

# COMMUNICATION AND DATA HANDLING SUBSYSTEM DESIGN

Below is the communication Subsystem overview :

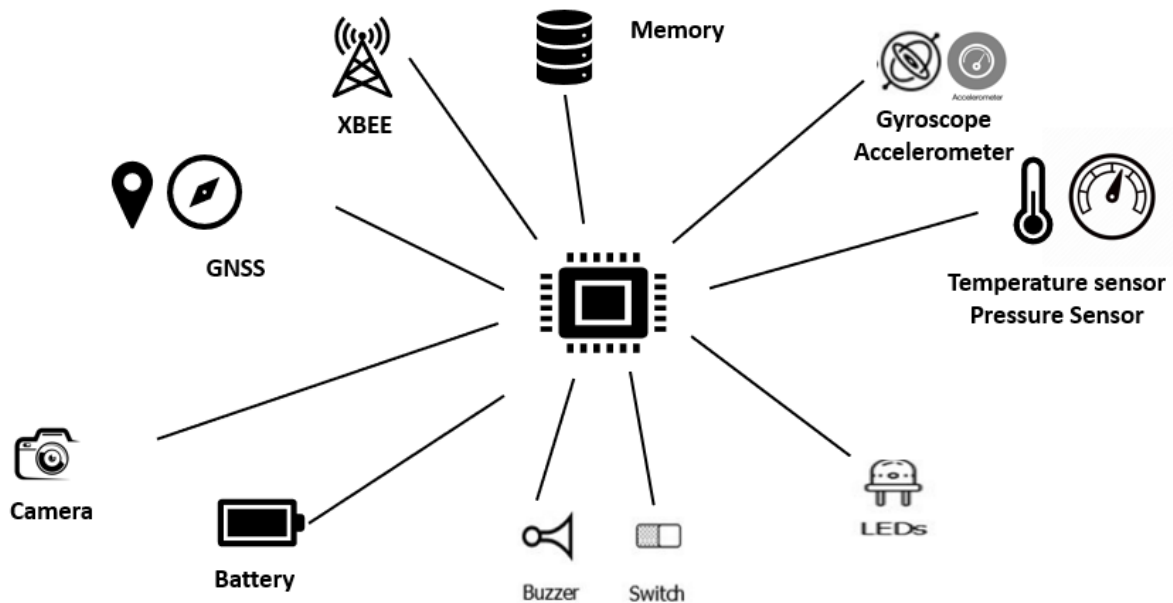


Fig. Communication and Data Handling Overview

Changes made after PDR in communication subsystem and data handling :

PDR	CDR	Rationale
Main processor was STM32F103	Main processor is changed to ATmega-328.	Due to problems faced in programming STM32F103 and because of previous experience of team in working with ATmega-328 and also meets all the requirements.

# TELEMETRY

- Upon powering up, the CANSAT collects the required telemetry at a 1 Hz sample rate.
- The telemetry data is being transmitted with ASCII comma separated fields followed by a carriage return in the following format :
- <TEAMID>,<TIMESTAMPING>,<PACKETCOUNT>,<ALTITUDE>,<PRESSURE>,<TEMP>,<VOLTAGE>,<GPSTIME>,<GPS LATITUDE>,<GPS LONGITUDE>,<GPS ALTITUDE>,<GPS SATS>,<ACCELEROMETER DATA>,<GYRO SPIN RATE>,<FLIGHT SOFTWARE STATE>,<ANY OPTIONAL DATA>
- *The received telemetry for the entire mission will be saved on the ground station computer as a comma-separated value (.csv) file with name of file as 2022-ASI-002.csv*

The NET ID/PAN ID of the Xbee is set to team no. as per the requirements and its setting is shown below :

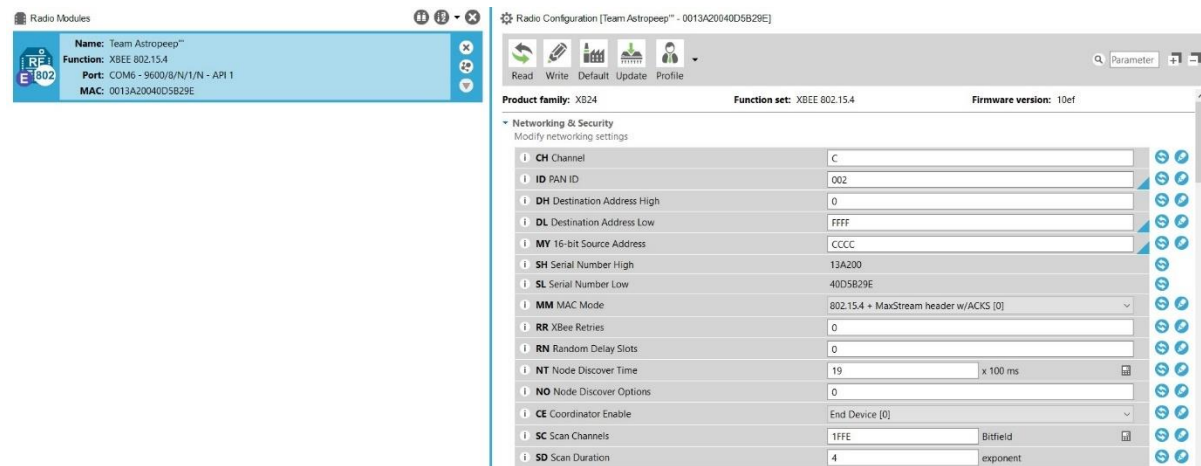


Fig. . Xbee 1 Setting for NET/PAN ID

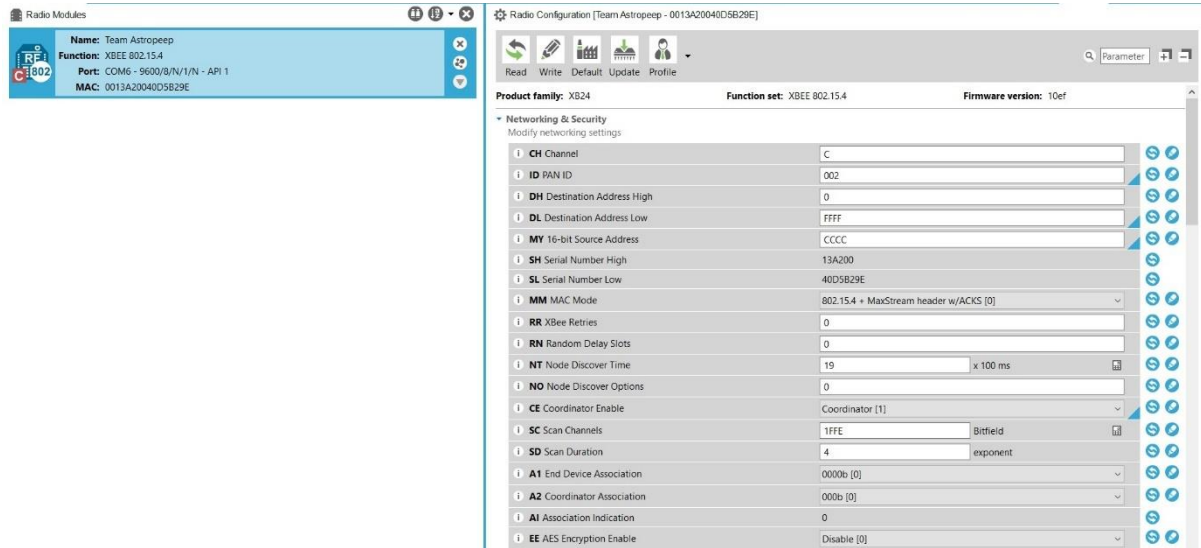


Fig. Xbee 2 Setting for NET/PAN ID

# TELEMETRY DATA FORMAT

TM Parameter	Function	Resolution /Format
<TEAM ID>	Team Number	2022ASI-002
<TIME STAMPING>	Time since the initial power	Seconds
<PACKET COUNT>	Count of transmitted packets	Integer
<ALTITUDE>	Altitude in units of meters and must be relative to ground	0.1 meters
<PRESSURE>	Measurement of atmospheric pressure	1 pascal
<TEMP>	Temperature in Celsius	0.01°C
<VOLTAGE>	Voltage of the CANSAT power bus	0.01 Volts
<GPS TIME>	Time generated by the GPS receiver	Seconds
<GPS LATITUDE>	Latitude generated by the GPS receiver	0.0001 degrees
<GPS LONGITUDE>	Longitude generated by the GPS receiver	0.0001 degrees
<GPS ALTITUDE>	Altitude generated by the GPS receiver	0.1 meters

# COMMUNICATION AND LINK BUDGET

Some changes are done in which we revisited the Gain calculation of receiving and transmitting antenna and we rectified accordingly as shown below.

