Weather Balloon Surveillance Package

Code Development Document

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

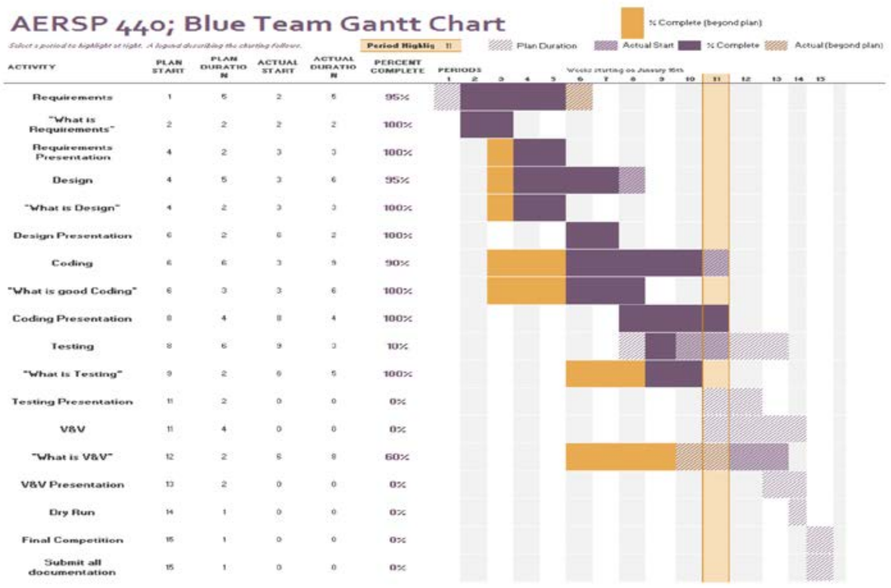
Table of Contents

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*



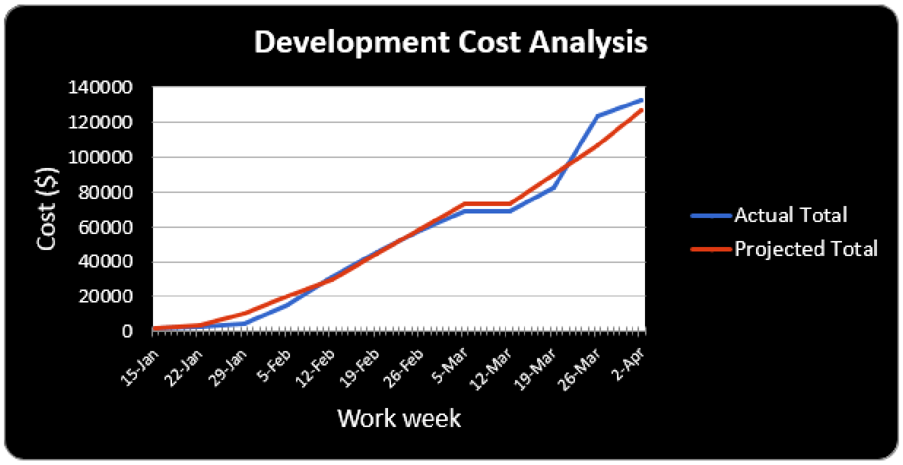
ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Requirements	1	5	2	5	95%
"What is Requirements"	2	2	2	2	100%
Requirements Presentation	4	2	3	3	100%
Design	4	5	3	6	95%
"What is Design"	4	2	3	3	100%
Design Presentation	6	2	6	2	100%
Coding	6	6	3	9	90%
"What is good Coding"	6	3	3	6	100%
Coding Presentation	8	4	8	4	100%
Testing	8	6	9	3	10%
"What is Testing"	9	2	6	5	100%
Testing Presentation	11	2	0	0	0%
V&V	11	4	0	0	0%
"What is V&V"	12	2	6	8	60%
V&V Presentation	13	2	0	0	0%
Dry Run	14	1	0	0	0%
Final Competition	15	1	0	0	0%
Submit all documentation	15	1	0	0	0%

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.

