Modeling & Testing of a Nano-Satellite And Implementation of a BalloonSat With Data Analysis

A Thesis submitted to the
Department of Electrical & Electronics Engineering
Of
BRAC University
By

MD. Ishraque Bin Shafique - 12321015
Fazle Rabbi - 12321048
MA.Razzaq Halim – 12321026
Arik Mahmood – 12321074

Supervised By
Dr. Md Khalilur Rhaman
Associate Professor
Department of Computer Science and Engineering
School of Engineering & Computer Science
BRAC University

In partial fulfillment of the requirements for the degree of
Bachelor of Science in Electrical and Electronic Engineering
April 2016
BRAC University, Dhaka

Declaration:

We hereby declare that research work titled “Modeling & Testing of a Nano-Satellite And Implementation of a BalloonSat With Data Analysis” is our own work. This paper has not been presented elsewhere for assessment. Where materials were used from other sources it has been properly acknowledged/ Referred.

Signature of the Supervisor
Dr. Md Khalilur Rhaman

Signatures of the Authors

Md. Ishraque Bin Shafique

Md. Razzaq Halim

Arik Mahmud

Fazle Rabbi

Table of Contents

Acknowledgement

We are very thankful to Dr. Md Khalilur Rhaman sir for giving us the opportunity to conduct this thesis program, without his endless support this thesis would not have been possible.

We are grateful BRAC University for providing us with all the resources we needed in order to conduct our work. We were very fortunate to receive their endless support.

We must thank Asst. Prof. Arifur R. Khan of Kyushu Institute of Technology for his help regarding our structural analysis. Adding to that we would like to thank Abdulla Hil Kafi, Maisun Ibn Monowar, Raihana Shams Islam Antara and Munir Muhammad Maruf for giving us their support and guidance. Never the less, we cannot thank Shifur Rahman Shakil enough for being our light in the darkness.

Finally, we would like to thank Md. Ashfaque Bin Shafique for his never ending support throughout the entire period of the project.

ABSTRACT

Satellites are very expensive for developing countries like Bangladesh to facilitate, for which these countries can approach Nano-satellite or balloon satellite which are cheaper compared to a satellite. A Nano-satellite should be designed in such a manner that it can withstand the harsh environments of space and meet all the criteria of carrier vehicle companies. On the other hand, a balloon satellite is much cheaper, easier to construct and is an excellent illustration of a Nano-satellite although it could not perform all the possible functions of it. This paper demonstrates a remodeled version of a Nano-satellite model proposed by an earlier group from BRAC University and also the structural feasibility of that remodeled structure by performing a vibration analysis on it. Additionally, it also implements a balloon satellite circuit for monitoring crop health and deforestation activities by sending images over which are later processed on the ground station. The circuit also sends latitude, longitude, temperature, pressure, light intensity, pitch, roll, heading, UV index, infrared, CPU temperature data using sensors and modules, the balloon satellite is tracked using the latitude and longitude readings. The data transmission occurs wirelessly and is maintained by a Raspberry Pi B+ which runs by itself on boot. These data are received on a ground platform where they are displayed in a GUI in a proper manner and are being saved as well.

CHAPTER 1
INTRODUCTION

1.1 Introduction

Space age technology has become essential for development, with a satellite a country has more knowledge of what is going on within its boundaries and also around it to maintain national security, weather forecast and others. Those satellites are very expensive to obtain and also require huge amount of funds to maintain, which is very difficult for developing countries like Bangladesh to facilitate. Due to the lack of finance, developing countries resort to smaller versions of satellite such as Nano-satellite or BalloonSat, which are cheaper compared to the larger ones. Although they are smaller in size and can perform fewer functions, they still can provide with enough information that can open doorways to space research. With a strong and durable Nano-satellite, a country is looking forward to many years of information that can help develop many sectors such as agriculture. On the other hand, a BalloonSat, is much cheaper with a tradeoff of shorter flight period. Both Nano-satellite and BalloonSat can provide with valuable information which can be received on a ground station. The information can contain data or image, data that can be used for understand the condition of our environment or for research purposes and images to identify certain aspects of interest such as the color of the crop leaves to estimate nitrogen levels, which in return could help us improve the yield to crop production. This application could help develop countries like Bangladesh much faster.

1.2 Literature Review

A group from BRAC University proposed a model for a possible Nano-satellite structure in 2015, their aim was to design the structure in such a way that it could endure and remain stable in space. The structure consisted of 6 pyramids clipped together, giving the overall structure many openings for solar panels and a cubic space at the center for the battery. As we know, Bangladesh is a victim of many natural disasters such as hurricanes and typhoons and others. Due to heavy rainfall, floods occur damaging the agricultural lands of the country which in return harms the country's economy since Bangladesh is an agricultural country. Therefore, they attempted to construct a Nano-satellite for Bangladesh in order to monitor the weather, but due to the lack of equipment and other resources were unable to do so. So instead they proposed the structural model and hoped for support from expert countries.[1]

A BalloonSat program conducted in University of Colorado at Boulder intended to give students the opportunity to work on satellite, their intention was to introduce engineering students to the challenges faced when working with satellites. Since satellites are out of reach for students, they are instead introduced to BalloonSat which is a cheap alternative to satellites. The students implemented a number of sensors to retrieve data from high altitudes and also to track it. The small satellite circuits were attached with a parachute and a balloon and was intended to float until the balloon burst. The data was collected from the satellite after it landed back from its flight. They collected temperature, humidity and other data for illustration purposes.[2]

A BalloonSat with an onboard camera can be used in a broad spectrum of image processing. One such perfect example is to estimate the health of crops. By this process a large vegetation area can be sampled in a short time. We have studied a paper written by a research team at Stanford University related to this technique. In this thesis paper, however we have implemented image processing algorithms to determine the nitrogen contents of plant leaves in a field.

In the following chapter we discussed about a model and the theories of random vibration analysis. In chapter 3 we approached a BalloonSat circuit with off the shelf components. Following the data received from the BalloonSat, we conversed about two possible applications in chapter 4. The result and analysis of our thesis were discussed in chapter 5.

CHAPTER 2
STRUCTURAL ANALYSIS

2.1 Model Description

Nano satellite provides an excellent opportunity for our country as it will be affordable and its varied applications can be used to benefit our country. Before moving on to the applications of the satellite, we have to make sure the body of the satellite adheres to the requirements of the launcher vehicle and is properly tested in an environment simulating the harshness of the space. In this chapter, we have presented the physical model and its specifications, talked briefly about design specification and lastly explained the tests we have performed on the model.

Our physical model was an extension of a proposal by a thesis group from BRAC University on their paper titled "Exploring Modular Architecture for Nano Satellite and Opportunity for Developing Countries" as stated earlier. Their structure is given in Appendix-8.1 in figure-41. Adding to the idea of six individual pyramid structure whose vertex has been clipped; we have reimagined and assembled our CAD model with a set of objectives in mind.

- Dimension of the Nano satellite is to be 10cm*10cm*10cm.
- The mass of the satellite has to be less than 1.33 kilograms.
- Deployable should be constrained by the Nano satellite.
- A system of auto deployment of solar panels.

3D model of the satellite was produced from scratch by ourselves using a CAD software. We produced each pyramid face on its own and then mated the six individuals together to form a CubeSat structure. The biggest advantage of this process is each pyramid body can be tested on its own reducing the cost of producing expensive prototypes. Each of these pyramid faces will have support structure at the four corners where the protective plates will be screwed in. Also two extruded support are in the inner two faces (opposite to each other) for the electronics. The remaining free inner faces are used to position the hinges. One hinge is placed higher up than the other so that solar panels, when un-deployed will be on top of one another. The final assembly is a 10 cm cube with mass of 420 grams when using structural steel as the material. The inclusion of battery, integrated circuits, sensors and solar panels will add to this mass. On figure-1 the full model and exploded view is visible. It consists of six independent pyramid shaped subassembly joining together to form a cube, 6 thick plates that protect the inner circuitry, a pair of hinges that control solar panel deployment.

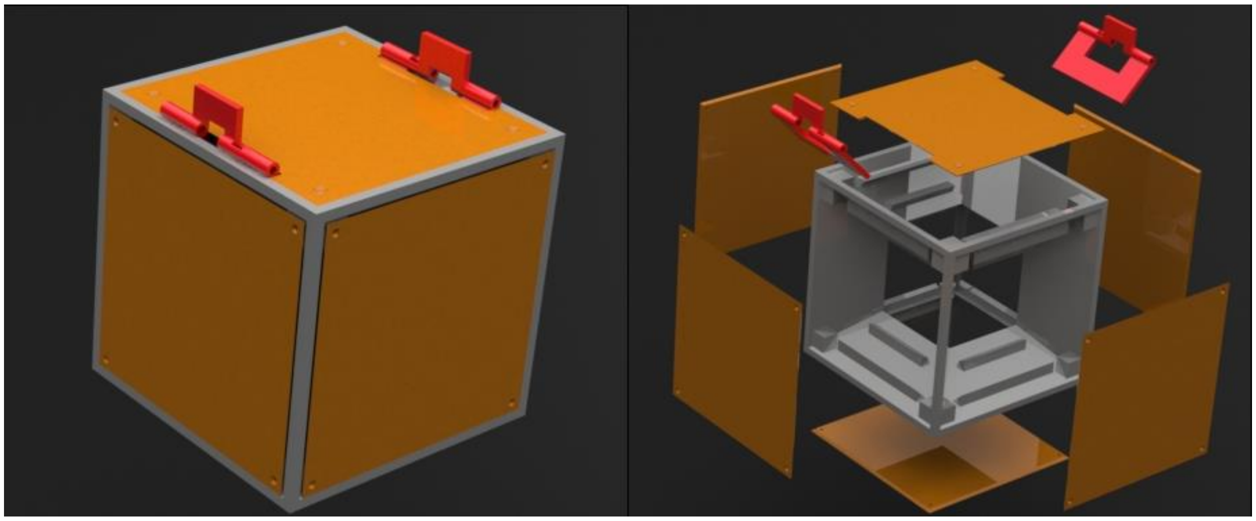


Figure 1: Full Nano-satellite model & exploded view

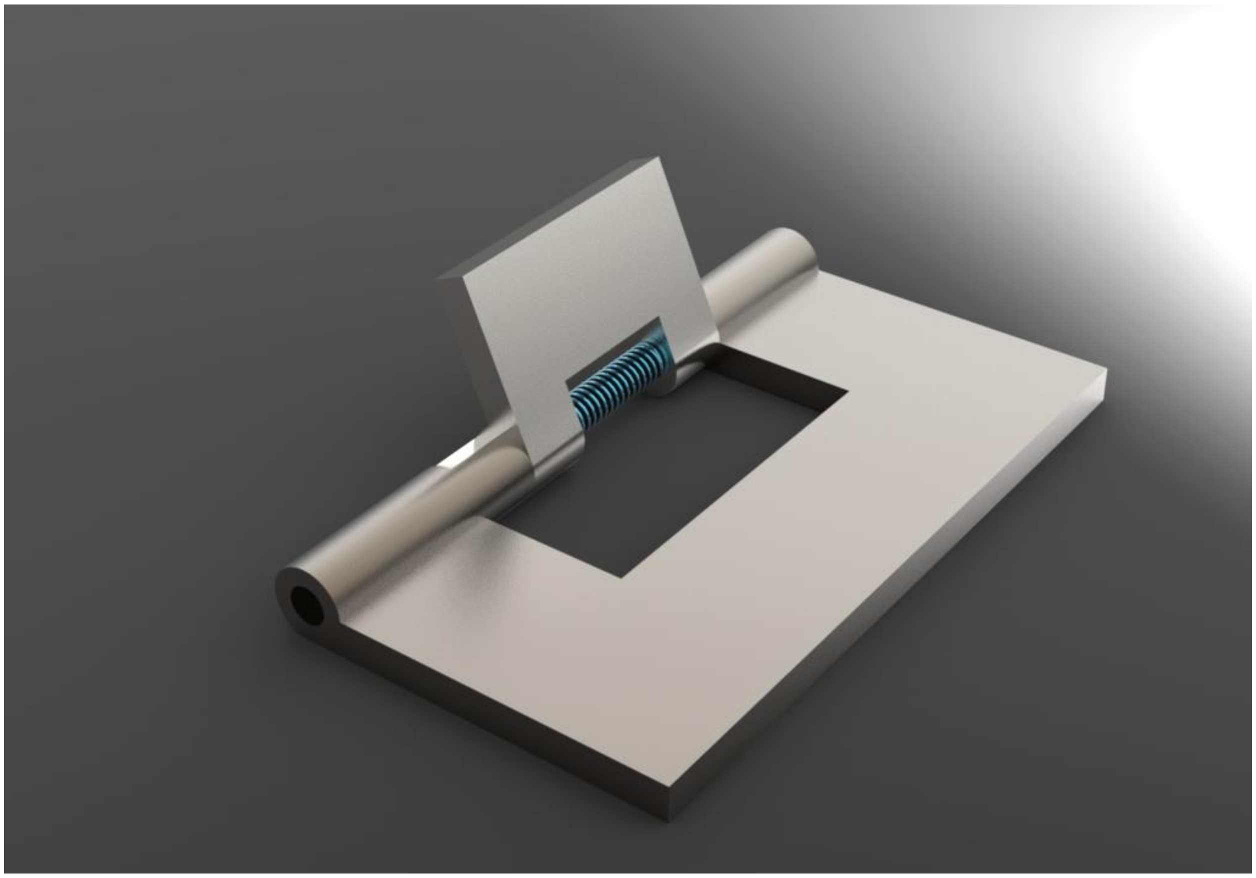


Figure 2: Mechanical hinge with spring

Every pyramid subassembly has two slots for circuits to be placed on. Different sensors can be placed on these slots allowing space for adequate electronic devices to monitor the earth. The empty space on the center is reserved for the battery as it requires the most amount of thermal protection. Next, the hinges are an important mechanical part of this design. It is a 2.5 cm x 5 cm hinge assembly which is kept in place by a 0.26cm diameter 5 cm long pin. The un-deployed and deployed angles with the subassembly surface are 45° and 180°. As seen on figure-2 and figure-3 the hinges allow for a fixed amount of movement by mechanical means.

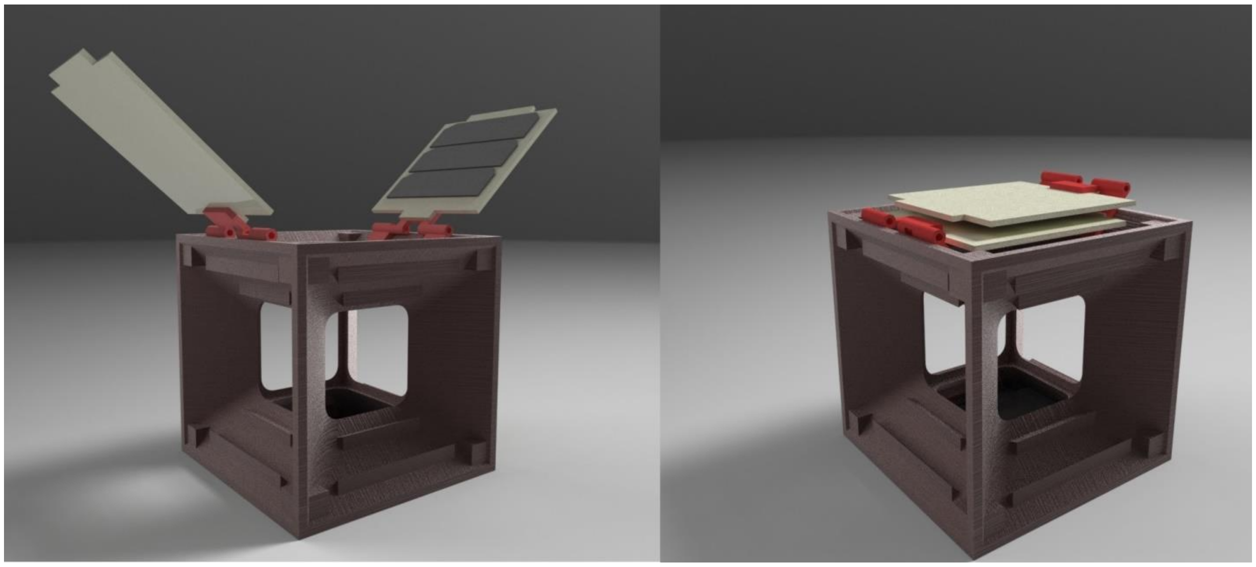


Figure 3: Solar panels in un-deployed and deployed position

We also note that more of these hinge pairs can be added to the other pyramid subassemblies to support more solar panels if power requirement increases. This is particularly important because this provides us with a modular design and also the previous group's proposal of reaction wheels to control the orientation of the satellite that required huge power can be met. A model with all solar panels attached is shown on figure-4.

Lastly, the deployable solar panels will be un-deployed up until it reaches its orbit. This will be achieved by fastening the structure with nichrome wires with one end attached to a fuse. Nichrome wire is an alloy of nickel, chromium and iron which is usually used in resistance wires. After releasing from the carrier vehicle the fuse is blown from the ground station via a signal, thus deploying the solar panels.

