

Flight Software Description

1. Software Sequencing Timing Diagram

The success of any flight software system depends on its ability to coordinate key tasks in a well-defined sequence. A timing diagram is an essential tool that maps out when specific software operations—such as sensor data acquisition, telemetry transmissions, and execution of telecommands—must take place relative to the flight phase. This includes:

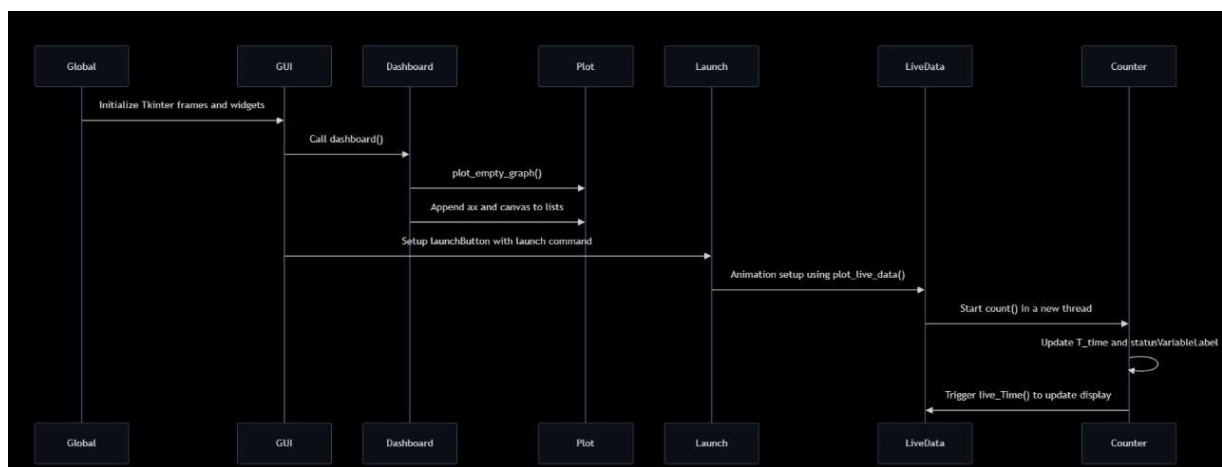
- Pre-launch phase: Software initialisation, sensor checks, and communication setup with the ground station.
- Launch and ascent: High-frequency sensor sampling, real-time telemetry transmission, and tracking of critical flight metrics like velocity, altitude, and acceleration.
- Descent and landing: Monitoring the parachute deployment system, slowing sensor sampling, and shutting down non-essential processes to preserve power and data integrity.

The sequencing diagram ensures that operations happen in a coordinated, non-overlapping manner, which is crucial for the smooth functioning of the rocket during critical phases.

Components of the Diagram

- **Global:-**
 - Initializes Tkinter frames and widgets, setting up the graphical user interface components that will be used later on.
- **GUI:-**
 - Responsible for calling the dashboard() function, which sets up the main components of the dashboard.
- **Dashboard:-**
 - Once the dashboard is called, an empty graph is plotted using the plot_empty_graph() function.
 - Axes and canvas for plotting are appended to lists, which will likely be used for dynamic graph updates.
 - The launchButton is set up with the command to launch the system, which connects the GUI to the core launch sequence.

- **Plot:-**
 - Handles the actual plotting of data.
 - `plot_live_data()` is used to set up the animation for real-time data visualization on the dashboard.
- **Launch:-**
 - The launch process is set up, likely involving a countdown or preparation stage before starting the actual telemetry tracking.
- **LiveData:-**
 - This section triggers the live data updates. The method `live_Time()` is invoked to update the display with real-time telemetry data.
- **Counter:-**
 - Manages a `count()` function that starts in a new thread, keeping track of time.
 - Updates `T_time` (likely a variable representing telemetry time) and `statusVariableLabel` to reflect the system's current state.



2. Sampling Rate

In-flight data from onboard sensors (accelerometers, gyroscopes, barometers, and temperature sensors) is gathered at a carefully calculated rate of 5 Hz (adjusted to actual rate). This high-frequency sampling ensures the rocket's rapid movements and environmental changes are captured with precision, allowing detailed analysis of the rocket's behavior during ascent and descent.

Rationale for the Sampling Rate:

- **Real-Time Monitoring:** A sampling rate of 5 Hz ensures that critical flight parameters are updated frequently on the telemetry dashboard, allowing for real-time monitoring of the rocket's performance throughout the mission.
- **Hardware Constraints:** The selected sampling rate was chosen to balance the capabilities of the avionics hardware (e.g., sensors, microcontrollers) with the computational resources of the telemetry software. The sensors (such as an altimeter, accelerometer, and gyroscope) can provide data up to **100 Hz**, but we have optimized the rate to **5 Hz** to avoid overloading the system while maintaining data accuracy.
- **Data Processing and Visualization:** The telemetry software, developed using Python, CustomTkinter, and Matplotlib, can process and display data in real-time at **5 Hz** without delays in the graphical user interface. This ensures smooth graph rendering and responsive data updates without significant latency.

Effects of Sampling Rate on Telemetry:

- **Lower Sampling Rate:** If the sampling rate is reduced to, for example, **1 Hz**, there may be a loss of detail in capturing fast-changing events, such as rapid changes in altitude or acceleration. This could potentially result in delayed or less accurate telemetry display.
- **Higher Sampling Rate:** A higher sampling rate, such as **20 Hz** or more, could lead to more precise data collection, but it might also increase the load on the system's processing capabilities, potentially causing lag in data visualization and a higher data throughput load.

3. Telemetry and Telecommand Details

Telemetry refers to the automatic transmission of data from the rocket's sensors to the ground control station. Data includes key flight parameters such as altitude, velocity, orientation, temperature, and battery levels, which are critical for monitoring the rocket's status.