2.1.1.7 The user interface will have a button for saving an image.

2.1.1.8 The user interface will have a button for rebooting the system manually.

Flight Payload

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

2.1.1.10 The flight computer will transmit data to the laptop.

2.1.1.11 The flight computer shall be able to reboot quickly.

2.1.1.12 The flight payload shall autonomously unfurl a red ribbon when altitude data reaches 1000 ft.

2.1.1.13 The flight payload shall autonomously unfurl a black/white ribbon when altitude data reaches 2000 ft.

2.1.1.14 The flight payload shall autonomously release the payload when altitude data reaches 3000 ft.

2.1.1.15 The flight payload shall respond to commands from the laptop.

2.1.1.16 The payload shall weigh no more than 4 pounds.

2.1.1.17 A parachute shall automatically deploy following payload separation.

Power

2.1.1.18 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

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 Altitude data shall be visible on a graph plotted against temperature data.

2.1.2.2.3.3 Altitude data shall be visible on a graph plotted against pressure data.

2.1.2.2.3.4 The user interface shall update pressure data every 0.5 seconds.

2.1.2.2.3.5 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.

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

Commands

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

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

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

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

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

2.1.2.4.4 One command shall release the payload from the balloon.
~~2.1.2.4.5 One command shall reboot the flight computer.~~
2.1.2.4.6 One command shall take a picture from the flight payload and does not require a 2-step confirmation procedure.
2.1.2.4.7 One control shall save the displayed picture and does not require a 2-step confirmation procedure.

Flight Payload

2.1.2.5 The flight computer shall continuously measure data from onboard instruments.
2.1.2.5.1 The flight computer shall sample altitude data at 4 Hz.
2.1.2.5.2 The flight computer shall sample pressure data at 0.2 Hz.
2.1.2.5.3 The flight computer shall sample temperature data at 0.2 Hz.
2.1.2.5.4 The flight computer shall sample GPS location at a rate of 0.2 Hz.
2.1.2.5.5 The flight computer shall take a picture once per 10 seconds.
2.1.2.6 The flight computer shall transmit data from onboard instruments to the user interface.
2.1.2.6.1 The flight computer shall transmit altitude measurements at a data rate not exceeding 1000 bps.
2.1.2.6.1.1 Altitude data shall use 5 significant figures.
2.1.2.6.2 The flight computer shall transmit pressure measurements at a data rate not exceeding 1000 bps.
2.1.2.6.2.1 Pressure data shall use 4 significant figures.
2.1.2.6.3 The flight computer shall transmit temperature measurements at a data rate not exceeding 1000 bps.
2.1.2.6.3.1 Temperature data shall use 3 significant figures.
2.1.2.6.4 The flight computer shall transmit an image once per 10 seconds.
2.1.2.6.5 The flight computer shall transmit images at a data rate of no more than 630,000 bps.
2.1.2.6.6 The flight computer shall transmit GPS location at a data rate not exceeding 1000 bps.
2.1.2.6.6.1 GPS data shall use 6 significant figures.
~~2.1.2.7 The flight computer shall be rebootable within 30 seconds of going offline.~~
~~2.1.2.7.1 A reboot shall occur if a command is received from the user interface.~~
~~2.1.2.7.2 A reboot shall occur autonomously following a flight software failure.~~
2.1.2.8 The flight payload shall autonomously unfurl a red ribbon when altitude data reaches 1000 ft.
2.1.2.8.1 The red ribbon unfurl event response time shall be no more than 1 second.
2.1.2.9 The flight payload shall autonomously unfurl a black/white ribbon when altitude data reaches 2000 ft.