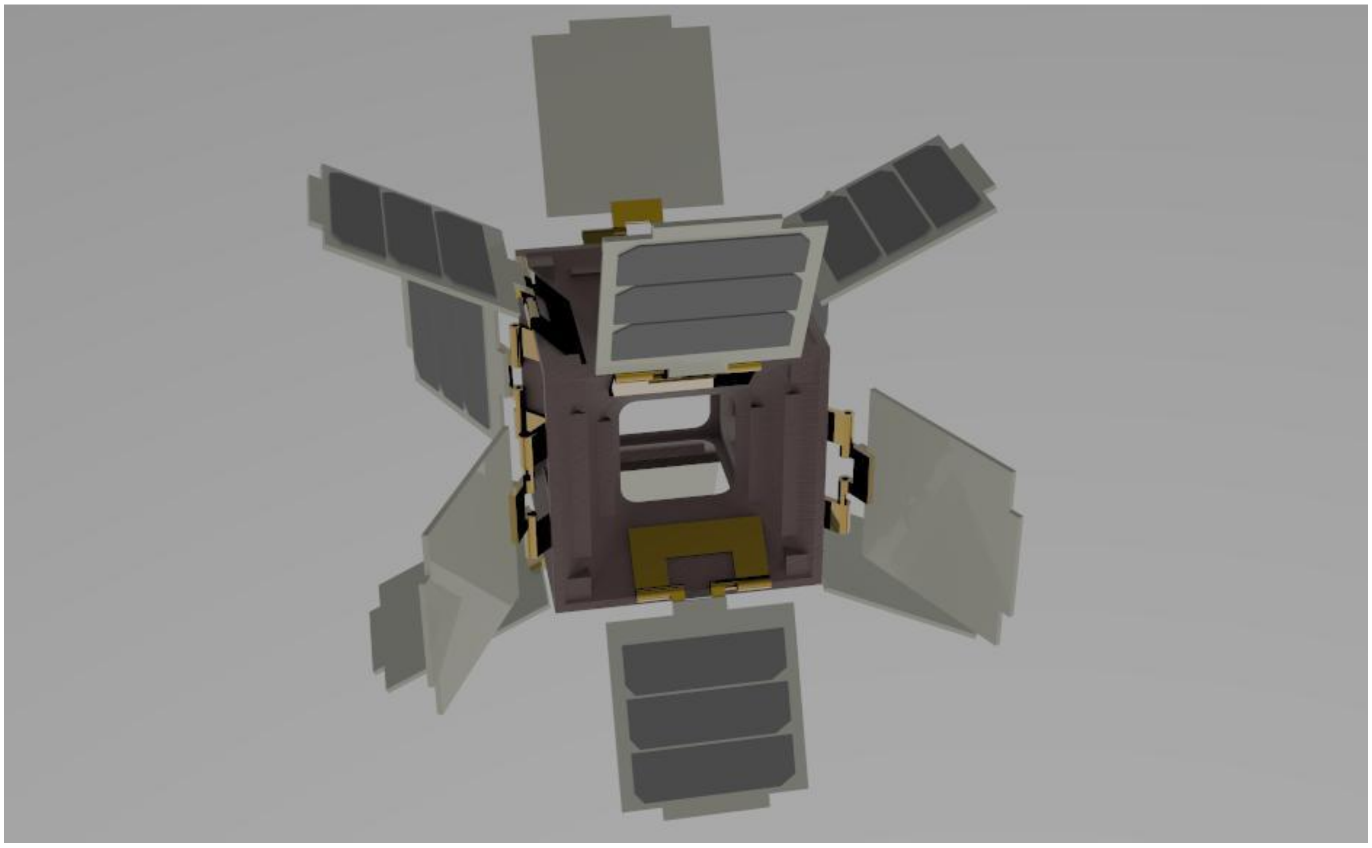


Figure 4: Full model with solar panels in all faces

2.2 CUBESat Design Specification

From revision 13 of the CUBESAT design specification, a CUBESat has to undergo the following tests to ensure safety.[3]

- Random Vibration: Random vibration testing shall be performed as defined by the launch provider
- Thermal Vacuum Bakeout: Thermal vacuum bakeout shall be performed to ensure proper outgassing of components. The test specification will be outlined by the launch provider.
- Shock Testing: Shock testing shall be performed as defined by the launch provider.

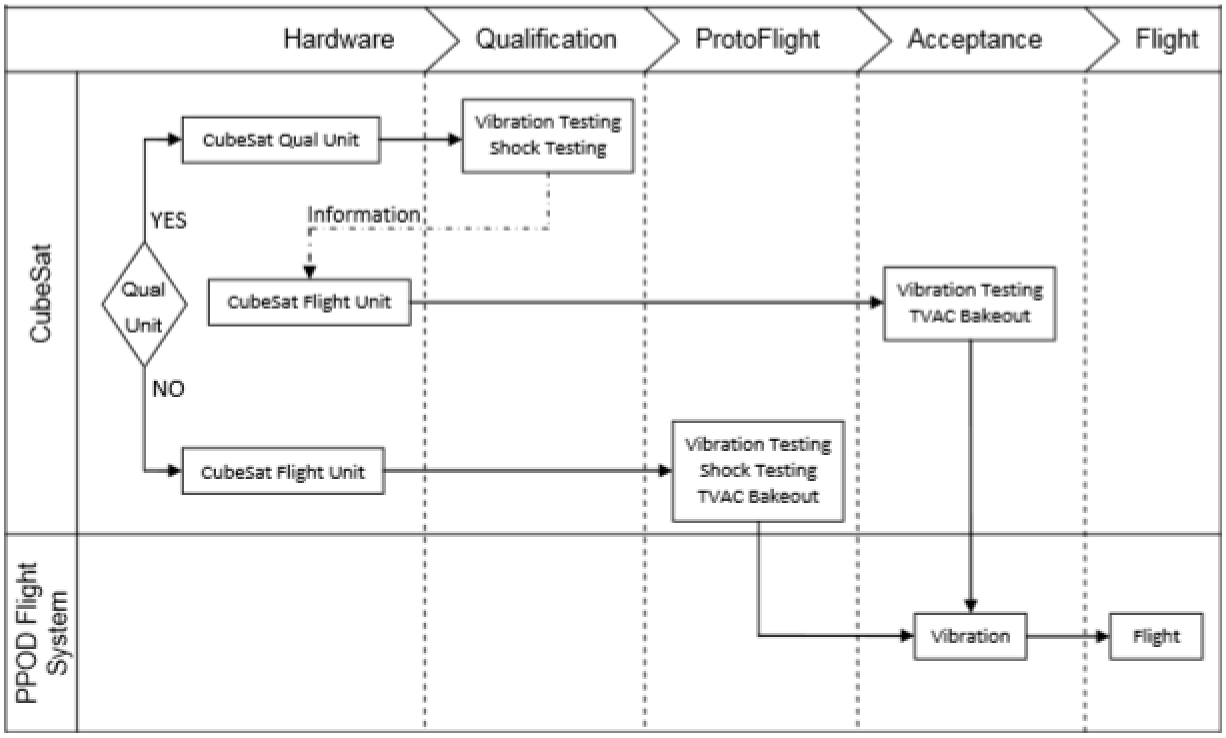


Figure 5: Qualification test flow courtesy of Cal Poly

As per the guideline the qualification test flow is shown in figure-5. We successfully simulated the satellite model against random vibration as per the requirement of a carrier vehicle. However, for lack of resources and external help we could not simulate the other two analyses.

2.3 Random Vibration

Random vibration, commonly known as white noise is vibration whose amplitudes are not deterministic. Random vibration analysis is performed usually on aeronautical structures that undergo high level of vibration. The process of random vibration involves applying random excitation to the structure to find design flaws, weaknesses and any changes that may be needed before the structure is physically produced. The analysis process is a statistical one as random vibration cannot be expressed precisely as a time function. Power spectral density is used as input to obtain equivalent stress and response power spectral density (PSD) plot.

Random vibration is analogous to white light. White light can be sent through a prism to reveal a continuous spectrum of colors as like random vibration can be passed through a spectrum analyzer to reveal a continuous spectrum of frequencies although the generally used meaning of the term "random" is not appropriate for this term. If this term were considered of having no specific pattern, it would not be possible to define a vibration environment, because the environment would behave in an unpredictable way. However, the majority of random processes fall in a special category termed stationary. This means that the parameters by which random vibration is characterized do not change significantly when analyzed statistically over a given period of time - the RMS amplitude is constant with time.[4] Random vibration usually is defined by power spectral density or acceleration spectral density. As random vibration is a statistical method, required statistical terms are given below:

Standard Deviation-standard deviation is a measure that is used to quantify the amount of variation or dispersion of a set of data values.

Skewness- skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive or negative, or even undefined.

Kurtosis- kurtosis is a measure of the sharpness of the peak of a frequency-distribution curve.
Mechanical failure and fatigue life is the most important aspect of random vibration analysis for a designer. When a sensitive device is excited at its natural frequency, relatively large displacements may result in failure of the model. In such a case, the malfunction might be fixed by lowering the amplitude of excitation at the frequency of concern - the modal frequency of the device. This might be accomplished by inserting a vibration isolator between the source of excitation and the device.[3] Alternatively, displacement might be reduced by adjusting the stiffness of the device, or by increasing

damping at the natural frequency of the device. The material of the model will affect the type of action that need to be taken.

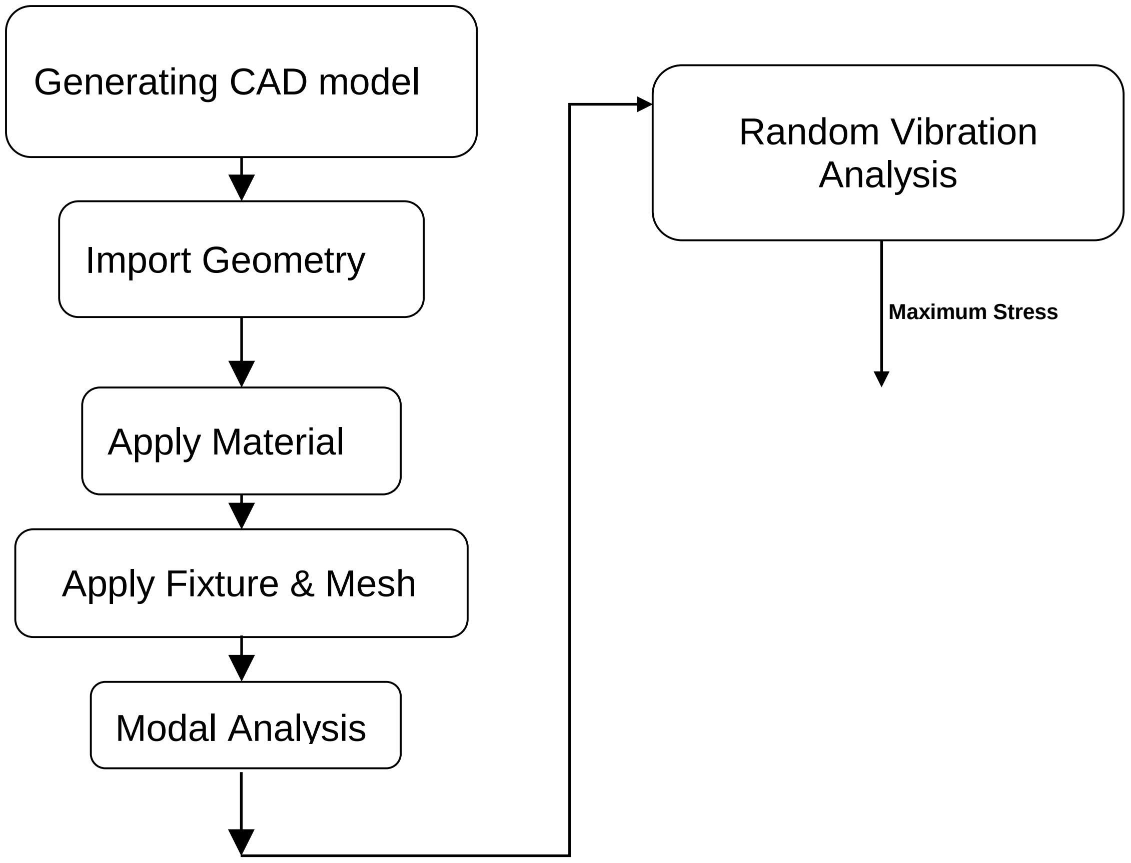


Figure 6:Flow chart of Random vibration process

For a single degree of freedom system, the maximum response from an input PSD can be given by,

$$c_n = \sqrt{2\ln(f_n T)}$$

$$C_n = c_n + \frac{0.5772}{c_n}$$

$$\text{Maximum peak} = \sigma_n C_n$$

Where, $f_n = natural\ frequency$

$T = duration$

$\sigma_n = standard\ deviation\ of\ the\ oscillation\ response$

$n = index$

Random vibration analysis performed on the Nano satellite model and the result is discussed on chapter-5. The step by step process of random vibration is given on the figure-6.

CHAPTER 3
BALLOON SATELLITE
&
GROUND STATION

3.1 Circuit Specifications

A balloon satellite is a smaller version of a Nano satellite, that being said it cannot facilitate us with many information and services. However, it can provide us with basic information and images from a desired height. So we have constructed a circuit that will be able to supply us with data or images from high altitudes.

A satellite performs many various functions and processes which require a capable processing unit, we are using a Raspberry Pi B+ module to handle all the processes we are intending to perform. We used a lot of sensors to collect data such as the UV sensor, IMU module, Adafruit Ultimate GPS module, Raspberry Pi Camera, and we also used the built in temperature sensor of the Raspberry Pi. UV sensor(si1145) is used to sense the UV radiation (purpose of reading change in intensity with height). The IMU module is a collection of few sensors (L3GD20H, LSM303, BMP180) which provides us with many data. The circuitry will be floating around if used as a balloon satellite or in space if applied to a Nano-satellite, so we require a means to track its position and that is why we are adding a GPS module. The Raspberry Pi has a built in temperature sensor which can be used to monitor the Raspberry Pi temperature conditions. This allows us to obtain the circuit boxes internal temperature. The camera was added to help monitor crop health and deforestation by taking photographs. All these data including the image taken is sent to the ground platform via XBee module wirelessly.

3.2 Components Used

3.2.1 Raspberry Pi 2 Model B+

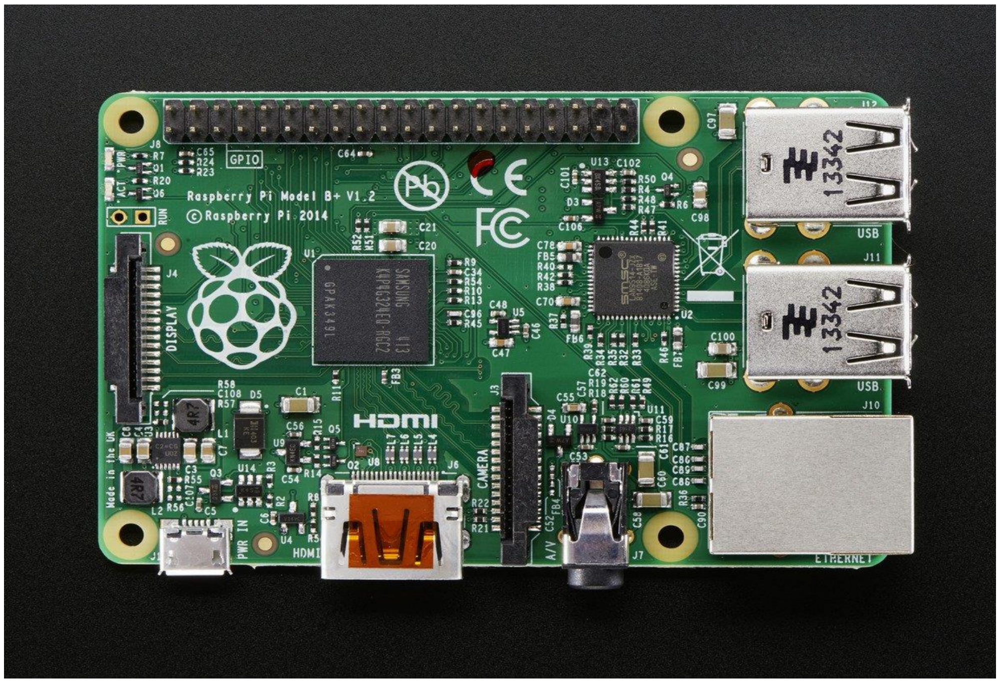


Figure 7: Raspberry Pi 2 B+

Raspberry Pi 2 Model B is a Linux based development platform. It uses a processor with clock speed of 900 MHz (which can be overclocked to 1000MHz) and 1GB RAM. The board has 40 GPIO (general purpose input/output pins) used to communicate/control low level peripherals. This Raspberry Pi can support 64GB microSD card. Online connectivity is achieved through the RJ45 port.

Features:

1) 40 GPIO pins
2) x USB 2.0
3) MicroSD Card Slot
4) HDMI output
5) Ethernet port
6) 3.5mm Audio Port
7) CSI (Camera Interface)
8) DSI (Display Interface)
9) VideoCore IV 3D Graphics

3.2.2 Adafruit Ultimate GPS Module

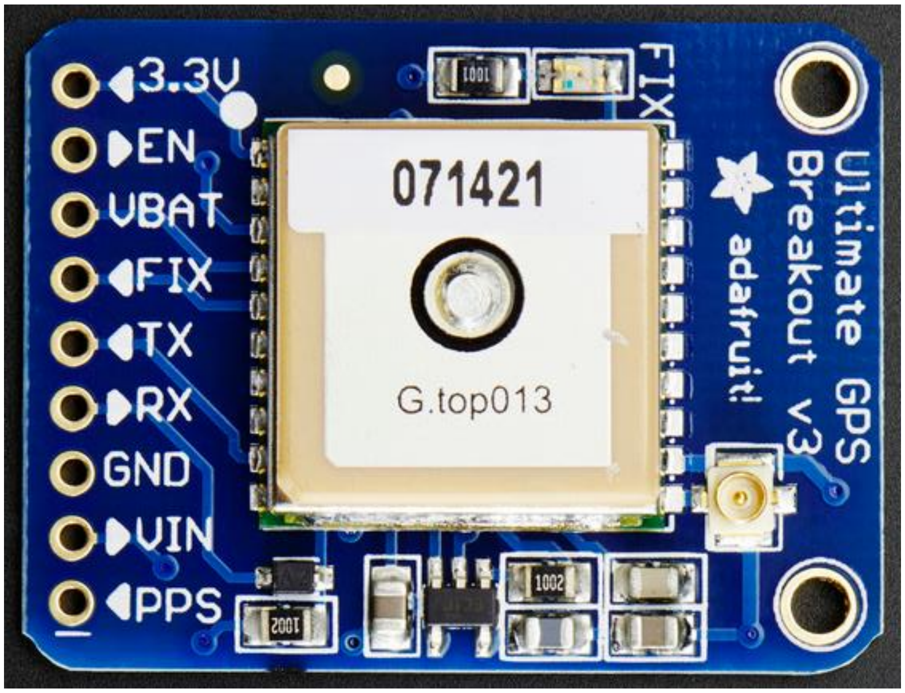


Figure 8:Adafruit ultimate GPS Module

The Adafruit Ultimate GPS Module can track up to 22 satellites on 66 channels, has an excellent high-sensitivity receiver (-165 dB tracking), and a built in antenna. It can do up to 10 location updates a second for high speed. It communicates using serial protocol.

