DESIGN AND DEVELOPMENT OF DISTRIBUTED LOWER ATMOSPHERIC INSTRUMENTATION PACKAGES

BY

JULIO CÈSAR GUARDADO

A Thesis
Submitted to the Department of Physical Sciences
In partial fulfillment of the requirements
for the degree of
Master of Science in Engineering Physics

04/2022
Embry-Riddle Aeronautical University
Daytona Beach, Florida
DESIGN AND DEVELOPMENT OF DISTRIBUTED LOWER ATMOSPHERIC INSTRUMENTATION PACKAGES

by

Julio Cèsar Guardado

This thesis was prepared under the direction of the candidate’s Thesis Committee Chair, Dr. Aroh Barjatya, Professor of Engineering Physics, Daytona Beach Campus, and has been approved by the Thesis Committee. It was submitted to the Department of Physical Sciences in partial fulfillment of the requirements of the degree of Master of Science in Engineering Physics

THESIS COMMITTEE:

Dr. Aroh Barjatya,
Committee Chair

Dr. Robert Clayton,
Committee Member

Dr. Edwin Mierkiewicz,
Graduate Program Chair,
Engineering Physics

Dr. Karen Gaines,
Dean, College of Arts and Sciences

Dr. Charles Lee,
Committee Member

Dr. John Hughes,
Department Chair,
Physical Sciences

Dr. Christopher Grant,
Associate Provost of Academic Support
Abstract

The recent miniaturization of embedded electronics and widespread availability of low-cost sensors, microcontrollers and transceivers has enabled development of sensor suites that can be used for high temporal and spatial resolution measurements of atmospheric parameters such as temperature, pressure, humidity, particulate counts, wind velocity and much more. This thesis presents the development of both airborne and ground based distributed atmospheric instrumentation suites. First, we present a low-cost radiosonde platform capable of simultaneous multipoint launches or high cadence back-to-back continuous launches. This radiosonde platform can do wind measurements on an approximately 60 m scale up to a minimum of 25 km. It is capable of simultaneous communication on the ISM band at ranges up to 100 km with multiple payloads. Second presented is a high-altitude balloon (HAB) instrumented platform capable of controlled slow descent. With open communication ports, this HAB bus can be used with any instrumentation an end user wants to include. Novel methods of enabling both extended flight and controlled descent are also presented. And lastly, we present a design of a ground based IOT air quality monitoring system. This system provides information about particulate matter, carbon dioxide, and volatile organic compound content present in a given area. Information is uploaded to an open source platform connected to a local network over WiFi and saved to files by a Python 3.8 based data logger. All software development for each of these systems was completed in C++ using the Arduino IDE and in MATLAB. Circuit and PCB design was done using Autodesk Eagle.
Acknowledgments

I would first like to thank Dr. Aroh Barjayta, my advisor throughout this degree program. None of the work described in this thesis would be possible without his support and guidance.

I also want to thank Dr. Robert E. Clayton, who spent untold hours answering my questions and taking the time out of his days to teach me something new. I would not have learned as much as I did without his guidance and friendship.

I would also like to thank all of the people who have worked with me, both directly and indirectly, in SAIL. Anthony Oreo, Nathan Graves, Henry Valentine, and Rachel Conway made all of the time I spent in the lab enjoyable and helped me grow both professionally and socially. Special thanks to Jacob Gambrill, who helped me with development and getting balloons in the air.

I want to thank my mother, Hazel Salazar, and my sister, Hazel Guardado. They have always supported me in all of my ventures and I would not be the person I am today without them.

I also want to thank Greta Fergus, who has been my strength and motivation through a Master’s program and a global pandemic. She dealt with my frustrations and helped me see a path through all of the things I thought I couldn’t solve. I will also thank my cats, Odin and Chonkers, who kept me sane with their company and their mischief.
List of Acronyms

AQM Air Quality Monitor
COTS Commercial Off The Shelf
GFS Global Forecast System
GS Ground Station
GPS Global Positioning System
GUI Graphical User Interface
I2C Inter-Integrated Circuit
IOT Internet of Things
LiPo Lithium Polymer
LLA Latitude, Longitude, and Altitude
MURI Multi University Research Initiative
PWM Pulse Width Modulation
RSSI Received Signal Strength Indicator
SPI Serial Peripheral Interface
TVOC Total Volatile Organic Compound
# Contents

Abstract iv  
Acknowledgments v  
List of Acronyms vi  

1 Introduction and Background 1  
  1.1 The Lower Atmosphere ........................................ 1  
    1.1.1 Troposphere ............................................. 1  
    1.1.2 Stratosphere ............................................. 2  
  1.2 Atmospheric Instrumentation ............................... 3  
    1.2.1 Radiosondes ............................................. 4  
  1.3 Ground Based Atmospheric Instrumentation ............... 7  
    1.3.1 Internet of Things ..................................... 8  
    1.3.2 COTS IoT Instrumentation Packages ................... 8  

2 Low Cost GPS Enabled Radiosonde 11  
  2.1 Requirements .................................................. 12  
  2.2 GPSsonde Components and Instrumentation .................. 12  
  2.3 Payload Software .............................................. 14  
    2.3.1 Version One ............................................. 15  
    2.3.2 Version Two ............................................. 15  
    2.3.3 Version Three ........................................... 16  
  2.4 Telemetry System ............................................. 18  

vii
4.3 Instrumentation ................................................................. 62
  4.3.1 Temperature and Pressure ............................................. 63
  4.3.2 eCO\textsuperscript{2} and TVOC ........................................... 64
  4.3.3 Particulate Matter ....................................................... 65
  4.3.4 PCB Layout ............................................................... 66
4.4 Cost Breakdown ............................................................... 67
4.5 Sample Data ................................................................. 68

5 Summary and Future Work .................................................. 72
  5.1 GPSsonde .................................................................... 72
  5.2 HAB Bus and CDU .......................................................... 73
  5.3 IoT Instrumentation Package ............................................. 74

A GPSsonde Reference .......................................................... 78
  A.1 Flight Software ............................................................ 78
  A.2 Ground Station Software ............................................... 85

B HAB Bus and CDU .............................................................. 122
  B.1 Bus Flight Software ....................................................... 122
  B.2 CDU Flight Software ..................................................... 153
  B.3 Ground Station Software ............................................... 159
  B.4 PCB Schematics and Layouts ........................................... 224
    B.4.1 Bus .................................................................... 224
    B.4.2 CDU .................................................................. 227

C AQM Reference ................................................................. 230
  C.1 AQM Software ............................................................. 230
  C.2 Logging Script ............................................................. 246
  C.3 Mosquitto Configuration ................................................. 250
  C.4 PCB Schematic and Layout ............................................. 275
List of Tables

2.1 List of necessary components to build and fly a GPSsonde in the base configuration. ............................................................... 31
2.2 List of necessary components that only need to be purchased once for GPSsonde flights. ........................................................... 32
4.1 JSON formatted data for the AQM ........................................ 60
4.2 Description of negative health effects from varying concentrations of carbon dioxide [1]. ....................................................... 64
4.3 Breakdown of component cost for the AQM. ......................... 67
List of Figures

1.1 Structure of earth’s atmosphere [2] . . . . . . . . . . . . . . . . . . . 2
1.2 Sounding balloon post launch carrying an ERAU radiosonde. . . . . 3
1.3 Commonly used COTS radiosondes. . . . . . . . . . . . . . . . . . . . 5
1.4 Logo for the nationwide Eclipse Ballooning Project. . . . . . . . . 6
1.5 Graphic demonstrating the capabilities of IoT [3]. . . . . . . . . . . 7
1.6 NCD Environmental Air Quality Sensor. . . . . . . . . . . . . . . . . 8
1.7 PurpleAir Air Quality Monitor. . . . . . . . . . . . . . . . . . . . . . 9

2.1 GPSsonde payload without insulating foam . . . . . . . . . . . . . . . 11
2.2 Block Diagram of GPSsonde Payload . . . . . . . . . . . . . . . . . . 13
2.3 Process of assembly of a complete GPSsonde payload. . . . . . . . 14
2.4 Flowchart of version 1 of the GPSsonde System. . . . . . . . . . . . 15
2.5 Flowchart of version 2 of the GPSsonde System. . . . . . . . . . . . 16
2.6 Flowchart of version 3 of the GPSsonde System. . . . . . . . . . . . 17
2.7 Ground station anatennas for the GPSsonde . . . . . . . . . . . . . . 19
2.8 Block Diagram of the GPSsonde System. . . . . . . . . . . . . . . . . 20
2.9 MATLAB based ground station GUI for multiple GPSsonde payloads
   with actual flight data displayed. . . . . . . . . . . . . . . . . . . . . . . 23
2.10 Flight profile for test flight completed on 02/25/22. . . . . . . . . . . 25
2.11 Velocity data from test flight completed on 02/25/22. . . . . . . . . . 26
2.12 Velocity data from test flight completed on 04/04/22. . . . . . . . . 28
2.13 Velocity data from test flight completed on 04/04/22. . . . . . . . . 29
2.14 Comparison of RSSI between the omni-directional and cloverleaf an-
   tennas. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
Chapter 1

Introduction and Background

1.1 The Lower Atmosphere

This thesis will focus on developing technologies that assist in understanding the lower atmosphere rather than the atmosphere as a whole. The lower atmosphere for the purposes of this thesis will include the troposphere and the stratosphere. This includes a 50 km region above earth’s surface where a variety of weather related phenomena occur. This chapter will outline the importance of understanding this region as well as some existing methods for conducting lower atmospheric science.

1.1.1 Troposphere

The troposphere is the densest region of earth’s atmosphere and therefore where the majority of weather related events occur. As this is the lowest region of the atmosphere, it is the layer that has the greatest noticeable impact on daily life. Understanding the trends that occur in the troposphere aids in understanding human impacts on the environment as well as predicting future weather events. This region varies in height depending on latitude and time of year, reaching up to 20 km near the equator and 7 km at the poles during winter [2]. Temperature in the troposphere slowly decreases until the tropopause is reached, which is where the coldest atmospheric temperatures are observed. This is the boundary layer between the
1.1. THE LOWER ATMOSPHERE

stratosphere and the troposphere and is typically identified by the difference in air temperature on either side of this boundary. Typically, temperature increases with altitude due to absorption of solar UV. Therefore, the tropopause can be defined thermally by the lowest point where the lapse rate of temperature decreases to 2 K/km [4].

1.1.2 Stratosphere

The stratosphere is the region of earth’s atmospheric immediately above the troposphere. The stratosphere typically extends to an altitude of approximately 50 km, with the lower boundary changing depending on latitude and time of year. Due to lower temperatures, the stratosphere contains little moisture and therefore rarely contains clouds. Although, temperature does increase with altitude in the stratosphere, most of the moisture has precipitated by the time radiative heating makes an impact. The exception to this however are polar stratospheric clouds, which are found
1.2 Atmospheric Instrumentation

in the polar latitudes when temperatures reach below -78° C. These clouds have been observed to play a role in causing holes in the ozone layer by providing a location for specific chemical interactions to occur that result in the destruction of ozone [5]. Ozone is a molecule found within a band of the stratosphere, known as the ozone layer, at altitudes between 15 to 30 km. This molecule blocks ultraviolet radiation from reaching earth’s surface and more specifically, UVB radiation which has been linked to diseases such as cancer in humans [6].

1.2 Atmospheric Instrumentation

![Sounding balloon post launch carrying an ERAU radiosonde.](image)

Figure 1.2: Sounding balloon post launch carrying an ERAU radiosonde.

Sounding balloons, or weather balloons, are balloons used to carry instrumentation through the lower stages of earth’s atmosphere. These balloons are typically filled with helium or hydrogen and can reach altitudes of up to 40km with proper fill and sizing. Figure 1.2 shows a Kaymont 150g balloon, which is rated to reach altitudes of ~25 km. The typical use case for sounding balloons are for meteorological observations, although they have a long history of being used for understanding the fundamental nature of the atmosphere. French meteorologist Leon Teisserenc de Bort
used sounding balloons filled with hydrogen to identify and define the troposphere and stratosphere as well as characterize temperature trends throughout these regions in 1896 [7]. Since then, sounding balloons have been used in a wide variety of fields, from collecting weather data to carrying food to near space for the novelty.

Sounding balloons are typically filled with helium, although it is not the only gas that can be used. Helium is a by-product of radioactive decay from heavy elements such as uranium and thorium [8]. Due to this, there is not always a reliable supply available and, at the time of writing, there is a helium shortage impacting the US. Hydrogen is easy to manufacture and provides similar functionality as helium. However, as hydrogen is flammable, it introduces serious risks that are not present with an inert gas like helium. The most clear example of this is the Hindenburg disaster where an airship filled with hydrogen caught fire. Despite this, hydrogen is widely used for sounding balloon flights due to the lower cost and higher availability. All of the flights described in this thesis have used helium, both for safety and consistency with cooperating universities.

1.2.1 Radiosondes

Instrumentation packages carried by sounding balloons that use radio telemetry to report atmospheric data to the ground are called radiosondes. These packages typically measure in-situ temperature, pressure, and humidity, while reporting their current location via GPS through the entire range of a sounding balloon. This allows the radiosonde to indirectly measure wind velocities though differential calculation of the Latitude, Longitude, and Altitude (LLA) being down-linked. Every day, meteorological organizations launch nearly 1600 radiosondes around the globe to provide atmospheric data at two specific times. These "snapshots" occur at 00:00 and 12:00 GMT and collect global weather data that supplement weather prediction models [9]. These organizations use Commercial Off The Shelf (COTS) radiosondes produced, for example, by companies such as the Finnish Vaisala and the German Graw. Two popularly used radiosondes from these companies are shown in Figure 1.3.

Both of these sondes claim to be capable of telemetry of data at ranges greater
1.2. ATMOSPHERIC INSTRUMENTATION

(a) Vaisala RS-41 Radiosonde. (b) Graw DFM-17 Radiosonde.

Figure 1.3: Commonly used COTS radiosondes.

than 160km, with the Graw package claiming ranges over 300 km. Both packages carry instrumentation typical of radiosondes, although they come in varying versions with some instrumentation included or excluded, depending on the needs of the end user. For bulk purchase of radiosondes for large organizations such as the National Weather Service, a price of $\sim200$ per unit [10] can be expected. However, for smaller organizations who are not expecting to purchase large quantities, such as universities and citizen scientists, this price will increase to $\sim300-400$ per unit. This does not include the cost of a ground station, a balloon, or helium. Therefore, existing radiosondes are prohibitively expensive and limit the amount of organizations that can conduct this type of science. Addressing this price point will be one of the objectives in the development of a low-cost GPS enable radiosonde that will be described in Chapter 2.

Radiosondes can be used for a wide variety of applications, one such being the Eclipse Ballooning Project. This project is a nationwide collaboration of higher learning institutions and government organizations like NASA. The objectives of this project is to complete a sounding balloon launch campaign in the locations on the path of totality of a solar eclipse. In the team preparing for a 2024 solar eclipse, there are 70 teams participating, each with an engineering and atmospheric science team [11]. The engineering team typically handles work relating to a science payload
the flies on a train of payloads on a single balloon. The atmospheric science team, however, is in charge of flying two radiosondes. This team uses the Graw DFM-17 package described above to attempt to answer the following questions, among others [12]:

- Can eclipse-induced atmospheric gravity waves be definitively detected in data across all sites?
- How much time lag is there between the temperature minimum, and minimum in solar flux?
- How do boundary layer heights vary during an eclipse?

Although the DFM-17 radiosondes are equipped to collect the necessary data to answer these questions, the functionality of that system limits the temporal resolution of the measurements. The Graw radiosonde ground station is not capable of communication with several radiosondes concurrently, which requires the Eclipse Balloon team to operate two separate ground stations. This not only doubles the expenses associated with a radiosonde flight, but makes the flight campaign more difficult to operate. Therefore, Chapter 2 will also attempt to address the issue of concurrent flight of radiosondes.
1.3 Ground Based Atmospheric Instrumentation

Ground based instrumentation is necessary to monitor and predict trends in the lower atmosphere. These instruments focus on in-situ measurement of the troposphere and provide valuable data about the phenomena that affect daily life. In a joint program between the NWS, FAA, and DOD, ground based Automated Surface Observing Systems (ASOS) were developed to monitor temperature, pressure, wind velocities, and current weather conditions. These systems report data continuously, 24 hours a day and have archives maintained by the NWS [13]. Aside from government funded systems, commercially available weather monitoring stations are available, albeit at a high cost. These systems, although very functional, are not easily accessible and therefore limit the amount of localized data that can be collected by interested parties. However, several COTS packages enable user friendly transmission of data by using the Internet of Things (IoT) to enable monitoring of data in real time from any location that is WiFi enabled.

Figure 1.5: Graphic demonstrating the capabilities of IoT [3].
1.3. GROUND BASED ATMOSPHERIC INSTRUMENTATION

1.3.1 Internet of Things

The internet of things describes a cloud based platform for communication with so-called "smart" objects. These objects are internet enabled and contain embedded sensors that help in communicating the state of either the environment or the object itself [3]. For example, internet enabled light bulbs, "smart-bulbs", can be controlled remotely through voice assistants or smart phone applications. In industrial applications, manufacturing equipment can be integrated into an IoT network for monitoring of progress or potential failures. In medicine, wearable IoT devices can relay medical information to healthcare providers for more responsive tracking of a patients health. There are an infinite number of uses for IoT devices, some of which are shown in Figure 1.5, although this thesis will focus on the data collection capabilities inherent in the Internet of Things.

1.3.2 COTS IoT Instrumentation Packages

Several COTS instrumentation packages exists. One of these packages, the NCD Environmental Air Quality sensor, shown in Figure 1.6, is a package that collects
1.3. GROUND BASED ATMOSPHERIC INSTRUMENTATION

data on ambient temperature, pressure, humidity, and gas resistance. Using this information, it determines an air quality on a scale of 0-100 and transmits this data wirelessly. This module claims to be able to transmit data at ranges of over 2 miles using a 900 MHz mesh network, but does not include a method of receiving this data [14]. Therefore, this module is not a user friendly option for those who do not have a mesh network receiver or IoT network previously set up. The lack of features such as a live dashboard and IoT server at the price point of ∼$200 also decrease the usability of the system for local observations of air quality.

PurpleAir is a manufacturer of COTS air quality monitors that can be easily deployed and monitored. Figure 1.7 shows the PA-II air quality sensor, which transmits information about particulate matter density, temperature, pressure, and humidity to a web based server. This server can be accessed by any smart device and has an integrated dashboard for displaying real time data in contrast to the NCD package [15]. However, due to the sampling time of the two on-board particulate matter sensors, the data rate of the PA-II is limited to 1 complete set of data per 6 seconds. This
IoT package is more complete and feature rich than the NCD monitor, although both lack modularity and high data throughput. To address this gap in instrumentation, 4 will describe the development of a low cost, internet enabled instrumentation package capable of high cadence transmission of sensor data. This will be a complete system which includes a server system for monitoring and logging of incoming data.
Chapter 2

Low Cost GPS Enabled Radiosonde

Figure 2.1: GPSsonde payload without insulating foam
A radiosonde is a device carried by a sounding balloon that is capable of transmitting atmospheric measurements to a ground station via radio. These packages can contain a wide assortment of instrumentation and, with proper balloon selection, can reach altitudes of up to 30 km. This section will describe the design and implementation of a low cost radiosonde with on-board COTS GPS for in-situ wind speed measurements. This radiosonde will be referred to as the “GPSsonde” from this point forward.

2.1 Requirements

The GPSsonde will attempt to address some of the issues with currently existing radiosonde systems. Therefore, performance of this system will be expected to work as well as COTS systems while improving on their weaknesses. The following are the overarching requirements for the design of this radiosonde system:

1. Telemetry of data will be achieved at ranges greater than or equal to 100 km.
2. Using a 150g balloon, the GPSsonde will achieve altitudes greater than or equal to 25 km.
3. The GPSsonde ground station will be able to receive data from 6 payloads simultaneously and display data from each individual payload in real time.
4. The per payload cost will not exceed $150.

The above requirements are not an exhaustive list, but will drive specific design requirements that will be discussed in the sections below.

2.2 GPSsonde Components and Instrumentation

The central component of the GPSsonde is the Adafruit Feather M0 micro-controller. This package is powered by an ATSAMD21G18 ARM Cortex M0 processor and includes several useful features such as an in built RFM95 LoRa radio module and a
LiPo battery power management system. The Feather M0 contains 20 GPIO pins, hardware serial, I2C, and SPI ports, 8 PWM pins, 10 analog inputs, and 1 analog output. Although there is only one port for serial communication (excluding the USB serial line for programming), there can theoretically be 127 devices connected to the I2C port, and any amount of devices connected to the SPI port given enough chip select pins. The GPSsonde is therefore very scalable and can be adapted to include any instrumentation that is desired. A limitation of note is that the time on air for radio communication increases with the size of the packet necessary to down-link all sensor readings. With more instruments integrated, the frequency of transmissions will be decreased proportionally to packet size.

Currently, the only sensor installed on this platform is the Ublox M8N GPS module. This module is highly capable and configurable, which makes it a good choice for this platform. Many GPS receivers are pre-configured to operate only below specific altitude thresholds, making them unusable for high altitude balloon applications. Although this module is not initially an exception to this, it is one of few that can be configured to operate above the factory set altitude limit, given the acceleration it experiences is below 1G. It can also concurrently communicate with up to 3 Global
2.3 PAYLOAD SOFTWARE

Navigation Satellite Systems (GNSS) with a receiver sensitivity of -167 dbm [16]. Information about the latitude, longitude, and altitude (LLA) of the payload in-flight can give insight into the wind velocities present at any given altitude that the balloon travels through. The Global Forcast System (GFS) generates predictions for several atmospheric variables, including wind speeds, that can be used as reference values for validation of flight data. These models are typically used to predict flight trajectories for high altitude balloons by software developed by Cambridge University Spaceflight (CUSF). A second point of validation for wind speeds derived from flight data can be obtained by running balloon flight predictions based on these derived values.

2.3 Payload Software

Although the hardware configuration has remained the same since the initial design, the software has gone through several versions. All versions of the flight code were developed in the Arduino v1.8.10 IDE in C++. The Radiohead software library was used throughout all versions of the code. This library provides drivers for several common radio modules, including the RFM95, and allows for simple transmission and reception of packetized data. To communicate with the Ublox M8N GPS module, two different driver libraries were considered: a modified version of the TinyGPS++
2.3. PAYLOAD SOFTWARE

2.3.1 Version One

The first iteration was designed as a test bed for the maximum range of the RFM95 communication system and to determine the feasibility of the system as a whole. This first version employed the Ublox GNSS software library to poll the GPS module for new data using the proprietary binary ubx protocol. This library was modified to retrieve east, north, and vertical velocity recorded by the GPS, along with the default LLA and UTC time. This version saw several challenges in-flight that will be discussed in Section 2.5, but was eventually successful in proving the feasibility of the system. This served as the foundation for the following versions of the code.

2.3.2 Version Two

The second iteration of the flight software was developed to allow simultaneous flight of multiple payloads. Because this system was designed with the capability of tracking multiple payloads in mind, it is crucial that payloads do not transmit freely, as they
are all operating on the same frequency. Therefore, the ground station polls any active payloads for data at specific intervals. This is achieved by maintaining a list of payload IDs input by the user that can be modified at any time using the add and remove payload buttons shown in 2.9. The payload is entirely inactive until it receives the proper command from the ground station. This command takes the form of the packet header, followed by the payload ID of the desired payload, followed again by the packet header. This removes the possibility of any payload identifying a data transmission as a GS command and removes the possibility of interference between payload transmissions. This version also saw a reduction in packet size, as the GPS velocities were no longer being transmitted.

2.3.3 Version Three

Due to limitations of two-way communication, this iteration shifted to time synchronized communication. By using GPS UTC time, the transmissions of each payload
2.3. PAYLOAD SOFTWARE

Figure 2.6: Flowchart of version 3 of the GPSsonde System.

can be completed sequentially according the the current UTC second. As each payload has a good time fix for the vast majority of the flight, interference free communication is made possible. In order for this time synchronization to work, the packet size of each transmission must be minimized. Otherwise, the time between each transmission will increase due to the slow bit rate. Specific bytes, such as battery voltage and GPS status, were removed. Previous flights have shown that battery is not a concern over the typical duration of GPSsonde flights and the GPS number of satellites byte conveys similar information as the status. This removes two bytes, and an additional two bytes can be removed by combining the packet header, payload ID, and GPS number of satellites into two bytes. This is done by generating a 9-bit packet header that is the same across all payloads, a 3-bit payload ID for each payload, and allocating 4 bits to the number of satellites being received. This results in a 16-bit, or 2 byte, value that contains three relevant data points: the packet header in the 9 most significant bits, the payload ID in the 3-bits following, and the number of satellites in the 4 least significant bits. With these modifications, the packet size is
2.4. TELEMETRY SYSTEM

reduced to 16 bytes, which takes slightly over 1 second to transmit with a 183 bps data rate. Therefore, communication between all payloads (limited to 6 concurrent payloads) occurs at 2 second intervals, with each payload transmitting a packet every 12 seconds. This rate can be changed by decreasing the amount of payloads in the air, i.e. 3 payloads = 6 second interval per payload. Each payload decides when to communicate according to the following logic:

\[
\text{utcSec} \mod (\text{NUM\_PAYLOADS} \times 2) = \text{payloadID} \times 2
\]

When this condition is true, the system then transmits the latest GPS information and resets the newData flag to 0. This prevents the system from transmitting constantly during the UTC second that makes the above true.

2.4 Telemetry System

All of the data collected in-flight is down-linked to a ground station capable of storing and displaying data in real time. Although the inclusion of GPS makes the possibility of recovery nonzero, there is currently no on-board storage of data. Therefore, the development of a robust system for RF communication is necessary. Throughout the development cycle, several modifications to the entire telemetry system were made. This section will describe the required components for accurate retrieval of flight data on the ground as well as the various versions of flight software developed.

2.4.1 RF Communication

The RF communication is entirely handled by two Semtech SX1276 based RFM95 LoRa transceivers. This module is programmed to operate on the sub-gigahertz US902-928 MHz unlicensed frequency band and functions using chirp spread spectrum technology. This system encodes individual bits into multiple chirps that encompass the entire frequency spectrum. These chirps are frequency modulated sinusoidal symbols that span a predefined bandwidth and are therefore highly resistant to interference and Doppler shift effects [19].
The bit-rate ($R_b$) for LoRa radios is highly configurable and can be set by modifying three parameters. The first, Spreading Factor (SF), describes the ratio between the nominal symbol rate and the chirp rate. This translates to the number of chirps transmitted per second and directly impacts the number of chirps necessary to transmit one bit. The next configurable parameter is the bandwidth of each chirp. An increase in bandwidth allows for a [19]. $R_b$ can be expressed as:

$$R_b = SF \times \frac{4 + CR}{2^SF} \times 1000$$

where SF represents the Spreading Factor, CR represents the Code Rate, and BW represents the bandwidth in kHz. In order to maximize range, the bit rate must be minimized. Therefore, the values chosen for this application are a SF of 12, a BW of
125kHz, and a CR of 4/8. This combination results in a bit-rate of 183 bps, which, when combined with a 2 dB transmitting antenna and a 2.7 dBi receiving antenna, results in an estimated maximum range of transmission being $\sim 100$ km. In the base configuration, i.e. no added instrumentation besides a GPS, the packet size of each transmission is 22 bytes. With this configured bit rate, each transmission will be complete in $\sim 0.95$ seconds. This packet size can be decreased by manipulating byte values into smaller formats and removing specific data from the structure. However, a transmission rate of 1 Hz is more than enough for characterization of larger scale wind patterns in the lower atmosphere.

2.4.2 Ground Station

MATLAB AppDesigner was used to develop a Graphical User Interface (GUI) capable of receiving incoming data, saving it to a file, and parsing it for real time display.
Using the RFM95 LoRa breakout module, incoming data is streamed to the GUI where storage and processing of the data occurs. All of the incoming data is saved to a binary file in the exact format it was received in. This allows for processing of all of the flight data post-flight without the need for multiple files. Scripts for processing of flight data were written in MATLAB and can be seen in Appendix A. Several versions of this ground station were developed to accommodate changing payload software. However, the below is applicable to all versions.

### General Functionality

In order to parse this data in real time, several methods for validation of packet integrity are carried out. The first of which determines the length of data received on the serial port connected to the radio. If this is the greater than the expected length of a GPSsonde transmission, the GUI begins parsing searching for a two byte packet header consisting of a generalized hexadecimal byte of 0x27 and a unique identifier for each payload. These unique identifiers take the form of repeated numbers in hexadecimal such as 0x11, 0x22, 0x33, etc., up to 0xFF. This allows for simultaneous transmission of up to 16 different payloads. With these two bytes, there is a $1 in 2^{32}$ chance that mis-identification of a packet header within the data will occur. Once the packet head is validated, a final check is conducted to ensure that the incoming data after the packet header is the expected length for a packet. Only after these checks will parsing and displaying begin.

Parsing of the payload data involves reconstructing sensor data from the incoming binary stream. The parsed data is only used for real time display during a flight and is not saved. Each payload receives its own tab for data display, with the GPS position of all active payloads displayed on the same 2D and 3D plots. The current LLA, GPS fix quality, UTC time, and number of satellites, calculated instantaneous velocities, and housekeeping information are displayed in text boxes and updated when new data is available from each payload. Although retrieval of velocities from the GPS module is possible, velocities are calculated based on point to point comparison of LLA values that are separated in time to shorten the length of transmissions. These calculated values are relatively close to GPS generated values and will be discussed
2.4. **TELEMETRY SYSTEM**

in section 2.3. This ground station software is also capable of recreating a flight by reading a previously created file. In this mode, it periodically reads a file and treats this data stream as if it were a serial stream coming from a radio. Therefore, parsing and operation of the GUI is the same as the standard mode.

**Version Specific Changes**

The initial version of the ground station was designed to test the performance of the RF communication system. Due to this, there were two separate sets of GS hardware running the same software. One copy was connected simply to a 915 MHz omni-direction antenna. This 2.7 dBi gain antenna was intended to be the final choice for the GPSsonde GS assuming that the desired communication range could be achieved. The second copy of the GS was operating with the same 17 dB yagi antenna that was discussed in Chapter 2. This version of the GS therefore also included the same pointing system described in Chapter 2 and converted the payload’s LLA to an azimuth and elevation for the antenna. This system is known to be functional at ranges up to 300 km and served as a reference point for RSSI values.

The second version of the GS was designed to account for bi-directional communication and multiple payloads transmitting on the same frequency. To avoid interference, this version actively commanded each payload in the air to transmit. For a payload to be commanded, the user is required to input it’s ID number after pressing the “Add Payload” button seen in Figure 2.9. As active commanding at this frequency is not allowed and the success of the omni-directional antenna, the tracking system and yagi antenna were removed from the configuration. This version, although functional, had severe limitations to command the payload, a specific sequence of 3 bytes was sent that, when parse by a payload, tells which ID to respond. This bi-directional communication slowed the data rate significantly, as a payload would have to hear the 3 byte command, and reply with a 22 byte packet. However, with the 8 byte preamble that is baked into LoRa communication, the whole transmission is a total of 41 bytes. At a communication rate of 183 bps, one complete transmission takes approximately 2 seconds. In addition, the software library used on this version of the payloads is somewhat unreliable in the amount of time data acquisition takes,
2.4. TELEMETRY SYSTEM

Figure 2.9: MATLAB based ground station GUI for multiple GPSsonde payloads with actual flight data displayed.

which extends the total transmission time anywhere from 3 to 4 seconds. With the processing time on both the GS and payload, the maximum rate of transmission between the GS and payload is approximately 6 seconds for one payload. With multiple payloads in the air, the data rate for a single payload increases by 6x, where x is the number of active payloads. Regardless of this limitation, a test flight was conducted to test bidirectional communication. This flight was unsuccessful and prompted the development of a third GS version.

The third and final version of the GPSsonde GS eliminated two way communication in favor of a payload driven synced communication scheme. Because this is a passive system, the mechanics of the GS are very similar to the first version with modifications to allow for multiple payload tracking. This GS parses all incoming data and automatically detects and displays active IDs. Due to the lessons learned
from the second test launch, a secondary ground station was operated during the third flight. This secondary GS was using the same hardware, but had a modified antenna. This antenna was the same 2 dB cloverleaf antenna that is flown on the GPSsonde payloads. A ground plane was also added to this antenna at a 1/4 wavelength distance to the radiating elements to improve the radiation pattern. This GS was again used as a point of comparison for RSSI and data loss for the third test flight.

2.5 Performance and Sample Flight Data

To create a functional version one of the GPSsonde system, several test flights were completed. These flights varied in scope, but were all designed to push the limits of the GPSsonde performance. This section will outline the procedure and performance for each version of the system that was flown.

2.5.1 Version One

Version 1 (V1) of the GPSsonde was tested by flying a single payload with two ground stations collecting data from it. This version was designed to test the maximum range of reception with the 2.7 dBi omni-directional antenna. Several flights of this version were completed, with varying degrees of success and minor modifications being made in between each flight. The first two flights were flown using Kaymont 600g balloons with an intended burst altitude of approximately 32 km and an ascent rate of 5 m/s. These flights performed nominally up until an altitude of \( \sim 18 \) km and a range of \( \sim 30 \) km, where GPS communications cut out. This was initially thought to be due to GPS configuration issues, as some GPS modules are factory programmed to stop operation above certain altitude limits and the range was significantly below the expected maximum range. However, through the use of the TestEquity temperature chamber, temperature was determined to be the cause of loss of communication. Packets were still being received, but GPS data was not being updated, which points to thermal shutdown of the GPS module. This was solved by increasing the amount
of insulation and the inclusion of a small polymide heating pad. This heating pad was wired in parallel to the rest of the GPSsonde electronics and was therefore always on when the payload was on. To account for this, the 400 mAh battery that was originally considered was replaced with a 1200 mAh battery to ensure longevity in flight. This can likely be re-evaluated in future iterations for a slight decrease in cost, as the 1200 mAh batteries are \( \sim \$4 \) more expensive than the 400 mAh. The size of the balloons for all flights moving forward was decreased to Kaymont 150g balloons to conserve helium and further reduce prices at the cost of maximum attainable altitude. With these changes, the V1 GPSsonde payload had a successful flight. The trajectory and sample velocity data can be seen in Figure 2.10 and 2.11 respectively. This flight had an apogee of 26.4 km and maintained communication up to a range of 92 km, which is exactly what the desired and expected performance was.

![Range and Altitude vs Time](image)

**Figure 2.10:** Flight profile for test flight completed on 02/25/22.
2.5. PERFORMANCE AND SAMPLE FLIGHT DATA

Figure 2.11: Velocity data from test flight completed on 02/25/22.
2.5. PERFORMANCE AND SAMPLE FLIGHT DATA

2.5.2 Version Two

Version 2 (V2) of the GPSsonde expanded on the success of V1 by adapting the software to account for multiple payloads. This version had a single test flight, with three payloads being actively commanded to transmit at specified intervals. The first payload (ID 0x11) was performing as expected for the initial portion of the flight, at a rate of 1 packet per 6 seconds. However, the winds carried the payload further south than expected and thus, line of sight and therefore communication was lost for a period. Movement of the antenna fixed this issue and payload 2 (ID 0x22) was turned on. Prior to launch of payload 2, approximately 20 minutes after payload 1, data reception from both payloads was confirmed. With two payloads active, the data rate for each payload dropped to 1 packet per 12 seconds due to the issues discussed in Section 3.3.2. After the release of payload 2, communication with payload 1 was intermittent, with several timeouts occurring on the GS. This further decreased the communication rate for all payloads, since the GS will attempt communication twice before moving onto other payloads. Despite this, payload 3 (ID 0x33) was proceeded to be released in a similar fashion to payload 2. At this point, payload 1 was at a range of approximately 15 km, where data transmission stopped. RSSI measurements from this payload showed that the last packet received had an RSSI of -118 dB, which is well above the minimum strength from reception. Payload 2 and payload 3 eventually followed this same pattern with similar ranges and similar RSSI values. This launch was therefore considered a failure, with the likely cause being reception of GS commands on the payload side. Although the 2.7 dBi omni antenna was excellent for reception, it was hypothesized that it did not perform as expected and therefore the payloads were not commanded to transmit. Despite this failure, this test provided insight into the multiple balloon launch procedure and was useful for operational knowledge.

2.5.3 Version Three

In response to the failures experienced with V2, version 3 (V3) departed from bi-directional communication and relied on syncing transmissions to GPS time. This
eliminated the possibility of missed GS commands and allowed the payloads to freely transmit regardless of range. This launch followed a similar procedure as V2, with payload being released at approximately 20 minute intervals. Each payload was turned on $\sim 5$ minutes prior to launch for verification of incoming data. This test flight was entirely successful and even exceeded expectations for range and altitude. Payload 2 was slightly under-filled, resulting in a slower ascent rate overall and an apogee of 27.2 km. Aside from this, performance of each payload was exactly as expected, with the data collected showing similar results. Figure 2.12 and Figure 2.13 show the flight profiles and recorded velocities for all three payloads respectively.

![Altitude vs Time](image1)

(a) Alt vs UTC Time for all 3 payloads

![Range vs Time](image2)

(b) Alt vs UTC Time for all 3 payloads

Figure 2.12: Velocity data from test flight completed on 04/04/22.
2.5. PERFORMANCE AND SAMPLE FLIGHT DATA

Figure 2.13: Velocity data from test flight completed on 04/04/22.

(a) Alt vs East Velocity

(b) Alt vs North Velocity

(c) Alt vs Up Velocity
This test flight also served to compare the performance of two different antenna configurations. The first configuration is the default 2.7 dBi omni-directional antenna that was used previously. The second antenna is the 2 dB 4 leafed cloverleaf antenna. These antenna have different radiation patterns and should therefore perform differently. Figure 2.14 shows the RSSI for each payload plotted against range. This comparison clearly shows that the cloverleaf antenna maintained higher RSSI throughout the whole flight than the omni antenna. For future flights, this antenna configuration should be used to maximize range.

(a) RSSI vs Range for the omni antenna.  (b) RSSI vs Range for the cloverleaf antenna.

Figure 2.14: Comparison of RSSI between the omni-directional and cloverleaf antennas.
2.6 Cost Breakdown

2.6.1 Per Payload Costs

The cost associated with each flight of a GPSsonde is relatively low when compared to commercially available packages. The associated cost of each necessary component of the GPSsonde can be seen in Table 2.1. Several of these components, such as rigging line and plastic tubes to contain the electronics, were purchased in bulk. This results in a very low cost for these materials per flight. One material that can wildly vary is helium. Because these flights are lightweight and use small (150g) balloons, they need a minimal amount of helium. However, at the time of writing in Florida, helium is very expensive. In order to achieve the desired ascent rate of 5 m/s and maximum altitude of 26000 m, 15 m$^3$ of helium is required. This is over 10% of the cost of the total payload, which is a price that can vary depending on the supply and demand of helium nationwide.

Table 2.1: List of necessary components to build and fly a GPSsonde in the base configuration.

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Feather M0 w/ RMF95 LoRa</td>
<td>1</td>
<td>$34.95</td>
</tr>
<tr>
<td>UFL SMT Connector</td>
<td>1</td>
<td>$0.62</td>
</tr>
<tr>
<td>Adafruit 1200 mAh LiPo Battery</td>
<td>1</td>
<td>$6.95</td>
</tr>
<tr>
<td>uBlox NEO-M8N GPS Module</td>
<td>1</td>
<td>$29.99</td>
</tr>
<tr>
<td>12V 7W Flexible Polyimide Heater</td>
<td>1</td>
<td>$4.75</td>
</tr>
<tr>
<td>UFL to RP-SMA Cable</td>
<td>1</td>
<td>$3.64</td>
</tr>
<tr>
<td>JST 2 Pin Connector Pair</td>
<td>1</td>
<td>$0.30/pair</td>
</tr>
<tr>
<td>Cloverleaf Antenna</td>
<td>1</td>
<td>$1.27</td>
</tr>
<tr>
<td>Kaymont HAB-150</td>
<td>1</td>
<td>$18.00</td>
</tr>
<tr>
<td>10ft of 0.36, 50 lb Fishing Line</td>
<td>1</td>
<td>$0.02/yd</td>
</tr>
<tr>
<td>2” OD x 3” Plastic Pipe w/ Cap ($4/ft)</td>
<td>1</td>
<td>$1.00</td>
</tr>
<tr>
<td>2” ID x 3” Pipe Insulation($5/ft)</td>
<td>1</td>
<td>$1.25</td>
</tr>
<tr>
<td>15 ft3 Helium Tank (FL prices, 2/2022)</td>
<td>1</td>
<td>$15</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td><strong>$120</strong></td>
</tr>
</tbody>
</table>
2.6.2 One Time Costs

All of the one time costs associated with this system are the necessary components of the GPSsonde ground station. The list of general items can be seen in Table 2.1. This list excludes certain items which depend on application, such as mounting hardware for the antenna. This cost could be reduced by employing an RFM95 breakout module and a lower cost micro-controller than the Feather M0. However, this is the simplest configuration and allows for bulk purchase of materials that can be used for either GPS sondes or ground stations.

Table 2.2: List of necessary components that only need to be purchased once for GPSsonde flights.

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>900 MHz 2.7 dbi Omnidirectional Antenna</td>
<td>1</td>
<td>$60.00</td>
</tr>
<tr>
<td>6.5 ft SMA Male to Female Cable</td>
<td>1</td>
<td>$12.99</td>
</tr>
<tr>
<td>Adafruit Feather M0 w/ RMF95 LoRa</td>
<td>1</td>
<td>$34.95</td>
</tr>
<tr>
<td>UFL SMT Connector</td>
<td>1</td>
<td>$0.62</td>
</tr>
</tbody>
</table>

Total: ~$108
Chapter 3

High Altitude Balloon Bus & Controlled Descent

Figure 3.1: Assembled HAB bus
High Altitude Balloons (HABs) have a wide variety of applications and can be used to carry instrumentation up to the middle stratosphere. Several commercial platforms exist for in-situ sensing for properties such as temperature, pressure, and wind speeds. One such package is the Vaisala RS41 radiosonde. This radiosonde is an integrated package that allows for measurement of high altitude weather. However, specific applications lack an integrated bus that provides telemetry and flight control for an arbitrary instrument.

A specific use case for sounding balloons is the measurement of small scale turbulence within the stratosphere. This is a property that must be understood to enable hyper-sonic flight through the middle stratosphere. The AFOSR have funded a project to provide statistically significant understanding of stratospheric turbulence through a Multidisciplinary University Research Initiative (MURI). This project is titled “Integrated Measurement and Modeling Characterization of Stratospheric Turbulence” and is operated by the University of Colorado Boulder (CU-B), University of Minnesota (UMN), and Embry-Riddle Aeronautical University (ERAU). This research initiative is aimed at answering the following questions [20]:

- What are the spatiotemporal statistics of small-scale turbulence in the middle and upper stratosphere, and to what extent are they dictated by larger-scale motions, primarily gravity waves (GWs) that arise from meteorological sources at lower altitudes?
- What are the distributions of particles in the stratosphere, and their dependence on underlying meteorology?
- What are the relative roles of particles and pre-existing atmospheric turbulence (“freestream turbulence”) for the laminar-turbulent transition at hypersonic speeds in the middle and upper stratosphere?
- What are the effects of particles, temperature “sheets” and small-scale turbulence in the middle and upper stratosphere on long-range optical propagation, and how can these effects be accurately represented in computational simulations?
3.1 Previous Work

There are a variety of methods to achieve an answer for all of the above methods. However, sounding balloon flights are particularly well suited to providing relevant data for the first two science objectives. As this project requires specific instrumentation, namely turbulence and particulate matter instruments, there is no commercially available package that can be used to obtain this information. Therefore, a generic, multipurpose bus to accommodate this instrumentation is necessary. The rest of this section will outline the previous work done for the MURI project as well as the development of a modular platform for high altitude balloon payloads.

3.1 Previous Work

Much of the groundwork for developing a bus aimed at satisfying the MURI project requirements was done by a previous Masters student working with the Space and Atmospheric Instrumentation Lab, Noemi Miguelez. Figure 3.2 shows an image of
3.1. PREVIOUS WORK

The state the HAB bus was in at the end of Noemi’s thesis. The primary focus of this design was to implement a low-loss, high data rate communication system capable of reaching altitudes of 30+ km. These goals were achieved, with several flights being completed with >100 kbps communication rates up to altitudes >30 km. However, this system did not contain instrumentation aside from thermistors and a 9-DOF inertial measurement unit. Therefore, more work was necessary before MURI instrumentation was integrated into this bus.

Due to flight requirements from the turbulence instrument developed by CU-B, a system for enabling controlled descent of the payload was necessary. This instrument required a stable descent of \( \sim 3 \) m/s, and so a separate payload was designed in parallel to facilitate this. This Controlled Descent Unit (CDU) went through several iterations in the span of Noemi’s work, but was inherited as a valve system once the work for this thesis was begun. Figure 3.3 shows a completed version of Noemi’s CDU, made from a plastic tube and machined delrin valve. This system was capable of achieving controlled descent for flights that achieved greater than 30 km in altitude with a 3000g balloon. However, several features lacked complete functionality and so improvements were made that will be described in the following sections.

In order to enable low loss, long range communications, a highly capable ground station system was also developed by Noemi. For reception of the bus over long distances, a high gain Yagi style antenna was connected to a radio matching the
3.1. PREVIOUS WORK

Figure 3.4: Ground station GUI for bus designed by Noemi.

parameters of the bus. Due to the small field of view of this antenna, a pointing system was developed that saw the Yagi antenna mounted on a Yeasu Az-El rotor system. Pointing of the antenna and reception of the data was controlled by laptop running a custom GUI developed in MATLAB. This GUI parsed and displayed incoming data in real time while converting GPS LLA into an azimuth and elevation for antenna pointing. Figures 3.4 and 3.5 show a version of the ground station GUI alongside a diagram describing the various components necessary for the antenna pointing system. The pointing system was highly effective, and was left largely untouched for the work done in this thesis. However, as new instrumentation and functionality was included, the ground station GUI went through several iterations that will be discussed in sections below.
3.2 MURI Integration

The first step in development of a generalized bus was continuing the work done by Noemi in developing a bus for MURI instrumentation. This involved updating the control board to include breakouts for different communication lines. Both the CU-B and UMN instruments transmitted their data over UART and were connected to open Serial ports on the Teensy 3.5. The power system was also updated to include surface mount components that allow for better heat dissipation for the voltage regulators. Since the board was now acting as a heat sink for the voltage regulators, thermal issues that were seen in Noemi’s version of the bus were eliminated with the added bonus of in-built thermal maintenance of control electronics. Figure 3.6 shows an updated bus with integration MURI instrumentation. This version also included upgraded communication systems with the CDU that is based on the Xbee3 short range radio module.
3.3 HAB GENERIC BUS

To accommodate various forms of instrumentation, a bus capable of integration of instruments and telemetering relevant data must be developed. To achieve this, a PCB based around the Teensy 3.5 was designed. The Teensy 3.5 provides a wide assortment of communication ports, making it ideal for a generalized instrumentation bus. This central board breakouts communication lines, just as serial and SPI ports, to allow for simple interfacing with exterior instrumentation. It also provides housekeeping and general environmental information, such as temperature, pressure, and GPS location. The rest of this section will detail the communication and software required to enable this system to achieve it’s desired flight performance.
3.3. HAB GENERIC BUS

3.3.1 Telemetry

The telemetry subsystem is built around the Xbee SX Pro radio module. This module is a 1 W transceiver capable of communication at rates of 110 kbps at the 915 MHz frequency. Communication is done in transparent mode, meaning there is no preamble information and no two way communication between a payload and a ground station. This is due to a limitation of the ground station design and the use of a yagi type antenna, which will be discussed below. The XBee Pro SX module is directly soldered onto the control board and connects to a 2 dB cloverleaf antenna. It communicates with the Teensy 3.5 at a Baud rate of 230400. All data is packetized and sent in packets of up to 256 bytes. The data structure depends on the instrumentation integrated into the bus, but each packet will always contain a header, a packet number, a timestamp, and all relevant housekeeping information.