Telemetry Data Transmission:

- The onboard sensors continuously send data at the configured sampling rate of **5 Hz** (five updates per second).
- Data is transmitted over a **radio frequency (RF) link** or other suitable communication systems to the ground station, where it is processed and displayed on the telemetry dashboard.
- Real-time data is visualized on a **graphical interface** for monitoring by the ground team, allowing for quick assessments of flight conditions.

Telecommand, on the other hand, allows ground operators to send control instructions back to the rocket. This is essential for mid-flight interventions, such as adjusting flight parameters, deploying parachutes, or triggering specific payload experiments. The software ensures smooth coordination between receiving telemetry data and executing telecommands to ensure real-time control during flight.

Telecommand Communication:

- The ground station sends commands over the same RF link used for telemetry or through a dedicated channel.
- Commands are processed in real-time by the rocket's onboard control system, which then executes the required actions.

4. Data Storage and Handling

Telemetry data from the rocket's sensors is stored locally during the flight and remotely for post-flight analysis. The storage system is designed to handle the high volume of real-time data generated by the rocket's avionics, ensuring that no critical information is lost, while also providing easy access for retrieval and visualization.

Local Data Logging:

- **Format:** CSV (Comma-Separated Values) format is used for local data logging. This format is lightweight, easy to read, and can be quickly parsed for post-flight analysis.
- **Purpose:** Local data logging acts as a backup to the real-time telemetry feed, ensuring that even if the live transmission is interrupted, a complete dataset can be retrieved after the flight.

Remote Data Storage:

Primary Database: MongoDB is used for remote data storage because of its scalability and ability to handle large amounts of data. It is particularly suited for real-time telemetry data storage due to its flexibility and support for concurrent data writes and reads.

Advantages of MongoDB:

- **High Throughput:** Handles large volumes of data from multiple sensors without performance degradation.
- **Flexible Schema:** The NoSQL structure of MongoDB allows for easy handling of complex telemetry data, including nested data structures from different sensor systems.
- **Scalability:** MongoDB's horizontal scalability makes it ideal for expanding the data system to accommodate additional sensor data and higher telemetry rates in future missions.

Backup Cloud Storage:

Firebase is used as a secondary backup solution. It provides a secure cloud-based storage system, ensuring that all critical telemetry data is backed up remotely in case of a local system failure.

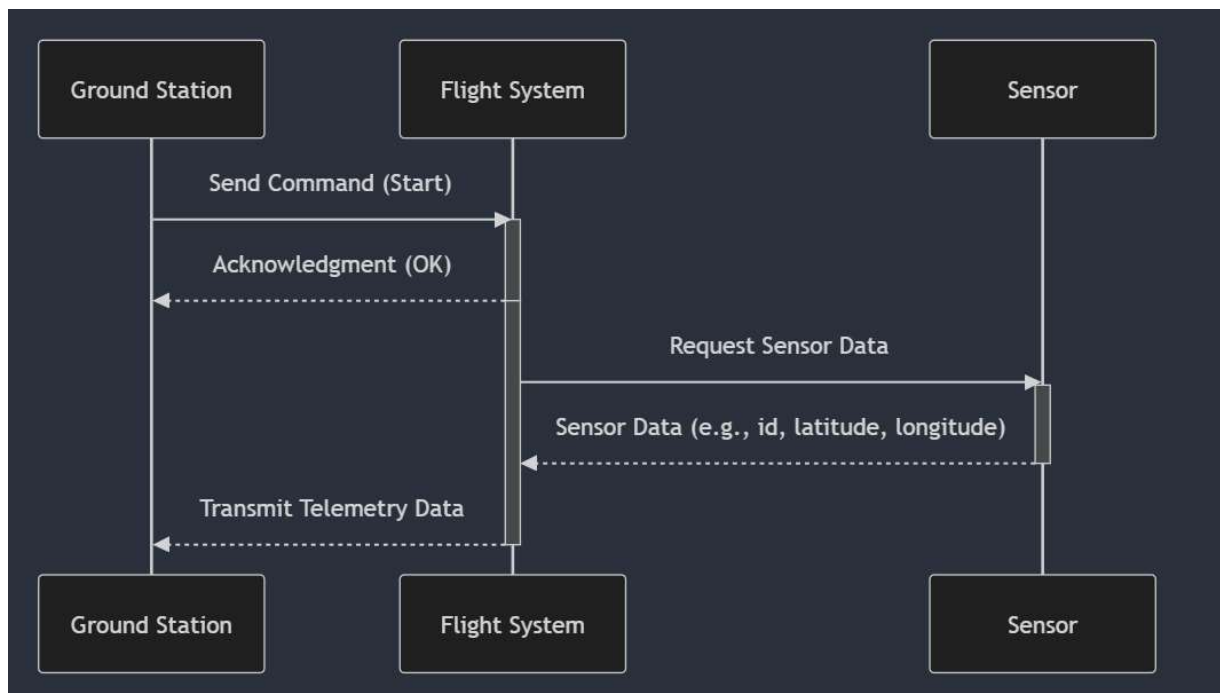
Limitations of Firebase:

- **Network Dependency:** Firebase's reliance on internet connectivity makes it less ideal for real-time operations, especially in situations where network conditions may be unstable or where rapid, low-latency data access is required.

- **Bandwidth:** Firebase's performance can degrade with high data transfer rates, making it better suited as a backup solution rather than the primary real-time database.

5. Software reset loop:

A key feature of the flight software is the **watchdog system**. This safety mechanism continuously monitors software performance. If any malfunction is detected (e.g., unresponsive sensors, memory overflows, or transmission errors), the watchdog triggers a software reset. This ensures that the system can recover autonomously and continue to operate with minimal downtime. The reset loop is designed to preserve essential flight data even after the reset, allowing the software to resume from the last stable state without loss of critical information.



6. **Simulation Mode Strategy**

Before flight, the flight software undergoes extensive testing in a simulation mode. This mode emulates real flight conditions by feeding virtual sensor data (generated by simulated physics engines) into the system. The simulated data mimics all aspects of a rocket launch, including sensor readings, telemetry transmission, and flight dynamics.