Features:

- Satellites: 22 tracking, 66 searching
- Patch Antenna Size: 15mm x 15mm x 4mm
- Update rate: 1 to 10 Hz
- Position Accuracy: < 3 meters (all GPS technology has about 3m accuracy)
- Velocity Accuracy: 0.1 meters/s
- Warm/cold start: 34 seconds
- Acquisition sensitivity: -145 dBm
- Tracking sensitivity: -165 dBm
- Maximum Velocity: 515m/s
- Input voltage range: 3.0-5.5V DC

3.2.3 Raspberry Pi Camera Board

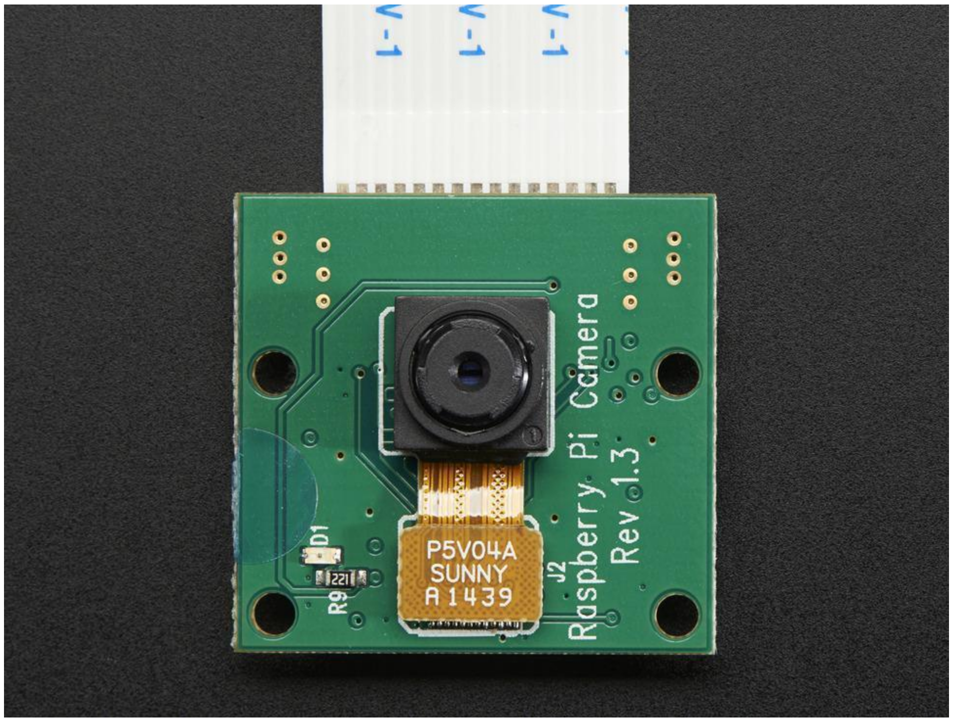


Figure 9:Camera board

The Raspberry Pi Camera Module attaches to Raspberry Pi by CSI interface, which was designed especially for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data.

The sensor itself has a native resolution of 5 megapixels, and has a fixed focus lens onboard. In terms of still images, the camera is capable of 2592x1944 pixel static images, and also supports 1080p30, 720p60 and 640x480p60/90 video.

Features:

- Small board size: 25mm x 20mm x 9mm
- A 5MP (2592×1944 pixels) Omnivision 5647 sensor in a fixed focus module
- Support 1080p30, 720p60 and 640x480p60/90 video record

3.2.4 Adafruit 10-DOF IMU Module

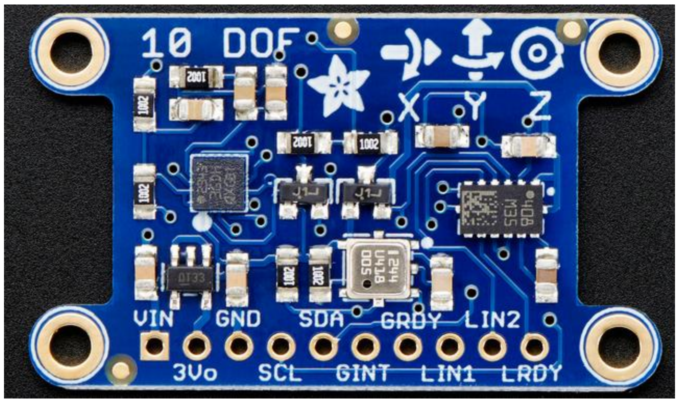


Figure 10: 10 DOF IMU module

This inertial-measurement-unit combines 3 sensors to give 11 axes of data: 3 axes of accelerometer data, 3 axes gyroscopic, 3 axes magnetic (compass), barometric pressure/altitude and temperature. It communicates using the I2C protocol

The Adafruit 10DOF IMU comprises of L3DG20H gyroscope, LSM303DLHC accelerometer and compass and BMP180 barometric/temperature sensors.

Features:

- Length: 1.5" / 38mm
- Width: 0.9" / 23mm
- Height: 0.125" / 3mm
- Weight: 2.8g
- This board/chip uses I2C 7-bit addresses 0x19 & 0x1E & 0x6B & 0x77
- L3GD20H 3-axis gyroscope: ±250, ±500, or ±2000 degree-per-second scale.
- LSM303 3-axis compass: ±1.3 to ±8.1 gauss magnetic field scale.
- LSM303 3-axis accelerometer: ±2g/±4g/±8g/±16g selectable scale.
- BMP180 barometric pressure/temperature: -40 to 85 °C, 300 - 1100hPa range, 0.17m resolution.

3.2.5 Adafruit UV Index/IR/Visible Light Module

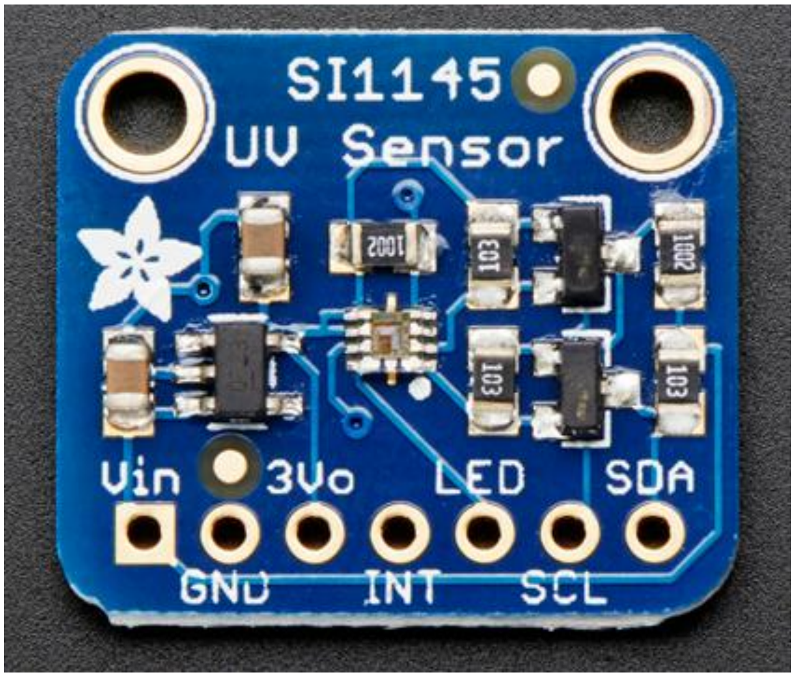


Figure 11: UV sensor

The SI1145 is a sensor with a calibrated UV sensing algorithm that can calculate UV Index. The sensor uses I2C bus and can communicate at very high speeds with the Raspberry Pi. Values of UV Index, IR Spectrum and Visible Light can be read from the sensors. The UV sensor like the IMU also communicates using the I2C protocol

Features:

- IR Sensor Spectrum: Wavelength: 550nm-1000nm (centered on 800)
- Visible Light Sensor Spectrum: Wavelength: 400nm-800nm (centered on 530)
- Voltage Supply: Power with 3-5VDC
- Output Type: I2C address 0x60 (7-bit)
- Operating Temperature: -40°C ~ 85°C
- 20mm x 18mm x 2mm / 0.8" x 0.7" x 0.08"
- Weight: 1.4g

3.2.6 XBee 2mW Wire Antenna-Series 2

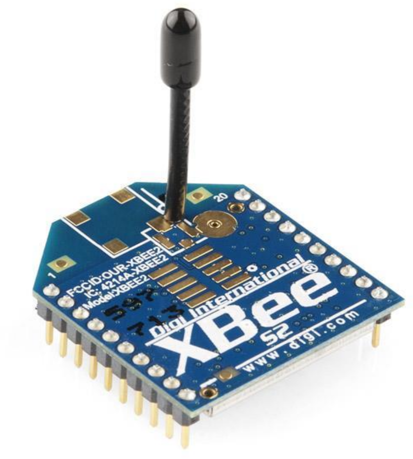


Figure 12: Xbee radio module

The XBee Series 2 is a wireless networking chip for wireless data transmission with which it is easy to create complex mesh networks based on the XBee ZB ZigBee mesh firmware. These modules allowed us a very reliable and simple communication between the Raspberry Pi and the Ground Station PC. Like the GPS module the XBee also communicates using the serial protocol with the Raspberry Pi/Arduino.

Features:

- 3.3V @ 40mA
- 250kbps Max data rate
- 2mW output (+3dBm)
- 400ft (120m) range
- Built-in antenna
- Fully FCC certified
- 6 10-bit ADC input pins
- 8 digital IO pins
- 128-bit encryption
- Local or over-air configuration
- AT or API command set

3.2.7 XBee Explorer USB

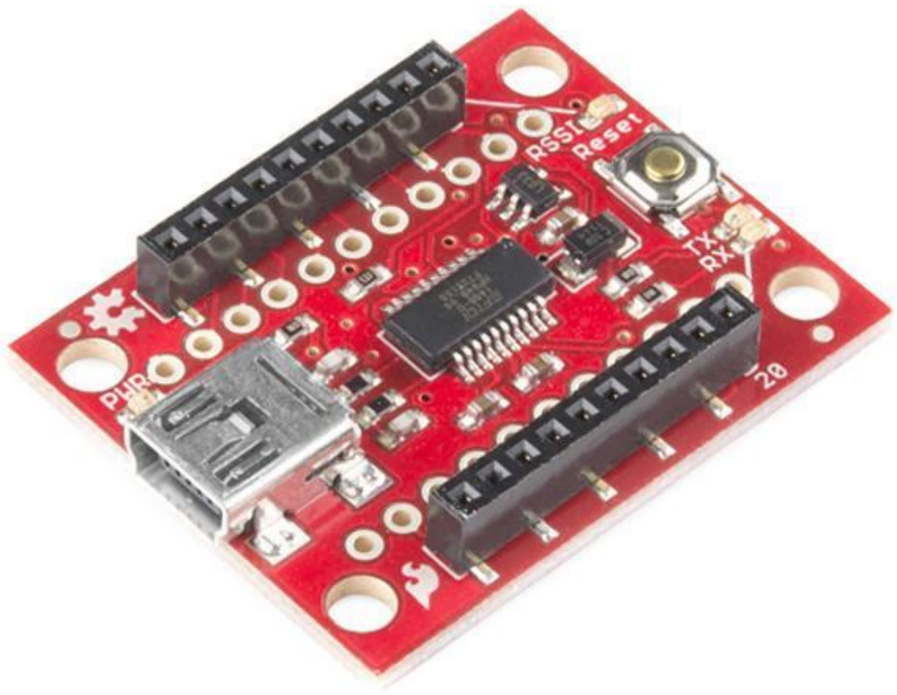


Figure 13:Xbee explorer USB

The XBee Explorer USB is a USB to serial base unit for the XBee wireless module. The main feature of this board is an FT231X USB-to-Serial converter which interprets data between a computer and the XBee. There is a reset button, and a voltage regulator. In addition, there are four LEDs that helps to configure the XBee: RX, TX, RSSI (Signal-Strength Indicator), and power indicator.

3.3 Circuit Assembly

The circuit is connected as shown in figure 14, the Raspberry Pi supplies power to all the components except the GPS module as it requires close to 5V, since the Raspberry Pi is supplying all the other components it is unable to supply exactly 5V to the gps which in return will end up adding noise to the gps data. The gps may even fail to get a fix and so the gps instead draws power directly from the battery. An Arduino shield is connected with the XBee module, it has no function but the issue was that the XBee pins spacing was too small and rails that match its spacing was not available, so the Arduino shield was connected so that we can connect it with the rest of the components on the same board. The voltage regulator regulates the voltage down to 5V, in consists of LM2940 chip, two 10µF capacitors at the input side and one 0.1µF capacitor at the output. The circuit was then assembled on a pcb (printed circuit board) board as shown in figure 14, the Raspberry Pi is attached below the pcb and the battery and camera are placed on a plane board below the Raspberry Pi.

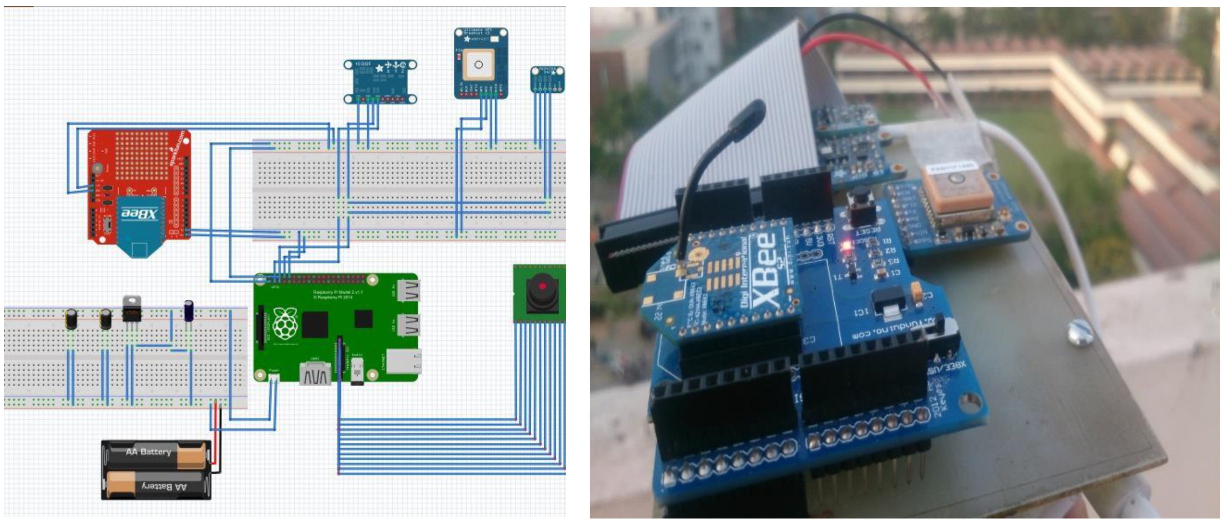


Figure 14: Circuit diagram of BalloonSat and the actual integrated setup

3.4 Communication Protocols Used

3.4.1 I^{2}C BUS

I^{2}C bus allows multiple slaves devices (up to 1008 devices) to communicate with one or multiple master devices whereas the SPI can only support one master device, it is preferred for short distance communication at a rate of 100HZ or 400HZ between digital circuits and only requires two pins or connections to do so whereas the SPI requires four connections, the two connections being SDA and SCL. SCL synchronizes the data transfer over the i^{2}c bus while the SDA is used to transfer the data which gives it an advantage over the serial communication, due to this synchronization no data overlapping occurs.

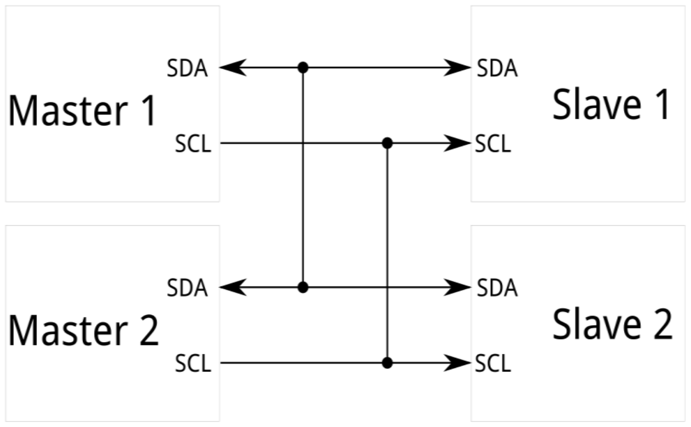


Figure 15:Basic i2c connection between master and slave devices

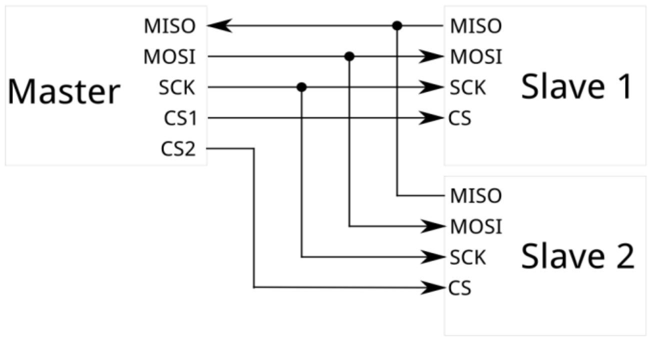


Figure 16: Basic SPI connections

3.4.2 Serial Communication

Serial communication is an asynchronous form of communication where data is sent one bit at a time over a channel while in the parallel communication multiple bits are sent at a time over multiple channels increasing the number to wires.

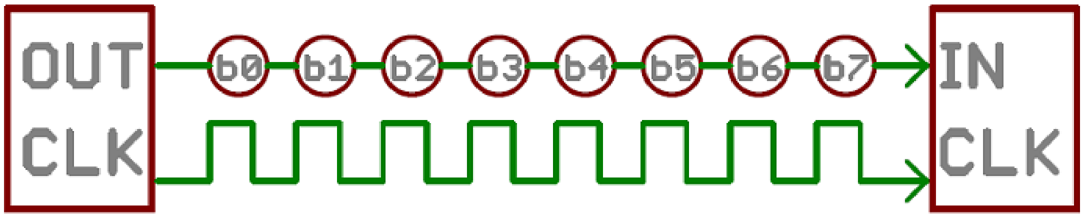


Figure 17: Serial communication sending one bit per clock

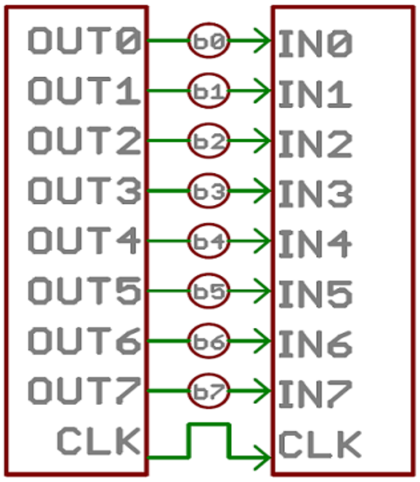


Figure 18: Parallel communication

Nowadays to reduce the number of wires, the clock has been removed which is why the serial communication is asynchronous. Baud rate is what determines how fast the data is being transmitted, it is required for both devices to operate at the same baud rate

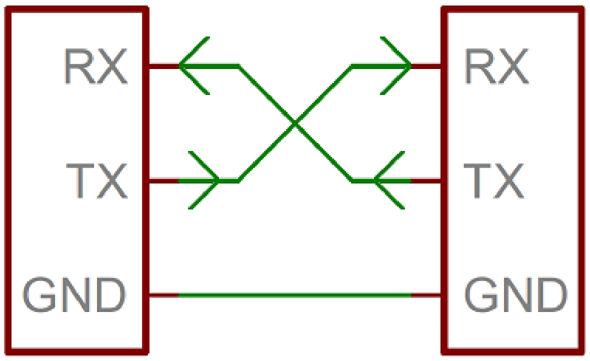


Figure 19:Hardware connections of serial communication

3.4.3 Wireless 2.4-GHz Communication

There are many modules are recognized worldwide as data communication device. Which are license free 2.4Ghz ISM band. ISM band stands for Industrial, Scientific and Medical band. It's a part of radio spectrum. It can be used almost any country without license. But these frequencies and power of these bands varies from country to country. Some common frequencies are:

- 2.4 GHz = nearly worldwide
- 915 MHz band = North America, South America, some other countries
- 868 MHz band = Europe

For any given distance, a 2.4 GHz installation will have roughly 8.5dB of additional path loss when compared to other frequencies. Lower frequencies always required larger antennas to reach the gain. As we know frequency rise as the like as bandwidth rises too. But the range and obstacle problem is reduced.

