HyRo

Team 28

Jason Klindtworth | Josh Asher | Layne Nolli

CS461

Fall 2016

Requirements Document

November 27, 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.

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 out put. 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.

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 software

Students:

Jason Klindtworth

.....

Signature

.....

Date

Josh Asher

.....

Signature

.....

Date

Layne Nolli

.....

Signature

.....

Date

Client:

Nancy Squires

.....

Signature

.....

Date