The simulation mode allows the development team to test and optimize:

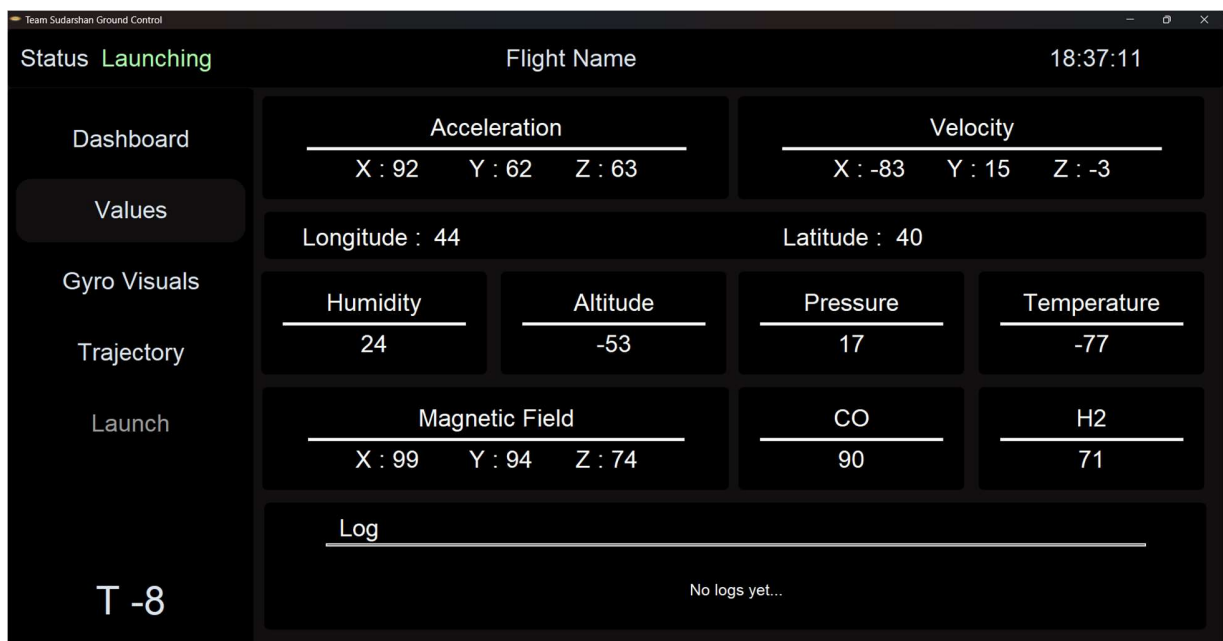
- Sensor data handling and processing,
- Telemetry transmission under different conditions,
- Command execution timing,
- Real-time data collection and visualization.

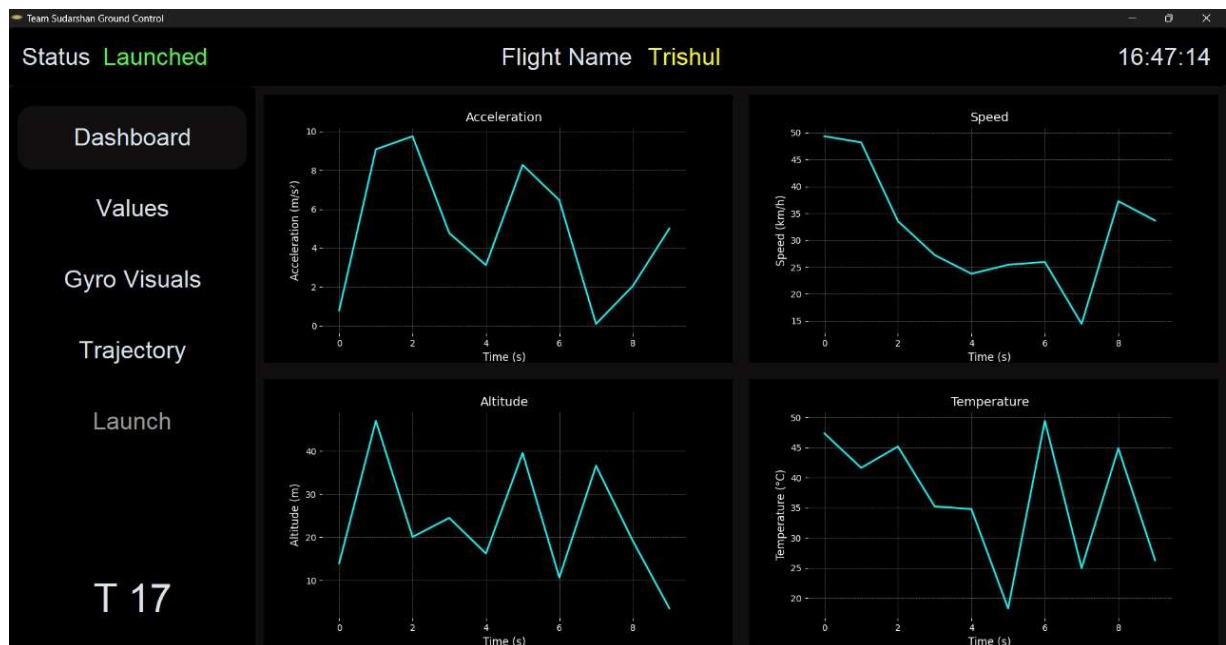
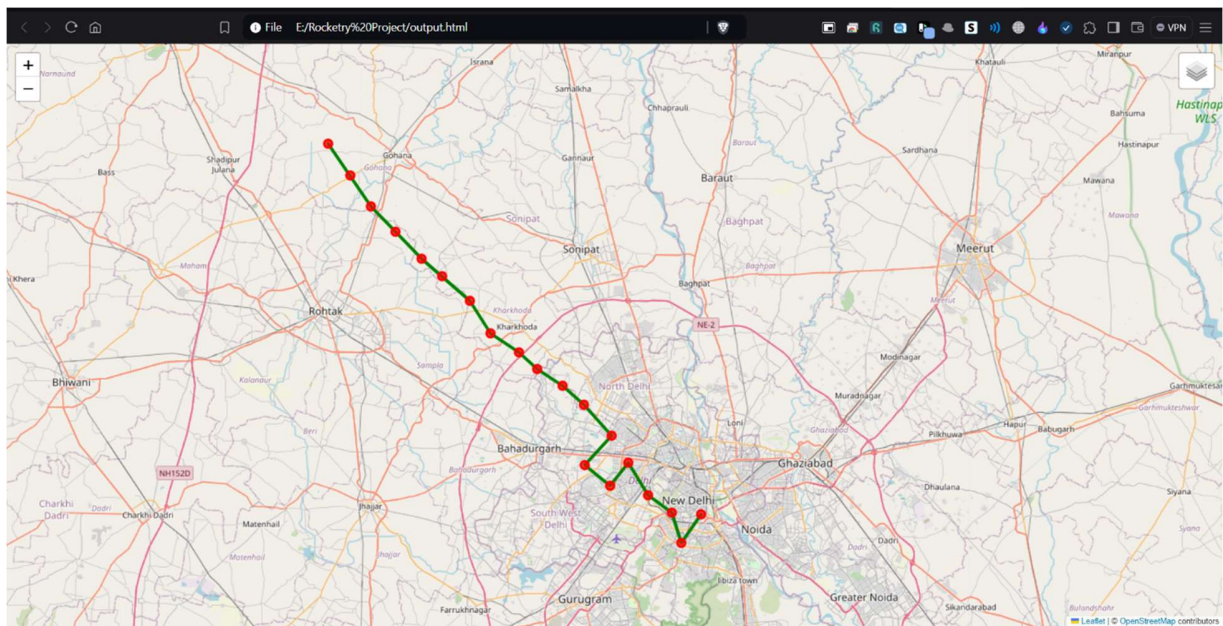
By using simulated data, the team can uncover and resolve bugs in a controlled environment without risking hardware or flight failures.