In order to enable long range communication, the ground station features a 17 dB yagi style antenna connected to a matching XBee Pro SX module. Due to the high gain of this antenna and the location near an airport, these antennas are required to be passive listeners. This excludes the possibility of using the XBee API mode, which has in built checking for packet delivery and provides useful RSSI information. Regardless of this, high throughput communication has been achieved using this system at ranges up to 300 km. This antenna is mounted onto a Yaesu G5500 rotor system that tracks the payload in flight to maintain strong reception.

3.3.2 Software

HAB Bus

The bus must be modular enough to accommodate various instrument packages while enabling high throughout communication with the ground station. This software is based on a header file system that contains various enables for different subsystems and instruments. For example, if a controlled descent system, which will be discussed below, is enabled, the relevant flag on the header file will be set high. This will enable the required functionality in the bulk of the code without user intervention. Due to the nature of some instrumentation, this system may not be as easily modified as the
inclusion of a supplementary system. For example, the turbulence instrument used in the MURI project has strict restrictions on the intervals where radio communication is allowed. This involves manipulation of the lower level code that is not included in the header file. However, as more and more instrumentation is used on these flights, a library of supported instruments can be maintained to allow future projects to use similar functionality.

To enable low loss of data, functionality for re-transmission and replaying of data was included. For each packet that is collected, a default of 4 repeating transmissions are completed. This reduces the amount of missed data by quadrupling the chances of reception. The header file system allows this number to be changed, although an increase could result in long time on air for the radio and reduce the battery life of the system. The replay functionality allows the radio to scan through the data stored on-board and re-broadcast the whole flight once a given altitude threshold is reached. For example, the science window for MURI is altitudes above 20km. Therefore, the relevant flight data can be replayed from storage rather than transmission of new data that may not be useful to the mission. Replaying of data will occur cyclically, with the transmissions restarting from the top of the science window once the end is reached.

All of the above functionality was written into its own parallel process, enabling multi-threaded operation of the Teensy 3.5. This allows for more control in the timing associated with the payload functions and makes integration of more instrumentation as simple as starting a new thread. Figure 3.7 shows a flowchart for this software. Each of the orange boxes encompasses a different parallel process, with threads for updating instrumentation, flight control, and telemetry control.
Figure 3.7: Flow chart of the parallelized software for the HAB bus.
Figure 3.8: Updated ground station GUI to accommodate MURI instrumentation.

**Ground Station**

The ground station design for this system is similar in functionality to the GPSsonde GS. They are both built in MATLAB, although this version is designed with higher communication rates in mind. For this reason, display of data is limited in rate to prevent overloading of MATLAB memory. In addition, the GS contains functionality for tracking the HAB in flight. This is done by converting current GPS LLA from the HAB to azimuth and elevation angles for the rotor system. In addition, this GS must account for the repeated transmissions received from the payload. Due to the reception of the same packet multiple times, parsing is limited to the first instance of a packet that is received. This also changes the behavior of the packet counting system from a basic addition system once a header is identified to something more complex.
3.4 Methods for Controlled Descent

In order to achieve a flight profile required by turbulence instrumentation, several methods for controlled descent were developed. The two methods of interest are the double balloon configuration and a valve configuration. The double balloon configuration involves using a primary ascent balloon alongside a smaller, under-filled balloon to simultaneously reach the target altitude and slow the descent of the payload, as shown in figure 1. By doing this, the maximum altitude achievable is dependent on the burst altitude of the primary balloon. This method is expensive, as the balloon cost is nearly doubled and more helium is necessary. There is also uncertainty in the max apogee, depending on the properties of the particular balloon that is used. There is also added complication in the fill calculation for the descent balloon. This must be done by estimating the amount of fill necessary to ascend a lighter payload than the one being flown. For example, the descent balloon for a 1 kg payload may be filled to accommodate a 600g payload at a rate of 5 m/s. This results in the weight of the real payload overcoming the upwards force of the balloon and descending. This method is viable, but in the interest of creating a more reliable, lower-cost system, the valve controlled descent method was pursued.

The valve method of controlled descent relies on a singular balloon with an actively controlled valve to slowly release helium. This method decreases the cost in helium and balloons, but adds in the complexity of a new system for control of the valve. This Controlled Descent Unit (CDU) must be capable of communicating with the primary payload and oscillating a valve depending on the pre-programmed control law. This system was tested on several balloons of varying size, with success seen on balloons up to 3000g. Therefore, these balloons became the default and each version of the CDU was designed with it in mind. The rest of this section will outline the various versions of the valve that were designed and tested.
3.4. METHODS FOR CONTROLLED DESCENT

3.4.1 Version 1

Assembly

Version 1 (V1) of the CDU was a prototype 3D printed valve that aided in proving the feasibility of the design. This valve was designed using Dassault Systems CATIA v5 and printed using the Raise3D V2 Plus 3D printer. It was operated by an Arduino Uno connected to a 9g micro servo, a Xbee3 radio module, and a Ublox M8N GPS module. In addition, it used 12 W polylime heating pads to maintain a high enough operating temperature for all of the electronics. A PCB shield was designed to make connection of all electronics simple. This was all housed in a Commercial Off the Shelf (COTS) Styrofoam box. Figure 3.9 shows both the open valve and interior of an assembled V1 Valve CDU. The 3D printed valve assembly consisted of several components, including a small hinge, an O-ring, and a threaded hook. The valve is printed into two separate pieces, the tube and the cap, with both pieces including a slot for the O-ring. The hinge connects both parts and allows for opening and closing of the valve in a smooth motion. The threaded hook is inserted opposite the hinge and is used as a tethering point for the string that holds the valve in the closed position. This string is attached to the micro servo, which, when rotated, opens and closes the valve. To ensure an airtight seal, low temperature valve grease is spread onto the O-ring prior to closing the valve. This tube was also designed to fit directly onto the neck of the ascent balloon and is secured by several zip ties. These tubes can be printed to fit with any balloon neck ranging from 3 to 5 cm diameter. This version of the CDU also had the capability of cutting away the payload in the event that the flight needs to be terminated prematurely.

Software

As this version of the CDU included a GPS module, the valve control was done entirely on board the CDU. Although the CDU was capable of communication with the main payload, on-board control was chosen to eliminate the possibility of communication drop out that was seen on some previous flights. This was achieved by monitoring both altitude and a calculated ascent rate. Once the balloon achieved a defined altitude
3.4. METHODS FOR CONTROLLED DESCENT

3.4.2 Version 2

Assembly

Version 2 (V2) of the CDU moved away from a 3D printed tube and incorporated COTS solutions. This valve design used commercially available extruded PETG tube with a diameter size for a 3000g Kaymont balloon. As this tube was purchased with matching caps, holes were cut in the caps to both allow airflow and provide a lip for a 3D printer stopper to rest on. This stopper was held firmly in place by using a thin pliable metal rod that was also attached to a servo motor. A notable improvement to this valve from V1 is also the orientation of the valve. V1 was designed as an “external” valve, with the internal pressure pushing against the action of the valve,
rather than aiding in the seal. V2 was designed as an “internal” valve, meaning the 3D printed stopper is pressed against the cap by both the retention rod and the internal helium pressure, as seen in Figure 3.10. This eliminates the need for valve grease and decreases the possibility of a leak. Another improvement is the transition from an Arduino Uno to a Teensy 3.5. The driving factor for this upgrade is the behavior of the servo. As the servo chosen is an analog servo, a PWM pin on the Uno was used as the signal line for the servo. However, the PWM driver on the Uno is unreliable and susceptible to noise. This limited options for the use of functionality such as software serial, which emulated a hardware serial port on PWM enable lines. While using this feature to communicate with either the GPS or radio module on the Uno, jittering was observed on the servo. The Teensy 3.5 has both multiple serial ports compared to the Uno’s one and a far more reliable PWM driver. The jittering issue was resolved with the added bonus of hardware communication lines to the GPS and radio modules.

Figure 3.10: Bottom view of the interior version of the CDU valve.
3.5. **TOP MOUNTED VALVE**

Software

The software for V2 of the CDU is fairly similar to V1, with improvements to the modularity of the code. As the whole HAB bus is designed to be used with arbitrary instrumentation, this version was written as a header file controlled code. This means that all relevant definitions and control law variables are placed in a header file, which minimizes the amount of base code changes that needs to be made for a change to the overall system. For example if a smaller balloon is being used, the maximum altitude where the valve system opens for descent can be changed in the header file, rather than in the bulk of the code. Another option in the header file can also disable control of the valve behavior on-board the CDU, allowing the payload to chose when to open. In this case, the CDU does not need to have a GPS module, which reduces both weight, size, and cost of the CDU.

3.5 **Top Mounted Valve**

With the success of the valve CDU, an alternative method was proposed that would further increase the precision with which the CDU can control ascent rate. In the normal configuration, the valve is inserted into the filler neck of the balloon. Because of the natural buoyancy of helium, the restoring force of the balloon is the only thing pushing helium out. This method proposes to place the CDU at the top of the balloon by cutting a hole and sealing it with the valve. This would allow the helium to freely leak from the valve under its own buoyancy and entirely removes the requirement of restoring force for CDU operation. Although effects from restoring force will still be experienced in the top valve configuration, helium can be more accurately released in a shorter period of time with this method. Furthermore, the maximum altitude achievable by this system is also extended. Typically, a balloon will rise and expand until the latex fails and bursts. Although the burst is avoided by using a CDU, the latex still deteriorates to the point where the restoring force is minimal. To account for this, the maximum altitude must be limited to ensure there is enough force left in the balloon to properly descend. In the top valve configuration, the balloon can
3.5. **TOP MOUNTED VALVE**

The TM-CDU is a miniaturized version of the standard valve configuration. As this version rests on top of the balloon, the total mass of the CDU must be minimized to decrease the load on the balloon. To achieve this, the Teensy 3.5 was replaced by a Seeeduino Xiao. This is a small form factor microcontroller with similar functionality to the Teensy 3.5. However, the Xiao has only one serial communication port, which excludes the inclusion of both a GPS and a radio module. Because the diagnostic information returned by the CDU is useful for troubleshooting, a radio module was

Figure 3.11: Bottom view of the interior version of the CDU valve.

ascend to higher altitudes without the concern that it will not fully descend. The rest of this section will outline the design choices and issues encountered with the top-mounted CDU (TM-CDU).

### 3.5.1 Assembly and Installation

The TM-CDU is a miniaturized version of the standard valve configuration. As this version rests on top of the balloon, the total mass of the CDU must be minimized to decrease the load on the balloon. To achieve this, the Teensy 3.5 was replaced by a Seeeduino Xiao. This is a small form factor microcontroller with similar functionality to the Teensy 3.5. However, the Xiao has only one serial communication port, which excludes the inclusion of both a GPS and a radio module. Because the diagnostic information returned by the CDU is useful for troubleshooting, a radio module was
3.5. **TOP MOUNTED VALVE**

prioritized. Therefore, all of the hardware on-board the TM-CDU is limited to the Xiao, an Xbee3, a microservo, and a 1000 mAh battery. This is all assembled onto the tube itself and wrapped in pipe insulation for thermal protection. Similar to the GPSsonde, these CDUs also have polymide heating pads wired in parallel to the CDU electronics for temperature stability. A reduction in tube length along with the above modifications results in a total mass of approximately 120g ± 20 depending on the amount of insulation used.

Installation onto the balloon must be done carefully to maintain the integrity of the latex. To install the TM-CDU, a circular incision must be made at the top of the balloon. This is done by laying out the balloon and cutting a bunched up section of latex. It is crucial that this cut is not linear, as the latex will continue to tear along the path of a linear cut with any amount of tension. Once a hole is made, the latex around the cut must be rolled over an elastic band, i.e. a section of balloon neck that has been cut off. This strengthens the cut section of latex and makes it easier to attach the CDU onto the balloon. Figure 3.12 shows these two steps for clarity. The elastic is stretched over the bottom part of the CDU and secured with a zip tie. The joint between the CDU and balloon is further strengthened by filling the seam with RTV silicone. This serves two purposes: ensure the CDU does not fall off of the balloon in flight, and create an airtight seal. This removes a point of failure and leaves the valve as the only place where helium can leak.

![Figure 3.12: Intermediate steps for installation of top valve.](image)

(a) Balloon top pre-cut.  
(b) Balloon hole cuffed over elastic.
3.5.2 Software

The software of the TM-CDU is functionally the same as the valve CDU. However, due to the limited form factor, there is no option for on-board control of the flight profile. This means that the communication between the main payload and the CDU must be reliable. In the event of a communication dropout, the flight software for the CDU has several fail safes to ensure a safe flight. One such fail safe can use the hardware reset line on the Xbee3 module to attempt to reestablish connection. This will occur on both the payload and CDU given enough time without contact. One variance in the TM-CDU is the inclusion of the MS5611. This feature is entirely optional, but can enable pressure based control of the CDU in the future. Due to this data, the data packet between the CDU and payload is larger by 4 bytes. As there is no limitation on data size or rate for the CDU, this increase does not cause any issues.

3.6 Flight Performance

The HAB bus described in 3.3 has seen success in flight with various combinations of bus and CDU. The data shown below will be from flights with the HAB bus flown with a standard CDU configuration as well as with the TM-CDU. The performance of each CDU will be examined in relation to one another, and the HAB bus will be examined through the lens of reliability of data transmission.

3.6.1 Bus Performance

The bus configuration for the data shown below is the most basic configuration; there is no instrumentation on-board beyond a simple camera module. These flights were intended to test the capabilities of the system and therefore were not seen fit to carry MURI instrumentation. However, the functionality of the system is the same with the one modification being transmission times. Due to requirements of RF silence during MURI instrumentation sampling, data down-linking was only allowed in a small window every 1.7 seconds. As these payloads did not have this limitation,
3.6. FLIGHT PERFORMANCE

Figure 3.13: Image of balloon down-linked from flight on 10/28/21.

transmissions were done freely with the camera timing being the only driving factor. Each image that was down-linked was taken at a small resolution of 320x240. This image when compressed to a JPEG, ranges from 8 to 12 KB in size. Due to the re-transmission system, a single image down-link is a total of 36 KB in size, at worst case. At a rate of 110 Kb/s, these transmissions take approximately 3 seconds to complete. To allow enough time for both transmissions and cool down of the radio module, new images were taken every 5 seconds. Figure 3.13 shows an image taken of the balloon mid-flight.

The performance of the bus was generally what was expected, with minor unexpected behaviors showing up later in the flight. The most noticeable variation from expectation is the control board resetting mid-flight. Figure 3.14 shows the packet number plotted against time. During this flight, the payload experienced three resets, as seen by the cycling of packet numbers. The most likely cause of this issue is temperature related shutdown of the control board. As the payload reaches higher altitudes, only radiative transfer of heat becomes available. With the radio module operating at a high data rate, a lot of heat is generated by the radio as well as the voltage regulator powering it. This may result in a loss of communication with certain components and activation of the watchdog timer kept by the Teensy 3.5. However, this issue is minimal and only affects the parsing of data post-launch. Throughout this flight, only 298 of 88970 packets were lost, which translates to 0.33% loss. This
3.6. FLIGHT PERFORMANCE

Figure 3.14: Plot of generated packet number vs time from flight on 10/28/21.

The flight discussed in section 3.6.1 was flown using the standard valve CDU configuration. This flight had a maximum apogee of 33.19 km and a maximum slant range of 253 km. The flight profile, shown in Figure 3.15, clearly demonstrates a controlled descent and validates the functionality of the system. A secondary point of validation is shown in Figure 3.16, which highlights the size of the balloon on a portion of the ascent and descent. As the balloon loses helium due to the valve operation, it begins to reduce in size. However, due to the stretching experienced in-flight, it never returns to its original shape. A similar profile has been replicated on multiple flights which indicates that this CDU is reliable and the valve is fit for the task of controlling helium flow out of the balloon. With this validation, the top valve system was attempted several times.

The initial flight of the top valve system was connected to a bus similar to the
3.6. **FLIGHT PERFORMANCE**

Figure 3.15: Altitude profile from flight on 10/28/21.

flight discussed previously. This system saw a communication dropout between the CDU and main payload. Therefore, this flight was considered a failure. To avoid this issue, temperature testing was done using a TestEquity temperature chamber to ensure the CDU would survive the low temperatures experience in flight. This test showed that the insulation installed on the CDU would be insufficient to maintain an operable temperature in flight. To solve this, a heating pad was wired in parallel to the main CDU electronics to constantly heat the required components. This led to the second launch of the TM-CDU system.

The second attempt at flying the TM-CDU also resulted in failure for a different reason than the first. This flight saw the balloon slowly leak helium, as evidenced by a slowly decreasing ascent rate. The maximum altitude attained by this flight was approximately 3 km, and was ruled a failure. However, as the balloon began its descent, the control law began to take effect. During its short descent, the CDU maintained a stable descent rate according to the pre-programmed parameters set in the header file. Although the flight was a failure, it still resulted in some validation of the CDU performance.

The third flight of the TM-CDU was also a failure. The inclusion of the heating
3.6. FLIGHT PERFORMANCE

Figure 3.16: Comparison of HAB size on different portions of a flight.

pad was not sufficiently tested and resulted in damage done to the plastic tube portion of the valve. This heating pad has a resistance of approximately 20 ohms, which, when connected to a 7.4v LiPo battery, produces enough power to reach the glass transition temperature of the PETG tube. Due to this, the structural integrity of the plastic was compromised and the internal pressure from the helium was able to create a hole in the tube. Due to the nature of this failure, no useful information about the performance of the CDU was obtained from this flight.
Chapter 4

Ground Based Distributed IoT Instrumentation Packages

Figure 4.1: Assembled AQM system.
Commercially available IoT instrumentation packages are both expensive and lacking in functionality. This chapter will overview the development of an instrumentation package capable of high data rate communication to a IoT server that can store incoming data for future use. The Distributed IoT Instrumentation Package (DIIP) system described below included instrumentation aimed at characterizing air quality, although it was designed to function with any instrumentation an end user deems appropriate. Understanding the effects of poor air quality on both human and animal populations is vital to the continued success of large metropolitan areas. In order to understand these effects, one must first quantify what “poor” air quality actually means. To satisfy this need, a low cost indoor/outdoor air quality monitoring station was developed on top of the DIIP to determine what the content of particulate matter and various chemical compounds is across a wide area. This package will be referred to as the Air Quality Monitor (AQM).

4.1 Objectives and Block Diagram

The following outline the overarching objectives for the DIIP system:

- Be able to connect to wireless internet services.
- Be able to collect and record high cadence data at 1 Hz or higher rates
- Be able to interface with a wide variety of COTS instruments

The above were achieved by developing the system described by the block diagram in Figure 4.2. This block diagram contains both the main components of the DIIP (Teensy 3.5 MCU, ESP8266 WiFi module, and Ublox M8N GPS module) as well as instrumentation that defines an AQM. Each of these components will be described in detail below along side a description of the data handling system for the DIIP.
4.1. OBJECTIVES AND BLOCK DIAGRAM

Figure 4.2: Block diagram for the AQM system.
4.2 Data Transmission and Logging

4.2.1 The ESP-01 Wifi Module

To enable the DIIP to transmit data without the need for a physical ground station located near the sensors, this system uses the ESP-01 WiFi module to upload data to the web. This module is an SOC that features an integrated TCP/IP networking chip, the ESP8266, and allows either the inbuilt or external MCU to connect to the internet. The integrated MCU is not powerful enough to operate all of the instruments in the base configuration, and therefore, integration with the Teensy 3.5 is necessary. Conveniently, this module is pre-programmed with an AT command based firmware that allows external MCUs to access it’s features. In this case, serial communication is used to command the ESP-01 to connect to a predefined network and to direct data to the desired IP address.

4.2.2 Data Transmission and MQTT

MQTT is a lightweight communication protocol commonly used in Internet Of Things (IOT) networks. This protocol typically operates over TCP/IP, but is agnostic to networking services so long as there is bi-directional transfer of data. This system is robust against data loss due to small data packets and three different levels of in-built verification of message delivery [21]. MQTT is a pub/sub based communication protocol where devices can publish new data to a topic and retrieve new publications from any topics that it has subscribed to. As such, the DIIP separates relevant data into topics based on the sensor it originated from. This data is formatted into various JSONs that are published at 1 Hz rate. 4.1 shows this data formatted into JSON as it is published.

Two solutions for MQTT brokers were tested, with the second being chosen as a more viable option. The first option is a commercially available IOT web server provided by Adafruit. Adafruit IO allows users to upload data and provides an dashboard to visualize any incoming data. This broker is very convenient, since it is a fully operational service. However, it is very limited in the amount of data that
### Table 4.1: JSON formatted data for the AQM

<table>
<thead>
<tr>
<th>Topic</th>
<th>JSON Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>{ &quot;Alt&quot;: &quot;27.300&quot;, &quot;Lat&quot;: &quot;29.233&quot;, &quot;Lon&quot;: &quot;-81.019&quot;, &quot;Sec&quot;: &quot;7.000&quot;, &quot;Min&quot;: &quot;49.000&quot;, &quot;Hr&quot;: &quot;14.000&quot; }</td>
</tr>
<tr>
<td>Breakout Sensors</td>
<td>{ &quot;Baro Temp&quot;: &quot;30.100&quot;, &quot;Pressure&quot;: &quot;1.010&quot;, &quot;TVOC&quot;: &quot;47.000&quot;, &quot;eCO2&quot;: &quot;712.000&quot; }</td>
</tr>
<tr>
<td>OPC N3</td>
<td>{ &quot;Temperature&quot;: &quot;26.832&quot;, &quot;Humidity&quot;: &quot;43.732&quot;, &quot;PM1&quot;: &quot;0.921&quot;, &quot;PM2_5&quot;: &quot;2.499&quot;, &quot;PM10&quot;: &quot;15.971&quot; }</td>
</tr>
</tbody>
</table>
can be published in a given interval. Adafruit offers a free version of its service that allows users to publish data to up to 7 topics, with 30 data points a minute. Because there are four topics per AQM, each being transmit once a second, this free broker is quickly overwhelmed and locked due to excessive incoming data. The paid tier of this service is marginally better, with unlimited topics and a 60 data point per minute cap. This severely limits the scalability of the DIIP and was therefore deemed not viable for this application.

The second option is an open source MQTT broker called Mosquitto MQTT. Mosquitto is highly configurable and highly scalable, making it ideal for this application. There is practically no limit in communication rates built into this broker and given enough bandwidth, no limit in the incoming data that can be received. This broker is run from the command line, making it not as user friendly as commercially available solutions. However, once a proper configuration is set, Python scripts or batch files can be used to enable easy operation of the broker. In the current configuration of the AQM there are 3 topics each being published at a 1 Hz rate. This broker can very easily handle this and leaves plenty of room for operation of multiple AQMs.

### 4.2.3 Python Data Logging Script

Although incoming data is received by the Mosquito broker, it is not immediately saved, necessitating a data logger. Using Python 3.8 and the Paho MQTT Python library, a logging script was developed that connects to the broker and subscribes to all of the topics produced by a given DIIP. Currently, this is only set to listen to a single DIIP, but can be set to log data from multiple sources. During the logging of data, each topic is parsed and saved into CSV format, with each field in the JSON receiving it’s own column. Each entry into the CSV file is also given a timestamp from the local time on the computer. This is not entirely necessary, as GPS UTC time is also saved, but it provides a reference for the amount of time a transmission takes. Once logging is terminated by the user, each of the generated CSV files are combined into a single excel file, with each CSV getting its own sheet.
4.3 Instrumentation

The central component of the DIIP is the Teensy 3.5 development board. This is a highly capable micro-controller board that is run by an ARM Cortex M4 micro-controller and has breakout pins that map to 64 digital I/O pins, 20 PWM pins, 27 analog inputs, 2 analog outputs, 6 serial, 3 SPI, and 3 I2C ports, and an included microSD card reader [22]. Because of the wide assortment of inputs, a very large amount of instruments can be integrated into this system given all the power and memory requirements are met. There exist a wide variety of COTS products for measurement of different aspects of air quality. In order to maintain a low cost, only a few instruments were integrated into what will be referred to as the “base” configuration of the AQM.
4.3. INSTRUMENTATION

Figure 4.4: Left: TMP102 digital temperature sensor on a SparkFun breakout board. Right: MS5611 barometric pressure sensor on the GY-63 breakout board.

4.3.1 Temperature and Pressure

The first instruments that will be discussed are instruments that describe typical atmospheric characteristics, such as temperature and pressure. There are three sensors that report temperature in the base configuration. However, one of these is incorporated into a sensor that will be discussed below and reports only the sensor temperature. For ambient temperature sensing, the Texas Instruments TMP102 digital temperature sensor and the Amsys MS5611 barometric pressure sensor (4.4 left and right respectively) are used. The TMP102 digital temperature sensor has a range of $-25^\circ$C to $+85^\circ$C and a resolution of $0.0625^\circ$C with an accuracy of $\pm 0.5^\circ$C [23]. The large range makes it an ideal choice for measurements in city environments around the world and although it is only accurate to $\pm 0.5^\circ$, it is accurate enough to establish a baseline understanding of the environment around the AQM.

The MS511 barometric pressure sensor has similar operating range and resolution for temperature as the TMP102, with the added benefit of primarily being a pressure sensor. However, this sensor is significantly more accurate, with an accuracy of $\pm 0.8^\circ$C in temperature and $\pm 1.5$ mbar in pressure[24]. This module also has configurable resolution settings that come at the cost of increased response time with increased resolution. However, because this system is not required to sample faster than 1 Hz, high resolution measurements can be made up to the highest resolution possible, i.e.
4.3. INSTRUMENTATION

Table 4.2: Description of negative health effects from varying concentrations of carbon dioxide [1].

<table>
<thead>
<tr>
<th>Concentration (ppm)</th>
<th>Health Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>None, average outdoor air level.</td>
</tr>
<tr>
<td>400 - 1000</td>
<td>None, typical level found in occupied spaces with good air exchange.</td>
</tr>
<tr>
<td>1000 - 2000</td>
<td>Level associated with complaints of drowsiness and poor air.</td>
</tr>
<tr>
<td>2000 - 5000</td>
<td>Level associated with headaches, sleepiness, and stagnant, stale, stuffy air. Poor concentration, loss of attention, increased heart rate and slight nausea may also be present.</td>
</tr>
<tr>
<td>5000+</td>
<td>This indicates unusual air conditions where high levels of other gases could also be present. Toxicity or oxygen deprivation could occur. This is the permissible exposure limit for daily workplace exposures.</td>
</tr>
</tbody>
</table>

0.012 mbar. Both of these sensors transfer data through I2C and are easily integrated with software libraries that act as instrument drivers.

4.3.2 eCO₂ and TVOC

Another important component of air quality is the content of compounds that are hazardous to health. One such compound is carbon dioxide (CO₂). Although CO₂ is the fourth most common compound in the earth’s atmosphere, larger concentrations can lead to serious negative health effects and even death. 4.2 describes these health effects and the levels of carbon dioxide where they typically start to manifest. Similarly, Volatile Organic Compounds (VOCs) are present throughout daily life and pose serious health effects, such as respiratory and cardiovascular disease, with both short and long term exposure. VOCs are released by a wide variety of household items such as cleaners, paints, and candles and vary greatly in chemical composition
4.3. INSTRUMENTATION

[25]. Therefore, quantifying the amount of VOCs depends on the sensitivity of the instrumentation to a wide variety of compounds.

To monitor the amount of CO\textsuperscript{2} and TVOC content in the air, the Adafruit CCS811 Air Quality breakout sensor was used. This sensor breakout contains a MOX gas sensor and an integrated MCU for processing of raw data into human readable data. Concentrations of 400-8192 ppm and 0-1187 ppb can be read for CO\textsuperscript{2} and TVOC content respectively. However, ideally these sensors would be calibrated against known sources. ERAU does not have the facilities to make this calibration and therefore, this sensor will only be used to indicate trends in chemical concentrations across a given sampling time rather than as absolute values. This data is passed to the Teensy 3.5 through I2C with the use of a software library provided by Adafruit, making integration into the AQM simple.

4.3.3 Particulate Matter

In a similar vein to CO\textsuperscript{2} and VOCs, Particulate Matter (PM) poses a hazard to health given high enough concentrations. The particulate matter of interest for this application consists of particles ranging from 1 micron to 10 microns in diameter when approximated as spheres. According to the Environmental Protection Agency (EPA), PM has been linked to health complications such as premature death in patients with lung and heart conditions, nonfatal heart attacks, and decreased lung function among others [26].

In order to monitor concentrations of PM, the Alphasense OPC-N3 optical particle sensor was used. This sensor is capable of measuring particles between 0.35 to 40 \( \mu \text{m} \) in diameter up to a maximum concentration of 2000 \( \mu \text{g}/\text{m}^3 \) [27]. This is achieved by using a proprietary algorithm to determine particle size from measure laser back scatter on a sample airflow. Similar to the CCS811 sensor, the OPC-N3 will not be used to determine accurate measurement of PM concentrations as there is currently no calibration being down in house. This is especially true for this sensor as a laboratory evaluation completed by the Air Quality Sensor Performance Evaluation Center (AQ-SPEC) demonstrated that the OPC-N3 significantly underestimates PM1, PM2.5,
and PM10 concentrations. When compared to a known source, the OPC-N3 was only able to detect anywhere from $\sim10$-$15\%$ of the actual value for PM1, $\sim15$-$25\%$ for PM2.5, and $\sim5\%$ for PM10$^{[28]}$. Although, the accuracy of this sensor is very low, it’s precision when adjusted for temperature and humidity is good enough to determine patterns in PM concentrations over larger periods of time. Conveniently, this sensor contained integrated temperature and humidity sensors which, along with PM concentration and raw bin counts, are transmitted through SPI lines to the Teensy 3.5 for storage.

### 4.3.4 PCB Layout

To integrate the above instrumentation, a PCB was designed using Autodesk Eagle. This board includes headers for the Teensy 3.5 and each of the breakout sensors as well as a surface mount footprint for the standalone MS5611 barometric pressure sensor. This allows for the use of either the breakout board or the IC if supply of one is every unreliable. There are also headers to connect to two external particulate matter sensors, the SPS30 and the OPC-N3. The SPS30 is a cheaper, and less reliable,
alternative to the OPC-N3, and was therefore excluded from the base configuration of the AQM.

This PCB also includes power management in the form of voltage regulators to supply proper voltage to the various components. The 5V power is handled by the STMicroelectronics L7805CDT-TR, which can regulate a maximum voltage of 35V down to 5V while supplying up to 1.5A. Similarly, the 3.3V power is handled by the STMicroelectronics LD29150DT33R, which has a maximum input voltage of 14V while also supplying up to 1.5A. With this combination, along with a DC power jack, the system can handle a wide range of voltage input while supplying enough current to power all of the integrated systems, which at maximum draw $\sim 500$ mA during measurements in the base configuration. A rendering of this PCB is shown in 4.5

### 4.4 Cost Breakdown

4.3 shows the cost per component for the AQM. This is not an exhaustive list, as it does not include miscellaneous components, such as, header blocks and electronics housing. The models of components like the voltage regulators are not explicitly stated due to uncertainty in the availability of certain products. As such, these can be any regulator that matches the requirements for 5V and 3.3V.

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PJRC Teensy 3.5</td>
<td>1</td>
<td>$24.25</td>
</tr>
<tr>
<td>ESP 8266 Wifi Module</td>
<td>1</td>
<td>$3.25</td>
</tr>
<tr>
<td>Sparkfun TMP102 Breakout</td>
<td>1</td>
<td>$5.50</td>
</tr>
<tr>
<td>GY-63 MS5611 Breakout</td>
<td>1</td>
<td>$8.99</td>
</tr>
<tr>
<td>Adafruit CCS811 Breakout</td>
<td>1</td>
<td>$19.95</td>
</tr>
<tr>
<td>Alphasense OPC-N3</td>
<td>1</td>
<td>$300</td>
</tr>
<tr>
<td>3.3V Regulator</td>
<td>1</td>
<td>$1.70</td>
</tr>
<tr>
<td>5V Regulator</td>
<td>1</td>
<td>$1.70</td>
</tr>
<tr>
<td>PCB</td>
<td>1</td>
<td>$\sim 1.50</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td>$\sim 350 **</td>
</tr>
</tbody>
</table>
4.5 Sample Data

The data shown below was recorded over a period of 10 hours, from approximately 11:00 AM EST to 10:00 PM EST. This system can run continuously, but was not left for a longer duration to conserve energy. The AQM system recording this data was placed on a window sill at ground level near a fairly busy street in Daytona Beach, as shown in Figure 4.6. Due to hardware issues, the TMP102 sensor was not included in this sample, although the temperature data from both the MS5611 and OPC N3 produce enough temperature data to excuse the omission of this temperature module.

Figure 4.7 shows the temperature and humidity data collected during the entire sampling period. Temperatures between the OPC N3 and MS5611 match fairly closely in overall trends, although there is higher variation in the readings from the MS5611. This can possibly be attributed to the location of each sensor. The MS5611
Figure 4.7: Temperature and Humidity data collected of an 10 hour period.

is located directly on the PCB, where it can be impacted from heat generated by other instruments or the voltage regulators. The OPC N3 also has higher airflow, due to the integrated fan, which can keep a more stable temperature. In addition, measurements from the MS5611 are corrected within the module for pressure differences, which also impacts the variability seen in the readings.

Figure 4.8 shows the concentrations of volatile organic compounds and carbon dioxide present in the air during sampling. The concentrations reported by this sensor show significantly larger concentrations during the first half of the sampling period when compared to the second half. This can possibly be attributed to temperature differences, as the trend of the concentrations somewhat matches the trend in temperatures. Because the CSS811 is a hot-plate MOx sensor, variation in temperature may affect the values output by the sensor. Another explanation could be the drop in traffic seen along the nearby road. This dip in TVOC and eCO\textsuperscript{2} coincides with the time after rush hour traffic has decreased, which therefore leads to lower pollutant concentrations. However, this drop is very drastic and must be examined further with other datasets.

Figure 4.9 shows the particulate matter density ranging from PM1 to PM10 over
the entire sampling period. Contrary to the data previously discussed, the concentration of particles in the PM10 range increased as time went on. As stated previously, these measurements are not reliable absolute values, as the OPC N3 tends to under report particle densities. However, trends can be identified and validated with a known source of particulate matter. In future sampling periods, periodic known sources should be place near the sensor, such as a lit match, to provide a source of comparison for the ambient air quality.
Figure 4.9: Particulate matter density data collected over a 10 hour period.
Chapter 5

Summary and Future Work

5.1 GPSsonde

In its current state, the GPSsonde can be considered a complete and operational system. The design described in this thesis has met all of the initial requirements that were set out to be met. Through the various flights that have been completed with this system, it has been shown that reliable communication beyond 100km, altitudes above 25 km, and concurrent flight of multiple payloads are all achievable given the current iteration of the GPSsonde. However, there is always room for improvement. An improvement that can be made is to increase the data rate by decreasing packet size. Currently, a packet size of 16 bytes (plus the 8 byte LoRa preamble) at a bit rate of 183 bps yields a maximum possible data rate of 1 packet per 1.05 seconds. Due to the method chosen for deciding when to transmit, only whole number choices are possible, i.e. 1 per second, 2 per second, etc., hence the 2 second communication rate. A reduction of a single byte would drop the transmission time of one packet below the 1 second threshold, thus allowing the data rate for all payloads to shift from 1 per 12 seconds to 1 per 6 seconds. There is no clear candidate for removal as every byte is either necessary housekeeping or GPS data. However, the packet number counter that is used to keep every packet in order takes up 2 bytes in the current packet. This is not as necessary for the GPSsonde as transmissions are reliant on accurate GPS time. Therefore, the packet counter can be removed assuming that losses can
be determined based on larger than expected intervals between packets. This reduces the total transmission length from 24 to 22 bytes and a transmission time of 0.94 seconds, which enables a 1 per 6 second data rate. Alternatively, the data rate of the GPSsonde radios can be increased. The preliminary link budget analysis that first indicated a 183 bps would have sufficient margin to communicate at ranges greater than 100 km also show that higher data rates are possible. This would have to be tested experimentally, but would greatly improve the capabilities of the GPSsonde as a whole.

5.2 HAB Bus and CDU

In its current state, the HAB bus is capable of carrying third party instrumentation to altitudes above 30km. This system can provide reliable telemetry and flight profile control for a wide variety of atmospheric instrumentation, such as the instruments required as a part of MURI. This is achieved through the use of high powered transceivers and the robust transmission systems enabled on the bus. More work remains to be done to enable reliable transmission of images and enable easy configuration of the hardware. The current PCB does not have generalized breakouts for all of the available communications lines, making integration of instruments more difficult. Moving towards a system that does not require recompilation of all of the software for header file control would also make operation of the system easier for an end user. This would require moving away from the Arduino platform, but may be a worthwhile effort.

Moreover, the novel controlled descent techniques outlined in this thesis are not yet entirely functional. Although several test flights have resulted in failures, there is still hope for a successful TM-CDU flight. In order for this to occur, a better thermal protection system must be implemented, alongside a well defined control law. In the meantime, the standard configuration for the valve CDU can attain proven altitudes of 32 km, which places instrumentation well within the stratosphere.
5.3 IoT Instrumentation Package

The air quality monitoring system is functional, but not complete. At the time of writing, the DIIP can only communicate with a server operating on the same network. To allow free uploading of data to any server around the world, port forwarding would have to be enabled for each network that an DIIP is operating on. Although this is not necessarily a complicated process, deployment is limited to locations where a network administrator is available for installation. In addition, enabling port forwarding introduces security risks, as a private network has to be opened up to the global internet. With proper care, this risk can be minimized, but will require more knowledge of networking than is currently available to the author.

Aside from this, the DIIP is capable of taking readings from an arbitrary number of environmental sensors and transferring that data over the internet. This data is saved either by using a logging script from python, or can be directly accessed by any device operating on the same network with the correct password. Further instrumentation can be added given enough open ports on the Teensy 3.5 at the need of an end user.
Bibliography


Appendix A

GPSsonde Reference

A.1 Flight Software
Julio Guardado

Main

#include <SPI.h>
#include <RH_RF95.h>
#include "Definitions.h"

RH_RF95 rf95(8, 3); // Adafruit Feather M0 with RFM95-
UbloxGPS gps(&Serial1);

void setup() {
  /*********SERIAL PORTS*********/
  Serial.begin(9600);
  Serial1.begin(GPS_BAUD);
  // SPI.begin();
  delay(2000);

  /*********LORA RADIO SETUP*********/
  if (!rf95.init()) {
    Serial.println("LoRa radio init failed");
    while (1);
  }

  Serial.println("LoRa radio init OK!");
  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while (1);
  }

  Serial.println("Set Freq to:"); Serial.println(RF95_FREQ);
  rf95.setTxPower(20);
  rf95.setModemConfig(RH_RF95::Bw125Cr48Sf4096); // Min Bw - Max Sfr for max range
  pinMode(13, OUTPUT);
  /*********GPS SETUP*********/
  initGPS();
  /**********SERIAL DEFINITIONS**********/
  STARTUP Monitor     Serial
  GPS UBlox           Serial1

  void setup() {
    /**********SERIAL DEFINITIONS**********/
    STARTUP Monitor     Serial
    GPS UBlox           Serial1
void loop() {
    updateGPS();
    // Send Packet based on timed interval i.e. modulus of timer with sync number
    if (((utcSec % (NUM_PAYLOADS * 2)) + 1) == (payloadID * 2) && newData == 1) {
        //    delay(100);
        newData = 0;
        sendDataRadio();
        Serial.println("OK");
    }
}
Definitions

Julio Guardado

1 /*---------------------------------------------*/
2 /* Main Variable Definitions */
3 /*---------------------------------------------*/
4
5 // packet header
6 uint16_t id_erau = 0b1010101010 << 7;  // 9 bit Header for packet
7 uint16_t payloadID = 3;
8 uint16_t packNum;  // Packet number counter
9 uint16_t header = id_erau | (payloadID << 4);
10
11 #if payloadID == NUM_PAYLOADS
12   // Set sync number based on number of balloons being launched
13   #define SYNC_NUM 0
14 #else
15   #define SYNC_NUM payloadID/0x11;
16 #endif
17
18 unsigned long timer = 0;
19  // *********** GPS **************/
20 bool ack = false;
21 // GPS Communicatoin
22 #define GPS_BAUD 9600
23
24 // GPS Data
25 float alt;
26 float latitude;
27 float longitude;
28 float velN;
29 float velE;
30 float velD;
31
32 uint8_t stat = 0;  //Status
33 uint16_t numSats = 0;  //Number of Satellites in View
34 uint8_t utcHour = 0;  //UTC Time - Hour
35 uint8_t utcMin = 0;  //UTC Time - Minutes
36 uint8_t utcSec = 0;  //UTC Time - Seconds
37 uint8_t prevSec = 0;
38  // Timing
39 #define UPDATE_INTERVAL_GPS 2500
40 unsigned long gpsTimer;
41 uint8_t newData = 0;
42  // *********** Battery Voltage **************/
43 #define V_BAT A7
44 uint16_t vbat;
45  
46 #define RADIO_BAUD 230400  // define RADIO_BAUD 230400
47 #define RADIO_SIZE 19  // define RADIO_SIZE 19
48 #define RF95_FREQ 915.0
49 byte dataPack[RADIO_SIZE];
A.1. FLIGHT SOFTWARE

```c
55 unsigned long dateTime;
56 #define DATA_INTERVAL 1000
57
58 uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
59 uint8_t len = sizeof(buf);
```
Data

Julio Guardado

1 ///////////////////////////////////////////////////////
2 /// Packet Definitions /////////////////////////////////
3 ///////////////////////////////////////////////////////
4 ///
5 void sendDataRadio() {
6   //******** IDENTIFIER ********/
7   dataPack[0] = header;   // 9 bit ERAU ID, 3 bit Payload ID, 4 bit numSats
8   dataPack[1] = header >> 8;
10  dataPack[3] = packNum >> 8;
11  packNum += 1;
12  //********GPS DATA********/
13  byte * point_alt = (byte*) &alt;
14  dataPack[4] = point_alt[0];
15  dataPack[5] = point_alt[1];
16  dataPack[6] = point_alt[2];
17  dataPack[7] = point_alt[3];
18  byte * point_lat = (byte*) &latitude;
19  dataPack[8] = point_lat[0];
20  dataPack[9] = point_lat[1];
21  dataPack[10] = point_lat[2];
22  dataPack[11] = point_lat[3];
23  byte * point_lon = (byte*) &longitude;
24  dataPack[12] = point_lon[0];
25  dataPack[13] = point_lon[1];
26  dataPack[14] = point_lon[2];
27  dataPack[15] = point_lon[3];
28  // ******** BATTERY VOLTAGE ********/
29  // vbat = analogRead(A0);
30  //
31  rf95.send(dataPack, RADIO_SIZE);
32  for (int i = 0; i < RADIO_SIZE; i++) {
33    Serial.print(dataPack[i], HEX); Serial.print(" ");
34  }
35  Serial.write("\n");
36  delay(10);
37 }

A.1. FLIGHT SOFTWARE
GPS Functions

Julio Guardado

```c
void initGPS() {
    unsigned long gpsTimer = millis();
    bool set_airborne = gps.setAirborne();
    while (Serial1.available()) Serial1.read();
    delay(50);
    while (!(set_airborne && (millis() - gpsTimer < 1 * 60 * 1000 ))) {
        set_airborne = gps.setAirborne();
        delay(1);
        Serial.println("Airborne Mode Not Set!");
    }
    if (set_airborne) Serial.println("GPS OK");
    else Serial.println("GPS Failure");
}

void updateGPS() {
    // Serial.println("gps loop");
    gps.update();
    //get measurements
    alt = (gps.getAlt_meters());
    latitude = (gps.getLat());
    longitude = (gps.getLon());
    numSats = gps.getSats();
    utcSec = gps.getSecond();
    utcMin = gps.getMinute();
    utcHour = gps.getHour();
    // Update header + numsats
    header = header & 0xFFF0; // Apply mask to clear lowest 4 bits
    header = header | numSats; // Populate lowest four bits
    if ( utcSec - prevSec != 0 ) {
        newData = 1;
        prevSec = utcSec;
        Serial.println("Transmit Flag: ");
        Serial.println(((utcSec % NUM_PAYLOADS) + 1 ) == payloadID));
    }
}
```
A.2 Ground Station Software
A.2. GROUND STATION SOFTWARE

Table of Contents

- Setup maps on initialization ................................................................. 1
- Parse Packet ......................................................................................... 3
- Update Display ...................................................................................... 15
- Calculate velocity north, east, and down .............................................. 16
- Parse Packet ......................................................................................... 23
- Update Display ...................................................................................... 24
- Calculate velocity north, east, and down .............................................. 24

```matlab
classdef HAB_GPSonde_GS_V5 < matlab.apps.AppBase

% Properties that correspond to app components
properties (Access = public)
    UIFigure matlab.ui.Figure
    Location2D matlab.ui.control.UIAxes
    Location3D matlab.ui.control.UIAxes
    VerticalAscentRate matlab.ui.control.UIAxes
    NorthVelocity matlab.ui.control.UIAxes
    EastVelocity matlab.ui.control.UIAxes
    HorizontalVelocity matlab.ui.control.UIAxes
    Menu matlab.ui.container.Menu
    RefreshCOMPortsMenu matlab.ui.container.Menu
    LoadMapsMenu matlab.ui.container.Menu
    LoadPredictionFileMenu matlab.ui.container.Menu
    BeginTrackingButton matlab.ui.control.Button
    EndTrackingButton matlab.ui.control.Button
    trackingStat matlab.ui.control.Lamp
    RxCOMPortDropDownLabel matlab.ui.control.Label
    RxCOMPortDropDown matlab.ui.control.DropDown
    ReplayButton matlab.ui.control.Button
    ReplaySpeedEditField matlab.ui.control.EditField
    ReplaySpeedEditFieldLabel matlab.ui.control.Label
    payloadPositions matlab.ui.container.TabPage
    ID1Tab matlab.ui.container.Tab
    gpsLAT matlab.ui.control.Label
    gpsLONG matlab.ui.control.Label
    gpsALT matlab.ui.control.Label
    gpsFIX matlab.ui.control.Label
    LatitudeLabel matlab.ui.control.Label
    LongitudeLabel matlab.ui.control.Label
    AltitudeLabel matlab.ui.control.Label
    FixQLabel matlab.ui.control.Label
    gpsSATS matlab.ui.control.Label
    gpsHOUR matlab.ui.control.Label
    gpsMIN matlab.ui.control.Label
    gpsSEC matlab.ui.control.Label
    SatsLabel matlab.ui.control.Label
    GPSUTCTimeLabel matlab.ui.control.Label
```
A.2. GROUND STATION SOFTWARE

vertRate                   matlab.ui.control.Label
vertRateLabel              matlab.ui.control.Label
northRate                  matlab.ui.control.Label
NorthRateLabel             matlab.ui.control.Label
EastRateLabel              matlab.ui.control.Label
eastRate                   matlab.ui.control.Label
HorizontalRateLabel        matlab.ui.control.Label
horizontalRate             matlab.ui.control.Label
RSSILabel                  matlab.ui.control.Label
RSSI                       matlab.ui.control.Label
BatteryLabel               matlab.ui.control.Label
battVolt                   matlab.ui.control.Label
packetnum                  matlab.ui.control.Label
range                      matlab.ui.control.Label
PacketLabel                matlab.ui.control.Label
RangeLabel                 matlab.ui.control.Label
Lamp                       matlab.ui.control.Lamp
RemovePayloadButton        matlab.ui.control.Button
end

properties  (Access = private)
xdata % For plotting x data
ydata % For plotting y data
zdata % For plotting z data
wdata % For plotting w data
flag = 0;
flag_GS = 0;
hC = 0;
maxC = 0;
myGSCoord = [29.187904, -81.048094, 45];
% CUSFPredicted;

startTime = 0;
newMapFlag = 0;
predFileFlag = 0;
predFileCol = [ 'k' , 'r', 'b', 'g', 'c' ];

loopBreak = 0; % Description
color = [ 'bgrcmyk' ]; % colors for plotting
payloadID; % array of ids to track
i = 2; % Description
name; % Description
handles % handles for ui features
end

properties  (Access = public)
dataFile;
gpsData;
scientificData;
serial_GS;
A; B;
s;

methods (Access = private)