Advantage of 2.4GHz band:

- It has wider signal coverage area
- Better penetration through walls

Disadvantage of 2.4GHz band:

- Might get interference from any kind of Bluetooth devices, wireless devices etc. which are also operates on same frequency.

There are many different types of RF network with various advantage and disadvantages. The main we are concern in these devices are network topology, data rate, power, consumption, range etc.

There are few devices that we can use:

XBee:

Low data rate (250 Kbit/s) and low power consumption. It's mainly used in mesh networking. We need two XBee to communicate each other and through each other via the mesh to device that are out of distance. This used probably as network device, particularly in sensor related.

Wi-Fi:

High data rate (54 Mbit/s). It also consumes high power. It need to connect directly through internet and pc. It also need external power source.

Bluetooth:

Medium data rate. It consumes medium power. Bluetooth are pairing type device. Pairing type network normally not so much useful for sensor networks. It is only good for laptops and mobile also it can be use in Arduino via serial RX and TX pins to communicate.

Power Limit:

2.4 - 2.4835 GHz	
Bluetooth	100m
Wi-Fi	802.11a/n/ac
Microwaves ovens	900
XBee	802.15.4

Table 1 Power Limit of 2.4GHz radio-band

3.5 Ground Station

We have developed a ground station (G.S) for our project. The Graphical User Interface (GUI) of the GS has been coded using LabVIEW, which is a system-design platform and development environment for a visual programming language from National Instruments. As LabVIEW uses visual language, it is very easy to implement and build the GUI. The interface is grouped into several tabs where data from the sensors are viewed separately. The GS also has a data logging option enabled. Every time the program is run, it will save the data in an Excel spreadsheet for further analysis.

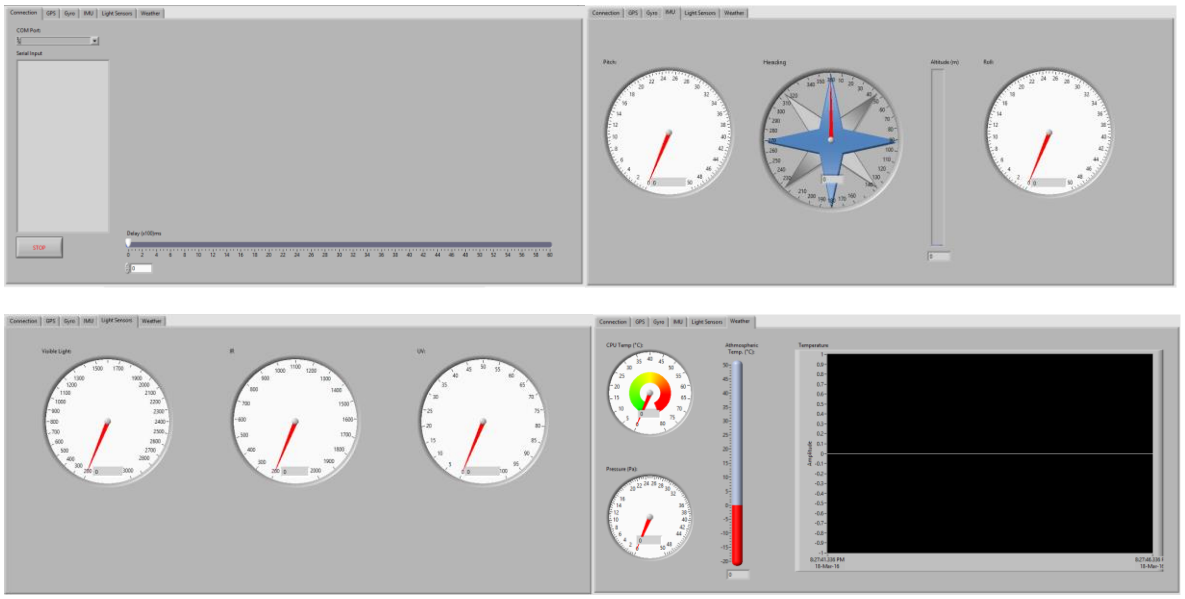


Figure 20: Graphical user interface(GUI)

The tabs are labeled as Connection, GPS, Gyro, IMU Data, Light Sensor and Weather.

The connection tab deals with the serial connection and shows the input in the com port. There is also a drop down list to select the correct com port to begin the connection. It also has a slider to adjust the delay of the receiving data.

GPS tab shows the satellite time and date and also the longitude and latitude of the current position. There is also a button which will call up an executable file (.exe) which can plot the positions in a map.

The Gyro tab shows the X, Y and Z components of the gyroscope in a live graph.

The IMU Data (inertial measurement unit data) tab displays the pitch, roll, heading and altitudes in radial dials and a bar.

The following tab is the Light Sensor tab which displays information from the SI1145 UV sensor. This tab has three dials to visualize Visible Light, Infrared and UV Index.

Finally, the Weather tab shows atmospheric temperature and pressure as well as the CPU temperature of the Raspberry Pi.

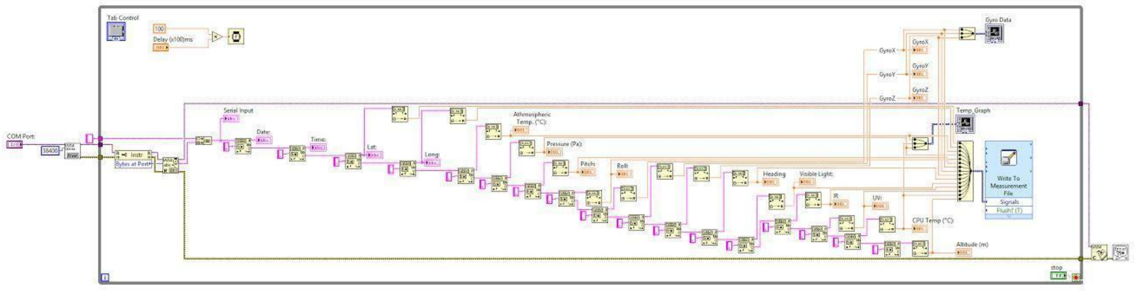


Figure 21: LabVIEW g-code for the ground station GUI

Additionally, we have used Visual Studio to create an executable file which once called will pull data from the data logging file and mark the position of the balloon in a Google Map window. The map can be zoomed in and out using the mouse scroll wheel. However, the ground station needs an active internet connection for the Google Map to load. We can see that in Appendix 8.2 figure 44.

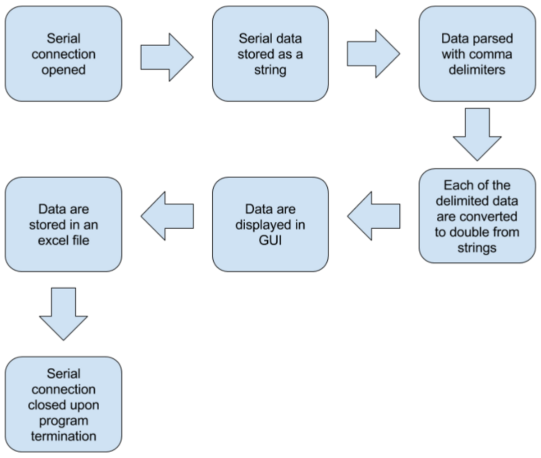
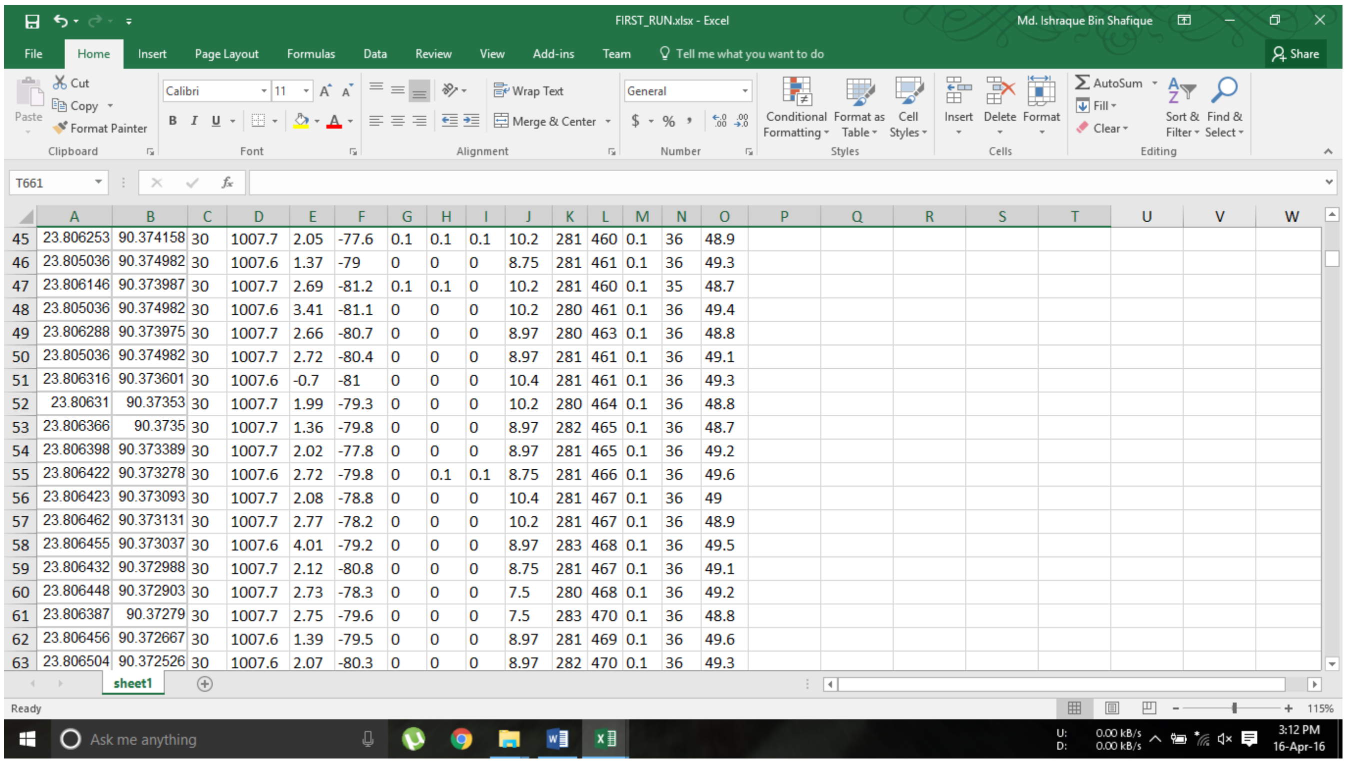

Serial connection opened
Serial data stored as a string
Data parsed with comma delimiters
Each of the delimited data are converted to double from strings
Data are displayed in GUI
Data are stored in an excel file
Serial connection closed upon program termination

Figure 22: Flowchart of program in LabVIEW

The above diagram summarizes the LabVIEW program used for the Ground Station. The serial connection opens with a predefined Baud Rate of 9600. The data that is being pulled from the XBee module connected to the computer is stored in a string where each data is separated using a comma delimiter. Then pattern match functions are used to parse each data before the commas. The parsed data are displayed in the GUI and are being saved in an Excel Spreadsheet for further analysis shown in figure 23. When the program stop button is clicked, the program is halted and the serial connection with the XBee module is closed.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
45	23.806253	90.374158	30	1007.7	2.05	-77.6	0.1	0.1	0.1	10.2	281	460	0.1	36	48.9
46	23.805036	90.374982	30	1007.6	1.37	-79	0	0	0	8.75	281	461	0.1	36	49.3
47	23.806146	90.373987	30	1007.7	2.69	-81.2	0.1	0.1	0	10.2	281	460	0.1	35	48.7
48	23.805036	90.374982	30	1007.6	3.41	-81.1	0	0	0	10.2	280	461	0.1	36	49.4
49	23.806288	90.373975	30	1007.7	2.66	-80.7	0	0	0	8.97	280	463	0.1	36	48.8
50	23.805036	90.374982	30	1007.7	2.72	-80.4	0	0	0	8.97	281	461	0.1	36	49.1
51	23.806316	90.373601	30	1007.6	-0.7	-81	0	0	0	10.4	281	461	0.1	36	49.3
52	23.80631	90.37353	30	1007.7	1.99	-79.3	0	0	0	10.2	280	464	0.1	36	48.8
53	23.806366	90.3735	30	1007.7	1.36	-79.8	0	0	0	8.97	282	465	0.1	36	48.7
54	23.806398	90.373389	30	1007.7	2.02	-77.8	0	0	0	8.97	281	465	0.1	36	49.2
55	23.806422	90.373278	30	1007.6	2.72	-79.8	0	0.1	0.1	8.75	281	466	0.1	36	49.6
56	23.806423	90.373093	30	1007.7	2.08	-78.8	0	0	0	10.4	281	467	0.1	36	49
57	23.806462	90.373131	30	1007.7	2.77	-78.2	0	0	0	10.2	281	467	0.1	36	48.9
58	23.806455	90.373037	30	1007.6	4.01	-79.2	0	0	0	8.97	283	468	0.1	36	49.5
59	23.806432	90.372988	30	1007.7	2.12	-80.8	0	0	0	8.75	281	467	0.1	36	49.1
60	23.806448	90.372903	30	1007.7	2.73	-78.3	0	0	0	7.5	280	468	0.1	36	49.2
61	23.806387	90.37279	30	1007.7	2.75	-79.6	0	0	0	7.5	283	470	0.1	36	48.8
62	23.806456	90.372667	30	1007.6	1.39	-79.5	0	0	0	8.97	281	469	0.1	36	49.6
63	23.806504	90.372526	30	1007.6	2.07	-80.3	0	0	0	8.97	282	470	0.1	36	49.3

Figure 23: Data logging in Excel

3.6 Processes Inside Raspberry Pi

The Raspberry Pi acts as the main brain of the BalloonSat. We have developed the necessary codes required for working using Python libraries. The code has been broken down to several sub-functions. This allows easy access to the specific works and each segments can be edited whenever needed. The sub-functions are gps_xl, IMUorientation, IMUtemp, UVsensor, getCPUtemp. The main program has been named MAIN_run. Whenever MAIN_run program is started it calls in the other functions and access data from the various sensors. The flowchart of the programs is given below:

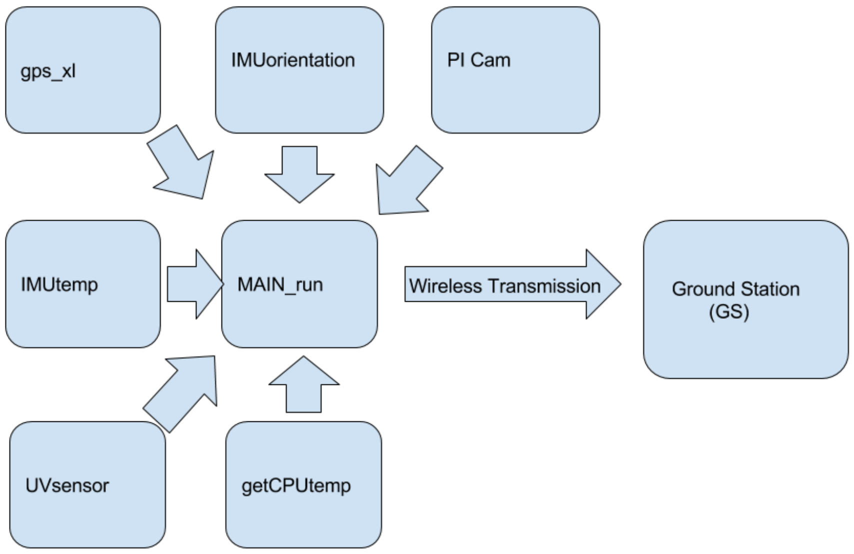


Figure 24: Sub-functions that are called by the main function(MAIN_run)

The RGB image captured by the Raspberry Pi was then devised to be sent wirelessly using the XBee. The particular version of XBee module have low data transfer rate. As a result, it takes a very long time to send a single image. Just to make a comparison we have sent images of various sizes and the time taken are recorded in the following table.

Image Resolution (W*H Pixels)	Time required to wirelessly transfer data
1280 x 960	3313s
1024 x 768	2144s
640 x 480	834s
300 x 300	247s

Table 2 Time required to process and send images of different resolutions.

From the table we can clearly see, even the smallest resolution takes 4 minutes 7 seconds to be transmitted. Therefore, we have suppressed the image transmission from our normal operation.

CHAPTER 4
IMAGE PROCESSING
&
VEGETATION ANALYSIS

4.1 Image Processing

4.1.1 Basics of Image Processing

Image processing is the processing of images using computational operations by using signal processing techniques. The input is an image, a series of images or a video; the product is maybe an image with a set of characteristics or parameters related to that image.[5] In recent days, majority of the image processing algorithms involve treating the image as a 2D signal and applying standard digital signal processing techniques to it. Processing can also be done in 3D with time being the third dimension or z-axis.

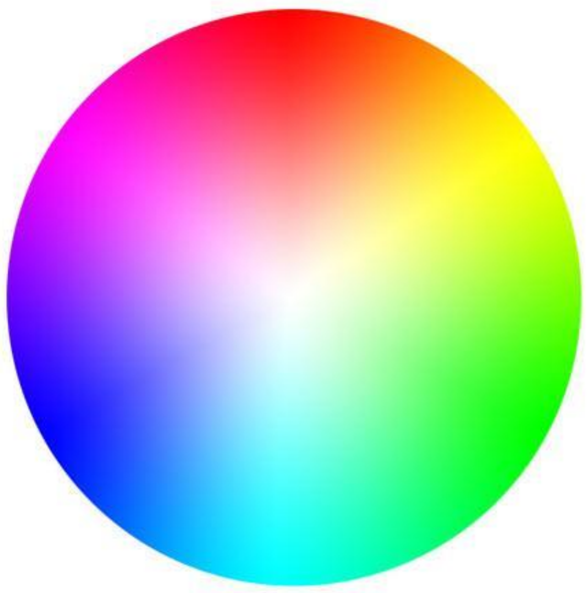


Figure 25: RGB color wheel

Digital image processing has many advantages over its analog counterpart. It allows a much broader spectrum of computational techniques to be applied to the data and can avoid problems such as noise during the process. In addition to that, the image being in digital form is very easy to store in storage devices and can also be transmitted easily to large distances. Before explaining the actual image processing that we have worked out in this project, let us at first discuss few of the basics of a digital image.

