HyRo

Team 28

Jason Klindtworth | Josh Asher | Layne Nolli

CS461

Fall 2016

Software Design Document

Abstract

High Altitude rockets have a distinct advantage in using Hybrid propulsion systems. These systems are complex and present challenges in remote telemetry including launch initialization and controlling remote fuel filling/disconnect. High altitude rockets also contain an array of sensors that collect data which needs to be visualized in a human friendly format. The goal of HyRo is to provide mechanisms for remotely launching and controlling the fuel systems on a hybrid propulsion system through onboard embedded circuitry/software that communicates to launch team via radio waves. This circuitry/software will also communicate sensor data to the ground. Sensor data is displayed in our visualization software in an appealing, human readable way.

December 1, 2016

Contents

1 Introduction

1.1 Scope

This Design Document described in detail both the structure and the components of the HyRo deployment software interface and database. It also includes the implementation details required to accurately and completely incorporate the specific requirements associated with this project as detailed in the Requirements Document. Because of the contents of this document, it is assumed that the reader has familiarized themselves with the projects Requirements Document and is aware of the specific needs of this software package and graphical user interface. There is also an assumed general knowledge of computer programing knowledge as this document will rely heavily on the explanation of specific software functions and processes that will be designed in order to accomplish all of the needed requirements for this project.

1.2 Purpose

This Design Document serves the following purposes:

- To fully describe the structure, functions, data, and algorithms which will be implemented in the software package.
- To point out and identify both specific and general system resources that will be utilized.
- To assist the HyRo team in the production and implementation of test cases.
- To be used in order to verify full compliance with the requirements of the project.
- To aid the team in the general overview of the software package and what the capabilities are.

1.3 Intended Audience

The majority of the document is written for the benefit of knowledgeable software development and design professionals in order to gain specific knowledge of the HyRo software system and its intended capabilities. The intended audience for this documents is the HyRo team, including the following:

- The project supervisor
- The computer science sub-team working on the software package itself.
- The electrical engineering sub-team working on the hardware in which the software package will be incorporated into.
- the mechanical engineering sub-team working on the mechanical operations of the rocket which the software package will control or manipulate.
- Any related sub-team for the project that requires knowledge of the computer system or interface system for the rocket.

1.4 Definitions

Hybrid Rocket A rocket with an engine that uses both solid and liquid fuel.
I/O Input and Output.
PWM Pulse Width Modulation is used to control signal level on a electrical wire.
Beagle Bone Black A miniature computer that will be used on-board our hybrid rocket to house our software.
Python Dictionary A associative array accessed by key value pairs.
Oxidizer Liquid gas used to accelerate the burning of solid fuel.
Accelerometer Measures acceleration.
Gyroscope Measure tilt relative to the earth.
Magnetometer Measures the electric field around it.
Multi Threaded A program that runs multiple methods in parallel to the main program allowing components to run independently.
Mutex Lock Used to lock control of an area of memory while an operation is completed. If any other parts of the system attempt to access this part of memory while the lock is in place they will not be allowed to.
Boolean A way to represent a true or false value in software.

1.5 Overview

The HyRo software system is composed of two major different components. The first component is housed on-board a hybrid rocket. This component is responsible for gathering sensor data from sensors on the the rocket, sending this data to a ground computer, and receiving commands from the ground system. These commands will initiate electrical components on the rocket. The second component of this system is software running on a traditional computer that is located on the ground near the launch site. This component is responsible for presenting the user with a graphical user interface that will allow a human to send commands to the rocket and monitor data from the rocket that has been visualized in graphs and gauges. This component will also log any important information, mainly commands, that the rocket team requests. The main purpose of our project is to provide remote filling and data visualization. This paper is sectioned into 3 views. The first two views, System Architecture On-board the Rocket and System Architecture on the traditional computer, are explained from a logical viewpoint. The final section, the user interface, is explained from and interface viewpoint.

