Weather Balloon Surveillance Package

Code Development Document

Timothy Sam, Kevin Parlak, Nawaf Abdullah, Evan Kerr, Ali Wahab

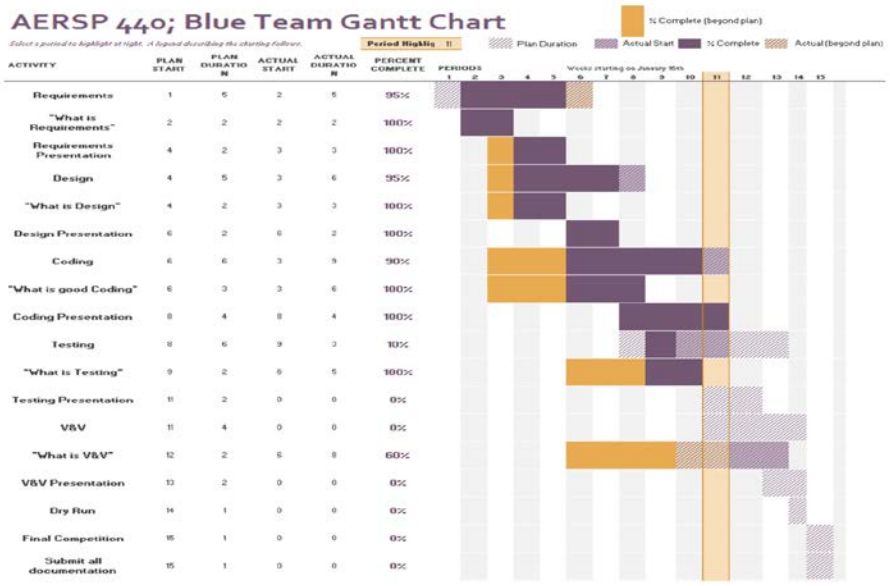
Table of Contents

1. Introduction

1.1 Project Summary

The project in working will carry a surveillance package that must transmit data such as: pictures, sensor readings, and location information to a ground station for which the user can interact with and observe the behavior of the package. The package will be carried by a weather balloon which must conform to airspace guidelines set by the FAA and GSBC. The package must finally be returned safely for the user's retrieval.

1.2 Project Progress Report

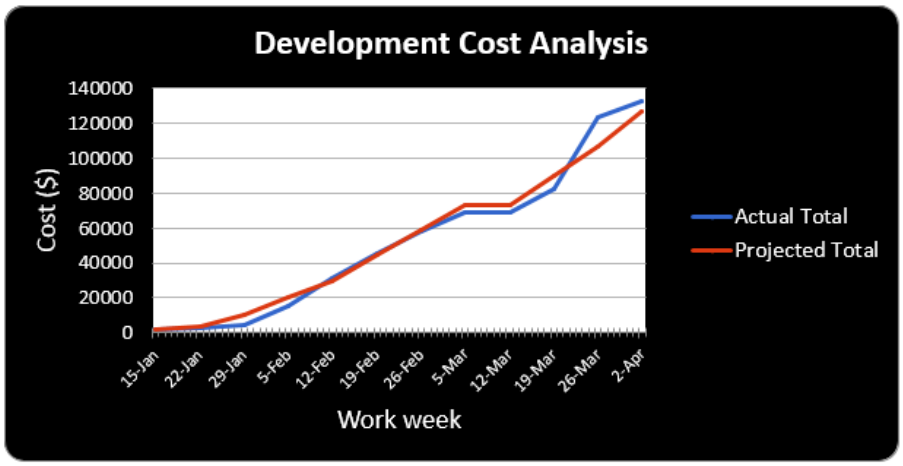


1.3 Financial Progress Report

COCOMO Estimation	
Type	Cost ($)
Organic	172934.40
Semi-detached	216500.40
Embedded	269219.27

Final estimated bill: $191200

Total current costs: $127200



2. Recap

2.1 Requirements

2.1.1 User Requirements

Regulatory

2.1.1.1 The system must comply with FAA regulations.

User Interface & Commands

2.1.1.2 The user interface shall display all mission information in a single window on a laptop.
- 2.1.1.2.1 The user interface will display temperature data.
- 2.1.1.2.2 The user interface will display pressure data.
- 2.1.1.2.3 The user interface will display altitude data.
- 2.1.1.2.4 The user interface will display GPS Location.
- 2.1.1.2.5 The user interface will display images taken by the system.

2.1.1.3 The user interface shall be able to restart within a short amount of time.

2.1.1.4 The user interface will have buttons for releasing ribbons in the event of the flight payload failing to do so.

2.1.1.5 The user interface will have a button for releasing the payload.

2.1.1.6 The user interface will have a button for taking an image.

Flight Payload

2.1.1.7 The flight computer will measure data with a fast sampling rate from onboard instruments.

2.1.1.8 The flight computer will transmit data to the laptop.
2.1.1.9 The flight payload shall autonomously unfurl a red ribbon when altitude data reaches 1000 ft.
2.1.1.10 The flight payload shall autonomously unfurl a black/white ribbon when altitude data reaches 2000 ft.
2.1.1.11 The flight payload shall autonomously release the payload when altitude data reaches 3000 ft.
2.1.1.12 The flight payload shall respond to commands from the laptop.
2.1.1.13 The payload shall weigh no more than 4 pounds.
2.1.1.14 A parachute shall automatically deploy following payload separation.

Power

2.1.1.15 The flight payload shall have enough battery life to complete the mission.

2.1.2 System Level Requirements

Regulations and Standards

2.1.2.1 The system shall comply with FAA regulations under Unmanned Free Balloons 14 CFR Part 101.
2.1.2.1.1 The flight payload shall not exceed 4 pounds.
2.1.2.1.2 The flight payload shall have a weight/size ratio not exceeding 3 oz/in^2 on any surface of the payload package.

User Interface

User Display

2.1.2.2 The user interface shall display flight system instrument readings on a laptop within a single window that shows live measurements and graphs data as a function of altitude.
2.1.2.2.1 The user interface shall display flight system temperature data.
2.1.2.2.1.1 The most recent temperature measurement shall be displayed clearly to the user in a table by using at least a 12pt font size and any standard font type.
2.1.2.2.1.2 Temperature data shall be visible to the user in a live graph plotted against altitude data.
2.1.2.2.1.3 The user interface shall update temperature data once every 5 seconds.
2.1.2.2.1.4 Temperature data shall be displayed in units of degrees F.
2.1.2.2.2 The user interface shall display flight system pressure data.
2.1.2.2.2.1 The most recent pressure measurement shall be displayed clearly to the user in a table by using at least a 12pt font size and any standard font type.
2.1.2.2.2.2 Pressure data shall be visible to the user in a live graph plotted against altitude data.

2.1.2.2.2.3 The user interface shall update pressure data once every 5 seconds.

2.1.2.2.2.4 Pressure data shall be displayed in units of pounds per square inch.

2.1.2.2.3 The user interface shall display flight system altitude data.

2.1.2.2.3.1 The most recent altitude measurement shall be displayed clearly to the user in a table by using at least a 12pt font size and any standard font type.

2.1.2.2.3.2 The user interface shall update pressure data every 5 seconds.

2.1.2.2.3.3 Altitude data shall be displayed in units of feet.

2.1.2.2.4 The user interface shall display GPS location.

2.1.2.2.4.1 The most recent GPS measurement shall be displayed clearly to the user in a table by using at least a 12pt font size and any standard font type.

2.1.2.2.4.2 Latitude and Longitude data shall be displayed in degrees N/S and E/W.

2.1.2.2.4.3 The user interface shall update GPS data once every 5 seconds.

2.1.2.2.5 The user interface shall display pictures taken by the flight system.

2.1.2.2.5.1 The pictures shall be updated on the window every 10 seconds.

Interface Functionalities

2.1.2.3 The user interface shall be able to restart within 30 seconds of going offline.

2.1.2.4 The user interface shall have power to run for no less than 2 hours.

2.1.2.5 The user interface shall have controls to manually send commands to the flight system as a redundant safety measure only.

2.1.2.5.1 Commands shall be user friendly by making a single, clearly labeled button for each command.

2.1.2.5.2 One command shall unfurl the flight payload’s red ribbon.

2.1.2.5.2.1 The commands shall have a 2-step confirmation procedure such that the code may not be sent by a single click from the user.

2.1.2.5.3 One command shall unfurl the flight payload’s black/white ribbon.

2.1.2.5.3.1 The commands shall have a 2-step confirmation procedure such that the code may not be sent by a single click from the user.

2.1.2.5.4 One command shall release the payload from the balloon.

2.1.2.5.4.1 The commands shall have a 2-step confirmation procedure such that the code may not be sent by a single click from the user.

Flight Payload

Data Sampling

2.1.2.6 The flight computer shall continuously measure data from onboard instruments.

2.1.2.6.1 The flight computer shall sample altitude data at 0.33 Hz.

2.1.2.6.2 The flight computer shall sample pressure data at 0.33 Hz.

2.1.2.6.3 The flight computer shall sample temperature data at 0.33 Hz.

2.1.2.6.4 The flight computer shall take a picture once per 10 seconds.

2.1.2.7 The flight computer shall save instrument data to an onboard location.

Data Transmission

2.1.2.8 Total data transmission shall not exceed a data rate of 54 Mbps.

2.1.2.9 The flight computer shall transmit altitude measurements to the user interface.

2.1.2.9.1 Altitude data shall not exceed 5 significant figures.

2.1.2.10 The flight computer shall transmit pressure measurements to the user interface.

2.1.2.10.1 Pressure data shall not exceed 5 significant figures.

2.1.2.11 The flight computer shall transmit temperature measurements to the user interface.

2.1.2.11.1 Temperature data shall not exceed 5 significant figures.

2.1.2.12 The flight computer shall transmit GPS location to the user interface.

2.1.2.12.1 GPS data shall not exceed 7 significant figures.

2.1.2.13 The flight computer shall transmit an image once per 10 seconds to the user interface.

Payload Functionalities

2.1.2.14 The flight payload shall autonomously unfurl a red ribbon when altitude data exceeds 990 ft.

2.1.2.14.1 The red ribbon unfurl event response time shall be no more than 1 second.

2.1.2.15 The flight payload shall autonomously unfurl a black/white ribbon when altitude data exceeds 1990 ft.

2.1.2.15.1 The black/white ribbon unfurl event response time shall be no more than 1 second.

2.1.2.16 The flight payload shall autonomously release the payload when altitude data exceeds 2990 ft.

2.1.2.16.1 The payload release event response time shall be no more than 1 second.

2.1.2.17 The flight payload shall receive, process, and execute commands from the user interface as a redundancy safety measure.