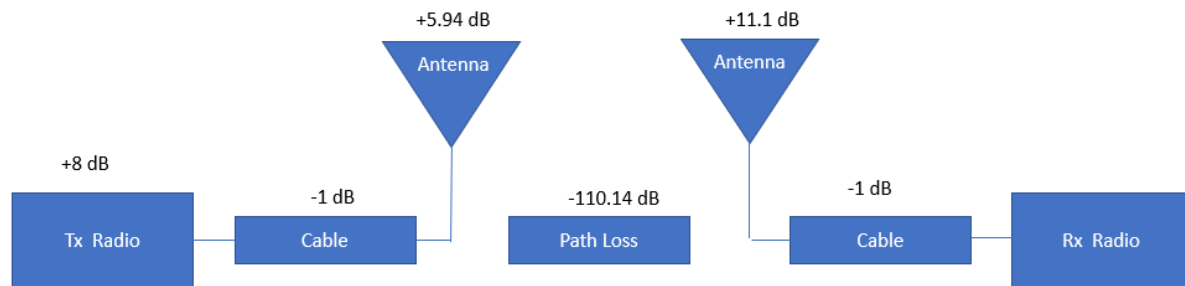


Fig. Communication and Link Budget

## Calculation :

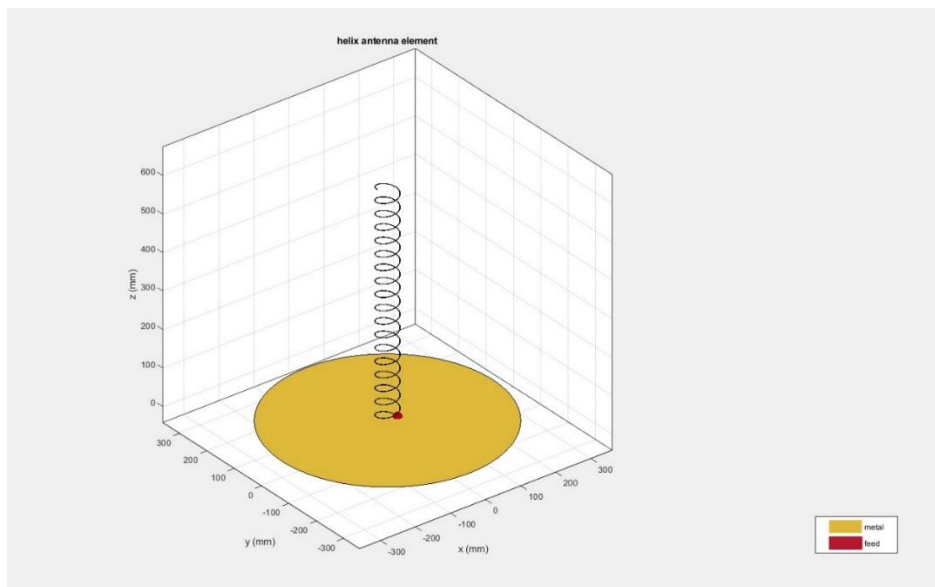
- Transmitted Power : 8 dB
- Transmitter Antenna Gain : 5.94 dB
- Transmitter Loss : 1 dB
- Free Space Loss : 110.14 dB
- Miscellaneous Loss : 1 dB
- Receiver Antenna Gain : 11.1 dB
- Receiver Loss : 1 dB
- Received Power : -84.14 dB
- Receiver Sensitivity : -101 dB
- Link Margin : 16.86 dB

# ANTENNA SELECTION

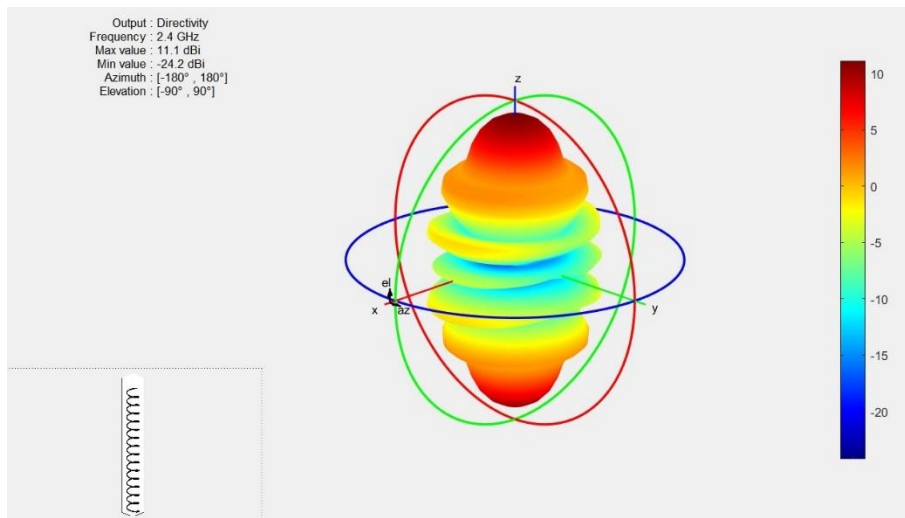
There is no changes since PDR just we simulated the radiation pattern of antennas and found the gains accordingly.

## GCS Antenna

Antenna Design :