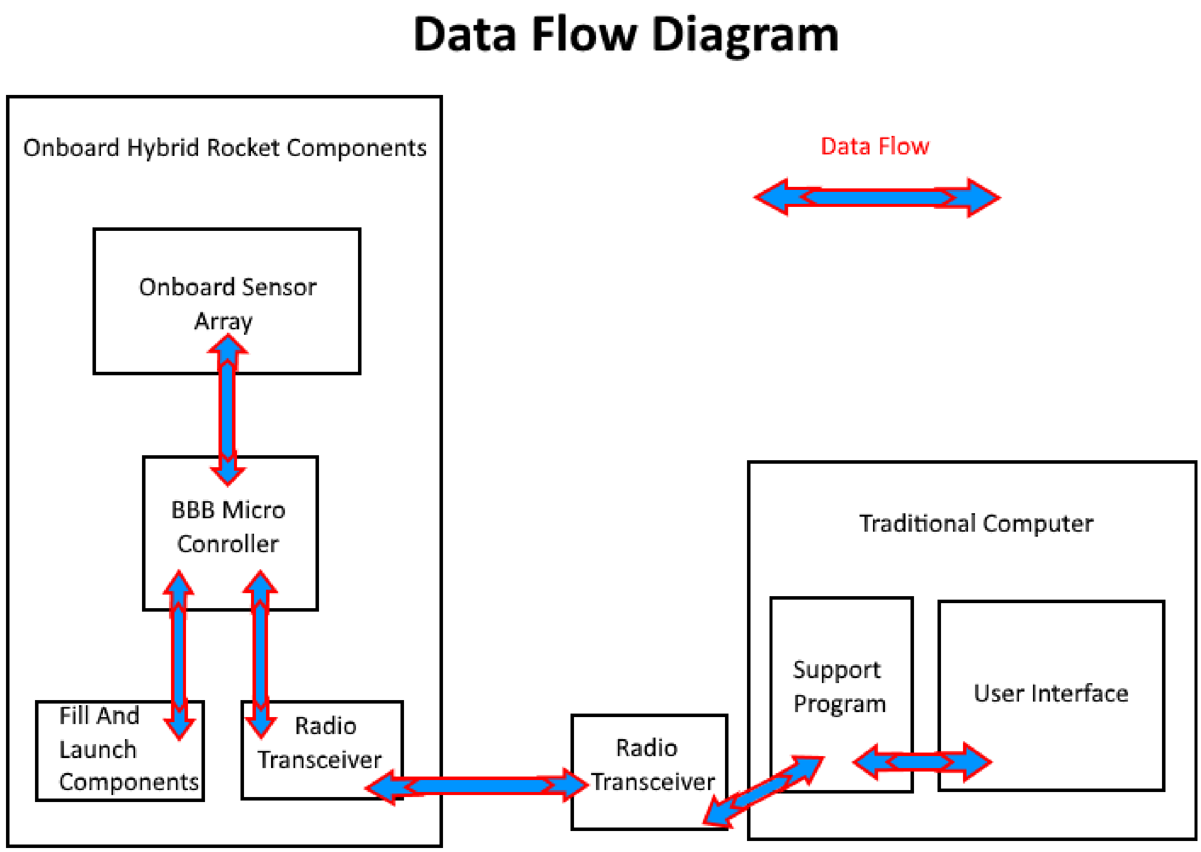
2 Stake Holders and Concerns

The primary stake holders in this project are Nancy Squires and the hybrid rocket team as a whole. The rocket team is composed of Mechanical Engineering students, Electrical Engineering Students, and Computer Science Engineering students. We all have the same design concerns. These concerns include remotely filling and launching a rocket. The launching processes requires a sequence of command to complete. All commands and responses need to be logged in this process. Finally the most important thing is to be able to visualize sensor data while the rocket is in flight. This communication needs to be fast and accurate. This project is an ongoing process and its design will be a living entity. Things will change and we must adapt our design to these changes as the project progresses.

3 System Architecture On-board the Rocket

The SDD module design of the software on-board the hybrid rocket detailed here after will explain the approach, methods, and properties of this component of the system. Software running on-board the rocket will communicate to software running on a traditional computer through a radio transceiver connected to the Beagle Bone Black. For convenience a diagram is provided below to show the entirety of the entire system. This section will only cover the design of the software running on the Beagle Bone Black.

Figure 1: Data Flow Between System Components.



3.1 Components

This section is intended to explain the components of the rocket we will be receiving and/or sending data too. It is a general overview of what could be part of the system as the rocket has not yet been designed. Exact sensors are not presented because they have not be chosen yet. Though collection of data from them will be the same regardless. This allows for easy expansion of sensors in the code if time allows.

At the moment we know the rocket will have an accelerometer, barometric/temperature sensor, and a servo to control filling, arming, and launching of the rocket. The sensors will be polled for data every 500 milliseconds and this data will be held in a buffer to be transmitted. Each sensor/servo has drivers built in python that will be used to access the sensors/servo data and in case of the servo sen PWM signals to control the movement of the servo.

The sensors and servo are connected to the Beagle Bone Black physically through I/O lines available on the board. The drivers define these connections and the functions to preform communication. There is a chance that the rocket design will change and we will need to create our own drivers for the new components. If that happens the driver design will be detailed under this section. At the moment all components used last year currently have drivers available.

3.2 Data Format

There are two buffers in this program. One to hold the sensor data and one to hold commands received. They are both globally available to the entire program.

3.2.1 Sensor Data Buffer

The sensor data buffer will be a python dictionary with keys relating to the name of the sensor output. For examples the temperature data will be stored in and accessed by the field "temperature". The timestamps field is important to the radio transceiver polling function (detailed below) as this is what it uses to decided if the data is new. Some of these data items are not planned on being used by the ground software, but are included for future data visualization expansion.

Buffer Name dataBuffer
Buffer Keys -

- **time_stamp** Time the current data was written to this buffer.
- **altitude** The altitude above ground level reading from the altitude sensor.
- **temp** The temperature reading from the temperature sensor.
- **a_x** Accelerometer data from x axis.
- **a_y** Accelerometer data from y axis.
- **a_z** Accelerometer data from z axis.
- **g_x** Gyroscope data from the x axis.
- **g_y** Gyroscope data from the y axis.
- **g_z** Gyroscope data from the z axis.
- **m_x** Magnetometer data from the x axis.
- **m_y** Magnetometer data from the x axis.
- **m_z** Magnetometer data from the x axis.
- **tank_pres** The pressure of the oxidizer tank.
- **chamber_pres** The pressure of the combustion chamber

3.2.2 Command Buffer

The command buffer will be used to store commands received from the traditional computer software component to be accessed by the command thread. A python list data structure will be used to provide easy pushing and popping of values. When a command is detected it is added to this array and as it is processed it is removed from the array.

Buffer Name: commandBuffer
Example:
commandBuffer = // empty
New command Received = arm add to buffer
commandBuffer = 0 : 'arm'
New command Received = disarm add to buffer
commandBuffer = 0 : 'arm', 1: 'disarm'
Command Processed = 'arm'
commandBuffer = 0 :'disarm'
Command Processed = 'disarm'
commandBuffer = //empty

3.3 Methods and Threads

This component will be designed in using multi threaded approach with helper functions. Each thread will be responsible for a certain aspect of the system. They will communicate through the two buffers detailed above. This will require mutex locks be put on the buffers prior to any action taken from the independent threads. The following is a list of the threads and helper functions with their descriptions and interactions.

