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.

November 28, 2016

Contents

1 Introduction

1.1 Scope

1.2 Purpose

1.3 Intended Audience

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 References

1.6 Overview

2 Design Considerations

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.

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

3.5 Language

4 System Architecture on a Traditional Computer

4.1 Overview

4.2 GUI and Graphing Toolkits

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

4.4 Log Format

4.5 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.

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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.

4.5.5 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.

4.5.6 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.

4.5.7 Button Event Listener

4.5.8 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.

4.5.9 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.

4.5.10 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.

4.5.10.1 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.

4.5.10.2 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.

4.5.10.3 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.

4.5.10.4 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.

4.5.10.5 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.

4.5.10.6 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.

4.5.11 Drawing Functions

Each of the following drawing functions are called by the redraw thread when data becomes available to be displayed on the screen. The do not take data as input to be displayed, but instead read the data from the data buffer. 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.

4.5.11.1 drawMessage()

4.5.11.2 drawStatus()

4.5.11.3 drawTemp()

4.5.11.4 drawCHPressure()

4.5.11.5 drawAltitude()

4.5.11.6 drawTankPressure()

4.5.11.7 drawVelocity()

4.5.11.8 drawTempAsGraph()

4.5.11.9 drawCHPAsGraph()

4.5.11.10 drawAltAsGraph()

4.5.11.11 drawTankPAsGraph()

5 User Interface Design

6 Index

Students:

Jason Klindtworth

.....

Signature

.....

Date

Josh Asher

.....

Signature

.....

Date

Layne Nolli

.....

Signature

.....

Date

Client:

Nancy Squires

.....

Signature

.....

Date