2.1.2.17.1 The flight payload shall process and execute a command to unfurl the red ribbon.

2.1.2.17.2 The flight payload shall process and execute a command to unfurl the black/white ribbon.

2.1.2.17.3 The flight payload shall process and execute a command to release from the balloon.

2.1.2.17.4 Commands shall be executed in no more than 5 seconds after being sent.

2.1.2.18 A parachute shall autonomously deploy following payload separation.

2.1.2.19 The payload shall fall no faster than 15 ft/s after separation.

Derived Hardware Constraints

2.1.2.20 The flight software shall be written in C or C++.

2.1.2.21 The flight software shall not exceed 30 GB.

2.1.2.22 The flight computer shall run using 0.8 GB of RAM.

2.1.2.23 The flight system shall not use more than 4 USB ports.

2.1.2.24 The flight system shall not use more than 1 HDMI port.

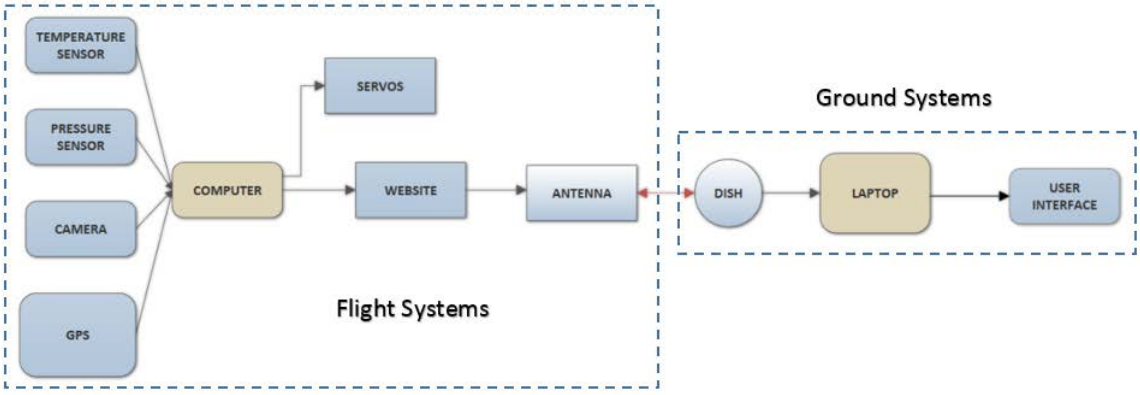
2.1.2.25 The flight system shall transmit data through Wi-Fi.

2.1.2.26 The user interface shall transmit commands through Wi-Fi.

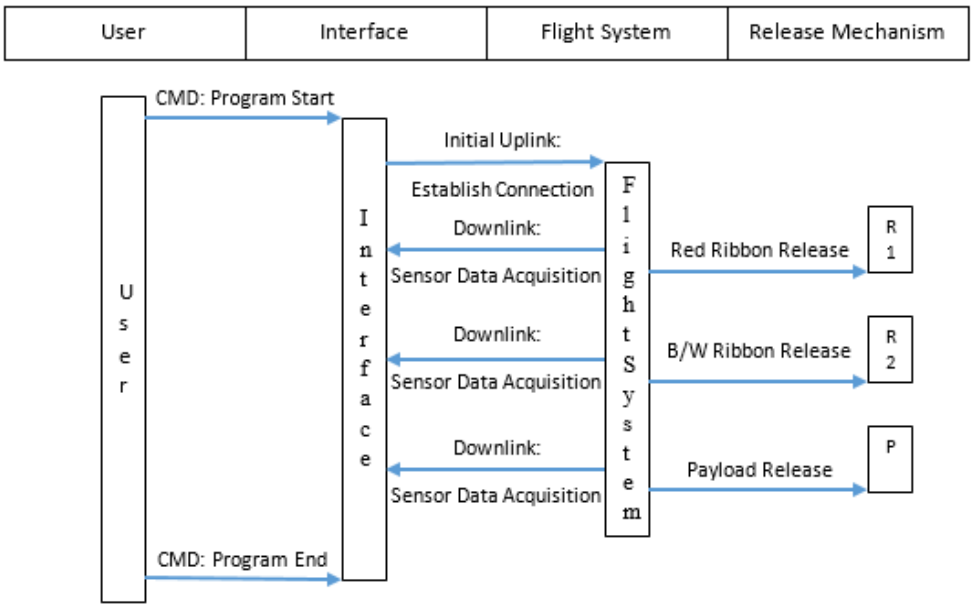
2.1.2.27 Image resolution shall not exceed 5 MP.

2.2 Design

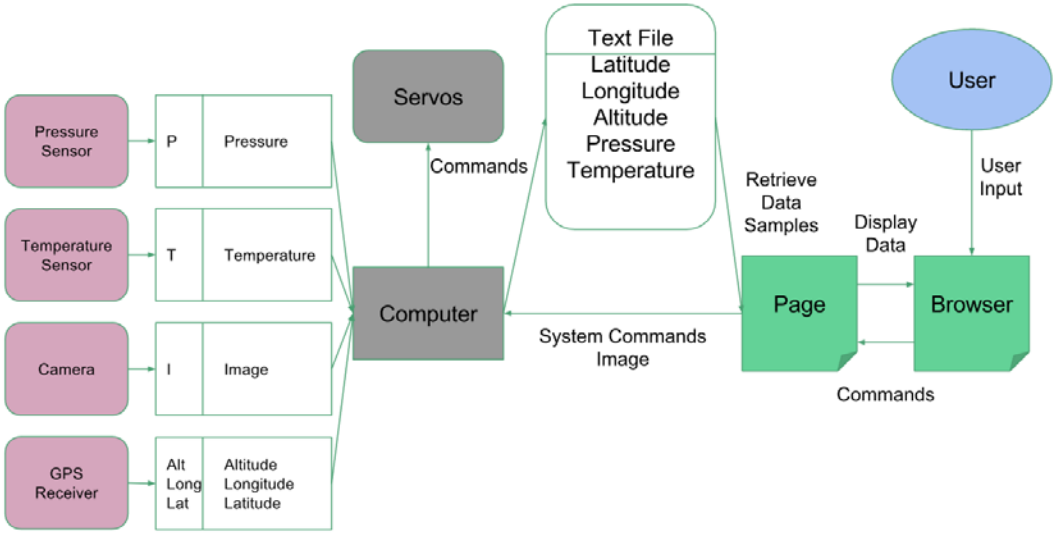
2.2.1 Structural Model



2.2.2 *Sequence Diagram*

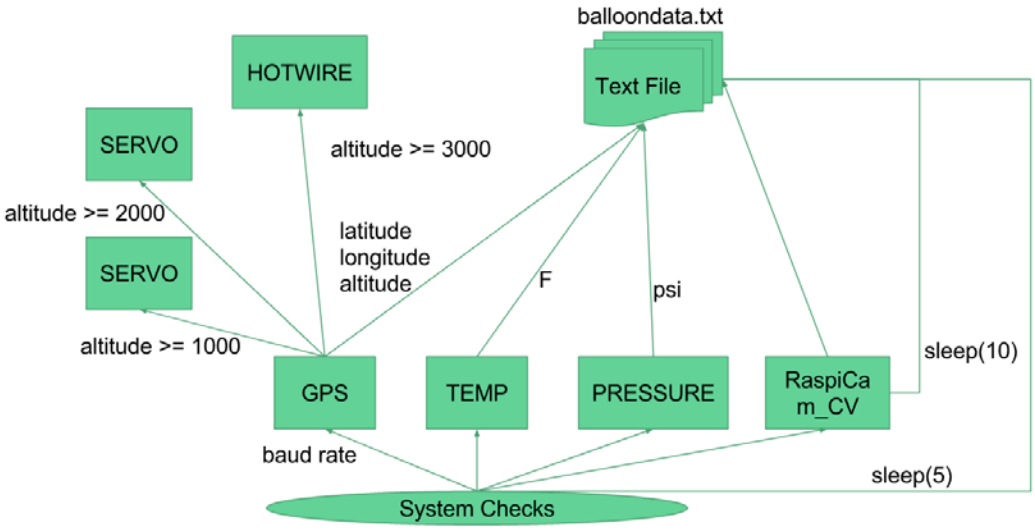


2.2.3 *Data Flow Model*

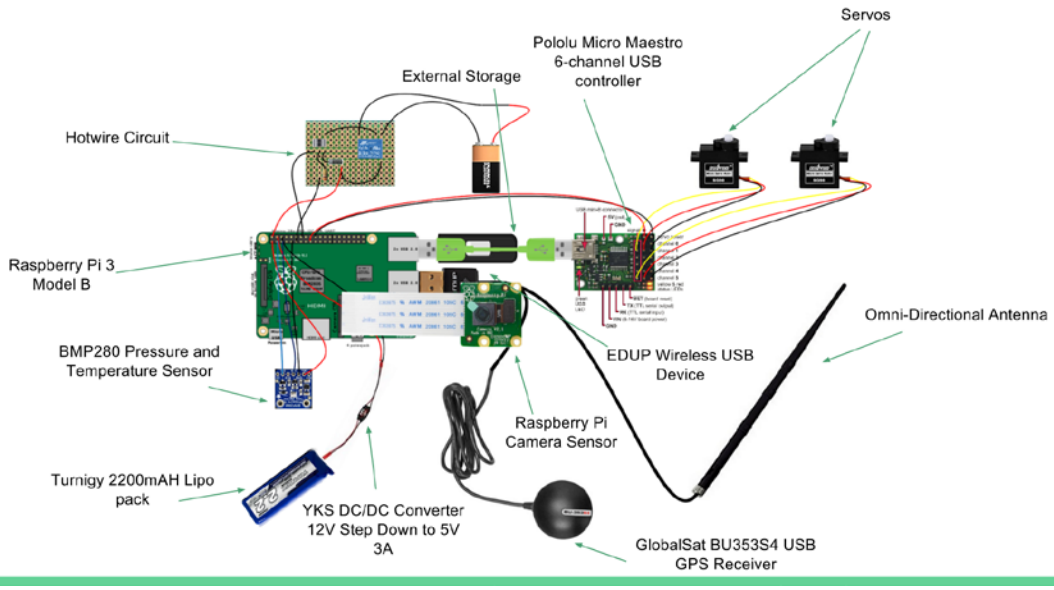


3. Flowcharts

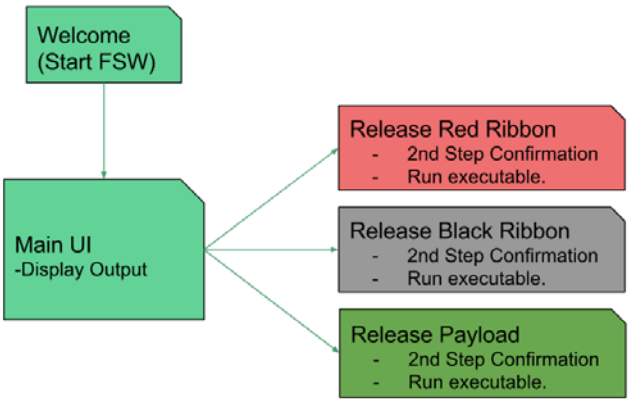
3.1 Class and Function Flow



3.2 Wiring Schematic

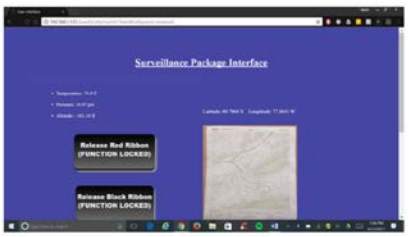


3.3 User Interface

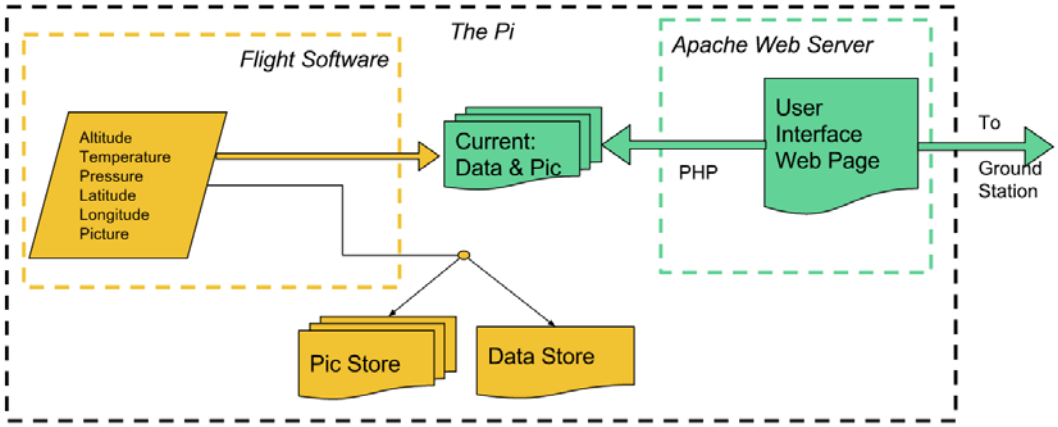


- Implemented utilizing Apache Web Server Software
- Consists of HTML pages and CSS style sheets
- Tasks executed using PHP scripting language