Setup maps on initialization

% sets x and y limits to the maps
function [latlim, lonlim] = getMapLimits(app, lat0, lon0, h0)
    if lat0 <= 90 && lat0 >= -90 && lon0 <= 180 && lon0 >= -180 && isnumeric(h0) && isreal(h0)
        az = [0 90 180 270];
        slantRange = 200*1000;
        elev = 0;
        lat = [0 0 0 0];
        lon = lat;
        h=lat;
        for f = 1:4
            [lat(f), lon(f), h(f)] = aer2geodetic(az(f), elev, slantRange, lat0, lon0, h0, app.spheroid);
        end
        latlim = [lat(3) lat(1)];
        lonlim = [lon(4) lon(2)];
    else
        errordlg('Check your position and try again');
    end
end

% load maps in from online prediction
function ZA = loadMaps(app, latlim, lonlim)
    ZA=[];
    prompt = {'Would you like to download new Map data?'; '(Requires Internet Connection)'};
title = 'WMS Map Update';
answ = questdlg(prompt, title, 'New Map', 'Load Map', 'Cancel', 'Cancel');
switch answ
    case 'New Map'
        numberOfAttempts = 5;
        attempt = 0;
        info = [];
        mundalisServer = 'http://ows.mundialis.de/services/service?';
        OSM_WMS_Uni_Heidelberg = 'http://129.206.228.72/cached/osm?';
serv2 = 0;
        while(isempty(info))
            try
                if serv2 == 0
                    info = wmsinfo(mundalisServer);
                    orthoLayer = info.Layer;
                elseif serv2 == 1

A.2. GROUND STATION SOFTWARE
A.2. GROUND STATION SOFTWARE

```matlab
info = wmsinfo(OSM_WMS_Uni_Heidelberg);
orthoLayer = info.Layer;
end
catch
    attempt = attempt + 1;
    if attempt > numberOfAttempts && serv2 == 0
        warning('Server 1 is not available. Trying Server 2');
        serv2 = 1;
        attempt = 0;
    end
    if serv2 == 1 && attempt > numberOfAttempts
        warndlg({'WMS servers are not available.';'Please load an existing Map'});
        return
    end
end
[ZA, ~] = wmsread(orthoLayer(2:3), 'Latlim', latlim, 'Lonlim', lonlim, ...
    'ImageFormat', 'image/png', 'ImageHeight', 2160, 'ImageWidth', 4096);
    app.newMapFlag = 1;
case 'Load Map'
    [newfile,path] = uigetfile({'*.map' 'Load Map File' 'map1.map'});
    figure(app.UIFigure);
    if newfile == 0
        return;
    end
    filename=fullfile(path,newfile);
    load(filename,'ZA','-mat');
    app.newMapFlag = 0;
case 'Cancel'
    return
end

%plots prediction data
function results = DrawMaps(app,ZA,latlim,lonlim)
    results = 0;
    imagesc(app.Location2D,lonlim,latlim,flipud(ZA));
    imagesc(app.Location3D,lonlim,latlim,flipud(ZA));
    xlim(app.Location2D,lonlim)
    ylim(app.Location2D,latlim)
xlim(app.Location3D,lonlim)
    ylim(app.Location3D,lonlim)
    view(app.Location3D,15,15)
```
if app.newMapFlag == 1
    q2 = questdlg('Would you like to save the map data?', 'Save?', 'Yes', 'No', 'Yes');
    switch q2
        case 'Yes'
            [newfile, path] = uiputfile('*.map', 'Create Data File', 'map1.map');
            if newfile == 0
                return;
            end
            filename = fullfile(path, newfile);
            save(filename, 'ZA');
        case 'No'
    end
end
results = 1;
end

function results = createNewTab(app, id)
    % Create a new uitab in this group
    app.handles.tabs(app.i) = uitab('Parent', app.payloadPositions, 'Title', sprintf('ID# %d', id));
    % Find number of payloads and add contents to new tab
    newSize = numel(findall(app.payloadPositions, 'type', 'uitab'));
    % Assign new payload to end of payload ID matrix
    app.payloadID(newSize) = (id);
    % Create contents of tab
    % Create gpsLAT
    app.gpsLAT(newSize) = uilabel(app.handles.tabs(app.i));
    app.gpsLAT(newSize).BackgroundColor = [1 1 1];
    app.gpsLAT(newSize).HorizontalAlignment = 'right';
    app.gpsLAT(newSize).VerticalAlignment = 'top';
    app.gpsLAT(newSize).Position = [22 97 61 15];
    app.gpsLAT(newSize).Text = '';
    % Create gpsLONG
    app.gpsLONG(newSize) = uilabel(app.handles.tabs(app.i));
    app.gpsLONG(newSize).BackgroundColor = [1 1 1];
    app.gpsLONG(newSize).HorizontalAlignment = 'right';
    app.gpsLONG(newSize).VerticalAlignment = 'top';
    app.gpsLONG(newSize).Position = [87 97 61 15];
    app.gpsLONG(newSize).Text = '';
    % Create gpsALT
    app.gpsALT(newSize) = uilabel(app.handles.tabs(app.i));

A.2. GROUND STATION SOFTWARE

app.gpsALT(newSize).BackgroundColor = [1 1 1];
app.gpsALT(newSize).HorizontalAlignment = 'right';
app.gpsALT(newSize).VerticalAlignment = 'top';
app.gpsALT(newSize).Position = [153 97 61 15];
app.gpsALT(newSize).Text = '';

% Create gpsFIX
app.gpsFIX(newSize) = uilabel(app.handles.tabs(app.i));
app.gpsFIX(newSize).BackgroundColor = [1 1 1];
app.gpsFIX(newSize).HorizontalAlignment = 'right';
app.gpsFIX(newSize).VerticalAlignment = 'top';
app.gpsFIX(newSize).Position = [225 97 25 15];
app.gpsFIX(newSize).Text = '';

% Create LatitudeLabel
app.LatitudeLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.LatitudeLabel(newSize).HorizontalAlignment = 'center';
app.LatitudeLabel(newSize).VerticalAlignment = 'top';
app.LatitudeLabel(newSize).Position = [22 119 60 15];
app.LatitudeLabel(newSize).Text = 'Latitude';

% Create LongitudeLabel
app.LongitudeLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.LongitudeLabel(newSize).HorizontalAlignment = 'center';
app.LongitudeLabel(newSize).Position = [88 119 58 15];
app.LongitudeLabel(newSize).Text = 'Longitude';

% Create AltitudeLabel
app.AltitudeLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.AltitudeLabel(newSize).HorizontalAlignment = 'center';
app.AltitudeLabel(newSize).VerticalAlignment = 'top';
app.AltitudeLabel(newSize).Position = [153 117 61 18];
app.AltitudeLabel(newSize).Text = 'Altitude';

% Create FixQLabel
app.FixQLabel(newSize) = uilabel(app.handles.tabs(app.i));
app.FixQLabel(newSize).HorizontalAlignment = 'center';
app.FixQLabel(newSize).VerticalAlignment = 'top';
app.FixQLabel(newSize).Position = [220 120 37 15];
app.FixQLabel(newSize).Text = 'Fix Q.';

% Create gpsSATS
app.gpsSATS(newSize) = uilabel(app.handles.tabs(app.i));
app.gpsSATS(newSize).BackgroundColor = [1 1 1];
app.gpsSATS(newSize).HorizontalAlignment = 'right';
app.gpsSATS(newSize).VerticalAlignment = 'top';
app.gpsSATS(newSize).Position = [262 97 25 15];
app.gpsSATS(newSize).Text = '';

% Create gpsHOUR
app.gpsHOUR(newSize) = uilabel(app.handles.tabs(app.i));
app.gpsHOUR(newSize).BackgroundColor = [1 1 1];
app.gpsHOUR(newSize).HorizontalAlignment = 'right';
app.gpsHOUR(newSize).VerticalAlignment = 'top';
app.gpsHOUR(newSize).Position = [299 97 25 15];
app.gpsHOUR(newSize).Text = '';

% Create gpsMIN
app.gpsMIN(newSize) = uilabel(app.handles.tabs(app.i));
app.gpsMIN(newSize).BackgroundColor = [1 1 1];
app.gpsMIN(newSize).HorizontalAlignment = 'right';
app.gpsMIN(newSize).VerticalAlignment = 'top';
app.gpsMIN(newSize).Position = [329 97 25 15];
app.gpsMIN(newSize).Text = '';

% Create gpsSEC
app.gpsSEC(newSize) = uilabel(app.handles.tabs(app.i));
app.gpsSEC(newSize).BackgroundColor = [1 1 1];
app.gpsSEC(newSize).HorizontalAlignment = 'right';
app.gpsSEC(newSize).VerticalAlignment = 'top';
app.gpsSEC(newSize).Position = [359 97 25 15];
app.gpsSEC(newSize).Text = '';

% Create SatsLabel
app.SatsLabel(newSize) = uilabel(app.handles.tabs(app.i));
app.SatsLabel(newSize).HorizontalAlignment = 'center';
app.SatsLabel(newSize).VerticalAlignment = 'top';
app.SatsLabel(newSize).Position = [256 119 37 15];
app.SatsLabel(newSize).Text = '#Sats';

% Create GPSUTCTimeLabel
app.GPSUTCTimeLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.GPSUTCTimeLabel(newSize).HorizontalAlignment = 'center';
app.GPSUTCTimeLabel(newSize).VerticalAlignment = 'top';
app.GPSUTCTimeLabel(newSize).Position = [299 119 85 15];
app.GPSUTCTimeLabel(newSize).Text = 'GPS UTC Time';

% Create vertRate
app.vertRate(newSize) = uilabel(app.handles.tabs(app.i));
app.vertRate(newSize).BackgroundColor = [1 1 1];
app.vertRate(newSize).HorizontalAlignment = 'right';
app.vertRate(newSize).VerticalAlignment = 'top';
app.vertRate(newSize).Position = [57 52 50 15];
app.vertRate(newSize).Text = '';

% Create vertRateLabel
app.vertRateLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.vertRateLabel(newSize).HorizontalAlignment = 'center';
app.vertRateLabel(newSize).VerticalAlignment = 'top';
app.vertRateLabel(newSize).Position = [45 67 74 22];
app.vertRateLabel(newSize).Text = 'Vertical Rate';
% Create northRate
app.northRate(newSize) = uilabel(app.handles.tabs(app.i));
app.northRate(newSize).BackgroundColor = [1 1 1];
app.northRate(newSize).HorizontalAlignment = 'right';
app.northRate(newSize).VerticalAlignment = 'top';
app.northRate(newSize).Position = [117 52 64 15];
app.northRate(newSize).Text = '';

% Create NorthRateLabel
app.NorthRateLabel(newSize) =
uilabel(app.handles.tabs(app.i));
app.NorthRateLabel(newSize).HorizontalAlignment = 'center';
app.NorthRateLabel(newSize).Position = [117 74 64 14];
app.NorthRateLabel(newSize).Text = 'North Rate';

% Create EastRateLabel
app.EastRateLabel(newSize) =
uilabel(app.handles.tabs(app.i));
app.EastRateLabel(newSize).HorizontalAlignment = 'center';
app.EastRateLabel(newSize).Position = [196 74 58 14];
app.EastRateLabel(newSize).Text = 'East Rate';

% Create eastRate
app.eastRate(newSize) = uilabel(app.handles.tabs(app.i));
app.eastRate(newSize).BackgroundColor = [1 1 1];
app.eastRate(newSize).HorizontalAlignment = 'right';
app.eastRate(newSize).VerticalAlignment = 'top';
app.eastRate(newSize).Position = [193 52 64 15];
app.eastRate(newSize).Text = '';

% Create HorizontalRateLabel
app.HorizontalRateLabel(newSize) =
uilabel(app.handles.tabs(app.i));
app.HorizontalRateLabel(newSize).HorizontalAlignment = 'center';
app.HorizontalRateLabel(newSize).Position = [269 75 88 13];
app.HorizontalRateLabel(newSize).Text = 'Horizontal Rate';

% Create horizontalRate
app.horizontalRate(newSize) = uilabel(app.handles.tabs(app.i));
app.horizontalRate(newSize).BackgroundColor = [1 1 1];
app.horizontalRate(newSize).HorizontalAlignment = 'right';
app.horizontalRate(newSize).VerticalAlignment = 'top';
app.horizontalRate(newSize).Position = [269 52 88 15];
app.horizontalRate(newSize).Text = '';

% Create RSSILabel
app.RSSILabel(newSize) = uilabel(app.handles.tabs(app.i));
app.RSSILabel(newSize).HorizontalAlignment = 'center';
app.RSSILabel(newSize).Position = [216 22 51 22];

A.2. GROUND STATION SOFTWARE
app.RSSILabel(newSize).Text = 'RSSI';

% Create RSSI
app.RSSI(newSize) = uilabel(app.handles.tabs(app.i));
app.RSSI(newSize).BackgroundColor = [1 1 1];
app.RSSI(newSize).HorizontalAlignment = 'right';
app.RSSI(newSize).VerticalAlignment = 'top';
app.RSSI(newSize).Position = [216 8 51 15];
app.RSSI(newSize).Text = '';

% Create BatteryLabel
app.BatteryLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.BatteryLabel(newSize).HorizontalAlignment = 'center';
app.BatteryLabel(newSize).Position = [279 22 51 22];
app.BatteryLabel(newSize).Text = 'Battery';

% Create battVolt
app.battVolt(newSize) = uilabel(app.handles.tabs(app.i));
app.battVolt(newSize).BackgroundColor = [1 1 1];
app.battVolt(newSize).HorizontalAlignment = 'right';
app.battVolt(newSize).VerticalAlignment = 'top';
app.battVolt(newSize).Position = [279 8 51 15];
app.battVolt(newSize).Text = '';

% Create packetnum
app.packetnum(newSize) = uilabel(app.handles.tabs(app.i));
app.packetnum(newSize).BackgroundColor = [1 1 1];
app.packetnum(newSize).HorizontalAlignment = 'right';
app.packetnum(newSize).VerticalAlignment = 'top';
app.packetnum(newSize).Position = [80 8 51 15];
app.packetnum(newSize).Text = '';

% Create range
app.range(newSize) = uilabel(app.handles.tabs(app.i));
app.range(newSize).BackgroundColor = [1 1 1];
app.range(newSize).HorizontalAlignment = 'right';
app.range(newSize).VerticalAlignment = 'top';
app.range(newSize).Position = [143 8 51 15];
app.range(newSize).Text = '';

% Create PacketLabel
app.PacketLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.PacketLabel(newSize).HorizontalAlignment = 'center';
app.PacketLabel(newSize).Position = [80 22 52 22];
app.PacketLabel(newSize).Text = 'Packet #';

% Create RangeLabel
app.RangeLabel(newSize) =
    uilabel(app.handles.tabs(app.i));
app.RangeLabel(newSize).HorizontalAlignment = 'center';
app.RangeLabel(newSize).Position = [143 22 51 22];
app.RangeLabel(newSize).Text = 'Range';
A.2. GROUND STATION SOFTWARE

% Create Lamp
app.Lamp(newSize) = uilamp(app.handles.tabs(app.i));
app.Lamp(newSize).Position = [2 127 20 20];
app.Lamp(newSize).Color = app.color(newSize);

app.i = app.i + 1;

end

end

% Callbacks that handle component events
methods (Access = private)

% Code that executes after component creation
function startupFcn(app)
    clc;
    app.RxCOMPortDropDown.Items = cellstr(seriallist);
    hold(app.Location3D, 'on');
    hold(app.Location2D, 'on');
    app.myGSCoord = [str2double(app.GS_Latitude.Value), ...
                     str2double(app.GS_Longitude.Value), str2double(app.GS_Altitude.Value)];
    app.UIFigure.Position = [0 0 1280 700];
    
    % Load 2D Map
    Lat = 29.187904;
    Lon = -81.048094;
    Z = 0;
    [latlim, lonlim] = getMapLimits(app, Lat, Lon, Z);
    try
        load('200km.map', 'ZA', '-mat');
    catch
        ZA = loadMaps(app, latlim, lonlim);
    end
    imagesc(app.Location2D, lonlim, latlim, flipud(ZA));
    imagesc(app.Location3D, lonlim, latlim, flipud(ZA));
    xlim(app.Location2D, lonlim)
    ylim(app.Location2D, latlim)
    xlim(app.Location3D, lonlim)
    ylim(app.Location3D, latlim)
    view(app.Location3D, 15, 15)
    zlim(app.Location3D, [-.001, 30])
    axis(app.VerticalAscentRate, 'tight')
    ylim(app.VerticalAscentRate, [0 5]);
    pause(0.001)
A.2. GROUND STATION SOFTWARE

```matlab
% Callback function
function LoadPredictionFileFunction(app, event)
pAns = questdlg('Are you sure you would like to add a prediction path?',...
    'Prediction Path Option',...
    'CUSF', 'Cancel');
switch pAns
    case 'CUSF'
        [newfile,path] = uigetfile('*.csv', 'Prediction Path File', 'flight_path.csv');
        figure(app.UIFigure);
        if newfile ~= 0
            app.predFileFlag = app.predFileFlag + 1;
            predFile = fullfile(path, newfile);
            predictor
            % http://predict.habhub.org
            app.CUSFpredicted = csvread(predFile);
            app.ydata = app.CUSFpredicted(:,2); % lat
            app.xdata = app.CUSFpredicted(:,3); % lon
            app.zdata = app.CUSFpredicted(:,4); % alt
            app.wdata = app.CUSFpredicted(:,1); % time
            app.wdata = app.wdata - app.wdata(1);
            plot3(app.Location3D, app.xdata, app.ydata, app.zdata./1000, app.predFileCol(app.predFileFlag), 'LineWidth', 2)
            % Predicted path plot from hab-hub.org
        end
        app.hC = app.CUSFpredicted(:,4); % alt
        app.maxC = find(app.hC == max(app.hC));
    case 'Cancel'
end
end
```
function LoadPositionButtonPushed(app, event)
    q1 = questdlg('Do you want to get the GS coordinates from the GPS [GS Connection Required]?','GS GPS Coordinates','Yes','No','Yes');
    switch q1
    case 'Yes'
        fprintf(app.serial_GS,'
');
        while (app.serial_GS.BytesAvailable == 0)
            end
        [coords] = fscanf(app.serial_GS,'%d,%d,%d
');
        app.GS_Latitude.Value = num2str(coords(1)/10000000);
        app.GS_Longitude.Value = num2str(coords(2)/10000000);
        app.GS_Altitude.Value = num2str(coords(3)/100);
    end
    prompt = {'Enter the decimal latitude of the center', 'Enter the decimal Longitude of the center', ...
        'Enter the Altitude in meters', 'Enter the desired Map Radius in km'};
    title = 'Map Configuration';
    dims = [1,35];
    default = {app.GS_Latitude.Value, app.GS_Longitude.Value, app.GS_Altitude.Value, '200'};
    answer = inputdlg(prompt,title,dims,default);
    if ~isempty(answer) && isreal(str2double(answer))
        mapZ = str2double(answer{3});
        mapLat = str2double(answer{1});
        mapLon = str2double(answer{2});
        % app.radius.Text = answer{4};
    else
        errordlg('Check the center position and try again');
        return
    end
    q2 = questdlg('Is the antenna position the same as the map center?','Antenna Position','Yes','No','Yes');
    switch q2
    case 'Yes'
        h0 = mapZ;
        lat0 = mapLat;
        lon0 = mapLon;
        app.GS_Latitude.Value = num2str(lat0);
        app.GS_Longitude.Value = num2str(lon0);
        app.GS_Altitude.Value = num2str(h0);
    case 'No'
        prompt = {'Enter the decimal latitude', 'Enter the decimal Longitude', ...
            'Enter the Altitude in meters'};
        answer = inputdlg(prompt,title,dims,default);
        if ~isempty(answer) && isreal(str2double(answer))
            mapZ = str2double(answer{3});
            mapLat = str2double(answer{1});
            mapLon = str2double(answer{2});
            % app.radius.Text = answer{4};
        else
            errordlg('Check the center position and try again');
            return
        end
    end
title = 'Antenna Position';
dims = [1,35];
answer = inputdlg(prompt,title,dims,default);
if ~isempty(answer) && isreal(str2double(answer))
    h0 = str2double(answer{3});
    lat0 = str2double(answer{1});
    lon0 = str2double(answer{2});
    app.GS_Latitude.Value = answer{1};
    app.GS_Longitude.Value = answer{2};
    app.GS_Altitude.Value = answer{3};
    app.radius.Text = answer{4};
else
    errordlg('Check your position and try again');
    return
end

[latlim, lonlim] = getMapLimits(app,mapLat,mapLon,mapZ);
ZA = loadMaps(app,latlim,lonlim);
success = 0;
if ~isempty(ZA)
success = DrawMaps(app,ZA,latlim,lonlim);
end
if success == 1
    plot3(app.Location3D, lon0, lat0, h0/1000, 'r*', 'LineWidth',2);
    plot(app.Location2D, lon0, lat0, 'r*')
    app.LoadPredictionfileButton.Enable = 'on';
    app.LoadPredictionFileMenu.Enable = 'on';
    app.BeginTrackingButton.Enable='on';
    app.GroundStationTestButton.Enable = 'on';
end
figure(app.UIFigure);

% Button pushed function: BeginTrackingButton
function BeginTrackingButtonPushed(app, event)
instrreset();
[newfile,path] = uiputfile('*.bin','Create Data File',strcat(datestr(datetime, 'yyyy_mm_dd'),',','_d' ,datestr(datetime, 'HH_MM_SS'),',');
figure(app.UIFigure);
if newfile == 0
    return;
end
filename=fullfile(path,newfile);
app.dataFile=fopen(filename, 'w+');
lat0=app.myGSCoord(1);
lon0 = app.myGSCoord(2);
h0 = app.myGSCoord(3);

% Serial for the radio communication or file
app.s = serial(app.RxCOMPortDropDown.Value);

% Set serial parameters
app.s.InputBufferSize = 15000000;
set(app.s, 'DataBits', 8);
set(app.s, 'StopBits', 1);
set(app.s, 'BaudRate', 115200);
set(app.s, 'Parity', 'none');

% Open the serial port
try
  fopen(app.s);
  app.trackingStat.Color = 'green';
  disp('open')
catch err
  fclose(app.s);
  warndlg('Make sure you select the correct Radio COM Port.');
end

binary_file = app.dataFile;

% Variables for packet counting
rcvd_packets = 0;
first_rcvd = 0;
firstFlag = 1;
prev_packNum = 0;

% Previous coordinates
latprev = zeros(1,6);
lonprev = zeros(1,6);
altprev = zeros(1,6);
timeprev = zeros(1,6);

% PLOT limits
minEvel = -20; maxEvel = 40;
minDvel = -20; maxDvel = 10;
minNvel = -20; maxNvel = 40;
minHvel = -20; maxHvel = 40;

% Expected packet lengths
TOTAL_PACKET_LENGTH = 23; % 19 byte packet + 4 byte rssi info from gs feather
messages = zeros(TOTAL_PACKET_LENGTH, 1, 'double');

% Read and Process Data
app.loopBreak = 0;
dataInterval = 1; 
numPayloads = 1; 
T = tic; 
idx = 1; 
timeout = 1; 

% clear the buffer 
while(app.s.BytesAvailable) 
    fread(app.s,app.s.BytesAvailable); 
end 

while(~app.loopBreak) 
    if (app.s.BytesAvailable > (TOTAL_PACKET_LENGTH) -1) 
        % Save data to binary file 
        read_packet = fread(app.s,TOTAL_PACKET_LENGTH)'; 
        fwrite(binary_file, read_packet); 
        rcvd_packets = rcvd_packets+1; 
        % Make sure header is correct 
        tmpheader = read_packet(1:2); 
        tmpheader = typecast(uint8(tmpheader), 'uint16' ); 
        header = bitand(tmpheader,0xFFF0); 
        switch header 
        case 43664 
            id = 1; 
        case 43680 
            id = 2; 
        case 43696 
            id = 3; 
        case 43712 
            id = 4; 
        case 43728 
            id = 5; 
        case 43744 
            id = 6; 
        case 43760 
            id = 7; 
        otherwise 
            id = 0; 
        end 

Parse Packet 

if id ~= 0 
    %replace first tab
if ~ismember(id, app.payloadID) && length(app.payloadID) == 0
    set(app.ID1Tab, 'Title', ['ID# ' num2str(id)]);
    app.payloadID = [app.payloadID id];
% create new tab
elseif ~ismember(id, app.payloadID)
    createNewTab(app, id);
    app.payloadID(app.payloadID == id) = id;
end

packNum = typecast(uint8(read_packet(3:4)), 'uint16');
alt = typecast(uint8(read_packet(5:8)), 'single');
lat = typecast(uint8(read_packet(9:12)), 'single');
lon = typecast(uint8(read_packet(13:16)), 'single');
utcSec = read_packet(17);
utcMin = read_packet(18);
utcHour = read_packet(19);
numSats = double(bitand(tmpheader, 0x000F));
rssi = typecast(uint8(read_packet(20:23)), 'int32');

Update Display

app.packetnum(app.payloadID == id).Text = num2str(packNum);
app.RSSI(app.payloadID == id).Text = num2str(rssi);

app.curpacknum.Text = ['Current Packet Number: ', num2str(packNum)];
app.ReceivedPackets.Text = ['Received Packets: ', num2str(rcvd_packets)];
app.ExpectedPackets.Text = ['Expected Packets: ', num2str(packNum + 1)];
received_percent = 100 - 100*(rcvd_packets/(packNum));
app.TotalLossLabel.Text = ['% Total Loss: ', num2str(received_percent)];

if numSats >= 3
Calculate velocity north, east, and down

\[
[-,-,range] = \text{geodetic2aer}(lat, \text{lon}, alt, lat0, \text{lon0}, h0, \text{app.spheroid});
\]

% Get Time difference
time = utcHour*3600 + utcMin*60 + utcSec;
dtime = time - timeprev(id);

% Calculate velocities
Rearth = 6378100;   % m
velE(id) = (deg2rad(lon - lonprev(id)) * (alt + Rearth))/dtime;
velN(id) = (deg2rad(lat - latprev(id)) * (alt + Rearth))/dtime;
velD(id) = (alt - altprev(id))/dtime;

% Get min and max values for plotting
limits
% East velocity
if  ((velE(id) > maxEvel) && (velE(id) < 100))
    maxEvel = velE(id);
elseif  ((velE(id) < minEvel) && (velE(id) > -100))
    minEvel = velE(id);
end
% North
if  ((velN(id) > maxNvel) && (velN(id) < 100))
    maxNvel = velN(id);
elseif  ((velN(id) < minNvel) && (velN(id) > -100))
    minNvel = velN(id);
end
% Down
if  ((velD(id) > maxDvel) && (velD(id) < 100))
    maxDvel = velD(id);
elseif  ((velD(id) < minDvel) && (velD(id) > -100))
    minDvel = velD(id);
end
% Horizontal
if ((horiVel(id) > maxHvel) && (horiVel(id) < 100))
    maxHvel = horiVel(id);
elseif ((horiVel(id) < minHvel) && (horiVel(id) > -100))
    minHvel = horiVel(id);
end

latprev(id) = lat;
lonprev(id) = lon;
altprev(id) = alt;
timeprev(id) = time;

% Print Current GPS Data
app.gpsALT(app.payloadID == id).Text = num2str(alt);
app.gpsLAT(app.payloadID == id).Text = num2str(lat);
app.gpsLONG(app.payloadID == id).Text = num2str(lon);
app.gpsSATS(app.payloadID == id).Text = num2str(numSats);
app.gpsFIX(app.payloadID == id).Text = num2str(stat);
app.gpsHOUR(app.payloadID == id).Text = num2str(utcHour);;
app.gpsMIN(app.payloadID == id).Text = num2str(utcMin);
app.gpsSEC(app.payloadID == id).Text = num2str(utcSec);
app.northRate(app.payloadID == id).Text = num2str(velN(id));
app.eastRate(app.payloadID == id).Text = num2str(velE(id));
app.vertRate(app.payloadID == id).Text = num2str(velD(id));
app.horizontalRate(app.payloadID == id).Text = num2str(horiVel(id));
app.range(app.payloadID == id).Text = num2str(range);

% Plot Ublox GPS Position
% [blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black'];
plot3(app.Location3D, lon, lat, 
alt/1000, '.', 'Color', app.color(id), 'LineWidth', 1, 'MarkerSize', 10)
plot(app.Location2D, lon, lat, '.', 'Color', app.color(id), 'MarkerSize', 10, 'LineWidth', 1)

% Plot Calculated Velocity Values
plot(app.VerticalAscentRate, velD,
alt/1000 , ', ', 'Color', app.color(id), 'MarkerSize', 10);
plot(app.NorthVelocity, velN,
alt/1000 , ', ', 'Color', app.color(id), 'MarkerSize', 10);
plot(app.EastVelocity,
velE, alt/1000 , ', ', 'Color', app.color(id), 'MarkerSize', 10);
plot(app.HorizontalVelocity,
horiVel, alt/1000 , ', ', 'Color', app.color(id), 'MarkerSize', 10);

xlim(app.VerticalAscentRate, [minDvel
maxDvel]);
xlim(app.EastVelocity, [minEvel
maxEvel]);
xlim(app.NorthVelocity, [minNvel
maxNvel]);
xlim(app.HorizontalVelocity, [minHvel
maxHvel]);

end  % numsats >=3  
%
numPayloads = numel(findall(app.payloadPositions, 'type', 'uitab'));
%
if numPayloads > 1
  if idx == numPayloads
    idx = 1;
  else
    idx = idx + 1;
  end
end  
%
end  % Header check  
end  % Bytes available  
pause(0.02)  
end  %while loop  
%
app.trackingStat.Color = 'white';  
fclose(app.s);  

end  
%
Callback function

function ConnectGSButtonPushed(app, event)
  % Initialize Serial Communication with Arduino and MATLAB.  
  % The Arduino sends a Char and waits for MATLAB to respond with the proper
  % Char. If no errors, setup ok indication is visible.

  app.serial_GS = serial(app.GSCOMPortDropDown.Value);

  %
  set(app.ConnectingLabel, 'Visible', 'on');  

  %Set serial parameters
app.serial_GS.InputBufferSize = 300000;
set(app.serial_GS, 'DataBits', 8);
set(app.serial_GS, 'StopBits', 1);
set(app.serial_GS, 'BaudRate', 230400);
set(app.serial_GS, 'Parity', 'none');

%Open the serial port
try
    fopen(app.serial_GS);
    set(app.gsPortStatus, 'Color', 'g');
catch err
    fclose(app.serial_GS);
    set(app.gsPortStatus, 'Color', 'r');
    error('Make sure you select the correct Arduino COM Port.');
end

app.flag_GS = 1;

set(app.ConnectGSButton, 'Enable', 'off');

while (app.serial_GS.BytesAvailable == 0)
    a=fscanf(app.serial_GS, '%e');
    fprintf(app.serial_GS, '%s
', 'getAz');
    while (app.serial_GS.BytesAvailable == 0)
        app.AzGauge.Value = fscanf(app.serial_GS, '%e');
        fprintf(app.serial_GS, '%s
', 'getEl');
        while (app.serial_GS.BytesAvailable == 0)
            app.ElevationGauge.Value = fscanf(app.serial_GS, '%e');
            % set(app.ConnectingLabel, 'Visible', 'off');
            % app.GroundStationTestButton.Enable = 'on';
            app.BeginTrackingButton.Enable = 'on';
end

% Close request function: UIFigure
function UIFigureCloseRequest(app, event)
    delete(instrfindall);
    delete(app)
end
% Callback function
function GSCOMPortDropDownValueChanged(app, event)
    %
cellstr(seriallist);
    app.RxCOMPortDropDown.Items = app.GSCOMPortDropDown.Items;
    end

% Value changed function: RxCOMPortDropDown
function RxCOMPortDropDownValueChanged(app, event)
    app.RxCOMPortDropDown.Items = cellstr(seriallist);
    %
    app.GSCOMPortDropDown.Items = app.RxCOMPortDropDown.Items;
end

% Menu selected function: RefreshCOMPortsMenu
function RefreshCOMPortsMenuSelected(app, event)
    app.RxCOMPortDropDown.Items = cellstr(seriallist);
    %
    app.GSCOMPortDropDown.Items = app.RxCOMPortDropDown.Items;
end

% Button pushed function: EndTrackingButton
function EndTrackingButtonPushed(app, event)
    prompt = {'Stop reading in data?'};
    title = 'Are you sure?';
    answ = questdlg(prompt, title, 'Yes', 'No', 'No');
    app.loopBreak = 1;
    switch answ
    case 'Yes'
        % Serial for the radio communication or file
        fclose(app.s);

        %
        app.BeginTrackingButton.Enable = 'on';
        % app.EndTrackingButton.Enable = 'off';
        %
        app.ListeningLamp.Color = 'white';
        %
        app.UseXBtosetGSPositionButton.Enable = 'off';
    end
end

% Button pushed function: ReplayButton
function ReplayButtonPushed(app, event)
    [newfile, path] = uigetfile('*.bin', 'Create Data File', 'data.bin');
    if newfile == 0
        return;
    end
    filename = fullfile(path, newfile);
    s1 = fopen(filename, 'r+');
lat0 = app.myGSCoord(1);
lon0 = app.myGSCoord(2);
h0 = app.myGSCoord(3);

app.ReplaySpeedEditField.Visible = true;
app.ReplaySpeedEditFieldLabel.Visible = true;

% Variables for packet counting
rcvd_packets = 0;
first_rcvd = 0;
firstFlag = 1;
prev_packNum = 0;

% Initial Position
lat = lat0;
lon = lon0;
alt = h0;

% previous coordinates
latprev = zeros(1,6);
lonprev = zeros(1,6);
altprev = zeros(1,6);
timeprev = zeros(1,6);

% PLOT limits
minEvel = -20; maxEvel = 40;
minDvel = -20; maxDvel = 10;
minNvel = -20; maxNvel = 40;
minHvel = -20; maxHvel = 40;
maxAlt = 5;

% Expected packet lengths
TOTAL_PACKET_LENGTH = 23; % 44 byte packet + 4 byte rssi

info from gs feather

dataInterval = 1;
umPayloads = 1;
T = tic;
idx = 1;
timeout = 1;

% Read and Process Data
data = fread(s1);
for i = 1:TOTAL_PACKET_LENGTH*7:(length(data) - TOTAL_PACKET_LENGTH)
    pause(str2double(app.ReplaySpeedEditField.Value));
    read_packet = data(i:i+TOTAL_PACKET_LENGTH-1);
% Make sure header is correct
tmpheader = read_packet(1:2);
tmpheader = typecast(uint8(tmpheader), 'uint16');
header = bitand(tmpheader, 0xFFF0);
switch header
    case 43664
        id = 1;
    case 43680
        id = 2;
    case 43696
        id = 3;
    case 43712
        id = 4;
    case 43728
        id = 5;
    case 43744
        id = 6;
    case 43760
        id = 7;
    otherwise
        id = 0;
end

Parse Packet

if id ~= 0

%replace first tab
if ~ismember(id, app.payloadID) &&
    length(app.payloadID) == 0
    set(app.ID1Tab,'Title',['ID# ' num2str(id)]);
    app.payloadID = [app.payloadID id];
end

elseif ~ismember(id, app.payloadID)
    createNewTab(app,id);
    app.payloadID(app.payloadID == id) = id;
end

packNum =
    typecast(uint8(read_packet(3:4)), 'uint16');

alt = typecast(uint8(read_packet(5:8)), 'single');
lat = typecast(uint8(read_packet(9:12)), 'single');
lon =
    typecast(uint8(read_packet(13:16)), 'single');
utcSec = read_packet(17);
utcMin = read_packet(18);
utcHour = read_packet(19);
umSats = double(bitand(tmpheader,0x000F));
rssi =
    typecast(uint8(read_packet(20:23)), 'int32');
Update Display

app.packetnum(app.payloadID == id).Text = num2str(packNum);
app.RSSI(app.payloadID == id).Text = num2str(rssi);

app.curpacknum.Text = ['Current Packet Number: ', num2str(packNum)];

app.ReceivedPackets.Text = ['Received Packets: ', num2str(rcvd_packets)];
app.ExpectedPackets.Text = ['Expected Packets: ', num2str(packNum + 1)];
received_percent = 100 - 100*(rcvd_packets/(packNum));
app.TotalLossLabel.Text = ['% Total Loss: ', num2str(received_percent)];

if numSats >= 3
    stat = 3;
end

Calculate velocity north, east, and down

[~,~,range] = geodetic2aer(lat,lon,alt,lat0,lon0,h0,app.spheroid);

time = utcHour*3600 + utcMin*60 + utcSec;
dtime = time - timeprev(id);

Rearth = 6378100;  %m
velE(id) = (deg2rad(lon - lonprev(id)) * (alt + Rearth))/dtime;
velN(id) = (deg2rad(lat - latprev(id)) * (alt + Rearth))/dtime;
horiVel(id) = sqrt(velE(id)^2 + velN(id)^2);
velD(id) = (alt - altprev(id))/dtime;

% Get min and max values for plotting limits
% East velocity
if ((velE(id) > maxEvel) && (velE(id) < -100))
    maxEvel = velE(id);
elseif ((velE(id) < minEvel) && (velE(id) > -100))
    minEvel = velE(id);
end
% North
if ((velN(id) > maxNvel) && (velN(id) < 100))
    maxNvel = velN(id);
elseif ((velN(id) < minNvel) && (velN(id) > -100))
    minNvel = velN(id);
end

% Down
if ((velD(id) > maxDvel) && (velD(id) < 100))
    maxDvel = velD(id);
elseif ((velD(id) < minDvel) && (velD(id) > -100))
    minDvel = velD(id);
end

% Horizontal
if ((horiVel(id) > maxHvel) && (horiVel(id) < 100))
    maxHvel = horiVel(id);
elseif ((horiVel(id) < minHvel) && (horiVel(id) > -100))
    minHvel = horiVel(id);
end

latprev(id) = lat;
lonprev(id) = lon;
altprev(id) = alt;
timeprev(id) = time;

% Print Current GPS Data
app.gpsALT(app.payloadID == id).Text = num2str(alt);
app.gpsLAT(app.payloadID == id).Text = num2str(lat);
app.gpsLONG(app.payloadID == id).Text = num2str(lon);
app.gpsSATS(app.payloadID == id).Text = num2str(numSats);
app.gpsFIX(app.payloadID == id).Text = num2str(stat);
app.gpsHOUR(app.payloadID == id).Text = num2str(utcHour), ':';
app.gpsMIN(app.payloadID == id).Text = num2str(utcMin), ':';
app.gpsSEC(app.payloadID == id).Text = num2str(utcSec);
app.northRate(app.payloadID == id).Text = num2str(velN(id));
app.eastRate(app.payloadID == id).Text = num2str(velE(id));
app.vertRate(app.payloadID == id).Text = num2str(velD(id));
app.horizontalRate(app.payloadID == id).Text = num2str(horiVel(id));

app.range(app.payloadID == id).Text = num2str(range);

%Plot Ublox GPS Position
%lue','green','red','cyan','magenta','yellow','black']
plot3(app.Location3D, lon, lat, alt/1000, '.', 'Color', app.color(id), 'LineWidth',1,'MarkerSize',10)
plot(app.Location2D, lon, lat, '.', 'Color', app.color(id), 'LineWidth',1,'MarkerSize',10)

% Plot Calculated Velocity Values
plot(app.VerticalAscentRate, velD(id), alt/1000, '.', 'Color', app.color(id), 'MarkerSize',10);
plot(app.NorthVelocity, velN(id), alt/1000, '.', 'Color', app.color(id), 'MarkerSize',10);
plot(app.EastVelocity, velE(id), alt/1000, '.', 'Color', app.color(id), 'MarkerSize',10);
plot(app.HorizontalVelocity, horiVel(id), alt/1000, '.', 'Color', app.color(id), 'MarkerSize',10);

xlim(app.VerticalAscentRate,[-1 10])
xlim(app.EastVelocity,[minEvel maxEvel])
xlim(app.NorthVelocity,[minNvel maxNvel])
xlim(app.HorizontalVelocity,[minHvel maxHvel])

% Set altitude limits
if (alt/1000 > maxAlt)
  maxAlt = maxAlt + 5;
end
ylim(app.VerticalAscentRate,[0 maxAlt])
ylim(app.EastVelocity,[0 maxAlt])
ylim(app.NorthVelocity,[0 maxAlt])
ylim(app.HorizontalVelocity,[0 maxAlt])

end % numsats >=3
%
numPayloads = numel(findall(app.payloadPositions, 'type', 'uitab'));
% if numPayloads > 1
%   if idx ==
numPayloads
   idx = 1;
else
   idx = idx + 1;
end
%
% Button pushed function: RemovePayloadButton
function RemovePayloadButtonPushed(app, event)
    id = str2double(cell2mat(inputdlg("Enter ID of Payload to Remove:", "Payload ID")));  
    idx = strfind(app.payloadID, id);
    app.payloadID(idx) = [];
    delete(app.handles.tabs(idx))
end

% Component initialization
methods (Access = private)
function createComponents(app)
    % Create UIFigure and hide until all components are created
    app.UIFigure = uifigure('Visible', 'off');
    app.UIFigure.Position = [0 0 1009 664];
    app.UIFigure.Name = 'MATLAB App';
    app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);
    app.UIFigure.Scrollable = 'on';
    % Create Location2D
    app.Location2D = uiaxes(app.UIFigure);
    title(app.Location2D, '2D position');
    xlabel(app.Location2D, 'Lon');
    ylabel(app.Location2D, 'Lat');
    app.Location2D.DataAspectRatio = [1 1 1];
A.2. GROUND STATION SOFTWARE

app.Location2D.PlotBoxAspectRatio = [1 1 1];
app.Location2D.FontSize = 11;
app.Location2D.YTick = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
0.9 1];
app.Location2D.Color = [0.902 0.902 0.902];
app.Location2D.NextPlot = 'replace';
app.Location2D.Position = [0 345 315 319];

% Create Location3D
app.Location3D = uiaxes(app.UIFigure);
title(app.Location3D, '3D position')
xlabel(app.Location3D, 'Lon')
ylabel(app.Location3D, 'Lat')
app.Location3D.PlotBoxAspectRatio = [1.00470588235294 1
1]);
app.Location3D.FontSize = 11;
app.Location3D.Box = 'on';
app.Location3D.Color = [0.9412 0.9412 0.9412];
app.Location3D.NextPlot = 'replace';
app.Location3D.XGrid = 'on';
app.Location3D.YGrid = 'on';
app.Location3D.ZGrid = 'on';
app.Location3D.Position = [0 10 315 319];

% Create VerticalAscentRate
app.VerticalAscentRate = uiaxes(app.UIFigure);
title(app.VerticalAscentRate, 'Alt vs. Vertical Ascent Rate')
xlabel(app.VerticalAscentRate, 'Vertical Ascent Rate (m/ s)')
ylabel(app.VerticalAscentRate, 'Altitude (km)')
zlabel(app.VerticalAscentRate, '')
app.VerticalAscentRate.XLim = [-40 10];
app.VerticalAscentRate.YLim = [0 40];
app.VerticalAscentRate.XTick = [-1 0 5 10];
app.VerticalAscentRate.NextPlot = 'add';
app.VerticalAscentRate.Position = [317 458 329 205];

% Create NorthVelocity
app.NorthVelocity = uiaxes(app.UIFigure);
title(app.NorthVelocity, 'Alt vs. Northward Velocity')
xlabel(app.NorthVelocity, 'Northward Velocity (m/s)')
ylabel(app.NorthVelocity, 'Altitude (km)')
zlabel(app.NorthVelocity, '')
app.NorthVelocity.YLim = [0 40];
app.NorthVelocity.XTick = [-20 -10 0 10 20 30 40];
app.NorthVelocity.XTickLabel = {
'-'20'; '-10'; '0'; '10'; '20'; '30'; '40'};
app.NorthVelocity.NextPlot = 'add';
app.NorthVelocity.Position = [669 458 329 205];

% Create EastVelocity
app.EastVelocity = uiaxes(app.UIFigure);
title(app.EastVelocity, 'Alt vs. Eastward Velocity')
A.2. GROUND STATION SOFTWARE

```matlab
xlabel(app.EastVelocity, 'Eastward Velocity (m/s)')
ylabel(app.EastVelocity, 'Altitude (km)')
zlabel(app.EastVelocity, '')
app.EastVelocity.YLim = [0 40];
app.EastVelocity.XTick = [-20 -10 0 10 20 30 40];
app.EastVelocity.XTickLabel = {'-20'; '-10'; '0'; '10'; '20'; '30'; '40'};
app.EastVelocity.NextPlot = 'add';
app.EastVelocity.Position = [669 224 329 205];

% Create HorizontalVelocity
app.HorizontalVelocity = uiaxes(app.UIFigure);
title(app.HorizontalVelocity, 'Alt vs. Horizontal Velocity')
xlabel(app.HorizontalVelocity, 'Horizontal Velocity (m/s)')
ylabel(app.HorizontalVelocity, 'Altitude (km)')
zlabel(app.HorizontalVelocity, '')
app.HorizontalVelocity.YLim = [0 40];
app.HorizontalVelocity.XTick = [-20 -10 0 10 20 30 40];
app.HorizontalVelocity.XTickLabel = {'-20'; '-10'; '0'; '10'; '20'; '30'; '40'};
app.HorizontalVelocity.NextPlot = 'add';
app.HorizontalVelocity.Position = [317 224 329 205];

% Create Menu
app.Menu = uimenu(app.UIFigure);
app.Menu.Text = 'Menu';

% Create RefreshCOMPortsMenu
app.RefreshCOMPortsMenu = uimenu(app.Menu);
app.RefreshCOMPortsMenu.MenuSelectedFcn = createCallbackFcn(app, @RefreshCOMPortsMenuSelected, true);
app.RefreshCOMPortsMenu.Text = 'Refresh COM Ports';

% Create LoadMapsMenu
app.LoadMapsMenu = uimenu(app.Menu);
app.LoadMapsMenu.MenuSelectedFcn = createCallbackFcn(app, @LoadPositionButtonPushed, true);
app.LoadMapsMenu.Text = 'Load Maps';

% Create LoadPredictionFileMenu
app.LoadPredictionFileMenu = uimenu(app.Menu);
app.LoadPredictionFileMenu.Text = 'Load Prediction File';

% Create BeginTrackingButton
app.BeginTrackingButton = uibutton(app.UIFigure, 'push');
app.BeginTrackingButton.ButtonPushedFcn = createCallbackFcn(app, @BeginTrackingButtonPushed, true);
app.BeginTrackingButton.BackgroundColor = [0.5882 0.9608 0.4784];
app.BeginTrackingButton.Position = [351 161 100 22];
app.BeginTrackingButton.Text = 'Begin Tracking';
```
% Create EndTrackingButton
app.EndTrackingButton = uibutton(app.UIFigure, 'push');
createCallbackFcn(app, @EndTrackingButtonPushed, true);
app.EndTrackingButton.BackgroundColor = [0.9608 0.6 0.6];
app.EndTrackingButton.Position = [351 131 100 22];
app.EndTrackingButton.Text = 'End Tracking';

% Create trackingStat
app.trackingStat = uilamp(app.UIFigure);
app.trackingStat.Position = [457 162 20 20];
app.trackingStat.Color = [1 0 0];

% Create RxCOMPortDropDownLabel
app.RxCOMPortDropDownLabel = uilabel(app.UIFigure);
app.RxCOMPortDropDownLabel.HorizontalAlignment = 'right'
app.RxCOMPortDropDownLabel.VerticalAlignment = 'top'
app.RxCOMPortDropDownLabel.Position = [353 192 76 15];
app.RxCOMPortDropDownLabel.Text = 'Rx COM Port';

% Create RxCOMPortDropDown
app.RxCOMPortDropDown = uidropdown(app.UIFigure);
app.RxCOMPortDropDown.Items = {};
createCallbackFcn(app, @RxCOMPortDropDownValueChanged, true);
app.RxCOMPortDropDown.Position = [440 188 61 22];
app.RxCOMPortDropDown.Value = {};

% Create ReplayButton
app.ReplayButton = uibutton(app.UIFigure, 'push');
createCallbackFcn(app, @ReplayButtonPushed, true);
app.ReplayButton.Position = [351 98 122 22];
app.ReplayButton.Text = 'Replay';

% Create ReplaySpeedEditField
app.ReplaySpeedEditField = uieditfield(app.UIFigure, 'text');
app.ReplaySpeedEditField.HorizontalAlignment = 'center'
app.ReplaySpeedEditField.Visible = 'off'
app.ReplaySpeedEditField.Position = [373 52 80 22];
app.ReplaySpeedEditField.Value = '0.15';