3.3.1 Sensor Polling Thread

Purpose:
This thread is used to pull data from the sensors every 500 milliseconds. The data is then placed into the sensor data buffer.

Definition:
sensor_thread()

Inputs:
Sensor data from the varies sensors. This is not a parameter input instead the function reads the data from the sensors as input. **Outputs:**
Sensor data to the sensor data buffer. **Process/Algorithm:**
When the thread starts it initially creates sensor objects to allow access to the sensors data through the use of their read functions. Afterwards this thread will run in a continuous loop. The loop will collect data from each of sensors in a temporary array until all sensors have been polled. Data by calling read or its equivalent of the related sensor object. When the thread has collected data from all sensors it will place a mutex lock on the sensor data buffer and write the new data to the buffer along with a time stamp.

3.3.2 Command Processing Thread

Purpose:
The command processing thread will monitor for new commands in the commands buffer. If commands are received it will process them and take appropriate action. This could include sending an error message to the ground computer through the radio transceiver send function.

Definition:
command_thread()

Inputs:
Data from the command buffer.

Outputs:
Signals to servo and messages to rf_send()

Calls:
rf_send(message)

Process/Algorithm:
When the thread initializes all servo objects will be instantiated to allow for access their corresponding electrical signal lines. The command processing thread will monitor the command buffer for new commands. If the command buffer length is not zero there is a new command in the buffer.The buffer will be checked every 250 milliseconds for a new command. Commands will be processed through a set of conditional statements. Commands must be performed in a specific sequence detailed below next to the command explanation. Boolean flags will be used to determine if a command has been previously processed. Each command will result in a call to the respective servo objects electrical signal line. This will cause the physical servo to change positions. The values output on the signal lines will be determined by the servo used. These details have not been solidified by the rocket team yet. Regardless the functionality will be the same only the values outputted will change once an appropriate device has been chosen.

If a command passes its conditional statement the servo function will be called on the appropriate servo object with a electrical signal value corresponding to its proper adjustments. The adjustment measurements will be determined by the mechanical engineers on the project and hard coded into the software. After each command is successfully processed an acknowledgment will be sent to rf_send(message) to inform the ground software a command was successfully processed. If a command does not pass a conditional a message will be sent to radio transceiver by calling rf_send(message) with the appropriate error message. The command will then be dropped and no action will be taken. Once the command sequence has been fulfilled the rocket is put into the launch state. This is dictated by a global boolean variable.

Commands:

fill This will adjust a servo to fill the oxidizer tank before launch. Sequence number 1.
arm This will adjust a servo prepare the rocket for ignition. Sequence number 2.
ignition This will send a signal to igniter to ignite the rocket fuel. Sequence number 3.
launch This will adjust a servo to release the rocket from its base. Sequence number 4.
disarm This will adjust a servo to reverse the arming process. Can be used at any time.
abort This will adjust all servos to their initial state. Can be used at any time.

3.3.3 Radio Transceiver Thread

Purpose:
The radio transceiver thread is responsible for monitoring the radio transceiver for incoming data from the software running on the traditional computer.

Definition:
comm_thread()

Inputs:
Data from the radio transceiver and data from the sensor data buffer.

Outputs:
Data to the command buffer and data to be sent by the radio transceiver.

Calls:
rf_send(message)

Process/Algorithm:
When the radio transceiver thread initializes it attempts to read from the radio transceiver every 100 milliseconds. At this point the rocket is in a the pre-launch state which is dictated by a boolean flag. While the rocket is in the pre-launch stage this thread will only poll the radio transceiver object for new messages. When a command is received it is placed into the command buffer by first putting a mutex lock on the buffer and pushing the new command onto the buffer. The thread will repeat this process until it detects that the pre-launch stage has passed.

After the system has enter the launch stage this thread will change its behavior. Instead of monitoring for commands it will monitor the sensor data buffer for new data every 250 milliseconds. It detects new data by checking the time stamp value in the sensor data buffer dictionary. It initially stores the first time value and then upon finding a newer time value it will replace the old time value with the new time stamp and send the buffer into the radio transceiver by calling rf_send(message) with the values from the dictionary as parameters. It repeats this process indefinitely.

3.3.4 Main Thread

Purpose:
The main thread in the entry point into the software. It runs all initialization functions and starts all threads.

Definition:
main()

Inputs:
None

Outputs:
None

Calls:
init(), comm_thread(), sensor_thread(), command_thread

Process/Algorithm:
The main function calls the initialization function then creates the radio transceiver thread, the command processing thread, and the sensor thread. It will shut down all threads on program exit.

3.3.5 Initialization Function

Purpose:
Initialize any servos, sensors, or global variables.

Definition:
init()

Inputs:
None

Outputs:
None

Process/Algorithm:
Initializes all global variables to default values. Then sets up any global comportments like servos to their default values defined by the mechanical engineering team.

3.3.6 Radio Transceiver Send Function

Purpose:
Writes a message to send to the radio transceiver.

Definition:
send_rf(message)

Inputs:
Message to send.

Outputs:
Message to radio transceiver.

Process/Algorithm:
Upon being called in turns calls the write function of the radio transceiver object with the message provided in the message parameter.

3.4 Rationale

Our projects major functions are to get data from a rocket to a traditional computer and visualize it. As far as the software running on the beagle bone black its main function is to process commands and transmit sensor data. The approach we have taken will allow for fast easy maneuvering of data by using a simply multi threaded process that will store the data directly in memory. The threads will allow for simultaneous operation of these features. The approach we chose is procedural instead of object oriented, but this is to provide simplicity. Our program will not be large compared to modern programs, but needs to preform very specific tasks.