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

2.1.2.10 The flight payload shall autonomously release the payload when altitude data reaches 3000 ft.

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

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

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

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

~~2.1.2.11.3 The flight payload shall process and execute a command to reboot.~~

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

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

2.1.2.12 A parachute shall autonomously deploy following payload separation.

Power

2.1.2.13 The flight payload shall have sufficient battery power to run flight hardware for no less than 2 hours.

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

Derived Hardware Constraints

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

2.1.2.16 The flight software shall not exceed 32 GB.

2.1.2.17 The flight computer shall run using 1 GB of RAM.

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

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

2.1.2.20 The flight system shall not use more than 1 Ethernet port.

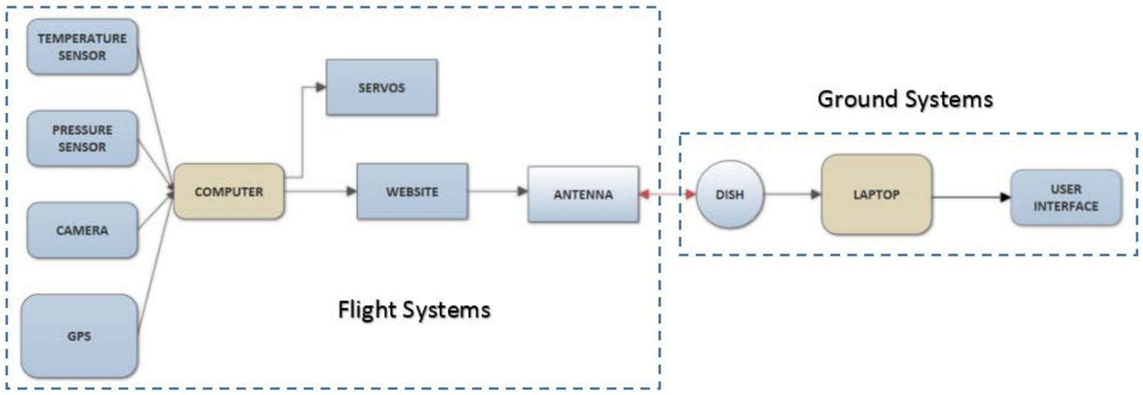
2.1.2.21 The flight system shall transmit data through WiFi.

2.1.2.22 The user interface shall transmit commands through WiFi.

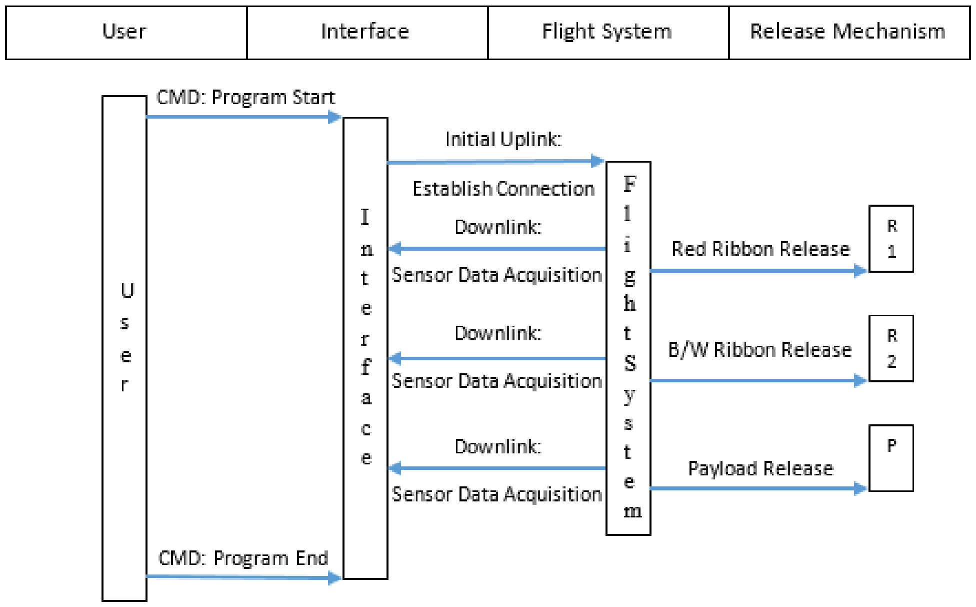
2.1.2.23 Image resolution shall be 5 MP.