3.4 Final User Interface



3.5 *Database*



4. Source Code

4.1 *Main*

```
#include <iostream>
#include <string>
#include <vector>
#include <iterator>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <cstdio>
#include <ctime>

    //GPS
    #include "gps.h"

    //Camera
    //RaspiCam libraries
    #include <raspicam/raspicam.h>
    #include <raspicam/raspicamtypes.h>
    #include <raspicam/raspicam_cv.h>

    //Servos
    #include "servo.h"
    #include <fcntl.h>

    //Temperature Sensor
    #include "temp.h"
    #include <stdint.h>
```

```
27.
28.      //Pressure Sensor
29.      #include "pressure.h"
30.
31.      //Hotwire
32.      #include "hotwire.h"
33.
34.      //Output
35.      #include <fstream>
36.      #include <iomanip>
37.
38.      using namespace std;
39.
40.      int main()
41.      {
42.          //----------System checks----------
43.              //*****GPS*****
44.                  //Set baud rate for GPS
45.                  system("stty -F /dev/ttyUSB0 4800");
46.                  //Input NMEA line for GPS class
47.                  string nmea;
48.                  fstream f;
49.                  //Open GPS USB device
50.                  f.open("/dev/ttyUSB0");
51.                  //Error output check
52.                  if(f.is_open())
53.                  {
54.                          cout<<"Device Opened..."<<endl;
55.                  }
56.                  else
57.                  {
58.                          cout<<"Unable to open GPS"<<endl;
59.                  }
60.                  //Call GPS class
61.                  GPS gps;
62.
63.              //*****SERVOS*****
64.                  //Open Pololu Micro Maestro 6-channel USB
controller
65.                  const char * device = "/dev/ttyACM0";  // Linux
66.                  int fd = open(device, O_RDWR | O_NOCTTY);
67.                  if (fd == -1)
68.                  {
69.                          perror(device);
70.                          return 1;
71.                  }
72.                  //Call SERVO class
73.                  SERVO servo1000(fd);
74.                  SERVO servo2000(fd);
75.
76.                  // Reset Checks Variables
77.                  bool servo1000DONE = false;
78.                  bool servoZERO = false;
79.                  bool servo2000DONE=false;
```

```
80.                 bool servo2ZERO=false;
81.
82.             //*****PICTURE*****
83.                 //Set count for every other 5 second delay
84.                 int count=0;
85.                 //Call RaspiCam directory classes
86.                 raspicam::RaspiCam_Cv Camera;
87.                 cv::Mat image;
88.
89.                 //Capture count
90.                 int nCount=30;
91.
92.                 //Set camera parameters
93.                 Camera.set(CV_CAP_PROP_FORMAT, CV_8UC1);
94.                 //Open Camera
95.                 cout<<"Opening Camera..."<<endl;
96.                 //Error output check
97.                 if(!Camera.open())
98.                 {
99.                     cerr<<"Error opening camera"<<endl;
100.                }
101.
102.            //*****PRESSURE*****
103.                //Call PRESSURE class
104.                PRESSURE pressure;
105.
106.            //*****TEMPERATURE*****
107.                //Call TEMP class
108.                TEMP temp;
109.
110.            //*****Saved file*****
111.                //Count for pictures saved
112.                int piccnt=1;
113.
114.                //Saved
115.                ofstream balloondataSAVED;
116.                balloondataSAVED.open("/media/pi/98E1-
3336/Data/balloondataSAVED.txt");
117.                //Error output check
118.                if(balloondataSAVED.is_open())
119.                {
120.                        cout<<"Saved data file open..."<<endl;
121.                }
122.                else
123.                {
124.                        cout<<"Error opening saved file"<<endl;
125.                }
126.
127.            //*****HOTWIRE*****
128.                //Call HOTWIRE class
129.                HOTWIRE hotwire;
130.                //False means hotwire is off
131.                bool hotwireON=false;
132.                //Error output check
```

```
133.                cout<<"Hotwire is OFF..."<<endl;
134.
135.    //Loop code and classes while hotwire is not on
136.    //Indicates end of mission
137.    while(hotwireON!=true)
138.    {
139.        //*****Overwritten file*****
140.                ofstream balloondata;
141.                balloondata.open("balloondata.txt");
142.                //Error output check
143.                if(balloondata.is_open())
144.                {
145.                        cout<<"Streaming data file
open..."<<endl;
146.                }
147.                else
148.                {
149.                        cout<<"Error opening streaming
file"<<endl;
150.                }
151.
152.        //----------Data Streaming----------
153.            //**********Latitude, Longitude, Altitude
Data**********
154.                //Set data to bad to start
155.                bool goodGPS=false;
156.                //Loop until a GPGGA point is read with good
data
157.                while(goodGPS!=true)
158.                {
159.                    //Stream data from GPS device
160.                    f>>nmea;
161.                    //Call GPS class
162.                    gps.GPSTest(nmea);
163.                        //Is the data GPGGA data only?
164.                        if(gps.isValidGGA(nmea))
165.                        {
166.                            //Save latitude data to overwritten
file
167.
balloondata<<setprecision(6)<<gps.latitude<<" "<<gps.latc<<endl;
168.                            //Save latitude data to appended
file
169.
balloondataSAVED<<setprecision(6)<<gps.latitude<<"
"<<gps.latc<<endl;
170.
171.                            //Save longitude data to
overwritten file
172.
balloondata<<setprecision(6)<<gps.longitude<<" "<<gps.lonc<<endl;
173.                            //Save longitude data to appended
file
```

```
174.
balloondataSAVED<<setprecision(6)<<gps.longitude<<"
"<<gps.lonc<<endl;
175.
176.                            //Save altitude data to overwritten
file
177.
balloondata<<setprecision(5)<<gps.altitude<<" ft"<<endl;
178.                            //Save altitude data to appended
file
179.
balloondataSAVED<<setprecision(5)<<gps.altitude<<" ft"<<endl;
180.
181.                            //Good data has been recorded, exit
GPS
182.                            goodGPS=true;
183.                        }
184.                }//End while loop - GPS
185.
186.            //**********Temperature Data**********
187.                //Save temperature data to overwritten file
188.                balloondata<<setprecision(3)<<temp.temp_F<<"
F"<<endl;
189.                //Save temperature data to appended file
190.
balloondataSAVED<<setprecision(3)<<temp.temp_F<<" F"<<endl;
191.
192.            //**********Pressure Data**********
193.                //Save pressure data to overwritten file
194.
balloondata<<setprecision(4)<<pressure.pressure_psi<<" psi"<<endl;
195.                //Save temperature data to appended file
196.
balloondataSAVED<<setprecision(4)<<pressure.pressure_psi<<"
psi"<<endl;
197.
198.            //**********Camera Data**********
199.                //Counter to take pictures every 10 seconds,
uses divisor check
200.                if(count%2==0)
201.                {
202.                    //Folder location
203.                    string path = "/media/pi/98E1-3336/Data/";
204.                    //Type of picture
205.                    string picfile = ".jpg";
206.                    //Overall filename for image
207.                    string filename;
208.                    //Incrementing integer for saved images
209.                    string fileint;
210.                    //Link variables
211.                    filename.append(path);
212.
213.                    //Convert integer to string and append
214.                    ostringstream convert;
```

```
215.                    convert<<piccnt;
216.                    fileint=convert.str();
217.                    filename=filename+fileint;
218.                    filename.append(picfile);
219.
220.                    //Convert filename to C string for code
221.                    const char *file=filename.c_str();
222.
223.                    //Start capture
224.                    for(int i=0;i<nCount;i++)
225.                    {
226.                        Camera.grab();
227.                        Camera.retrieve(image);
228.                    }
229.
230.                    //SAVED file name
231.                    cv::imwrite(file,image);
232.                    //OVERWRITTEN file name
233.                    cv::imwrite("image.jpg",image);
234.
235.                    //Increment picture count
236.                    piccnt++;
237.                }
238.                //Wait 10 seconds for new data
239.                count++;
240.
241.            //**********Servos**********
242.                //Altitude = 1000 feet AGL?
243.                if(gps.altitude>=900&&servo1000DONE!=true)
244.                {
245.                    //Call servo class with channel from
Maestro
246.                    //Set max position for servo1000
247.                    servo1000.maestroSetTarget(fd,1,30000);
248.                    //Reset
249.                    servo1000DONE=true;
250.                }
251.
252.                    //Reset servo1000
253.                    if(servo1000DONE==true&&servoZERO!=true)
254.                    {
255.                        sleep(1);
256.                        //Call servo class with channel 1 from
Maestro
257.                        //Reset position to 0
258.                        servo1000.maestroSetTarget(fd,1,0);
259.                        servoZERO=true;
260.                    }
261.
262.                //Altitude = 2000 feet AGL?
263.                if(gps.altitude>=1900&&servo2000DONE!=true)
264.                {
265.                    //Call servo class with channel from
Maestro
```

```
266.                     //Set max position for servo2000
267.                     servo2000.maestroSetTarget(fd,5,30000);
268.                     //Reset
269.                     servo2000DONE=true;
270.                 }
271.
272.                     //Reset servo2000
273.                     if(servo2000DONE==true)
274.                     {
275.                         sleep(1);
276.                         //Call servo class with channel 5 from
Maestro
277.                         //Reset position to 0
278.                         servo2000.maestroSetTarget(fd,5,0);
279.                         servo2ZERO=true;
280.                     }
281.
282.                 //Close port
283.                 close(fd);
284.
285.             //**********Output**********
286.                 //Closing overwritten text file
287.                 balloondata.close();
288.
289.             //**********Hotwire**********
290.                 //Ignite nichrome wire and release mechanism at
AGL >= 3000 ft
291.                     if(gps.altitude>=2900)
292.                     {
293.                         //Fire hotwire circuit for specified
amount of time
294.                         hotwire.HOTWIREFIRE(20);
295.                         //Shut off data streaming
296.                         hotwireON=true;
297.                         //Error output check
298.                         if(hotwireON==true)
299.                         {
300.                             cout<<"Hotwire is ON..."<<endl;
301.                         }
302.                     }
303.
304.             //Only sleep if final class has not been called
305.             if(hotwireON==false)
306.             {
307.                         //Sleep for 3 seconds before looping
308.                         sleep(3);
309.             }
310.
311.     }//End while loop - Mission
312.
313.             //Close appended file
314.         balloondataSAVED.close();
315.
```

```
316.    //-------------------------------------------------------------
----------------
317.    //MISSION ACCOMPLISHED
318.    cout<<"MISSION ACCOMPLISHED"<<endl;
319.
320.        return 0;
321.    }
```