3.5 Language

We have chosen to use python because of its simplicity and strong API for USB reading and graphic rendering. It also is a stronger cross platform language then other options we explored. Another determining factor in this choice was last years success with this language.

4 System Architecture on a Traditional Computer

The SDD module design of the software running on a traditional computer detailed here after will explain the approach, methods, and properties of this component of the system. Software running on the traditional computer will communicate to the software detailed above running on-board the rocket. This component of the system will be responsible for monitoring radio transmissions, sending commands, logging, and the user interface back end.

4.1 Overview

In the following sections we describe the components of this software minus the actual user interface that is detailed in the next section. We go over the graphical back end toolkits that will be used to draw the user interface, the data format of buffers and logs, worker threads, button listener functions, data conversion functions, and data visualization functions (drawing functions).

4.2 GUI and Graphing Toolkits

We will be using the Python TKinter library which provides a basis for windows, buttons, canvas, and other varies components we will need to create our graphical interface. This library includes event listeners to easily detect button presses and canvas to easily draw gauges and graphs. All sub components like buttons are called widgets in TKinter.

On top of Tkinter we will be using Matplotlib which is a graphical plotting library. Matplotlib provides functions to be easily able to graph data and scale those graphs. Graphs will be drawn on TKinter canvas widgets.

4.3 Data Format

4.3.1 Sensor Data Buffer

The sensor data buffer will be a python dictionary with keys relating to the name of the sensor output. For examples the temperature data will be stored in and accessed by the field "temperature". The timestamps field is important to redraw function (detailed below) as this is what it uses to decided if the data is new.

Buffer Name dataBuffer
Buffer Keys -

- **time_stamp** Time the current data was written to this buffer.
- **altitude** The altitude above ground level reading from the altitude sensor.
- **temp** The temperature reading from the temperature sensor.
- **accell** Accelerometer data from x axis.
- **velocity** Accelerometer data from y axis.
- **tank_pres** The pressure of the oxidizer tank.
- **chamber_pres** The pressure of the combustion chamber

4.3.2 Log Buffers

Log buffers will hold previously received data for a particular key from the sensor data buffer. Each key (listed in the last section) will have its own buffer. As data is processed it will be placed into its log buffer. The log buffers will each hold up to 256 of the last entries. This is to allow time graphs to be plotted. These buffers are 2 dimensional arrays that hold that sensors value and a time stamp.

Buffers -

- **altitude** The altitude above ground level reading from the altitude sensor.
- **temp** The temperature reading from the temperature sensor.
- **accell** Accelerometer data from x axis.
- **velocity** Accelerometer data from y axis.
- **tank_pres** The pressure of the oxidizer tank.
- **chamber_pres** The pressure of the combustion chamber

4.3.3 Command Response Message Format

Command response message will be sent as a string with the command that is being responded too followed by a colon followed then by the response message from the on-board system. All commands have not been finalized by the rocket team, but will follow this format. For example:

Launch: Sequence has not been followed. Rocket is not ready to launch.

4.3.4 Data Log Format

When data is logged each entry will be place on a separate line in a text file. The line will consist of a time stamp and then a list of key value pairs from the data buffer. Key value you pairs will have an equal sign between them. The time stamps are formatted as follows:

6-1-2016:12:05:20

Which represents Month-Day-Year : Hours : Minute : Second

Here is an example complete data log entry:

6-1-2016:12:05:20 altitude="740" temp="70" accel="10" velcoity="15" tank_pres="20" chamber_pres="30"

4.3.5 Command Log Format

When commands are logged each entry will be placed on a separate line of a text file. The line will consist of a time stamp followed by the command that was issued. Times stamps are in the same format as described in the previous section. Here is an example complete command log entry.

6-1-2016:12:05:20 Abort

4.3.6 Response Log Format

When responses are logged each entry will be placed on a separate line of a text file. The line will consist of a time stamp followed by the command that was responded to followed by a colon with the response to the command appended to the end. Times stamps are in the same format as described in the previous section. Here is an example complete response log entry.

6-1-2016:12:05:20 Ignite:Sequence has not been followed. Rocket not ready to ignite.

4.4 Methods and Threads

This component will be designed in using multi threaded approach with helper functions. Each thread will be responsible for a certain aspect of the system. They will communicate through the two buffers detailed above. This will require mutex locks be put on the buffers prior to any action taken from the independent threads. The following is a list of the threads and helper functions with their descriptions and interactions.

4.4.1 Main Thread

The main thread in the entry point into the software. It runs all initialization functions and starts all threads.

Definition:
main()

Inputs:
None

Outputs:
None

Calls:
init(), data_thread, redraw_thread

Process/Algorithm:
The main function calls the initialization function then creates the data thread and the redraw thread. It will shut down all threads on program exit.

4.4.2 Data Thread

The data thread is responsible for polling the radio transceiver for new data or command responses messages. Then passing the data to converter functions or the command response message to the command response function..

Definition:
data_thread()

Inputs:
None

Outputs:
New data to converter functions.

Calls:
convTemp(data), convCHPressure(data), convAlititude(pressure, temp), convTankPressure(d), convAccell(data), convVelocity(data)

Process/Algorithm:
The data thread continuously checks for data coming in on the radio transceiver by callings read on the radio transceiver object every 250 milliseconds. When a data packet arrives this thread processes the data by sending the value of the appropriator key to the appropriate data converter. For example the key "temperature" would be sent to the convTemp function. Which will handle the data from there on. If a command response message is received it is passed to the command response processing function.