4.1.2 Digital Image

According to James D. Foley, digital image is a numerical representation of a two dimensional image.[6] Depending on the resolution of the image, the image may be of raster or vector type; however usually a digital image refers to raster or bitmapped image. Any digital image comprises of its most elementary units called pixels (picture elements). Each pixel is made up of three channels named the RGB (Red Green Blue) channels. And a grayscale image has only one channel. The original image of figure 26 below was taken during the first test run of our BalloonSat.

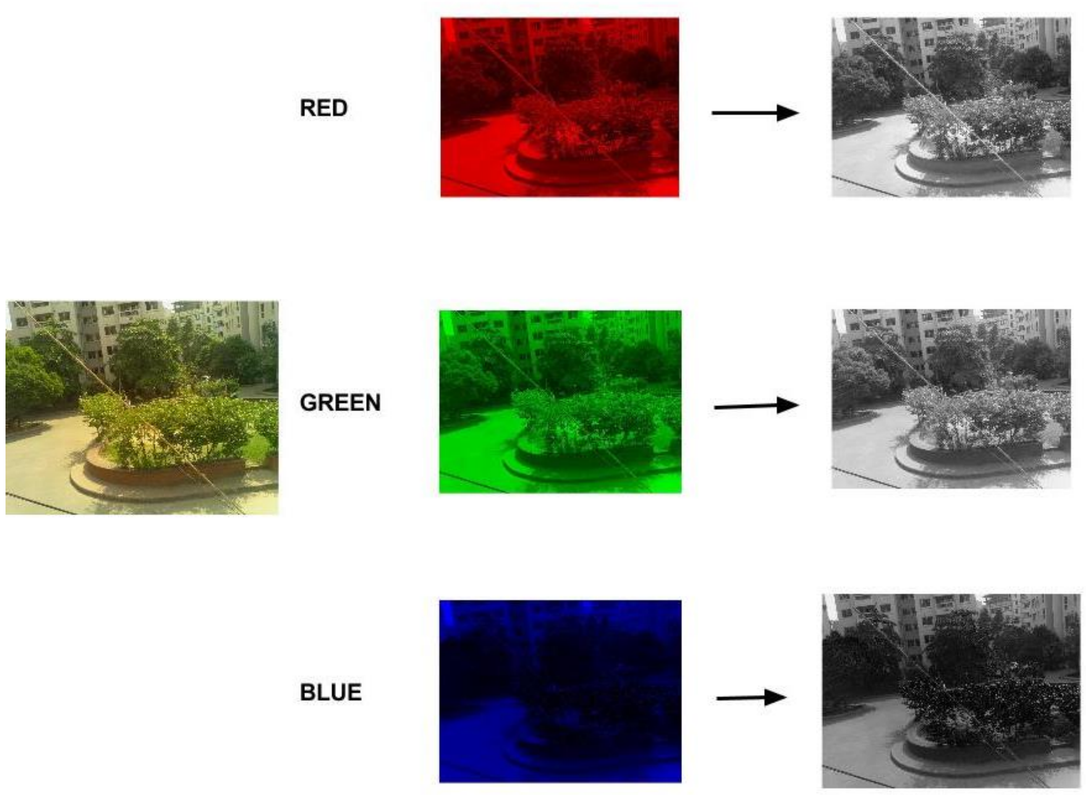


Figure 26: Image of a park segmented into its three color channels

4.1.3 Image Resolution

The resolution of an image is the details that an image holds. The term applies to the raster type of digital images. It is identified by the width and height of the image as well as the total number of pixels in the image. The total number of pixels of an image is calculated by multiplying the height and the width. Then the calculated result is divided by 1 million; this will give the resolution in megapixels (MP). The greater the value of the MP, the bigger is the image size.

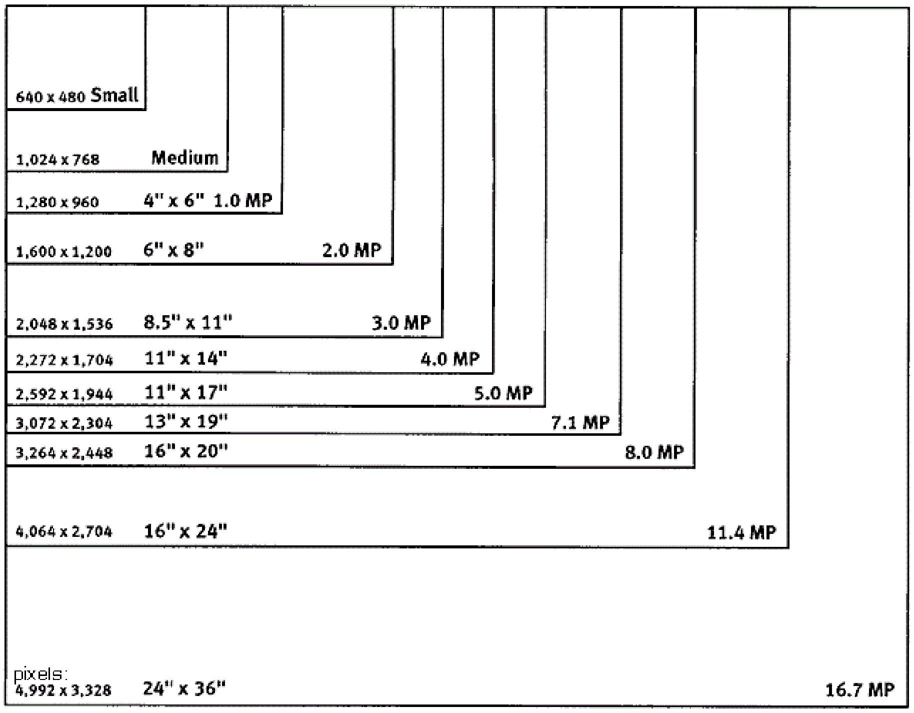

640 x 480 Small
1,024 x 768 Medium
1,280 x 960 4" x 6" 1.0 MP
1,600 x 1,200 6" x 8" 2.0 MP
2,048 x 1,536 8.5" x 11" 3.0 MP
2,272 x 1,704 11" x 14" 4.0 MP
2,592 x 1,944 11" x 17" 5.0 MP
3,072 x 2,304 13" x 19" 7.1 MP
3,264 x 2,448 16" x 20" 8.0 MP
4,064 x 2,704 16" x 24" 11.4 MP
pixels: 4,992 x 3,328 24" x 36" 16.7 MP

Figure 27: Comparison of images of different resolutions courtesy of Canon (2005)

4.1.4 Image Thresholding

Image thresholding is the simplest method of image segmentation. From a RGB image, thresholding can be used to create binary images.[7] The simplest thresholding methods replace each pixel in an image with a black pixel (0) if the image intensity $I_{x,z}$ is less than some fixed constant T (that is, $I_{x,z} < T$), or a white pixel (1) if the image intensity is greater than the constant. Color images can also be thresholded. The approach is to designate a separate threshold for each of the RGB components of the image and combine them with an AND operation.

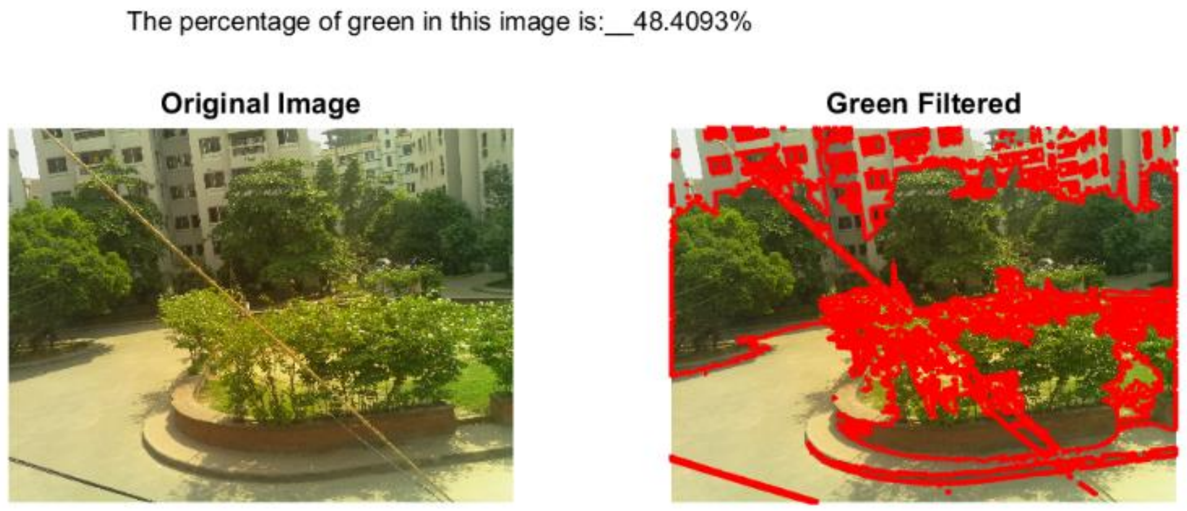


Figure 28: Color thresholding of an image of a park

The above figure shows the original image as input and the green filtered image after color thresholding. The pixels of the original image which have values within the defined range of RGB values are represented as white (1) and the rest as black (0). Later we used boundary detection to identify patches of greenery which was inspired by the work of Maloof et al. [8]

This algorithm can be used to easily estimate the percentage of a particular color range in a given image. The use of this technique is explained in the following section.

The onboard Pi-Cam can capture images of 5MP (2592×1944 pixels) resolution. Due to limitation in onboard-processing power and as well as storage capacity, we had to acquire the images at 600 x 400pixels resolution. We processed the images in MATLAB later on.

We let the Raspberry Pi to take a series of photographs using its camera. The images are saved in a directory with sequential naming (Image1.jpeg, Image2.jpeg and so on) for easier handling. Later we took a batch of 30 such images from the Raspberry Pi using USB flash drive and saved it in a MATLAB directory. The MATLAB script loads the files one by one, apply color threshold technique, calculate the green of green and also saves a resulting output in the directory sequentially. We have experimentally found that MATLAB is very fast in processing the images which is a positive side. On the contrary, we have to wait for a long time before we can get a batch of images.

The color thresholding limits of each channels are as:

0≤	RED_channel	≤106.0
0≤	GREEN_channel	≤255.0
0≤	BLUE_channel	≤114.0

Image Resolution (W*H Pixels)	**Number of image files**	**Total Time required (Seconds)**
1280 x 960	30	33.88
1024 x 768	30	32.86
640 x 480	30	29.60
300 x 300	30	28.02

Table 3 Time required for processing images captured at different resolutions.

4.2 Vegetation Health Analysis by Color Thresholding

Image processing is very useful for agriculture, we can take an image of a vegetation field and match the leaf color with the leaf color charts to understand the Nitrogen fertilizer dosage of the field. C. Witt (2005) says that, the leaf color chart (LCC) helps the farmers to evaluate plant nitrogen (N) demand. [9] The LCC was made by calculating the leaf spectral reflectance measurement, this reflectance varies with the color of the leaf. The color chart has four different green shades, so we took each shade and found its corresponding RGB composition in MATLAB.

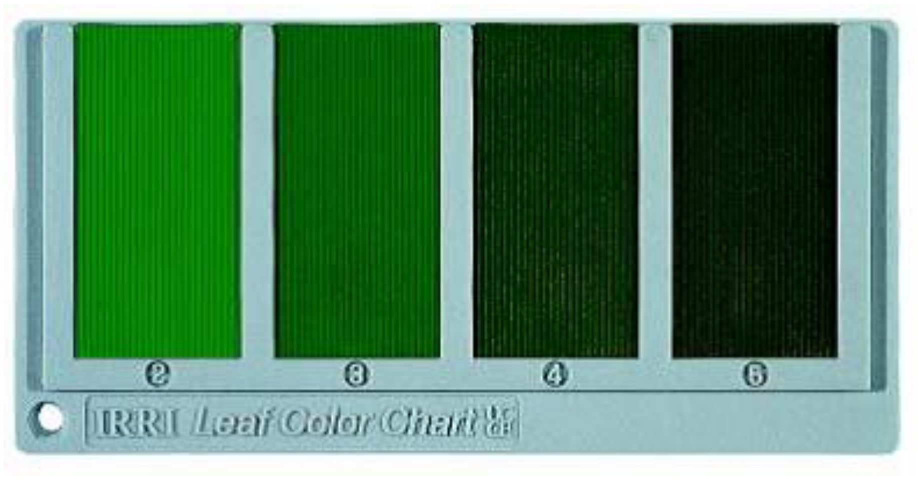


Figure 29: LEAF Color Chart(LLC) courtesy of Irrigated Rice Research Consortium

Strip Number	Red	Green	Blue
2	103-255	145-255	0-12
3	83-98	145-165	0-30
4	0-106	111-113	0-30
5	0-104	0-105	0-62

Table 4 RGB Spectrum of IRRI Leaf Color Chart (LCC)

From an image of a crop field, we can now understand the Nitrogen fertilizer level by finding that images RGB composition and comparing it with these ranges in MATLAB, if the color range falls in the 1st strip then the Nitrogen level is low, if it falls in the 4th strip then there is a surplus of Nitrogen, if it falls in the 2nd strip then the Nitrogen level is below the critical level of Nitrogen composition, if in the 3rd strip then the Nitrogen composition is above the critical Nitrogen composition and so therefore for a healthy and good yield the color of the leaf must be between the 2nd strip and the 3rd strip.

CHAPTER 5
ANALYSIS & RESULT

5.1 Random Vibration Analysis

The finite element analysis was a stationary random vibration test on a student version of the analysis software Ansys. For simplification of the process, the fillets and chamfers of the model were ignored. Adding to that, the protective plates on each face of the model were suppressed to visualize the analysis better. Lastly, threaded connections were simplified as faces glued together.

We took three space grade materials that are used commonly in space missions and compared the results to determine the best selection of material. Some physical properties are shown on the next table.

Material	**Yield Strength(Pa)**	**Shear Modulus(Pa)**	**Poisson's Ratio**	**Density (g/cm^3)**
Aluminum 6061-T6	276×10^6	26×10^9	0.33	2.7
Aluminum 7075-T6	503×10^6	26.9×10^9	0.33	2.81
Ti-6Al-4V	970×10^6	44×10^9	0.342	4.43

Table 5: Intrinsic Properties of Aerospace Material

Mode analysis is done as a prerequisite of random vibration test which gives us the natural frequencies of the structure. To perform this, we had to add fixtures to simulate how the Nano satellite will be placed in test environment. A fixed geometry was added to the bottom face of the model simulating how structures are placed on the shaker table in real life. The modal frequencies are listed on the table below. Top four of the modes are shown pictorially in figure 30 to figure 33.