4.2 GPS

4.2.1 Class Definition

```
1. #ifndef GPS_H
2. #define GPS_H
3.
4. #include <vector>
5. #include <string>
6.
7. //Length of minutes data in NMEA
8. #define MINUTE_LENGTH 9
9.
10.     using namespace std;
11.
12.     //Class definition for GPS
13.     class GPS
14.     {
15.     public:
16.         //Default constructor
17.         GPS();
18.         void GPSTest(const string GGASentence);
19.         double  latitude;
20.         double  longitude;
21.         double  altitude;
22.         char latc;
23.         char lonc;
24.         bool isValidGGA(const string GGASentence);
25.         //Destructor
26.         ~GPS();
27.
28.     private:
29.         void setValuesGGA(const string GGASentence);
30.         vector<string> splitStringByComma(const string);
31.         double stringToDouble(const string);
32.         double getCoordinates(string);
33.     };
34.
35.     double degreesToDecimal(const int Degrees,const double
Minutes,const int seconds = 0);
36.
37.     #endif // GPS_H
```

4.2.2 Class Functions

```
1. #include <iostream>
2. #include <string>
```

```
3. #include <vector>
4. #include <assert.h>
5. #include <sstream>
6. #include <stdlib.h>
7. //GPS header file
8. #include "gps.h"
9.
10.      using namespace std;
11.
12.      //Constructor
13.      GPS::GPS()
14.      {
15.          latitude            =   0;
16.          longitude           =   0;
17.          altitude            =   0;
18.      }
19.
20.      //Destructor
21.      GPS::~GPS()
22.      {
23.          latitude            =   0;
24.          longitude           =   0;
25.          altitude            =   0;
26.      }
27.
28.      //GPS::GPS(const string GGASentence)
29.      void GPS::GPSTest(const string GGASentence)
30.      {
31.          //If a GPGGA sentence, set output
32.          if (isValidGGA(GGASentence))
33.              setValuesGGA(GGASentence);
34.          else
35.          {
36.              latitude    =   latitude;
37.              longitude   =   longitude;
38.              altitude    =   altitude;
39.          }
40.      }
41.
42.      //Validation function
43.      bool GPS::isValidGGA(const string GGASentence)
44.      {
45.          //Set data to valid to begin
46.          bool returnBool = true;
47.          //Call split by comma function
48.          vector<string> elementVector =
splitStringByComma(GGASentence);
49.
50.          //Validating GPGGA data
51.              //Is the size of the line not equal to 15?
52.              if (elementVector.size() != 15)
returnBool = false;
53.              else
54.              {
```

```
55.                 //Is header GPGGA?
56.                 if (elementVector[0] != "$GPGGA")
returnBool = false;
57.                 //0 indicates invalid data
58.                 if (elementVector[6] == "0")
returnBool = false;
59.                 //Is the length of the longitude data less than
length of a minute
60.                 string str1;
61.                 str1=elementVector[4];
62.                 if (str1.length() < MINUTE_LENGTH)
returnBool = false;
63.                 //Is the length of the latitude data less than
length of a minute
64.                 string str2;
65.                 str2=elementVector[2];
66.                 if (str2.length() < MINUTE_LENGTH)
returnBool = false;
67.                 if (elementVector[7] == "0")
returnBool = false;
68.             }
69.
70.         return returnBool;
71.     }
72.
73.     //Setter function
74.     void GPS::setValuesGGA(string GGA)
75.     {
76.         //Elevation
77.         int elevation = 1154;   //ft
78.         //STL vector calls split function
79.         vector<string> elementVector = splitStringByComma(GGA);
80.
81.         // Assert we have a GGA sentence
82.         assert(elementVector[0] == "$GPGGA");
83.
84.         //North-South
85.         //Grab latitude from element vector 2
86.         this->latitude           =
getCoordinates(elementVector[2]);
87.
88.         //If direction is 'S', set coordinate to south
89.         if (elementVector[3] == "S")
90.             latc='S';
91.         else
92.             latc='N';
93.
94.         //East - West
95.         //Grab longitude from element vector 4
96.         this->longitude          =
getCoordinates(elementVector[4]);
97.         //If direction is 'W', set coordinate to west
98.         if (elementVector[5] == "W")
99.             lonc='W';
```

```
100.        else
101.            lonc='E';
102.
103.        //Grab altitude and convert to ft with - elevation
above SSL
104.        this->altitude              =
stringToDouble(elementVector[9]);
105.        this->altitude              =   (3.28*(this->altitude)-
elevation);
106.    }
107.
108.    //Split by comma function
109.    vector<string> GPS::splitStringByComma(string input)
110.    {
111.        //Initializing variables
112.        vector<string>  returnVector;
113.        stringstream    ss(input);
114.        string          element;
115.
116.        //Separate data based on comma seperation
117.            while(getline(ss, element, ','))
118.            {
119.                returnVector.push_back(element);
120.            }
121.
122.        return returnVector;
123.    }
124.
125.    //Decimal degrees function
126.    double degreesToDecimal(int degrees, double minutes, int
seconds )
127.    {
128.        //Initializing variables
129.        double returnDouble = 0;
130.            //Calculate coordinate in decimal degrees
131.            returnDouble = degrees + minutes/60 +
seconds/3600.0f;
132.
133.        return returnDouble;
134.    }
135.
136.    //String to double conversion function
137.    double GPS::stringToDouble(string inputString)
138.    {
139.        //If string empty, return 0
140.        double returnValue = 0;
141.            //Convert to string
142.            istringstream istr(inputString);
143.            istr >> returnValue;
144.
145.        return (returnValue);
146.    }
147.
148.    //Coordinate function
```

```
149.    double GPS::getCoordinates(string array)
150.    {
151.        //Initializing variables
152.        double decimalDegrees = 0;
153.        string degreeArray;
154.        string minuteArray;
155.
156.        //Check for length of latitude and longitude and parse
correctly
157.        if (array.length() > MINUTE_LENGTH)
158.        {
159.
160.            degreeArray.assign(array.begin()+1, array.end() -
7);
161.            minuteArray.assign(array.end() - 7, array.end());
162.        }
163.        else
164.        {
165.            degreeArray.assign(array.begin(),array.end()-7);
166.            minuteArray.assign(array.begin()+2,array.end());
167.        }
168.
169.            //Convert strings into numbers
170.            int degrees;
171.            double minutes;
172.            degrees = atoi(degreeArray.c_str());
173.            minutes = stringToDouble(minuteArray);
174.
175.            //Convert degrees and minutes into decimal
176.            decimalDegrees = degreesToDecimal(degrees,minutes);
177.
178.        return decimalDegrees;
179.    }
```

4.3 Temperature

4.3.1 Class Definition

```
1. // Distributed with a free-will license.
2. // Use it any way you want, profit or free, provided it fits in
the licenses of its associated works.
3. // BMP280
4. // This code is designed to work with the BMP280_I2CS I2C Mini
Module available from ControlEverything.com.
5. //
https://www.controleverything.com/content/Barometer?sku=BMP280_I2CSs
#tabs-0-product_tabset-2
6.
7. #ifndef TEMP_H
8. #define TEMP_H
9.
10.     //Class definition for temperature sensor
11.     class TEMP
12.     {
13.     public:
```

```
        //Default constructor
        TEMP();
        double temp_F;
        //Destructor
        ~TEMP();
    private:
        void tempout();
    };

    #endif /* TEMP_H */
```

4.3.2 Class Functions

```
// Distributed with a free-will license.
// Use it any way you want, profit or free, provided it fits in the licenses of its associated works.
// BMP280
// This code is designed to work with the BMP280_I2CS I2C Mini Module available from ControlEverything.com.
// https://www.controleverything.com/content/Barometer?sku=BMP280_I2CSs#tabs-0-product_tabset-2

#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
    #include <sys/ioctl.h>
    #include <unistd.h>
    #include <fcntl.h>
    #include <math.h>
    //Temperature header file
    #include "temp.h"

    using namespace std;

    //Constructor
    TEMP::TEMP()
    {
        tempout();
    };

    void TEMP::tempout()
    {
        //Open i2c bus on GPIO pins
        int file;
        char *bus = "/dev/i2c-1";
            //Error check on opening the bus
        if((file = open(bus, O_RDWR)) < 0)
        {
                printf("Failed to open the bus. \n");
                exit(1);
        }
        //Slave address for i2c device, BMP280 is 0x77(108)
        ioctl(file, I2C_SLAVE, 0x77);

```

```
39.         //Read 24 bytes of data from address(0x88)
40.         char reg[1] = {0x88};
41.         write(file, reg, 1);
42.         char data[24] = {0};
43.             //Error check on reading input data
44.         if(read(file, data, 24) != 24)
45.         {
46.                 printf("Error : Input/output Error \n");
47.                 exit(1);
48.         }
49.
50.         //Convert the data
51.         //Temperature coefficients
52.         int dig_T1 = data[1] * 256 + data[0];
53.         int dig_T2 = data[3] * 256 + data[2];
54.         if(dig_T2 > 32767)
55.         {
56.             dig_T2 -= 65536;
57.         }
58.         int dig_T3 = data[5] * 256 + data[4];
59.         if(dig_T3 > 32767)
60.         {
61.             dig_T3 -= 65536;
62.         }
63.
64.         //Pressure coefficients
65.         int dig_P1 = data[7] * 256 + data[6];
66.         int dig_P2  = data[9] * 256 + data[8];
67.         if(dig_P2 > 32767)
68.         {
69.             dig_P2 -= 65536;
70.         }
71.         int dig_P3 = data[11]* 256 + data[10];
72.         if(dig_P3 > 32767)
73.         {
74.             dig_P3 -= 65536;
75.         }
76.         int dig_P4 = data[13]* 256 + data[12];
77.         if(dig_P4 > 32767)
78.         {
79.             dig_P4 -= 65536;
80.         }
81.         int dig_P5 = data[15]* 256 + data[14];
82.         if(dig_P5 > 32767)
83.         {
84.             dig_P5 -= 65536;
85.         }
86.         int dig_P6 = data[17]* 256 + data[16];
87.         if(dig_P6 > 32767)
88.         {
89.             dig_P6 -= 65536;
90.         }
91.         int dig_P7 = data[19]* 256 + data[18];
92.         if(dig_P7 > 32767)
```

```
93.         {
94.             dig_P7 -= 65536;
95.         }
96.         int dig_P8 = data[21]* 256 + data[20];
97.         if(dig_P8 > 32767)
98.         {
99.             dig_P8 -= 65536;
100.        }
101.        int dig_P9 = data[23]* 256 + data[22];
102.        if(dig_P9 > 32767)
103.        {
104.            dig_P9 -= 65536;
105.        }
106.
107.        //Select control measurement register(0xF4)
108.        //Normal mode, temperature and pressure over sampling
rate = 1(0x27)
109.        char config[2] = {0};
110.        config[0] = 0xF4;
111.        config[1] = 0x27;
112.        write(file, config, 2);
113.
114.        //Select config register(0xF5)
115.        //Stand_by time = 1000 ms(0xA0)
116.        config[0] = 0xF5;
117.        config[1] = 0xA0;
118.        write(file, config, 2);
119.        sleep(1);
120.
121.        //Read 8 bytes of data from register(0xF7)
122.        //pressure msb1, pressure msb, pressure lsb, temp msb1,
temp msb, temp lsb, humidity lsb, humidity msb
123.        reg[0] = 0xF7;
124.        write(file, reg, 1);
125.            //Error check on register data
126.        if(read(file, data, 8) != 8)
127.        {
128.                printf("Error : Input/output Error \n");
129.                exit(1);
130.        }
131.
132.        //Convert pressure and temperature data to 19-bits
133.        long adc_p = (((long)data[0] * 65536) + ((long)data[1]
* 256) + (long)(data[2] & 0xF0)) / 16;
134.        long adc_t = (((long)data[3] * 65536) + ((long)data[4]
* 256) + (long)(data[5] & 0xF0)) / 16;
135.
136.        //Temperature offset calculations
137.        double var1 = (((double)adc_t) / 16384.0 -
((double)dig_T1) / 1024.0) * ((double)dig_T2);
138.        double var2 = ((((double)adc_t) / 131072.0 -
((double)dig_T1) / 8192.0) *(((double)adc_t)/131072.0 -
((double)dig_T1)/8192.0)) * ((double)dig_T3);
139.        double t_fine = (long)(var1 + var2);
```

```
        double cTemp = (var1 + var2) / 5120.0;
        double fTemp = cTemp * 1.8 + 32;
            this->temp_F=fTemp;

        //Pressure offset calculations
        var1 = ((double)t_fine / 2.0) - 64000.0;
        var2 = var1 * var1 * ((double)dig_P6) / 32768.0;
        var2 = var2 + var1 * ((double)dig_P5) * 2.0;
        var2 = (var2 / 4.0) + (((double)dig_P4) * 65536.0);
        var1 = (((double) dig_P3) * var1 * var1 / 524288.0 +
((double) dig_P2) * var1) / 524288.0;
        var1 = (1.0 + var1 / 32768.0) * ((double)dig_P1);
        double p = 1048576.0 - (double)adc_p;
        p = (p - (var2 / 4096.0)) * 6250.0 / var1;
        var1 = ((double) dig_P9) * p * p / 2147483648.0;
        var2 = p * ((double) dig_P8) / 32768.0;
        double pressure = (p + (var1 + var2 + ((double)dig_P7))
/ 16.0) / 100;
    };

    //Destructor
    TEMP::~TEMP()
    {

    };
```