% Create ReplaySpeedEditFieldLabel
app.ReplaySpeedEditFieldLabel = uilabel(app.UIFigure);
app.ReplaySpeedEditFieldLabel.HorizontalAlignment = 'right'
app.ReplaySpeedEditFieldLabel.Visible = 'off'
app.ReplaySpeedEditFieldLabel.Position = [374 73 81 22];
app.ReplaySpeedEditFieldLabel.Text = 'Replay Speed';

% Create payloadPositions
app.payloadPositions = uitabgroup(app.UIFigure);
app.payloadPositions.Position = [539 44 406 173];
% Create ID1Tab
app.ID1Tab = uitab(app.payloadPositions);
app.ID1Tab.Title = 'ID #1';

% Create gpsLAT
app.gpsLAT = uilabel(app.ID1Tab);
app.gpsLAT.BackgroundColor = [1 1 1];
app.gpsLAT.HorizontalAlignment = 'right';
app.gpsLAT.VerticalAlignment = 'top';
app.gpsLAT.Position = [22 97 61 15];
app.gpsLAT.Text = ''; 

% Create gpsLONG
app.gpsLONG = uilabel(app.ID1Tab);
app.gpsLONG.BackgroundColor = [1 1 1];
app.gpsLONG.HorizontalAlignment = 'right';
app.gpsLONG.VerticalAlignment = 'top';
app.gpsLONG.Position = [87 97 61 15];
app.gpsLONG.Text = ''; 

% Create gpsALT
app.gpsALT = uilabel(app.ID1Tab);
app.gpsALT.BackgroundColor = [1 1 1];
app.gpsALT.HorizontalAlignment = 'right';
app.gpsALT.VerticalAlignment = 'top';
app.gpsALT.Position = [153 97 61 15];
app.gpsALT.Text = ''; 

% Create gpsFIX
app.gpsFIX = uilabel(app.ID1Tab);
app.gpsFIX.BackgroundColor = [1 1 1];
app.gpsFIX.HorizontalAlignment = 'right';
app.gpsFIX.VerticalAlignment = 'top';
app.gpsFIX.Position = [225 97 25 15];
app.gpsFIX.Text = ''; 

% Create LatitudeLabel
app.LatitudeLabel = uilabel(app.ID1Tab);
app.LatitudeLabel.HorizontalAlignment = 'center';
app.LatitudeLabel.VerticalAlignment = 'top';
app.LatitudeLabel.Position = [22 119 60 15];
app.LatitudeLabel.Text = 'Latitude'; 

% Create LongitudeLabel
app.LongitudeLabel = uilabel(app.ID1Tab);
app.LongitudeLabel.HorizontalAlignment = 'center';
app.LongitudeLabel.Position = [88 119 58 15];
app.LongitudeLabel.Text = 'Longitude'; 

% Create AltitudeLabel
app.AltitudeLabel = uilabel(app.ID1Tab);
app.AltitudeLabel.HorizontalAlignment = 'center';
app.AltitudeLabel.VerticalAlignment = 'top';
app.AltitudeLabel.Position = [153 117 61 18];
app.AltitudeLabel.Text = 'Altitude';

% Create FixQLabel
app.FixQLabel = uilabel(app.ID1Tab);
app.FixQLabel.HorizontalAlignment = 'center';
app.FixQLabel.VerticalAlignment = 'top';
app.FixQLabel.Position = [220 120 37 15];
app.FixQLabel.Text = 'Fix Q.';

% Create gpsSATS
app.gpsSATS = uilabel(app.ID1Tab);
app.gpsSATS.BackgroundColor = [1 1 1];
app.gpsSATS.HorizontalAlignment = 'right';
app.gpsSATS.VerticalAlignment = 'top';
app.gpsSATS.Position = [262 97 25 15];
app.gpsSATS.Text = '';

% Create gpsHOUR
app.gpsHOUR = uilabel(app.ID1Tab);
app.gpsHOUR.BackgroundColor = [1 1 1];
app.gpsHOUR.HorizontalAlignment = 'right';
app.gpsHOUR.VerticalAlignment = 'top';
app.gpsHOUR.Position = [299 97 25 15];
app.gpsHOUR.Text = '';

% Create gpsMIN
app.gpsMIN = uilabel(app.ID1Tab);
app.gpsMIN.BackgroundColor = [1 1 1];
app.gpsMIN.HorizontalAlignment = 'right';
app.gpsMIN.VerticalAlignment = 'top';
app.gpsMIN.Position = [329 97 25 15];
app.gpsMIN.Text = '';

% Create gpsSEC
app.gpsSEC = uilabel(app.ID1Tab);
app.gpsSEC.BackgroundColor = [1 1 1];
app.gpsSEC.HorizontalAlignment = 'right';
app.gpsSEC.VerticalAlignment = 'top';
app.gpsSEC.Position = [359 97 25 15];
app.gpsSEC.Text = '';

% Create SatsLabel
app.SatsLabel = uilabel(app.ID1Tab);
app.SatsLabel.HorizontalAlignment = 'center';
app.SatsLabel.VerticalAlignment = 'top';
app.SatsLabel.Position = [256 119 37 15];
app.SatsLabel.Text = '#Sats';

% Create GPSUTCTimeLabel
app.GPSUTCTimeLabel = uilabel(app.ID1Tab);
app.GPSUTCTimeLabel.HorizontalAlignment = 'center';
app.GPSUTCTimeLabel.VerticalAlignment = 'top';
app.GPSUTCTimeLabel.Position = [299 119 85 15];
app.GPSUTCTimeLabel.Text = 'GPS UTC Time';

% Create vertRate
app.vertRate = uilabel(app.ID1Tab);
app.vertRate.BackgroundColor = [1 1 1];
app.vertRate.HorizontalAlignment = 'right';
app.vertRate.VerticalAlignment = 'top';
app.vertRate.Position = [57 52 50 15];
app.vertRate.Text = ''; 

% Create vertRateLabel
app.vertRateLabel = uilabel(app.ID1Tab);
app.vertRateLabel.HorizontalAlignment = 'center';
app.vertRateLabel.VerticalAlignment = 'top';
app.vertRateLabel.Position = [45 67 74 22];
app.vertRateLabel.Text = 'Vertical Rate';

% Create northRate
app.northRate = uilabel(app.ID1Tab);
app.northRate.BackgroundColor = [1 1 1];
app.northRate.HorizontalAlignment = 'right';
app.northRate.VerticalAlignment = 'top';
app.northRate.Position = [117 52 64 15];
app.northRate.Text = ''; 

% Create NorthRateLabel
app.NorthRateLabel = uilabel(app.ID1Tab);
app.NorthRateLabel.HorizontalAlignment = 'center';
app.NorthRateLabel.Position = [117 74 64 14];
app.NorthRateLabel.Text = 'North Rate';

% Create EastRateLabel
app.EastRateLabel = uilabel(app.ID1Tab);
app.EastRateLabel.HorizontalAlignment = 'center';
app.EastRateLabel.Position = [196 74 58 14];
app.EastRateLabel.Text = 'East Rate';

% Create eastRate
app.eastRate = uilabel(app.ID1Tab);
app.eastRate.BackgroundColor = [1 1 1];
app.eastRate.HorizontalAlignment = 'right';
app.eastRate.VerticalAlignment = 'top';
app.eastRate.Position = [193 52 64 15];
app.eastRate.Text = ''; 

% Create HorizontalRateLabel
app.HorizontalRateLabel = uilabel(app.ID1Tab);
app.HorizontalRateLabel.HorizontalAlignment = 'center';
app.HorizontalRateLabel.Position = [269 75 88 13];
app.HorizontalRateLabel.Text = 'Horizontal Rate';

% Create horizontalRate
app.horizontalRate = uilabel(app.ID1Tab);
app.horizontalRate.BackgroundColor = [1 1 1];
app.horizontalRate.HorizontalAlignment = 'center';
app.horizontalRate.Position = [269 52 64 15];
app.horizontalRate.Text = '';
app.horizontalRate.HorizontalAlignment = 'right';
app.horizontalRate.VerticalAlignment = 'top';
app.horizontalRate.Position = [269 52 88 15];
app.horizontalRate.Text = ''; 

% Create RSSILabel
app.RSSILabel = uilabel(app.ID1Tab);
app.RSSILabel.HorizontalAlignment = 'center';
app.RSSILabel.Position = [216 22 51 22];
app.RSSILabel.Text = 'RSSI';

% Create RSSI
app.RSSI = uilabel(app.ID1Tab);
app.RSSI.BackgroundColor = [1 1 1];
app.RSSI.HorizontalAlignment = 'right';
app.RSSI.VerticalAlignment = 'top';
app.RSSI.Position = [216 8 51 15];
app.RSSI.Text = ''; 

% Create BatteryLabel
app.BatteryLabel = uilabel(app.ID1Tab);
app.BatteryLabel.HorizontalAlignment = 'center';
app.BatteryLabel.Position = [279 22 51 22];
app.BatteryLabel.Text = 'Battery';

% Create battVolt
app.battVolt = uilabel(app.ID1Tab);
app.battVolt.BackgroundColor = [1 1 1];
app.battVolt.HorizontalAlignment = 'right';
app.battVolt.VerticalAlignment = 'top';
app.battVolt.Position = [279 8 51 15];
app.battVolt.Text = ''; 

% Create packetnum
app.packetnum = uilabel(app.ID1Tab);
app.packetnum.BackgroundColor = [1 1 1];
app.packetnum.HorizontalAlignment = 'right';
app.packetnum.VerticalAlignment = 'top';
app.packetnum.Position = [80 8 51 15];
app.packetnum.Text = ''; 

% Create range
app.range = uilabel(app.ID1Tab);
app.range.BackgroundColor = [1 1 1];
app.range.HorizontalAlignment = 'right';
app.range.VerticalAlignment = 'top';
app.range.Position = [143 8 51 15];
app.range.Text = ''; 

% Create PacketLabel
app.PacketLabel = uilabel(app.ID1Tab);
app.PacketLabel.HorizontalAlignment = 'center';
app.PacketLabel.Position = [80 22 52 22];
app.PacketLabel.Text = 'Packet #';
% Create RangeLabel
app.RangeLabel = uilabel(app.ID1Tab);
app.RangeLabel.HorizontalAlignment = 'center';
app.RangeLabel.Position = [143 22 51 22];
app.RangeLabel.Text = 'Range';

% Create Lamp
app.Lamp = uilamp(app.ID1Tab);
app.Lamp.Position = [2 127 20 20];
app.Lamp.Color = [0 0 1];

% Create RemovePayloadButton
app.RemovePayloadButton = uibutton(app.UIFigure, 'push');
app.RemovePayloadButton.ButtonPushedFcn = createCallbackFcn(app, @RemovePayloadButtonPushed, true);
app.RemovePayloadButton.Visible = 'off';
app.RemovePayloadButton.Position = [781 14 107 22];
app.RemovePayloadButton.Text = 'Remove Payload';

% Show the figure after all components are created
app.UIFigure.Visible = 'on';
end
end

% App creation and deletion
methods (Access = public)

% Construct app
function app = HAB_GPSonde_GS_V5
    % Create UIFigure and components
    createComponents(app)
    % Register the app with App Designer
    registerApp(app, app.UIFigure)
    % Execute the startup function
    runStartupFcn(app, @startupFcn)
    if nargout == 0
        clear app
    end
end

% Code that executes before app deletion
function delete(app)
    % Delete UIFigure when app is deleted
    delete(app.UIFigure)
end
end
Error using dbstatus
Error: File: C:\Users\Julio\OneDrive\MURI Balloon\Personal Folders\JulioG\Software\Ground Station\HAB_GPSonde_GS_V5_exp.m Line: 1
         Column: 10
Class name and filename do not agree.

Published with MATLAB® R2019b
Appendix B

HAB Bus and CDU

B.1 Bus Flight Software
B.1. BUS FLIGHT SOFTWARE

Main

Julio Guardado

1 /***************************************************************************
2     NAME: HAB_Parallel_Payload.ino
3
4     PURPOSE: HIGH ALTITUDE BALLOON - Modular Payload.
5
6     DEVELOPMENT HISTORY:
7     Date      Author    Version                  Description Of Change
8     --------  ------    -------                    ------------------------------


Code written to support parallel processing of sensor data

**********SERIAL DEFINITIONS**********

Startup Monitor Serial
GPS Ublox Serial3
Xbee3 Serial4
Xbee SX Pro' Serial5

/#********* Communications **********/

Serial.begin(115200);  //Serial Monitor
Serial3.begin(9600);    //GPS
Serial4.begin(38400);  //Xbee3
Serial5.begin(230400);  //Radio

Wire.begin(); 
SPI.begin();

}
delay(200);

/************ Instrument Setup **************/
// Sparkfun LSM9DS1 9DOF IMU Setup
if ( IMU_SYS ) {
  setupGyro();
  setupAccel();
  setupMag();
  if (imu.begin() == false) {
    Serial.println("Failed to communicate with LSM9DS1.");
  } else Serial.println("IMU OK");
}

// HiLetGo GY-63 MS5611 Barometer Setup
if ( BARO_SYS ) {
  if (!baro.begin(MS5611_ULTRA_HIGH_RES)) {
    Serial.println(F("Could not find a valid MS5611 sensor, check wiring!")));
  } else Serial.println("BAROMETER OK");
  baroReferencePressure = baro.readPressure();                    // Get a reference
  pressure for relative altitude
}

// Arducam mini 2mp plus Setup
if ( CAM_SYS ) {
  setupCam();
} else {
  for (int k = IMG_DATA_START_INDEX; k < 254; k++) {
    radio_packet[k] = 0;
  }
}

// Internal ADC Setup
analogReadResolution(10);
adc->setReference(ADC_REFERENCE::REF_3V3, ADC_0);

// uBlox HYGPSV5-NEO M8N Setup
initGPS();

/*********SD CARD*************/
sdCount = 0;
while (!SD.begin(chipSelect)) {
  Serial.println("Insert SD Card");
  delay(200);
}
Serial.println("SD OK");

//Read and set file number and name
address = 0;
fileNum = EEPROM.read(address);
fileNum = fileNum + 1;
EEPROM.write(address, fileNum);
fileName = "data";
String m1 = fileName + fileNum;
fileName = m1 + ext;

//Open file
flightData = SD.open(fileName1.c_str(), FILE_WRITE);

/******* Thread Definitions *******/
threads.addThread(updateSensors);

3/5
68 delay(200);
69
70 /************ Instrument Setup *************/
71 // Sparkfun LSM9DS1 9DOF IMU Setup
72 if ( IMU_SYS ) {
73 setupGyro();
74 setupAccel();
75 setupMag();
76 if (imu.begin() == false) {
77   Serial.println("Failed to communicate with LSM9DS1.");
78 } else Serial.println("IMU OK");
79 }
80
81
82 // HiLetGo GY-63 MS5611 Barometer Setup
83 if ( BARO_SYS ) {
84 if (!baro.begin(MS5611_ULTRA_HIGH_RES)) {
85   Serial.println(F("Could not find a valid MS5611 sensor, check wiring!")));
86 } else Serial.println("BAROMETER OK");
87  baroReferencePressure = baro.readPressure();                    // Get a reference
88  pressure for relative altitude
89 }
90
91
92 // Arducam mini 2mp plus Setup
93 if ( CAM_SYS ) {
94 setupCam();
95 } else {
96 for (int k = IMG_DATA_START_INDEX; k < 254; k++) {
97   radio_packet[k] = 0;
98 }
99 }
100
101
102 // Internal ADC Setup
103 analogReadResolution(10);
104 adc->setReference(ADC_REFERENCE::REF_3V3, ADC_0);
105
106 // uBlox HYGPSV5-NEO M8N Setup
107 initGPS();
108
109 /*********SD CARD*************/
110 sdCount = 0;
111 while (!SD.begin(chipSelect)) {
112 Serial.println("Insert SD Card");
113 delay(200);
114 }
115 Serial.println("SD OK");
116
117 //Read and set file number and name
118 address = 0;
119 fileNum = EEPROM.read(address);
120 fileNum = fileNum + 1;
121 EEPROM.write(address, fileNum);
122 fileName = "data";
123 String m1 = fileName + fileNum;
124 fileName1 = m1 + ext;
125
126 //Open file
127 flightData = SD.open(fileName1.c_str(), FILE_WRITE);
128
129
130 /******* Thread Definitions *******/
131 threads.addThread(updateSensors);
threads.addThread(updateGPS);
threads.addThread(valveSysCheck);
if (debug) {
    threads.addThread(serialOut);
    threads.setTimeSlice(4, 30);    // update valve sys
}

// Set timeslice of each function
threads.setTimeSlice(0, 5000);    // main loop
threads.setTimeSlice(1, 50);    // update Sensors
threads.setTimeSlice(2, 50);    // update gps
threads.setTimeSlice(3, 1000);    // update valve sys

// **********WATCHDOG TIMER**********/
// noInterrupts();
// WDOG_UNLOCK = WDOG_UNLOCK_SEQ1;
// WDOG_UNLOCK = WDOG_UNLOCK_SEQ2;
// delayMicroseconds(1);
// WDOG_TOVALH = 1000 / 5; //0x006d;
// WDOG_TOVALL = 0;//0xdd00;
// WDOG_PRESC = 0x400;
// WDOG_STCTRLH |= WDOG_STCTRLH_ALLOWUPDATE |
//                  WDOG_STCTRLH_WDOGEN | WDOG_STCTRLH_WAITEN |
//                  WDOG_STCTRLH_STOPEN | WDOG_STCTRLH_CLKSRC;
// interrupts();

///////////////////////////////////////////////////////////////////////////////////////////////
/// ************* BEGINNING OF MAIN PROGRAM LOOP
///////////////////////////////////////////////////////////////////////////////////////////////

void loop() {
    //*********** GROUND STATION COMMUNICATION ************/
    // Send packets based on camera data
    if (CAM_SYS) {
        if (camUpdate > UPDATE_INTERVAL_CAM && !retransmitFlag) {
            Threads::Scope scope(Wire_lock);
            // Refresh update timer
            camUpdate = 0;
            // take image and send down all data
            im_length = take_img(LR);
            send_img(LR, im_length);
        }
    }

    else {
        // Send Packets based on timer
        if (dataUpdate > UPDATE_INTERVAL_DATA && !retransmitFlag) {
            dataUpdate = 0;
            // update data packet
            write_erau_data();
            // Send data x number of times
            for (int i = 0; i < DATA_ReTX; i++) {
            ```
B.1. BUS FLIGHT SOFTWARE

194     Serial5.write(radio_packet, RADIO_SIZE);
195 }
196 }
197 }
198 /********CHECK FOR RETRANSMISSION PROCEDURE**********/
199 if (!retransmitFlag) {
200     retransmissionCheck();
201 } else {
202     retransmitProcedure();
203 }
204 }
205 }
206 /********CHECK FOR COMMANDS FROM GS***************
207 uplinkCheck();
208 }
209 /********PERIODICALLY SAVE SD CARD***************/
210 if (((millis() - timeSD) > UPDATE_INTERVAL_DATA) && !retransmitFlag) {
211     appendPos = flightData.position();
212     flightData.close();
213     flightData = SD.open(fileName1.c_str(), FILE_WRITE);
214     timeSD = millis();
215     flightData.seek(appendPos);
216 }
217 }
218 }
219     threads.yield();
220 }
// Function to check if payload is above a defined altitude threshold
bool altThreshCheck( int altThresh ) {
    if (alt > altThresh) {
        Threads::Scope scope(Serial_lock);
        valveAltCnt += UPDATE_INTERVAL_CDU / 1000;
        // if (debug) Serial.print("Valve Alt Count: "); Serial.println(valveAltCnt);
        if (valveAltCnt > VALVE_TIME_THRESH) {
            //increment flags
            altFlag += 1;
            altInterval += 1;
            valveFlag = 1;
            return true;
        }
    } else {
        valveAltCnt = 0;
        return false;
    }
}

// Main valve behavior
void valveSysCheck() {
    while (1) {
        Threads::Scope scope(Serial4_lock);
        if ( altTimer > UPDATE_INTERVAL_CDU ) {
            altTimer = 0;       //reset timer
            CDUcnt += UPDATE_INTERVAL_CDU / 1000;
            //check for LOS
            if (CDUcnt > CDU_LOS) {
                CDUStatus = 0xAF;
            }
        }
    }
    //check for LOS
    if (CDUcnt > CDU_LOS) {
        //CDUStatus = 0xAF;
    }

    // Function to check if payload is floating on ascent
    if ((alt > 5000) && (avgADRate < 0.5) && (numSats > 5) && (altInterval <= NUM_STEPS)) {
        floatAscentCnt += UPDATE_INTERVAL_CDU / 1000;
        // if (debug) Serial.print("float Count: ");
        Serial.println(floatAscentCnt);
        if (floatAscentCnt > FLOAT_TIME) {
            altInterval = 3;
        } else floatAscentCnt = 0;
    }
    // check if payload is floating on descent
    if ((alt < 25000) && (avgADRate > 0.5) && (numSats > 5) && (altInterval > NUM_STEPS)) {
...
floatDescentCnt += UPDATE_INTERVAL_CDU / 1000;
if (debug) Serial.print("float Count: ");
Serial.println(floatDescentCnt);
if (floatDescentCnt > FLOAT_TIME) {
  altInterval = 3;
}
else floatDescentCnt = 0;

altThreshCheck(altThresholds[altInterval]);
Serial.println(altThresholds[altInterval]);
// open valve and start ascent rate checks when threshold is passed
if (altInterval <= NUM_STEPS) {
  if (altFlag > -1) {
    // open valve if ascent rate is above threshold
    if (avgADRate > ascentRates[altFlag] && (valveFlag == 0)) {
      sendPack[1] = OPENcode;
      Serial4.write(sendPack, sizeof(sendPack));
      valveFlag = 1;
    }
    // close valve if ascent rate is below threshold
    if (avgADRate < ascentRates[altFlag]) {
      sendPack[1] = CLOSEcode;
      Serial4.write(sendPack, sizeof(sendPack));
      valveFlag = 0;
    }
  }
  else Serial4.write(sendPack, sizeof(sendPack));
}
else if (altInterval > NUM_STEPS) {
  if (avgADRate > descentRateMIN && (valveFlag == 0)) {
    sendPack[1] = OPENcode;
    Serial4.write(sendPack, sizeof(sendPack));
    valveFlag = 1;
  }
  // close valve if ascent rate is below threshold
  if (avgADRate < descentRateMAX) {
    sendPack[1] = CLOSEcode;
    Serial4.write(sendPack, sizeof(sendPack));
    valveFlag = 0;
  }
}
//      Serial4.write(sendPack, sizeof(sendPack));

void pnpoly(int nvert, float * vertx, float * verty, float testx, float testy)
{
  int i, sign = 0;
  for (i = 0; i < nvert; i++)
  {
    int j = (i + 1) % nvert;
    if (verty[i] <= testy)
      sign = -sign;
    if (vertx[i] > testx)
  

B.1. BUS FLIGHT SOFTWARE

int pnpoly(int nvert, float * vertx, float * verty, float testx, float testy)
int i, j, c = 0;
for (i = 0, j = nvert - 1; i < nvert; j = i++) {
    if (((verty[i] >= testy) != (verty[j] >= testy)) &&
        (testx <= (vertx[j] - vertx[i]) * (testy - verty[i]) / (verty[j] - verty[i]) +
        vertx[i]) )
    c = !c;
    return c;
}

void cutSysCheck() {
    int inside, OOBFlag, boundCnt, inCnt;
    int32_t highAlt = 0;
    static int cutAltCnt;
    // check every second
    if (cutTimer > UPDATE_INTERVAL_CUT) {
        // Reset Timer
        cutTimer = 0;
        if (alt > highAlt) highAlt = alt;   // update highest altitude
        if (GEOFENCE_SYS) {
            if (longitude < LON_LIM) {
                inside = pnpoly(nvert, vertx, verty, longitude, latitude);
                // OOB Counter
                if (inside == 0) {
                    OOBFlag = 1;
                    if (debug) Serial.println("OUT OF BOUNDS");
                    boundCnt += UPDATE_INTERVAL_CUT / 1000;
                    if (debug) Serial.print("OOB Counter: "); Serial.println(boundCnt);
                    if (boundCnt > CUT_TIME) {       // Check if signal is accurate for CUT_TIME_THRESH
                        digitalWrite(CUT_ENABLE, HIGH);
                    } else OOBFlag = 0;
                } else OOBFlag = 0;
            } else OOBFlag = 0;
        }
        // altitude check
        if (((highAlt - alt - 100 > 0) && (alt < CUT_ALT_THRESH)) || cmd == CUTcode) {
            // BT Command
            if (cmd == CUTcode) {
                digitalWrite(CUT_ENABLE, HIGH);
            } else if (inside == 1 && boundCnt != 0) {   // Reset counter if payload goes back in bounds
                inCnt += UPDATE_INTERVAL_CUT / 1000;
                OOBFlag = 0;
                if (inCnt > INSIDE_RESET) {
                    boundCnt = 0;
                    inCnt = 0;
                }
            } else OOBFlag = 0;
        }
        // Below CUT_ALT_THRESH
        if ((highAlt - alt - 100 > 0) && (alt < CUT_ALT_THRESH) && (numSats > 5)) {
            cutAltCnt += UPDATE_INTERVAL_CUT / 1000;
            if (debug) Serial.println(cutAltCnt);
            if (cutAltCnt > CUT_TIME) {
                digitalWrite(CUT_ENABLE, HIGH);
            }
        }
    }
}
void xbeeListen() {
  // get data
  while (Serial4.available() > CDU_RX_SIZE - 1) {
    Serial.println("Theres bytes here");
    byte cutID = Serial4.read();
    cut_id = cutID;
    Serial.println(cut_id, HEX);
    threads.delay(1);
    byte c2 = Serial4.read();
    threads.delay(1);
    // Serial.println(cutID);
    if (cut_id == id_CDU) {
      CDUcnt = 0;
      if ((c2 == CHECKcode) || (c2 == OPENcode) || (c2 == CUTcode) || (c2 == CLOSEcode)) {
        code = c2;
        temp1 = Serial4.read();
        threads.delay(1);
        temp2 = Serial4.read();
        threads.delay(1);
        CDUStatus = Serial4.read();
        threads.delay(1);
        // read gps data from cdu
        for (int i = 0; i < 15; i++) {
          CDUgps[i] = Serial4.read();
          threads.delay(1);
        }
        CDUVolt1 = Serial4.read();
        threads.delay(1);
        CDUVolt2 = Serial4.read();
        // Clear Buffer
        while (Serial4.available()) Serial4.read();
      }
    }
  }
}

void uplinkCheck() {
  while (Serial5.available() > GS_RX_SIZE - 1) {
    byte id = Serial5.read();
    rec_id = id;
    byte cmd1 = Serial5.read();
    if (cmd1 == CUTcode || cmd1 == OPENcode || cmd1 == CLOSEcode || cmd1 == CHECKcode) {
      cmd = cmd1;
      uplinkFlag = 1;
      recieveTime = millis();
    }
  }
  if (uplinkFlag == 1 && (millis() - recieveTime > 6 * UPDATE_INTERVAL_GPS)) {
    uplinkFlag = 0;
  }
}
Camera

Julio Guardado

```c
byte imgData[150000];

uint32_t take_img(int res) {
  File outFile;
  uint8_t tempor = 0, templast = 0;
  uint32_t length = 0;
  int i;
  static int k = 0;
  char str[16];

  // Set camera resolution
  if (res == HR) myCAM.OV2640_set_JPEG_size(OV2640_1600x1200);
  else myCAM.OV2640_set_JPEG_size(OV2640_320x240);

  // take image
  myCAM.flush_fifo();
  myCAM.clear_fifo_flag();
  myCAM.start_capture();
  start_capture = 0;

  // Wait for capture to finish or timeout
  elapsedMillis camCapture;
  bool capDone = false;
  while (camCapture < 500) capDone = myCAM.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK);
  if (!capDone) {
    //    Serial.println("Capture Failed.");
    return 0;
  }

  else {
    threads.delay(50);

    // Get buffer length
    length = myCAM.read_fifo_length();
    myCAM.CS_LOW();
    myCAM.set_fifo_burst();//Set fifo burst mode

    // Request data until entire FIFO buffer is read
    i = 0;
    while (length--) {
      // get data from SPI line
      templast = tempor;
      tempor = SPI.transfer(0x00);

      //Read JPEG data from FIFO
      if ( (tempor == 0xD9) && (templast == 0xFF) ) //If find the end ,break while,
        { 
          //      Serial.println(tempor, HEX);

          //Write the remain bytes in the buffer
```
B.1. BUS FLIGHT SOFTWARE

4/13/22, 6:29 PM Camera

```cpp
55    imgData[i++] = tempor;
56
57    // write img to SD card
58    outFile.write(imgData, i);
59    outFile.close();
60
61    myCAM.CS_HIGH();
62    is_header = false;
63    myCAM.CS_LOW();
64    myCAM.set_fifo_burst();
65    //    imgNum += 1;
66    return i;
67 }
68
69    if (is_header == true) // Read JPEG data once header is found
70    {
71        //Write image data to buffer if not full
72        imgData[i++] = tempor;
73        //        Serial.println(tempor, HEX);
74    }
75
76    else if ((tempor == 0xD8) & (templast == 0xFF)) {   // If JPEG header is identified,
77        start reading data
78        is_header = true;
79        myCAM.CS_HIGH();
80        myCAM.CS_LOW();
81        myCAM.set_fifo_burst();
82        //Create an image file
83        k = k + 1;
84        itoa(k, str, 10);
85        strcat(str, "_.jpg");
86        //Open the new file
87        outFile = SD.open(str, O_WRITE | O_CREAT | O_TRUNC);
88        if (! outFile) {
89            //          Serial.println(F("File open failed"));
90            while (1);
91        }
92
93        // start storing data
94        imgData[i++] = templast;
95        imgData[i++] = tempor;
96    }
97
98    //    noInterrupts();
99    //    WDOG_REFRESH = 0xA602;
100    //    WDOG_REFRESH = 0xB480;
101    //    interrupts();
102 }
103    myCAM.CS_HIGH();
104 }
105 }
106
107 void send_img(int res, uint32_t length) {
108    int i, write_counter = 0;
109    uint16_t packNum = 0;
110    uint16_t counter = 0;
111    // static int imgNum = 0;
112
113    //populate header info
114    radio_packet[86] = imgNum;         // image number
115    radio_packet[87] = imgNum >> 8;
116    radio_packet[88] = res;            //image resolution
117    imgNum += 1;
118```
B.1. BUS FLIGHT SOFTWARE

4/13/22, 6:29 PM  Camera

119  radio_packet[89] = length;         // image length
120  radio_packet[90] = length >> 8;
121  radio_packet[91] = length >> 16;
122  radio_packet[92] = length >> 24;
123
124  // Scan through image and send data in chunks
125  i = IMG_DATA_START_INDEX;
126  while (length--) {
127
128  // stuff outgoing array with data
129  if (i < 254) {
130    radio_packet[i++] = imgData[write_counter++];
131  }
132
133  // send packet 4 times and reset pointers
134  else if (counter < 80) {
135    counter++;
136    radio_packet[93] = packNum;        //packet number for rTx reconstruction
137    radio_packet[94] = packNum >> 8;
138    packNum++;
139
140  // Update science data every 4 packets
141    write_erau_data();
142
143  for (int i = 0; i < DATA_ReTX; i++) {
144    Serial5.write(radio_packet, 256);
145    threads.delay(10);
146  }
147
148  //write to SD card
149  flightData.write(radio_packet, RADIO_SIZE);
150
151  // Reset Pointers
152  i =IMG_DATA_START_INDEX;
153
154  // noInterruptions();
155  // WDOG_REFRESH = 0xA602;
156  // WDOG_REFRESH = 0xB480;
157  // interrupts();
158  }
159  else break;
160  }
161}
Data

void write_erau_data() {
  radio_packet[0] = id_erau[0];
  radio_packet[1] = id_erau[1];
  radio_packet[2] = packet_number;
  radio_packet[3] = packet_number >> 8;
  packet_number += 1;
  /****PACKET TIMESTAMP****/
  time_packet = millis();
  radio_packet[7] = time_packet >> 8;
  radio_packet[8] = time_packet >> 16;
  radio_packet[9] = time_packet >> 24;
  /*******GPS DATA*******/
  radio_packet[10] = latitude;
  radio_packet[12] = latitude >> 16;
  radio_packet[14] = longitude;
  radio_packet[15] = longitude >> 8;
  radio_packet[16] = longitude >> 16;
  radio_packet[17] = longitude >> 24;
  radio_packet[18] = alt;
  radio_packet[19] = alt >> 8;
  radio_packet[20] = alt >> 16;
  radio_packet[21] = alt >> 24;
  radio_packet[22] = stat;
  radio_packet[23] = numSats;
  radio_packet[24] = utcHour;
  radio_packet[25] = utcMin;
  radio_packet[26] = utcSec;
  /*****SCIENCE DATA*****/
  // Analog Sensors
  radio_packet[27] = tempExt_h;
  radio_packet[28] = tempExt_h >> 8;
  radio_packet[29] = tempInt;
  radio_packet[30] = tempInt >> 8;
radio_packet[31] = tempExt_l;
radio_packet[32] = tempExt_l >> 8;
radio_packet[33] = voltage;
radio_packet[34] = voltage >> 8;

if (IMU_SYS) {
    //**** IMU ****/
    // Accel
    radio_packet[35] = accel_x;
    radio_packet[36] = accel_x >> 8;
    radio_packet[37] = accel_y;
    radio_packet[38] = accel_y >> 8;
    radio_packet[39] = accel_z;
    radio_packet[40] = accel_z >> 8;
    // gyro
    radio_packet[41] = gyro_x;
    radio_packet[42] = gyro_x >> 8;
    radio_packet[43] = gyro_y;
    radio_packet[44] = gyro_y >> 8;
    radio_packet[45] = gyro_z;
    radio_packet[46] = gyro_z >> 8;
    // Mag
    radio_packet[47] = mag_x;
    radio_packet[48] = mag_x >> 8;
    radio_packet[49] = mag_y;
    radio_packet[50] = mag_y >> 8;
    radio_packet[51] = mag_z;
    radio_packet[52] = mag_z >> 8;
}

if (BARO_SYS) {
    // Pressure Sensor
    byte* point_temp = (byte*)&mstemp;
    radio_packet[53] = point_temp[0];
    radio_packet[54] = point_temp[1];
    radio_packet[55] = point_temp[2];
    radio_packet[56] = point_temp[3];
    byte* point_pres = (byte*)&pressure;
    radio_packet[57] = point_pres[0];
    radio_packet[58] = point_pres[1];
    radio_packet[59] = point_pres[2];
    radio_packet[60] = point_pres[3];
}

for (int i = 0; i < sizeof(CDUgps); i++) {
    radio_packet[66 + i] = CDUgps[i];
}
B.1. BUS FLIGHT SOFTWARE

```cpp
radio_packet[81] = CDUvolt1;
radio_packet[82] = CDUvolt2;
//Uplink Status
radio_packet[83] = uplinkFlag;
/**********CAMERA Data**********/
// Image Header
radio_packet[84] = id_img[0];
radio_packet[85] = id_img[1];
// Stop Bytes
radio_packet[254] = id_erau[1];
radio_packet[255] = id_erau[0];
```
// Debugger output
void serialOut() { 
  while (1) { 
    if (debugTimer > UPDATE_INTERVAL_DEBUG) { 
      debugTimer = 0; 
      { Threads::Scope scope(Serial_lock); 
        Serial.println("OUtput Packet"); 
        Serial.println("GPS Data"); 
        Serial.println("Barometer Data"); 
        Serial.println("Camera Data"); 
      } 
      Serial.writeln("n"); 
      // Display Output Packet 
      Serial.println("********** Current Packet **********"); 
      for (int k = 0; k < IMG_DATA_START_INDEX; k++) { 
        Serial.print(radio_packet[k], HEX); Serial.print(" "); 
      } 
      Serial.writeln("n"); 
      // Altitude Information 
      Serial.println("********** GPS Data **********"); 
      Serial.println("Current Altitude: "); 
      Serial.println("Current Ascent Rate: "); 
      if ( CDU_SYS ) { 
        Serial.println("Current Altitude Interval: "); 
        if ( valveAltCnt != 0 ) { 
          Serial.println("Valve Counter: "); 
        } 
        if ( floatAscentCnt != 0 ) { 
          Serial.println("Float Ascent Counter: "); 
        } 
        if ( floatDescentCnt != 0 ) { 
          Serial.println("Float Descent Counter: "); 
        } 
      } 
      Serial.writeln("n"); 
      // Barometer 
      if ( BARO_SYS ) { 
        Serial.println("********** Barometer Data **********"); 
        Serial.println("Current Pressure: "); 
        Serial.println("Current Temperature: "); 
      } 
      Serial.writeln("n"); 
      // Camera 
    } 
  } 
}
if ( CAM_SYS ) {
    Serial.println("*************** Camera Data ***************");
    Serial.println("Current Image Number: "); Serial.println(imgNum);
    Serial.println("Image Length: "); Serial.println(im_length);
    Serial.write("\n\n");
}

// CDU DATA
Serial.println("*************** Last Reviewed CDU Data ***************");
Serial.println("Cut ID: "); Serial.println(id_CDU, HEX);
Serial.println("Received Code: "); Serial.println(code, HEX);

    temperature = (int16_t) (temp1 + (temp2 << 8));
    Serial.println("Temperature: "); Serial.println(float(temperature) / 100);
Serial.println("CDU Status: "); Serial.println(CDUStatus);

24));
    Serial.println("Altitude (m): "); Serial.println(CDUalt / 100);
Serial.println("GPS Time: "); Serial.println(CDUgps[12]); Serial.println(":");
Serial.println(CDUgps[13]); Serial.println(":"); Serial.println(CDUgps[14]);

    CDUvoltage = (int16_t) (CDUvolt1 + (CDUvolt2 << 8));
    Serial.println("CDU Battery Monitor: "); Serial.println(CDUvoltage);
Serial.write("\n\n");
}

threads.yield();
B.1. BUS FLIGHT SOFTWARE

Definitions

Julio Guardado

///****************************** Main Variable Definitions ****************************///

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

///////////////////////////////////////////////////////////////////////////////////////////////

///

////////////////////////////////////////////////////////////////////////////

//////////// ******************* Data Definitions ******************** /////////////

#define RADIO_SIZE 256

byte radio_packet[RADIO_SIZE];

byte id_erau[2] = {0xA0, 0xB1}; //Identifier for GPS + ERAU packet.

uint32_t packet_number; //Packet number/counter.

unsigned long time_packet; //Time stamp used to mark ERAU packets - close to UTC time stamp

unsigned char write_counter; //Array indexer for writing data.

unsigned char read_counter; //Array indexer for reading data.

// Timing

#define UPDATE_INTERVAL_DATA 500

elapsedMillis dataUpdate;

uint8_t uplinkFlag;

unsigned long recieveTime;

#define UPDATE_INTERVAL_DEBUG 1000

elapsedMillis debugTimer;

//////////// ******************* SD Card Definitions ******************** /////////////

File flightData;

String ext = ".bin";

String fileName1;

String fileName2;

unsigned long timeSD; // Time stamp - used for determining when to save SD card

int appendPos; // File position marker - used for saving and re-opening SD file

unsigned int fileNum;

unsigned int address;

int sdCount;

const int chipSelect = BUILTIN_SDCARD;

//////////// ******************* Retransmission Definitions *********************** /////////////

#define RETRANSMIT_MIN_ALT_DROP 200*100 // measured in cm

#define RETRANSMIT_HIGH_ALT_LIM 20000*100 // measured in cm

#define RETRANSMIT_TRIGGER_LIMIT 20

#define RETRAN_INTERVAL 50

#define RETRAN_LIM 20

#define UPDATE_INTERVAL_RETRAN 1000

int retranCnt;

byte retransmitTriggerCount = 0;

bool retransmitFlag = false;

int retransPosition;

// Time

#define RETRAN_INTERVAL 50

elapsedMillis retransUpdate;
B.1. BUS FLIGHT SOFTWARE

4/13/22, 6:30 PM Definitions

54 ////////////// *************** GPS Variables ********************** /////////////
55 // GPS Data
56 volatile int32_t latitude = 0;  //Latitude
57 volatile int32_t longitude = 0;  //Longitude
58 volatile int32_t alt = 0;       //Altitude
59 volatile byte stat = 0;         //Status
60 volatile uint8_t numSats = 0;   //Number of Satellites in View
61 volatile uint8_t utcHour = 0;   //UTC Time - Hour
62 volatile uint8_t utcMin = 0;    //UTC Time - Minutes
63 volatile uint8_t utcSec = 0;    //UTC Time - Seconds
64
65 // Avg ascent rate vars
66 //ascent/descent rate variables
67 #define AVERAGE_COUNT 10
68 unsigned long altTime;
69 float ADRateSum;
70 float avgADRate;
71 float prevAlt;
72 unsigned long prevAltTime;
73 int avgCounter;
74 int descentFlag;               //determines whether payload is descending
75 int floatAscentCnt = 0;
76 int floatDescentCnt = 0;
77
78 // Timing
79 #define UPDATE_INTERVAL_GPS 1000
80 elapsedMillis gpsUpdate;
81
82 ////////////// *************** Analog Variables ********************** /////////////
83 #define TEMP_EXT_H A9        //External Temperature - High Range.
84 #define TEMP_EXT_L A8        //External Temperature - Low Range.
85 #define TEMP_INT A7          //Internal Temperature.
86 #define VOLTAGE A6           //Voltage Monitor (VBat).
87
88 volatile int tempExt_l = 0;   //External Temperature - Low Range.
89 volatile int tempExt_h = 0;   //External Temperature - High Range.
90 volatile int tempInt = 0;     //Internal Temperature.
91 volatile int voltage = 0;     //Voltage Monitor (VBat).
92
93 ////////////// *************** Sensor Variables ********************** /////////////
94 #define AG_REFRESH_RATE 9
95 #define MAG_REFRESH_RATE 13
96
97 elapsedMillis AGTimer;       //Timer for updating accelerometer and gyroscope
98 #define MAG_VARIABLES DEFINITION
99 volatile int accel_x = 0;
100 volatile int accel_y = 0;
101 volatile int accel_z = 0;
102
103 volatile int gyro_x = 0;
104 volatile int gyro_y = 0;
105 volatile int gyro_z = 0;
106
107 elapsedMillis magTimer;     //Timer for updating magnetometer
108 volatile int mag_x = 0;
109 volatile int mag_y = 0;
110 volatile int mag_z = 0;
111
112 //MS5611 Setup
113 #define BARO_REFRESH_RATE 5
114 elapsedMillis baroTimer;
115 volatile float pressure;
116 volatile float baroReferencePressure;
117 volatile float mstemp;       //temperature from MS5611
118
119
///*************** CDU Definitions *********************** /////
#define CUT_ENABLE 22

// Packet Identifier
byte id_CDU = 0x27;
byte id_Payload = 0x24;

// Data RX and TX
#define CDU_RX_SIZE 22
#define GS_RX_SIZE 2
byte recPack[CDU_RX_SIZE];
byte sendPack[] = {0x27, 0xEE}; // Default packet to send to CDU
volatile byte rec_id;
volatile byte cmd;

// CDU Variables
volatile float CDUalt;
volatile float CDUlat;
volatile float CDUlon;
volatile int8_t CDUhour;
volatile int8_t CDUmin;
volatile int8_t CDUsec;

// Command Codes
#define OPENcode = 0xBB;
#define CLOSEcode = 0xCC; // Command used to close the valve indefinitely until a new position is commanded
#define CUTcode = 0xDD; // Command used to activate a cutting-thread system to terminate the flight - cuts the balloon attached to the system
#define CHECKcode = 0xEE; // Command used to check the status/internal temperature of the system and that the communications link is still maintained

// Time/Control Variables
#define UPDATE_INTERVAL_CDU 5000
#define UPDATE_INTERVAL_CUT 1000
#define VALVE_TIME_THRESH 20
int valveAltCnt = 0;
#define CDU_LOS 25
int CDUcnt = 0;
int altFlag = -1; // determines if alt is higher than thresh
int valveFlag = 0; // determines state of valve (initial or opened or closed)
uint8_t altInterval = 0;
elapsedMillis altTimer;
elapsedMillis cutTimer;
#define INSIDE_RESET 15 // time where OOB counter resets after coming back into bounds
float CUT_ALT_THRESH = 15000; // Alt where box is cut during descent
volatile byte CDUgps[15]; // array of cdu gps data
byte cut_id;
volatile byte code, temp1, temp2; // command sent from CDU Box A
int16_t temperature; // full temperature reading comprised of temp1 and temp2 from box A
int CDUstatus = 0xAF;
uint16_t CDUvoltage; // full voltage value (adc counts)
byte CDUvolt1, CDUvolt2;

///////*************** Camera Variables *************** /////
byte id_img[2] = {0x27, 0x24}; // Packet header for image packet
const int camCS = 2;
bool is_header = false;
int mode = 0;
uint8_t start_capture = 0;
int imgFlag;
uint16_t imgNum = 0;

// Define camera module

#define UPDATE_INTERVAL_CAM 5000
elapsedMillis camUpdate;
uint8_t imCnt = 0;

volatile char filename[16];

#define HR 0      // High res image
#define LR 1      // Low res image

#define IMG_DATA_START_INDEX 95
#define IMG_REPEAT 10
Flight Plan

Julio Guardado

// Number of repeats for each packet
int DATA_ReTX = 4;

#define IMU_SYS false
#define BARO_SYS false
#define CAM_SYS false
#define CUT_SYS false
#define CDU_SYS true
#define GEOFENCE_SYS false

#define NUM_STEPS 2

#define ALT_LIM_1 2700000
#define ALT_LIM_2 2900000
#define ALT_LIM_3 3200000
#define CUT_ALT_LIM 1500000

int32_t altThresholds[] = {ALT_LIM_1, ALT_LIM_2, ALT_LIM_3};

#define ascentRate_1 3.5
#define ascentRate_2 2.5

float ascentRates[] = {ascentRate_1, ascentRate_2};

#define descentRateMAX -4.0
#define descentRateMIN -4.0

#define VENT_ALT_TIME 10    // Seconds for valid altitude reading before activating vent at
each step
#define FLOAT_TIME 300      // Seconds for valid float condition
#define CUT_TIME 20         // Seconds for valid altitude reading before activating cutting
/* Geofencing Limits */

// Define polygon of acceptable lat,lon values
float LON_LIM = -81.241629; // longitude where geofencing becomes enabled
float vertx[] = {-83.460609,-81.241629,-81.241629,-82.601988}; // longitudes
float verty[] = {30.150493,30.121166,28.914220,28.934370}; // latitudes
int nvert = sizeof(vertx) / 4; // # of vertices in polygon described by above coordinates
// divide by 4 due to sizeof() returning bytes (float = 4 bytes)
GPS

Julio Guardado

```c
void updateGPS() {
    while (1) {
        Threads::Scope scope(Serial3_lock);
        if (gpsUpdate > UPDATE_INTERVAL_GPS) {
            gpsUpdate = 0;
            //get measurements
            alt = (int32_t) (gps.getAlt_meters() * 100);
            latitude = (int32_t) (gps.getLat() * 10000000);
            longitude = (int32_t) (gps.getLon() * 10000000);
            numSats = gps.getSat();
            utcSec = gps.getSecond();
            utcMin = gps.getMinute();
            utcHour = gps.getHour();

            //Calculate Average Ascent/Descent Rate for valve checking
            altTime = millis();
            float timeDiff = (altTime - prevAltTime) / 1000;
            float altDiff = (gps.getAlt_meters() - prevAlt);  
            ADRateSum += altDiff / timeDiff;   //keep running sum of ascent rates
            prevAltTime = altTime;
            prevAlt = gps.getAlt_meters();
            avgCounter += 1;

        }
    }
    threads.yield();
}
```
Retransmission

Julio Guardado

```c
void retransmissionCheck() {
  if (descentFlag == 1 && alt < RETRANSMIT_HIGH_ALT_LIM && retranUpdate > UPDATE_INTERVAL_RETRAN) {
    retransmitTriggerCount += UPDATE_INTERVAL_RETRAN / 1000;
    if (debug) Serial.print("Retransmission Timer: "); Serial.println(retransmitTriggerCount);
    if (retransmitTriggerCount > RETRANSMIT_TRIGGER_LIMIT) {
      retransmitFlag = true;
      flightData.seek(retranPosition);
    }
  }
}

void retransmitProcedure() {
  if (retranUpdate > UPDATE_INTERVAL_RETRAN) {
    retranUpdate = 0;
    if (flightData.available() >= RADIO_SIZE) {
      flightData.read(radio_packet, RADIO_SIZE);
      Serial.println("Retransmitting");
      Serial.print("Bytes Left: "); Serial.println(flightData.available());
      /* Give most up-to-date GPS fix with old data*/
      radio_packet[10] = latitude;
      radio_packet[12] = latitude >> 16;
      radio_packet[14] = longitude;
      radio_packet[15] = longitude >> 8;
      radio_packet[16] = longitude >> 16;
      radio_packet[17] = longitude >> 24;
      radio_packet[18] = alt;
      radio_packet[19] = alt >> 8;
      radio_packet[20] = alt >> 16;
      radio_packet[21] = alt >> 24;
      radio_packet[22] = stat;
      radio_packet[23] = numSats;
      radio_packet[24] = utcHour;
      radio_packet[25] = utcMin;
      radio_packet[26] = utcSec;
      //most recent battery data
      voltage = analogRead(VOLTAGE);
      radio_packet[33] = voltage;
      radio_packet[34] = voltage >> 8;
    }
  }
}```
// Most recent CDU Data
radio_packet[61] = cut_id;
radio_packet[62] = code;
radio_packet[63] = temp1;
radio_packet[64] = temp2;
radio_packet[65] = CDUstatus;
for (int i = 0; i < sizeof(CDUgps); i++) {
  radio_packet[66 + i] = CDUgps[i];
}
radio_packet[81] = CDUvolt1;
radio_packet[82] = CDUvolt2;
Serial5.write(radio_packet, RADIO_SIZE);
// check number of retransmissions for cutting
retranCnt += 1;
if (retranCnt > RETRAN_LIM) {
  cutFlag = 1;
}
else {
  flightData.seek(retranPosition);
  Serial.println("Retransmission Position Reset");
}
}
Sensor Functions

Julio Guardado

```c
void initGPS() {
    unsigned long gpsTimer = millis();
    bool set_airborne = gps.setAirborne();
    delay(50);
    while (!set_airborne && (millis() - gpsTimer < 1 * 60 * 1000 )) {
        set_airborne = gps.setAirborne();
    //    Serial.println("Airborne Mode Not Set!");
    }
    if (set_airborne) Serial.println("GPS OK");
    else Serial.println("GPS Failure");
}

void setupGyro() {
    // [enabled] turns the gyro on or off.
    imu.settings.gyro.enabled = true;  // Enable the gyro
    // [scale] sets the full-scale range of the gyroscope.
    // scale can be set to either 245, 500, or 2000
    imu.settings.gyro.scale = 500; // Set scale to +/-500dps
    // [sampleRate] sets the output data rate (ODR) of the gyro
    // sampleRate can be set between 1-6
    // 1 = 16.9  4 = 238
    // 2 = 59.5  5 = 476
    // 3 = 119   6 = 952
    imu.settings.gyro.sampleRate = 3; // 119Hz ODR
    // [bandwidth] can set the cutoff frequency of the gyro.
    // Allowed values: 0-3. Actual value of cutoff frequency
    // depends on the sample rate. (Datasheet section 7.12)
    imu.settings.gyro.bandwidth = 0;
    // [lowPowerEnable] turns low-power mode on or off.
    imu.settings.gyro.lowPowerEnable = false; // LP mode off
    // [HPFEnable] enables or disables the high-pass filter
    imu.settings.gyro.HPFEnable = true; // HPF disabled
    // [HPFCutoff] sets the HPF cutoff frequency (if enabled)
    // Allowable values are 0-9. Value depends on ODR.
    // (Datasheet section 7.14)
    imu.settings.gyro.HPFCutoff = 1; // HPF cutoff = 4Hz
    // [flipX], [flipY], and [flipZ] are booleans that can
    // automatically switch the positive/negative orientation
```

B.1. BUS FLIGHT SOFTWARE
void setupGyro() {
    // [enabled] turns the gyroscope on or off.
    imu.settings.gyro.enabled = true; // Enable gyroscope
    // [flipX], [flipY], and [flipZ] can turn on or off
    // select axes of the gyroscope.
    imu.settings.gyro.flipX = false; // Don't flip X
    imu.settings.gyro.flipY = false; // Don't flip Y
    imu.settings.gyro.flipZ = false; // Don't flip Z
}

void setupAccel() {
    // [enabled] turns the accelerometer on or off.
    imu.settings.accel.enabled = true; // Enable accelerometer
    // [enableX], [enableY], and [enableZ] can turn on or off
    // select axes of the accelerometer.
    imu.settings.accel.enableX = true; // Enable X
    imu.settings.accel.enableY = true; // Enable Y
    imu.settings.accel.enableZ = true; // Enable Z

    // [scale] sets the full-scale range of the accelerometer.
    // accel scale can be 2, 4, 8, or 16
    imu.settings.accel.scale = 2; // Set accel scale to +/-2g.