7. **Test Methodology**

To ensure reliability, the flight software is subjected to a rigorous multi-phase testing process:

- Unit testing: Each software module (e.g., telemetry transmission, data logging) is tested independently to verify that it functions as expected.
- Integration testing: Once individual modules are verified, they are combined, and tests are run to ensure smooth interaction between all components.
- System testing: Full-system tests are conducted in simulated real-time flight environments. This includes verifying sensor inputs, telemetry outputs, and telecommand execution.
- Field testing: After successful lab simulations, the software is tested in sub-scale prototype rockets. This allows the team to observe its behavior under real-world conditions and make necessary adjustments before full-scale deployment.





8. Proto Version Testing

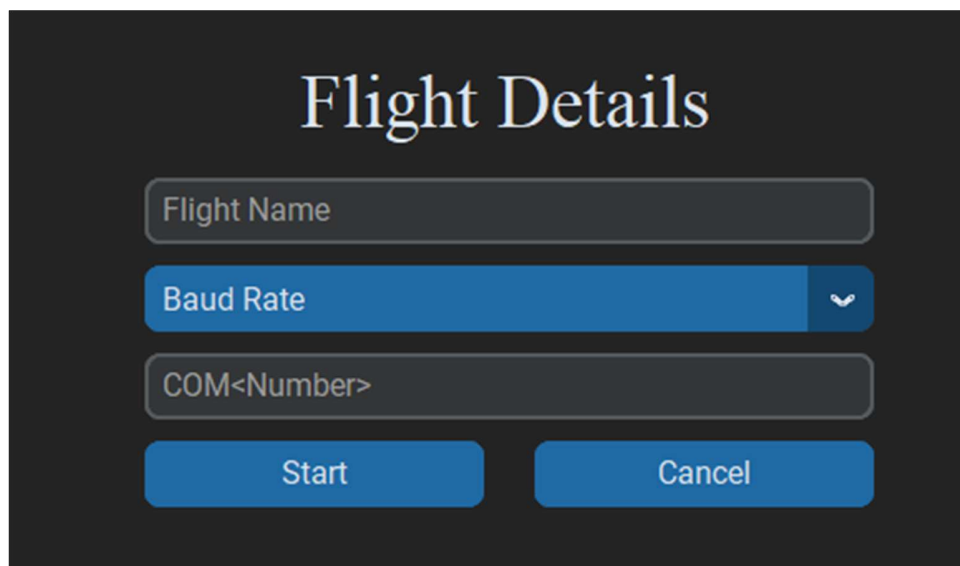
Early-stage testing of the software is conducted using smaller, sub-scale rocket models. These models replicate key features of the full-scale rocket but on a smaller scale, allowing the team to:

- Test telemetry and data transmission range,
- Verify the reliability of command execution in flight,
- Identify potential issues with sensor integration.