2.2 Design

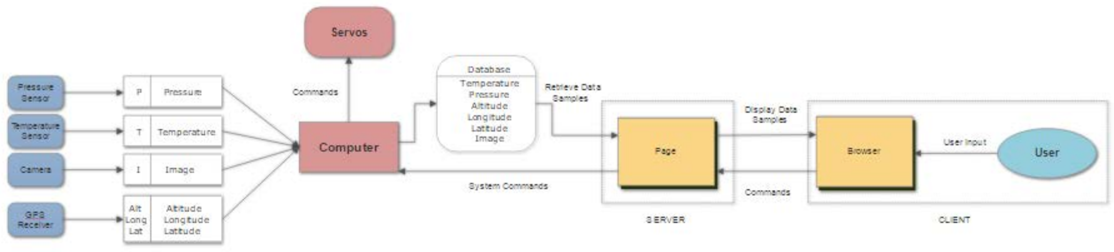
2.2.1 Structural Model



2.2.2 Sequence Diagram

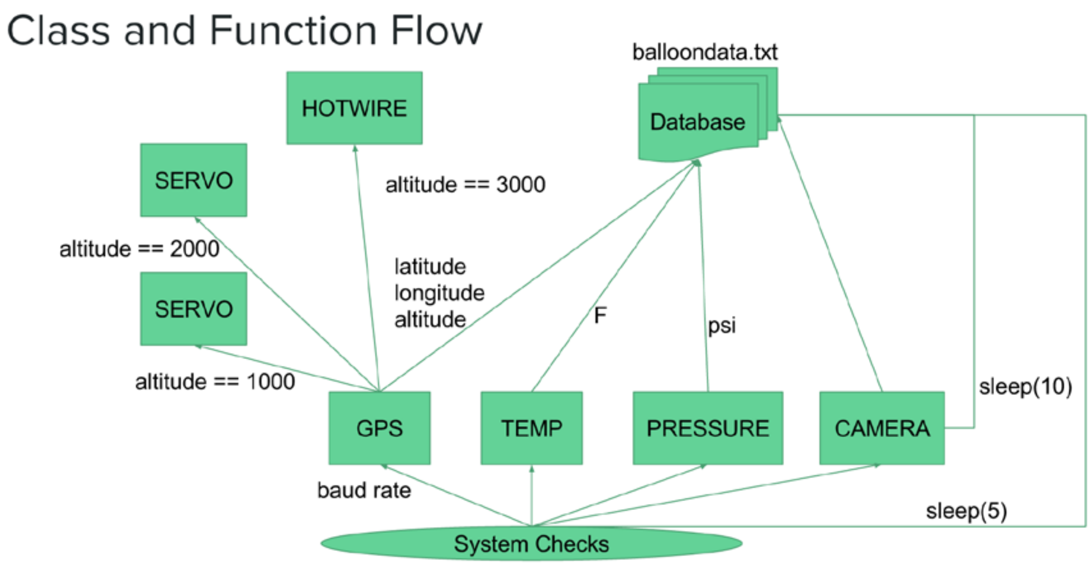


2.2.3 Data Flow Model

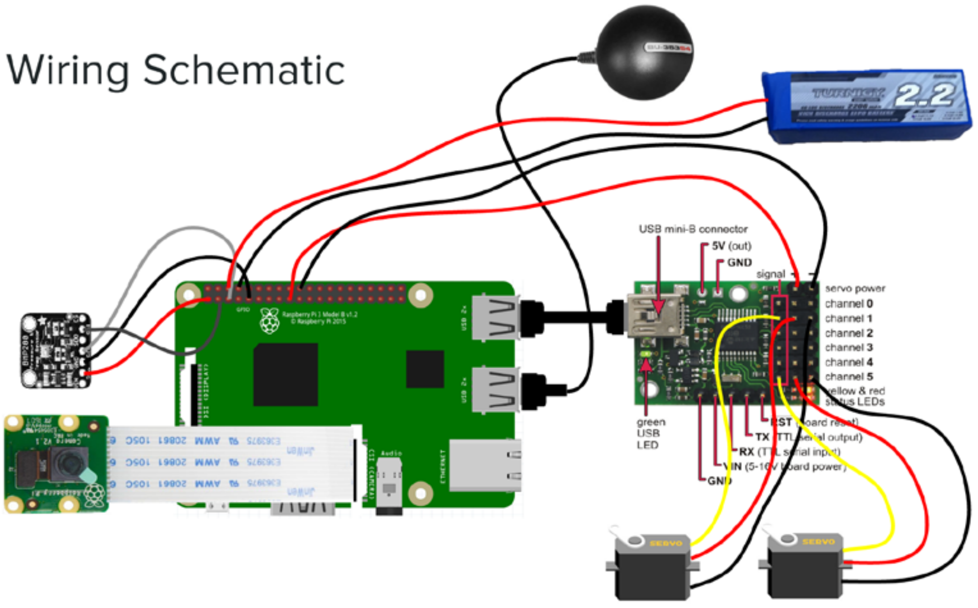


3. Flowcharts

3.1 Class and Function Flow