    // [sampleRate] sets the output data rate (ODR) of the
    // accelerometer. ONLY APPLICABLE WHEN THE GYROSCOPE IS
    // DISABLED! Otherwise accel sample rate = gyro sample rate.
    // accel sample rate can be 1-6
    // 1 = 10 Hz   4 = 238 Hz
    // 2 = 50 Hz   5 = 476 Hz
    // 3 = 119 Hz  6 = 952 Hz
    imu.settings.accel.sampleRate = 1; // Set accel to 10Hz.

    // [bandwidth] sets the anti-aliasing filter bandwidth.
    // Accel cutoff frequency can be any value between -1 - 3.
    // -1 = bandwidth determined by sample rate
    // 0 = 408 Hz   2 = 105 Hz
    // 1 = 211 Hz   3 = 50 Hz
    imu.settings.accel.bandwidth = 0; // BW = 408Hz

    // [highResEnable] enables or disables high resolution
    // mode for the accelerometer.
    imu.settings.accel.highResEnable = false; // Disable HR

    // [highResBandwidth] sets the LP cutoff frequency of
    // the accelerometer if it's in high-res mode.
    // can be any value between 0-3
    // LP cutoff is set to a factor of sample rate
    // 0 = ODR/50   2 = ODR/9
    // 1 = ODR/100  3 = ODR/400
    imu.settings.accel.highResBandwidth = 0;
}

void setupMag() {
    // [enabled] turns the magnetometer on or off.
    imu.settings.mag.enabled = true; // Enable magnetometer

    // [scale] sets the full-scale range of the magnetometer
    // mag scale can be 4, 8, 12, or 16
    imu.settings.mag.scale = 4; // Set mag scale to +/-4 Gs

    // [sampleRate] sets the output data rate (ODR) of the
    // magnetometer.
    // mag data rate can be 0-7:
    // 0 = 0.625 Hz   4 = 10 Hz
    // 1 = 1.25 Hz   5 = 20 Hz
    // 2 = 2.5 Hz   6 = 40 Hz
    // 3 = 5 Hz   7 = 80 Hz
    imu.settings.mag.sampleRate = 5; // Set OD rate to 20Hz
// [tempCompensationEnable] enables or disables temperature compensation of the magnetometer.
imu.settings.mag.tempCompensationEnable = false;

// [XYPerformance] sets the x and y-axis performance of the magnetometer to either:
// 0 = Low power mode  Z = high performance
// 1 = medium performance 3 = ultra-high performance
imu.settings.mag.XYPerformance = 3; // Ultra-high perform.

// [ZPerformance] does the same thing, but only for the z
imu.settings.mag.ZPerformance = 3; // Ultra-high perform.

// [lowPowerEnable] enables or disables low power mode in the magnetometer.
imu.settings.mag.lowPowerEnable = false;

// [operatingMode] sets the operating mode of the magnetometer. operatingMode can be 0-2:
// 0 = continuous conversion
// 1 = single-conversion
// 2 = power down
imu.settings.mag.operatingMode = 0; // Continuous mode

void setupCam() {
  pinMode(camCS, OUTPUT);
  digitalWrite(camCS, HIGH);
  delay(300);

  myCAM.write_reg(0x07, 0x80);
  delay(100);
  myCAM.write_reg(0x07, 0x00);
  delay(100);

  uint8_t temp;
  unsigned long camTimer = millis();
  while (millis() - camTimer < 1 * 60 * 1000) {
    //Check if the ArduCAM SPI bus is OK
    myCAM.CS_LOW();
    //SPI.beginTransaction(cam);
    myCAM.write_reg(ARDUCHIP_TEST1, 0x55);
    temp = myCAM.read_reg(ARDUCHIP_TEST1);
    myCAM.CS_HIGH();
    if (temp != 0x55) {
      Serial.println(F("ACK CMD SPI interface Error!END"));
      delay(1000); continue;
    } else {
      Serial.println(F("CAMERA OK")); break;
    }
  }

  //Change to JPEG capture mode and initialize the OV5642 module
  myCAM.set_format(JPEG);
  myCAM.InitCAM();
  myCAM.OV2640_set_JPEG_size(OV2640_1600x1200);
  delay(1000);
  myCAM.clear_fifo_flag();
  myCAM.write_reg(ARDUCHIP_FRAMES, 0x00);

  void updateSensors() {
  */
  void updateSensors() {
}
while (1) {
    Threads::Scope scope(Wire_lock);
    /////////////*********** IMU ***********///////////
    if ( IMU_SYS ) {
        if ( AGTimer > AG_REFRESH_RATE ) {
            AGTimer = 0;
            // accelerometer
            imu.readAccel();
            accel_x = imu.ax;
            accel_y = imu.ay;
            accel_z = imu.az;
            // Gyroscope
            imu.readGyro();
            gyro_x = imu.gx;
            gyro_y = imu.gy;
            gyro_z = imu.gz;
        }
        if ( magTimer > MAG_REFRESH_RATE ) {
            magTimer = 0;
            imu.readMag();
            mag_x = imu.mx;
            mag_y = imu.my;
            mag_z = imu.mz;
        }
    }
    /////////////*********** BARO ***********///////////
    if ( BARO_SYS ) {
        if ( baroTimer > BARO_REFRESH_RATE ) {
            baroTimer = 0;
            mstemp = baro.readTemperature();
            pressure = baro.readPressure() * 0.00000986923;
        }
    }
    /////////////*********** Analog ***********///////////
    adc->setConversionSpeed(ADC_CONVERSION_SPEED::LOW_SPEED);   // Or MED_SPEED ?
    tempExt_h = analogRead(TEMP_EXT_H);
    tempInt = analogRead(TEMP_INT);
    tempExt_l = analogRead(TEMP_EXT_L);
    voltage = analogRead(VOLTAGE);
    // Give up time when function is done
    threads.yield();
}
B.2 CDU Flight Software
B.2. CDU FLIGHT SOFTWARE

Main

Julio Guardado

/*****************************************************************************/
/* NAME: HAB_CDU_Seed.ino */
/* PURPOSE: AFOSR-MURI HIGH ALTITUDE BALLOON - Controlled Descent Unit - Valve. */
/* AUTHORS: Julio Cesar Guardado */
/* DEVELOPMENT HISTORY: */
/* Date Author Version Description Of Change */
---
---
---
---
CDU FLIGHT SOFTWARE

CDU code rewritten to work with seeeduino, just a servo and radio connections

******************************************************************************/

/******SERIAL DEFINITIONS******
Startup Monitor Serial
XBee 3 Comms Serial1
/

#include //xbee3 api communication
#include "Definitions.h"

bool debug = true;

void setup() {
  // Begin Serial Communications
  Serial.begin(115200);   //Serial Monitor
  Serial1.begin(38400);   //Serial Monitor

  //initialize servo+++++++++++++++++++++++++++++++++++++++++++++++
  valve.attach(SERVO_PIN);
  valve.write(POS_CLOSED);
  delay(1000);
  valveOPCl(2);

  //adc setup
  analogReadResolution(12);
  for (int i = 0; i < PACKET_SIZE; i++ ) {
    dataPack[i] = 0;
  }

  //flush xbee3 buffer
  while (Serial1.available() > 1) {
    Serial1.read();
  }

  if (debug) Serial.println("Finished init");
}

void loop() {
  /***************XBEE3 Communication**************
  updateCDUStatus();
  xbeelisten();
  if (timer > 1000){
    timer = 0;
    Serial.println("waiting...");
  }
}
Definitions

Julio Guardado

1 /////////////////////////////////////////////////////////////////////////////////////////////
2 //
3 /////////////////////////////////////////////////////////////////////////////////////////////
4 // MAIN PROGRAM DEFINITIONS
5 /////////////////////////////////////////////////////////////////////////////////////////////
6 // Main CDU Status Variable
7 uint8_t CDUstatus;
8 elapsedMillis timer;
9
10 #define SERVO_PIN 1
11 Servo valve;
12
13 #define POS_OPEN 180 // angle where valve is open [deg]
14 #define POS_CLOSED 115 // angle where valve is closed [deg]
15 int valveFlag = 0; // determines state of valve (initial or opened or closed)
16
17 //********************Servo Variables*****************/
18
19
20 //********************Xbee3 Variables******************/
21 #define XBEE3_BAUD 9600
22 #define RX_SIZE 2
23
24 byte id_CDU = 0x27;
25
26 //Commanding Variables
27 byte cmd;
28 byte rec_id;
29 byte recPack[RX_SIZE];
30
31 //Command codes
32 byte OPCIcode = 0xAA;
33 byte OPENCode = 0xBB;
34 byte CLOSECode = 0xCC;
35 byte CUTcode = 0xDD; // command to cut
36 byte CHECkCode = 0xEE; // command to return temperature and status
37
38 #define PACKET_SIZE 22
39 byte stopByte = 0x27;
40 byte dataPack[PACKET_SIZE];
41 unsigned int packNum = 0; // Packet number/counter.
42 int write_counter;
Functions

Julio Guardado

1 /////////////////////////////////////////////////////////////////////////////////////////////////////
2 /////////////////////////////////////////////////////////////////////////////////////////////////////
3 ////////////////////////////////////////////////////////// MAIN PROGRAM FUNCTION DEFINITIONS
4 //////////////////////////////////////////////////////////
5 /////////////////////////////////////////////////////////////////////////////////////////////////////
6 /////////////////////////////////////////////////////////////////////////////////////////////////////
7 /*************** Oscillate Valve ***************
8 void valveOPCL(int num) {
9     for (int i = 0; i < num; i++) {
10         valve.write(POS_OPEN);
11         delay(2000);
12         valve.write(POS_CLOSED);
13     }
14     while (Serial1.available()) {
15         Serial1.read();
16     }
17 }
18
19 /******** Update Status *********
20 void updateCDUStatus() {
21     if (valveFlag){          //valve open
22         CDUstatus = 0;
23     }
24     else CDUstatus = 8;    //valve closed
26 }
27
28 /*************** Check for received commands ***************
29 void xbeeListen() {
30     //check if data is available from radio and reply with dataPack
31     while (Serial1.available() > RX_SIZE - 1) {
32         byte id = Serial1.read();
33         rec_id = id;
34         if (rec_id == id_CDU) {
35             byte cmd1 = Serial1.read();
36             if (debug) Serial.printf("Last recieved Command: "); Serial.println(cmd1, HEX);
37             if (cmd1 == CHECKcode || cmd1 == CUTcode || cmd1 == OPENcode || cmd1 == CLOSEcode || cmd1 == OPCLcode) {
38                 // update packet and send to payload
39                 cmd = cmd1;
40                 dataPack[0] = id_CDU;
41                 dataPack[1] = cmd;
42                 Serial1.write(dataPack, sizeof(dataPack));
43             }
44             // Enact Command
45             if (cmd1 == OPENcode) {
46                 if (valveFlag == 0) {
47                     valveOPCL(1);
48                     valve.write(POS_OPEN);
49                     valveFlag = 1;
50                 }
51             }
52             if (cmd1 == CLOSEcode) {
53                 valve.write(POS_CLOSED);
54             }
55         }
56     }
valveFlag = 0; 
if (debug) {
  (Serial.println("Message Sent.");
  for ( int k = 0; k < sizeof(dataPack); k++) {
    Serial.print(dataPack[k], HEX); Serial.print(" ");
  }
  Serial.write("\n");
} //flush buffer
while (Serial1.available()) Serial1.read();
delay(50);
B.3 Ground Station Software
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grab Image Data</td>
<td>1</td>
</tr>
<tr>
<td>Grab ERAU Packet</td>
<td>15</td>
</tr>
<tr>
<td>Grab SPS30 Packet</td>
<td>35</td>
</tr>
<tr>
<td>Grab CU Instrument Packet</td>
<td>35</td>
</tr>
<tr>
<td>Grab CU Gondola Packet</td>
<td>35</td>
</tr>
<tr>
<td>Parse ERAU data</td>
<td>36</td>
</tr>
<tr>
<td>Parse CU Data</td>
<td>38</td>
</tr>
<tr>
<td>Parse UMN Data</td>
<td>40</td>
</tr>
<tr>
<td>Get RSSI For last packet received</td>
<td>47</td>
</tr>
</tbody>
</table>

classdef HAB_GS_DEV_v2 < matlab.apps.AppBase

% Properties that correspond to app components
properties (Access = public)
    UIFigure                      matlab.ui.Figure
    Location3D                    matlab.ui.control.UIAxes
    Location2D                    matlab.ui.control.UIAxes
    TempInt                       matlab.ui.control.UIAxes
    ConfigureMenu                 matlab.ui.container.Menu
    RefreshCOMPortsMenu           matlab.ui.container.Menu
    LoadMapsMenu                  matlab.ui.container.Menu
    LoadPredictionFileMenu        matlab.ui.container.Menu
    BeginTrackingButton           matlab.ui.control.Button
    ElevationGauge
    matlab.ui.control.SemicircularGauge
    AzGauge                       matlab.ui.control.LinearGauge
    ConnectGSButton               matlab.ui.control.Button
    DisconnectGSButton            matlab.ui.control.Button
    GSCOMPortDropDownLabel        matlab.ui.control.Label
    GSCOMPortDropDown             matlab.ui.control.DropDown
    RxCOMPortDropDownLabel        matlab.ui.control.Label
    RxCOMPortDropDown             matlab.ui.control.DropDown
    gpsLAT                        matlab.ui.control.Label
    gpsLONG                       matlab.ui.control.Label
    gpsALT                        matlab.ui.control.Label
    gpsFIX                        matlab.ui.control.Label
    LatitudeLabel                 matlab.ui.control.Label
    LongitudeLabel                matlab.ui.control.Label
    AltitudeLabel                 matlab.ui.control.Label
    FixQLabel                     matlab.ui.control.Label
    gpsSATs                       matlab.ui.control.Label
    gpsHOUR                       matlab.ui.control.Label
    gpsMIN                        matlab.ui.control.Label
    gpsSEC                        matlab.ui.control.Label
    SatsLabel                     matlab.ui.control.Label
    GPSUTCTimeLabel               matlab.ui.control.Label
    AscentRate                    matlab.ui.control.Label
### B.3. GROUND STATION SOFTWARE

<table>
<thead>
<tr>
<th>Label</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADRateLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>GroundStationTestButton</td>
<td>matlab.ui.control.Button</td>
</tr>
<tr>
<td>gsPortStatus</td>
<td>matlab.ui.control.Lamp</td>
</tr>
<tr>
<td>TabGroup</td>
<td>matlab.ui.container.TabGroup</td>
</tr>
<tr>
<td>PositionTab</td>
<td>matlab.ui.container.Tab</td>
</tr>
<tr>
<td>AltitudeEditFieldLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>GS_Altitude</td>
<td>matlab.ui.control.EditField</td>
</tr>
<tr>
<td>LongitudeEditFieldLabel</td>
<td>matlab.ui.control.EditField</td>
</tr>
<tr>
<td>GS_Longitude</td>
<td>matlab.ui.control.EditField</td>
</tr>
<tr>
<td>LatitudeEditFieldLabel</td>
<td>matlab.ui.control.EditField</td>
</tr>
<tr>
<td>GS_Latitude</td>
<td>matlab.ui.control.EditField</td>
</tr>
<tr>
<td>DeclinationEditFieldLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>DeclinationEditField</td>
<td>matlab.ui.control.EditField</td>
</tr>
<tr>
<td>ElBiasEditFieldLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>ElBiasEditField</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>LoadPositionButton</td>
<td>matlab.ui.control.Button</td>
</tr>
<tr>
<td>LoadPredictionfileButton</td>
<td>matlab.ui.control.Button</td>
</tr>
<tr>
<td>ReproduceFlightButton</td>
<td>matlab.ui.control.Button</td>
</tr>
<tr>
<td>TrackByTab</td>
<td>matlab.ui.container.Tab</td>
</tr>
<tr>
<td>ButtonGroup</td>
<td>matlab.ui.container.ButtonGroup</td>
</tr>
<tr>
<td>GPSButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>PredictionFileButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>PredictionFileButtonGroup</td>
<td>matlab.ui.container.ButtonGroup</td>
</tr>
<tr>
<td>AscentButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>DescentButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>PredictionMode</td>
<td>matlab.ui.container.ButtonGroup</td>
</tr>
<tr>
<td>ManualButton2</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>AutomaticButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>AltmSpinnerLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>AltmSpinner</td>
<td>matlab.ui.control.Spinner</td>
</tr>
<tr>
<td>OptionsTab</td>
<td>matlab.ui.container.Tab</td>
</tr>
<tr>
<td>AntenaTrakingModeButtonGroup</td>
<td>matlab.ui.container.ButtonGroup</td>
</tr>
<tr>
<td>AzElButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>AzOnlyButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>ElOnlyButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>ManualButton</td>
<td>matlab.ui.control.RadioButton</td>
</tr>
<tr>
<td>EndTrackingButton</td>
<td>matlab.ui.control.Button</td>
</tr>
<tr>
<td>trackingStat</td>
<td>matlab.ui.control.Lamp</td>
</tr>
<tr>
<td>ControlledDescentSystemPanel</td>
<td>matlab.ui.container.Panel</td>
</tr>
<tr>
<td>TemperatureLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>LatitudeLabel_2</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>LongitudeLabel_2</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>AltitudeLabel_2</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>GPSTimeLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>BatteryVoltageLabel_2</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>LastCodeLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>TabGroup2</td>
<td>matlab.ui.container.TabGroup</td>
</tr>
<tr>
<td>ValveStatusTab</td>
<td>matlab.ui.container.Tab</td>
</tr>
<tr>
<td>VentclosedLampLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>VentclosedLamp</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>VentopenLampLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>VentopenLamp</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>LineCutLampLabel</td>
<td>matlab.ui.control.Label</td>
</tr>
<tr>
<td>LineCutLamp</td>
<td>matlab.ui.control.Label</td>
</tr>
</tbody>
</table>
B.3. GROUND STATION SOFTWARE

```matlab
properties (Access = private)
    xdata % For plotting x data
    ydata % For plotting y data
    zdata % For plotting z data
    wdata % For plotting w data
    flag = 0;
    flag_GS = 0;
    hC = 0;
    maxC = 0;
end
```
myGSCoord;  
WYPredicted;  
spheroid = referenceEllipsoid('WGS 84');  
CUSFPredicted;  

startTime = 0;  
newMapFlag = 0;  
predFileFlag = 0;  
predFileCol = [ 'k.' , 'r' , 'b' , 'g' , 'c' ];  

loopBreak = 0;  % Description  
cutCnt = 0;  % Description  

properties  (Access = public)  
dataFile;  
gpsData;  
scientificData;  
serial_GS;  
A;  B;  
s;  
end  

methods  (Access = private)  

function  [latlim, lonlim] = getMapLimits(app,lat0,lon0,h0)  
    if  lat0 <= 90 && lat0 >= -90 && lon0 <= 180 && lon0 >= -180 && isnumeric(h0) && isreal(h0)  
        az = [0 90 180 270];  
        slantRange = 200*1000;  
        elev = 0;  
        lat = [0 0 0 0];  
        lon = lat;  
        h=lat;  
        for  f = 1:4  
            [lat(f),lon(f),h(f)] = aer2geodetic(az(f),elev,slantRange,lat0,lon0,h0,app.spheroid);  
        end  
        latlim = [lat(3) lat(1)];  
        lonlim = [lon(4) lon(2)];  
    else  
        errordlg('Check your position and try again');  
    end  
end  

function  ZA = loadMaps(app,latlim,lonlim)  
ZA=[];  
prompt = {'Would you like to download new Map data?' , '(Requires Internet Connection)'};  
title = 'WMS Map Update';  
answ = questdlg(prompt,title,'New Map','Load Map','Cancel','Cancel');  
switch answ  

4
case 'New Map'
    numberOfAttempts = 5;
    attempt = 0;
    info = [];
    mundalisServer = 'http://ows.mundialis.de/services/service?';
    OSM_WMS_Uni_Heidelberg = 'http://129.206.228.72/cached/osm?';
    serv2 = 0;
    while (isempty(info))
        try
            if serv2 == 0
                info = wmsinfo(mundalisServer);
                orthoLayer = info.Layer(2);
            elseif serv2 == 1
                info = wmsinfo(OSM_WMS_Uni_Heidelberg);
                orthoLayer = info.Layer(2);
            end
            catch
                attempt = attempt + 1;
                if attempt > numberOfAttempts && serv2 == 0
                    warning('Server 1 is not available.
Trying Server 2');
                    serv2 = 1;
                    attempt = 0;
                end
            end
        end
        if serv2 == 1 && attempt > numberOfAttempts
            warndlg ({'WMS servers are not
available.'; 'Please load an existing Map'});
            return
        end
    end
    [ZA, ~] = wmsread(orthoLayer, 'Latlim', latlim, 'Lonlim', lonlim, ...
                      'ImageFormat', 'image/png');
    case 'Load Map'
        [newfile,path] = uigetfile('*.map', 'Load Map File', 'map1.map');
        figure(app.UIFigure);
        if newfile == 0
            return;
        end
        filename = fullfile(path,newfile);
        load(filename,'ZA','-mat');
        app.newMapFlag = 0;
    case 'Cancel'
        return
function results = DrawMaps(app,ZA,latlim,lonlim)
    results = 0;
    imagesc(app.Location2D,lonlim,latlim,flipud(ZA));
    imagesc(app.Location3D,lonlim,latlim,flipud(ZA));
    %                   demcmap(double(ZA))
    %
    lat=linspace(latlim(2),latlim(1),size(ZA,1));
    %
    lon=linspace(lonlim(1),lonlim(2),size(ZA,2));
    %
    pcolor(app.Location3D,lon,lat,ZA);
    %
    pcolor(app.Location2D,lon,lat,ZA);
    %
    app.Location2D.DataAspectRatio=[abs(diff(lonlim)),abs(diff(latlim)),1];
    %                         [cmap,~] = demcmap(ZA);
    %                         colormap(app.Location3D,cmap);
    %                         shading(app.Location3D,
    %                   'interp');
    %                         colormap(app.Location2D,cmap);
    %                         shading(app.Location2D,
    %                   'interp');
    xlim(app.Location2D,lonlim)
    ylim(app.Location2D,latlim)
    view(app.Location3D,15,15)
    zlim(app.Location3D,[-.001,40])
    if  app.newMapFlag == 1
        q2 = questdlg( 'Would you like to save the map data?', 'Save?', 'Yes', 'No', 'Yes' );
        switch q2
            case 'Yes'
                [newfile,path] = uiputfile( '*.map', 'Create Data File', 'map1.map' );
                if  newfile == 0
                    return;
                end
                filename=fullfile(path,newfile);
                save( filename, 'ZA' );
            case 'No'
        end
    end
    results = 1;
end

B.3. GROUND STATION SOFTWARE

165
function results = updateCDUStatus(app,valveStatusA)
    switch valveStatusA
    case 0
        app.VentclosedLamp.Color = 'white';
        app.VentopenLamp.Color = 'green';
        app.LineCutLamp.Color = 'white';
        app.HPOffLamp.Color = 'white';
        app.HPOnLamp.Color = 'green';
        app.InBoundsLamp.Color = 'green';
        app.NoSignalLamp.Color = 'white';
    case 1
        app.VentclosedLamp.Color = 'white';
        app.VentopenLamp.Color = 'green';
        app.LineCutLamp.Color = 'green';
        app.HPOffLamp.Color = 'white';
        app.HPOnLamp.Color = 'green';
        app.InBoundsLamp.Color = 'green';
        app.NoSignalLamp.Color = 'white';
    case 2
        app.VentclosedLamp.Color = 'white';
        app.VentopenLamp.Color = 'green';
        app.LineCutLamp.Color = 'white';
        app.HPOffLamp.Color = 'green';
        app.HPOnLamp.Color = 'white';
        app.InBoundsLamp.Color = 'green';
        app.NoSignalLamp.Color = 'white';
    case 3
        app.VentclosedLamp.Color = 'white';
        app.VentopenLamp.Color = 'green';
        app.LineCutLamp.Color = 'green';
        app.HPOffLamp.Color = 'white';
        app.HPOnLamp.Color = 'green';
        app.InBoundsLamp.Color = 'green';
        app.NoSignalLamp.Color = 'white';
    case 4
        app.VentclosedLamp.Color = 'white';
        app.VentopenLamp.Color = 'green';
        app.LineCutLamp.Color = 'white';
        app.HPOffLamp.Color = 'white';
        app.HPOnLamp.Color = 'green';
        app.InBoundsLamp.Color = 'red';
        app.NoSignalLamp.Color = 'white';
    case 5
        app.VentclosedLamp.Color = 'white';
        app.VentopenLamp.Color = 'green';
        app.LineCutLamp.Color = 'green';
        app.HPOffLamp.Color = 'white';
        app.HPOnLamp.Color = 'green';
        app.InBoundsLamp.Color = 'red';
        app.NoSignalLamp.Color = 'white';
    case 6
        app.VentclosedLamp.Color = 'white';

app.VentopenLamp.Color = 'green';
app.LineCutLamp.Color = 'white';
app.HPOffLamp.Color = 'white';
app.HPOnLamp.Color = 'green';
app.InBoundsLamp.Color = 'red';
app.NoSignalLamp.Color = 'white';

**case** 7
app.VentclosedLamp.Color = 'white';
app.VentopenLamp.Color = 'green';
app.LineCutLamp.Color = 'green';
app.HPOffLamp.Color = 'green';
app.HPOnLamp.Color = 'white';
app.InBoundsLamp.Color = 'red';
app.NoSignalLamp.Color = 'white';

**case** 8
app.VentclosedLamp.Color = 'green';
app.VentopenLamp.Color = 'white';
app.LineCutLamp.Color = 'white';
app.HPOffLamp.Color = 'white';
app.HPOnLamp.Color = 'green';
app.InBoundsLamp.Color = 'green';
app.NoSignalLamp.Color = 'white';

**case** 9
app.VentclosedLamp.Color = 'green';
app.VentopenLamp.Color = 'white';
app.LineCutLamp.Color = 'green';
app.HPOffLamp.Color = 'white';
app.HPOnLamp.Color = 'green';
app.InBoundsLamp.Color = 'green';
app.NoSignalLamp.Color = 'white';

**case** 10
app.VentclosedLamp.Color = 'green';
app.VentopenLamp.Color = 'white';
app.LineCutLamp.Color = 'white';
app.HPOffLamp.Color = 'white';
app.HPOnLamp.Color = 'green';
app.InBoundsLamp.Color = 'green';
app.NoSignalLamp.Color = 'white';

**case** 11
app.VentclosedLamp.Color = 'green';
app.VentopenLamp.Color = 'white';
app.LineCutLamp.Color = 'green';
app.HPOffLamp.Color = 'white';
app.HPOnLamp.Color = 'white';
app.InBoundsLamp.Color = 'green';
app.NoSignalLamp.Color = 'white';

**case** 12
app.VentclosedLamp.Color = 'green';
app.VentopenLamp.Color = 'white';
app.LineCutLamp.Color = 'white';
app.HPOffLamp.Color = 'green';
app.HPOnLamp.Color = 'white';
app.InBoundsLamp.Color = 'red';
app.NoSignalLamp.Color = 'white';
case 13
    app.VentclosedLamp.Color = 'green';
    app.VentopenLamp.Color = 'white';
    app.LineCutLamp.Color = 'green';
    app.HPOffLamp.Color = 'white';
    app.HPOnLamp.Color = 'green';
    app.InBoundsLamp.Color = 'red';
    app.NoSignalLamp.Color = 'white';

case 14
    app.VentclosedLamp.Color = 'green';
    app.VentopenLamp.Color = 'white';
    app.LineCutLamp.Color = 'white';
    app.HPOffLamp.Color = 'green';
    app.HPOnLamp.Color = 'white';
    app.InBoundsLamp.Color = 'red';
    app.NoSignalLamp.Color = 'white';

case 15
    app.VentclosedLamp.Color = 'green';
    app.VentopenLamp.Color = 'white';
    app.LineCutLamp.Color = 'green';
    app.HPOffLamp.Color = 'green';
    app.HPOnLamp.Color = 'white';
    app.InBoundsLamp.Color = 'red';
    app.NoSignalLamp.Color = 'white';

case 175
    app.NoSignalLamp.Color = 'white';
    app.VentclosedLamp.Color = 'white';
    app.VentopenLamp.Color = 'white';
    app.LineCutLamp.Color = 'white';
    app.HPOffLamp.Color = 'white';
    app.HPOnLamp.Color = 'white';
    app.InBoundsLamp.Color = 'white';
    app.NoSignalLamp.Color = 'red';
end

end
end

% Callbacks that handle component events
methods (Access = private)

% Code that executes after component creation
function startupFcn(app)
    clc;
    app.RxCOMPortDropDown.Items = cellstr(seriallist);
    app.GSCOMPortDropDown.Items = app.RxCOMPortDropDown.Items;
    %hold(app.Voltage, 'on');
    hold(app.TempInt, 'on');
    %datetick(app.Voltage,'x', 'HH:MM:SS');
    hold(app.Location3D,'on');
    hold(app.Location2D,'on');
    app.myGSCoord = [str2double(app.GS_Latitude.Value), ...
                     str2double(app.GS_Longitude.Value), str2double(app.GS_Altitude.Value)];

end
B.3. GROUND STATION SOFTWARE

app.UIFigure.Position = [0 0 1280 700];
%             app.UIFigure.WindowState = 'maximized';
%             instrreset;

% Load 2D Map
Lat = 29.187904;
Lon = -81.048094;
Z = 0;

[latlim, lonlim] = getMapLimits(app,Lat,Lon,Z);
load('200km.map', 'ZA', '-mat');
imagesc(app.Location2D,lonlim,latlim,flipud(ZA));
imagesc(app.Location3D,lonlim,latlim,flipud(ZA));
xlim(app.Location2D,lonlim)
ylim(app.Location2D,latlim)
xlim(app.Location3D,lonlim)
ylim(app.Location3D,latlim)
view(app.Location3D,15,15)
zlim(app.Location3D,[-.001,40])

end

% Callback function: LoadMapsMenu, LoadPositionButton
function  LoadPositionButtonPushed(app, event)
q1 = questdlg( 'Do you want to get the GS coordinates from the GPS [GS Connection Required]' , 'GS GPS Coordinates' , 'Yes' , 'No' , 'Yes' );
switch  q1
    case  'Yes'
        fprintf(app.serial_GS, '%s
' , 'getLoc' );
        while  (app.serial_GS.BytesAvailable == 0)
            end
        [coords] = fscanf(app.serial_GS, '%d,%d,%d
' );
        app.GS_Latitude.Value = num2str(coords(1)/10000000);
        app.GS_Longitude.Value = num2str(coords(2)/10000000);
        app.GS_Altitude.Value = num2str(coords(3)/100);
    end
    end

prompt = { 'Enter the decimal latitude of the center' , 'Enter the decimal Longitude of the center' , ...
            'Enter the Altitude in meters' , 'Enter the desired Map Radius in km' };
title = 'Map Configuration';
dims = [1,35];
default = {app.GS_Latitude.Value, app.GS_Longitude.Value, app.GS_Altitude.Value, '200'};
answer = inputdlg(prompt,title,dims,default);
if  ~isempty(answer) && isreal(str2double(answer))
mapZ = str2double(answer{3});
mapLat = str2double(answer{1});
mapLon = str2double(answer{2});
%     app.radius.Text = answer{4};
else
    errordlg('Check the center position and try again');
    return
end

q2 = questdlg('Is the antenna position the same as the map center?','Antenna Position','Yes','No','Yes');
switch q2
    case 'Yes'
        h0 = mapZ;
        lat0 = mapLat;
        lon0 = mapLon;
        app.GS_Latitude.Value = num2str(lat0);
        app.GS_Longitude.Value = num2str(lon0);
        app.GS_Altitude.Value = num2str(h0);
    case 'No'
        prompt = {'Enter the decimal latitude', 'Enter the decimal Longitude', ...
            'Enter the Altitude in meters'};
        title = 'Antenna Position';
        dims = [1,35];
        default = {app.GS_Latitude.Value,
            app.GS_Longitude.Value, app.GS_Altitude.Value, app.radius.Text};
        answer = inputdlg(prompt,title,dims,default);
        if ~isempty(answer) && isreal(str2double(answer))
            h0 = str2double(answer{3});
            lat0 = str2double(answer{1});
            lon0 = str2double(answer{2});
            app.GS_Latitude.Value = answer{1};
            app.GS_Longitude.Value = answer{2};
            app.GS_Altitude.Value = answer{3};
            app.radius.Text = answer{4};
        else
            errordlg('Check your position and try again');
            return
        end
end

[latlim, lonlim] = getMapLimits(app,mapLat,mapLon,mapZ);
ZA = loadMaps(app,latlim,lonlim);
success =0;
if ~isempty(ZA)
    success = DrawMaps(app,ZA,latlim,lonlim);
end
if success == 1
    plot3(app.Location3D, lon0, lat0,
        h0/1000, 'r*', 'LineWidth',2) 
    plot(app.Location2D, lon0, lat0, 'r*') 
    app.LoadPredictionfileButton.Enable = 'on';
```matlab
app.LoadPredictionFileMenu.Enable = 'on';
app.ReproduceFlightButton.Enable='on';
app.LaunchTimeButton.Enable='on';
app.BeginTrackingButton.Enable= 'on' ;
app.GroundStationTestButton.Enable = 'on' ;
end
figure(app.UIFigure);
end

function BeginTrackingButtonPushed(app, event)

[newfile,path] = uiputfile( '*.bin' , 'Create Data
File',strcat(datestr(datetime, 'yyyy_mm_dd' ),'__',datestr(datetime, 'HH_MM_SS' ),'_d
figure(app.UIFigure);
if  newfile == 0
    return;
end
filename=fullfile(path,newfile);
app.dataFile=fopen(filename, 'w+' );
[newfile,path] = uiputfile( '*.bin' , 'Create Data
File',strcat(datestr(datetime, 'yyyy_mm_dd' ),'__',datestr(datetime, 'HH_MM_SS' ),'_R
rssiFile = fullfile(path,newfile);
RSSIFile = fopen(rssiFile, 'w+' );

lat0=app.myGSCoord(1);
lon0=app.myGSCoord(2);
h0=app.myGSCoord(3);

%Serial for the radio communication or file
app.s = serial(app.RxCOMPortDropDown.Value);

%Set serial parameters
app.s.InputBufferSize = 15000000;
set(app.s, 'DataBits', 8);
set(app.s, 'StopBits', 1);
set(app.s, 'BaudRate', 230400);
set(app.s, 'Parity', 'none');

%Open the serial port
try
    fopen(app.s);
catch err
    fclose(app.s);
    warndlg('Make sure you select the correct Radio COM
Port.');
end
```

B.3. GROUND STATION SOFTWARE
% Current time
CurTime = datetime;

% packet headers
id_erau = [160, 177];
id_umn = [66, 1];
id_cu_instr = [193, 9];
id_cu_gond = [210, 168];

binary_file = app.dataFile;

rcvd_packets = 0;
lost_packets = 0;
expected = 0;
prev_pack_num = 0;
lost_total = 0;
packet_num = 0;
new_packet_number = 0;
range = 0;
timer_1 = tic;
timer_3 = tic;
timer_5 = tic;
firstFlag = 1;

% External High Thermistor Coefficients:
p1_ex = 0.4975;
p2_ex = 2.342;
p3_ex = 1.396;
p4_ex = 16.54;
p5_ex = -7.183;
mean_ex = 547.6;
std_ex = 236.2;

% Extra External Low Thermistor Coefficients:
p1_ex2 = -1.129;
p2_ex2 = 4.101;
p3_ex2 = -1.778;
p4_ex2 = 15.41;
p5_ex2 = -33.61;
mean_ex2 = 386.2;
std_ex2 = 255.8;

% Internal Thermistor Coefficients:
p1_in = -0.344;
p2_in = -1.26;
p3_in = -2.305;
p4_in = -13.18;
p5_in = 7.039;

mean_in = 844.7;
std_in = 97.26;
%Voltage ADC Calibration
p1_v = -0.003335;
p2_v = 1.082;
p3_v = 1.653;

mean_v = 510;
std_v = 334.6;

%Initial Position
lat = lat0;
lon = lon0;
h = h0;

%Initial time and threshold of the timer (time between GS checks)
timerIni = tic;
timeThreshold = 5;

%Ascent Rate Monitor Variables
prevAlt = 35;
prevTime = 0;
ascentRate = 0;

% Expected packet lengths
IMAGE_HR = 100000;
IMAGE_LR = 10000;

TOTAL_PACKET_LENGTH = 256;
messages = zeros(TOTAL_PACKET_LENGTH,1,'double');
DATA_ERAU = messages;
app.trackingStat.Color = 'green';

%Read and Process Data
loop_time = toc;
activecmd_time = toc;
ctr = 0;
im_flag = 0;

mkdir images

while (~app.loopBreak)
    if(app.s.BytesAvailable>TOTAL_PACKET_LENGTH*(11*4))
        % Display bytes available
        pause(3.5)
        test = tic;

        % Save data to binary file
read_packet = fread(app.s,app.s.BytesAvailable)’;
fwrite(binary_file, read_packet);

% Parse read packets to get packet numbers
idx = strfind(read_packet(1:end-TOTAL_PACKET_LENGTH),[160 177]);
% disp(length(idx))
% rcvd_packets = rcvd_packets + length(idx)+1;

imgData = [];
for i = 1:length(idx)
    if (read_packet(idx(i)+254:idx(i)+255) == [hex2dec('B1'),hex2dec('A0')])
        messages = read_packet(idx(i):idx(i)+TOTAL_PACKET_LENGTH-1);
        rcvd_packets = rcvd_packets + 1;
        packet_num = messages(94) + bitshift(messages(95),8);

        % check for lost packets
        if firstFlag
            prev_pack_num = messages(94) + bitshift(messages(95),8);
            first_rcvd = (double(typecast(uint8(messages(3:6)),'uint32')))*4;
            firstFlag = 0;
        end

        % skipped packets
        packDiff = packet_num - prev_pack_num;
        prev_pack_num = packet_num;

        % Grab Image Data
        if packDiff ~= 0
            if (messages(85:86) == [39,36])
                imgData = [imgData messages(96:end-2)];
            end
        end
    end
end
rcvd_packets = rcvd_packets + 1;

% Save and display image
% expected = expected + (packet_num)*4-first_rcvd;
imgNum = DATA_ERAU(87) + bitshift(DATA_ERAU(88),8);
app.ImageNumberLabel.Text = ['Image Number: ', num2str(imgNum)];
B.3. GROUND STATION SOFTWARE

fname = ['images/',
strcat(datestr(datetime, 'yyyy_mm_dd'), '__', datestr(datetime, 'HH_MM_SS'), '__')
num2str(imgNum), '.jpg'];
file = fopen(fname, 'w+');
fwrite(file, imgData);
fclose(file);

app.Image.ImageSource = fname;

if ~isempty(messages)
    % Parse science data
    DATA_ERAU = messages;
    packet_num = double(typecast(uint8(DATA_ERAU(3:6)), 'uint32'));
    expected = (packet_num)*4-first_rcvd+4;
    packet_time = typecast(uint8(DATA_ERAU(7:10)), 'uint32');
    packet_time = double((packet_time)/1000);
    app.PacketTimeLabel.Text = ['Packet Time:', num2str(packet_time/60)];
    app.curpacknum.Text = ['Current Packet Number:', num2str(packet_num)];
    app.ReceivedPackets.Text = ['Received Packets:', num2str(rcvd_packets)];
    app.ExpectedPackets.Text = ['Expected Packets:', num2str(expected)];
    received_percent = 100 - 100*(rcvd_packets/(expected));
    app.TotalLossLabel.Text = ['% Total Loss:', num2str(received_percent)];

    lat = double(typecast(uint8(DATA_ERAU(11:14)), 'int32'))/10000000;
    lon = double(typecast(uint8(DATA_ERAU(15:18)), 'int32'))/10000000;
    h = double(typecast(uint8(DATA_ERAU(19:22)), 'int32'))/100;
    stat = DATA_ERAU(23);
    numSats = DATA_ERAU(24);
    utcHour = DATA_ERAU(25);
    utcMin = DATA_ERAU(26);
    utcSec = DATA_ERAU(27);

    % Print Current GPS Data
    app.gpsLAT.Text = num2str(lat);
    app.gpsLONG.Text = num2str(lon);
    app.gpsHOUR.Text = num2str(utcHour);
    app.gpsMIN.Text = num2str(utcMin);
    app.gpsSEC.Text = num2str(utcSec);
B.3. GROUND STATION SOFTWARE

% Plot Ublox GPS Data
plot3(app.Location3D, lon, lat, h/1000, 'b.', 'LineWidth',1)
plot(app.Location2D, lon, lat, 'b. ')

% Calculate Ascent Rate
newAlt = h;
newTime = utcHour*3600+utcMin*60+utcSec;
if (newTime < 2*60*60)
    newTime = newTime + 24*60*60;
end

ascentRate = double((newAlt - prevAlt)/(newTime - prevTime));
prevAlt = newAlt;
prevTime = newTime;
app.AscentRate.Text = num2str(ascentRate);

% Science data
% External High Temperature Conversion
temp = typecast(uint8(DATA_ERAU(28:29)),'uint16');
temp = double(temp);
temp = (temp-mean_ex)/std_ex;
temp_ex = p1_ex*temp^4 + p2_ex*temp^3 + p3_ex*temp^2 + p4_ex*temp + p5_ex;

% Internal Temperature Conversion
temp = typecast(uint8(DATA_ERAU(30:31)),'uint16');
temp = double(temp);
temp = (temp-mean_in)/std_in;
temp_in = p1_in*temp^4 + p2_in*temp^3 + p3_in*temp^2 + p4_in*temp + p5_in;

% External Low Temperature Conversion
temp = typecast(uint8(DATA_ERAU(32:33)),'uint16');
temp = double(temp);
temp = (temp-mean_ex2)/std_ex2;
temp_ex2 = p1_ex2*temp^4 + p2_ex2*temp^3 + p3_ex2*temp^2 + p4_ex2*temp + p5_ex2;

% Accelerometer Data
accel_z = double(typecast((uint8(DATA_ERAU(46:47))),'int16'))*6.1e-5*9.8;
gyro_z = double(typecast((uint8(DATA_ERAU(52:53))),'int16'))*8.75e-3;
% disp(accel_z)
app.AccelerationZLabel.Text = ['Acceleration (Z): ',num2str(accel_z)];
% Voltage Monitor
voltage = typecast(uint8(DATA_ERAU(34:35)),'uint16');
voltage = double(voltage);
voltage = (voltage-mean_v)/std_v;
volt_supply = 3*(p1_v*voltage^2 + p2_v*voltage + p3_v);
app.BatteryVoltageLabel.Text = ['Battery Voltage: ' , num2str(volt_supply) ];

% Barometer Data
baro_pressure = typecast(uint8(DATA_ERAU(58:61)),'single');
baro_temp = typecast(uint8(DATA_ERAU(54:57)),'single');

% update text fields
app.PressureLabel.Text = ['Pressure: ' , num2str(baro_pressure), ' atm' ];
app.TemperatureLabel_2.Text = ['Temperature: ' , num2str(baro_temp), ' C' ];

% CDU Data
tempCDUA = double(typecast(uint8(DATA_ERAU(64:65)),'int16'))/100;
commandCodeA = DATA_ERAU(63);
valveStatusA = DATA_ERAU(66);
altCDU = double(typecast(uint8(DATA_ERAU(67:70)),'int32'))/100;
latCDU = double(typecast(uint8(DATA_ERAU(71:74)),'int32'))/10000000;
lonCDU = double(typecast(uint8(DATA_ERAU(75:78)),'int32'))/10000000;
hourCDU = DATA_ERAU(79);
minCDU = DATA_ERAU(80);
secCDU = DATA_ERAU(81);
voltageCDU = DATA_ERAU(82) + bitshift(DATA_ERAU(83),8);
voltageCDU = double(voltageCDU);
voltageCDU = (voltageCDU-mean_v)/std_v;
% volt_supplyCDU = 3*5/3.3*(p1_v*voltageCDU^2 + p2_v*voltageCDU + p3_v);
volt_supplyCDU = 3*(p1_v*voltageCDU^2 + p2_v*voltageCDU + p3_v);
app.BatteryVoltageLabel_2.Text = ['Battery Voltage: ' , num2str(volt_supplyCDU) ];
updateCDUStatus(app,valveStatusA);
% Uplink command status
uplinkflag = DATA_ERAU(84);
if uplinkflag == 1
    app.ReceivedLamp.Color = 'green';
else
    app.ReceivedLamp.Color = 'red';
end

% Send pointing to rotor
if (numSats > 6 && numSats < 13)
    % Compute the AZ/EL parameters for the GS
    and range.
    if app.GPSButton.Value == 1
        [az, el, range] =
        geodetic2aer(lat, lon, h, lat0, lon0, h0, app.spheroid);
    end
    app.ElLabel.Text = ['El: ' num2str(az
        +str2double(app.AzBiasEditField.Value))];
    app.ElevationGauge.Value=el
        +str2double(app.ElBiasEditField.Value);
    app.AzLabel.Text = ['Az: ' num2str(el
        +str2double(app.ElBiasEditField.Value))];
    app.AzGauge.Value = az-
        str2double(app.DeclinationEditField.Value);
    app.RangekmLabel.Text=['Range (km):'
        num2str(range/1000)];
end

% Send desired pointing to Arduino-Rotor
if (app.flag_GS == 1)
    app.ConnectingLabel.Text = 'Moving';
    app.ConnectingLabel.Visible = 'on';

    % Send messages
    try
        if app.AzElButton.Value == 1
            fprintf(app.serial_GS, '%s\n',
                ['ElAz', num2str(app.ElevationGauge.Value, '%03.0f'),
                    num2str(app.AzGauge.Value, '%03.0f')]);
        elseif app.AzOnlyButton.Value ==1
            fprintf(app.serial_GS, '%s\n',
                ['setAz', num2str(az-
                    str2double(app.DeclinationEditField.Value), '%03.0f')]);
        elseif app.ElOnlyButton.Value ==1
            fprintf(app.serial_GS, '%s\n',
                ['setEl', num2str(el+str2double(app.ElBiasEditField.Value), '%03.0f')]);
        elseif app.ManualButton.Value == 1
            % No control
            end
            pause(0.00001);
        catch
            end
        end
    end
catch
fprintf('GS Serial Connection Error. Retrying connection. Time: ');
fprintf(sprintf(string(datetime)));
fprintf('
');
set(app.gsPortStatus, 'Color', 'r');
fclose(app.serial_GS);
try
fopen(app.serial_GS);
set(app.gsPortStatus, 'Color', 'g');
catch
fclose(app.serial_GS);
fprintf('Failed to open GS serial port.