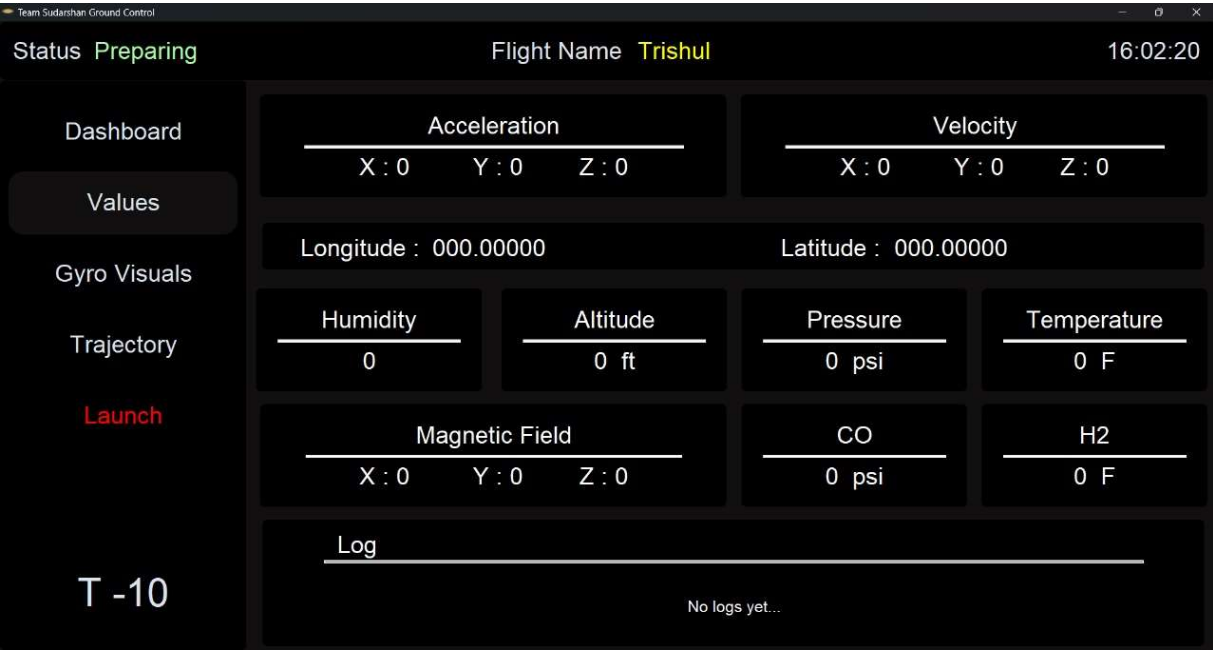
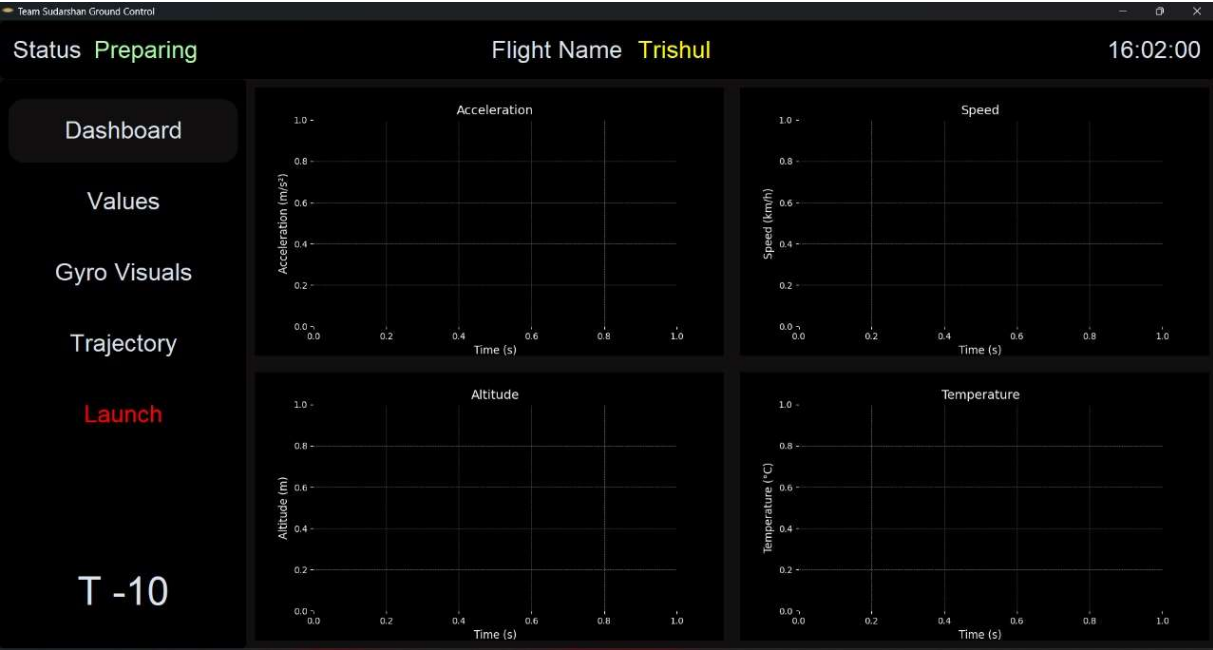
Proto version testing serves as a cost-effective way to identify and resolve any early-stage problems, ensuring that the final version of the software is robust and reliable.