Mode	Frequency [Hz]
1	843.4
2	1854.7
3	1864.5
4	2808.4
5	3903.6
6	3924.6

Table 6: Natural Frequency at Different Modes

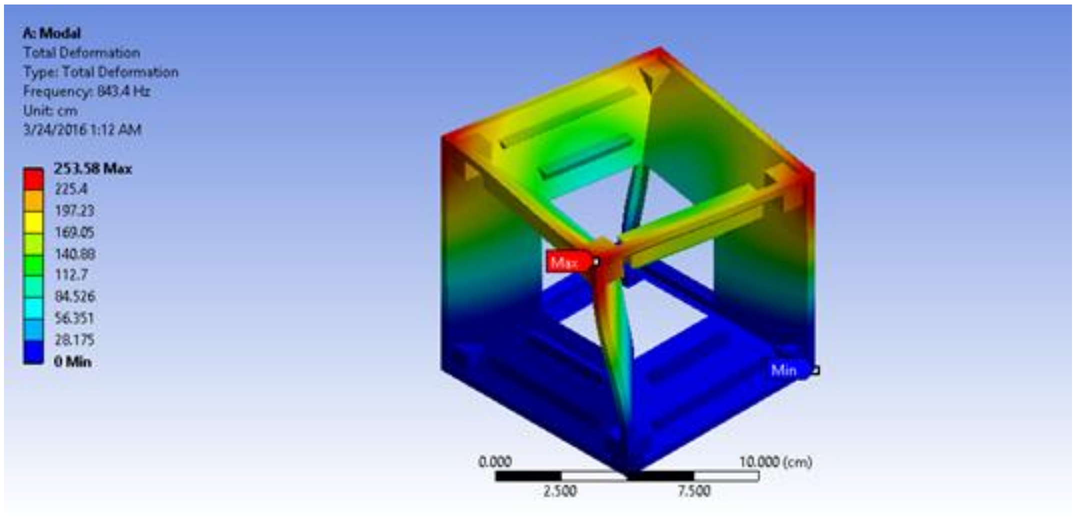


Figure 30: Mode 1

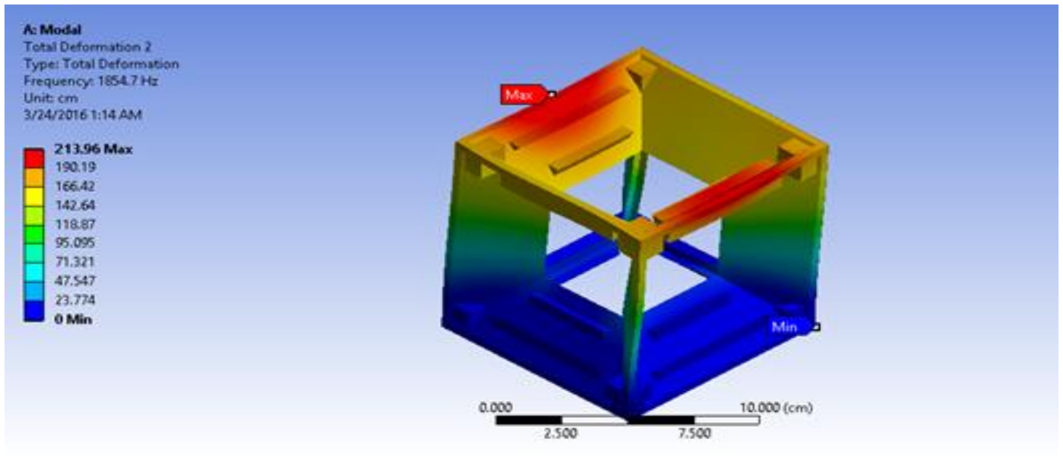


Figure 31: Mode 2

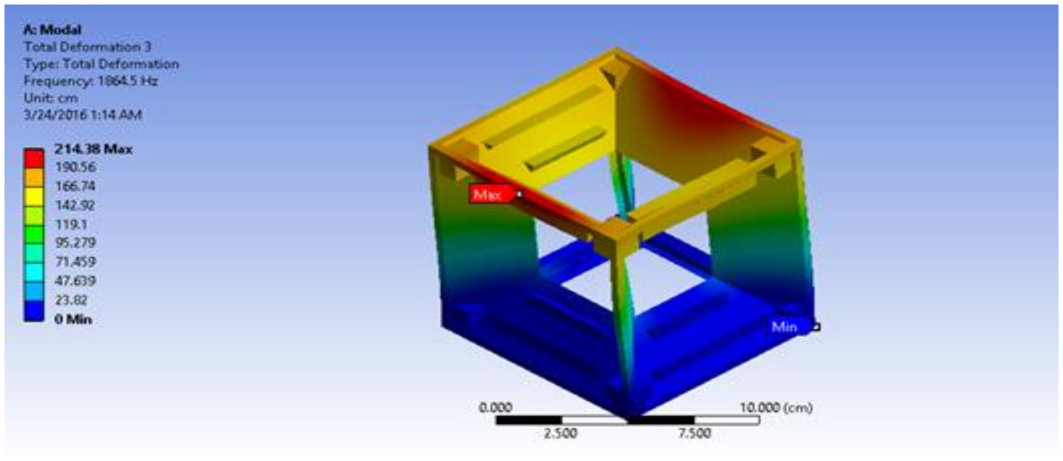


Figure 32: Mode 3

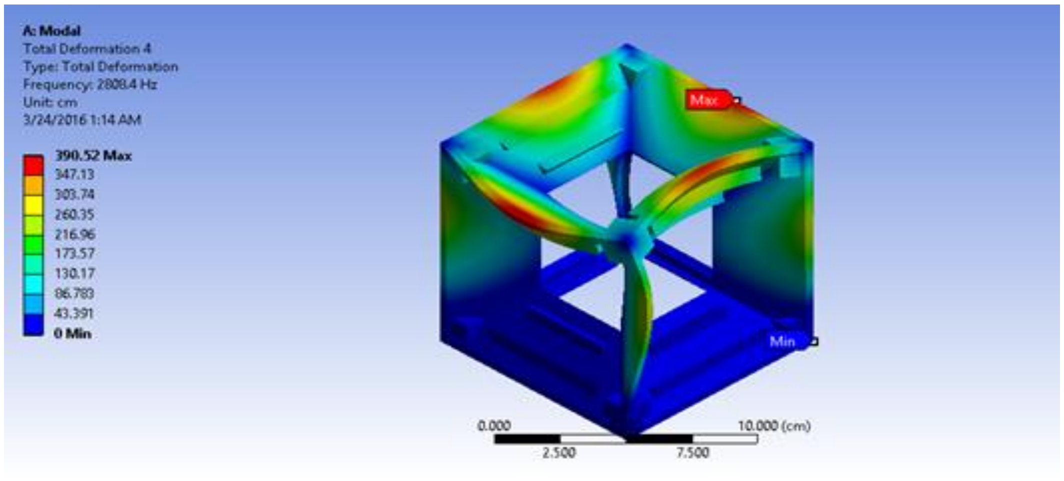


Figure 33: Mode 4

Nano satellites are deployed to orbit by means of transfer vehicle that are launched from the earth station. The satellites positioned in this carrier vehicle faces tremendous random vibration excitation ranging from 20 to 2000 Hz. So as per the NASA guideline random vibration tests have to be conducted with real vibration profile provided by the CubeSat deployer companies. In our case, we have used the test profile provided by NanoRacks LLC company, in their NanoRacks CubeSat deployer interface control document. [10]

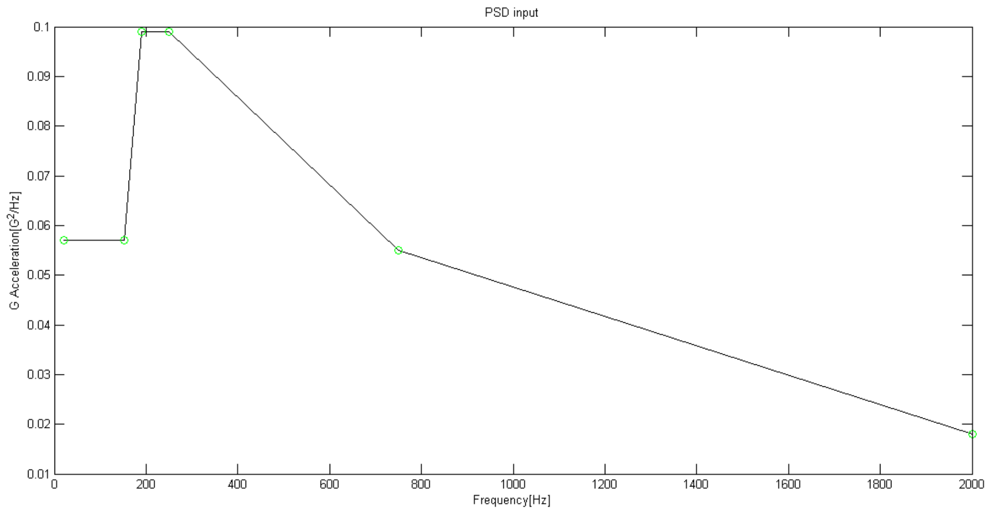


Figure 34: Input PSD spectrum

To acquire the solution, we put the test profile in X, Y, Z direction as PSD acceleration and selected equivalent von-mises stress and response power spectral density as the expected output. Setting the system damping ratio to 3% and meshing the structure using program controlled settings we solved the analysis and figured out the interpretations of the results. We chose the vertex that showed maximum stress on the equivalent stress plot as the node where the response PSD is probed. We can take a look at the 1σ equivalent stress plots of the three materials in figure 35 to figure 37. The response PSD on the three axes are shown on figure 38 to figure 40.

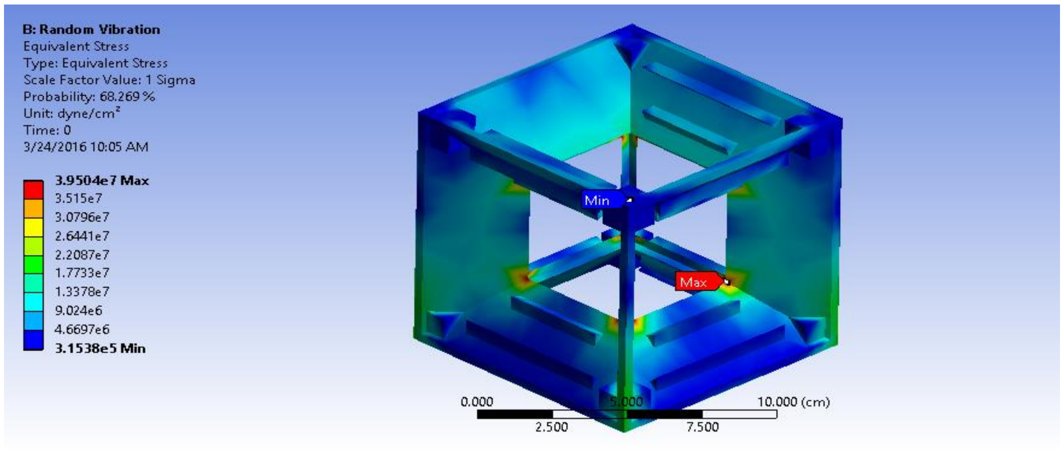


Figure 35: Aluminum 6061-T6 1σ equivalent stress plot

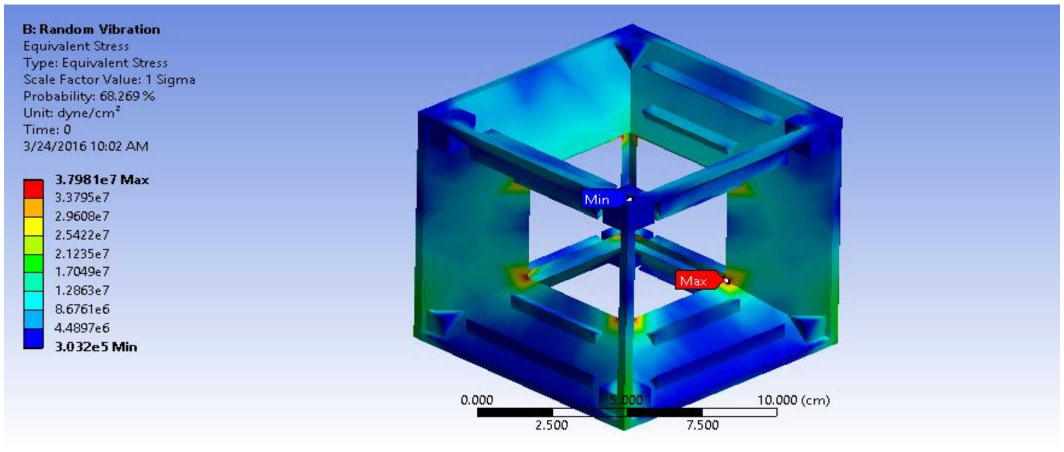


Figure 36: Aluminum 7075-T6 1σ equivalent stress plot

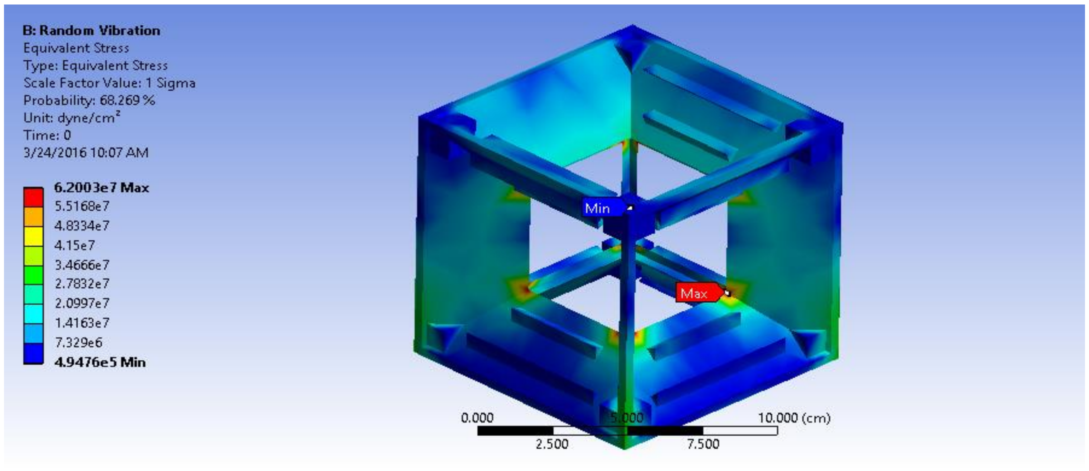


Figure 37: Ti-6Al-4V 1σ equivalent stress plot

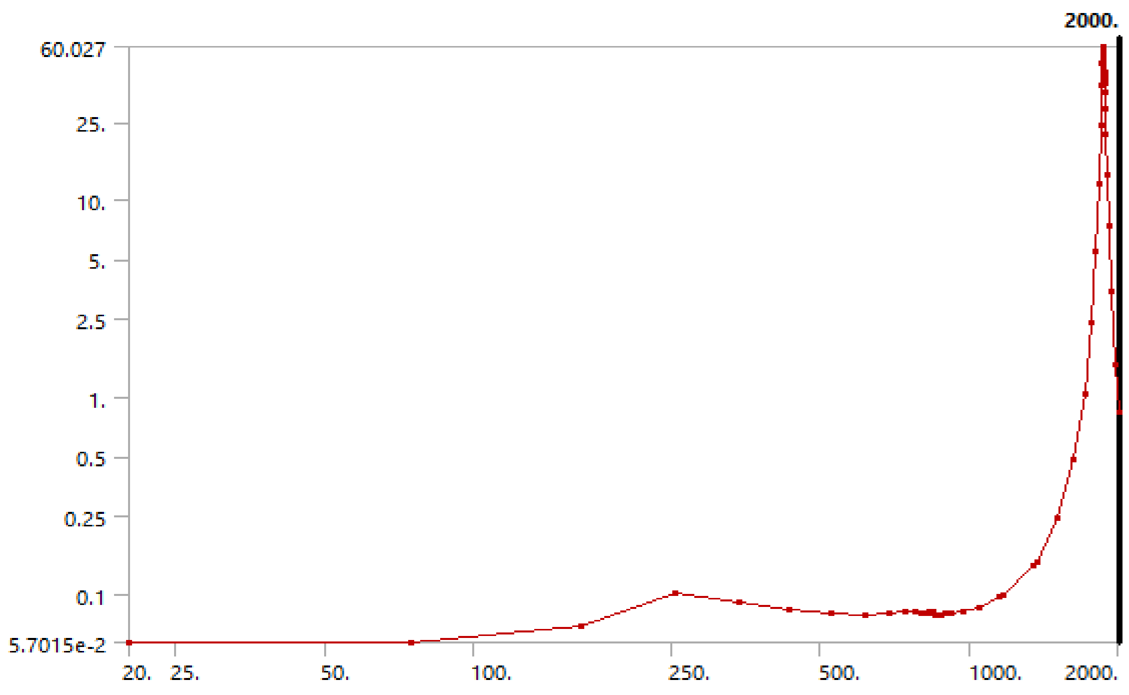


Figure 38: Response PSD on X axis

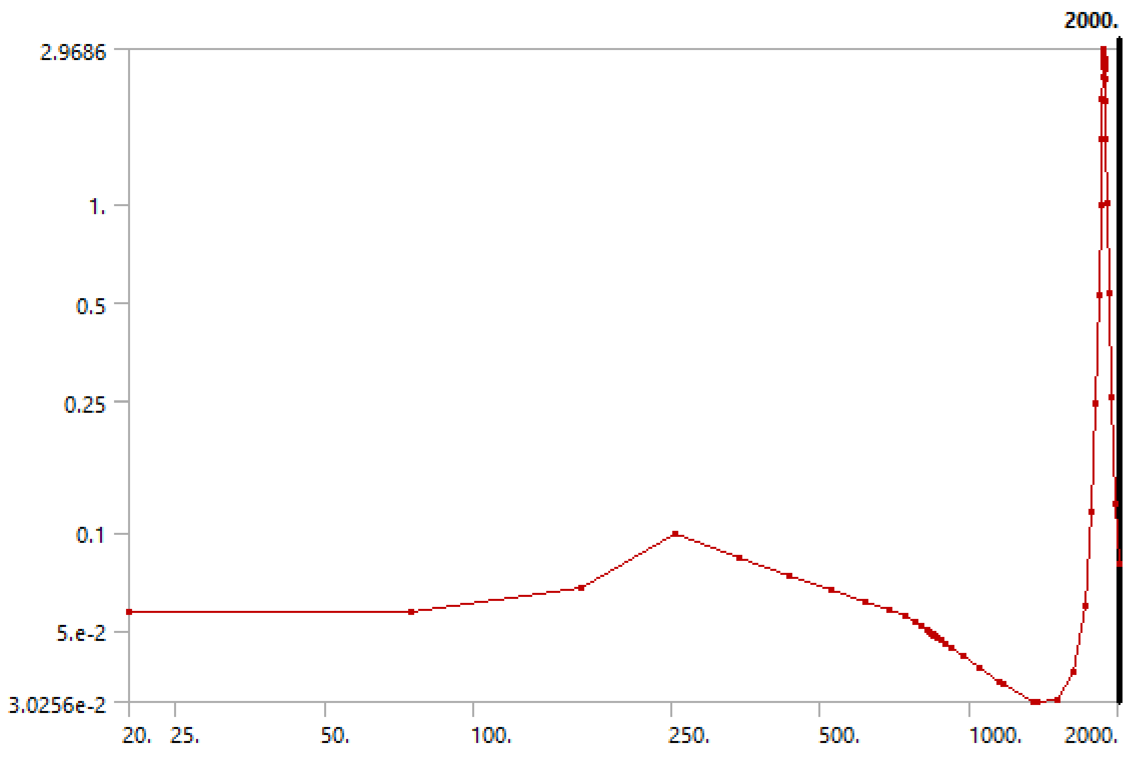


Figure 39: Response PSD on Y axis

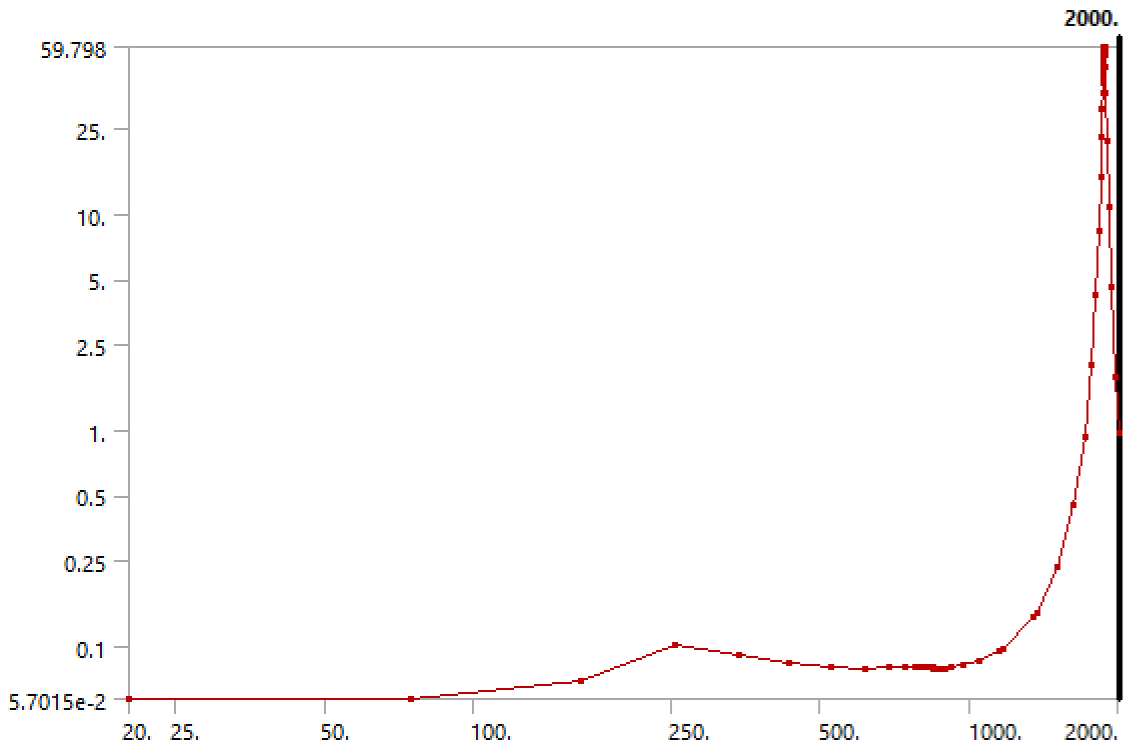


Figure 40: Response PSD on Z axis

As shown on the figure 35 to figure 37 maximum stress occurs in the inner corner points of the pyramid structures. So it can be said that vibration fatigue damage will harm this area the most. We can take a detailed look on the stress and displacement values for the three materials in the table 7.