');
end
end
end
elseif (latCDU ~= 0)
%Compute the AZ/EL parameters for the GS
and range.

if app.GPSButton.Value == 1
[az,el,range] =
geodetic2aer(latCDU,lonCDU,altCDU,lat0,lon0,h0,app.spheroid);
end

app.ElevationGauge.Value=el+str2double(app.ElBiasEditField.Value);
app.AzGauge.Value = az-str2double(app.DeclinationEditField.Value);
app.sltRange.Text=num2str(range/1000);

%Send desired pointing to Arduino-Rotor
if (app.flag_GS == 1)
app.ConnectingLabel.Text = 'Moving';
app.ConnectingLabel.Visible = 'on';

%send messages
if app.AzElButton.Value == 1
fprintf(app.serial_GS,'%s
',['ElAz',num2str(app.ElevationGauge.Value,'%03.0f'),num2str(app.AzGauge.Value, '%03
 disp(num2str(app.AzGauge.Value, '%03.0f')));
elseif app.AzOnlyButton.Value ==1
fprintf(app.serial_GS,'%s
','setAz',num2str(az-str2double(app.DeclinationEditField.Value), '%03.0f')));
elseif app.ElOnlyButton.Value ==1
fprintf(app.serial_GS,'%s
',['El',num2str(app.ElevationGauge.Value,'%03.0f')));
end
end

B.3. GROUND STATION SOFTWARE
B.3. GROUND STATION SOFTWARE

```
fprintf(app.serial_GS, '%s
',
['setEl',num2str(el+str2double(app.ElBiasEditField.Value), '%03.0f')]);
elseif app.ManualButton.Value == 1
  % no control
  pause(0.00001);
end

app.TemperatureLabel.Text = ['Temperature: ',
num2str(tempCDUA)];
app.LastCodeLabel.Text = ['Last Code: ',
dec2hex(commandCodeA)];
app.ValveStatusLabel.Text = ['Status: ',
num2str(valveStatusA)];
app.AltitudeLabel_2.Text = ['Altitude: ',
num2str(altCDU)];
app.LatitudeLabel_2.Text = ['Latitude: ',
num2str(latCDU)];
app.LongitudeLabel_2.Text = ['Longitude: ',
num2str(lonCDU)];
app.GPSTimeLabel.Text = ['GPS Time: ',
num2str(hourCDU), ':', num2str(minCDU), ':', num2str(secCDU)];

%Plot Temperature Sensors Data
plot(app.TempInt, temp_ex, h/1000, 'b.' );
plot(app.TempInt, temp_in, h/1000, 'r.' );
plot(app.TempInt, temp_ex2, h/1000, 'g.' );
plot(app.TempInt, tempCDUA, h/1000, 'm.' );
legend(app.TempInt, 'External High', 'Internal', 'External Low', 'CDU', 'Location', 'northeast')
pause(0.00001);
end
toc(test)

%%%%%%%%%%%%%%%%%%%%%%
%                 if ((toc-activecmd_time > 900) && (exist('read_packet','var') ~= 0))
% activecmd_time = toc;
% % get RSSI
% fwrite(app.s,[43 43 43]);
% % wait for OK response
% pause(1.15)
% % disp('trying to read')
% read_packet = fread(app.s,app.s.BytesAvailable);
% fwrite(app.dataFile, read_packet);
% if read_packet(end-2:end) == [79; 75; 13]
%     disp('Read OK')
```

---

21
fwrite(app.s,[65 84 68 66 13]);
fwrite(app.s,[36 238]);

% Check active commanding
if  (app.AutomaticButton.Value == 1)
predTime = app.CUSFpredicted(:,5);

% Check timer
if  ((app.PredictionFileButton.Value == 1)&&(timerCheck>timeThreshold))
%Reset Timer
timerIni = tic;

if  (app.ManualButton2.Value == 1)
%Select altitude from GUI
alt = app.AltmSpinner.Value;

%Altitude during ascent or descent?
%Grab data accordingly
if  (app.AscentButton.Value == 1)
hConC = app.hC(1:app.maxC);
distC = abs(hConC-alt);
rowC = find(distC == min(distC));

elseif  (app.DescendButton.Value == 1)
hConC = app.hC(app.maxC:end);
distC = abs(hConC-alt);
rowC = find(distC == min(distC)) + (app.maxC-1);
end
end

if  (app.AutomaticButton.Value == 1)
predTime = app.CUSFpredicted(:,5);
timeNow = datetime - appstartTime;
vecTime = datevec(timeNow);
totalSecs = (vecTime(4)*3600) + (vecTime(5)*60) + (vecTime(6));

distSecs = abs(totalSecs - predTime);
rowC = find(distSecs == min(distSecs));
end
%Grab the data from the selected row. Only for the
selected pred. file
%Compute AZ/El for the rotor controller and range
for the GUI

lat=app.CUSFpredicted(rowC(1),2); % lat
lon=app.CUSFpredicted(rowC(1),3); % lon
h=app.CUSFpredicted(rowC(1),4); % alt

[az,el,range] =
geodetic2aer(lat,lon,h,lat0,lon0,h0,app.spheroid);

%Send desired pointing to Arduino-Rotor
if  (app.flag_GS == 1) %If the ground station is
connected
    app.ConnectingLabel.Text = 'Moving';
    app.ConnectingLabel.Visible = 'on';
    if  app.ManualButton.Value == 1
        % no control due to movement occurring via
        GS rotor controller
            elseif  app.AzElButton.Value == 1
                fprintf(app.serial_GS, '%s
' ,
                ['ElAz',num2str(el +str2double(app.ElBiasEditField.Value),'%03.0f'),num2str(az-
str2double(app.DeclinationEditField.Value), '%03.0f')));
            elseif  app.AzOnlyButton.Value ==1
                fprintf(app.serial_GS, '%s
' ,
                ['setAz',num2str(az-
str2double(app.DeclinationEditField.Value), '%03.0f')));
            elseif  app.ElOnlyButton.Value ==1
                fprintf(app.serial_GS, '%s
' ,
                ['setEl',num2str(el+str2double(app.ElBiasEditField.Value),'%03.0f'))]);

end

%Print Current Data from prediction file to the
labels and gauges
app.gpsLAT.Text  = num2str(lat);
app.gpsLONG.Text = num2str(lon);
app.gpsALT.Text  = num2str(h);
app.gpsFIX.Text  = "N/A";
app.gpsSATS.Text = "N/A";
app.gpsHOUR.Text = "N/A";
app.gpsMIN.Text  = "N/A";
app.gpsEC.Text = "N/A";
app.ElevationGauge.Value = el;
app.AzGauge.Value = az -
str2double(app.DeclinationEditField.Value);
app.sltRange.Text = num2str(range/1000);

% Delete previous plots for A and B properties of
GUI
delete(app.A);
delay(app.B);

% Set the data for the plots to be the values of
LLA for that specific prediction file
app.xdata = lon;
app.ydata = lat;
app.zdata = h/1000;

% Actually plot the prediction trajectory
app.A =
plot(app.Location2D,app.xdata,app.ydata, 'b*', 'LineWidth',1);
app.B =
plot3(app.Location3D,app.xdata,app.ydata,app.zdata, 'b*', 'LineWidth',1);

% pause(3);
end
pause(0.00002);

app.trackingStat.Color = 'white';
end

% Button pushed function: ConnectGSButton
function ConnectGSButtonPushed(app, event)
% Initialize Serial Communication with Arduino and MATLAB.
% The Arduino sends a Char and waits for MATLAB to respond
with the proper
% Char. If no errors, setup ok indication is visible.
app.serial_GS = serial(app.GSCOMPortDropDown.Value);

set(app.ConnectingLabel, 'Visible', 'on');

% Set serial parameters
app.serial_GS.InputBufferSize = 300000;
set(app.serial_GS, 'DataBits', 8);
set(app.serial_GS, 'StopBits', 1);
set(app.serial_GS, 'BaudRate', 230400);
set(app.serial_GS, 'Parity', 'none');

% Open the serial port
try
    fopen(app.serial_GS);
    set(app.gsPortStatus, 'Color', 'g');
catch err
    fclose(app.serial_GS);
    set(app.gsPortStatus, 'Color', 'r');
end

24
error('Make sure you select the correct Arduino COM Port.'); 
app.flag_GS = 1;
set(app.ConnectGSButton,'Enable','off');
set(app.DisconnectGSButton,'Enable','on');
while (app.serial_GS.BytesAvailable == 0)
end

a=fscanf(app.serial_GS,'%e');
fprintf(app.serial_GS,'%s
','getAz');

while (app.serial_GS.BytesAvailable == 0)
end

app.AzGauge.Value = fscanf(app.serial_GS,'%e');
fprintf(app.serial_GS,'%s
','getEl');
while (app.serial_GS.BytesAvailable == 0)
end

app.ElevationGauge.Value = fscanf(app.serial_GS,'%e');

set(app.ConnectingLabel,'Visible','off');
set(app.ConnectGSButton,'Enable','on');
set(app.DisconnectGSButton,'Enable','off');
%set(app.OKLabel,'Visible','off');
set(app.AutoButton,'Visible','off');

end

% Button pushed function: DisconnectGSButton
function DisconnectGSButtonPushed(app, event)

fclose(app.serial_GS);
delete(app.serial_GS);
clear app.serial_GS;
set(app.ConnectGSButton,'Enable','on');
set(app.ConnectingLabel,'Visible','off');
set(app.DisconnectGSButton,'Enable','off');
%set(app.OKLabel,'Visible','off');
set(app.AutoButton,'Visible','off');

end
% Callback function
function AutoButtonPushed(app, event)
    % function to load current position to gs_lat, lon and alt
    from gs gps
    fprintf(app.serial_GS, 'getLoc');
    location=fgetl(app.serial_GS);
    M =strsplit(location,',');
    while length(M) ~= 6
        fprintf(app.serial_GS, 'getLoc');
        location=fgetl(app.serial_GS);
        M =strsplit(location,',');
    end
    if string(M(1))=='lat'
        app.GS_Latitude.Value=str2double(cell2mat(M(2)));
    end
    if string(M(3))=='lon'
        app.GS_Longitude=str2double(cell2mat(M(4)));
    end
    if string(M(5))=='alt'
        app.GS_Altitude=str2double(cell2mat(M(6)));
    end
end

% Close request function: UIFigure
function UIFigureCloseRequest(app, event)
    delete(instrfindall);
    delete(app)
end

% Callback function
function GPS_Selection(app, event)
    %disp("GPS CHANGED!");
end

% Value changed function: GSCOMPortDropDown
function GSCOMPortDropDownValueChanged(app, event)
    app.GSCOMPortDropDown.Items = cellstr(seriallist);
    app.RxCOMPortDropDown.Items = app.GSCOMPortDropDown.Items;
end

% Value changed function: RxCOMPortDropDown
function RXCOMPortDropDownValueChanged(app, event)
    app.RxCOMPortDropDown.Items = cellstr(seriallist);
    app.GSCOMPortDropDown.Items = app.RxCOMPortDropDown.Items;
end

% Callback function
function CalibrateGSButtonPushed(app, event)
    prompt = ['You are about to perform an initial
calibration. Please set the GS-5500 to an azimuth of
180°, char(176), ' then select Next'];
end
type = questdlg(prompt, 'Initial Calibration', 'Next', 'Cancel');
switch type
    case 'Next'
        GSCal;
    case 'Cancel'
        return
end

% Callback function: LoadPredictionFileMenu,
% LoadPredictionfileButton
function LoadPredictionfileButtonPushed(app, event)
    pAns = questdlg('Which type of prediction path would you like to plot?',
        'Prediction Path Option', ...
        'University of Wyoming', 'CUSF', 'Cancel', 'CUSF');
    switch pAns
        case 'CUSF'
            [newfile, path] = uigetfile('*.csv', 'Prediction Path File', 'flight_path.csv');
            figure(app.UIFigure);
            if newfile ~= 0
                app.predFileFlag = app.predFileFlag + 1;
                predFile = fullfile(path, newfile);

                predictor
                % http://predict.habhub.org
                app.CUSFpredicted = csvread(predFile);
                app.ydata = app.CUSFpredicted(:,2); % lat
                app.xdata = app.CUSFpredicted(:,3); % lon
                app.zdata = app.CUSFpredicted(:,4); % alt
                app.wdata = app.CUSFpredicted(:,1); % time
                app.wdata = app.wdata - app.wdata(1);
                plot3(app.Location3D, app.xdata, app.ydata, app.zdata./1000, app.predFileCol(app.predFileFlag), 'LineWidth', 2)

                app.hC = app.CUSFpredicted(:,4); % alt
                app.maxC = find(app.hC == max(app.hC));
            end
        case 'Cancel'
        case 'University of Wyoming'
27
[newfile,path] = uigetfile('*.csv','Pediction Path File','flight_path.csv');
figure(app.UIFigure);
if newfile ~= 0
    predFile=fullfile(path,newfile);

    predFile=fullfile(path,newfile);

    % Predicted path plot from hab-hub.org
    % http://predict.habhub.org
    app.WYpredicted=csvread(predFile,3,1);
    app.xdata=app.WYpredicted(:,1); % lat
    app.ydata=app.WYpredicted(:,2); % lon
    app.zdata=app.WYpredicted(:,3); % alt

    %plot3(app.Location3D,app.xdata,app.ydata,app.zdata./1000,'r','LineWidth',2)
    %plot(app.Location2D,app.xdata,app.ydata,'r','LineWidth',2)

    app.xdata=app.WYpredicted(:,10); % Temperature
    if min(app.xdata) < app.TempInt.XLim(1)+5
        app.TempInt.XLim(1) = min(app.xdata) - 15;
    end

    plot(app.TempInt,app.xdata,app.zdata./1000, 'k' );  % Plot temperature prediction

end

%Prediction File Parameters
app.hC = app.CUSFpredicted(:,4); % alt
app.maxC = find(app.hC==max(app.hC));
app.BeginTrackingButton.Enable='on';

% case 'Create New CUSF'
% web('http://predict.habhub.org','-new','-noaddressbox', '-notoolbar');
% uiwait(msgbox('Opening HabHub.org Prediction tool. Save the file in csv format. Then Press OK.',...
% 'Get Prediction File'));
% [newfile,path] = uigetfile('*.csv','Pediction Path File','flight_path.csv');
% if newfile ~= 0
% %
% predFile=fullfile(path,newfile);
% from hab-hub.org predictor http://predict.habhub.org
predFile=csvread(predFile);
app.ydata=predicted(:,2); % lat
app.xdata=predicted(:,3); % lon
app.zdata=predicted(:,4); % alt
plot3(app.Location3D,app.xdata,app.ydata,app.zdata./1000,'y','LineWidth',2)
plot(app.Location2D,app.xdata,app.ydata,'y','LineWidth',2)
plot3(app.Location3D,lon0, lat0, h0/1000, 'y*', 'LineWidth',1)
plot(app.Location2D,lon0, lat0, 'y*')
end
end
% Callback function
function PredictionFileDropDownValueChanged(app, event)
    value = app.PredictionFileDropDown.Value;
    if  strcmp(value, 'CUSF')
        app.AltmSpinner.Limits(2) = max(app.CUSFpredicted(:,4));
    elseif  strcmp(value, 'Wyoming')
        app.AltmSpinner.Limits(2) = max(app.WYpredicted(:,3));
    end
end
% Menu selected function: RefreshCOMPortsMenu
function RefreshCOMPortsMenuSelected(app, event)
    app.RxCOMPortDropDown.Items = cellstr(seriallist);
    app.GS.COMPortDropDown.Items = app.RxCOMPortDropDown.Items;
end
% Callback function
function LaunchTimeButtonPushed(app, event)
    app.startTime = datetime;
end

B.3. GROUND STATION SOFTWARE

188
end

function GroundStationTestButtonPushed(app, event)

while(true)
    pause(3);
    lat0=app.myGSCoord(1);
    lon0=app.myGSCoord(2);
    h0=app.myGSCoord(3);
    alt = app.AltmSpinner.Value;
    app.hC = app.CUSFpredicted(:,4); % alt
    app.maxC = find(app.hC==max(app.hC));
    if  (app.AscentButton.Value == 1)
        hConC = app.hC(1:app.maxC);
        distC = abs(hConC-alt);
        rowC = find(distC == min(distC));
    elseif  (app.DescentButton.Value == 1)
        hConC = app.hC(app.maxC:end);
        distC = abs(hConC-alt);
        rowC = find(distC == min(distC)) + (app.maxC-1);
    end

    if  (app.ManualButton2.Value == 1)
        autoFlag = 0;
        %Grab the data from the selected row. Only for the
        %selected pred. file
        %Compute AZ/El for the rotor controller and range
        lat=app.CUSFpredicted(rowC(1),2); % lat
        lon=app.CUSFpredicted(rowC(1),3); % lon
        h=app.CUSFpredicted(rowC(1),4); % alt
        [az,el,range] =
        geodetic2aer(lat,lon,h,lat0,lon0,h0,app.spheroid);
        if  (app.flag_GS == 1) %If the ground station is
            app.ConnectingLabel.Text = 'Moving';
            app.ConnectingLabel.Visible = 'on';
            if  app.ManualButton.Value == 1
                % no control due to movement occuring via
                GS rotor controller
                elseif  app.AzElButton.Value == 1
                    fprintf(app.serial_GS,'%s
',[el, num2str(el+str2double(app.ElBiasEditField.Value),'%03.0f'), num2str(az-str2double(app.DeclinationEditField.Value), '%03.0f')));
                elseif  app.AzOnlyButton.Value ==1
                    fprintf(app.serial_GS,'%s
',[az, num2str(az+str2double(app.AzBiasEditField.Value),'%03.0f')));
                end
            else
                app.ConnectingLabel.Text = 'Disconnected';
                app.ConnectingLabel.Visible = 'off';
                % no control due to movement occuring via
                GS rotor controller
            end
        else
            fprintf(app.serial_GS,'
');
    end

end

B.3. GROUND STATION SOFTWARE

189
fprintf(app.serial_GS, '%s
', ['setAz', num2str(az-str2double(app.DeclinationEditField.Value), '%03.0f')]);
elseif app.ElOnlyButton.Value ==1
fprintf(app.serial_GS, '%s
', ['setEl', num2str(el+str2double(app.ElBiasEditField.Value), '%03.0f')]);
end
end

%Print Current Data from prediction file to the labels and gauges
app.gpsLAT.Text  = num2str(lat);
app.gpsLONG.Text = num2str(lon);
app.gpsALT.Text  = num2str(h);
app.gpsFIX.Text = "N/A" ;
app.gpsSATS.Text = "N/A" ;
app.gpsHOUR.Text = "N/A" ;
app.gpsMIN.Text = "N/A" ;
app.gpsSEC.Text = "N/A" ;
app.ElevationGauge.Value=el;
app.AzGauge.Value = az-str2double(app.DeclinationEditField.Value);
app.Rangekm.Text=num2str(range/1000);
end

%Delete previous plots for A and B properties of GUI
delete(app.A);
delete(app.B);
%Set the data for the plots to be the values of LLA for that specific prediction file
app.xdata = lon;
app.ydata = lat;
app.zdata = h/1000;
%Actually plot the prediction trajectory
app.A = plot(app.Location2D,app.xdata,app.ydata, 'b*' , 'LineWidth',1);
app.B = plot3(app.Location3D,app.xdata,app.ydata,app.zdata, 'b*' , 'LineWidth',1);
end

if (app.AutomaticButton.Value == 1)
    if autoFlag == 0
        for i = rowC:length(app.wdata)
pause(3);
            %Grab the data from the selected row. Only for the selected pred. fil
            lat=app.CUSFpredicted(i,2); % lat
            lon=app.CUSFpredicted(i,3); % lon
            h=app.CUSFpredicted(i,4); % alt
            [az,el,range] =
geodetic2aer(lat,lon,h,lat0,lon0,h0,app.spheroid);
    end

B.3. GROUND STATION SOFTWARE

190
if (app.flag_GS == 1) %If the ground station is connected
app.ConnectingLabel.Text = 'Moving';
app.ConnectingLabel.Visible = 'on';
if app.ManualButton.Value == 1
    % no control due to movement occuring
    via GS rotor controller
elseif app.AzElButton.Value == 1
    fprintf(app.serial_GS, '%s
', ['ElAz', num2str(el + str2double(app.ElBiasEditField.Value), '%03.0f'), num2str(az - str2double(app.DeclinationEditField.Value), '%03.0f')]);
elseif app.AzOnlyButton.Value ==1
    fprintf(app.serial_GS, '%s
', ['setAz', num2str(az - str2double(app.DeclinationEditField.Value), '%03.0f')]);
elseif app.ElOnlyButton.Value ==1f
    fprintf(app.serial_GS, '%s
', ['setEl', num2str(el + str2double(app.ElBiasEditField.Value), '%03.0f')]);
end
end

%Print Current Data from prediction file to the labels and gauges
app.gpsLAT.Text  = num2str(lat);
app.gpsLONG.Text = num2str(lon);
app.gpsALT.Text  = num2str(h);
app.gpsFIX.Text = "N/A" ;
app.gpsSATS.Text = "N/A" ;
app.gpsHOUR.Text = "N/A" ;
app.gpsMIN.Text = "N/A" ;
app.gpsSEC.Text = "N/A" ;
app.ElevationGauge.Value=el;
app.AzGauge.Value = az - str2double(app.DeclinationEditField.Value);
app.sltRange.Text=num2str(range/1000);

%Delete previous plots for A and B properties of GUI
delete(app.A);
delete(app.B);

%Set the data for the plots to be the values of LLA for that specific prediction file
app.xdata = lon;
app.ydata = lat;
app.zdata = h/1000;
%Actually plot the prediction trajectory
app.A = plot(app.Location2D,app.xdata,app.ydata,'b*','LineWidth',1);
app.B = plot3(app.Location3D,app.xdata,app.ydata,app.zdata,'b*','LineWidth',1);
if (i==length(app.wdata))
autoFlag = 1;
% Button pushed function: ReproduceFlightButton
function  ReproduceFlightButtonPushed(app, event)
    [newfile,path] = uigetfile( '*.bin' , 'Create Data File', 'data.bin');
    if  newfile == 0
        return;
    end
    filename=fullfile(path,newfile);
    s1=fopen(filename, 'r+');
    lat0=app.myGSCoord(1);
    lon0=app.myGSCoord(2);
    h0=app.myGSCoord(3);
    %packet headers
    id_erau = [160, 177]';
    id_umn = [66, 1]';
    id_cu_instr = [193,9]';
    id_cu_gond = [210,168]';

    rcvd_packets = 0;

    %External High Thermistor Coefficients:
    p1_ex = 0.4975;
    p2_ex = 2.342;
    p3_ex = 1.396;
    p4_ex = 16.54;
    p5_ex = -7.183;
    mean_ex = 547.6;
    std_ex = 236.2;

    %Extra External Low Thermistor Coefficients:
    p1_ex2 = -1.129;
    p2_ex2 = 4.101;
    p3_ex2 = -1.778;
    p4_ex2 = 15.41;
    p5_ex2 = -33.61;
    mean_ex2 = 386.2;
    std_ex2 = 255.8;

    %Internal Thermistor Coefficients:
    p1_in = -0.344;
    p2_in = -1.26;
    p3_in = -2.305;
p4_in = -13.18;
p5_in = 7.039;
mean_in = 844.7;
std_in = 97.26;

%Voltage ADC Calibration
p1_v = -0.003335;
p2_v = 1.082;
p3_v = 1.653;
mean_v = 510;
std_v = 334.6;

%Ascent Rate Monitor Variables
prevAlt = 35;
prevTime = 0;

% Expected packet lengths
ERAU_LENGTH = 47+20;
UMN_LENGTH = 46+4;
CU_I_LENGTH = 81-18+4;
CU_G_LENGTH = 70-24+4;

TOTAL_PACKET_LENGTH = ERAU_LENGTH+UMN_LENGTH+CU_I_LENGTH;
messages = zeros(TOTAL_PACKET_LENGTH,1,'double');
fvals = [2.5, 5.5, 11.5, 23.5, 47.5, 95.5, 191.5, 383.5, 767.5];  %[Hz] freq. of spectral averaged values

%Read and Process Data
read_packet = fread(s1);
i = 1;
while i < (length(read_packet)-TOTAL_PACKET_LENGTH-2)
    %Look for the start of a packet.
    if ((read_packet(i:i+1)==id_erau))
        %Check if the packet has been completely received.
        if {{(read_packet(i+Erau_LENGTH:i+1+Erau_LENGTH)==id_umn) & ...
            (read_packet(i+Erau_LENGTH+umn_LENGTH:i+1+Erau_LENGTH+umn_LENGTH)==id_cu_instr | ...
                read_packet(i+Erau_LENGTH+umn_LENGTH:i+1+Erau_LENGTH+umn_LENGTH)==id_cu_gond)}

        messages(1:TOTAL_PACKET_LENGTH)=read_packet(i:i+TOTAL_PACKET_LENGTH-1);
i = i+10000;

B.3. GROUND STATION SOFTWARE
Grab ERAU Packet

```matlab
if (messages(1:2) == id_erau)
    for packet_index = 1:ERAU_LENGTH
        DATA_ERAU(packet_index) = messages(packet_index);
        end
    packet_num_erau = packet_num_erau + 1;
end
else DATA_ERAU = [];
end
```

Grab SPS30 Packet

```matlab
if (messages(ERAU_LENGTH+1:ERAU_LENGTH+2) == id_umn)
    for packet_index = 1:UMN_LENGTH
        DATA_UMN(packet_index) = messages(ERAU_LENGTH + packet_index);
        end
    else DATA_UMN = [];
end
```

Grab CU Instrument Packet

```matlab
if (messages(ERAU_LENGTH+UMN_LENGTH+1:ERAU_LENGTH+UMN_LENGTH+2) == id_cu_instr)
    for packet_index = 1:CU_I_LENGTH
        DATA_CU_I(packet_index) = messages(ERAU_LENGTH + UMN_LENGTH + packet_index);
        end
    else DATA_CU_I = [];
end
```

Grab CU Gondola Packet

```matlab
if (messages(ERAU_LENGTH+UMN_LENGTH+1:ERAU_LENGTH+UMN_LENGTH+2) == id_cu_gond)
    for packet_index = 1:CU_G_LENGTH
        DATA_CU_G(packet_index) = messages(ERAU_LENGTH + UMN_LENGTH + packet_index);
        end
    else DATA_CU_G = [];
end
```
Parse ERAU data

```matlab
if ~isempty(DATA_ERAU)
    packet_time =
typecast(uint8(DATA_ERAU(5:8)), 'uint32');
    packet_time = double((packet_time)/1000);
    app.PacketTimeLabel.Text = ['Packet Time: ', num2str(packet_time/60)];
%
gps
lat =
double(typecast(uint8(DATA_ERAU(9:12)), 'int32'))/10000000;
lon =
double(typecast(uint8(DATA_ERAU(13:16)), 'int32'))/10000000;
h =
double(typecast(uint8(DATA_ERAU(17:20)), 'int32'))/100;
stat = DATA_ERAU(21);
numSats = DATA_ERAU(22);
utcHour = DATA_ERAU(23);
utcMin = DATA_ERAU(24);
utcSec = DATA_ERAU(25);
%
app.gpsLAT.Text = num2str(lat);
app.gpsLONG.Text = num2str(lon);
app.gpsALT.Text = num2str(h);
app.gpsFIX.Text = num2str(stat);
app.gpsSATS.Text = num2str(numSats);
app.gpsHOUR.Text = [num2str(utcHour), ': '];
app.gpsMIN.Text = [num2str(utcMin), ': '];
app.gpsSEC.Text = num2str(utcSec);
%
app.TemperatureLabel.Text = ['Temperature: ', num2str(tempCDU)];
%
app.LastCodeLabel.Text = ['Last Code: ', dec2hex(commandCode)];
%
Plot Ublox GPS Data
plot3(app.Location3D, lon, lat,
h/1000,'b.','LineWidth',1)
plot(app.Location2D, lon, lat,'b.')
%
Calculate Ascent Rate
newAlt = h;
newTime = packet_time;
ascentRate = double((newAlt - prevAlt)/(newTime - prevTime));
prevAlt = newAlt;
prevTime = newTime;
num2str(ascentRate);
```
temp = typecast(uint8(DATA_ERAU(26:27)), 'uint16');
temp = double(temp);
temp = (temp-mean_ex)/std_ex;
temp_ex = p1_ex*temp^4 + p2_ex*temp^3 + p3_ex*temp^2 + p4_ex*temp + p5_ex;

% Internal Temperature Conversion

temp = typecast(uint8(DATA_ERAU(28:29)), 'uint16');
temp = double(temp);
temp = (temp-mean_in)/std_in;
temp_in = p1_in*temp^4 + p2_in*temp^3 + p3_in*temp^2 + p4_in*temp + p5_in;

% External Low Temperature Conversion

temp = typecast(uint8(DATA_ERAU(30:31)), 'uint16');
temp = double(temp);
temp = (temp-mean_ex2)/std_ex2;
temp_ex2 = p1_ex2*temp^4 + p2_ex2*temp^3 + p3_ex2*temp^2 + p4_ex2*temp + p5_ex2;

% Accelerometer Data

accel_z = double(typecast(uint8(DATA_ERAU(38:39)), 'uint16'))/1000;
app.AccelerationZLabel.Text = ['Acceleration (Z): ',num2str(accel_z)];

% Voltage Monitor

typecast(uint8(DATA_ERAU(32:33)), 'uint16');
voltage = double(voltage);
voltage = (voltage-mean_v)/std_v;
volt_supply = 3*(p1_v*voltage^2 + p2_v*voltage + p3_v);
app.BatteryVoltageLabel.Text = ['Battery Voltage: ', num2str(volt_supply)];

% CDU Data

tempCDUA = double(typecast(uint8(DATA_ERAU(48:49)), 'int16'))/100;
commandCodeA = DATA_ERAU(47);
valveStatusA = DATA_ERAU(50);
altCDU = double(typecast(uint8(DATA_ERAU(51:54)), 'int32'))/100;
latCDU = double(typecast(uint8(DATA_ERAU(55:58)), 'int32'))/10000000;
lonCDU =
double(typecast(uint8(DATA_ERAU(59:62)),'int32'))/10000000;
hourCDU = DATA_ERAU(63);
minCDU = DATA_ERAU(64);
secCDU = DATA_ERAU(65);

app.TemperatureLabel.Text = ['Temperature: ', num2str(tempCDUA)];
app.LastCodeLabel.Text = ['Last Code: ',
dec2hex(commandCodeA)];
app.ValveStatusLabel.Text = ['Status: ',
num2str(valveStatusA)];
app.AltitudeLabel_2.Text = ['Alt: ',
num2str(altCDU)];
app.LatitudeLabel_2.Text = ['Lat: ',
num2str(latCDU)];
app.LongitudeLabel_2.Text = ['Lon: ',
num2str(lonCDU)];
app.GPSTimeLabel.Text = ['Time: ',
num2str(hourCDU), ':', num2str(minCDU), ':', num2str(secCDU)];

%Plot Temperature Sensors Data
plot(app.TempInt, temp_ex, h/1000, 'b.' );
plot(app.TempInt, temp_in, h/1000, 'r.' );
plot(app.TempInt, temp_ex2, h/1000, 'g.' );
plot(app.TempInt, tempCDUA, h/1000, 'm.' );
legend(app.TempInt, 'External High', 'Internal', 'External Low', 'CDU', 'Location', 'northwest')

Parse CU Data

instrument

if ~isempty(DATA_CU_I)

%Spectral averages
CW_SA_0 = double(swapbytes(typecast(uint8(DATA_CU_I(8:9)),'uint16'))/(-4680)+2;  %[log10(V^2/Hz])

CW_SA_1 =
double(swapbytes(typecast(uint8(DATA_CU_I(10:11)),'uint16'))/(-4680)+2;  %[^*^]

CW_SA_2 =
double(swapbytes(typecast(uint8(DATA_CU_I(12:13)),'uint16'))/(-4680)+2;

CW_SA_3 =
double(swapbytes(typecast(uint8(DATA_CU_I(14:15)),'uint16'))/(-4680)+2;

CW_SA_4 =
double(swapbytes(typecast(uint8(DATA_CU_I(16:17)),'uint16'))/(-4680)+2;
B.3. GROUND STATION SOFTWARE

```matlab
% Additional values for CW/HW
 CW_meas_Vo1 = double(swapbytes(typecast(uint8(DATA_CU_I(48:49)), 'int16')))*2.5/32768;

 HW_meas_Vol =
 double(swapbytes(typecast(uint8(DATA_CU_I(58:59)), 'int16')))*2.5/32768;

 % -----Update display values-----
 app.CWVoltageLabel.Text = [‘CW Voltage (V): ’, num2str(CW_meas_Vo1)];
 app.HWVoltageLabel.Text = [‘HW Voltage (V): ’, num2str(HW_meas_Vol)];
```
B.3. GROUND STATION SOFTWARE

Parse UMN Data

if ~isempty(DATA_UMN)
    sps_hits = typecast(uint8(DATA_UMN(21:22)),'uint16');
    umn_pressure = typecast(uint8(DATA_UMN(17:20)),'single');

    app.HitsLabel.Text = ['Hits: ', num2str(sps_hits)];
    app.PressureLabel.Text = ['Pressure: ', num2str(umn_pressure), ' psi'];
end

pause(0.0001);
else
    i = i+1;
end
pause(0.0002);
end
pause(0.0002);

%             id_scient=[160,177]';
%             id_gps=[192,209]';
%             messages=zeros(1,100);
%             rcvd_packets = 0;
%             packets_sci = 0;
%             packets_gps = 0;
%             lost_packets = 0;
%             lost_total = 0;
%             packet_num = 0;
%             new_packet_number = 0;
%             range = 0;
%             timer_1 = tic;
%
%             %External Thermistor Coefficients:
%             p1_ex = 13.6;
%             p2_ex = -6.838;
%             p3_ex = 20.3;
%             p4_ex = -14.81;
%
%             mean_ex = 423.8;
%             std_ex = 358.5;
%
%             %Internal Thermistor Coefficients:
%             p1_in = -5.2;
%             p2_in = -9.875;
%             p3_in = -24.22;
%             p4_in = 19.94;
%
%             mean_in = 742.8;
%             std_in = 224.8;
%
%             %Initial Position
%             lat = lat0;
%             lon = lon0;
%             h = h0;
%             %Ascent Rate Monitor Variables
%             prevAlt = 35;
%             prevTime = 0;
B.3. GROUND STATION SOFTWARE

```matlab
% ascentRate = 0;
% min_gps = 10*4;
% min_sci = 10*125;
read_packet = fread(sl);
for i=1:(length(read_packet)-102)
    %Loop for the start of a scientific or
    if((read_packet(i:i+1)==id_scient) | 
    (read_packet(i+100:i+101)==id_gps))
        %Check if the packet has been
        if ((read_packet(i+100:i+101)==id_scient) | (read_packet(i+100:i+101)==id_gps))
            rcvd_packets = rcvd_packets+1;
            messages(1:100)=read_packet(i:i+99);
            packet_num =
            typecast(uint8(messages(3:4)),'uint16');
            packet_num = double(packet_num);
            if (rcvd_packets == 1)
                prev_packet_number = packet_num;
            end
            %Check if it is a Scientific
            if (read_packet(i:i+1)==id_scient)
                packets_sci = packets_sci+1;
                timer_2 = toc(timer_1);
                if (packets_sci==min_sci)
                    packets_sci=0;
                %External Temperature
                temp =
                typecast(uint8(messages(5:6)),'uint16');
                temp = double(temp);
                temp = (temp-mean_ex)/
std_ex;
                p2_ex*temp^2 + p3_ex*temp + p4_ex;
                %Internal Temperature
                temp =
                typecast(uint8(messages(7:8)),'uint16');
                temp = double(temp);
                temp = (temp-mean_in)/
std_in;
```

```
42
```
temp_in = p1_in*temp^3 + p2_in*temp^2 + p3_in*temp + p4_in;

% Voltage Monitor
voltage = typecast(uint8(messages(13:14)),'uint16');
voltage = double(voltage);
volt_supply = 3*((3.3/1023)*voltage);

app.BatteryVoltageLabel.Text = ['Battery Voltage: ', num2str(volt_supply)];

% 9DoF Monitor
accel_X = typecast(uint8(messages(37:38)),'int16');
accel_x = double(accel_X)/1000;
accel_Y = typecast(uint8(messages(39:40)),'int16');
accel_y = double(accel_Y)/1000;
accel_Z = typecast(uint8(messages(41:42)),'int16');
accel_z = double(accel_Z)/1000;


% Plot Temperature Sensors Data
plot(app.TempInt, temp_ex, h/1000, 'b.');
plot(app.TempInt, temp_in, h/1000, 'r.');

% Plot Accelerometer Data
plot(app.Voltage, datetime, volt_supply,'Marker','.', 'Color','r');
plot(app.Voltage, datetime, accel_x,'Marker','.', 'Color','r');
plot(app.Voltage, datetime, accel_y,'Marker','.', 'Color','g');
plot(app.Voltage, datetime, accel_z, 'Marker','.', 'Color','b');

% Print received and lost packets information
```
% lost_total = lost_total + double(lost_packets);
% app.ReceivedPackets.Text = ['Received Packets: ',num2str(rcvd_packets)];
% app.LostPackets.Text = ['Lost Packets: ',num2str(lost_total)];
% app.LostPacketsLabel.Text = ['%Losses: ',num2str(100*(lost_total/(lost_total+rcvd_packets)))];
% lost_packets = 0;
% pause(0.001);
end

% Check if it is a GPS Packet and parse it
elseif ((read_packet(i:i+1)==id_gps))
    packets_gps = packets_gps+1;
    if (packets_gps==min_gps)
        packets_gps=0;
    end
    lat = double(typecast(uint8(messages(5:8)),'int32'))/10000000;
    lon = double(typecast(uint8(messages(9:12)),'int32'))/10000000;
    h = double(typecast(uint8(messages(13:16)),'int32'));
    stat = messages(17);
    numSats = messages(18);
    utcHour = messages(19);
    utcMin = messages(20);
    utcSec = messages(21);
    packetTime = double(typecast(uint8(messages(27:30)),'uint32'));
    newAlt = h;
    newTime = double(packetTime/1000);
    ascentRate = double((newAlt - prevAlt)/(newTime - prevTime));
    app.AscendRate.Text = num2str(ascentRate);
    prevAlt = h;
    prevTime = double(packetTime/1000);
end
```

B.3. GROUND STATION SOFTWARE
geodetic2aer(lat, lon, h, lat0, lon0, h0, app.spheroid);

% Plot GS Location.
plot3(app.Location3D, lon0, lat0, h0/1000, 'r*', 'LineWidth',3)
plot(app.Location2D, lon0, lat0, 'r*')

% Plot Ublox GPS Data
plot3(app.Location3D, lon, lat, h/1000,'b*','LineWidth',1)
plot(app.Location2D, lon, lat,'b*')

% Print Current GPS Data
app.gpsLAT.Text  = num2str(lat);
app.gpsLONG.Text = num2str(lon);
app.gpsALT.Text  = num2str(h);
app.gpsFIX.Text = num2str(stat);
app.gpsSATS.Text = num2str(numSats);
app.gpsHOUR.Text = [num2str(utcHour), ':'
];
app.gpsMIN.Text = [num2str(utcMin), ':']
; app.gpsSEC.Text = num2str(utcSec);

app.ElevationGauge.Value=el+str2double(app.ElBiasEditField.Value);
app.AzGauge.Value = az-
str2double(app.DeclinationEditField.Value);
app.sltRange.Text=num2str(range/1000);

% Compute the number of lost packets in this considered data block
if (((new_packet_number-
prev_packet_number)~=1)&&((new_packet_number-
prev_packet_number)~=-65535))
45
% if ((new_packet_number-prev_packet_number)>1)
% lost_packets = lost_packets + (new_packet_number - prev_packet_number - 1);
% end
% if ((new_packet_number-prev_packet_number)<0)
% lost_packets = lost_packets + (65535 - prev_packet_number) + new_packet_number;
% end
% prev_packet_number = packet_num;
% end
%
% Callback function
function RestartRxPortButtonPushed(app, event)
% Serial for the radio communication or file
fclose(app.s);
app.s = serial(app.RxCOMPortDropDown.Value);

% Set serial parameters
app.s.InputBufferSize = 1000000;
set(app.s, 'DataBits', 8);
set(app.s, 'StopBits', 1);
set(app.s, 'BaudRate', 230400);
set(app.s, 'Parity', 'none');

% Open the serial port
try
fopen(app.s);
catch err
fclose(app.s);
warndlg('Make sure you select the correct Radio COM Port.');
end
end

% Button pushed function: EndTrackingButton
function EndTrackingButtonPushed(app, event)
prompt = {'Stop reading in data?'};
title = 'Are you sure?';
answ = questdlg(prompt,title,'Yes','No');
switch answ
  case 'Yes'
    % Serial for the radio communication or file
    fclose(app.s);
    app.loopBreak = 1;
    app.BeginTrackingButton.Enable = 'on';
app.EndTrackingButton.Enable = 'off';
%
app.ListeningLamp.Color = 'white';
%
app.UseX8tosetGSPositionButton.Enable = 'off';
end
end

% Button pushed function: CutLineButton
function CutLineButtonPushed(app, event)
    app.cutCnt = app.cutCnt + 1;
    if app.cutCnt == 1
        app.cut1.Color = 'red';
    elseif app.cutCnt == 2
        app.cut2.Color = 'red';
    else
        app.cut3.Color = 'green';
        fwrite(app.s, [36 221]);
    end
end

% Button pushed function: GetRSSIButton
function GetRSSIButtonPushed(app, event)

    Get RSSI For last packet received
    fwrite(app.s, [43 43 43]);
    % wait for OK response
    pause(1.15)
    disp('trying to read')
    read_packet = fread(app.s, app.s.BytesAvailable);
    fwrite(app.dataFile, read_packet);
    if read_packet(end-2:end) == [79; 75; 13]
        disp('Read OK')
        fwrite(app.s, [65 84 68 66 13]);
        pause(.1)
        if app.s.BytesAvailable > 2
            disp('trying to read RSSI')
            RSSI = fread(app.s, app.s.BytesAvailable);
            app.RSSILabel.Text = ['RSSI: ', char((RSSI))];
            fwrite(app.s, [65 84 67 78 13]);
        end
    end
end

% Button pushed function: ValveOpenButton
function ValveOpenButtonPushed(app, event)
    disp("Sending Open Command")
    fwrite(app.s, [36 187])
end

% Button pushed function: ValveCloseButton
function ValveCloseButtonPushed(app, event)
    disp('Sending Close Command')
    fwrite(app.s,[36 204])
end

% Button pushed function: CheckButton
function CheckButtonPushed(app, event)
    disp('Sending Check Command')
    fwrite(app.s,[36 238])
end

% Component initialization
methods (Access = private)
% Create UIFigure and components
function createComponents(app)
    % Create UIFigure and hide until all components are created
    app.UIFigure = uifigure('Visible', 'off');
    app.UIFigure.Position = [100 100 1263 725];
    app.UIFigure.Name = 'UI Figure';
    app.UIFigure.CloseRequestFcn = createCallbackFcn(app,
    @UIFigureCloseRequest, true);
    app.UIFigure.Scrollable = 'on';

    % Create Location3D
    app.Location3D = uiaxes(app.UIFigure);
    title(app.Location3D, '3D position')
    xlabel(app.Location3D, 'Lon')
    ylabel(app.Location3D, 'Lat')
    app.Location3D.PlotBoxAspectRatio = [1.00470588235294 1 1];
    app.Location3D.FontSize = 11;
    app.Location3D.Box = 'on';
    app.Location3D.Color = [0.902 0.902 0.902];
    app.Location3D.NextPlot = 'replace';
    app.Location3D.XGrid = 'on';
    app.Location3D.YGrid = 'on';
    app.Location3D.ZGrid = 'on';
    app.Location3D.Position = [15 366 372 353];

    % Create Location2D
    app.Location2D = uiaxes(app.UIFigure);
    title(app.Location2D, '2D position')
    xlabel(app.Location2D, 'Lon')
    ylabel(app.Location2D, 'Lat')
    app.Location2D.DataAspectRatio = [1 1 1];
    app.Location2D.PlotBoxAspectRatio = [1 1 1];
    app.Location2D.FontSize = 11;
    app.Location2D.Color = [0.902 0.902 0.902];
    app.Location2D.NextPlot = 'replace';
    app.Location2D.Position = [423 365 398 353];
B.3. GROUND STATION SOFTWARE

% Create TempInt
app.TempInt = uiaxes(app.UIFigure);
title(app.TempInt, 'Temperature')
xlabel(app.TempInt, 'Temperature (C)')
ylabel(app.TempInt, 'Altitude (km)')
app.TempInt.PlotBoxAspectRatio = [1.69333333333333 1 1];
app.TempInt.XLim = [-60 40];
app.TempInt.YLim = [-1 50];
app.TempInt.NextPlot = 'replace';
app.TempInt.Position = [838 483 416 235];

% Create ConfigureMenu
app.ConfigureMenu = uimenu(app.UIFigure);
app.ConfigureMenu.Text = 'Configure';

% Create RefreshCOMPortsMenu
app.RefreshCOMPortsMenu = uimenu(app.ConfigureMenu);
app.RefreshCOMPortsMenu.MenuSelectedFcn = createCallbackFcn(app, @RefreshCOMPortsMenuSelected, true);
app.RefreshCOMPortsMenu.Accelerator = 'R';
app.RefreshCOMPortsMenu.Text = 'Refresh COM Ports';