## Antenna Radiation Pattern :



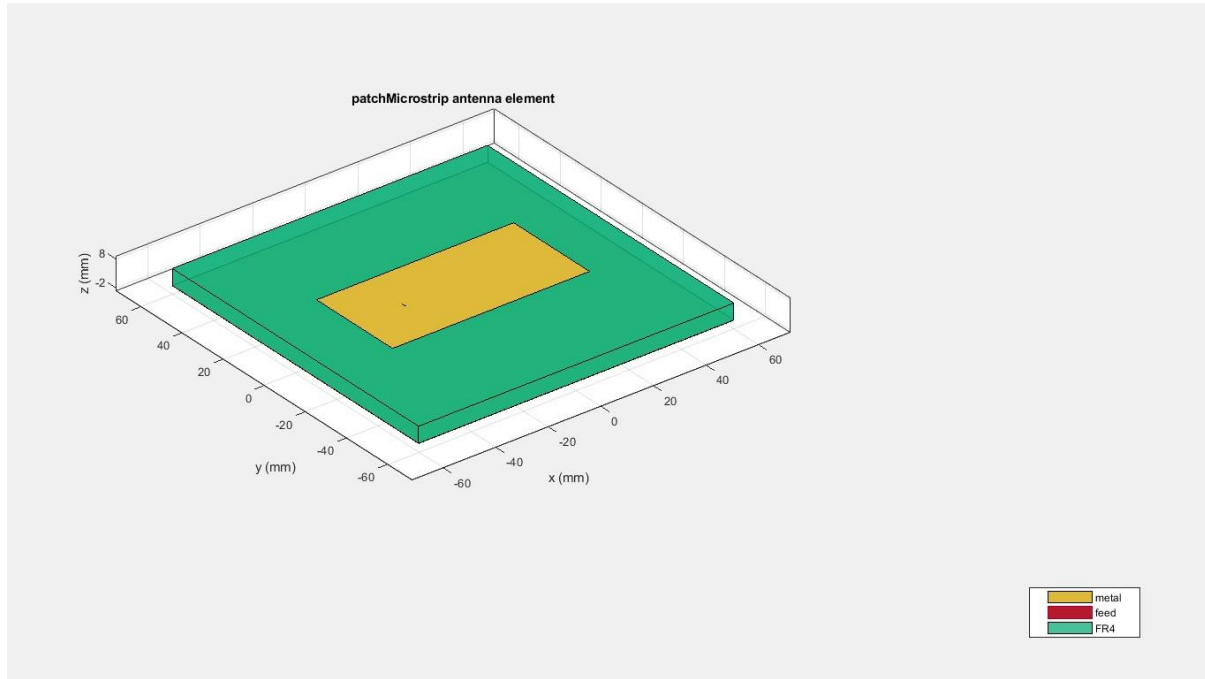
## Antenna Design Code :

```

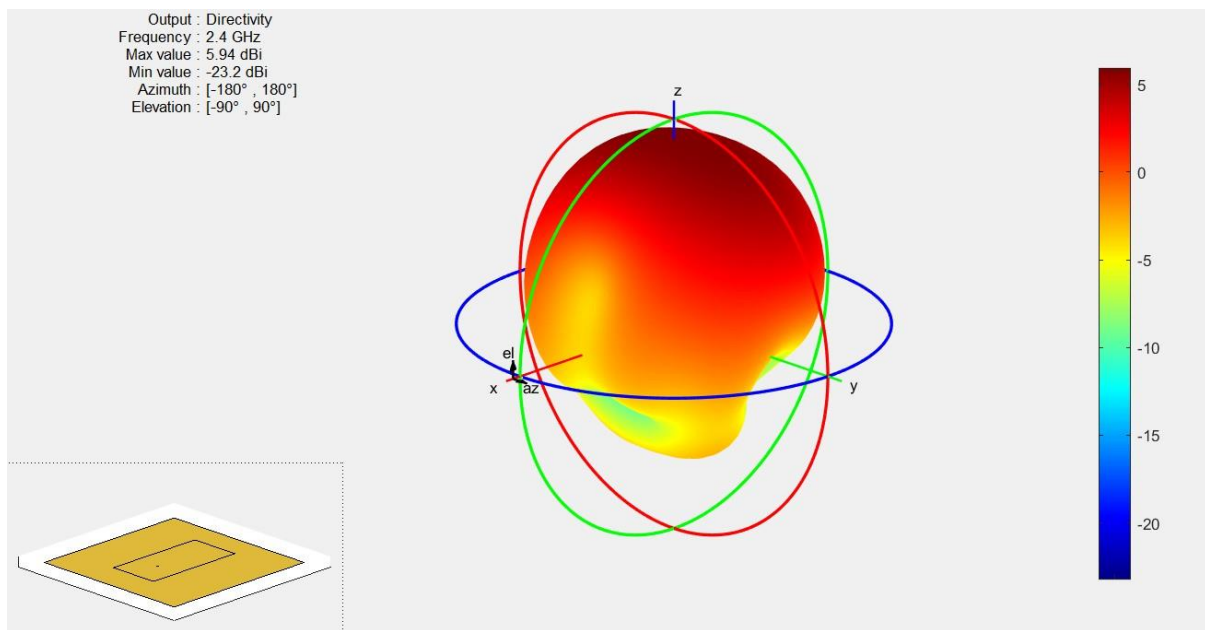
Editor - Untitled2*
Untitled2* x +
1      r      = 0.3e-3;
2      width   = cylinder2strip(r);
3      feedheight = 3*r;
4      D       = 56e-3;
5      radius  = D/2;
6      turns   = 17.5;
7      pitch   = 11.2;
8      spacing = helixpitch2spacing(pitch,radius);
9      side    = 600e-3;
10     radiusGP = side/2;
11     fc = 2.4e9;
12     hx = helix('Radius',radius,'Width',width,'Turns',turns,...
13              'Spacing',spacing,'GroundPlaneRadius',radiusGP,...
14              'FeedStubHeight',feedheight);
15     figure;
16     show(hx);
    
```

# CANSAT Antenna

Antenna Design :



Antenna Radiation Pattern :





## Antenna Design Code :

```

Editor - Untitled*
Untitled* x +
1 d = dielectric('FR4');
2 pm = patchMicrostrip('Length',75e-3,'Width',37e-3, ...
3     'GroundPlaneLength',120e-3,'GroundPlaneWidth',120e-3, ...
4     'Substrate',d);

```

The gain obtained after analysis are :

- GCS Antenna – 11.1 db
- Cansat Antenna – 5.94 db

After calculating we rectified the changes in the Communication and Link Budget.

# ELECTRICAL POWER SUBSYSTEM

There are only some minor changes in Electrical power subsystem as mentioned below :

- The battery configuration is changed from series to parallel which made the step down process more efficient.
- The position of switch is also modified from bottom to panel side of CANSAT.

## Electrical Block Diagram

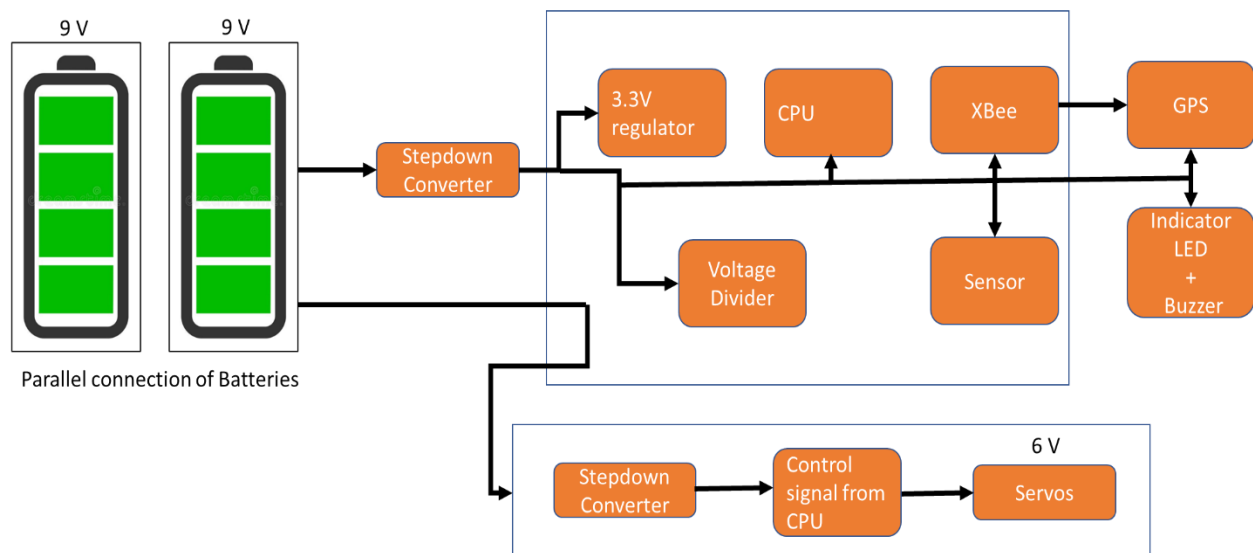


Fig. Electrical Power Subsystem Block Diagram

# FLIGHT SOFTWARE DESIGN

## Overview of the CanSat FSW Design

- The CanSat will collect sensor data then save to SD card and send to ground station via XBee.
- The CanSat will deploy 2<sup>nd</sup> Parachute deploy mechanism after reaching 500m altitude.
- The buzzer will keep beeping after landing until turned off with power switch.

## Programming Language

- C/C++ for CanSat container
- Python/C++ for ground station

## Development Environment

- Arduino IDE
- Visual Studio Code
- Spyder

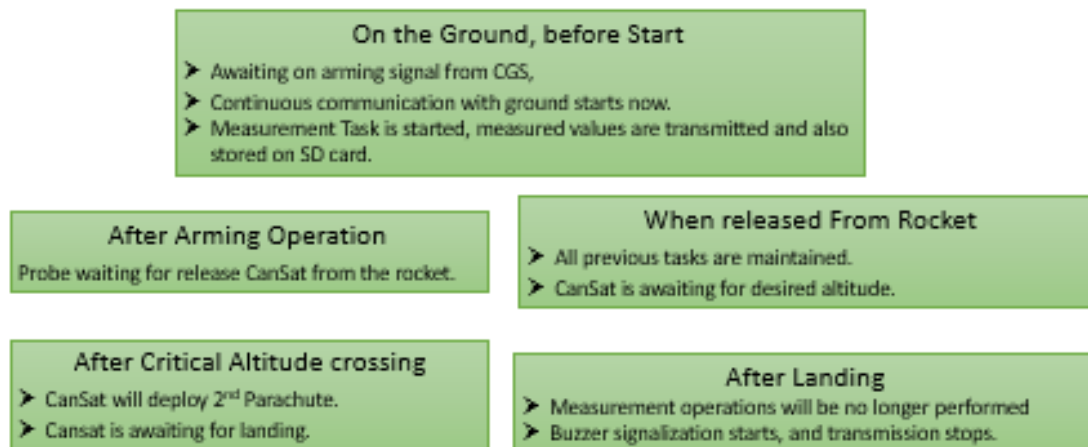


Fig. Flight Software Design Overview

## Momentary Power Cut Precaution Strategy :

As far as power management goes, all devices are ON for all time except two (Servo motor for Deployment of 2<sup>nd</sup> Parachute and Buzzer for recovery) as Servo motor will be deployed only for some time after crossing critical altitude (500m) and Buzzer will be turned ON after landing for location and recovery of CanSat.