4.4.3 Redraw Thread

The redraw thread is responsible for monitoring for new converted data and passing the data to the appropriate redraw functions.

Definition:
redraw_thread()

Inputs:
None

Outputs:
None

Calls:
drawTemp(data), drawCHPressure(data), drawAlititude(data), drawTankPressure(data), drawAccel(data), drawVelovity(data)

Process/Algorithm:
The redraw function will check the time stamp value of the data buffer and if it is newer than the one it had previously recorded it will process the data. Upon detecting new data the redraw function will send each new data value to its appropriate draw function. The data at this point has already been converted for the drawing functions so it may be

passed directly to them. For example it will pass the temperature data from the data buffer to the drawTemp() function. This thread is only responsible for data delivery to the temperature, chamber pressure, altitude, tank pressure, acceleration, and velocity redraw functions. Each draw function is responsible for graphing the data to its specific canvas.

4.4.4 Radio Transceiver Send Function

Purpose:
Writes a message to send to the radio transceiver.

Definition:
send_rf(message)

Inputs:
Message to send.

Outputs:
Message to radio transceiver.

Process/Algorithm:
Upon being called in turns calls the write function of the radio transceiver object with the message provided in the message parameter.

4.4.5 Initialization Function

Purpose:
Initializes global buffers, global objects, global variables, and the user interface.

Definition:
init()

Inputs:
None.

Outputs:
None.

Process/Algorithm:
This function initializes all global buffers, objects and variables to their default values. It then creates the window object, all button widgets, all canvas widgets for graphs, and the graphing menu widget and buttons. Once all user interface items have been initialized it calls the main windows drawing method to display the initial interface. This function is called before any of the threads in this component are started.

4.4.6 Process Command Response Function

Purpose:
Receives command response messages from the data thread and processes them. This function will inform the user of any failed or out of sequence commands.

Definition:
command_response(message)

Inputs:
Command response to process.

Outputs:
Logs responses to text file and message to screen.

Calls: drawMessage()

Process/Algorithm:
Upon receiving a command response message from the data thread this function opens the command response log and appends the message to the log with a newline. It then passes the message to the message draw function.

4.4.7 Button Event Listeners

Button listeners are provided by Tkinter as call back functions bound to the button the window. There will be two sets of button event listeners in the project. One set will be bound the the command buttons and the other to the graphing options buttons.

4.4.7.1 Command Button Listeners

Each command button placed on the screen (shown in section 5) will have a function bound to it that will send the corresponding command string to the rf_send() function. When a button is pressed that string is sent directly to the rf_send function which in turns write it to the radio transceiver. If a command fails the rocket will send a error response. It is not the responsibility of these functions to make sure the command is in the correct order. Below is an example command button callback function.

sendFill()

Each button will have its own functions like this. In this example the sendFill() function will call the rf_send() function with the message "filll".

4.4.7.2 Graphing Options Buttons

Each graphing option button placed on the screen (shown in section 5) will have a function bound to it that will change the graph view of the multi functional graph window. When a graphing button is pressed it call its corresponding draw function that will replace the current graph in this window. Below is an example graphing option callback function:

tempAsTimeClicked()

Each button will have its own functions like this. In this example the tempAsTimeClicked() function will call the drawTempAsGraph() function whitch will draw the temperature time graph in the multi functional graph window.

4.4.8 Data Log Function

Purpose:
Logs the data it is passed to the data log text file.

Definition:
data_log(data)

Inputs:
Dictionary of data to log.

Outputs:
A line representing the data passed in to the data log text file.

Calls: drawMessage() Process/Algorithm:
This function takes the data from data parameter and converts it to the data log format string. It then writes this string as a new line to the data log text file.

4.4.9 Command Log Function

Purpose:
Logs the data it is passed to the command log text file.

Definition:
command_log(data)

Inputs:
Command to log.

Outputs:
A line representing the command passed in to the command log text file.

Calls: drawMessage() Process/Algorithm:
This function takes the a command message from data parameter and converts it to the command log format string. It then writes this string as a new line to the command log text file.

4.4.10 Converter Functions

Each of the follow functions convert data they receive to the proper format for the drawing functions. They then store the data in the main data buffer for the drawing functions to access. Each converter function is described in its specific details below.

4.4.10.1 Temperature Converter

Purpose:
Converts raw temperature sensor data to data suitable for the drawing functions.

Definition:
convTemp(data)

Inputs:
Raw temperature data.

Outputs:
Converted data to data buffer.

Process/Algorithm:
This function receives data from the data thread in its data parameter to convert to a temperature value suitable for graphing. It then will store the converted temperature in the data buffer. It will convert the data by using a multiplier and offset provided on the data sheet of the temperature sensor.Temperature will be converted to Fahrenheit.

4.4.10.2 Chamber Pressure Converter

Purpose:
Converts raw pressure sensor data to data suitable for the drawing functions.

Definition:
convCHPressure(data)

Inputs:
Raw chamber pressure data.

Outputs:
Converted data to data buffer.

Process/Algorithm:
This function receives data from the data thread in its data parameter to convert to a pressure value suitable for graphing. It then will store the converted pressure in the data buffer. It will convert the data by using a multiplier and offset provided on the data sheet of the pressure sensor used in the chamber of the rocket. Pressure will be converted to pounds per square inch.

4.4.10.3 Altitude Converter

Purpose:
Converts raw barometric pressure sensor readings to data suitable for the drawing functions.

Definition:
convAltitude(pressure, temp)

Inputs:
Raw barometric pressure data.

Outputs:
Converted data to data buffer.