% Create LoadMapsMenu
app.LoadMapsMenu = uimenu(app.ConfigureMenu);
app.LoadMapsMenu.MenuSelectedFcn = createCallbackFcn(app, @LoadPositionButtonPushed, true);
app.LoadMapsMenu.Accelerator = 'M';
app.LoadMapsMenu.Text = 'Load Maps';

% Create LoadPredictionFileMenu
app.LoadPredictionFileMenu = uimenu(app.ConfigureMenu);
app.LoadPredictionFileMenu.MenuSelectedFcn = createCallbackFcn(app, @LoadPredictionFileButtonPushed, true);
app.LoadPredictionFileMenu.Enable = 'off';
app.LoadPredictionFileMenu.Accelerator = 'P';
app.LoadPredictionFileMenu.Text = 'Load Prediction File';

% Create BeginTrackingButton
app.BeginTrackingButton = uibutton(app.UIFigure, 'push');
app.BeginTrackingButton.ButtonPushedFcn = createCallbackFcn(app, @BeginTrackingButtonPushed, true);
app.BeginTrackingButton.BackgroundColor = [0.5882 0.9608 0.4784];
app.BeginTrackingButton.Position = [13 306 100 22];
app.BeginTrackingButton.Text = 'Begin Tracking';

% Create ElevationGauge
app.ElevationGauge = uigauge(app.UIFigure, 'semicircular');
app.ElevationGauge.Limits = [0 180];
app.ElevationGauge.MajorTicks = [0 45 90 135 180];
app.ElevationGauge.ScaleColors = [0.470588235294118 0.670588235294118 0.188235294117647];
app.ElevationGauge.ScaleColorLimits = [0 4];
app.ElevationGauge.Position = [15 201 118 64];

% Create AzGauge
app.AzGauge = uigauge(app.UIFigure, 'linear');
app.AzGauge.Limits = [0 360];
app.AzGauge.MajorTicks = [0 45 90 135 180 225 270 315 360];
app.AzGauge.ScaleColors = [0.470588235294118 0.670588235294118 0.188235294117647];
app.AzGauge.ScaleColorLimits = [0 5];
app.AzGauge.Position = [13 147 302 40];

% Create ConnectGSButton
app.ConnectGSButton = uibutton(app.UIFigure, 'push');
app.ConnectGSButton.ButtonPushedFcn = createCallbackFcn(app, @ConnectGSButtonPushed, true);
app.ConnectGSButton.Position = [180 306 88 22];
app.ConnectGSButton.Text = 'Connect GS';

% Create ConnectingLabel
app.ConnectingLabel = uilabel(app.UIFigure);
app.ConnectingLabel.HorizontalAlignment = 'right';
app.ConnectingLabel.VerticalAlignment = 'top';
app.ConnectingLabel.Visible = 'off';
app.ConnectingLabel.Position = [321 280 66 15];
app.ConnectingLabel.Text = 'Connecting';

% Create DisconnectGSButton
app.DisconnectGSButton = uibutton(app.UIFigure, 'push');
app.DisconnectGSButton.ButtonPushedFcn = createCallbackFcn(app, @DisconnectGSButtonPushed, true);
app.DisconnectGSButton.Enable = 'off';
app.DisconnectGSButton.Position = [271 306 88 22];
app.DisconnectGSButton.Text = 'Disconnect GS';

% Create GSCOMPortDropDownLabel
app.GSCOMPortDropDownLabel = uilabel(app.UIFigure);
app.GSCOMPortDropDownLabel.HorizontalAlignment = 'right';
app.GSCOMPortDropDownLabel.VerticalAlignment = 'top';
app.GSCOMPortDropDownLabel.Position = [179 338 78 15];
app.GSCOMPortDropDownLabel.Text = 'GS COM Port';

% Create GSCOMPortDropDown
app.GSCOMPortDropDown = uidropdown(app.UIFigure);
app.GSCOMPortDropDown.Items = {};
app.GSCOMPortDropDown.ValueChangedFcn = createCallbackFcn(app, @GSCOMPortDropDownValueChanged, true);
app.GSCOMPortDropDown.Position = [267 333 70 22];
app.GSCOMPortDropDown.Value = {};

% Create RxCOMPortDropDownLabel
app.RxCOMPortDropDownLabel = uilabel(app.UIFigure);
app.RxCOMPortDropDownLabel.HorizontalAlignment = 'right';
app.RxCOMPortDropDownLabel.VerticalAlignment = 'top';
app.RxCOMPortDropDownLabel.Position = [15 337 76 15];
app.RxCOMPortDropDownLabel.Text = 'Rx COM Port';

% Create RxCOMPortDropDown
app.RxCOMPortDropDown = uidropdown(app.UIFigure);
app.RxCOMPortDropDown.Items = []; app.RxCOMPortDropDown.ValueChangedFcn = createCallbackFcn(app, @RXCOMPortDropDownValueChanged, true);
app.RxCOMPortDropDown.Position = [102 333 61 22];
app.RxCOMPortDropDown.Value = {};

% Create gpsLAT
app.gpsLAT = uilabel(app.UIFigure);
app.gpsLAT.BackgroundColor = [1 1 1];
app.gpsLAT.HorizontalAlignment = 'right';
app.gpsLAT.VerticalAlignment = 'top';
app.gpsLAT.Position = [377 316 61 15];
app.gpsLAT.Text = '';

% Create gpsLONG
app.gpsLONG = uilabel(app.UIFigure);
app.gpsLONG.BackgroundColor = [1 1 1];
app.gpsLONG.HorizontalAlignment = 'right';
app.gpsLONG.VerticalAlignment = 'top';
app.gpsLONG.Position = [442 316 61 15];
app.gpsLONG.Text = '';

% Create gpsALT
app.gpsALT = uilabel(app.UIFigure);
app.gpsALT.BackgroundColor = [1 1 1];
app.gpsALT.HorizontalAlignment = 'right';
app.gpsALT.VerticalAlignment = 'top';
app.gpsALT.Position = [508 316 61 15];
app.gpsALT.Text = '';

% Create gpsFIX
app.gpsFIX = uilabel(app.UIFigure);
app.gpsFIX.BackgroundColor = [1 1 1];
app.gpsFIX.HorizontalAlignment = 'right';
app.gpsFIX.VerticalAlignment = 'top';
app.gpsFIX.Position = [579 316 25 15];
app.gpsFIX.Text = '';

% Create LatitudeLabel
app.LatitudeLabel = uilabel(app.UIFigure);
app.LatitudeLabel.HorizontalAlignment = 'right';
app.LatitudeLabel.VerticalAlignment = 'top';
app.LatitudeLabel.Position = [381 338 49 15];
app.LatitudeLabel.Text = 'Latitude';

% Create LongitudeLabel
app.LongitudeLabel = uilabel(app.UIFigure);
app.LongitudeLabel.HorizontalAlignment = 'right';
app.LongitudeLabel.VerticalAlignment = 'top';
app.LongitudeLabel.Position = [443 338 58 15];
app.LongitudeLabel.Text = 'Longitude';
% Create AltitudeLabel
app.AltitudeLabel = uilabel(app.UIFigure);
app.AltitudeLabel.VerticalAlignment = 'top';
app.AltitudeLabel.Position = [508 336 46 18];
app.AltitudeLabel.Text = 'Altitude';

% Create FixQLabel
app.FixQLabel = uilabel(app.UIFigure);
app.FixQLabel.VerticalAlignment = 'top';
app.FixQLabel.Position = [574 339 37 15];
app.FixQLabel.Text = 'Fix Q.';

% Create gpsSATS
app.gpsSATS = uilabel(app.UIFigure);
app.gpsSATS.BackgroundColor = [1 1 1];
app.gpsSATS.HorizontalAlignment = 'right';
app.gpsSATS.VerticalAlignment = 'top';
app.gpsSATS.Position = [619 316 25 15];
app.gpsSATS.Text = '';

% Create gpsHOUR
app.gpsHOUR = uilabel(app.UIFigure);
app.gpsHOUR.BackgroundColor = [1 1 1];
app.gpsHOUR.HorizontalAlignment = 'right';
app.gpsHOUR.VerticalAlignment = 'top';
app.gpsHOUR.Position = [671 316 25 15];
app.gpsHOUR.Text = '';

% Create gpsMIN
app.gpsMIN = uilabel(app.UIFigure);
app.gpsMIN.BackgroundColor = [1 1 1];
app.gpsMIN.HorizontalAlignment = 'right';
app.gpsMIN.VerticalAlignment = 'top';
app.gpsMIN.Position = [701 316 25 15];
app.gpsMIN.Text = '';

% Create gpsSEC
app.gpsSEC = uilabel(app.UIFigure);
app.gpsSEC.BackgroundColor = [1 1 1];
app.gpsSEC.HorizontalAlignment = 'right';
app.gpsSEC.VerticalAlignment = 'top';
app.gpsSEC.Position = [731 316 25 15];
app.gpsSEC.Text = '';

% Create SatsLabel
app.SatsLabel = uilabel(app.UIFigure);
app.SatsLabel.VerticalAlignment = 'top';
app.SatsLabel.Position = [613 338 37 15];
app.SatsLabel.Text = '#Sats';

% Create GPSUTCTimeLabel
app.GPSUTCTimeLabel = uilabel(app.UIFigure);
app.GPSUTCTimeLabel.VerticalAlignment = 'top';
app.GPSUTCTimeLabel.Position = [673 337 88 15];
app.GPSUTCTimeLabel.Text = 'GPS UTC Time';

% Create AscentRate
app.AscentRate = uilabel(app.UIFigure);
app.AscentRate.BackgroundColor = [1 1 1];
app.AscentRate.HorizontalAlignment = 'right';
app.AscentRate.VerticalAlignment = 'top';
app.AscentRate.Position = [771 316 50 15];
app.AscentRate.Text = ''; 

% Create ADRateLabel
app.ADRateLabel = uilabel(app.UIFigure);
app.ADRateLabel.HorizontalAlignment = 'right';
app.ADRateLabel.Position = [772 330 54 22];
app.ADRateLabel.Text = 'A/D Rate';

% Create GroundStationTestButton
app.GroundStationTestButton = uibutton(app.UIFigure, 'push');
app.GroundStationTestButton.ButtonPushedFcn = createCallbackFcn(app, @GroundStationTestButtonPushed, true);
app.GroundStationTestButton.Enable = 'off';
app.GroundStationTestButton.Position = [180 276 122 22];
app.GroundStationTestButton.Text = 'Ground Station Test';

% Create gsPortStatus
app.gsPortStatus = uilamp(app.UIFigure);
app.gsPortStatus.Position = [344 336 20 20];
app.gsPortStatus.Color = [1 0 0];

% Create TabGroup
app.TabGroup = uitabgroup(app.UIFigure);
app.TabGroup.Position = [377 186 473 112];

% Create PositionTab
app.PositionTab = uitab(app.TabGroup);
app.PositionTab.Title = 'Position';

% Create AltitudeEditFieldLabel
app.AltitudeEditFieldLabel = uilabel(app.PositionTab);
app.AltitudeEditFieldLabel.HorizontalAlignment = 'center';
app.AltitudeEditFieldLabel.Position = [163 63 46 15];
app.AltitudeEditFieldLabel.Text = 'Altitude';

% Create GS_Altitude
app.GS_Altitude = uieditfield(app.PositionTab, 'text');
app.GS_Altitude.Editable = 'off';
app.GS_Altitude.HorizontalAlignment = 'right';
app.GS_Altitude.Position = [155 37 61 22];
app.GS_Altitude.Value = '45';

% Create LongitudeEditFieldLabel
app.LongitudeEditFieldLabel = uilabel(app.PositionTab);
B.3. GROUND STATION SOFTWARE

app.LongitudeEditFieldLabel.HorizontalAlignment = 'center';
app.LongitudeEditFieldLabel.VerticalAlignment = 'top';
app.LongitudeEditFieldLabel.Position = [86 63 58 15];
app.LongitudeEditFieldLabel.Text = 'Longitude';

% Create GS_Longitude
app.GS_Longitude = uieditfield(app.PositionTab, 'text');
app.GS_Longitude.Editable = 'off';
app.GS_Longitude.HorizontalAlignment = 'right';
app.GS_Longitude.Position = [84 37 61 22];
app.GS_Longitude.Value = '-81.048094';

% Create LatitudeEditFieldLabel
app.LatitudeEditFieldLabel = uilabel(app.PositionTab);
app.LatitudeEditFieldLabel.HorizontalAlignment = 'center';
app.LatitudeEditFieldLabel.VerticalAlignment = 'top';
app.LatitudeEditFieldLabel.Position = [21 63 48 15];
app.LatitudeEditFieldLabel.Text = 'Latitude';

% Create GS_Latitude
app.GS_Latitude = uieditfield(app.PositionTab, 'text');
app.GS_Latitude.Editable = 'off';
app.GS_Latitude.HorizontalAlignment = 'right';
app.GS_Latitude.Position = [14 37 61 22];
app.GS_Latitude.Value = '29.187904';

% Create DeclinationEditFieldLabel
app.DeclinationEditFieldLabel = uilabel(app.PositionTab);
app.DeclinationEditFieldLabel.HorizontalAlignment = 'right';
app.DeclinationEditFieldLabel.VerticalAlignment = 'top';
app.DeclinationEditFieldLabel.Position = [297 56 65 22];
app.DeclinationEditFieldLabel.Text = 'Declination';

% Create DeclinationEditField
app.DeclinationEditField = uieditfield(app.PositionTab, 'text');
app.DeclinationEditField.HorizontalAlignment = 'right';
app.DeclinationEditField.Position = [300 37 61 22];
app.DeclinationEditField.Value = '-5.5';

% Create ElBiasEditFieldLabel
app.ElBiasEditFieldLabel = uilabel(app.PositionTab);
app.ElBiasEditFieldLabel.HorizontalAlignment = 'right';
app.ElBiasEditFieldLabel.VerticalAlignment = 'top';
app.ElBiasEditFieldLabel.Position = [224 56 65 22];
app.ElBiasEditFieldLabel.Text = 'El Bias';

% Create ElBiasEditField
app.ElBiasEditField = uieditfield(app.PositionTab, 'text');
app.ElBiasEditField.HorizontalAlignment = 'right';
app.ElBiasEditField.Position = [230 37 61 22];
app.ElBiasEditField.Value = '0';

% Create LoadPositionButton
app.LoadPositionButton = uibutton(app.PositionTab, 'push');
app.LoadPositionButton.ButtonPushedFcn = createCallbackFcn(app, @LoadPositionButtonPushed, true);
app.LoadPositionButton.Position = [14 6 88 22];
app.LoadPositionButton.Text = 'Load Position';

% Create LoadPredictionFileButton
app.LoadPredictionFileButton = uibutton(app.PositionTab, 'push');
app.LoadPredictionFileButton.ButtonPushedFcn = createCallbackFcn(app, @LoadPredictionFileButtonPushed, true);
app.LoadPredictionFileButton.Position = [116 6 118 22];
app.LoadPredictionFileButton.Text = 'Load Prediction file';

% Create ReproduceFlightButton
app.ReproduceFlightButton = uibutton(app.PositionTab, 'push');
app.ReproduceFlightButton.ButtonPushedFcn = createCallbackFcn(app, @ReproduceFlightButtonPushed, true);
app.ReproduceFlightButton.Enable = 'off';
app.ReproduceFlightButton.Position = [250 6 107 22];
app.ReproduceFlightButton.Text = 'Reproduce Flight';

% Create TrackByTab
app.TrackByTab = uitab(app.TabGroup);
app.TrackByTab.Title = 'Track By';

% Create ButtonGroup
app.ButtonGroup = uibuttongroup(app.TrackByTab);
app.ButtonGroup.Position = [3 2 466 85];

% Create GPSButton
app.GPSButton = uiradiobutton(app.ButtonGroup);
app.GPSButton.Text = 'GPS';
app.GPSButton.Position = [11 63 57 15];
app.GPSButton.Value = true;

% Create PredictionFileButton
app.PredictionFileButton = uiradiobutton(app.ButtonGroup);
app.PredictionFileButton.Text = 'Prediction File';

% Create PredictionFileButtonGroup
app.PredictionFileButtonGroup = uibuttongroup(app.ButtonGroup);
app.PredictionFileButtonGroup.Title = 'Prediction Part';
app.PredictionFileButtonGroup.Position = [110 21 100 55];
% Create AscentButton
B.3. GROUND STATION SOFTWARE

app.AscentButton = uiradiobutton(app.PredictionFileButtonGroup);
app.AscentButton.Text = 'Ascent';
app.AscentButton.Value = true;

% Create DescentButton
app.DescentButton = uiradiobutton(app.PredictionFileButtonGroup);
app.DescentButton.Text = 'Descent';

% Create PredictionMode
app.PredictionMode = uibuttongroup(app.ButtonGroup);
app.PredictionMode.Title = 'Prediction Mode';
app.PredictionMode.Position = [217 22 100 55];

% Create ManualButton2
app.ManualButton2 = uiradiobutton(app.PredictionMode);
app.ManualButton2.Value = true;

% Create AutomaticButton
app.AutomaticButton = uiradiobutton(app.PredictionMode);
app.AutomaticButton.Text = 'Automatic';
app.AutomaticButton.Position = [11 -1 75 22];

% Create AltmSpinnerLabel
app.AltmSpinnerLabel = uilabel(app.ButtonGroup);
app.AltmSpinnerLabel.HorizontalAlignment = 'right';
app.AltmSpinnerLabel.VerticalAlignment = 'top';
app.AltmSpinnerLabel.Position = [329 36 46 22];
app.AltmSpinnerLabel.Text = 'Alt. [m]:';

% Create AltmSpinner
app.AltmSpinner = uispinner(app.ButtonGroup);
app.AltmSpinner.Limits = [1 Inf];
app.AltmSpinner.Position = [390 39 67 22];
app.AltmSpinner.Value = 1;

% Create OptionsTab
app.OptionsTab = uitab(app.TabGroup);
app.OptionsTab.Title = 'Options';

% Create AntenaTrakingModeButtonGroup
app.AntenaTrakingModeButtonGroup = uibuttongroup(app.OptionsTab);
app.AntenaTrakingModeButtonGroup.Title = 'Antena Traking Mode';
app.AntenaTrakingModeButtonGroup.Position = [8 12 145 66];

% Create AzElButton
app.AzElButton =
uiradiobutton(app.AntenaTrakingModeButtonGroup);
app.AzElButton.Text = 'Az/El';
app.AzElButton.Position = [7 20 57 15];
app.AzElButton.Value = true;

% Create AzOnlyButton
app.AzOnlyButton =
uiradiobutton(app.AntenaTrakingModeButtonGroup);
app.AzOnlyButton.Text = 'Az Only';
app.AzOnlyButton.Position = [71 20 64 15];

% Create ElOnlyButton
app.ElOnlyButton =
uiradiobutton(app.AntenaTrakingModeButtonGroup);
app.ElOnlyButton.Text = 'El Only';
app.ElOnlyButton.Position = [71 2 63 15];

% Create ManualButton
app.ManualButton =
uiradiobutton(app.AntenaTrakingModeButtonGroup);
app.ManualButton.Position = [7 2 61 15];

% Create EndTrackingButton
app.EndTrackingButton = uibutton(app.UIFigure, 'push');
app.EndTrackingButton.ButtonPushedFcn =
createCallbackFcn(app, @EndTrackingButtonPushed, true);
app.EndTrackingButton.BackgroundColor = [0.9608 0.6 0.6];
app.EndTrackingButton.Position = [13 276 100 22];
app.EndTrackingButton.Text = 'End Tracking';

% Create trackingStat
app.trackingStat = uilamp(app.UIFigure);
app.trackingStat.Position = [132 307 20 20];
app.trackingStat.Color = [1 0 0];

% Create ControlledDescentSystemPanel
app.ControlledDescentSystemPanel = uipanel(app.UIFigure);
app.ControlledDescentSystemPanel.Title = 'Controlled Descent System';
app.ControlledDescentSystemPanel.Position = [377 8 473 164];

% Create TemperatureLabel
app.TemperatureLabel =
uilabel(app.ControlledDescentSystemPanel);
app.TemperatureLabel.Position = [10 18 186 22];
app.TemperatureLabel.Text = 'Temperature: ';

% Create LatitudeLabel_2
app.LatitudeLabel_2 =
uilabel(app.ControlledDescentSystemPanel);
app.LatitudeLabel_2.Position = [10 95 100 22];
app.LatitudeLabel_2.Text = 'Latitude:';

% Create LongitudeLabel_2
app.LongitudeLabel_2 = uilabel(app.ControlledDescentSystemPanel);
app.LongitudeLabel_2.Position = [10 76 100 22];
app.LongitudeLabel_2.Text = 'Longitude:';

% Create AltitudeLabel_2
app.AltitudeLabel_2 = uilabel(app.ControlledDescentSystemPanel);
app.AltitudeLabel_2.Position = [10 115 102 22];
app.AltitudeLabel_2.Text = 'Altitude:';

% Create GPSTimeLabel
app.GPSTimeLabel = uilabel(app.ControlledDescentSystemPanel);
app.GPSTimeLabel.Position = [10 58 100 22];
app.GPSTimeLabel.Text = 'GPS Time:';

% Create BatteryVoltageLabel_2
app.BatteryVoltageLabel_2 = uilabel(app.ControlledDescentSystemPanel);
app.BatteryVoltageLabel_2.Position = [10 37 157 22];
app.BatteryVoltageLabel_2.Text = 'Battery Voltage: ';

% Create LastCodeLabel
app.LastCodeLabel = uilabel(app.ControlledDescentSystemPanel);
app.LastCodeLabel.Position = [10 2 103 22];
app.LastCodeLabel.Text = 'Last Code:';

% Create TabGroup2
app_TABGroup2 = uitabgroup(app.ControlledDescentSystemPanel);
app_TABGroup2.Position = [207 3 260 137];

% Create ValveStatusTab
app.ValveStatusTab = uitab(app_TABGroup2);
app.ValveStatusTab.Title = 'Valve Status';

% Create VentclosedLampLabel
app.VentclosedLampLabel = uilabel(app.ValveStatusTab);
app.VentclosedLampLabel.HorizontalAlignment = 'right';
app.VentclosedLampLabel.Position = [10 77 68 22];
app.VentclosedLampLabel.Text = 'Vent closed';

% Create VentclosedLamp
app.VentclosedLamp = uilamp(app.ValveStatusTab);
app.VentclosedLamp.Position = [93 80 16 16];
app.VentclosedLamp.Color = [1 1 1];

% Create VentopenLampLabel
app.VentopenLampLabel = uilabel(app.ValveStatusTab);
app.VentopenLampLabel.HorizontalAlignment = 'right';
app.VentopenLampLabel.Position = [18 57 60 22];
app.VentopenLampLabel.Text = 'Vent open';

% Create VentopenLamp
app.VentopenLamp = uilamp(app.ValveStatusTab);
app.VentopenLamp.Position = [93 60 16 16];
app.VentopenLamp.Color = [1 1 1];

% Create LineCutLampLabel
app.LineCutLampLabel = uilabel(app.ValveStatusTab);
app.LineCutLampLabel.HorizontalAlignment = 'right';
app.LineCutLampLabel.Position = [18 37 60 22];
app.LineCutLampLabel.Text = 'Line Cut';

% Create LineCutLamp
app.LineCutLamp = uilamp(app.ValveStatusTab);
app.LineCutLamp.Position = [93 40 16 16];
app.LineCutLamp.Color = [1 1 1];

% Create HPOffLampLabel
app.HPOffLampLabel = uilabel(app.ValveStatusTab);
app.HPOffLampLabel.HorizontalAlignment = 'right';
app.HPOffLampLabel.Position = [138 77 68 22];
app.HPOffLampLabel.Text = 'HP Off';

% Create HPOffLamp
app.HPOffLamp = uilamp(app.ValveStatusTab);
app.HPOffLamp.Position = [221 80 16 16];
app.HPOffLamp.Color = [1 1 1];

% Create HPOnLampLabel
app.HPOnLampLabel = uilabel(app.ValveStatusTab);
app.HPOnLampLabel.HorizontalAlignment = 'right';
app.HPOnLampLabel.Position = [146 57 60 22];
app.HPOnLampLabel.Text = 'HP On';

% Create HPOnLamp
app.HPOnLamp = uilamp(app.ValveStatusTab);
app.HPOnLamp.Position = [221 60 16 16];
app.HPOnLamp.Color = [1 1 1];

% Create InBoundsLampLabel
app.InBoundsLampLabel = uilabel(app.ValveStatusTab);
app.InBoundsLampLabel.HorizontalAlignment = 'right';
app.InBoundsLampLabel.Position = [146 37 60 22];
app.InBoundsLampLabel.Text = 'In Bounds';

% Create InBoundsLamp
app.InBoundsLamp = uilamp(app.ValveStatusTab);
app.InBoundsLamp.Position = [221 40 16 16];
app.InBoundsLamp.Color = [1 1 1];

% Create NoSignalLampLabel
app.NoSignalLampLabel = uilabel(app.ValveStatusTab);
app.NoSignalLampLabel.HorizontalAlignment = 'right';
app.NoSignalLampLabel.Position = [81 12 60 22];
app.NoSignalLampLabel.Text = 'No Signal';

% Create NoSignalLamp
app.NoSignalLamp = uilamp(app.ValveStatusTab);
app.NoSignalLamp.Position = [156 15 16 16];
app.NoSignalLamp.Color = [1 1 1];

% Create ValveControlTab
app.ValveControlTab = uitab(app.TabGroup2);
app.ValveControlTab.Title = 'Valve Control';

% Create cut1
app.cut1 = uilamp(app.ValveControlTab);
app.cut1.Position = [156 12 16 16];
app.cut1.Color = [1 1 1];

% Create cut2
app.cut2 = uilamp(app.ValveControlTab);
app.cut2.Position = [186 12 16 16];
app.cut2.Color = [1 1 1];

% Create cut3
app.cut3 = uilamp(app.ValveControlTab);
app.cut3.Position = [215 12 16 16];
app.cut3.Color = [1 1 1];

% Create CutLineButton
app.CutLineButton = uibutton(app.ValveControlTab, 'push');
app.CutLineButton.ButtonPushedFcn = createCallbackFcn(app, @CutLineButtonPushed, true);
app.CutLineButton.Position = [29 9 100 22];
app.CutLineButton.Text = 'Cut Line';

% Create HeatingPadsSwitchLabel
app.HeatingPadsSwitchLabel = uilabel(app.ValveControlTab);
app.HeatingPadsSwitchLabel.HorizontalAlignment = 'center';
app.HeatingPadsSwitchLabel.Enable = 'off';
app.HeatingPadsSwitchLabel.Visible = 'off';
app.HeatingPadsSwitchLabel.Position = [161 89 78 22];
app.HeatingPadsSwitchLabel.Text = 'Heating Pads';

% Create HeatingPadsSwitch
app.HeatingPadsSwitch = uiswitch(app.ValveControlTab, 'slider');
app.HeatingPadsSwitch.Enable = 'off';
app.HeatingPadsSwitch.Visible = 'off';
app.HeatingPadsSwitch.Position = [177 70 45 20];

% Create RecievedLampLabel
app.RecievedLampLabel = uilabel(app.ValveControlTab);
app.RecievedLampLabel.HorizontalAlignment = 'right';
app.RecievedLampLabel.Position = [145 71 55 22];
app.RecievedLampLabel.Text = 'Recieved';

% Create RecievedLamp
app.RecievedLamp = uilamp(app.ValveControlTab);
app.RecievedLamp.Position = [215 71 20 20];
app.RecievedLamp.Color = [1 1 1];

% Create ValveOpenButton
app.ValveOpenButton = uibutton(app.ValveControlTab, 'push');
app.ValveOpenButton.ButtonPushedFcn = createCallbackFcn(app, @ValveOpenButtonPushed, true);
app.ValveOpenButton.Position = [29 61 100 22];
app.ValveOpenButton.Text = 'Valve Open';

% Create ValveCloseButton
app.ValveCloseButton = uibutton(app.ValveControlTab, 'push');
app.ValveCloseButton.ButtonPushedFcn = createCallbackFcn(app, @ValveCloseButtonPushed, true);
app.ValveCloseButton.Position = [29 35 100 22];
app.ValveCloseButton.Text = 'Valve Close';

% Create CheckButton
app.CheckButton = uibutton(app.ValveControlTab, 'push');
app.CheckButton.ButtonPushedFcn = createCallbackFcn(app, @CheckButtonPushed, true);
app.CheckButton.Position = [29 87 100 22];
app.CheckButton.Text = 'Check';

% Create SensorDataPanel
app.SensorDataPanel = uipanel(app.UIFigure);
app.SensorDataPanel.Title = 'Sensor Data:';
app.SensorDataPanel.FontWeight = 'bold';
app.SensorDataPanel.Position = [888 9 339 97];

% Create PressureLabel
app.PressureLabel = uilabel(app.SensorDataPanel);
app.PressureLabel.Position = [8 55 173 22];
app.PressureLabel.Text = 'Pressure: ';

% Create AccelerationZLabel
app.AccelerationZLabel = uilabel(app.SensorDataPanel);
app.AccelerationZLabel.Position = [167 36 169 22];
app.AccelerationZLabel.Text = 'Acceleration (Z): ';

% Create TemperatureLabel
app.TemperatureLabel = uilabel(app.SensorDataPanel);
app.TemperatureLabel.Position = [8 36 173 22];
app.TemperatureLabel.Text = 'Temperature: ';

% Create BatteryVoltageLabel
app.BatteryVoltageLabel = uilabel(app.SensorDataPanel);
B.3. GROUND STATION SOFTWARE

app.BatteryVoltageLabel.VerticalAlignment = 'top';
app.BatteryVoltageLabel.Position = [167 53 158 22];
app.BatteryVoltageLabel.Text = 'Battery Voltage: ';

% Create GyroscopeZLabel
app.GyroscopeZLabel = uilabel(app.SensorDataPanel);
app.GyroscopeZLabel.Position = [167 17 169 22];
app.GyroscopeZLabel.Text = 'Gyroscope (Z):' ;

% Create LossesPanel
app.LossesPanel = uipanel(app.UIFigure);
app.LossesPanel.Title = 'Losses';
app.LossesPanel.FontWeight = 'bold';
app.LossesPanel.Position = [132 42 233 92];

% Create ReceivedPackets
app.ReceivedPackets = uilabel(app.LossesPanel);
app.ReceivedPackets.VerticalAlignment = 'top';
app.ReceivedPackets.Position = [15 39 212 15];
app.ReceivedPackets.Text = 'Received Packets: ';

% Create curpacknum
app.curpacknum = uilabel(app.LossesPanel);
app.curpacknum.VerticalAlignment = 'top';
app.curpacknum.Position = [14 47 213 22];
app.curpacknum.Text = 'Current Packet Number: ';

% Create ExpectedPackets
app.ExpectedPackets = uilabel(app.LossesPanel);
app.ExpectedPackets.VerticalAlignment = 'top';
app.ExpectedPackets.Position = [15 15 212 22];
app.ExpectedPackets.Text = 'Expected Packets: ';

% Create TotalLossLabel
app.TotalLossLabel = uilabel(app.LossesPanel);
app.TotalLossLabel.Position = [14 0 176 22];
app.TotalLossLabel.Text = '% Total Loss: ';

% Create RSSILabel
app.RSSILabel = uilabel(app.UIFigure);
app.RSSILabel.VerticalAlignment = 'top';
app.RSSILabel.Position = [170 15 191 14];
app.RSSILabel.Text = 'RSSI: ';

% Create GetRSSIButton
app.GetRSSIButton = uibutton(app.UIFigure, 'push');
app.GetRSSIButton.ButtonPushedFcn = createCallbackFcn(app, @GetRSSIButtonPushed, true);
app.GetRSSIButton.Position = [41 11 100 22];
app.GetRSSIButton.Text = 'Get RSSI';

% Create ImageFeedLabel
app.ImageFeedLabel = uilabel(app.UIFigure);
app.ImageFeedLabel.HorizontalAlignment = 'center';
app.ImageFeedLabel.FontWeight = 'bold';
app.ImageFeedLabel.Position = [1022 440 72 22];
app.ImageFeedLabel.Text = 'Image Feed';

% Create ImageNumberLabel
app.ImageNumberLabel = uilabel(app.UIFigure);
app.ImageNumberLabel.HorizontalAlignment = 'center';
app.ImageNumberLabel.Position = [994 119 128 22];
app.ImageNumberLabel.Text = 'Image Number: ';

% Create Image
app.Image = uimage(app.UIFigure);
app.Image.Position = [896 147 342 294];

% Create AntennaPointingPanel
app.AntennaPointingPanel = uipanel(app.UIFigure);
app.AntennaPointingPanel.Title = 'Antenna Pointing';
app.AntennaPointingPanel.FontWeight = 'bold';
app.AntennaPointingPanel.Position = [13 41 112 93];

% Create AzLabel
app.AzLabel = uilabel(app.AntennaPointingPanel);
app.AzLabel.Position = [9 45 82 22];
app.AzLabel.Text = 'Az: ';

% Create ElLabel
app.ElLabel = uilabel(app.AntennaPointingPanel);
app.ElLabel.Position = [9 18 82 22];
app.ElLabel.Text = 'El: ';

% Create Panel
app.Panel = uipanel(app.UIFigure);
app.Panel.Position = [142 193 223 75];

% Create RangekmLabel
app.RangekmLabel = uilabel(app.Panel);
app.RangekmLabel.Position = [6 128 22];
app.RangekmLabel.Text = 'Range (km):';

% Create SerialBufferLabel
app.SerialBufferLabel = uilabel(app.Panel);
app.SerialBufferLabel.Position = [6 191 22];
app.SerialBufferLabel.Text = 'Serial Buffer: ';

% Create PacketTimeLabel
app.PacketTimeLabel = uilabel(app.Panel);
app.PacketTimeLabel.Position = [6 227 22];
app.PacketTimeLabel.Text = 'Packet Time: '

% Show the figure after all components are created
app.UIFigure.Visible = 'on';
% App creation and deletion methods (Access = public)

% Construct app
function app = HAB_GS_DEV_v2
% Create UIFigure and components
createComponents(app)

% Register the app with App Designer
registerApp(app, app.UIFigure)

% Execute the startup function
runStartupFcn(app, @startupFcn)

if nargout == 0
    clear app
end

end

% Code that executes before app deletion
function delete(app)

% Delete UIFigure when app is deleted
delete(app.UIFigure)
end

end

Error using load
'200km.map' is not found in the current folder or on the MATLAB path, but exists in:
C:\Users\Julio\OneDrive\MURI Balloon\Personal Folders\JulioG\Software\Ground Station

Change the MATLAB current folder or add its folder to the MATLAB path.

Error in HAB_GS_DEV_v2/startupFcn (line 459)
load('200km.map', 'ZA', '-mat');
B.4 PCB Schematics and Layouts

B.4.1 Bus
B.4.2 CDU
B.4. PCB SCHEMATICS AND LAYOUTS
Appendix C

AQM Reference

C.1 AQM Software
C.1. AQM SOFTWARE

Main

Julio Guardado

```cpp
/****************************************************************************
NAME: AQM.ino
PURPOSE: Ground based air quality station.
AUTHORS: Julio Cesar Guardado
/
#include //UMN GPS library
#include // The SD library allows for reading from and writing to SD cards.
#include // Electrically-Erasable Programmable Read-Only Memory
#include
#include // API for alphasense OPC N3
#include // Serial Peripheral Interface - communication protocol
#include //barometer
#include //xbee3 api communication
#include "WiFiEsp.h"
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include
#include
#include "Definitions.h"
bool debug = true;
char c;
void setup() {
    // Begin Communication Lines
    Serial.begin(115200);   // Serial Monitor
    Serial1.begin(9600);    // GPS
    Serial4.begin(115200);  // Xbee3
    SPI.begin();            // OPCN3
    Wire.begin();           // MS5611 / CCS811 / TMP102
    Wire2.begin();          // SPS30
    delay(50);
    Serial.print("Starting Init...");
    //************ WIFI ************
    WiFi.init(&Serial4);
    // check for the presence of the shield
    if (WiFi.status() == WL_NO_SHIELD) {
        Serial.println("WiFi shield not present");
        // don't continue
        while (true);
    }
    // attempt to connect to WiFi network
    while (status != WL_CONNECTED) {
        Serial.print("Attempting to connect to WPA SSID: ");
        Serial.println(ssid);
```
4/13/22, 7:07 PM Main

58     // Connect to WPA/WPA2 network
59     status = WiFi.begin(ssid, pass);
60   }
61
62   //**********SD CARD**********/
63   while (!SD.begin(chipSelect)) {
64       Serial.println("Insert SD Card");
65       delay(200);
66   }
67   Serial.println("SD OK");
68
69   //Read and set file number and name
70         address = 0;
71         fileNum = EEPROM.read(address);
72         fileNum = fileNum + 1;
73         EEPROM.write(address, fileNum);
74         fileN = "data";
75         String m1 = fileN + fileNum;
76         fileName1 = m1 + ext;
77
78   //*************** Initialize Sensors ***************/
79         initGPS();
80   // initTMP();
81         initBaro();
82         opcInit();
83         cssInit();
84   // initSPS();
85
86   // Internal ADC Setup
87         analogReadResolution(10);
88         adc->setReference(ADC_REFERENCE::REF_3V3, ADC_0);
89         Serial.println("); Serial.println(");
90         Serial.println("Finished init");
91         Serial.println("); Serial.println(");
92 }  
93
94   void loop() {
95     //***************Maintain Server Connection*************/
96         MQTT_connect();
97     
98     if (Serial.available()) {
99         while (1);
100     }
101 
102     //***************GPS Data Monitoring***************/
103         updateGPS();
104 
105     //*************** Update Sensors ***************/
106         updateSensors();
107 
108     //*************** Upload Data ***************/
109     if (updateTimer > UPDATE_INTERVAL) {
110         updateTimer = 0;
111         publishData();
112     }
113     
114     //***************PERIODICALLY SAVE SD CARD*************/
115     if ((millis() - timeSD) > 5000) {  // Save once every 5 sec
116         appendPos = dataLog.position();
117         dataLog.close();
118         dataLog = SD.open(fileName1.c_str(), FILE_WRITE);
119         timeSD = millis();
120         dataLog.seek(appendPos);
C.1. AQM SOFTWARE

```c
//
}
}
```
Data

Julio Guardado

void write_data() {
    //******** IDENTIFIER ********/
    dataPack[0] = id_erau[0];
    dataPack[1] = id_erau[1];
    dataPack[2] = packNum;
    dataPack[3] = packNum >> 8;
    packNum += 1;

    //******** TIMESTAMP ********/
    dataPack[4] = timer;
    dataPack[5] = timer >> 8;
    dataPack[6] = timer >> 16;
    dataPack[7] = timer >> 24;

    //*******GPS DATA*******
    dataPack[8] = (int32_t) (alt * 100);
    dataPack[9] = (int32_t) (alt * 100) >> 8;
    dataPack[10] = (int32_t) (alt * 100) >> 16;
    dataPack[11] = (int32_t) (alt * 100) >> 24;

    dataPack[12] = (int32_t) (latitude * 10000000);
    dataPack[13] = (int32_t) (latitude * 10000000) >> 8;
    dataPack[14] = (int32_t) (latitude * 10000000) >> 16;
    dataPack[15] = (int32_t) (latitude * 10000000) >> 24;

    dataPack[16] = (int32_t) (longitude * 10000000);
    dataPack[17] = (int32_t) (longitude * 10000000) >> 8;
    dataPack[18] = (int32_t) (longitude * 10000000) >> 16;
    dataPack[19] = (int32_t) (longitude * 10000000) >> 24;

    dataPack[20] = numSats;

    dataPack[21] = utcHour;
    dataPack[22] = utcMin;
    dataPack[23] = utcSec;

    //******** BATTERY MONITOR ********/
    voltage = analogRead(VOLTAGE);
    dataPack[24] = voltage;
    dataPack[25] = voltage >> 8;

    //******** TMP102*******
    byte* point_tmp = (byte*) &temperature;
    dataPack[26] = point_tmp[0];
    dataPack[27] = point_tmp[1];
    dataPack[28] = point_tmp[2];
    dataPack[29] = point_tmp[3];

    //******** BAROMETER*******/
void updateSensors() {
    // Sensors
    if (breakoutTimer > BREAKOUT_INTERVAL) {
        breakoutTimer = 0;
    }
    byte* point_temp = (byte*)&mstemp;
    dataPack[30] = point_temp[0];
    dataPack[31] = point_temp[1];
    dataPack[32] = point_temp[2];
    dataPack[33] = point_temp[3];
    // pressure
    byte* point_pres = (byte*)&pressure;
    dataPack[34] = point_pres[0];
    dataPack[35] = point_pres[1];
    dataPack[36] = point_pres[2];
    dataPack[37] = point_pres[3];
    // histogram
    write_counter = 37;
    for (int i = 0; i < 24; i++) {
        dataPack[write_counter] = hist.binCounts[i];
        dataPack[write_counter + 1] = hist.binCounts[i] >> 8;
        write_counter += 2;
    }
    // PM1
    byte* point_pm1 = (byte*)&hist.pm1;
    dataPack[write_counter++] = point_pm1[0];
    dataPack[write_counter++] = point_pm1[1];
    dataPack[write_counter++] = point_pm1[2];
    dataPack[write_counter++] = point_pm1[3];
    // PM2.5
    byte* point_pm2_5 = (byte*)&hist.pm2_5;
    dataPack[write_counter++] = point_pm2_5[0];
    dataPack[write_counter++] = point_pm2_5[1];
    dataPack[write_counter++] = point_pm2_5[2];
    dataPack[write_counter++] = point_pm2_5[3];
    // PM10
    byte* point_pm10 = (byte*)&hist.pm10;
    dataPack[write_counter++] = point_pm10[0];
    dataPack[write_counter++] = point_pm10[1];
    dataPack[write_counter++] = point_pm10[2];
    dataPack[write_counter++] = point_pm10[3];
    // temperature
    byte* point_tempOPC = (byte*)&tempOPC;
    dataPack[write_counter++] = point_tempOPC[0];
    dataPack[write_counter++] = point_tempOPC[1];
    dataPack[write_counter++] = point_tempOPC[2];
    dataPack[write_counter++] = point_tempOPC[3];
    // humidity
    byte* point_hum = (byte*)&humidity;
    dataPack[write_counter++] = point_hum[0];
    dataPack[write_counter++] = point_hum[1];
    dataPack[write_counter++] = point_hum[2];
    dataPack[write_counter++] = point_hum[3];
    // Stop byte
    dataPack[write_counter] = stopByte;
}
void updateSensors() {
    // Sensors
    if (breakoutTimer > BREAKOUT_INTERVAL) {
        breakoutTimer = 0;
    }
    byte* point_temp = (byte*)&mstemp;
    dataPack[30] = point_temp[0];
    dataPack[31] = point_temp[1];
    dataPack[32] = point_temp[2];
    dataPack[33] = point_temp[3];
    // pressure
    byte* point_pres = (byte*)&pressure;
    dataPack[34] = point_pres[0];
    dataPack[35] = point_pres[1];
    dataPack[36] = point_pres[2];
    dataPack[37] = point_pres[3];
    // histogram
    write_counter = 37;
    for (int i = 0; i < 24; i++) {
        dataPack[write_counter] = hist.binCounts[i];
        dataPack[write_counter + 1] = hist.binCounts[i] >> 8;
        write_counter += 2;
    }
    // PM1
    byte* point_pm1 = (byte*)&hist.pm1;
    dataPack[write_counter++] = point_pm1[0];
    dataPack[write_counter++] = point_pm1[1];
    dataPack[write_counter++] = point_pm1[2];
    dataPack[write_counter++] = point_pm1[3];
    // PM2.5
    byte* point_pm2_5 = (byte*)&hist.pm2_5;
    dataPack[write_counter++] = point_pm2_5[0];
    dataPack[write_counter++] = point_pm2_5[1];
    dataPack[write_counter++] = point_pm2_5[2];
    dataPack[write_counter++] = point_pm2_5[3];
    // PM10
    byte* point_pm10 = (byte*)&hist.pm10;
    dataPack[write_counter++] = point_pm10[0];
    dataPack[write_counter++] = point_pm10[1];
    dataPack[write_counter++] = point_pm10[2];
    dataPack[write_counter++] = point_pm10[3];
    // temperature
    byte* point_tempOPC = (byte*)&tempOPC;
    dataPack[write_counter++] = point_tempOPC[0];
    dataPack[write_counter++] = point_tempOPC[1];
    dataPack[write_counter++] = point_tempOPC[2];
    dataPack[write_counter++] = point_tempOPC[3];
    // humidity
    byte* point_hum = (byte*)&humidity;
    dataPack[write_counter++] = point_hum[0];
    dataPack[write_counter++] = point_hum[1];
    dataPack[write_counter++] = point_hum[2];
    dataPack[write_counter++] = point_hum[3];
    // Stop byte
    dataPack[write_counter] = stopByte;
}
// TMP
TMP.wakeup();
temperature = TMP.readTempC();

// MSS611
mstemp = baro.readTemperature();
pressure = baro.readPressure() * 0.00000986923;

// Voltage
voltage = analogRead(VOLTAGE);

// TVOC and eCO2
if (ccs.available()) {
    if (!ccs.readData()) {
        tvoc = ccs.getTVOC();
        eco2 = ccs.geteCO2();
    }
}

// Save packet to SD card
dataLog.write(dataPack, sizeof(dataPack));

// write breakout data to json
char breakbuf[] = ""
StaticJsonDocument<400> breakdoc;

// dtostrf(temperature, 4, 3, breakbuf);
// breakdoc["TMP Temp"] = breakbuf;
dtostrf(mstemp, 4, 3, breakbuf);
breakdoc["Baro Temp"] = breakbuf;
dtostrf(pressure, 4, 3, breakbuf);
breakdoc["Pressure"] = breakbuf;
dtostrf(tvoc, 4, 3, breakbuf);
breakdoc["TVOC"] = breakbuf;
dtostrf(eco2, 4, 3, breakbuf);
breakdoc["eCO2"] = breakbuf;

dtostrf(voltage, 4, 3, breakbuf);
breakdoc["Volt"] = breakbuf;
serializeJson(breakdoc, breakJson);

if (debug) {
    serializeJsonPretty(breakdoc, Serial); Serial.println("");}

// Particle Counters
if (opcTimer > OPC_INTERVAL) {
    // Read OPC
    opcTimer = 0;
hist = myOPCN3.readHistogramData();

    // Get Temperature
tempOPC = hist.getTempC();

    // Get Humidity
    humidity = hist.getHumidity();

    // Get particle densities
    opcPM1 = hist.pm1;
Data

```c
opcPM2_5 = hist.pm2_5;
opcPM10 = hist.pm10;
// Serial.println(opcPM10);

// Write OPC Data to JSON
char buf[] = "";
StaticJsonDocument<400> OPCdoc;
dtostrf(tempOPC, 4, 3, buf);
OPCdoc["Temperature"] = buf;
dtostrf(humidity, 4, 3, buf);
OPCdoc["Humidity"] = buf;
dtostrf(opcPM1, 4, 3, buf);
OPCdoc["PM1"] = buf;
dtostrf(opcPM2_5, 4, 3, buf);
OPCdoc["PM2_5"] = buf;
dtostrf(opcPM10, 4, 3, buf);
OPCdoc["PM10"] = buf;
// Serial.println(measureJson(doc));
serializeJson(OPCdoc, opcJson); Serial.println("");

if ( debug ) {
    serializeJsonPretty(OPCdoc, Serial); Serial.println("");  
}

// Read SPS30
// do {
//    ret = sps30.GetValues(&sps);
//    delay(500);
//    } while (ret != ERR_OK);
//    spsPM1 = sps.MassPM1;
//    spsPM2_5 = sps.MassPM2;
//    spsPM4 = sps.MassPM4;
//    spsPM10 = sps.MassPM10;
//    Serial.println(spsPM10);
//    // Write SPS Data to JSON
//    char sps_buf[] = "";
//    StaticJsonDocument<400> SPSdoc;
//    dtostrf(spsPM1, 4, 3, sps_buf);
//    SPSdoc["PM1"] = sps_buf;
//    dtostrf(spsPM2_5, 4, 3, sps_buf);
//    SPSdoc["PM2_5"] = sps_buf;
//   (dtostrf(spsPM4, 4, 3, sps_buf);
//    SPSdoc["PM4"] = sps_buf;
//    dtostrf(spsPM10, 4, 3, sps_buf);
//    SPSdoc["PM10"] = sps_buf;
//    serializeJson(SPSdoc, spsJson); Serial.println("");
  }

void publishData() {
  // Publish all feeds
  tmp.publish(temperature);
  baroTemp.publish(mstemp);
  baroPres.publish(pressure);
```