After reset of processor in flight mode, previous state of mission together with last packet count can be retrieved from flash.

This information with current GNSS time is sufficient to continue mission.

# GCS DESIGN AND OVERVIEW

## Overview

- Main Computer is a laptop running GS application. The application is configured in GUI so that an appropriate port and baud rate can be selected.
- QHA antenna receives data from the probe (i.e. CanSat CPU) and transmits commands.
- Xbee PRO S2C module forwards and receive data to form communication link between CanSat and GCS.
- GS application, written in C++/Python, saves and displays data. It also saves received data in a local area network.

There is no changes since PDR.

## Data Flow and Components

- The antenna receives data over radio from the probe.
- Using Xbee explorer cable , forwards data to the Xbee module.
- Xbee PRO S2C module forwards data to GS's Main Computer using micro USB to USB cable.
- Software developed in C++ parses and then displays data in engineering units and save them into a CSV file.
- Main computer is capable of sending predefined commands to the probe.

## Progress Since PDR :

- The login page of GUI is completed and currently working on dashboard page of GUI and the overall data receiving software is tested for sensors individually.
- The development of helical antenna of GCS is started.

## Ground Control Station GUI :

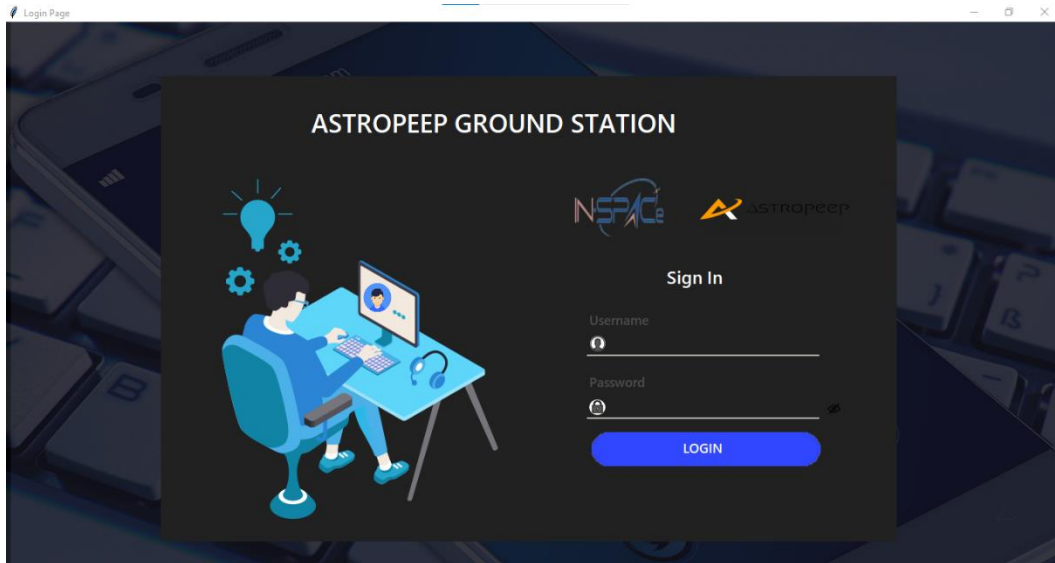
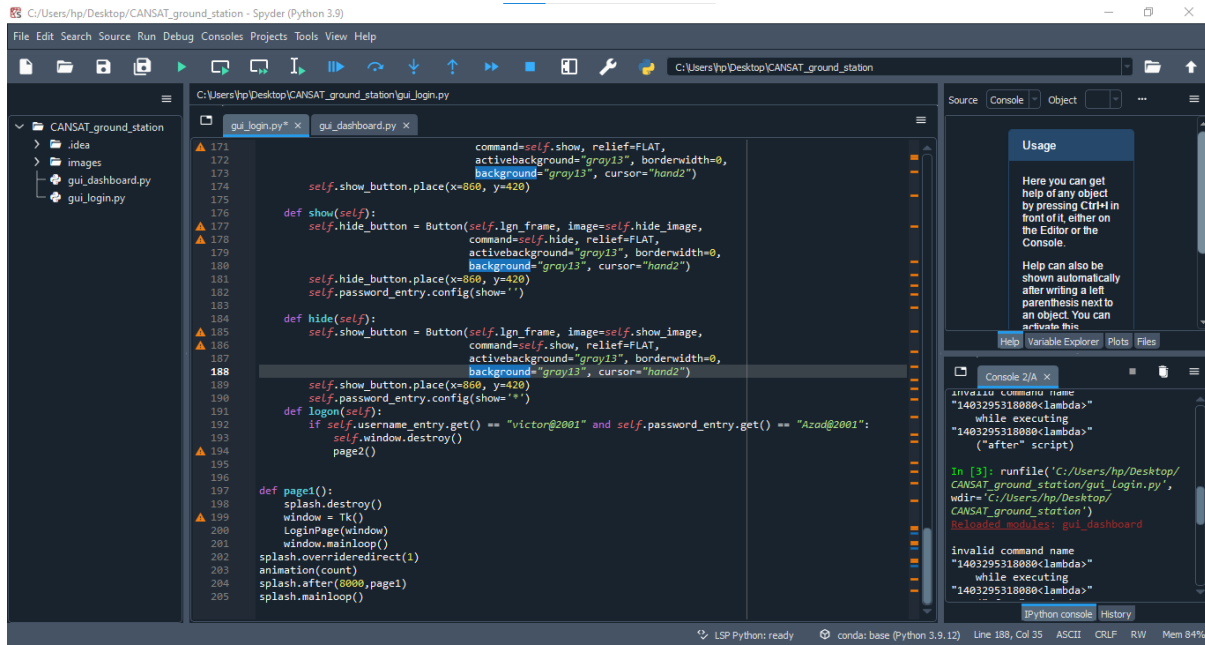


Fig. GCS GUI login page

The GUI is build using SPYDER IDE using Python language and the work is still in progress the login page is build as shown above and the dashboard work is in progress and for testing we individually tested the sensor by sending data using our microprocessor and plotted graph in the SPYDER IDE using Python function matplotlib.

The screenshots of code written in SPYDER IDE are attached below :