Material	**Maximum Equivalent Stress (1σ) [Pa]**	**Maximum Equivalent Stress (3σ) [Pa]**	**Maximum Displacement (Y Axis) [cm]**
Al 6061-T6	39.504×10^6	113.94×10^6	0.00011747
Al 7075-T6	37.98×10^6	118.51×10^6	0.0001181
Ti-6Al-4V	62.003×10^6	186.01×10^6	0.00011319

Table 7 Results from Random Vibration Analysis

From table 7 it is visible that maximum displacement (on the vertical direction) occurs on the material Al 7075-T6 that is 0.118mm. If we look at stress values similar results are seen; The 3σ values are almost the same for the two aluminum alloys. The maximum surface stress for material-2 is 37.98MPa which is located at the joint of the inner corner. The probability of surface stress under 37.98MPa is 68.2% wherein probability of stress under 118.51MPa is 99.73%. So the material that shows the best promise from these materials is Al 7075-T6. With a yield strength of 503 MPa and considering a safety factor of 2, maximum allowed stress is 118.51*2=237.02MPa that is less than 503 MPa. [11] So the Nano satellite structure withstands the input excitation of random vibration with the selected material.

As an extension to the analysis performed, the maximum stress value can be used in Steinberg's three interval method and Miner cumulative damage theory to estimate fatigue life of the Nano satellite [12]. We encourage performing this analysis additionally if any group decides to further this research and hopes to make a working prototype of the model.

The acceleration response is more important in random vibration tests; in which our structure showed a tendency to get spikes in the three Axes around the frequency 1854-1864 Hz. This proves that the 2nd and 3rd order modes are contributing to the response PSD. In addition to that, the y axis response on the node shows another high acceleration value around the frequency 253.4 Hz.[13]

CHAPTER 6
DISCUSSIONS
&
CONCLUSION

6.1 Discussions

6.1.1 Structural Analysis

Finite element analysis is an important part of this thesis. We have successfully completed the analysis of random vibration of the 3D model. Note that, the 3D model used in the analysis consisted solely of the outer body; protective plates, hinges and screws were suppressed to reduce complexity. By analyzing three different materials we also determined which one would be the best choice for the structure. Additionally, the results of our random vibration analysis has passed the specification set by NanoRacks LLC, a carrier vehicle production facility. However, we could not perform all the test such as shock test and outgassing test that was necessary to comply with international standard. This was mainly due to lack of resources and logistical support.

6.1.2 BalloonSat

We were unable to launch the BalloonSat due to the shortage in our budget; the cost of the balloon and the Helium gas was too high. Additionally, the Xbee module used was of hundred feet range, higher range radio modules are not available in Bangladesh. The 2.4-GHz radio band is very prone to noise, as a result the data received had errors and many delays. The gps communicates at 9600 baud rate which is consistent but slow. We had the facility to increase the baud rate of the Xbee to increase data transmission. This however was not feasible because one serial port cannot support two different baud rates. The image data takes too much time to transmit, so we were unable to send the image data along with the other data.

6.2 Conclusion

Satellite are expensive to build and complicated to maintain, however with the introduction of Nano-satellites it is possible to achieve space research by developing countries like Bangladesh. From our thesis works, we found out that BalloonSat can be used transmit data wirelessly from the balloon to the Ground Station(GS). Most of the earlier researches conducted on BalloonSat relied on the data being saved in a logger; which is then collected manually. We constructed the BalloonSat from off-the shelf sensor modules, as a result anyone can follow our paper and construct the circuit on their own.

The sensory data collected at GS can be used by other researchers to analyze local weather fluctuation patterns. In addition, each BalloonSat unit also easy to setup and launch. Many of such units can be used in different places to gather data on a large area. These data can be analyzed using deep learning to better understand climate changes such as global warming. [14] In addition satellite imaging can be used to detect oil spills in rivers and sea beds. [15]

The imaging algorithm used in the BalloonSat can be used to determine the nitrogen concentration of leaves. Farmers can easily inspect the health of crop in a vast area without having to manually check small sample volumes. Furthermore, the same algorithm can be implemented to measure the rate of deforestation.

CHAPTER 7
REFERENCES

[1] Rhaman, M. K., M. I. Monowar, S. R. Shakil, A. H. Kafi, and R. S. I., Antara. "Exploring Modular Architecture for Nano Satellite and Opportunity for Developing Countries." IOP Conf. Series: Earth and Environmental Science 23 (2015). N.p., n.d. Web. 12 May 2015. <(http://iopscience.iop.org/1755-1315/23/1/012017)>.

doi:10.1088/1755-1315/23/1/012017

[2] Koehler, Chris. "BalloonSat: missions to the edge of space." (2002).

[3] California Polytechnic State University. CubeSat Design Specification Rev. 13. N.p.: The CubeSat Program, Cal Poly SLO, 20 ` Feb. 2014. PDF

[4] ANDREWS, FRANCIS J. "RANDOM VIBRATION—AN OVERVIEW." (n.d.): n. pag. Web. 14 Apr. 2016.

[5] Rafael C. Gonzalez; Richard E. Woods (2008). Digital Image Processing. Prentice Hall. pp. 1–3. ISBN 978-0-13-168728-8.

[6] James D. Foley (1995). Computer Graphics: Principles and Practice. Addison-Wesley Professional. p. 13. ISBN 0-201-84840-6.

[7] Shapiro, Linda G. & Stockman, George C. (2002). "Computer Vision". Prentice Hall. ISBN 0-13-030796-3

[8] Maloof, M., Langley, P., Binford, T., Nevatia, R., and Sage, S. (to appear). Improved rooftop detection in aerial images with machine learning. Machine Learning. http://www.kluweronline.com/issn/0885-6125.

[9] C. Witt, J.M.C.A. Pasuquin, R. Mutters, and R.J. Buresh. (2005) "New Leaf Color Chart for Effective Nitrogen Management in Rice"- Better Crops/Vol. 89 (2005, No. 1)

[10] NanoRacks, LLC. "NanoRacks CubeSat Deployer (NRCSD) Interface Control Document." 0.36 (2013): n. pag. 10 Dec. 2013. Web. 12 Nov. 2015.

[11] Junqiu Li, Helei Tian and Puen Wu, "Analysis of random vibration of power battery box in electric vehicles," Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), 2014 IEEE Conference and Expo, Beijing, 2014, pp. 1-5.

doi: 10.1109/ITEC-AP.2014.6940829

[12] S. X. Qu, D. Xu and R. Kang, "Analysis of random vibration life of mechanical parts of actuating cylinder based on the finite element," Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE), 2013 International Conference on, Chengdu, 2013, pp. 917-920.

doi: 10.1109/QR2MSE.2013.6625717

[13] Xiaoguang Lu: The analysis of dynamic performance and optimization of the structure of the aerospace computer case,2010

[14] Levner, Ilya, et al. "Learning Robust Object Recognition Strategies." The 8th Australian and New Zealand Conference on Intelligent Information Systems. 2003.

[15] Kubat, Miroslav, Robert C. Holte, and Stan Matwin. "Machine learning for the detection of oil spills in satellite radar images." Machine learning 30.2-3 (1998): 195-215.

[16] J. Zhang and Limin Li, "A PSD analysis of an online thickness measuring system," Computer-Aided Industrial Design and Conceptual Design, 2008. CAID/CD 2008. 9th International Conference on, Kunming, 2008, pp. 388-394.

doi: 10.1109/CAIDCD.2008.4730595

[17] Berni, J. A. J., Zarco-Tejada, P. J., Suarez, L., and E. Fereres, 2009, “Thermal and Narrowband Multispectral Remote Sensing for Vegetation Monitoring from an Unmanned Aerial Vehicle,” IEEE Transactions on Geoscience and Remote Sensing, 47(3):722–738.

[18] Sur, Ritobrata, Shengkai Wang, and Wending Lu. "Counting Trees in Deforested Areas from Aerial P Counting Trees in Deforested Areas from Aerial Photographs hotographs."

CHAPTER 8
APPENDIX

8.1 Codes

8.1.1 imu_orientation

```
#!/usr/bin/python

def IMUorientation():

        try:

                import time

                import math

                import smbus

                import datetime


                                #IMU register addresses

                GYR_ADDRESS= 0x6B

                CTRL_REG1= 0x20

                CTRL_REG4= 0x23

                OUT_X_L=0x28

                OUT_X_H=0x29

                OUT_Y_L=0x2A

                OUT_Y_H=0x2B

                OUT_Z_L=0x2C

                OUT_Z_H=0x2D


                MAG_ADDRESS   = (0x3C>> 1)

                ACC_ADDRESS   = (0x32>>1)


                CTRL_REG1_A             =               0x20

                CTRL_REG2_A             =       0x21

                CTRL_REG3_A             =       0x22
```

CTRL_REG4_A	=	0x23
CTRL_REG5_A	=	0x24
CTRL_REG6_A	=	0x25
HP_FILTER_RESET_A	=	0x25
REFERENCE_A	=	0x26
STATUS_REG_A	=	0x27
OUT_X_L_A	=	0x28
OUT_X_H_A	=	0x29
OUT_Y_L_A	=	0x2A
OUT_Y_H_A	=	0x2B
OUT_Z_L_A	=	0x2C
OUT_Z_H_A	=	0x2D
FIFO_CTRL_REG_A	=	0x2E
FIFO_SRC_REG_A	=	0x2F
INT1_CFG_A	=	0x30
INT1_SRC_A	=	0x31
INT1_THS_A	=	0x32
INT1_DURATION_A	=	0x33
INT2_CFG_A	=	0x34
INT2_SRC_A	=	0x35
INT2_THS_A	=	0x36
INT2_DURATION_A	=	0x37
CLICK_CFG_A	=	0x38
CLICK_SRC_A	=	0x39

```
CLICK_THS_A         =              0x3A
TIME_LIMIT_A        =              0x3B
TIME_LATENCY_A      =       0x3C
TIME_WINDOW_A       =       0x3D
CRA_REG_M           =       0x00
CRB_REG_M           =      0x01
MR_REG_M            =      0x02

OUT_X_H_M           =       0x03
OUT_X_L_M           =       0x04

SR_REG_M            =       0x09
IRA_REG_M           =       0x0A
IRB_REG_M           =       0x0B
IRC_REG_M           =       0x0C

WHO_AM_I_M          =      0x0F

TEMP_OUT_H_M        =              0x31
TEMP_OUT_L_M        =              0x32
OUT_Y_H_M           =              0x07
OUT_Y_L_M           =              0x08
OUT_Z_H_M           =       0x05
OUT_Z_L_M                  =       0x06

bus = smbus.SMBus(1)
```

```
RAD_TO_DEG = 57.29578
M_PI = 3.14159265358979323846
G_GAIN = 0.070  # [deg/s/LSB]  If you change the dps for gyro, you need to
update this value accordingly
LP = 0.052          # Loop period = 41ms.    This needs to match the time it
takes each loop to run
AA =  0.80      # Complementary filter constant



def writeACC(register,value):
    bus.write_byte_data(ACC_ADDRESS , register, value)
    return -1
def writeMAG(register,value):
    bus.write_byte_data(MAG_ADDRESS, register, value)
    return -1


def writeGRY(register,value):
    bus.write_byte_data(GYR_ADDRESS, register, value)
    return -1


def readACCx():
    acc_l = bus.read_byte_data(ACC_ADDRESS, OUT_X_L_A)
    acc_h = bus.read_byte_data(ACC_ADDRESS, OUT_X_H_A)
    acc_combined = (acc_l | acc_h<<8 )


    if acc_combined > 32768:
        acc_combined -= 65536
    return acc_combined >> 4
```

```
def readACCz():
        acc_l = bus.read_byte_data(ACC_ADDRESS, OUT_Y_L_A)
        acc_h = bus.read_byte_data(ACC_ADDRESS, OUT_Y_H_A)
        acc_combined = (acc_l |acc_h<<8 )

        if acc_combined > 32768:
                 acc_combined -= 65536
        return acc_combined >> 4



def readACCy():
        acc_l = bus.read_byte_data(ACC_ADDRESS, OUT_Z_L_A)
        acc_h = bus.read_byte_data(ACC_ADDRESS, OUT_Z_H_A)
        acc_combined = (acc_l | acc_h<<8 )

        if acc_combined > 32768:
                acc_combined -= 65536
        return acc_combined >> 4
def readMAGx():
        mag_l = bus.read_byte_data(MAG_ADDRESS, OUT_X_L_M)
        mag_h = bus.read_byte_data(MAG_ADDRESS, OUT_X_H_M)
        mag_combined = (mag_l | mag_h<<8 )

        return mag_combined  if mag_combined < 32768 else mag_combined -
65536
```

```
        def readMAGy():
                mag_l = bus.read_byte_data(MAG_ADDRESS, OUT_Y_L_M)
                mag_h = bus.read_byte_data(MAG_ADDRESS, OUT_Y_H_M)
                mag_combined = (mag_l | mag_h<<8 )

                return mag_combined if mag_combined < 32768 else mag_combined -
65536

        def readMAGz():
                mag_l = bus.read_byte_data(MAG_ADDRESS, OUT_Z_L_M)
                mag_h = bus.read_byte_data(MAG_ADDRESS, OUT_Z_H_M)
                mag_combined = (mag_l | mag_h<<8 )

                return mag_combined  if mag_combined < 32768 else mag_combined -
65536



        def readGYRx():
                gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_X_L)
                gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_X_H)
                gyr_combined = (gyr_l | gyr_h <<8)

                return gyr_combined   if gyr_combined < 32768 else gyr_combined -
65536
```

```
def readGYRy():
    gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_Y_L)
    gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_Y_H)
    gyr_combined = (gyr_l | gyr_h <<8)

    return gyr_combined  if gyr_combined < 32768 else gyr_combined -
65536

def readGYRz():
    gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_Z_L)
    gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_Z_H)
    gyr_combined = (gyr_l | gyr_h <<8)

    return gyr_combined  if gyr_combined < 32768 else gyr_combined -
65536



#initialise the accelerometer
writeACC(CTRL_REG1_A, 0b00100111) #z,y,x axis enabled, continuos
update,  10Hz data rate
writeACC(CTRL_REG4_A, 0b00111000)

#initialise the magnetometer
writeMAG(CRA_REG_M, 0b10011000) #Temp enable, M data rate = 50Hz
writeMAG(CRB_REG_M, 0b11100000) #+/-12gauss
writeMAG(MR_REG_M, 0b00000000) #Continuous-conversion mode
```

```
#initialise the gyroscope
writeGRY(CTRL_REG1, 0b00001111) #Normal power mode, all axes enabled
writeGRY(CTRL_REG4, 0b00110000) #Continuos update, 2000 dps full scale

gyroXangle = 0.0
gyroYangle = 0.0
gyroZangle = 0.0
CFangleX = 0.0
CFangleY = 0.0


#Read our accelerometer,gyroscope and magnetometer  values
ACCx = readACCx()
ACCy = readACCy()
ACCz = readACCz()
GYRx = readGYRx()
GYRy = readGYRx()
GYRz = readGYRx()
MAGx = readMAGx()
MAGy = readMAGy()
MAGz = readMAGz()
##Convert Accelerometer values to degrees
AccXangle =  (math.atan2(ACCy,ACCz)+M_PI)*RAD_TO_DEG
AccYangle =  (math.atan2(ACCz,ACCx)+M_PI)*RAD_TO_DEG
AccZangle =  (math.atan2(ACCx,ACCy)+M_PI)*RAD_TO_DEG
```

```
#Convert Gyro raw to degrees per second
rate_gyr_x =  GYRx * G_GAIN
rate_gyr_y =  GYRy * G_GAIN
rate_gyr_z =  GYRz * G_GAIN



#Calculate the angles from the gyro. LP = loop period
gyroXangle+=rate_gyr_x*LP
gyroYangle+=rate_gyr_y*LP
gyroZangle+=rate_gyr_z*LP

#Complementary filter used to combine the accelerometer and gyro values.
CFangleX=AA*(CFangleX+rate_gyr_x*LP) +(1 - AA) * AccXangle
CFangleY=AA*(CFangleY+rate_gyr_y*LP) +(1 - AA) * AccYangle

#Calculate headin
heading = 180 * math.atan2(MAGy,MAGx)/M_PI



if heading < 0:
    heading += 360

#Normalize accelerometer raw values.
accXnorm = ACCx/math.sqrt(ACCx * ACCx + ACCy * ACCy + ACCz * ACCz)
accYnorm = ACCy/math.sqrt(ACCx * ACCx + ACCy * ACCy + ACCz * ACCz)
```

```
        #Calculate pitch and roll
        #pitch = math.asin(accXnorm)
        #roll = -math.asin(accYnorm/math.cos(pitch))
        roll=(math.atan2(-ACCy,ACCz)*180)/M_PI
        pitch=(math.atan2(ACCx,math.sqrt(ACCy*ACCy+ACCz*ACCz))*180)/M_PI

        #Calculate the new tilt compensated values
        magXcomp = MAGx*math.cos(pitch)+MAGz*math.sin(pitch)
        magYcomp = MAGx*math.sin(roll)*math.sin(pitch)+MAGy*math.cos(roll)-
MAGz*math.sin(roll)*math.cos(pitch)

        #Calculate tiles compensated heading
        tiltCompensatedHeading = 180 * math.atan2(magYcomp,magXcomp)/M_PI

        if tiltCompensatedHeading < 0:
            tiltCompensatedHeading += 360
        time.sleep(0.25)

        pitch = round(pitch,2)
        roll = round(roll,2)
        gyroXangle = round(gyroXangle,2)
        gyroYangle = round(gyroYangle,2)
        gyroZangle = round(gyroZangle,2)
        heading = round(heading,2)
        return pitch,roll,gyroXangle,gyroYangle,gyroZangle,heading
```

```
    except IOError:
        roll='i/o error'
        pitch=roll
        gyroXangle=roll
        gyroYangle=roll
        gyroZangle=roll
        heading=roll
        return pitch,roll,gyroXangle,gyroYangle,gyroZangle,heading
```