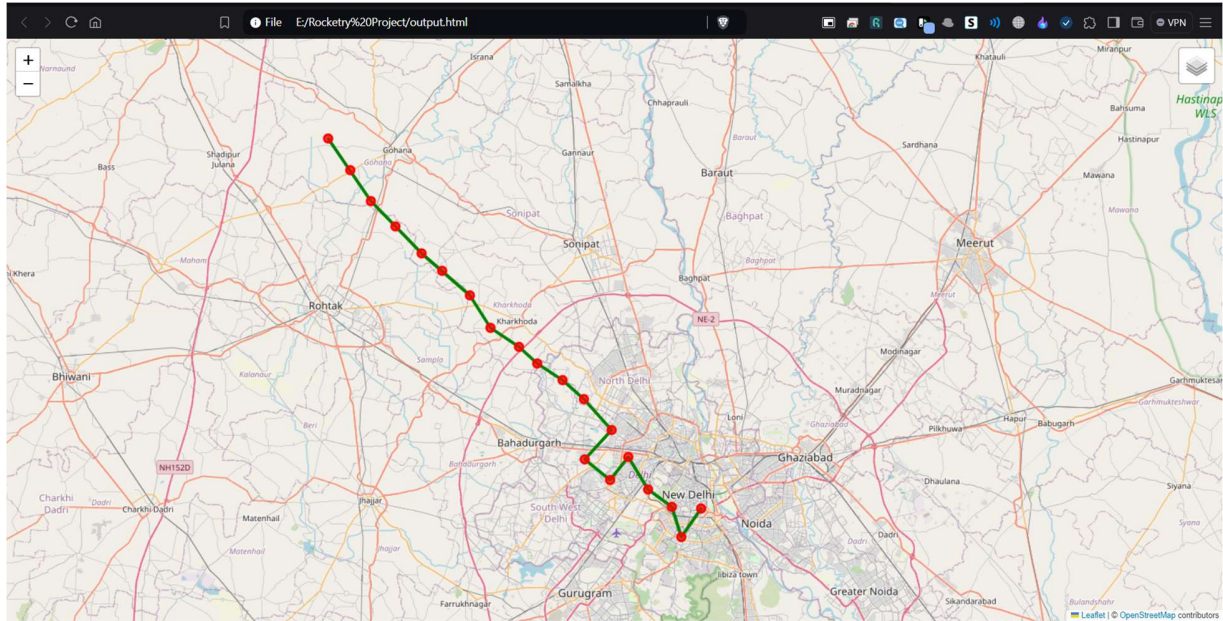
9. Telemetry Display

The telemetry data is displayed in real-time on the ground station using **CustomTkinter** for a graphical user interface. The visual components of the display are powered by **Matplotlib**, providing graphical representations of sensor metrics such as altitude, velocity, and system health. Data is fetched from **MongoDB**, which stores the telemetry data in real time, allowing for quick access and display during flight. **Firebase** acts as a secondary backup to log data, but due to its reliance on high-speed internet, MongoDB is prioritized for real-time display. The interface is designed to ensure smooth visualization of telemetry data without delays, even when processing large volumes of data.



The image shows a graphical user interface titled "Flight Details" on a dark background. It features three input fields: "Flight Name", "Baud Rate" (which is a dropdown menu with a blue background and a white chevron icon), and "COM<Number>". Below these fields are two buttons: "Start" and "Cancel", both with blue backgrounds and white text.





10. Command Software and Interface

The ground control station is equipped with a user-friendly command interface, built using Python and CustomTkinter, which allows operators to send telecommands to the rocket during flight. This interface presents critical commands such as emergency abort, parachute deployment, and flight mode changes in an accessible format, ensuring that operators can quickly react to unexpected situations. The interface is designed with safety in mind, preventing accidental command execution through confirmation prompts and fail-safes.

Key Telecommand Functions:

1. **Launch Command:** Initiates the countdown and ignition sequence of the rocket.
2. **Abort Command:** Triggers the immediate shutdown of the launch process or flight, if any anomaly is detected.
3. **Stage Separation Command:** Commands the separation of rocket stages (if applicable).
4. **Parachute Deployment:** Sends a command to deploy the recovery parachute at the appropriate altitude or in case of emergency.

5. **Cutoff Commands:** Can cut power to non-essential systems to preserve battery life during critical phases of flight.

Telecommand Communication:

- The ground station sends commands over the same RF link used for telemetry or through a dedicated channel.
- Commands are processed in real-time by the rocket's onboard control system, which then executes the required actions.

11. Real-time Data Collection Scheme

The flight computer collects real-time data from onboard sensors and transmits it to the ground control station using **XBee** wireless modules. XBee was selected because it offers reliable, low-power, and long-range communication, making it ideal for transmitting telemetry data during a rocket flight. Unlike Wi-Fi or Bluetooth, XBee is less prone to interference and signal loss over long distances, which is crucial for continuous data transmission during the rocket's ascent. The incoming data is received as a continuous string, which is then filtered to extract relevant telemetry information such as altitude, velocity, and system health. This data filtering is handled using **Python** for its robust text processing capabilities and ease of integration with other modules. The filtered data is displayed on the ground control system in real-time using **CustomTkinter** for the interface, as it allows for the creation of a dynamic and responsive GUI. Data packets are processed every 20 milliseconds to ensure accurate visualization of key flight metrics. **MongoDB** is used to log this data because of its capability to handle real-time queries efficiently without performance bottlenecks, which other databases like **MySQL** cannot handle as effectively when dealing with high-frequency queries. **Firebase** is not used for real-time telemetry display because it is cloud-based, requiring a strong and fast internet connection, which is not always feasible during flight. Instead, Firebase serves as a backup for storing data post-flight.

12. Last Data Command at Console Display

To prevent data loss due to communication interruptions, the system stores and displays the last successfully received command and telemetry data on the ground station console. This allows operators to assess the rocket's last known status and respond accordingly, even if real-time data transmission is temporarily lost. By retaining the last known command, the system provides operators with a reference point, ensuring continuity in decision-making.

13. User Interface for Real-time Data Collection

The **CustomTkinter**-based interface provides ground control operators with a clear, real-time display of critical flight parameters such as speed, altitude, orientation, temperature, and system health. This interface is designed to be user-friendly and responsive, ensuring that operators can easily monitor and control the flight in real-time.

The interface integrates **Matplotlib** for graphical representation of telemetry data, offering dynamic charts and graphs to display real-time changes in key metrics like velocity and altitude. Additionally, **Folium** is embedded within the interface to visualize the rocket's flight path on a map in real-time, providing spatial data to track the rocket's geographic position throughout the flight. This adds a valuable extra dimension to the monitoring process, helping operators ensure that the rocket is following the expected flight path.

- **MongoDB** is used for storing and retrieving telemetry data during the flight. It was chosen because it efficiently handles real-time data storage and high-frequency queries, making it ideal for managing the continuous flow of telemetry information. The data stored in MongoDB can be quickly accessed and displayed without delays.
- **Threading** is used to handle multiple processes simultaneously, such as receiving real-time telemetry data, updating the display, and processing operator commands. This ensures the system remains responsive even when handling multiple tasks at once, which is critical for real-time operations.

