when 2.4GHz is used. Additionally, there are similar issues regarding the maximum transmission power of 2.4GHz. Finally, we have also chosen to include an OSD (On screen display) of the battery stats of the quadcopter, so the pilot would know when the quadcopter is about to run out of battery. Though we have tentatively ordered one from HobbyKing (in fact the HobbyKing OSD), we feel that it is actually inefficient, and in future revisions we hope to include a custom OSD in the power distribution board.

# Communication

Given the large amounts of data required, a number of different data transmission methods are being used to stream data to and from the Can. As most RC systems work of 2.4GHz, we are following this trend, as this allows to exploit the large number of small TX and RX units specifically built for RC helicopters and indeed quadcopters. Additionally, the use of this transmission method links up well with the HobbyKing Control Board, without much work, thus allowing us to simplify the process required to get the flight system working manually, thus allowing more time to produce autonomous movement.

For the camera system, as discussed above, we have chosen to make use of the 5.8 GHz transmission systems, though there are some problems with the low penetrating power of this high frequency. However, given the low distance that the CanSat will travel, we have concluded that this is not a large problem.

Finally, for the sensor system, we are going with the tried-and-tested 434 MHz RF transmission frequency, given the high penetrating power and thus large distance we could easily get using this protocol. This ensures a high reliability of data transfer, and though this means we could transfer less data, this is not a large problem, since the data, probably 32 bits in length, would only need to be transferred once a second. Given the specifics of the spectrum provided to us, we can also begin to determine the specifics we can use. We have been provided with a specific frequency of 434.07 MHz. Given the teams around us have 433.98MHz and 434.25MHz, a safe bandwith would be around 10MHz, thus the nearest setting, 7.8kHz will be used. Making use of the LoRa Modem Calculator Tool made by SemTech, this will allow a data rate of 417 bps, assuming the lowest possible Spreading Factor (6), with a link budget of 150 dB and a resistor sensitivity of -130 DBm.

# 