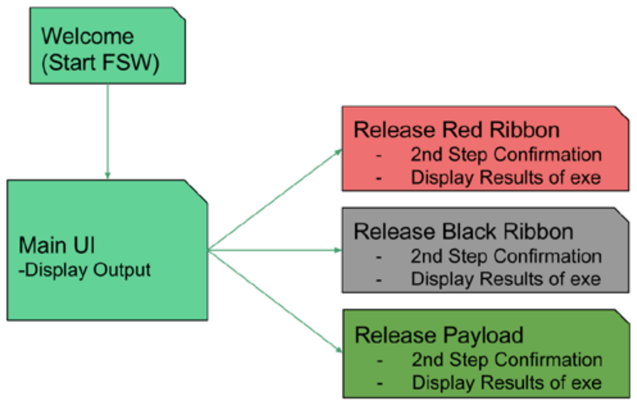


3.2 Wiring Schematic



3.3 User Interface

User Interface

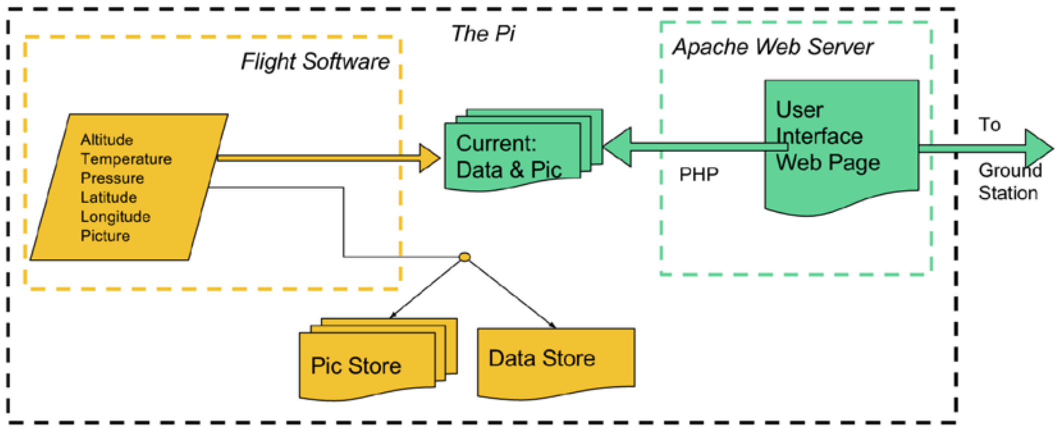


- Implemented utilizing Apache Web Server Software
- Consists of HTML pages and CSS style sheets
- Tasks executed using PHP scripting language