Why These Technologies Were Chosen:

- **CustomTkinter** was selected because of its ability to create modern, customizable graphical user interfaces (GUIs) in Python, making it ideal for building a real-time, interactive control panel.
- Alternatives like **PyQt** or **Kivy** were considered, but CustomTkinter was chosen for its simplicity, lightweight nature, and ease of integration with other Python libraries.
- **Matplotlib** is used for visualizing telemetry data in real-time. It allows for dynamic and customizable graphing, ensuring that operators have a clear, visual representation of key flight metrics.

- **Plotly** and **Bokeh** are other potential options for real-time graphing, but Matplotlib was chosen for its simplicity, strong community support, and ease of integration with Tkinter.
- **Folium** is used for mapping and tracking the rocket's flight path in real-time. It was selected for its interactive map capabilities and Python compatibility, allowing seamless integration into the CustomTkinter interface.
- Other mapping libraries like **Google Maps API** or **Leaflet.js** could be used, but Folium offers a Python-friendly environment that is easy to customize and lightweight.
- **MongoDB** was selected for telemetry data storage because of its scalability and ability to handle high-frequency queries efficiently. It allows for real-time data insertion and retrieval without the performance issues faced by relational databases like **MySQL**.
- **MySQL** was not used because it struggles with handling a large number of simultaneous queries, and **Firebase** was not selected because it relies on cloud connectivity, which may not be reliable during flight operations.
- **Threading** was used to handle multiple tasks simultaneously, ensuring that the interface remains responsive while managing real-time telemetry data updates, user inputs, and command execution.
- **Multiprocessing** was considered but would have added complexity in communication between processes. **Threading** is simpler to implement and more than sufficient for the real-time requirements of the system.

14. Ground System Transmission and Uploading of Data


The ground control system receives real-time telemetry data via **XBee** wireless modules. **XBee** was chosen for its long-range, reliable communication over other alternatives like **Wi-Fi** or **Bluetooth**, which are more prone to interference, shorter ranges, and limited bandwidth, making them unsuitable for transmitting rocket telemetry over long distances.

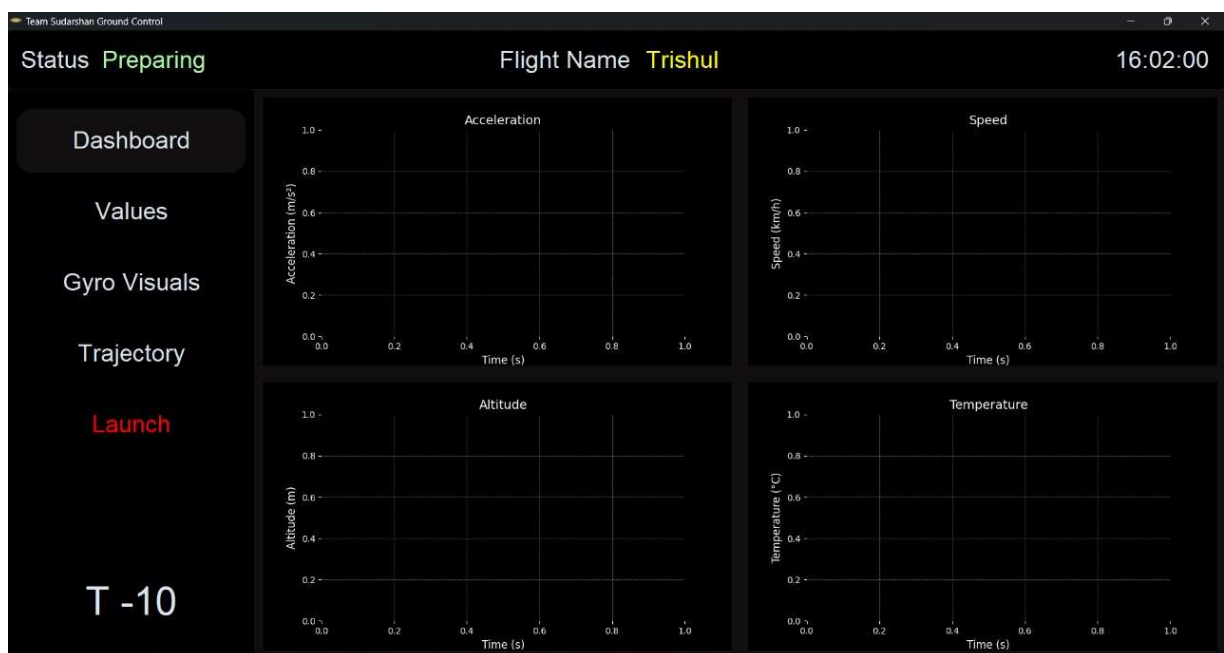
The data is stored locally in **MongoDB** during the flight, which was chosen for its ability to handle real-time queries and large amounts of telemetry data without performance issues. **MongoDB** allows for high-speed data insertion and retrieval, making it ideal for managing continuous telemetry streams during flight. **MySQL** was not used because it struggles with handling a high number of concurrent queries, leading to potential performance bottlenecks. **Firebase** was also not chosen because it is cloud-based and requires a fast, stable internet connection, which may not be available during the flight, especially in remote locations. **Firebase** would introduce latency in real-time operations, making it less reliable for immediate data storage and access.