4.4 Pressure

4.4.1 Class Definition

```
// Distributed with a free-will license.
// Use it any way you want, profit or free, provided it fits in
the licenses of its associated works.
// BMP280
// This code is designed to work with the BMP280_I2CS I2C Mini
Module available from ControlEverything.com.
//
https://www.controleverything.com/content/Barometer?sku=BMP280_I2CSs
#tabs-0-product_tabset-2

#ifndef PRESSURE_H
#define PRESSURE_H

    //Class definition for pressure sensor
    class PRESSURE
    {
    public:
        //Default constructor
        PRESSURE();
        double pressure_psi;
        //Destructor
        ~PRESSURE();
    private:
        void pressureout();
    };
```

```
22.
23.     #endif /* PRESSURE_H */
```

4.4.2 Class Functions

```
1. // Distributed with a free-will license.
2. // Use it any way you want, profit or free, provided it fits in
the licenses of its associated works.
3. // BMP280
4. // This code is designed to work with the BMP280_I2CS I2C Mini
Module available from ControlEverything.com.
5. //
https://www.controleverything.com/content/Barometer?sku=BMP280_I2CSs
#tabs-0-product_tabset-2
6.
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <linux/i2c-dev.h>
10.     #include <sys/ioctl.h>
11.     #include <unistd.h>
12.     #include <fcntl.h>
13.     #include <math.h>
14.     //Pressure header file
15.     #include "pressure.h"
16.
17.     using namespace std;
18.
19.     //Constructor
20.     PRESSURE::PRESSURE()
21.     {
22.         pressureout();
23.     };
24.
25.     void PRESSURE::pressureout()
26.     {
27.         //Open i2c bus on GPIO pins
28.         int file;
29.         char *bus = "/dev/i2c-1";
30.             //Error check on opening the bus
31.         if((file = open(bus, O_RDWR)) < 0)
32.         {
33.                 printf("Failed to open the bus. \n");
34.                 exit(1);
35.         }
36.         //Slave address for i2c device, BMP280 is 0x77(108)
37.         ioctl(file, I2C_SLAVE, 0x77);
38.
39.         //Read 24 bytes of data from address(0x88)
40.         char reg[1] = {0x88};
41.         write(file, reg, 1);
42.         char data[24] = {0};
43.             //Error check on reading input data
44.         if(read(file, data, 24) != 24)
45.         {
46.                 printf("Error : Input/output Error \n");
```

```
                exit(1);
        }

        //Convert the data
        //Temperature coefficients
        int dig_T1 = data[1] * 256 + data[0];
        int dig_T2 = data[3] * 256 + data[2];
        if(dig_T2 > 32767)
        {
            dig_T2 -= 65536;
        }
        int dig_T3 = data[5] * 256 + data[4];
        if(dig_T3 > 32767)
        {
            dig_T3 -= 65536;
        }

        //Pressure coefficients
        int dig_P1 = data[7] * 256 + data[6];
        int dig_P2  = data[9] * 256 + data[8];
        if(dig_P2 > 32767)
        {
            dig_P2 -= 65536;
        }
        int dig_P3 = data[11]* 256 + data[10];
        if(dig_P3 > 32767)
        {
            dig_P3 -= 65536;
        }
        int dig_P4 = data[13]* 256 + data[12];
        if(dig_P4 > 32767)
        {
            dig_P4 -= 65536;
        }
        int dig_P5 = data[15]* 256 + data[14];
        if(dig_P5 > 32767)
        {
            dig_P5 -= 65536;
        }
        int dig_P6 = data[17]* 256 + data[16];
        if(dig_P6 > 32767)
        {
            dig_P6 -= 65536;
        }
        int dig_P7 = data[19]* 256 + data[18];
        if(dig_P7 > 32767)
        {
            dig_P7 -= 65536;
        }
        int dig_P8 = data[21]* 256 + data[20];
        if(dig_P8 > 32767)
        {
            dig_P8 -= 65536;
        }
```

```
        int dig_P9 = data[23]* 256 + data[22];
        if(dig_P9 > 32767)
        {
            dig_P9 -= 65536;
        }

        //Select control measurement register(0xF4)
        //Normal mode, temperature and pressure over sampling
rate = 1(0x27)
        char config[2] = {0};
        config[0] = 0xF4;
        config[1] = 0x27;
        write(file, config, 2);

        //Select config register(0xF5)
        //Stand_by time = 1000 ms(0xA0)
        config[0] = 0xF5;
        config[1] = 0xA0;
        write(file, config, 2);
        sleep(1);

        //Read 8 bytes of data from register(0xF7)
        //pressure msb1, pressure msb, pressure lsb, temp msb1,
temp msb, temp lsb, humidity lsb, humidity msb
        reg[0] = 0xF7;
        write(file, reg, 1);
            //Error check on register data
        if(read(file, data, 8) != 8)
        {
                printf("Error : Input/output Error \n");
                exit(1);
        }

        //Convert pressure and temperature data to 19-bits
        long adc_p = (((long)data[0] * 65536) + ((long)data[1]
* 256) + (long)(data[2] & 0xF0)) / 16;
        long adc_t = (((long)data[3] * 65536) + ((long)data[4]
* 256) + (long)(data[5] & 0xF0)) / 16;

        //Temperature offset calculations
        double var1 = (((double)adc_t) / 16384.0 -
((double)dig_T1) / 1024.0) * ((double)dig_T2);
        double var2 = ((((double)adc_t) / 131072.0 -
((double)dig_T1) / 8192.0) *(((double)adc_t)/131072.0 -
((double)dig_T1)/8192.0)) * ((double)dig_T3);
        double t_fine = (long)(var1 + var2);
        double cTemp = (var1 + var2) / 5120.0;
        double fTemp = cTemp * 1.8 + 32;

        //Pressure offset calculations
            double hpapsiconversion=.01450377377302;
        var1 = ((double)t_fine / 2.0) - 64000.0;
        var2 = var1 * var1 * ((double)dig_P6) / 32768.0;
        var2 = var2 + var1 * ((double)dig_P5) * 2.0;
```

```
148.         var2 = (var2 / 4.0) + (((double)dig_P4) * 65536.0);
149.         var1 = (((double) dig_P3) * var1 * var1 / 524288.0 +
((double) dig_P2) * var1) / 524288.0;
150.         var1 = (1.0 + var1 / 32768.0) * ((double)dig_P1);
151.         double p = 1048576.0 - (double)adc_p;
152.         p = (p - (var2 / 4096.0)) * 6250.0 / var1;
153.         var1 = ((double) dig_P9) * p * p / 2147483648.0;
154.         var2 = p * ((double) dig_P8) / 32768.0;
155.         double pressure = (p + (var1 + var2 + ((double)dig_P7))
/ 16.0) / 100;
156.         this->pressure_psi = pressure * hpapsiconversion;
157.     };
158.
159.     //Destructor
160.     PRESSURE::~PRESSURE()
161.     {
162.
163.     };
```

4.5 Servos

4.5.1 Class Definition

```
1. #ifndef SERVO_H
2. #define SERVO_H
3.
4. //Class definition for servo motors
5. class SERVO
6. {
7. public:
8.     //Default constructor
9.     SERVO(int fd);
10.          int maestroGetPosition(int fd, unsigned char channel);
11.          int maestroSetTarget(int fd, unsigned char channel,
unsigned short target);
12.          //Destructor
13.          ~SERVO();
14.     };
15.
16.     #endif /* SERVO_H */
```

4.5.2 Class Functions

```
1. #include <iostream>
2. #include <fcntl.h>
3. #include <stdio.h>
4. #include <unistd.h>
5. //Servo header file
6. #include "servo.h"
7.
8. using namespace std;
9.
10.     //C code
11.     #ifdef _WIN32
12.     #define O_NOCTTY 0
```

```
13.     #else
14.     #include <termios.h>
15.     #endif
16.
17.     //Constructor
18.     SERVO::SERVO(int fd)
19.     {
20.         //C code
21.         #ifdef _WIN32
22.             _setmode(fd, _O_BINARY);
23.         #else
24.             struct termios options;
25.                 tcgetattr(fd, &options);
26.                 options.c_iflag &= ~(INLCR | IGNCR | ICRNL |
IXON | IXOFF);
27.                 options.c_oflag &= ~(ONLCR | OCRNL);
28.                 options.c_lflag &= ~(ECHO | ECHONL | ICANON |
ISIG | IEXTEN);
29.                 tcsetattr(fd, TCSANOW, &options);
30.         #endif
31.     };
32.
33.     //Destructor
34.     SERVO::~SERVO()
35.     {
36.
37.     };
38.
39.     //Position function
40.     int SERVO::maestroGetPosition(int fd, unsigned char
channel)
41.     {
42.         //Find the channel the servo is connected to
43.         unsigned char command[] = {0x90, channel};
44.         //Error check for channel
45.         if(write(fd, command, sizeof(command)) == -1)
46.         {
47.             perror("error writing");
48.             return -1;
49.         }
50.
51.         //Response check for channel
52.         unsigned char response[2];
53.         if(read(fd,response,2) != 2)
54.         {
55.             perror("error reading");
56.             return -1;
57.         }
58.
59.         //Return position of the servo
60.         return response[0] + 256*response[1];
61.     }
62.
63.     //Target function
```

```
64.     int SERVO::maestroSetTarget(int fd, unsigned char channel,
unsigned short target)
65.     {
66.         //Declare position
67.         int position = maestroGetPosition(fd, channel);
68.         //Output current position
69.         printf("Current position is %d.\n", position);
70.         //Assign command to move on servo channel
71.         unsigned char command[] = {0x84, channel, target &
0x7F, target >> 7 & 0x7F};
72.         //Error check for channel
73.         if (write(fd, command, sizeof(command)) == -1)
74.         {
75.             perror("error writing");
76.             return -1;
77.         }
78.
79.       return 0;
80.     }
```