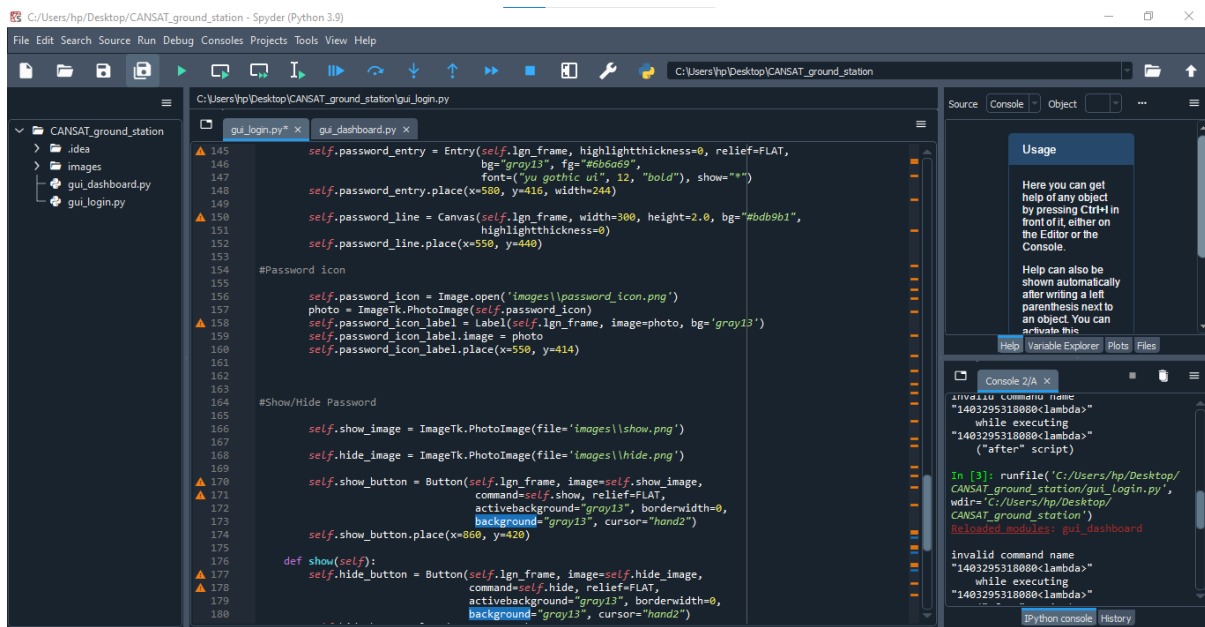
```

C:\Users\hp\Desktop\CANSAT_ground_station - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\hp\Desktop\CANSAT_ground_station\gui_login.py

171         command=self.show, relief=FLAT,
172         activebackground='gray13', borderwidth=0,
173         background='gray13', cursor='hand2')
174     self.show_button.place(x=860, y=420)
175
176     def show(self):
177         self.hide_button = Button(self.lgn_frame, image=self.hide_image,
178         command=self.hide, relief=FLAT,
179         activebackground='gray13', borderwidth=0,
180         background='gray13', cursor='hand2')
181         self.hide_button.place(x=860, y=420)
182         self.password_entry.config(show='')
183
184     def hide(self):
185         self.show_button = Button(self.lgn_frame, image=self.show_image,
186         command=self.show, relief=FLAT,
187         activebackground='gray13', borderwidth=0,
188         background='gray13', cursor='hand2')
189         self.show_button.place(x=860, y=420)
190         self.password_entry.config(show='')
191     def login(self):
192         if self.username_entry.get() == "victor@2001" and self.password_entry.get() == "Azad@2001":
193             self.window.destroy()
194             page2()
195
196     def page1():
197         splash.destroy()
198         window = Tk()
199         LoginPage(window)
200         window.mainloop()
201     splash.overridedirect(1)
202     animation(count)
203     splash.after(8000, page1)
204     splash.mainloop()
205

```



```

C:\Users\hp\Desktop\CANSAT_ground_station - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\hp\Desktop\CANSAT_ground_station\gui_login.py

145     self.password_entry = Entry(self.lgn_frame, highlightthickness=0, relief=FLAT,
146     bg='gray13', fg='#6b6a69',
147     font=('yu gothic ui', 12, 'bold'), show='')
148     self.password_entry.place(x=580, y=416, width=244)
149
150     self.password_line = Canvas(self.lgn_frame, width=300, height=2.0, bg='#bdb9b1',
151     highlightthickness=0)
152     self.password_line.place(x=550, y=440)
153
154     #Password icon
155
156     self.password_icon = ImageTk.PhotoImage(self.password_icon.png')
157     photo = ImageTk.PhotoImage(self.password_icon)
158     self.password_icon_label = Label(self.lgn_frame, image=photo, bg='gray13')
159     self.password_icon_label.image = photo
160     self.password_icon_label.place(x=550, y=414)
161
162
163     #Show/Hide Password
164
165     self.show_image = ImageTk.PhotoImage(file='images\show.png')
166     self.hide_image = ImageTk.PhotoImage(file='images\hide.png')
167
168     self.show_button = Button(self.lgn_frame, image=self.show_image,
169     command=self.show, relief=FLAT,
170     activebackground='gray13', borderwidth=0,
171     background='gray13', cursor='hand2')
172     self.show_button.place(x=860, y=420)
173
174     def show(self):
175         self.hide_button = Button(self.lgn_frame, image=self.hide_image,
176         command=self.hide, relief=FLAT,
177         activebackground='gray13', borderwidth=0,
178         background='gray13', cursor='hand2')
179

```

C:\Users\hp\Desktop\CANSAT\_ground\_station - Spyder (Python 3.9)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\hp\Desktop\CANSAT\_ground\_station\gui\_login.py

```

109         font=("yu gothic ui", 12, "bold"))
110
111     self.username_entry.place(x=580, y=359, width=270)
112     self.username_line = Canvas(self.lgn_frame, width=300, height=2.0, bg="#bdb9b1",
113                               highlightthickness=0)
114     self.username_line.place(x=550, y=359)
115
116     #Username_icon
117
118     self.username_icon = Image.open('images\username_icon.png')
119     photo = ImageTk.PhotoImage(self.username_icon)
120     self.username_icon_label = Label(self.lgn_frame, image=photo, bg='gray13')
121     self.username_icon_label.image = photo
122     self.username_icon_label.place(x=550, y=332)
123
124     #Login_button
125
126     self.lgn_button = Image.open('images\lbtn1.png')
127     photo = ImageTk.PhotoImage(self.lgn_button)
128     self.lgn_button_label = Label(self.lgn_frame, image=photo, bg='gray13')
129     self.lgn_button_label.image = photo
130     self.lgn_button_label.place(x=550, y=450)
131     self.login = Button(self.lgn_button_label, text='LOGIN', font=("yu gothic ui", 13, "bold"),
132                        width=25, bd=0,
133                        bg="#3047ff", cursor='hand2', activebackground="#3047ff",
134                        fg='white', command=self.login)
135     self.login.place(x=20, y=10)
136
137
138     #Password
139
140     self.password_label = Label(self.lgn_frame, text="Password", bg="gray13", fg="#4f4e4d",
141                               font=("yu gothic ui", 13, "bold"))
142     self.password_label.place(x=550, y=380)
143
144

```

Usage

Here you can get help of any object by pressing Ctrl+In front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this

Help Variable Explorer Plots Files

Console 2/A X

```

Invalid Command name:
"1403295318080<lambda>"
while executing
"1403295318080<lambda>"
(after" script)

In [3]: runfile('C:/Users/hp/Desktop/
CANSAT_ground_station/gui_login.py',
wd1='C:/Users/hp/Desktop/
CANSAT_ground_station')
Reloaded modules: gui_dashboard

Invalid command name:
"1403295318080<lambda>"
while executing
"1403295318080<lambda>"
(after" script)

IPython console History

```

C:\Users\hp\Desktop\CANSAT\_ground\_station - Spyder (Python 3.9)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\hp\Desktop\CANSAT\_ground\_station\gui\_login.py

```

73
74
75     #logos
76     #in_space
77
78     self.sign_in_image = Image.open('images\in_space.png')
79     self.sign_in_image = self.sign_in_image.resize((120,75))
80     photo = ImageTk.PhotoImage(self.sign_in_image)
81     self.sign_in_image_label = Label(self.lgn_frame, image=photo, bg='gray13')
82     self.sign_in_image_label.image = photo
83     self.sign_in_image_label.place(x=530, y=130)
84
85     #astro_logo
86
87     self.sign_in_image2 = Image.open('images\astro_logo.png')
88     self.sign_in_image2 = self.sign_in_image2.resize((300,75))
89     photo = ImageTk.PhotoImage(self.sign_in_image2)
90     self.sign_in_image_label = Label(self.lgn_frame, image=photo, bg='gray13')
91     self.sign_in_image_label.image = photo
92     self.sign_in_image_label.place(x=650, y=130)
93
94     #signin_label
95
96     self.sign_in_label = Label(self.lgn_frame, text="Sign In", bg="gray13", fg="white",
97                               font=("yu gothic ui", 17, "bold"))
98     self.sign_in_label.place(x=650, y=240)
99
100     #username
101
102     self.username_label = Label(self.lgn_frame, text="Username", bg="gray13", fg="#4f4e4d",
103                               font=("yu gothic ui", 13, "bold"))
104     self.username_label.place(x=550, y=380)
105
106     self.username_entry = Entry(self.lgn_frame, highlightthickness=0, relief=FLAT, bg="gray13",
107                               fg="#6b6b69",
108

```

Usage

Here you can get help of any object by pressing Ctrl+In front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this

Help Variable Explorer Plots Files

Console 2/A X

```

Invalid Command name:
"1403295318080<lambda>"
while executing
"1403295318080<lambda>"
(after" script)

In [3]: runfile('C:/Users/hp/Desktop/
CANSAT_ground_station/gui_login.py',
wd1='C:/Users/hp/Desktop/
CANSAT_ground_station')
Reloaded modules: gui_dashboard

Invalid command name:
"1403295318080<lambda>"
while executing
"1403295318080<lambda>"
(after" script)

IPython console History