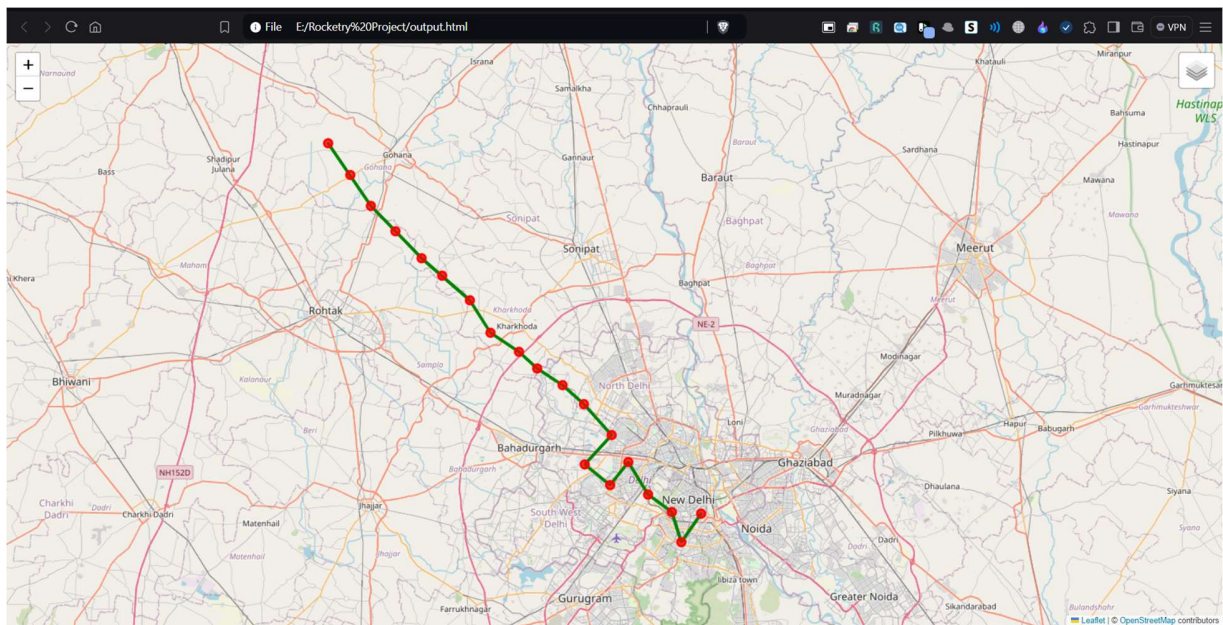
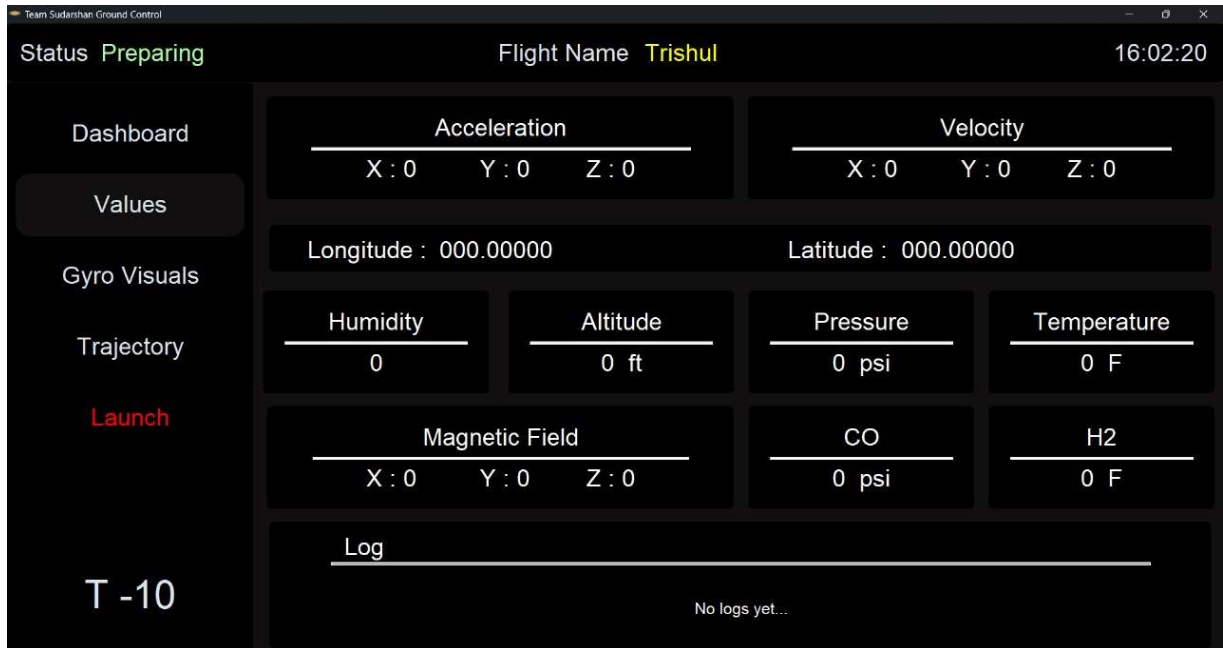
To prevent data loss during brief communication interruptions, telemetry data is logged locally in **CSV** format, ensuring that there is a fail-safe backup available on the ground.

station. This allows the ground control team to access and analyze telemetry data even if the primary data transmission is interrupted temporarily.

Flight Details







15. Sensor and Payload Telemetry

The flight software is integrated with an onboard sensor suite, providing data such as acceleration, temperature, pressure, and orientation. If the rocket is carrying experimental payloads, their data is transmitted alongside the rocket's core telemetry. The software manages multiple telemetry streams, ensuring that payload data is synchronized with the core flight data, allowing scientists and engineers to monitor the experiment's performance during the mission.

16. Libraries Used

- **Tkinter:** For creating graphical user interfaces (GUIs) in the ground control station interface, enabling real-time display of flight data and commands.
- **OS Module:** Used to interact with the operating system, primarily for managing file paths when saving or retrieving data logs.
- **Subprocess Module:** Allows the software to run external processes, such as executing supplementary Python scripts that handle data processing or simulation tasks.
- **Webbrowser Module:** Used to open URLs or local files in the default web browser, such as opening HTML files that contain map or telemetry information for analysis.
- **Folium:** A Python library for creating interactive maps, used to visualize the rocket's real-time location and flight path.
- **Threading:** To maintain multiple processes at one time, ensuring real-time data updates from sensors and smooth performance.
- **MongoDB:** For fast and efficient storage of telemetry data, chosen due to its ability to handle large numbers of queries simultaneously and provide high-performance real-time insertion and retrieval.
- **CSV:** For local data storage and logging, ensuring raw telemetry data is safely stored during flight.