Process/Algorithm:
This function receives data from the data thread in two parameters. One is pressure and one is temperature, both required to calculate altitude. Altitude will be measured in feet. It then will store the converted temperature in the data buffer. This conversion is currently provided as a function call in the current sensors library. If the design of the rocket changes the barometric pressure equation will have to be used. If that is the case that will be explained in this section. For now converting the temperature and pressure to altitude requires a call to the altitude conversion function which is part of the altitude sensor library.

4.4.10.4 Tank Pressure Converter

Purpose:
Converts raw pressure sensor data to data suitable for the drawing functions.

Definition:
convTankPressure(data)

Inputs:
Raw tank pressure data.

Outputs:
Converted data to data buffer.

Process/Algorithm:
This function receives data from the data thread in its data parameter to convert to a pressure value suitable for graphing. It will convert the data by using a multiplier and offset provided on the data sheet of the pressure sensor used on the oxidizer tank of the rocket. Pressure will be converted to pounds per square inch.

4.4.10.5 Acceleration Converter

Purpose:
Converts raw accelerometer sensor data to data suitable for the drawing functions.

Definition:
convAccell(data)

Inputs:
Raw acceleration data.

Outputs:
Converted data to data buffer.

Process/Algorithm:
This function receives data from the data thread in its data parameter to convert to an acceleration value suitable for graphing. The parameter will be a raw acceleration reading from the sensor. It will convert this to the Meters per second squared with the formula provided by the sensor manufacturer. This may include more math if the accelerometer API is not complete This will be added to this document if need be. It then will store the data in the data buffer.

4.4.10.6 Velocity Converter

Purpose:
Converts raw acceleration data to velocity data suitable for the drawing functions.

Definition:
convVelocity(data)

Inputs:
Raw acceleration data.

Outputs:
Converted data to data buffer.

Process/Algorithm:
This function receives data from the data thread in its data parameter to convert to a velocity value suitable for graphing. The parameter will be a raw acceleration reading from the sensor. It will convert this to the Meters per second by first converting it to meters per second squared (normal acceleration) with the formula provided by the sensor manufacturer. Then it will take the integral of the acceleration to obtain the velocity. It then will store the data in the data buffer.

4.4.11 Drawing Functions

Each of the following drawing functions are called by the redraw thread when data becomes available to be displayed on the screen. They do not take data as input to be displayed, but instead read the data from the data buffer or the log buffer. Except for the message drawing routing it takes a input message to be drawn in the message window. Each function is responsible for a different data item and will display it according to the format of that data. The following is a list of the drawing functions and their specific operation.

4.4.11.1 Message Drawing Routine

Purpose:
Display command response message in the message window.

Definition:
drawMessage(message)

Inputs:
Message string to draw.

Outputs:
Message to the message window.

Process/Algorithm:
This function will take as its input parameter a command message and then draw the string in the message window of the user interface.

4.4.11.2 Temperature Drawing Routine

Purpose:
Draw the temperature data on the user interface.

Definition:
drawTemp()

Inputs:
None.

Outputs:
Temperature gauge on the user interface.

Process/Algorithm:
This function will pull temperature data from the data buffer and draw a gauge representing the data on the temperature

canvas.

4.4.11.3 Chamber Pressure Drawing Routing

Purpose:
Draw chamber pressure on the user interface.

Definition:
drawCHPressure()

Inputs:
None.

Outputs:
Chamber pressure gauge on the user interface.

Process/Algorithm:
This function will pull chamber pressure data from the data buffer and draw a gauge representing the data on the chamber pressure canvas.

4.4.11.4 Altitude Drawing Routine

Purpose:
Draw rocket altitude on the user interface.

Definition:
drawAltitude()

Inputs:
None.

Outputs:
Altitude gauge on the user interface.

Process/Algorithm:
This function will pull altitude data from the data buffer and draw a gauge representing the data on the altitude canvas.

4.4.11.5 Tank Pressure Drawing Routine

Purpose:
Draw oxidizer tank pressure on the user interface.

Definition:
drawTankPressure()

Inputs:
None.

Outputs:
Oxidizer tank pressure gauge on the user interface.

Process/Algorithm:
This function will pull tank pressure data from the data buffer and draw a gauge representing the data on the tank pressure canvas.

4.4.11.6 Acceleration Drawing Routine

Purpose:
Draw acceleration graph on the user interface.

Definition:
drawAccell()

Inputs:
None.

Outputs:
Acceleration graph on the user interface.

Process/Algorithm:
This function will pull acceleration data from the data buffer and draw a graph representing the data on the acceleration canvas.

4.4.11.7 Velocity Drawing Routine

Purpose:
Draw velocity graph on the user interface.

Definition:
drawVelocity()

Inputs:
None.

Outputs:
Velocity graph on the user interface.

Process/Algorithm:
This function will pull velocity data from the data buffer and draw a graph representing the data on the velocity canvas.

4.4.11.8 Draw Temperature as Graph Routine

Purpose:
Draws temperature vs time as graph on the user interface.

Definition:
drawTempAsGraph()

Inputs:
None.

Outputs:
Temperature vs time graph on the user interface.

Process/Algorithm:
This function will take multiple data points from the temperature log buffer and plot them with their timestamps on the user interface in the time graph canvas.

4.4.11.9 Draw Chamber Pressure as Graph Routine

Purpose:
Draws chamber pressure vs time as graph on the user interface.

Definition:
drawCHPAsGraph()

Inputs:
None.

Outputs:
Chamber pressure vs time graph on the user interface.

Process/Algorithm:
This function will take multiple data points from the chamber pressure log buffer and plot them with their timestamps on the user interface in the time graph canvas.