4.6 Hotwire

4.6.1 Class Definition

```
1. #ifndef HOTWIRE_H
2. #define HOTWIRE_H
3.
4. class HOTWIRE
5. {
6. public:
7.     //Default constructor
8.     HOTWIRE();
9.     void HOTWIREFIRE(int time);
10.         //Destructor
11.         ~HOTWIRE();
12.     };
13.
14.     #endif /* HOTWIRE_H */
```

4.6.2 Class Functions

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <stdint.h>
4. #include <unistd.h>
5. //WiringPi libraries
6. #include <wiringPi/wiringPi.h>
7. //Hotwire header file
8. #include "hotwire.h"
9.
10.     //Define output GPIO pin
11.     #define GPIO        17
12.
13.     //Constructor
14.     HOTWIRE::HOTWIRE()
```

```
15.     {
16.         //Configure Pi to use GPIO pin outputs
17.         wiringPiSetupGpio();
18.     };
19.
20.     void HOTWIRE::HOTWIREFIRE(int time)
21.     {
22.         //Set pinmode
23.         pinMode(GPIO, OUTPUT);
24.         //Write high = 3.3V from the GPIO pin
25.         digitalWrite(GPIO, HIGH);
26.         //Wait (seconds)
27.         sleep(time);
28.         //Write low = 0 V from the GPIO pin
29.         digitalWrite(GPIO, LOW);
30.     };
31.
32.     //Destructor
33.     HOTWIRE::~HOTWIRE()
34.     {
35.
36.     };
```

4.7 Red Ribbon Button

```
1. #include <iostream>
2. #include <string>
3. #include <vector>
4. #include <iterator>
5. #include <stdlib.h>
6. #include <stdio.h>
7. #include <unistd.h>
8. #include <cstdio>
9. #include <ctime>
10.
11.     //Servos
12.     #include "servo.h"
13.     #include <fcntl.h>
14.
15.     using namespace std;
16.
17.     int main()
18.     {
19.         //Call servo class with channel from Maestro
20.         //Set max position for servo1000
21.             SERVO servo1000(fd,1,9600);
22.
23.     //-------------------------------------------------------------
----------------
24.     //Red Ribbon Redundancy Measure Initiated
25.     cout<<"Red Redunancy Initiated..."<<endl;
26.
27.         return 0;
28.     }
```

4.8 Black and White Ribbon Button

```
1. #include <iostream>
2. #include <string>
3. #include <vector>
4. #include <iterator>
5. #include <stdlib.h>
6. #include <stdio.h>
7. #include <unistd.h>
8. #include <cstdio>
9. #include <ctime>
10.
11.      //Servos
12.      #include "servo.h"
13.      #include <fcntl.h>
14.
15.      using namespace std;
16.
17.      int main()
18.      {
19.          //Call servo class with channel from Maestro
20.          //Set max position for servo2000
21.              SERVO servo2000(fd,5,9600);
22.
23.      //-------------------------------------------------------------
-----------------
24.      //Black and White Ribbon Redundancy Measure Initiated
25.      cout<<"Black and White Redunancy Initiated..."<<endl;
26.
27.          return 0;
28.      }
```

4.9 Hotwire Button

```
1. #include <iostream>
2. #include <string>
3. #include <vector>
4. #include <iterator>
5. #include <stdlib.h>
6. #include <stdio.h>
7. #include <unistd.h>
8. #include <cstdio>
9. #include <ctime>
10.
11.      //Hotwire
12.      #include "hotwire.h"
13.
14.      using namespace std;
15.
16.      int main()
17.      {
18.          //Call HOTWIRE class for specified number on seconds
19.              HOTWIRE hotwire(20);
20.
```

```
21.     //-----------------------------------------------------------
----------------
22.     //Hotwire Redunancy Measure Initiated
23.     cout<<"Hotwire Redunancy Initiated..."<<endl;
24.
25.         return 0;
26.     }
```

4.10 User Interface

4.10.1 PHP

4.10.1.1 UI

```
1. <!doctype html>
2. <html>
3. <head>
4. <!--Header (note: page refreshes every 5 sec) -->
5. <meta charset="utf-8">
6. <meta http-equiv="refresh" content="5" />
7. <title>User Interface</title>
8. <link rel="stylesheet" href="style.css">
9. </head>
10.
11.     <body style="background-color:navy;">
12.
13.     <!--********************** Column 1
********************** -->
14.
15.
16.     <div>
17.         <h1><center><u><font color="white">Surveillance Package
Interface</font></u></center></h1>
18.     </div>
19.
20.     <div class="content col-1-2">
21.     <!-- /// Temp, Pres, and Alt reader block - start /// -->
22.     <?php
23.         //Load DATA file
24.         $filename = "balloondata.txt";
25.         $fileContents = file_get_contents($filename);
26.         $data = explode("\n", $fileContents);
27.     ?>
28.
29.     <div>
30.       <ul><font color="white">
31.         <li>Temperature: <?php   echo $data[3]   ?> </li> <!--
temperature input from cpp code -->
32.         <br>
33.         <li>Pressure: <?php echo $data[4]   ?></li> <!--
pressure input from cpp code -->
34.         <br>
35.         <li>Altitude: <?php echo $data[2] ?></li> <!-- altitude
input from cpp code -->
36.           </font>
37.       </ul>
```

```
38.      </div>
39.      <!--/// End /// -->
40.
41.
42.      <!-- /// Ribbon buttons - start /// -->
43.      <div>
44.          <form action="servo.php" method="post">
45.          <input type="hidden" name="color" value="red" />
46.          <input type="hidden" name="exec" value="request" />
47.          <input type="submit" id="button1" value="Release Red
Ribbon" />
48.          </form>
49.
50.          <form action="servo.php" method="post">
51.          <input type="hidden" name="color" value="black" />
52.          <input type="hidden" name="exec" value="request" />
53.          <input type="submit"  id="button2" value="Release Black
Ribbon"/>
54.          </form>
55.      </div>
56.      <!--/// End /// -->
57.
58.      <br>
59.      <br>
60.      <br>
61.      <h1><center>
62.        <font color="white">==========================</font>
63.      </center></h1>
64.
65.      <!-- /// Payload Button - start /// -->
66.      <div>
67.          <form action="payload.php">
68.          <input type="hidden" name="exec" value="request" />
69.          <input type="submit" id="button3" value="Release
Payload"/>
70.          </form>
71.      </div>
72.      <!--/// End /// -->
73.      </div>
74.
75.
76.      <!--********************** Column 2
********************** -->
77.
78.      <br>
79.      <br>
80.      <br>
81.      <br>
82.      <br>
83.      <br>
84.
85.      <div>
86.      <!-- /// Latitude and Longtitude block - start /// -->
87.      <div class="wrapper" >
```

```
88.      <font color="white">
89.          <div id="latitude">
90.              Latitude: <?php echo $data[0] ?> <!-- Latitude
input from cpp code -->
91.          </div>
92.          <div id="longtitude">
93.              Longtitude: <?php echo $data[1] ?> <!-- Longtitude
input from cpp code -->
94.          </div>
95.          </font>
96.      </div>
97.      <!--/// End /// -->
98.
99.      <br>
100.     <br>
101.     <br>
102.
103.     <!-- /// placeholder 1 - start /// -->
104.     <div>
105.     <img src="placeholder1.jpg">
106.     </div>
107.     <!--/// End /// -->
108.
109.
110.     <!-- /// Current Image - start /// -->
111.     <div class="content">
112.     <img src="image.jpg">
113.     </div>
114.     <!--/// End /// -->
115.     </div>
116.
117.
118.
119.
120.     </body>
121.
122.
123.     </html>
```

4.10.1.2 Guest UI

```
1.<!doctype html>
2.<html>
3.<head>
4.<!--Header (note: page refreshes every 5 sec) -->
5.<meta charset="utf-8">
6.<meta http-equiv="refresh" content="5" />
7.<title>User Interface</title>
8.<link rel="stylesheet" href="Gueststyle.css">
9.</head>
10.
11.      <body style="background-color:navy;">
12.
13.      <!--********************** Column 1
********************** -->
```

```
14.
15.
16.     <div>
17.         <h1><center><u><font color="white">Surveillance Package
Interface</font></u></center></h1>
18.     </div>
19.
20.     <div class="content col-1-2">
21.     <!-- /// Temp, Pres, and Alt reader block - start /// -->
22.     <?php
23.         //Load DATA file
24.         $filename = "/home/timothy/Documents/testResource.txt";
25.         $fileContents = file_get_contents($filename);
26.         $data = explode("\n", $fileContents);
27.     ?>
28.
29.     <div>
30.       <ul><font color="white">
31.         <li>Temperature (&deg;C): <?php   echo $data[0]   ?>
</li> <!-- temperature input from cpp code -->
32.         <br>
33.         <li>Pressure (psi): <?php echo $data[1] ?></li> <!--
pressure input from cpp code -->
34.         <br>
35.         <li>Altitude (ft): <?php echo $data[2] ?></li> <!--
altitude input from cpp code -->
36.           </font>
37.       </ul>
38.     </div>
39.     <!--/// End /// -->
40.
41.
42.     <!-- /// Ribbon buttons - start /// -->
43.     <div>
44.             <span id="button1">Release Red Ribbon<br>(FUNCTION
LOCKED)</font></span>
45.             <span id="button2">Release Black
Ribbon<br>(FUNCTION LOCKED)</font></span>
46.     </div>
47.     <!--/// End /// -->
48.
49.     <br>
50.     <br>
51.     <br>
52.     <h1><center>
53.       <font color="white">==========================</font>
54.     </center></h1>
55.
56.     <!-- /// Payload Button - start /// -->
57.     <div>
58.         <span id="button3">Release Payload<br>(FUNCTION
LOCKED)</span>
59.     </div>
60.     <!--/// End /// -->
```