```

LSP Python: ready conda: base (Python 3.9.12) Line 188, Col 35 ASCII CRLF RW Mem 82%



C:\Users\hp\Desktop\CANSAT\_ground\_station - Spyder (Python 3.9)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\hp\Desktop\CANSAT\_ground\_station

gui\_login.py\* x gui\_dashboard.py x

```

1 from gui_dashboard import *
2 from tkinter import *
3 from PIL import ImageTk, Image
4 #import customtkinter
5
6 #splash_screen
7
8 splash = Tk()
9 splash.title("Astropeep Ground Station")
10 height = 400
11 width = 600
12 x = ((splash.winfo_screenwidth()//2)-(width//2))
13 y = ((splash.winfo_screenheight()//2)-(height//2))
14 splash.geometry('{}x{}+{}+{}'.format(width,height,x,y))
15
16 gif_img = "images\\splash.gif"
17 open_img = Image.open(gif_img)
18 frames = open_img.n_frames
19 imageObject = [PhotoImage(file = gif_img,format = f"gif -index {i}") for i in range(frames)]
20 count = 0
21 showAnimation = None
22
23
24 def animation(count):
25     global showAnimation
26     new_img = imageObject[count]
27     gif_label.configure(image = new_img)
28     count += 1
29     if count == frames:
30         count = 0
31         showAnimation = splash.after(18,lambda: animation(count))
32
33 gif_label = Label(splash,image = "")
34 gif_label.place(x=0,y=0,width = 600,height = 400)
35
36

```

Usage

Here you can get help of any object by pressing Ctrl+H in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this:

Help Variable Explorer Plots Files

Console 2/A x

```

Invalid command name:
"1403295318080<lambda>"
while executing
"1403295318080<lambda>"
("after" script)

In [3]: runfile('C:/Users/hp/Desktop/
CANSAT_ground_station/gui_login.py',
wd1='C:/Users/hp/Desktop/
CANSAT_ground_station')
Reloaded modules: gui_dashboard

Invalid command name:
"1403295318080<lambda>"
while executing
"1403295318080<lambda>"

```

IPython console History

New file

ISP Python: ready conda: base (Python 3.9.12) Line 188, Col 35 ASCII CRUF RW Mem 88%

The development of dashboard were all the data will be plotted is in progress and after development the dashboard will be like as shown below :

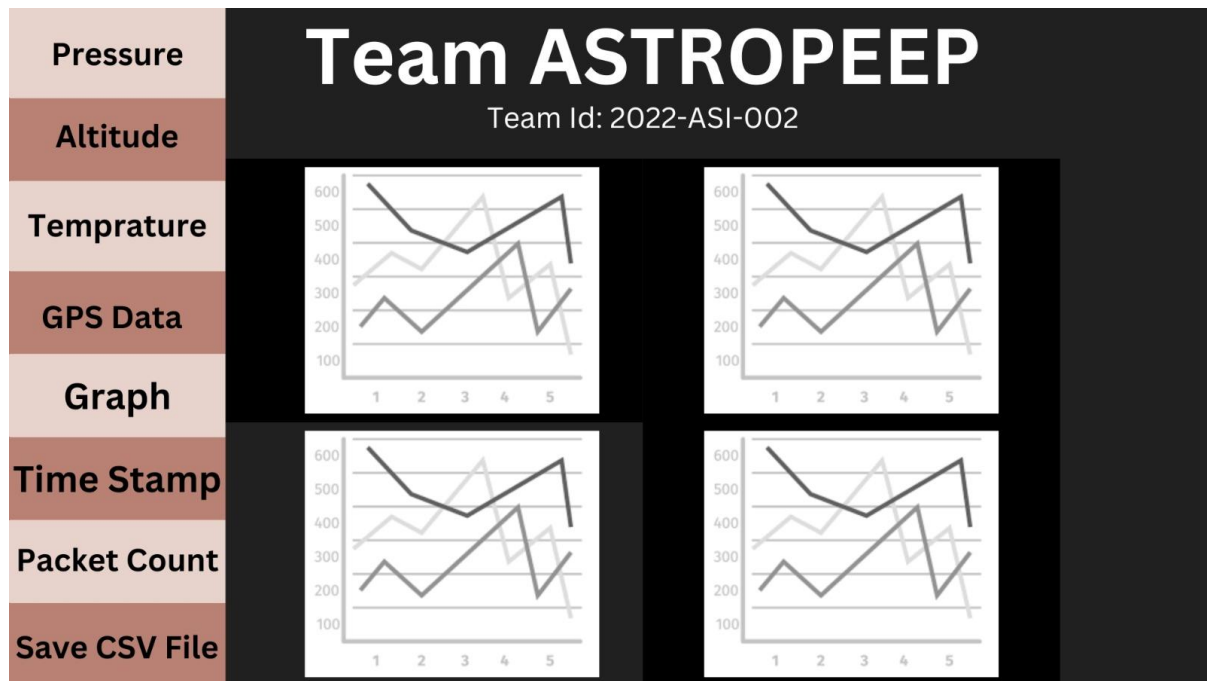


Fig. GCS GUI Dashboard

# CANSAT ALGORITHM

The CANSAT algorithm is developed in C/C++ language using Visual Studio Code and Arduino IDE and the flow chart of the algorithm is shown below :

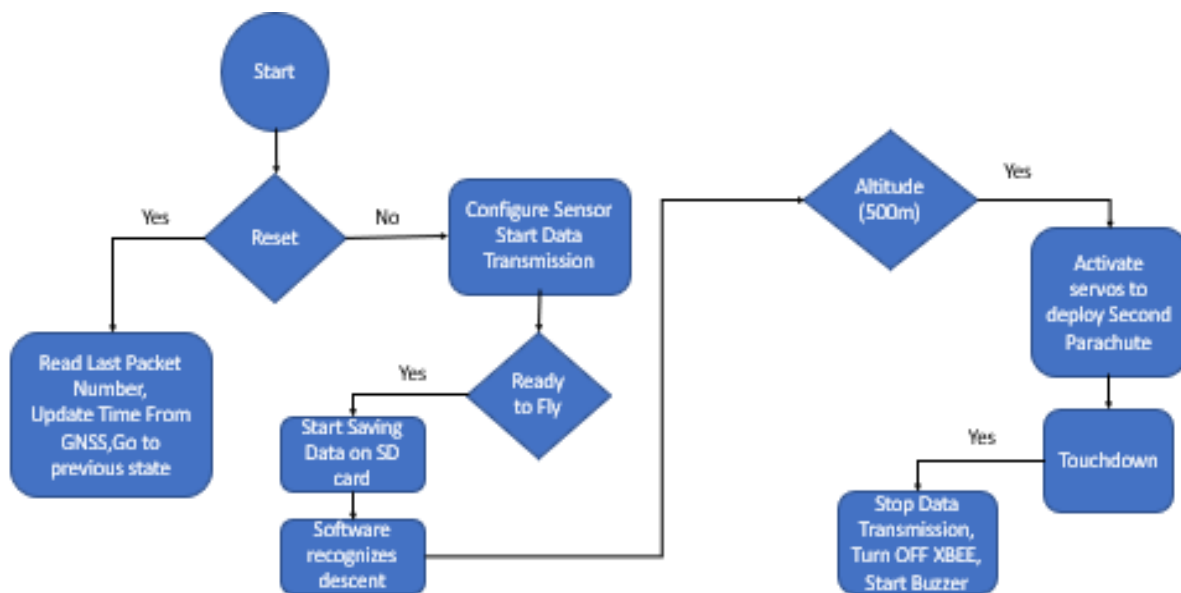


Fig. Cansat Algorithm Flow Chart

# GROUND STATION FINAL TEST RESULTS

The Cansat Ground Station final test results is still in progress as the test of data collection is done individually for all the sensors and in next step we will be combining the overall data into our GUI.