C.1. AQM SOFTWARE
C.1. AQM SOFTWARE

```c
250 //
251 Breakouts.publish(breakJson);
252 OpC.publish(opcJson);
253 // SpS.publish(spsJson);
254 ///// OPChum.publish(humidity);
255 ///// OPCTemp.publish(tempOPC);
256 ///// opcPm1.publish(opcPM1);
257 ///// opcPm2_5.publish(opcPM2_5);
258 ///// opcPm10.publish(opcPM10);
259 //
260 ///// spspm1.publish(spsPM1);
261 ///// spspm2_5.publish(spsPM2_5);
262 ///// spspm4.publish(spsPM4);
263 ///// spspm10.publish(spsPM10);
264 //
265 ///// sourceVolt.publish((float)voltage);
266 // TVOC.publish((float) tvoc);
267 // eCO2.publish((float) eCO2);
268 // GPSLoc.publish(gpsJson);
269 Serial.println("Published");
270 }
```
C.1. AQM SOFTWARE

4/13/22, 7:08 PM

Definition

Julio Guardado

1 /////////////////////////////////////////////////////////////////////////////////////////////////
2 ///
3 ///************ MAIN PROGRAM DEFINITIONS
4 //////////////////////////////////////////////////////////////////////////////////////////////////
5 ///
6 //************************** Data Timers **********************************************
7 elapsedMillis updateTimer;
8 #define UPDATE_INTERVAL 1000
9 elapsedMillis sdTimer;
10 #define SD_INTERVAL 50
11 elapsedMillis breakoutTimer;
12 #define BREAKOUT_INTERVAL 1000
13 char breakJson[255];
14 //************************** Data **********************************************
15 #define PACKET_SIZE 108
16 byte id_erau[2] = {0xA0, 0xB1}; //Identifier for ERAU packet.
17 byte stopByte = 0x27;
18 byte dataPack[PACKET_SIZE];
19 unsigned int packNum = 0; //Packet number/counter.
20 int write_counter;
21 char dataJson[400];
22 // SD Card
23 File dataLog;
24 String fileN;
25 String fileName1;
26 String ext = ".bin";
27 unsigned int address;
28 unsigned int fileNum;
29 const int chipSelect = BUILTIN_SDCARD;
30 const int appendPos = 0; // File position marker - used for saving and re-opening SD file
31 unsigned int timeSD; // Time stamp - used for determining when to save SD card
32 #define OPC_INTERVAL 1000
33 elapsedMillis opcTimer;
34 //********************SENSORS******************
35 TMP102 TMP(0x48); //digital temperature sensor
36 UbloxGPS gps(&Serial1); //GPS
37 //SPS30 sps30;
38 OPCN3 myOPCN3(27);
39 HistogramData hist;
40 unsigned long OPCtime;
41 float tempOPC;
42 float humidity;
43 float opcPM1;
44 float opcPM2_5;
45 float opcPM10;
46 #define OPC_INTERVAL 1000
47 elapsedMillis opcTimer;
char opcJson[255];

/***********SPS30 SETUP*************/
//struct sps_values sps;
uint16_t data_ready;
uint8_t ret;

float spsPM1;
float spsPM2_5;
float spsPM4;
float spsPM10;

char spsJson[255];

/********************TMP Variables*******************/
#define TMP_UPPER_LIM 4    //Limit where TMP alert is turned off
#define TMP_LOWER_LIM 0     //Limit where TMP alert is turned on
#define PAD_ENABLE 16        //Heating Pad enable pin
float temperature;
boolean alertPinState, alertRegisterState;

#define TMP_INTERVAL 50
elapsedMillis tmpTimer;

/***********MS5611 SETUP*************/
MS5611 baro;
float pressure;
float baroReferencePressure;
float mstemp;       //temperature from MS5611

#define BARO_INTERVAL 50
elapsedMillis baroTimer;

/********************TVOC and eCO2**********************/
Adafruit_CCS811 ccs;
uint16_t tvoc;
uint16_t eco2;

/********************GPS Variables**********************/
#define GPS_BAUD 9600
// GPS Data
float alt = 0;
float highAlt = 0;
float latitude = 0;
float longitude = 0;
uint8_t utcSec;
uint8_t utcMin;
uint8_t utcHour;
uint8_t numSats;
char gpsJson[255];

/********************UTILITY**********************/
// Timing variable
#define UPDATE_INTERVAL 1000
unsigned long timer;

//battery monitor
ADC *adc = new ADC();
#define VOLTAGE A6
int16_t voltage;

// Valve status
int valveStatus = 0;
C.1. AQM SOFTWARE

/* **************** WIFI ****************/ 
char ssid[] = "Thicc Boiiiiz 2.4"; // your network SSID (name) 
char pass[] = "honey0124"; // your network password 
//char ssid[] = "EagleNet"; // your network SSID (name) 
//char pass[] = " "; // your network password 
int status = WL_IDLE_STATUS; // the wifi radio's status

#define IO_SERVER      "192.168.1.213" 
#define IO_SERVERPORT 1883 // use 8883 for SSL

// Initialize the Ethernet client object 
WiFiEspClient client; 
Adafruit_MQTT_Client mqtt(&client, IO_SERVER, IO_SERVERPORT,"mqtt","SAIL123");

/**************************** Feeds ***************************************/ 
Adafruit_MQTT_Publish tmp = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/TMP102 Temperature"); 
// 
//Adafruit_MQTT_Publish baroTemp = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/MS5611 Pressure"); 
Adafruit_MQTT_Publish Breakouts = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/Breakouts"); 
Adafruit_MQTT_Publish OpC = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/OPC3"); 
Adafruit_MQTT_Publish SpS = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/SPS30"); 
//Adafruit_MQTT_Publish OPChum = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/OPC Humidity"); 
//Adafruit_MQTT_Publish TVOC = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/TVOC"); 
//Adafruit_MQTT_Publish eCO2 = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/eCO2"); 
//Adafruit_MQTT_Publish sourceVolt = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/Source Voltage"); 
//Adafruit_MQTT_Publish GPSLoc = Adafruit_MQTT_Publish(&mqtt, "Sensor_Data/AQM 1/GPS Location");
Functions

Julio Guardado

```c
void updateGPS() {
    gps.update();
    if (millis() - timer > UPDATE_INTERVAL) {
        timer = millis();
        // get measurements
        alt = gps.getAlt_meters();
        latitude = gps.getLat();
        longitude = gps.getLon();
        numSats = gps.getSats();
        utcSec = gps.getSecond();
        utcMin = gps.getMinute();
        utcHour = gps.getHour();
        // Write GPS Data
        char buf[] = "";
        StaticJsonDocument<400> doc;
        dtostrf(alt, 4, 3, buf);
        doc["Alt"] = buf;
        dtostrf(latitude, 4, 3, buf);
        doc["Lat"] = buf;
        dtostrf(longitude, 4, 3, buf);
        doc["Lon"] = buf;
        dtostrf(utcSec, 4, 3, buf);
        doc["Sec"] = buf;
        dtostrf(utcMin, 4, 3, buf);
        doc["Min"] = buf;
        dtostrf(utcHour, 4, 3, buf);
        doc["Hr"] = buf;
        // Serial.println(measureJson(doc));
        serializeJson(doc, gpsJson); Serial.println("");
        if ( debug ){
            serializeJsonPretty(doc, Serial); Serial.println("");
        }
    }
}
```

void array_to_string(byte array[], unsigned int len, char buffer[])
{
}
functions

```c
for (unsigned int i = 0; i < len; i++)
{
    byte nib1 = (array[i] >> 4) & 0x0F;
    byte nib2 = (array[i] >> 0) & 0x0F;
    buffer[i * 2 + 0] = nib1 < 0xA ? '0' + nib1 : 'A' + nib1 - 0xA;
    buffer[i * 2 + 1] = nib2 < 0xA ? '0' + nib2 : 'A' + nib2 - 0xA;
}
buffer[len * 2] = '\0';
```

// Function to connect and reconnect as necessary to the MQTT server.
// Should be called in the loop function and it will take care if connecting.
void MQTT_connect() {
    int8_t ret;
    // Stop if already connected.
    if (mqtt.connected()) {
        return;
    }
    Serial.print("Connecting to MQTT...");
    uint8_t retries = 3;
    while ((ret = mqtt.connect()) == 0) { // connect will return 0 for connected
        Serial.println(mqtt.connectErrorString(ret));
        Serial.println("Retrying MQTT connection in 5 seconds...");
        mqtt.disconnect();
        delay(5000); // wait 5 seconds
        retries--;
        if (retries == 0) {
            // basically die and wait for WDT to reset me
            while (1);
        }
    }
    Serial.println("MQTT Connected!");
}


/// /////////////////************ MAIN PROGRAM SENSOR INITIALIZATION
**************///

//////////////////////////////////////////////////////////////////////////////////////////////
///
//////////////////////////////////////////////////////////////////////////////////////////////

///
//////////////////////////////////////////////////////////////////////////////////////////////
///
//////////////////////////////////////////////////////////////////////////////////////////////
///

/************************************************************************** Initiate GPS **********/

void initGPS() {
  Serial.println("/
  Serial.println("/************** GPS INIT **************/");

  while(Serial1.available()) Serial1.read();

  gps.init();
  bool gpsconfig = gps.setAirborne();
  while (gpsconfig == false) {
    Serial.println("GPS initializing...");
    gpsconfig = gps.setAirborne();
    //    GPSSerial.listen();
    delay(5);
  }
  delay(50);
  if (debug) {
    if (gpsconfig) {
      Serial.println("Airborne mode set...");
      Serial.println("GPS configured");
    }
    else {
      Serial.println("Airborne Mode Not Set");
    }
  }
  Serial.println("GPS OK");
}

/************************************************************************** Initiate TMP **********/

void initTMP() {
  Serial.println("/
  Serial.println("/************** TMP102 INIT **************/");

  PINMode(ALERT_PIN,INPUT);
  TMP.begin();

  // Set the number of consecutive faults before activate pin.
  // 0-3: 0:1 fault, 1:2 faults, 2:4 faults, 3:6 faults.
  TMP.setFault(2);

  // Set the polarity of the Alarm. (0:Active LOW, 1:Active HIGH).
  TMP.setAlertPolarity(0); // Active Low

  // Set the sensor in Comparator Mode (0) or Interrupt Mode (1).
  TMP.setAlertMode(0); // Comparator Mode.

  // Set the Conversion Rate (how quickly the sensor gets a new reading)
  // 0-3: 0:0.25Hz, 1:1Hz, 2:4Hz, 3:8Hz
  TMP.setConversionRate(1);

  // Set Mode.
  // 0:12-bit Temperature(-55C to +128C) 1:13-bit Temperature(-55C to +150C)
  TMP.setExtendedMode(0);
}
4/13/22, 7:08 PM Sensor Init

55 // Set the upper limit to turn off the alert
56 TMP.setHighTempC(TMP_UPPER_LIM); // set T_HIGH in C
57
58 // Set the lower limit to turn on the alert
59 TMP.setLowTempC(TMP_LOWER_LIM); // set T_LOW in C
60
61 Serial.println("TMP102 OK");
62}
63
64 // Pressure sensor setup
65 void initBaro() {
66 Serial.println("-- BAROMETER INIT --");
67 Serial.println("BAROMETER OK");
68 }
69
70 // Pressure sensor setup
71 void initSPS() {
72 Serial.println("-- SPS30 INIT --");
73 Serial.println("SPS30 OK");
74 }
75
76 // Pressure sensor setup
77 void cssInit()
78 }
79
80 // Pressure sensor setup
81 void opcInit()
82 }
83
84 // Pressure sensor setup
85 void cssInit()
86 }
87
88 // Pressure sensor setup
89 void opcInit()
90 }
91
92 // Pressure sensor setup
93 void cssInit()
94 }
95
96 // Pressure sensor setup
97 void opcInit()
98 }
99
100 // Pressure sensor setup
101 void cssInit()
102 }
103
104 // Pressure sensor setup
105 void opcInit()
106 }
107
108 // Pressure sensor setup
109 void cssInit()
110 }
111
112 // Pressure sensor setup
113 void opcInit()
C.2 Logging Script
# %% Import required libraries
import paho.mqtt.client as mqtt
from datetime import datetime
import time
import csv
import json
import os
import pandas as pd
import glob
# %% Subscribe to messages
def on_connect(client, userdata, flags, rc):
    # Sensors
    client.subscribe("Sensor_Data/AQM 1/Breakouts")
    client.subscribe("Sensor_Data/AQM 1/GPS Location")
    # Particle Counters
    client.subscribe("Sensor_Data/AQM 1/OPCN3")
    client.subscribe("Sensor_Data/AQM 1/SPS30")
# %% Message Callbacks
def breakout_message(client, userdata, message):
    breakoutData = json.loads(str(message.payload.decode("utf-8")))
    breakoutData = {
        'Timestamp': datetime.now().strftime("%H:%M:%S.%f")
    }
    with open('Sensors.csv', 'a') as breakout_file:
        breakout_writer = csv.DictWriter(breakout_file, fieldnames=breakoutNames)
        if userdata[0] == 0:
            breakout_writer.writerow(breakoutData)
            userdata[0] += 1
        breakout_writer.writerow(breakoutData)

def opc_message(client, userdata, message):
    opcData = json.loads(str(message.payload.decode("utf-8")))
    opcData = {
        'Timestamp': datetime.now().strftime("%H:%M:%S.%f")
    }
    with open('OPCN3.csv', 'a') as opc_file:
        opc_writer = csv.DictWriter(opc_file, fieldnames=opcNames)
        if userdata[1] == 0:
            opc_writer.writerow(opcData)
            userdata[1] += 1
        opc_writer.writerow(opcData)

def sps_message(client, userdata, message):
    spsData = json.loads(str(message.payload.decode("utf-8")))
    spsData = {
        'Timestamp': datetime.now().strftime("%H:%M:%S.%f")
    }
    with open('SPS30.csv', 'a') as sps_file:
        sps_writer = csv.DictWriter(sps_file, fieldnames=spsNames)
        if userdata[2] == 0:
            sps_writer.writerow(spsData)
            userdata[2] += 1
        sps_writer.writerow(spsData)

def gps_message(client, userdata, message):
    gpsData = json.loads(str(message.payload.decode("utf-8")))
    gpsData = {
        'Timestamp': datetime.now().strftime("%H:%M:%S.%f")
    }
    with open('GPS.csv', 'a') as gps_file:
        gps_writer = csv.DictWriter(gps_file, fieldnames=gpsNames)
        if userdata[3] == 0:
            gps_writer.writerow(gpsData)
            userdata[3] += 1
        gps_writer.writerow(gpsData)
with open('GPS.csv', 'a') as gps_file:
gps_writer = csv.DictWriter(gps_file, fieldnames=gpsNames)
if userdata[3] == 0:
gps_writer.writerow()
 userdata[3] += 1
gps_writer.writerow(gpsData)

# % % Set up CSV file Headers for data writing
countSensors = 0
countOPC = 0
countSPS = 0
countGPS = 0

fileOpenTime = datetime.now().strftime('%H-%M-%S')
breakoutNames = ['Timestamp', 'TMP Temp', 'Baro Temp', 'Pressure', 'TVOC', 'eCO2', 'Volt']
opcNames = ['Timestamp', 'Temperature', 'Humidity', 'PM1', 'PM2.5', 'PM10']
spsNames = ['Timestamp', 'Alf', 'Lat', 'Lon', 'Hr', 'Min', 'Sec']
gpsNames = ['Timestamp', 'Alt', 'Lat', 'Lon', 'Sec']

# % % Set up client and client callbacks
client = mqtt.Client(userdata=[countSensors, countOPC, countSPS, countGPS]) #create new instance
client.username_pw_set(username='mqtt', password='SAIL123')
client.on_connect = on_connect

# Attach callbacks for each message type
client.message_callback_add("Sensor_Data/AQM 1/OPCN3", opc_message)
client.message_callback_add("Sensor_Data/AQM 1/SPS30", sps_message)
client.message_callback_add("Sensor_Data/AQM 1/Breakouts", breakout_message)
client.message_callback_add("Sensor_Data/AQM 1/GPS Location", gps_message)

# % % Connect to broker and start loop
broker_address = "192.168.1.213"
client.connect(broker_address) #connect to broker
client.loop_start() #start the loop

print("Subscribing to topic", "Sensor_Data/AQM 1")

try:
    while True:
        time.sleep(1)
        pass
except KeyboardInterrupt:
    print("interrupted by keyboard")

client.loop_stop() #stop the loop

# % % Merge all files into one excel file with multiple files
path = os.getcwd()
all_files = glob.glob(os.path.join(path, "*.csv"))
writer = pd.ExcelWriter(datetime.now().strftime("%d-%m-%Y") + 1 + fileOpenTime + ' Air Quality Data.xlsx', engine='xlsxwriter')

for f in all_files:
df = pd.read_csv(f)
df.to_excel(writer, sheet_name=os.path.basename(f), index=False)

# Auto-adjust columns width
for column in df:
column_width = max(df[column].astype(str).map(len).max(), len(column))
col_idx = df.columns.get_loc(column)
writer.sheets[os.path.basename(f)].set_column(col_idx, col_idx, column_width)

# Delete all unnecessary files
C.2. LOGGING SCRIPT

```python
os.remove(f)
writer.save()
```
C.3 Mosquitto Configuration
Mosquitto Config

Julio Guardado

1 # Config file for mosquitto
2 #
3 # See mosquitto.conf(5) for more information.
4 #
5 # Default values are shown, uncomment to change.
6 #
7 # Use the # character to indicate a comment, but only if it is the
8 # very first character on the line.
9 #
10 #=================================================================
11 # General configuration
12 #=================================================================
13 #
14 # Use per listener security settings.
15 #
16 # It is recommended this option be set before any other options.
17 #
18 # If this option is set to true, then all authentication and access control
19 # options are controlled on a per listener basis. The following options are
20 # affected:
21 #
22 # acl_file
23 # allow_anonymous
24 # allow_zero_length_clientid
25 # auto_id_prefix
26 # password_file
27 # plugin
28 # plugin_opt_*
29 # psk_file
30 #
31 # Note that if set to true, then a durable client (i.e. with clean session set
32 # to false) that has disconnected will use the ACL settings defined for the
33 # listener that it was most recently connected to.
34 #
35 # The default behaviour is for this to be set to false, which maintains the
36 # setting behaviour from previous versions of mosquitto.
37 #per_listener_settings false
38 #
39 # This option controls whether a client is allowed to connect with a zero
40 # length client id or not. This option only affects clients using MQTT v3.1.1
41 # and later. If set to false, clients connecting with a zero length client id
42 # are disconnected. If set to true, clients will be allocated a client id by
43 # the broker. This means it is only useful for clients with clean session set
44 # to true.
45 #allow_zero_length_clientid true
46 #
47 # If allow_zero_length_clientid is true, this option allows you to set a prefix
48 # to automatically generated client ids to aid visibility in logs.
49 #Defaults to 'auto-'
50 #auto_id_prefix auto-
51 #
52 # This option affects the scenario when a client subscribes to a topic that has
53 # retained messages. It is possible that the client that published the retained
54 # message to the topic had access at the time they published, but that access
55 # has been subsequently removed. If check_retain_source is set to true, the
56 # default, the source of a retained message will be checked for access rights
# before it is republished. When set to false, no check will be made and the
# retained message will always be published. This affects all listeners.
check_retain_source true

# QoS 1 and 2 messages will be allowed inflight per client until this limit
# is exceeded. Defaults to 0. (No maximum)
max_inflight_messages

# The maximum number of QoS 1 and 2 messages currently inflight per
# client.
max_inflight_messages 20

# For MQTT v5 clients, it is possible to have the server send a "server
# keepalive" value that will override the keepalive value set by the client.
# This is intended to be used as a mechanism to say that the server will
# disconnect the client earlier than it anticipated, and that the client should
# use the new keepalive value. The max_keepalive option allows you to specify
# that clients may only connect with keepalive less than or equal to this
# value, otherwise they will be sent a server keepalive telling them to use
# max_keepalive. This only applies to MQTT v5 clients. The default, and maximum
# value allowable, is 65535.
max_keepalive 65535

# Set to 0 to allow clients to set keepalive = 0, which means no keepalive
# checks are made and the client will never be disconnected by the broker if no
# messages are received. You should be very sure this is the behaviour that you
# want.
max_keepalive 0

# For MQTT v3.1.1 and v3.1 clients, there is no mechanism to tell the client
# what keepalive value they should use. If an MQTT v3.1.1 or v3.1 client
# specifies a keepalive time greater than max_keepalive they will be sent a
# CONNACK message with the "identifier rejected" reason code, and disconnected.
max_keepalive 65535

# For MQTT v5 clients, it is possible to have the server send a "maximum packet
# size" value that will instruct the client it will not accept MQTT packets
# with size greater than max_packet_size bytes. This applies to the full MQTT
# packet, not just the payload. Setting this option to a positive value will
# set the maximum packet size to that number of bytes. If a client sends a
# packet which is larger than this value, it will be disconnected. This applies
# to all clients regardless of the protocol version they are using, but v3.1.1
# and earlier clients will of course not have received the maximum packet size
# information. Defaults to no limit. Setting below 20 bytes is forbidden
# because it is likely to interfere with ordinary client operation, even with
# very small payloads.
max_packet_size 0

# QoS 1 and 2 messages above those currently in-flight will be queued per
# client until this limit is exceeded. Defaults to 0. (No maximum)
max_queued_messages

# If both max_queued_messages and max_queued_bytes are specified, packets will
# be queued until the first limit is reached.
max_queued_messages

# Set the maximum QoS supported. Clients publishing at a QoS higher than
# specified here will be disconnected.
max_qos 2

# The maximum number of QoS 1 and 2 messages to hold in a queue per client
# above those that are currently in-flight. Defaults to 1000. Set
C.3. MOSQUITTO CONFIGURATION

4/13/22, 7:14 PM Mosquitto Config

123 # to 0 for no maximum (not recommended).
124 # See also queue_qos0_messages.
125 # See also max_queued_bytes.
126 #max_queued_messages 1000
127 #
128 # This option sets the maximum number of heap memory bytes that the broker will
129 # allocate, and hence sets a hard limit on memory use by the broker. Memory
130 # requests that exceed this value will be denied. The effect will vary
131 # depending on what has been denied. If an incoming message is being processed,
132 # then the message will be dropped and the publishing client will be
133 # disconnected. If an outgoing message is being sent, then the individual
134 # message will be dropped and the receiving client will be disconnected.
135 # Defaults to no limit.
136 #memory_limit 0
137 #
138 # This option sets the maximum publish payload size that the broker will allow.
139 # Received messages that exceed this size will not be accepted by the broker.
140 # The default value is 0, which means that all valid MQTT messages are
141 # accepted. MQTT imposes a maximum payload size of 268435455 bytes.
142 #message_size_limit 0
143 #
144 # This option allows persistent clients (those with clean session set to false)
145 # to be removed if they do not reconnect within a certain time frame.
146 #
147 # This is a non-standard option in MQTT V3.1 but allowed in MQTT v3.1.1.
148 #
149 # Badly designed clients may set clean session to false whilst using a randomly
150 # generated client id. This leads to persistent clients that will never
151 # reconnect. This option allows these clients to be removed.
152 #
153 # The expiration period should be an integer followed by one of h d w m y for
154 # hour, day, week, month and year respectively. For example
155 #
156 # persistent_client_expiration 2m
157 # persistent_client_expiration 14d
158 # persistent_client_expiration 1y
159 #
160 # The default if not set is to never expire persistent clients.
161 #persistent_client_expiration
162 #
163 # Write process id to a file. Default is a blank string which means
164 # a pid file shouldn’t be written.
165 # This should be set to /var/run/mosquitto/mosquitto.pid if mosquitto is
166 # being run automatically on boot with an init script and
167 # start-stop-daemon or similar.
168 #pid_file
169 #
170 # Set to true to queue messages with QoS 0 when a persistent client is
171 # disconnected. These messages are included in the limit imposed by
172 # max_queued_messages and max_queued_bytes
173 # Defaults to false.
174 # This is a non-standard option for the MQTT v3.1 spec but is allowed in
175 # v3.1.1.
176 #queue_qos0_messages false
177 #
178 # Set to false to disable retained message support. If a client publishes a
179 # message with the retain bit set, it will be disconnected if this is set to
180 # false.
181 #retain_available true
182 #
183 # Disable Nagle’s algorithm on client sockets. This has the effect of reducing
184 # latency of individual messages at the potential cost of increasing the number
185 # of packets being sent.
186 #set_tcp_nodelay false
188 # Time in seconds between updates of the $SYS tree.
189 # Set to 0 to disable the publishing of the $SYS tree.
190 #sys_interval 10
191
192 # The MQTT specification requires that the QoS of a message delivered to a
193 # subscriber is never upgraded to match the QoS of the subscription. Enabling
194 # this option changes this behaviour. If upgrade_outgoing_qos is set true,
195 # messages sent to a subscriber will always match the QoS of its subscription.
196 # This is a non-standard option explicitly disallowed by the spec.
197 #upgrade_outgoing_qos false
198
199 # When run as root, drop privileges to this user and its primary
200 # group.
201 # Set to root to stay as root, but this is not recommended.
202 # If set to "mosquitto", or left unset, and the "mosquitto" user does not exist
203 # then it will drop privileges to the "nobody" user instead.
204 # If run as a non-root user, this setting has no effect.
205 # Note that on Windows this has no effect and so mosquitto should be started by
206 # the user you wish it to run as.
207 #user mosquitto
208
209 # =================================================================
210 # Listeners
211 # =================================================================
212
213 # Listen on a port/ip address combination. By using this variable
214 # multiple times, mosquitto can listen on more than one port. If
215 # this variable is used and neither bind_address nor port given,
216 # then the default listener will not be started.
217 # The port number to listen on must be given. Optionally, an ip
218 # address or host name may be supplied as a second argument. In
219 # this case, mosquitto will attempt to bind the listener to that
220 # address and so restrict access to the associated network and
221 # interface. By default, mosquitto will listen on all interfaces.
222 # Note that for a websockets listener it is not possible to bind to a host
223 # name.
224 #
225 # On systems that support Unix Domain Sockets, it is also possible
226 # to create a # Unix socket rather than opening a TCP socket. In
227 # this case, the port number should be set to 0 and a unix socket
228 # path must be provided, e.g.
229 # listener 0 /tmp/mosquitto.sock
230 #
231 # listener port-number [ip address/host name/unix socket path]
232 listener 1883
233 listener 36968
234
235 # By default, a listener will attempt to listen on all supported IP protocol
236 # versions. If you do not have an IPv4 or IPv6 interface you may wish to
237 # disable support for either of those protocol versions. In particular, note
238 # that due to the limitations of the websockets library, it will only ever
239 # attempt to open IPv6 sockets if IPv6 support is compiled in, and so will fail
240 # if IPv6 is not available.
241 #
242 # Set to `ipv4` to force the listener to only use IPv4, or set to `ipv6` to
243 # force the listener to only use IPv6. If you want support for both IPv4 and
244 # IPv6, then do not use the socket_domain option.
245 #
246 #socket_domain
247
248 # Bind the listener to a specific interface. This is similar to
249 # the [ip address/host name] part of the listener definition, but is useful
250 # when an interface has multiple addresses or the address may change. If used
251 # with the [ip address/host name] part of the listener definition, then the
252 # bind_interface option will take priority.
C.3. MOSQUITTO CONFIGURATION

# Not available on Windows.
# Example: bind_interface eth0
#bind_interface

# When a listener is using the websockets protocol, it is possible to serve http data as well. Set http_dir to a directory which contains the files you wish to serve. If this option is not specified, then no normal http connections will be possible.
#http_dir

# The maximum number of client connections to allow. This is a per listener setting.
# Default is -1, which means unlimited connections.
# Note that other process limits mean that unlimited connections are not really possible. Typically the default maximum number of connections possible is around 1024.
#max_connections -1

# The listener can be restricted to operating within a topic hierarchy using the mount_point option. This is achieved by prefixing the mount_point string to all topics for any clients connected to this listener. This prefixing only happens internally to the broker; the client will not see the prefix.
#mount_point

# Choose the protocol to use when listening.
# This can be either mqtt or websockets.
# Certificate based TLS may be used with websockets, except that only the cafile, certfile, keyfile, ciphers, and ciphers_tls13 options are supported.
#protocol mqtt

# Set use_username_as_clientid to true to replace the clientid that a client connected with with its username. This allows authentication to be tied to the clientid, which means that it is possible to prevent one client disconnecting another by using the same clientid.
# If a client connects with no username it will be disconnected as not authorised when this option is set to true.
# Do not use in conjunction with clientid_prefixes.
# See also use_identity_as_username.
#use_username_as_clientid

# Change the websockets headers size. This is a global option, it is not possible to set per listener. This option sets the size of the buffer used in the libwebsocket library when reading HTTP headers. If you are passing large header data such as cookies then you may need to increase this value. If left unset, or set to 0, then the default of 1024 bytes will be used.
#websockets_headers_size

#-----------------------------------------------
# Certificate based SSL/TLS support

# -----------------------------------------------------------
# The following options can be used to enable certificate based SSL/TLS support for this listener. Note that the recommended port for MQTT over TLS is 8883, but this must be set manually.
# See also the mosquitto-tls man page and the "Pre-shared-key based SSL/TLS support" section. Only one of certificate or PSK encryption support can be enabled for any listener.
# Both of certfile and keyfile must be defined to enable certificate based TLS encryption.
# Path to the PEM encoded server certificate.
#certfile
# Path to the PEM encoded keyfile.
#keyfile
# If you wish to control which encryption ciphers are used, use the ciphers option. The list of available ciphers can be obtained using the "openssl ciphers" command and should be provided in the same format as the output of that command. This applies to TLS 1.2 and earlier versions only. Use ciphers_tls1.3 for TLS v1.3.
#ciphers
# Choose which TLS v1.3 ciphersuites are used for this listener.
# Defaults to "TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256"
#ciphers_tls1.3
# If you have require_certificate set to true, you can create a certificate revocation list file to revoke access to particular client certificates. If you have done this, use crlfile to point to the PEM encoded revocation file.
#crlfile
# To allow the use of ephemeral DH key exchange, which provides forward security, the listener must load DH parameters. This can be specified with the dhparamfile option. The dhparamfile can be generated with the command e.g. "openssl dhparam -out dhparam.pem 2048"
#dhparamfile
# By default an TLS enabled listener will operate in a similar fashion to a https enabled web server, in that the server has a certificate signed by a CA and the client will verify that it is a trusted certificate. The overall aim is encryption of the network traffic. By setting require_certificate to true, the client must provide a valid certificate in order for the network connection to proceed. This allows access to the broker to be controlled outside of the mechanisms provided by MQTT.
#require_certificate false
# If require_certificate is true, you may set use_identity_as_username to true
# to use the CN value from the client certificate as a username. If this is true, the password_file option will not be used for this listener.
#use_identity_as_username false
# -----------------------------------------------------------
C.3. MOSQUITTO CONFIGURATION

370  # Pre-shared-key based SSL/TLS support
371  # -----------------------------------------------------------
The following options can be used to enable PSK based SSL/TLS support for this listener. Note that the recommended port for MQTT over TLS is 8883, but this must be set manually.

# See also the mosquitto-tls man page and the "Certificate based SSL/TLS support" section. Only one of certificate or PSK encryption support can be enabled for any listener.

The psk_hint option enables pre-shared-key support for this listener and also acts as an identifier for this listener. The hint is sent to clients and may be used locally to aid authentication. The hint is a free form string that doesn't have much meaning in itself, so feel free to be creative.

If this option is provided, see psk_file to define the pre-shared keys to be used or create a security plugin to handle them.

When using PSK, the encryption ciphers used will be chosen from the list of available PSK ciphers. If you want to control which ciphers are available, use the "ciphers" option. The list of available ciphers can be obtained using the "openssl ciphers" command and should be provided in the same format as the output of that command.

# Use_identity_as_username to have the psk identity sent by the client used as its username. Authentication will be carried out using the PSK rather than the MQTT username/password and so password_file will not be used for this listener.

# Persistence

If persistence is enabled, save the in-memory database to disk every autosave_interval seconds. If set to 0, the persistence database will only be written when mosquitto exits. See also autosave_on_changes.

Note that writing of the persistence database can be forced by sending mosquitto a SIGUSR1 signal.

# Autosave_interval 1800

# If true, mosquitto will count the number of subscription changes, retained messages received and queued messages and if the total exceeds autosave_interval then the in-memory database will be saved to disk.

# Autosave_interval as a time in seconds.

# Save persistent message data to disk (true/false).

# This saves information about all messages, including subscriptions, currently in-flight messages and retained messages.

# Retained persistence is a synonym for this option.

# The filename to use for the persistent database, not including the path.

# Location for persistent database.

Default is an empty string (current directory).

# Set to e.g. /var/lib/mosquitto if running as a proper service on Linux or similar.
#persistence_location

# Logging

# Places to log to. Use multiple log_dest lines for multiple
# logging destinations.
# Possible destinations are: stdout stderr syslog topic file dlt
# stdout and stderr log to the console on the named output.
# syslog uses the userspace syslog facility which usually ends up
# in /var/log/messages or similar.
# in /var/log/messages or similar.
# log_dest stderr

# Types of messages to log. Use multiple log_type lines for logging
# multiple types of messages.
# Possible types are: debug, error, warning, notice, information,
# none, subscribe, unsubscribe, websockets, all.
# Note that debug type messages are for decoding the incoming/outgoing
# network packets. They are not logged in "topics".
#log_type error
#log_type warning
#log_type notice
#log_type information

# If set to true, client connection and disconnection messages will be included
# in the log.
#connection_messages true

# If using syslog logging (not on Windows), messages will be logged to the
# "daemon" facility by default. Use the log_facility option to choose which of
# local0 to local7 to log to instead. The option value should be an integer
# value, e.g. "log_facility 5" to use local5.
#log_facility

# If set to true, add a timestamp value to each log message.
#log_timestamp true

# Set the format of the log timestamp. If left unset, this is the number of
# seconds since the Unix epoch.
# This is a free text string which will be passed to the strftime function. To
# get an ISO #601 datetime, for example:
#log_timestamp_format %Y-%m-%d%H:%M:%S
#log_timestamp_format

# Change the websockets logging level. This is a global option, it is not possible to set per listener. This is an integer that is interpreted by libwebsocket as a bit mask for its lws_log_levels enum. See the libwebsocket documentation for more details. "log_type websockets" must also be enabled.

#websockets_log_level 0

# Security

# If set, only clients that have a matching prefix on their clientid will be allowed to connect to the broker. By default, all clients may connect. For example, setting "secure-" here would mean a client "secure-client" could connect but another with clientid "mqtt" couldn't.

#clientid_prefixes

# Boolean value that determines whether clients that connect without providing a username are allowed to connect. If set to false then a password file should be created (see the password_file option) to control authenticated client access.

allow_anonymous false

# -----------------------------------------------------------
# Default authentication and topic access control

# -----------------------------------------------------------
# Control access to the broker using a password file. This file can be
# generated using the mosquitto_passwd utility. If TLS support is not compiled
# into mosquitto (it is recommended that TLS support should be included) then
# plain text passwords are used, in which case the file should be a text file
# with lines in the format:
# username:password
# The password (and colon) may be omitted if desired, although this
# offers very little in the way of security.
#
# See the TLS client require_certificate and use_identity_as_username options
# for alternative authentication options. If a plugin is used as well as
# password_file, the plugin check will be made first.
password_file passwd

# Access may also be controlled using a pre-shared-key file. This requires
# TLS-PSK support and a listener configured to use it. The file should be text
# lines in the format:
# identity:key
# The key should be in hexadecimal format without a leading "0x".
# If an plugin is used as well, the plugin check will be made first.
#psk_file

# Control access to topics on the broker using an access control list
# file. If this parameter is defined then only the topics listed will
# have access.
# If the first character of a line of the ACL file is a # it is treated as a
# comment.
# Topic access is added with lines of the format:
# topic [read|write|readwrite|deny]
#
# The access type is controlled using "read", "write", "readwrite" or "deny".
# This parameter is optional (unless contains a space character) - if
# not given then the access is read/write. can contain the + or #
# wildcards as in subscriptions.
#
# The "deny" option can used to explicitly deny access to a topic that would
# otherwise be granted by a broader read/write/readwrite statement. Any "deny"
# topics are handled before topics that grant read/write access.
#
# The first set of topics are applied to anonymous clients, assuming
# allow_anonymous is true. User specific topic ACLs are added after a
# user line as follows:
#
# The username referred to here is the same as in password_file. It is
# not the clientid.
#
# If is also possible to define ACLs based on pattern substitution within the
# topic. The patterns available for substitution are:
#
# %c to match the client id of the client
# %u to match the username of the client
#
# The substitution pattern must be the only text for that level of hierarchy.
#
# The form is the same as for the topic keyword, but using pattern as the
# keyword.
# Pattern ACLs apply to all users even if the "user" keyword has previously
# been given.
# If using bridges with usernames and ACLs, connection messages can be allowed
# with the following pattern:
# pattern write $SYS/broker/connection/%c/state
# pattern [read|write|readwrite]
# Example:
# pattern write sensor/%u/data
# If an plugin is used as well as acl_file, the plugin check will be made first.
# acl_file
C.3. MOSQUITTO CONFIGURATION

# External authentication and topic access plugin options
# -----------------------------------------------------------
# External authentication and access control can be supported with the plugin option. This is a path to a loadable plugin. See also the plugin_opt_* options described below.

# The plugin option can be specified multiple times to load multiple plugins. The plugins will be processed in the order that they are specified here. If the plugin option is specified alongside either of password_file or acl_file then the plugin checks will be made first.

# If the per_listener_settings option is false, the plugin will be apply to all listeners. If per_listener_settings is true, then the plugin will apply to the current listener being defined only.

# This option is also available as 'auth_plugin', but this use is deprecated and will be removed in the future.

# plugin

# If the plugin option above is used, define options to pass to the plugin here as described by the plugin instructions. All options named using the format plugin_opt_* will be passed to the plugin, for example:

# This option is also available as 'auth_opt_*', but this use is deprecated and will be removed in the future.

# plugin_opt_db_host
# plugin_opt_db_port
# plugin_opt_db_username
# plugin_opt_db_password

# Bridges

# A bridge is a way of connecting multiple MQTT brokers together. Create a new bridge using the "connection" option as described below. Set options for the bridges using the remaining parameters. You must specify the address and at least one topic to subscribe to.

# Each connection must have a unique name.

# The address line may have multiple host address and ports specified. See below in the round_robin description for more details on bridge behaviour if multiple addresses are used. Note that if you use an IPV6 address, then you are required to specify a port.

# The direction that the topic will be shared can be chosen by specifying out, in or both, where the default value is out.

# The QoS level of the bridged communication can be specified with the next topic option. The default QoS level is 0, to change the QoS the topic direction must also be given.

# The local and remote prefix options allow a topic to be remapped when it is bridged to/from the remote broker. This provides the ability to place a topic in an appropriate location.

# For more details see the mosquitto.conf man page.

# Multiple topics can be specified per connection, but be careful not to create any loops.

# If you are using bridges with cleansession set to false (the default), then
you may get unexpected behaviour from incoming topics if you change what topics you are subscribing to. This is because the remote broker keeps the subscription for the old topic. If you have this problem, connect your bridge with cleansession set to true, then reconnect with cleansession set to false as normal.

#connection

#address [:] 

#topic [[[out | in | both] qos-level] local-prefix remote-prefix]

# If you need to have the bridge connect over a particular network interface, use bridge_bind_address to tell the bridge which local IP address the socket should bind to, e.g. `bridge_bind_address 192.168.1.10`

#bridge_bind_address

# If a bridge has topics that have "out" direction, the default behaviour is to send an unsubscribe request to the remote broker on that topic. This means that changing a topic direction from "in" to "out" will not keep receiving incoming messages. Sending these unsubscribe requests is not always desirable, setting bridge_attempt_unsubscribe to false will disable sending the unsubscribe request.

#bridge_attempt_unsubscribe true

# Set the version of the MQTT protocol to use with for this bridge. Can be one of mqttv50, mqttv311 or mqttv31. Defaults to mqttv311.

#bridge_protocol_version mqttv311

# Set the clean session variable for this bridge. When set to true, when the bridge disconnects for any reason, all messages and subscriptions will be cleaned up on the remote broker. Note that with cleansession set to true, there may be a significant amount of retained messages sent when the bridge reconnects after losing its connection, the subscriptions and messages are kept on the remote broker, and delivered when the bridge reconnects.

#cleansession false

# Set the amount of time a bridge using the lazy start type must be idle before it will be stopped. Defaults to 60 seconds.

#idle_timeout 60

# Set the keepalive interval for this bridge connection, in seconds.

#keepalive_interval 60

# Set the clientid to use on the local broker. If not defined, this defaults to 'local'. If you are bridging a broker to itself, it is important that local_clientid and clientid do not match.

#local_clientid

# If set to true, publish notification messages to the local and remote brokers giving information about the state of the bridge connection. Retained messages are published to the topic $SYS/broker/connection/state unless the notification_topic option is used.

#notifications true

# Choose the topic on which notification messages for this bridge are published. If not set, messages are published on the topic $SYS/broker/connection/state

#notification_topic

# Set the client id to use on the remote end of this bridge connection. If not defined, this defaults to 'name.hostname' where name is the connection name
C.3. MOSQUITTO CONFIGURATION

4/13/22, 7:14 PM Mosquitto Config

745 # and hostname is the hostname of this computer.
746 # This replaces the old "clientid" option to avoid confusion. "clientid"
747 # remains valid for the time being.
748 #remote_clientid
749
750 # Set the password to use when connecting to a broker that requires
751 # authentication. This option is only used if remote_username is also set.
752 # This replaces the old "password" option to avoid confusion. "password"
753 # remains valid for the time being.
754 #remote_password
755
756 # Set the username to use when connecting to a broker that requires
757 # authentication.
758 # This replaces the old "username" option to avoid confusion. "username"
759 # remains valid for the time being.
760 #remote_username
761
762 # Set the amount of time a bridge using the automatic start type will wait
763 # until attempting to reconnect.
764 # This option can be configured to use a constant delay time in seconds, or to
765 # use a backoff mechanism based on "Decorrelated Jitter", which adds a degree
766 # of randomness to when the restart occurs.
767 #
768 # Set a constant timeout of 20 seconds:
769 # restart_timeout 20
770 #
771 # Set backoff with a base (start value) of 10 seconds and a cap (upper limit) of
772 # 60 seconds:
773 # restart_timeout 10 30
774 #
775 # Defaults to jitter with a base of 5 and cap of 30
776 #restart_timeout 5 30
777
778 # If the bridge has more than one address given in the address/addresses
779 # configuration, the round_robin option defines the behaviour of the bridge on
780 # a failure of the bridge connection. If round_robin is false, the default
781 # value, then the first address is treated as the main bridge connection. If
782 # the connection fails, the other secondary addresses will be attempted in
783 # turn. Whilst connected to a secondary bridge, the bridge will periodically
784 # attempt to reconnect to the main bridge until successful.
785 # If round_robin is true, then all addresses are treated as equals. If a
786 # connection fails, the next address will be tried and if successful will
787 # remain connected until it fails
788 #round_robin false
789
790 # Set the start type of the bridge. This controls how the bridge starts and
791 # can be one of three types: automatic, lazy and once. Note that RSBM provides
792 # a fourth start type "manual" which isn't currently supported by mosquitto.
793 #
794 # "automatic" is the default start type and means that the bridge connection
795 # will be started automatically when the broker starts and also restarted
796 # after a short delay (30 seconds) if the connection fails.
797 #
798 # Bridges using the "lazy" start type will be started automatically when the
799 # number of queued messages exceeds the number set with the "threshold"
800 # parameter. It will be stopped automatically after the time set by the
801 # "idle_timeout" parameter. Use this start type if you wish the connection to
802 # only be active when it is needed.
803 #
804 # A bridge using the "once" start type will be started automatically when the
805 # broker starts but will not be restarted if the connection fails.
806 #start_type automatic
807
808 # Set the number of messages that need to be queued for a bridge with lazy
809 # start type to be restarted. Defaults to 10 messages.
# Must be less than max_queued_messages.
#threshold 10

# If try_private is set to true, the bridge will attempt to indicate to the
# remote broker that it is a bridge not an ordinary client. If successful, this
# means that loop detection will be more effective and that retained messages
# will be propagated correctly. Not all brokers support this feature so it may
# be necessary to set try_private to false if your bridge does not connect
# properly.
#try_private true

# Some MQTT brokers do not allow retained messages. MQTT v5 gives a mechanism
# for brokers to tell clients that they do not support retained messages, but
# this is not possible for MQTT v3.1.1 or v3.1. If you need to bridge to a
# v3.1.1 or v3.1 broker that does not support retained messages, set the
# bridge_outgoing_retain option to false. This will remove the retain bit on
# all outgoing messages to that bridge, regardless of any other setting.
#bridge_outgoing_retain true

# If you wish to restrict the size of messages sent to a remote bridge, use the
# bridge_max_packet_size option. This sets the maximum number of bytes for
# the total message, including headers and payload.
# Note that MQTT v5 brokers may provide their own maximum-packet-size property.
# In this case, the smaller of the two limits will be used.
# Set to 0 for "unlimited".
#bridge_max_packet_size 0

# ------------------------------------------------------------
C.3. MOSQUITTO CONFIGURATION

# Certificate based SSL/TLS support
# -----------------------------------------------------------
# Either bridge_cafile or bridge_capath must be defined to enable TLS support
# for this bridge.
# bridge_cafile defines the path to a file containing the
# Certificate Authority certificates that have signed the remote broker
# certificate.
# bridge_capath defines a directory that will be searched for files containing
# the CA certificates. For bridge_capath to work correctly, the certificate
# files must have ".crt" as the file ending and you must run "openssl rehash
# " each time you add/remove a certificate.
#bridge_cafile
#bridge_capath

# If the remote broker has more than one protocol available on its port, e.g.
# MQTT and WebSockets, then use bridge_alpn to configure which protocol is
# requested. Note that WebSockets support for bridges is not yet available.
#bridge_alpn

# When using certificate based encryption, bridge_insecure disables
# verification of the server hostname in the server certificate. This can be
# useful when testing initial server configurations, but makes it possible for
# a malicious third party to impersonate your server through DNS spoofing, for
# example. Use this option in testing only. If you need to resort to using this
# option in a production environment, your setup is at fault and there is no
# point using encryption.
#bridge_insecure false

# Path to the PEM encoded client certificate, if required by the remote broker.
#bridge_certfile

# Path to the PEM encoded client private key, if required by the remote broker.
#bridge_keyfile

# -----------------------------------------------
C.3. MOSQUITTO CONFIGURATION

# PSK based SSL/TLS support
# -----------------------------------------------------------
# Pre-shared-key encryption provides an alternative to certificate based
# encryption. A bridge can be configured to use PSK with the bridge_identity
# and bridge_psk options. These are the client PSK identity, and pre-shared-key
# in hexadecimal format with no "0x". Only one of certificate and PSK based
# encryption can be used on one
# bridge at once.
#bridge_identity
#bridge_psk

# =================================================================
# External config files
# =================================================================
# External configuration files may be included by using the
# include_dir option. This defines a directory that will be searched
# for config files. All files that end in '.conf' will be loaded as
# a configuration file. It is best to have this as the last option
# in the main file. This option will only be processed from the main
# configuration file. The directory specified must not contain the
# main configuration file.
# Files within include_dir will be loaded sorted in case-sensitive
# alphabetical order, with capital letters ordered first. If this option is
# given multiple times, all of the files from the first instance will be
# processed before the next instance. See the man page for examples.
#include_dir
C.4  PCB Schematic and Layout
C.4. PCB SCHEMATIC AND LAYOUT

[Diagram of a PCB schematic showing various components and connections, including capacitors, resistors, and microcontroller pins.]
C.4. PCB SCHEMATIC AND LAYOUT