8.1.2 imu_temp

```
#!/usr/bin/python
def IMUtemp():
    try:
        from smbus import SMBus
        from time import sleep
        from ctypes import c_short

        addr = 0x77
        oversampling = 3        # 0.3

        bus = SMBus(1);         # 0 for R-Pi Rev. 1, 1 for Rev. 2

        # return two bytes from data as a signed 16-bit value
        def get_short(data, index):
            return c_short((data[index] << 8) + data[index + 1]).value

        # return two bytes from data as an unsigned 16-bit value
```

```
def get_ushort(data, index):
        return (data[index] << 8) + data[index + 1]

(chip_id, version) = bus.read_i2c_block_data(addr, 0xD0, 2)

# Read whole calibration EEPROM data
cal = bus.read_i2c_block_data(addr, 0xAA, 22)

# Convert byte data to word values
ac1 = get_short(cal, 0)
ac2 = get_short(cal, 2)
ac3 = get_short(cal, 4)
ac4 = get_ushort(cal, 6)
ac5 = get_ushort(cal, 8)
ac6 = get_ushort(cal, 10)
b1 = get_short(cal, 12)
b2 = get_short(cal, 14)
mb = get_short(cal, 16)
mc = get_short(cal, 18)
md = get_short(cal, 20)

p0=1013.25
#print "Starting temperature conversion"
bus.write_byte_data(addr, 0xF4, 0x2E)
sleep(0.005)
(msb, lsb) = bus.read_i2c_block_data(addr, 0xF6, 2)
ut = (msb << 8) + lsb
```

```
#print "Starting pressure conversion"
bus.write_byte_data(addr, 0xF4, 0x34 + (oversampling << 6))
sleep(0.04)
(msb, lsb, xsb) = bus.read_i2c_block_data(addr, 0xF6, 3)
up = ((msb << 16) + (lsb << 8) + xsb) >> (8 - oversampling)

#print "Calculating temperature"
x1 = ((ut - ac6) * ac5) >> 15
x2 = (mc << 11) / (x1 + md)
b5 = x1 + x2
t = (b5 + 8) >> 4

#print "Calculating pressure"
b6 = b5 - 4000
b62 = b6 * b6 >> 12
x1 = (b2 * b62) >> 11
x2 = ac2 * b6 >> 11
x3 = x1 + x2
b3 = (((ac1 * 4 + x3) << oversampling) + 2) >> 2

x1 = ac3 * b6 >> 13
x2 = (b1 * b62) >> 16
x3 = ((x1 + x2) + 2) >> 2
b4 = (ac4 * (x3 + 32768)) >> 15
b7 = (up - b3) * (50000 >> oversampling)
```

```
            p = (b7 * 2) / b4

            x1 = (p >> 8) * (p >> 8)
            x1 = (x1 * 3038) >> 16
            x2 = (-7357 * p) >> 16
            p = p + ((x1 + x2 + 3791) >> 4)
            t=t/10.0
            p=p/100.0

                        #altitude;only applicable for h<11km(because temp lapse
rate varies with altitude)
                        h=((pow(p0/p,0.1903)-1)*(t+273.15))/0.0065
            t = round(t,2)
            p = round(p,2)
            h = round(h,2)
            return t,p,h
      except IOError:
            t='i/o error'
            p=t
            h=t
            return t,p,h
```

8.1.3 gps

```
#!/usr/bin/python
def gps_xl():
    try:
        import serial
        import time
        ser= serial.Serial('/dev/ttyAMA0', 9600)
        s1='$GPRMC'
        Time='0.0'
        Latitude='0.0'
        Longitude='0.0'
        Date='0.0'
    while 1:
            s=ser.readline()
            if s.find(s1)!=-1:
                 break
        if s[18] == 'A':

                #time
                hour= s[7:9]
                minute= s[9:11]
                second= s[11:13]

                #latitude
                lat= int(s[20:22])
                decilat= float(s[22:29])
                positionlat= s[30]
```

```
pos=lat+(decilat/60)

#longitude
lon= int(s[32:35])
decilong= float(s[35:42])
positionlong= s[43]
poslong=lon+(decilong/60)

if s[55]==',':

    #date
    day= s[56:58]
    month= s[58:60]
    year= s[60:62]
else:

    day= s[57:59]
    month=s[9:61]
    year= s[61:63]

Time='{}:{}:{}'.format(hour,minute,second)
position='Coordinate: {}:{}'.format(pos,poslong)
Date= '{}/{}/20{}'.format(day,month,year)

pos = round(pos,4)
poslong = round(poslong,4)
ser.close()
```

```
            return Date,Time,pos,poslong
        else:
                          Time='N/A'     #if gps don't get fix
            pos='N/A'
            poslong='N/A'
            Date='N/A'
            ser.close()
            return Date,Time,pos,poslong
    except IOError:
        Date='i/o error'   #if the module get disconnected or broken
        Time=Date
        pos=Date
        poslong=Date

        return Date,Time,pos,poslong
```

8.1.4 uv

```
def UVsensor():
    try:
        import time
        import SI1145.SI1145 as SI1145
        sensor=SI1145.SI1145()
        vis = sensor.readVisible()
        IR = sensor.readIR()
        UV = sensor.readUV()/100.0
        vis =round(vis,2)
        IR = round(IR,2)
        UV =round(UV,2)
        return vis,IR,UV
    except IOError:
        vis ='i/o error'
        IR = vis
        UV =vis
        return vis,IR,UV
```

8.1.5 cputemp

```
def getCPUtemperature():
    try:
        import os
        res = os.popen('vcgencmd measure_temp').readline()
        res = res.replace("temp=","").replace("'C\n","")
        res=float(res)
        res=round(res,2)
        return res
    except IOError:
        res='i/o error'
        return res
```

8.1.6 MAIN_run

```
#!/usr/bin/python
from serial import Serial
from IMUtemp import *
from gps_xl import *
from IMUorientation import *
from UVsensor import*
from getCPUtemperature import *
import os
from time import sleep
import numpy as np
import datetime
from math import *
time=0
i=1
while 1:
	a = gps_xl()
	b = IMUtemp()
	c = IMUorientation()
	d = UVsensor()
	e = getCPUtemperature()

	data = '{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{},{}\r\n'.format(a[0],a[1],a[2],a[3],b[0],b[1],c[0],c[1],c[2],c[3],c[4],c[5],d[0],d[1],d[2],e,b[2])
	ser = Serial('/dev/ttyAMA0',38400)
	ser.write(data)
	ser.close()
```

8.1.7 picture transmission

```
from PIL import Image
import numpy
from serial import Serial
from time import sleep
im=Image.open('pic1.jpg')
pixRGB=im.load()
r=numpy.zeros((640,480),dtype=numpy.int)
g=numpy.zeros((640,480),dtype=numpy.int)
b=numpy.zeros((640,480),dtype=numpy.int)
i=0
while(i<3):
    ser = Serial('/dev/ttyAMA0',38400)
    for row in range(0,640):
        for col in range(0,480):
            pix_xy=pixRGB[row,col]
            if(i==0):
                r[row,col]=pix_xy[0]
                ser.write(str(r[row,col]))
                ser.write(',')
            if(i==1):
                g[row,col]=pix_xy[1]
                ser.write(str(g[row,col]
                        ser.write(',')
            if(i==2):
                b[row,col]=pix_xy[2]
            ser.write(str(b[row,col]))
            ser.write(',')
    ser.write('E')
    i=i+1
```

8.1.8 MATLAB Image Processing Code

```
clc, close all, clear all
time=0;
for count=1:3
tic
%loading the image and calculating image parameters
image_no=num2str(count);
read_name=strcat('image',image_no,'.jpg');
image=imread(read_name);
[bw,masked]=createMaskGreen(image);
[B,L,N,A] = bwboundaries(bw);
res=size(bw);
h=res(1);                              %height of photo
w=res(2);                              %width of photo
truevalue=sum(sum(bw));                %pixels with value==1
total=h*w;                             %total number of pixels
pcrtnge = (truevalue/total)*100;       %percentage of green
per=num2str(pcrtnge);
                                       %for annotating image
g_per=strcat('The percentage of green in this image is:__',per,'%');
%-----------------------------------------------------------------------
%plotting the images on a figure window
```

```
f=figure('name',' Green Color Thresholding');
subplot(1,2,1)                          %plotting original image
imshow(image)
title('Original Image')
subplot(1,2,2)                          %plotting filtered image
imshow(read_name)
hold on;
for k=1:length(B),
   if(~sum(A(k,:)))
      boundary = B{k};
      plot(boundary(:,2), boundary(:,1), 'r','LineWidth',2);
   end
end
title('Green Filtered')
dim = [.2 .5 .3 .3];
annotation('textbox',dim,'String',g_per,'FitBoxToText','on');
%-----------------------------------------------------------------------
%saving image processed files
save_name= strcat('p_image_',image_no);
%savefig(save_name)                          %savig .fig file
print(f,'-dtiffn',save_name)                %saving as .tiff file
close figure 1                              %closing the figure
%-----------------------------------------------------------------------
time=time+toc;
end
time
```

8.1.9 Green Mask in MATLAB

```
function [BW,maskedRGBImage] = createMaskGreen(RGB)
I = RGB;
% Define thresholds for REDchannel(R)
rMin = 0.000;       rMax = 106.000;
% Define thresholds for GREENchannel(G)
gMin = 0.000;       gMax = 255.000;
% Define thresholds for BLUEchannel(B)
bMin = 0.000;       bMax = 114.000;
% Create mask based on chosen thresholds
BW = (I(:,:,1) >= rMin ) & (I(:,:,1) <= rMax) & (I(:,:,2) >= gMin ) & (I(:,:,2) <= gMax) & (I(:,:,3) >=
bMin ) & (I(:,:,3) <= bMax);
% Initialize output masked image based on input image.
maskedRGBImage = RGB;
% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
end
```

8.2 Additional Images

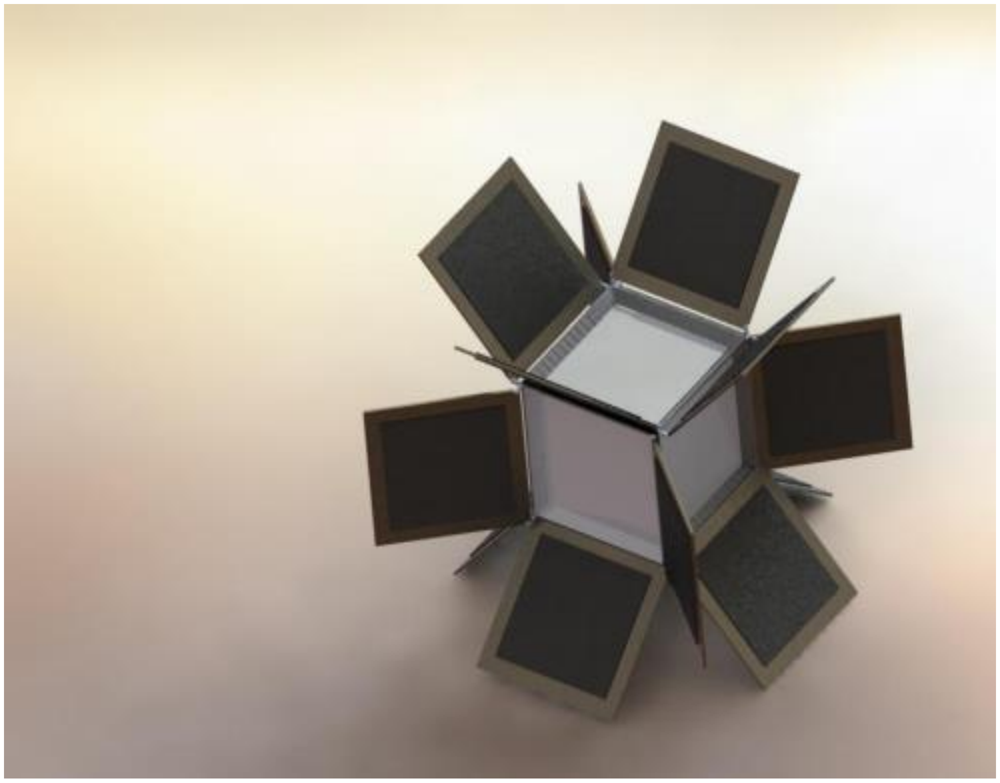


Figure 41: Nano-satellite structure of previous group

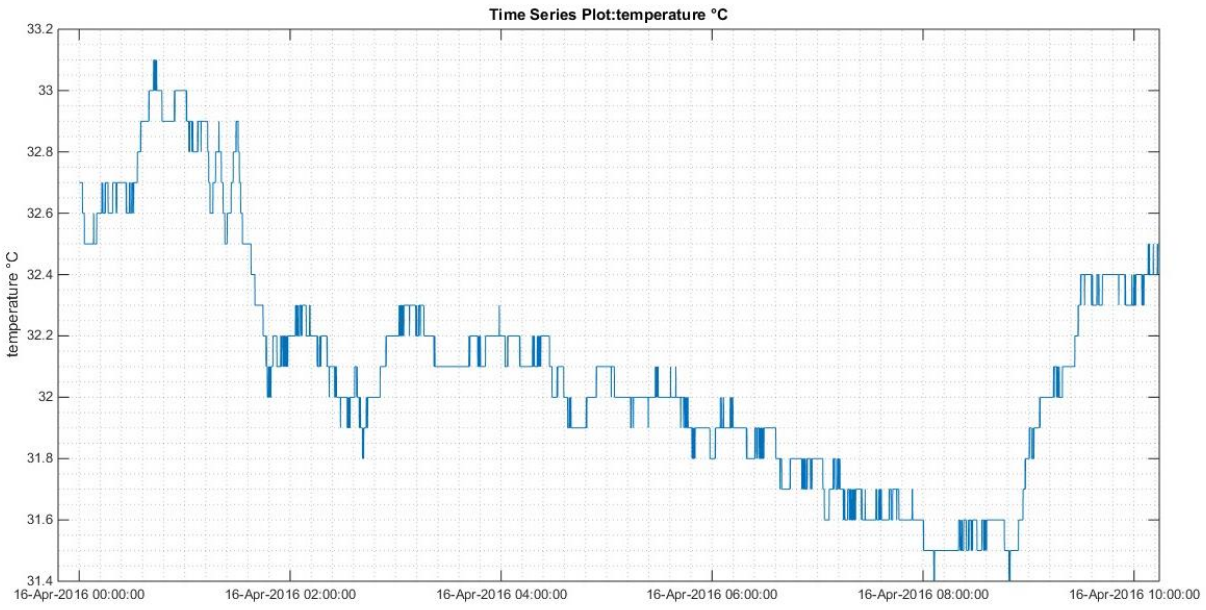


Figure 42: Temperature data logged over 10 hour period

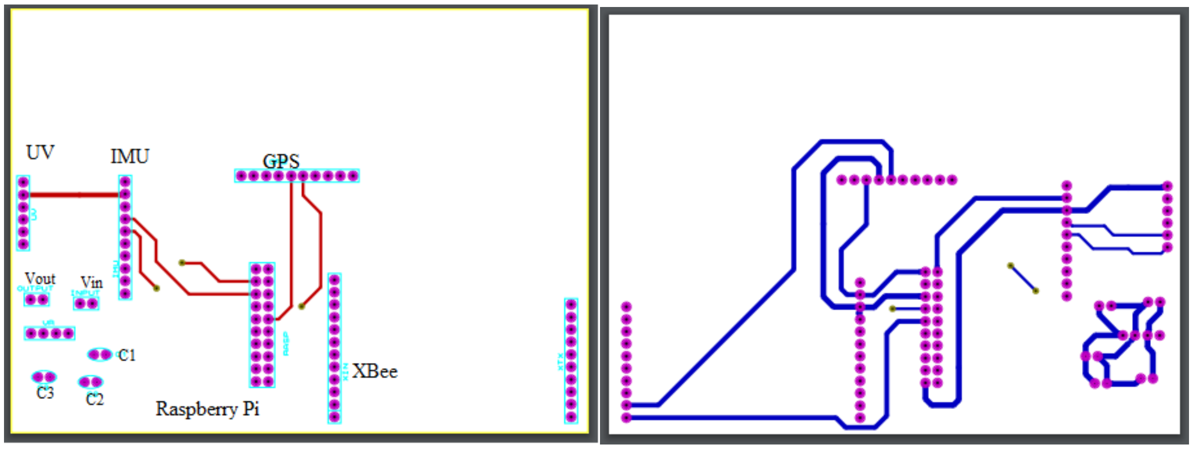


Figure 43: Topside and bottom side of the PCB

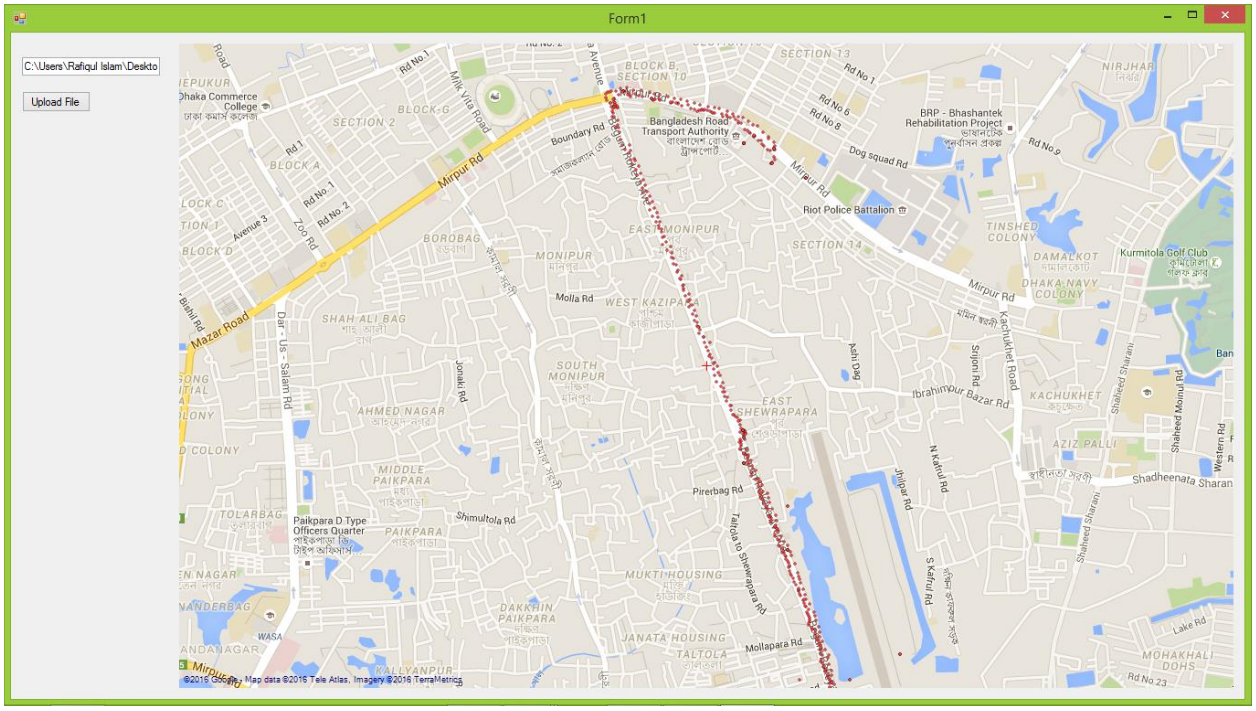


Figure 44: Plotting the BalloonSat path using gps data