Our GUI work is also still in progress and final glimpse of our GUI is shown below :

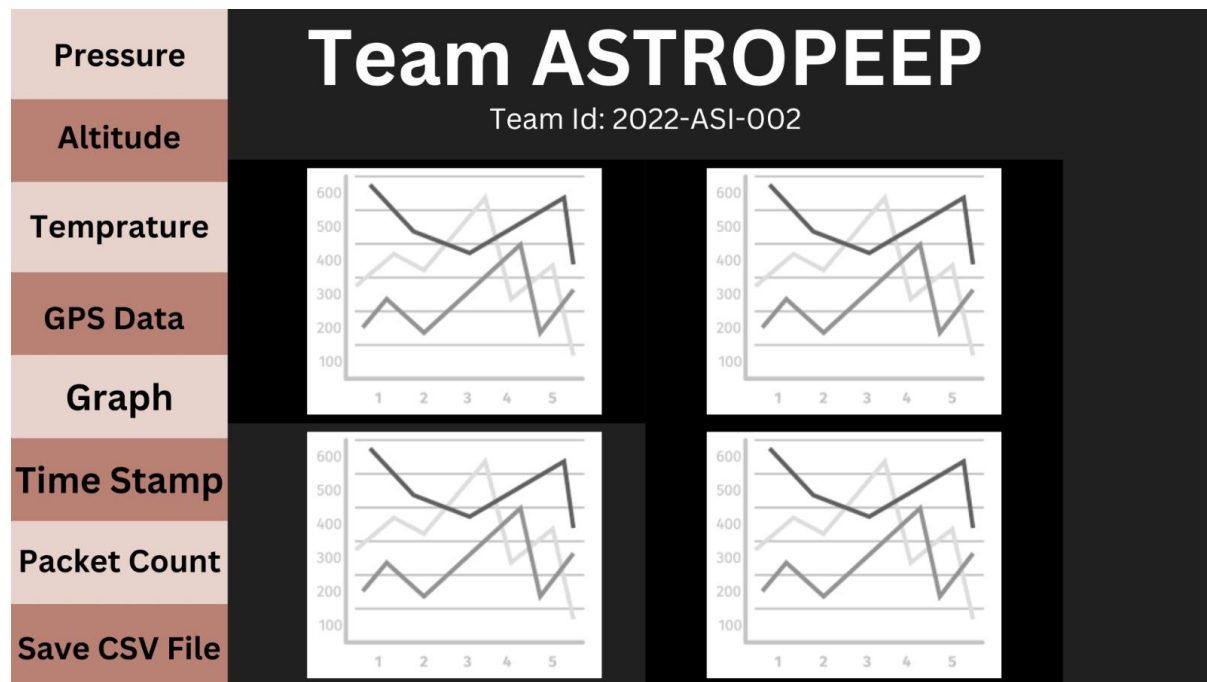


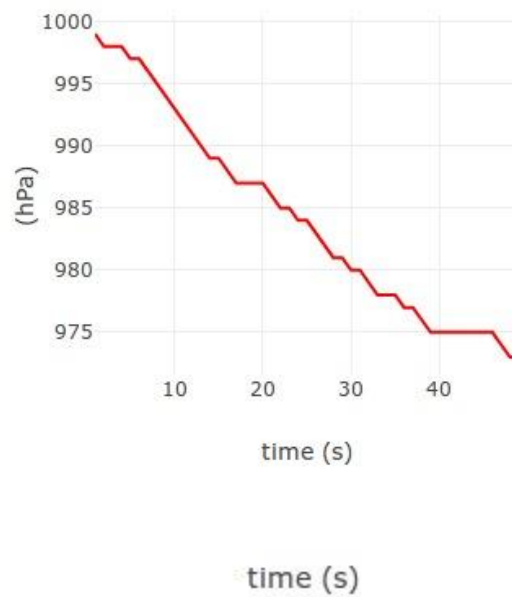
Fig. CANSAT GUI Dashboard

The test result plots of the individual sensor data are shown below :

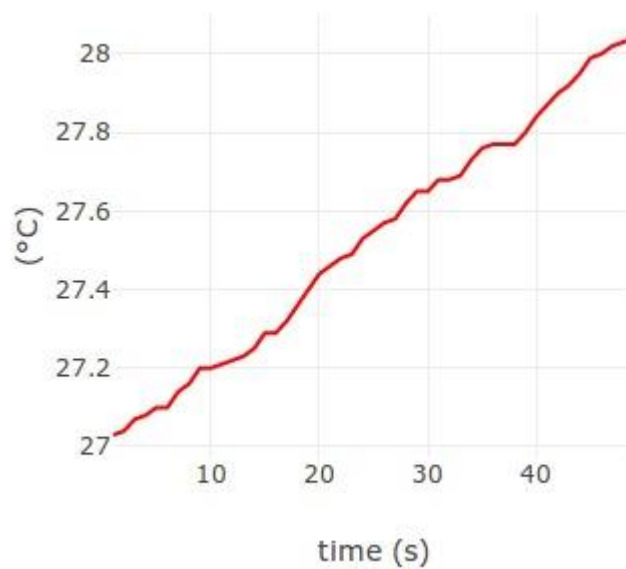
Altitude Test Plot :

Pressure Test Plot :

Temperature Test Plot :



The testing of some of the components are still in progress like GPS , Gyroscope and Accelerometer. We are still working on the testing of the above mentioned components.



# CANSAT INTEGRATION AND TESTING

## Testing before Integration :

CanSat subsystems level tests (before integration)	
Sensors	Sensors will be checked and properly calibrated separately, before mounting them to a CanSat. It will assure us, that all used components are fully functional and will not cause mission failure. Calibration will be done with usage of oscilloscope sending the data via UART to PC and checking if the values are correct.
CDH	Preliminary tests for data acquisition – checking if sample data send via radio modules can be properly handled by ground station software. Checking the time of sending data along working under various conditions. Checking if data frame format is as desired.
EPS	Supplying a voltage divider with various voltages of a battery working range from laboratory power supply to determine the correct behaviour of the circuitry. Checking for damage such as short circuits. Battery tests will be performed - maximum current consumed by hot wires need to be compromised with battery parameters - checking if they make hotwire hot enough. Checking battery lifetime. Voltage checking on the checkpoints.
Radio Communication	Sending and receiving known amount of packets and checking the loss percentage in various areas, such as between buildings/trees or on an open field. Verification of radio modules parameters and communication quality on ground.
FSW	Checking designed software functionalities such as: proper visualization of sample data, sending remote commands to the onboard computer. Sequences chronology and order will be tested in all possible scenarios and events.

CanSat subsystems level tests (before integration)	
Mechanical	All mechanical elements separately must be checked if they are made properly and have good dimensions. Then, components must be preassembled in order to see if a whole structure is stiff enough and have no looseness. Checking if fast moving and rotating structure does not cause any problems. If not, then we test all mechanisms used separately if they work properly – e.g. if wings can be opened and closed easily. Container has to be checked on the ground for its stiffness and durability and visually if a composite does not have any cracks and flaws. Container must be fluorescent color to be easy to find - test of luminescent additions to glass fiber composite
Descent Control	Checking if parachute descent is stable with a sample mass attached. For probe - checking if descent rate of the designed wings is as desired and what is their rotation rate. Test performed in aerodynamic tunnel. Rotation rate checked by suitable sensor.



CanSat subsystems level tests (after integration)	
Descent Testing	Checking if the flight is stable and detumbled with parachute deployed. Testing if the flight is stable with rotors. We check it by deploying CanSat from high building with known height - recording the test with the camera lets us calculate descent rates of parachute and rotors. Testing the parachute from a smaller height if it is stable and if the descent rate is appropriate. Testing if detumbling system works as desired and stabilizes Probe during auto-gyro descent and if recording from camera is stable. Preliminary tests were performed with our rocket and will be repeated with the drone and rocket again.
Communication	Checking on the ground if communication between CanSat and Ground Station works. Checking wireless modules, every sensor, the power modules, CanSat – ground station application. On that stage we can perform final data flow from sensors to its acquisition on the ground station application.
Mechanism	Testing if rotors open well and work in desired way and if elastic bands give enough force for unfolding rotors. Everything is well-fitted and the capsule is able to slide easily in the structure.