```
61.     </div>
62.
63.
64.     <!--************************ Column 2
********************** -->
65.
66.     <br>
67.     <br>
68.     <br>
69.     <br>
70.     <br>
71.     <br>
72.
73.     <div>
74.     <!-- /// Latitude and Longtitude block - start /// -->
75.     <div class="wrapper" >
76.     <font color="white">
77.         <div id="latitude">
78.             Latitude (&deg;N): <?php echo $data[3] ?> <!--
Latitude input from cpp code -->
79.         </div>
80.         <div id="longtitude">
81.             Longtitude (&deg;W): <?php echo $data[4] ?> <!--
Longtitude input from cpp code -->
82.         </div>
83.         </font>
84.     </div>
85.     <!--/// End /// -->
86.
87.     <br>
88.     <br>
89.     <br>
90.
91.     <!-- /// placeholder 1 - start /// -->
92.     <div>
93.     <img src="placeholder1.jpg">
94.     </div>
95.     <!--/// End /// -->
96.
97.
98.     <!-- /// placeholder 2 - start /// -->
99.     <div class="content">
100.    <img src="placeholder2.jpg">
101.    </div>
102.    <!--/// End /// -->
103.    </div>
104.
105.
106.
107.
108.    </body>
109.
110.
111.    </html>
```

4.10.1.3 Servo Buttons

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Servo Release</title>
</head>
<body>
<!-- Html Display regardless of where in the process -->
<h1>Ribbon Release Confirmation</h1>
    <hr>

    <!-- ///Check if this is initial action call or 2nd
confirmation-->
    <?php if ('request' == $_POST['exec']) {
        // Ask user to confirm they want to release ribbon.
    ?>
    <!-- ///Optional html display if previous condition met. --
>

    <p>Are you sure you want to release <?php echo
$_POST['color']; ?> ribbon?</p>
    <form action="servo.php" method="post">
        <!-- /// Handle confirm action by resubmitting this
page -->
            <input type="hidden" value="run" name="exec" />
        <input type="hidden" name="color" value=<?php echo
$_POST['color']; ?> />
        <input type="submit" value="Yes" />
    </form>
    <form action="UI.php" method="post">
        <!-- ///Handle cancel action by returning to UI page --
>
        <input type="hidden" value="cancel" name="exec" />
        <input type="submit" value="No" />
    </form>
    <?php
    } elseif ('run' == $_POST['exec']) {
        // Command Confirmed, run servo script.
        // Select which color.
        if ('red' == $_POST['color']) {
            // Red Ribbon
            system("./redbutton 2>&1");
        } else {
            // Black White Ribbon
            system("./blackwhitebutton 2>&1");
        }
    } else {
        // Assume Command is not confirmed/rescinded, return to
UI, or some other error.
        // TEMP FIX - header('Location: 192.168.1.131/UI.php');
        header('192.168.1.131/UI.php');
        exit;
    }
```

```
47.      ?>
48.      </body>
49.      </html>
```

4.10.1.4 Hotwire Button

```
1.<!doctype html>
2.<html>
3.<head>
4.<meta charset="utf-8">
5.<title>Servo Release</title>
6.</head>
7.<body>
8.<!-- ///Html Display regardless of where in the process -->
9.<h1>Payload Release Confirmation</h1>
10.      <hr>
11.
12.      <!-- ///Check if this is initial action call or 2nd
confirmation-->
13.      <?php if ('request' == $_POST['exec']) {
14.          // Ask user to confirm they want to release ribbon.
15.      ?>
16.      <!-- ///Optional html display if previous condition met. --
>
17.
18.      <p>Are you sure you want to release payload?</p>
19.      <form action="payload.php" method="post">
20.          <!-- /// Handle confirm action by resubmitting this
page -->
21.              <input type="hidden" value="run" name="exec" />
22.          <input type="submit" value="Yes" />
23.      </form>
24.      <form action="UI.php" method="post">
25.          <!-- ///Handle cancel action by returning to UI page --
>
26.          <input type="hidden" value="cancel" name="exec" />
27.          <input type="submit" value="No" />
28.      </form>
29.      <?php
30.      } elseif ('run' == $_POST['exec']) {
31.          // Command Confirmed, run payload script.
32.          system("./hotwire 2>&1");
33.          }
34.      } else {
35.          // Assume Command is not confirmed/rescinded, return to
UI, or some other error.
36.          // TEMP FIX - header('Location: 192.168.1.131/UI.php');
37.          header('192.168.1.131/UI.php');
38.          exit;
39.      }
40.      ?>
41.      </body>
42.      </html>
```

4.10.2CSS

4.10.2.1 Style

```
.wrapper {
    width:600px;
    margin: 0 auto;
}

#latitude {
    float:left;
    font-size:20px;
}

    #longtitude {
        float:right;
        font-size:20px;
    }

    #contentWrapper{
      width: 90%;
        margin: 0 auto;
        text-align: center;
    }
    #contents{
        text-align:center;
    }

    #button1{
      font-family: "Arial Black", Gadget, sans-serif;
        text-align:center;
        margin: 50px auto;
        padding: 20px 0px;
        display:block;
        height:70px;
        width:250px;

        cursor:pointer;
        font-size:20px;
        color:#F0F0F0;
        border-radius:14px;
        border-bottom:4px solid #230001;
        box-shadow: 6px 6px 6px #999;

        -webkit-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        -o-user-select: none;
        user-select: none;

        background: -webkit-linear-gradient(#d30000, #ff6a92);
        background: -o-linear-gradient(#d30000, #ff6a92);
        background: -moz-linear-gradient(#d30000, #ff6a92);
        background: linear-gradient(#d30000, #ff6a92);

```

```
        -webkit-transition: background 0.05s linear;
        -moz-transition: background 0.05s linear;
        -o-transition: background 0.05s linear;
        transition: background 0.05s linear;

        background-size:1px 200px;

        transition: all .05s linear;
    }

    #button1:hover{
        background-position:100px;
    }

    #button1:active {
        box-shadow: 2px 2px 2px #777;
        border-bottom:1px solid #230001;
        transform: translateY(3px);
    }

    #button2{
      font-family: "Arial Black", Gadget, sans-serif;
        text-align:center;
        margin:20px auto;
        padding: 20px 0px;
        display:block;
        height:70px;
        width:260px;

        cursor:pointer;
        font-size:20px;
        color:#F0F0F0;
        border-radius:14px;
        border-bottom:4px solid #3f3f3f;
        box-shadow: 6px 6px 6px #999;

        -webkit-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        -o-user-select: none;
        user-select: none;

        background: -webkit-linear-gradient(#000000, #828282);
        background: -o-linear-gradient(#000000, #828282);
        background: -moz-linear-gradient(#000000, #828282);
        background: linear-gradient(#000000, #828282);

        -webkit-transition: background 0.05s linear;
        -moz-transition: background 0.05s linear;
        -o-transition: background 0.05s linear;
        transition: background 0.05s linear;

        background-size:1px 200px;

```

```
        transition: all .05s linear;
    }

    #button2:hover{
        background-position:100px;
    }

    #button2:active {
        box-shadow: 2px 2px 2px #777;
        border-bottom:1px solid #3f3f3f;
        transform: translateY(3px);
    }

    #button3{
      font-family: Impact, Charcoal, sans-serif;
        text-align:center;
        margin:90px auto;
        padding: 20px 0px;
        display:block;
        height:70px;
        width:260px;

        cursor:pointer;
        font-size:22px;
        color:#F0F0F0;
        border-radius:14px;
        border-bottom:4px solid #00470a;
        box-shadow: 6px 6px 6px #999;

        -webkit-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        -o-user-select: none;
        user-select: none;

        background: -webkit-linear-gradient(#156b22, #2ed347);
        background: -o-linear-gradient(#156b22, #2ed347);
        background: -moz-linear-gradient(#156b22, #2ed347);
        background: linear-gradient(#156b22, #2ed347);

        -webkit-transition: background 0.05s linear;
        -moz-transition: background 0.05s linear;
        -o-transition: background 0.05s linear;
        transition: background 0.05s linear;

        background-size:1px 200px;

        transition: all .05s linear;
    }

    #button3:hover{
        background-position:100px;
    }

```

```
    #button3:active {
        box-shadow: 2px 2px 2px #777;
        border-bottom:1px solid #00470a;
        transform: translateY(3px);
    }


    body{padding:5%;}
    .content{
      padding:3% 5%;
      margin-bottom:3%;
      box-shadow:0px 2px 2px rgba(0,0,0,0.1);
      border-radius:3px;
      color:black;
      border-top:1px solid rgba(255,255,255,0.15);
      text-shadow: 0px 1px 0px rgba(0,0,0,0.15);
      font-size:20px;
    }
    .content:after{
      content:' ';
      background:rgba(255,255,255,0.15);
      width:100%;
      height:100%;
      display:block;
      position: absolute;
      top:0px;
      left:0px;
      border-radius:3px;
      z-index:1;
    }

    .content *{z-index:2;position:relative;}
    .col-1-2{width:40.5%;float:left;}
    .col-1-2 + .col-1-2{margin-left:1%;}
    .clear{clear:both;}

    .title {
        font-family:"Arial Black", Gadget, sans-serif;
        color: white;
    }
```

4.10.2.2 Guest Style

```
.wrapper {
    width:600px;
    margin: 0 auto;
}

#latitude {
    float:left;
}

    #longtitude {
        float:right;
    }
```

```
13.
14.     #contentWrapper{
15.       width: 90%;
16.         margin: 0 auto;
17.         text-align: center;
18.     }
19.     #contents{
20.         text-align:center;
21.     }
22.
23.     #button1{
24.       font-family: "Arial Black", Gadget, sans-serif;
25.         text-align:center;
26.         margin: 50px auto;
27.         padding: 20px 0px;
28.         display:block;
29.         height:70px;
30.         width:259px;
31.
32.         cursor:pointer;
33.         font-size:20px;
34.         color:#F0F0F0;
35.         border-radius:14px;
36.         border-bottom:4px solid #000000;
37.         box-shadow: 6px 6px 6px #999;
38.
39.         -webkit-user-select: none;
40.         -moz-user-select: none;
41.         -ms-user-select: none;
42.         -o-user-select: none;
43.         user-select: none;
44.
45.         background: -webkit-linear-gradient(#000000, #c1c1c1);
46.         background: -o-linear-gradient(#000000, #c1c1c1);
47.         background: -moz-linear-gradient(#000000, #c1c1c1);
48.         background: linear-gradient(#000000, #c1c1c1);
49.
50.         -webkit-transition: background 0.05s linear;
51.         -moz-transition: background 0.05s linear;
52.         -o-transition: background 0.05s linear;
53.         transition: background 0.05s linear;
54.
55.         background-size:1px 200px;
56.
57.         transition: all .05s linear;
58.     }
59.
60.     #button1:hover{
61.         background-position:100px;
62.     }
63.
64.
65.
66.     #button2{
```

```
      font-family: "Arial Black", Gadget, sans-serif;
        text-align:center;
        margin:20px auto;
        padding: 20px 0px;
        display:block;
        height:70px;
        width:250px;

        cursor:pointer;
        font-size:20px;
        color:#F0F0F0;
        border-radius:14px;
        border-bottom:4px solid #000000;
        box-shadow: 6px 6px 6px #999;

        -webkit-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        -o-user-select: none;
        user-select: none;

        background: -webkit-linear-gradient(#000000, #c1c1c1);
        background: -o-linear-gradient(#000000, #c1c1c1);
        background: -moz-linear-gradient(#000000, #c1c1c1);
        background: linear-gradient(#000000, #c1c1c1);

        -webkit-transition: background 0.05s linear;
        -moz-transition: background 0.05s linear;
        -o-transition: background 0.05s linear;
        transition: background 0.05s linear;

        background-size:1px 200px;

        transition: all .05s linear;
    }

    #button2:hover{
        background-position:100px;
    }


    #button3{
      font-family: Impact, Charcoal, sans-serif;
        text-align:center;
        margin:90px auto;
        padding: 20px 0px;
        display:block;
        height:80px;
        width:260px;

        cursor:pointer;
        font-size:20px;
        color:#F0F0F0;
        border-radius:14px;
```

```
        border-bottom:4px solid #000000;
        box-shadow: 6px 6px 6px #999;

        -webkit-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        -o-user-select: none;
        user-select: none;

        background: -webkit-linear-gradient(#000000, #c1c1c1);
        background: -o-linear-gradient(#000000, #c1c1c1);
        background: -moz-linear-gradient(#000000, #c1c1c1);
        background: linear-gradient(#000000, #c1c1c1);

        -webkit-transition: background 0.05s linear;
        -moz-transition: background 0.05s linear;
        -o-transition: background 0.05s linear;
        transition: background 0.05s linear;

        background-size:1px 200px;

        transition: all .05s linear;
    }

    #button3:hover{
        background-position:100px;
    }



    body{padding:5%;}
    .content{
      padding:3% 5%;
      margin-bottom:3%;
      box-shadow:0px 2px 2px rgba(0,0,0,0.1);
      border-radius:3px;
      color:black;
      border-top:1px solid rgba(255,255,255,0.15);
      text-shadow: 0px 1px 0px rgba(0,0,0,0.15);
    }
    .content:after{
      content:' ';
      background:rgba(255,255,255,0.15);
      width:100%;
      height:100%;
      display:block;
      position: absolute;
      top:0px;
      left:0px;
      border-radius:3px;
      z-index:1;
    }

    .content *{z-index:2;position:relative;}
```

```
175.     .col-1-2{width:40.5%;float:left;}
176.     .col-1-2 + .col-1-2{margin-left:1%;}
177.     .clear{clear:both;}
178.
179.     .title {
180.         font-family:"Arial Black", Gadget, sans-serif;
181.         color: white;
182.     }
```