4.4.11.10 Draw Altitude as Graph Routine

Purpose:
Draws altitude vs time as graph on the user interface.

Definition:
drawAltAsGraph()

Inputs:
None.

Outputs:
Altitude vs time graph on the user interface.

Process/Algorithm:
This function will take multiple data points from the altitude log buffer and plot them with their timestamps on the user interface in the time graph canvas.

4.4.11.11 Draw Tank Pressure as Graph Routine

Purpose:
Draws tank pressure vs time as graph on the user interface.

Definition:
drawTankPAsGraph()

Inputs:
None.

Outputs:
Tank pressure vs time graph on the user interface.

Process/Algorithm:
This function will take multiple data points from the tank pressure log buffer and plot them with their timestamps on the user interface in the time graph canvas.

4.5 Rationale

This part of our system reads and sends data. A big part of this decision was the choice on Python. There are great graphics libraries available for Python that are tailored to our needs. This will provide us with and easy to use and build graphical user interface. The back in is a procedural based multi threaded system that provides simplicity and easy data access.

4.6 Language

We have chosen to use python because of its simplicity and strong API for USB reading and graphic rendering. It also is a stronger cross platform language then other options we explored. Another determining factor in this choice was last years success with this language.

5 User Interface Design

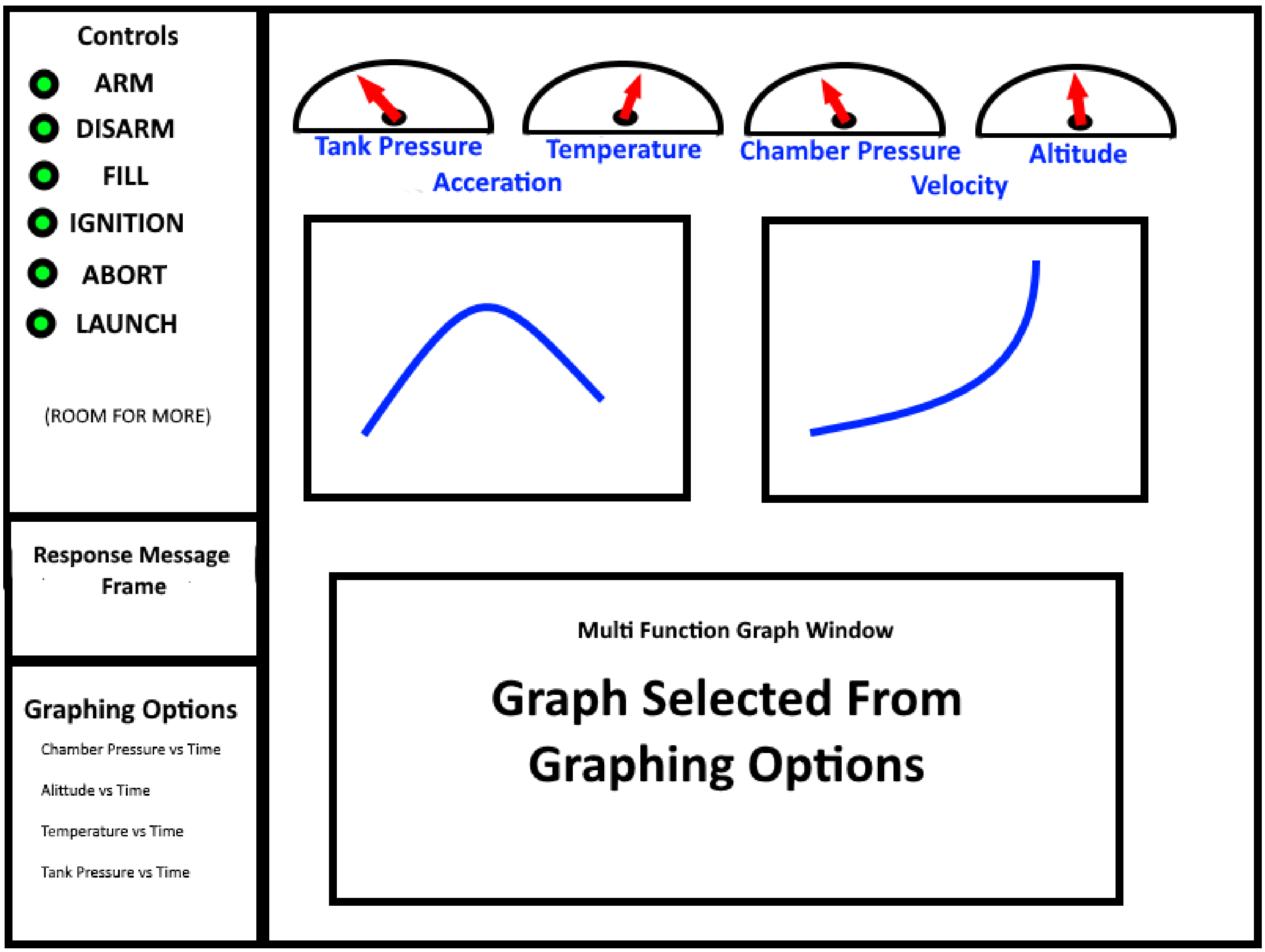
The graphical user interface is the driving force behind this entire project. It may not seem as complicated as some modern interfaces, but its serves a very important purpose. The purpose is to visually represent data from a hybrid rocket and provide button to issue the rocket commands. These functions have been lacking in previous years rockets team and are the reason we were brought on board this project. Detailed in the following sections are these sections of the user interface.

- The Command Buttons
- The Message Frame
- The Graphing Options Buttons
- The Gauges and Graphs

5.1 Drawing of Complete Screen

Here is a basic drawing of the entire screen. This drawing will be replaced with a screen shot once the program has reached a stage where the components are all visible.