CanSat subsystems level tests (before integration)	
Sensors	Sensors will be checked and properly calibrated separately, before mounting them to a CanSat. It will assure us, that all used components are fully functional and will not cause mission failure. Calibration will be done with usage of oscilloscope sending the data via UART to PC and checking if the values are correct.
CDH	Preliminary tests for data acquisition – checking if sample data send via radio modules can be properly handled by ground station software. Checking the time of sending data along working under various conditions. Checking if data frame format is as desired.
EPS	Supplying a voltage divider with various voltages of a battery working range from laboratory power supply to determine the correct behaviour of the circuitry. Checking for damage such as short circuits. Battery tests will be performed - maximum current consumed by hot wires need to be compromised with battery parameters - checking if they make hotwire hot enough. Checking battery lifetime. Voltage checking on the checkpoints.
Radio Communication	Sending and receiving known amount of packets and checking the loss percentage in various areas, such as between buildings/trees or on an open field. Verification of radio modules parameters and communication quality on ground.
FSW	Checking designed software functionalities such as: proper visualization of sample data, sending remote commands to the onboard computer. Sequences chronology and order will be tested in all possible scenarios and events.

The above mentioned steps are for testing the cansat before integration and after this testing we will go ahead for integration.

## Integration of CANSAT :

- As our primary structure fabrication process which will let be a homogenous structure with no mechanical joints.
- Other than this the outer body panels will be connected using the shock absorbing screws.
- Batteries , PCB and Servo motors are connected to the particular damper used which are then connected to the structure of CANSAT using screws and other mechanical joints as shown below.

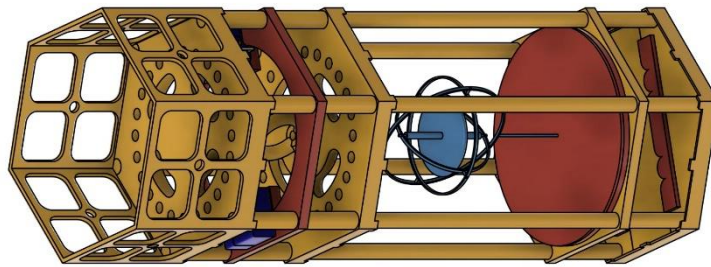


Fig. CANSAT Integration Stage 1

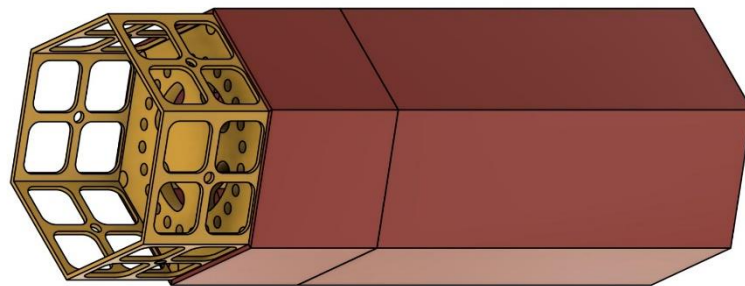
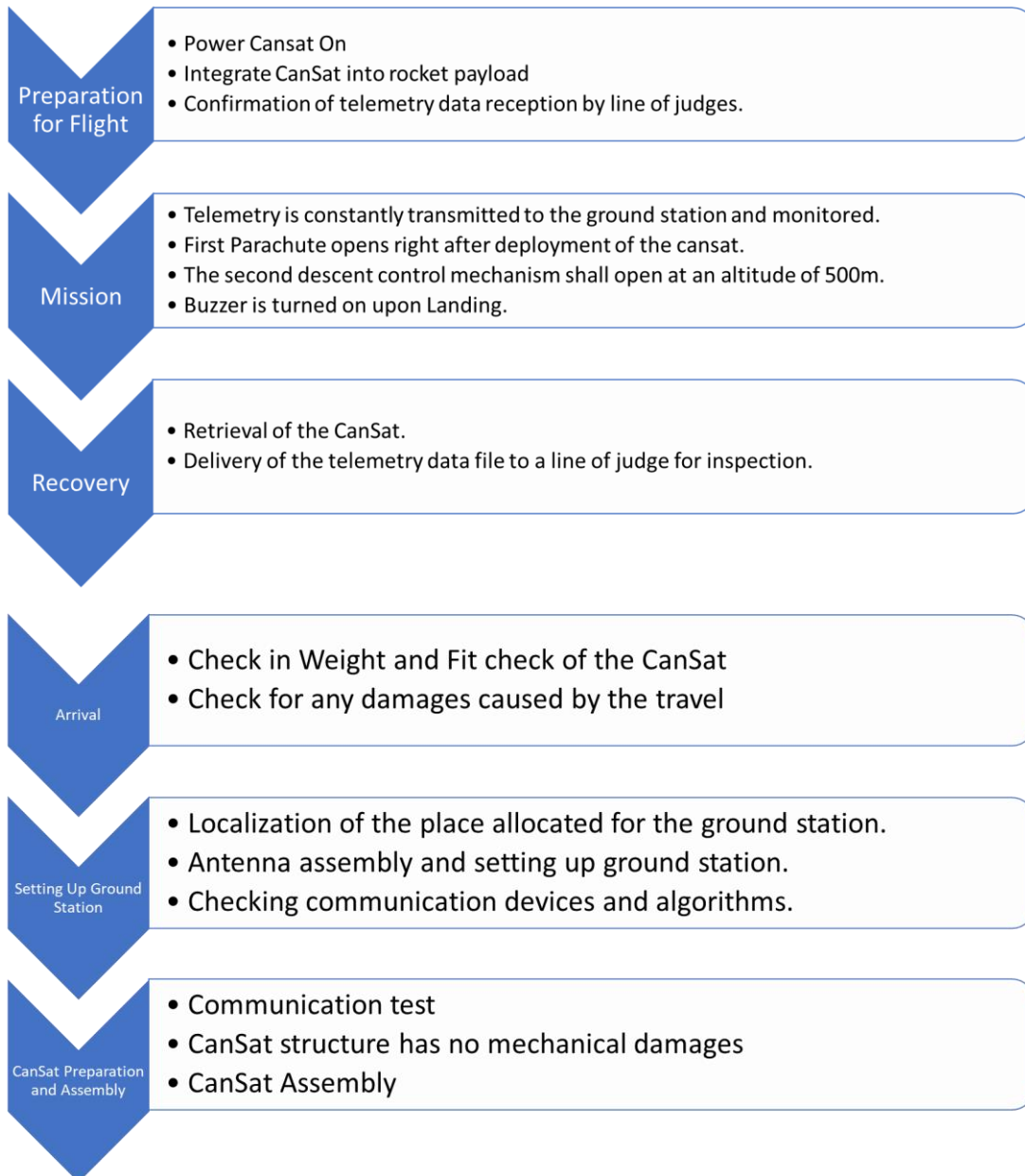


Fig. CANSAT Integration Stage 2



# MISSION OPERATION AND ANALYSIS



## Analysis

- Analysis of the obtained data
- Mission assessment and analysis

# PRE-FLIGHT REQUIREMENT- CHECK-ANALYSIS

## Setting up GS

- Computer and Antenna

## Communication Test

## Pre-Flight check and Tests

- Structure
- Electronics
- DCS components

## Cansat Integration

- Container assembly

## CanSat recovery

# **LOGISTICS AND** **TRANSPORTATION:**

For the Logistics and Transportation:

- We will be packing the fragile and non secured elements independently using cartons and bubble wraps.
- Whereas other all the elements will be packed in a wooden sheet box by using Thermocol sheets, Bubble Wraps, Cardboard and Straps as per the requirements of the fabricated model of CANSAT.

# **CANSAT READY TO LAUNCH** **AND FINAL COMMENTS**

- The testing were made on similar kind of prototypes and we found the rectifications in the model as observed.
- Accordingly we have made changes and as of that our software part is almost done however in Hardware our work is still in progress.
- Due to the delay in the delivery of the materials and the electronic components required for the final fabrication of the CANSAT.
- The work will be completed in accordance to the availability of material however all the analysis , simulations and test results are completed through the alternative scale models of the similar materials.

# CONCLUSION

## Accomplishments

- Telemetry is tested.
- GCS GUI is developed up to some extent.
- Antenna testing is completed.
- Mechanical analysis (Stress-strain , Vibration and Fit check) is completed.

## Unfinished work

- The fully integrated electronic & mechanical subsystem are not entirely functional yet and needs to be tested.
- Prototypes are being tested and more refinements are required.
- Flight software is in progress.

## Next stage of development

- Final Fabrication work completion
- Code Review
- Testing of integrated system
- Assembly of complete CANSAT
- Integrated testing of CANSAT.