5. User Manual

5.1 Assembly

5.1.1 Insert the SD card into the SD port on the Raspberry

5.1.2 Connect the Pressure Sensor (PS) cables to the GPIO Pins on the Raspberry Pi

5.1.2.1 Connect PS GND cable to Pin 9

5.1.2.2 Connect PS 3V PWR cable to Pin 1

5.1.2.3 Connect PS SDI cable to Pin 3 (I2C SDA port)

5.1.2.4 Connect PS SCK cable to Pin 5 (I2C SCL port)

5.1.3 Assemble and connect the Servo Assembly

5.1.3.1 Connect Red Ribbon Servo (Servo 1) to Channel 1 of the Maestro Micro-controller 3 pin servo hub

5.1.3.1.1 1st channel is channel 0, not 1

5.1.3.1.2 Ensure Orange (Signal) cable is connected to inboard pin of 3 pin channel and Purple (GND) is outboard pin, with Red (PWR) cable in middle

5.1.3.2 Connect Black Ribbon Servo (Servo 2) to Channel 5 on the Maestro Micro-Controller Servo hub

5.1.3.3 Connect GND pin for Maestro servo hub (outboard pin of the 2 pin row at top of servo hub) to Pin 20 on Pi

5.1.3.4 Connect PWR pin for Maestro servo hub (inboard pin of the 2 pin row at top of servo hub) to Pin 17 on Pi

5.1.3.5 Connect Micro-USB to USB cable to the Maestro and any open USB port on Pi

5.1.4 Connect Hot Wire Assembly (HW)

5.1.4.1 Connect the HW assembly circuit power to pin 2

5.1.4.2 Connect the HW assembly circuit ground to pin 14

5.1.4.3 Connect the HW circuit signal input to pin 11 corresponding to GPIO pin 17

5.1.5 Connect GPS Receiver to any open USB port

5.1.6 Connect Wi-Fi adapter to any open USB port

5.1.7 Connect external memory to any open USB port

5.1.8 Connect Battery Assembly to Pi

5.1.8.1 Connect cable jack into pins 4 and 6

5.1.8.2 Ensure Red cable (PWR) is connected to Pin 4

5.1.8.3 Ensure Black cable (GND) is connected to Pin 6

5.1.9 Hardware assembly should now be complete

5.1.9.1 Raspberry Pi circuit card should have illuminated red LED and blinking green LED

5.1.9.2 Maestro Micro-controller should have steady illuminated green LED and blinking orange LED

5.2 Startup

5.2.1 Connect Linksys router and ensure it is powered on

5.2.2 Reconfigure UI device to connect to Linksys router

5.2.3 Download Putty via http://www.putty.org/

5.2.4 Secure Shell (SSH) into the Pi by typing 192.168.1.131

5.2.4.1 Login as “pi”

5.2.4.2 Password as “blueteam”

5.2.5 This will open a terminal window for the Pi

5.2.5.1 Type “sudo –s”

5.2.5.2 Navigate to the flight software by typing:

5.2.5.2.1 “cd” then type “cd /var/www/html/weather_balloon”

5.2.6 From User Workstation open web browser

5.2.7 Type IP Address: 192.168.1.131

5.2.8 Login with username “TeamBlue” and password “rosebowl”

5.2.8.1 Hit “Login” to get to UI

5.2.8.2 Hit “Guest” to get to disabled UI

5.3 Operation

5.3.1 User Interface will display data – read only

5.3.2 OPTIONAL:

5.3.2.1 Press “Release Red Ribbon

5.3.2.1.1 Servo 1 should actuate

5.3.2.2 Press “Release Black Ribbon”

5.3.2.2.1 Servo 2 should actuate

5.3.2.3 Press “Release Payload”

5.3.2.3.1 Relay should click and HW should heat up

5.4 Shutdown

5.4.1 Main FSW will terminate after HW execution – Physically recover system components

5.4.2 Shutdown OS

5.4.2.1 OPT 1: Use SSH to remote into the Pi via Putty

5.4.2.2 OPT 2: Use external peripherals to connect to PI

5.4.2.2.1 Disconnect two devices from USB ports

5.4.2.2.2 Connect Keyboard and Mouse

5.4.2.2.3 Connect HDMI cable and associated required video adapters to HDMI port on the Pi and to external display

5.4.2.3 Click the Raspberry Pi Home button on top left corner of screen
5.4.2.4 Select the “Shutdown” option from the dropdown menu
5.4.2.5 Select “Shutdown” from the submenu
5.4.2.6 The Raspberry Pi should shut down.
5.4.3 Disconnect hardware, starting with Battery Assembly
5.4.4 Remove SD Card and place carefully back into USB holder
5.4.5 Simply remove external memory source to retrieve saved data

5.5 Pre/Post Conditions

5.5.1 Hot Wire Assembly is permanently assembled into standalone assembly except for transistor base terminal
5.5.2 Camera is already installed
5.5.3 Battery and transformer are permanently assembled and are standalone assembly
5.5.4 User workstation and ground antenna are connected and configured
5.5.5 Static IP’s have been assigned to user workstation and Raspberry Pi
5.5.6 All OS and SW components and updates are pre-loaded on SD card

6. Testing

6.1 GPS

The GlobalSat BU353S4 GPS Receiver aboard the payload contains vital altitude data that is communicated to other components. A margin of error for altitude readings must be determined to better enhance the software. To test the GPS, it is suggested that:

1. The GPS is visible to the sky in an open area.
2. Appropriate applications are installed on the testing teams’ computer to show the array of NMEA data read in and analyzed.
3. Units are converted to feet, as they are output in meters.
4. Units are converted to height AGL as they are output in MSL.
5. The GPS unit is elevated at a rate comparable to the ascent rate of the weather balloon.

Fixes: The GPS was further tested outside and it was determined that the margin of error was not important within requirements and was not a mission critical issue.

6.2 Camera

The camera aboard the payload is a standard camera for the Raspberry Pi. The camera outputs a RGB photo that is 1280 x 960. The camera is already integrated into the Raspberry Pi. The software for the camera needs to be tested to ensure the pictures are coming in clear and in the right orientation. To test the camera, it is suggested that:

1. The camera is taking a picture of a distinct object to ensure quality of the photo.
2. The folder in which the picture is taken is easy to access.
3. The terminal is used to ensure functionality of the camera.

Unit tests can begin on the camera.

Fixes: Testing determined that Windows OS machines did not understand the format from which the camera output images. It was determined that the code written output in .ppm (portable pixel map) format which is a format that only the Raspberry Pi understands. Code was re-written with OpenCV to rebuild the RaspiCam libraries. The new camera code no longer includes a CAMERA class but calls an object from the Raspicam_CV library. As well, the new camera code outputs a black and white .jpg file, which is now compatible with Windows machines.

6.3 Sensor

The sensor aboard the payload consists of a BMP280 Pressure, Temperature, and Humidity sensor. This sensor relies on the correct installation of the i2C configuration of the Raspberry Pi. To test the Sensor, it is suggested that:

1. The Sensor is correctly pinned to the Raspberry Pi via the diagram in 3.2.
2. The Sensor is in a controlled environment (hot or cold) to ensure temperature data.
3. The Sensor is in a controlled environment (high or low pressure) to ensure pressure data.

Unit tests can begin on Pressure and Temperature classes.
Fixes: The pressure and temperature sensor were successful in unit tests and the code did not cause issues. The printing issue was fixed by deleting a printf statement in the pressure.cpp and temp.cpp class functions.

6.4 Hotwire

The Hotwire aboard the payload is a nichrome wire for which current will be sent through. The nichrome wire must heat up enough to cut the balloon cord at the payload release height. To test the Hotwire, it is suggested that:

1. The Hotwire is in a controlled environment as it will become extremely hot.
2. The Hotwire has an independent circuit as to not destroy the Raspberry Pi.
3. The Hotwire is clocked at testing to determine a burn time.

Unit tests can begin on HOTWIRE class and integrated hardware.
Fixes: Testing the hotwire led to the construction of a separate circuit using an external power source with a transistor and relay to supply enough current for the nichrome wire to heat up. The HOTWIRE class itself produced few errors and required simple fixes such as calling WiringPiSetupGPIO to enable GPIO pins on the Pi. GPIO pin 17 is used for the hotwire circuit.

6.5 Servos

The Osoyoo SG90 9G Servo Motors aboard the payload connect to a Pololu Micro Maestro-6-Channel USB Controller and must be powered via the Raspberry Pi. It is crucial that the Servos assume any channel on the USB Controller. To test the Servos, it is suggested that:

1. The Servos are correctly pinned to the Raspberry Pi via the diagram in 3.2.
2. The Servos are tested with user installed software via the Pololu website.

3. The Servos are tested in both directions of movement.
4. The Servos are tested on all channels of the USB Controller.

Unit tests can begin on Servo classes and integrated ribbon deployment systems. Fixes: Testing found that adding a return function for the servos would be beneficial for the mission. It was determined that adding a movement in the opposite direction was too difficult to implement because the servos accept a frequency reading and not an actual position input. The servos are now shut off after 1 second of actuation if altitude is correct with a call of zero to their target function.

6.6 Interface

The User Interface (UI) displayed on the ground station can be accessed from any PC with the correct IP address. To test the User Interface, it is suggested that:

1. The UI is accessed from different PCs.
2. Small executable are called from button clicks (i.e. stubs).

Fixes: The UI required a lot of work due to Linux permissions as a root or Pi user. Originally, the flight software would be called from the UI. This caused the code to hang at the same page until it finished. The user must now SSH into the Pi and start the software with command "./weather_balloon". This method now allows the UI page to appear with no issues.