Figure 2: Mock up of the graphical user interface



5.2 The Command Buttons

The command buttons are housed on the left side of the screen as depicted in the image above. These buttons will cause a command to be sent through the radio transceiver on the traditional computer to the radio transceiver on-board the rocket. The rocket will process the command and send a response back to be displayed in the message frame. The command buttons and their descriptions are as follows.

Fill This will attempt to initiate the oxidizer fill process while the rocket is on the ground.
Arm This will adjust a servo prepare the rocket for ignition. Sequence number 2.

Ignition This will send a signal to igniter to ignite the rocket fuel. Sequence number 3.
Launch This will adjust a servo to release the rocket from its base. Sequence number 4.
Disarm This will adjust a servo to reverse the arming process. Can be used at any time.
Abort This will adjust all servos to their initial state. Can be used at any time.

5.3 The Message Frame

This is a small message box on the left hand side of the user interface directly under the command buttons. This box will display response message from the rocket. As message come in they will be appended to this message box. The box will be able to scroll if the messages exceeded the size of the frame.

5.4 The Graphing Options Buttons

These buttons are located on the left side of the user interface under the message frame. Each button will correspond to a specific graphing capability. When clicked on there will change which graph is displayed in the multi functional graphing window. Buttons and their corresponding graphs are listed below.

Chamber Pressure vs. Time Clicking this button will change the multi function graph window to a graph of chamber pressure vs. time
Altitude vs. Time Clicking this button will change the multi function graph window to a graph of altitude vs. time.
Tank Pressure vs. Time Clicking this button will change the multi function graph window to a graph of tank pressure vs. time.
Temperature vs. Time Clicking this button will change the multi function graph window to a graph of temperature vs. time.

5.5 The Gauges and Graphs

Taking up most of the user interface on the right hand side are the gauges and graph canvases. Currently their will be 4 gauges that look like traditional gauges you would see on car dash board. They will be a semi circle with marks representing possible values of the data item. A arrow will be draw to represent the current value of the data. Below those gauges are two graphs representing velocity and acceleration these will graph these two data items vs time. Below the velocity and acceleration graphs is the multi function graphing window. This will display a user selected graph from the graphing options buttons selection menu. This will default to altitude vs time. Listed below are the gauges, graphs, and their units.

5.5.1 Tank Pressure Gauge

Units
Tick marks are measured in pounds per square inch from 0 to 200.

Description
This gauge represents the current pressure of the oxidizer tank. It will rise and fall along with the tank pressure. This value can also be graphed in the multifunction window.

5.5.2 Temperature Gauge

Units
Tick marks are measured in degrees Fahrenheit from 32 to 300. **Description**
This gauge represents the temperature wherever the temperature sensor is located. It will rise and fall along with the temperature surrounding the temperature sensor. This value can also be graphed in the multifunction window.

5.5.3 Chamber Pressure Gauge

Units
Tick marks are measured in pounds per square inch from 0 to 200.

Description
This gauge represents the pressure of the combustion chamber of the rocket. It will rise and fall along with the chamber pressure of the rocket. This value can also be graphed in the multifunction window.

5.5.4 Altitude Gauge

Units
Tick marks are measured in feet above sea level from 0 to 10000.

Description
This gauge represents the altitude of the rocket. It will rise and fall along with the altitude of the rocket. This value can also be graphed in the multifunction window.

5.5.5 Acceleration Graph

Units
The x axis of the graph will be measured in meters per second squared and the y axis will be time from launch in seconds.

Description
This graph represents the acceleration of the rocket with respect to time. The graph will be able to slide if the amount of data points exceeds the size of the graph window.

5.5.6 Velocity Graph

Units
The x axis of the graph will be measured in meters per second and the y axis will be time from launch in seconds.

Description
This graph represents the velocity of the rocket with respect to time. The graph will be able to slide if the amount of data points exceeds the size of the graph window.

5.5.7 Multi Function Graph

Units
The x axis of the graph will be measured in different units depending on the graph chosen and the y axis will be time from launch in seconds.

Description
This multi function graph will be the place where user selected graphing items are displayed.

6 Summary

The previous section have in detailed described our hybrid rockets software system. We have described how we will collect sensors data, send sensor data, receive and act on commands, and provide a command and visualization interface for the rocket team members on the ground. We have detailed the data formats, log formats, radio transceiver routines, data conversion routines, drawing routines and the threads that will control all these components. These are all housed in a threaded procedural system for simplicity and speed. All these components combined will allow for remote filling, launching, and data visualization of a hybrid rocket.

7 Bibliography

References

[1] BeagleBone.org,
(Tues November 29 2016),
Beagle Bone Black product information and website,
URL http://beagleboard.org/black

[2] Unimap.edu example SDD,
(Tues November 29 2016),
Unimap software development examples,
URL http://portal.unimap.edu.my/portal/page/portal30/Lecturer%20Notes/KEJURUTERAAN_KOMPUTER/Semester%202%20Sidang%20Akademik%2

[3] IEE 1016 Software Design Specifications,
(Tues November 29 2016),
IEE 1016 Document availble on campus,

[4] Python.org wiki,
(Tues November 29 2016),
Tkinter python wiki on python.org,
URL https://wiki.python.org/moin/TkInter

[5] Matplotlib.org,
(Tues November 29 2016),
Matplotlib.org website documentation,
URL http://matplotlib.org/

Students:

Jason Klindtworth

Signature

Date

Josh Asher

Signature

Date

Layne Nolli

Signature

Date

Client:

Nancy Squires

Signature

Date