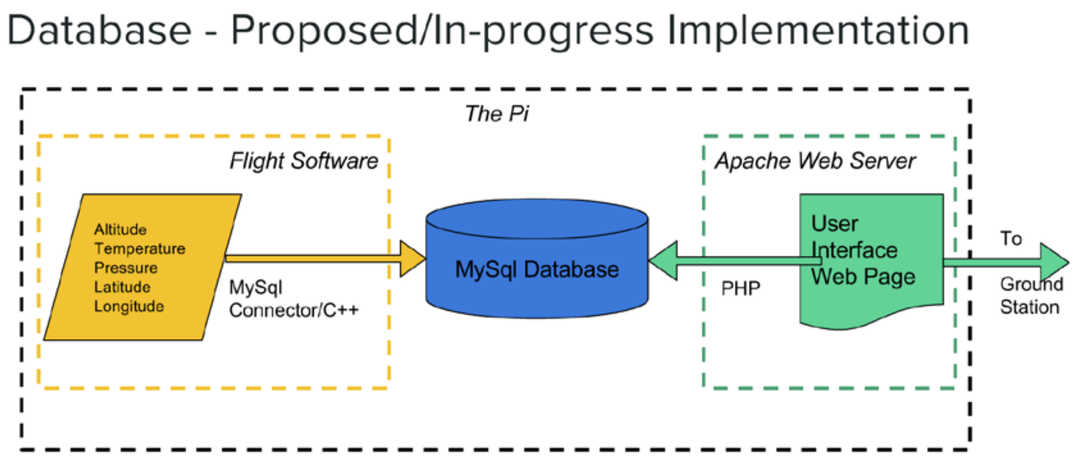
3.4 Database

3.4.1 Current Implementation

Database - Current Implementation



3.4.2 *Propose/In-progress Implementation*



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
    #include "camera.h"

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

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

    //Pressure Sensor
    #include "pressure.h"
```

```cpp

//Output
#include <fstream>

using namespace std;

int main()
{
    //----------System checks----------
        //*****TIMER*****
        //Start timer
        clock_t startTimer=clock();

        //*****GPS*****
        //Set baud rate for GPS
        system("stty -F /dev/ttyUSB0 4800");
        //Input NMEA line for GPS class
        string nmea;
        fstream f;
        //Open GPS USB device
        f.open("/dev/ttyUSB0");

        //*****SERVOS*****
        //Open Pololu Micro Maestro 6-channel USB controller
        const char * device = "/dev/ttyACM0";  // Linux
        int fd = open(device, O_RDWR | O_NOCTTY);
        if (fd == -1)
        {
            perror(device);
            return 1;
        }

        //*****PICTURE*****
        int count=0;

        //*****OUTPUT FILE*****
        ofstream balloondata;
        balloondata.open("balloondata.txt");

        bool hotwire=false;
while(hotwire!=true)
{
    //----------Data Streaming----------
        //**********Latitude, Longitude, Altitude Data**********
        //Call GPS class
        f>>nmea;
        GPS gps(nmea);
            //Is the data GPGGA data only?
            if(gps.isValidGGA(nmea))
            {
                balloondata<<gps.latitude<<" "<<gps.latc<<endl;
                balloondata<<gps.longitude<<" "<<gps.lonc<<endl;
                balloondata<<gps.altitude<<" ft"<<endl;
            }

        //**********Temperature Data**********
        //Call TEMP class
        TEMP temp;
        balloondata<<temp.temp_F<<" F"<<endl;

        //**********Pressure Data**********
        //Call PRESSURE class
        PRESSURE pressure;
        balloondata<<pressure.pressure_psi<<" psi"<<endl;

        //**********Camera Data**********
        //Counter to take pictures every 10 seconds, uses divisor check
        if(count%2==0)
        {
            //Call Raspicam directory classes
            CAMERA camera("Pictures/raspicam_image2.jpg");
        }
        //**********Servos**********
        //Altitude = 1000 feet?
        if(gps.altitude==1000)
        {
            //Call servo class with channel from Maestro
            //Set max position for servo1
            SERVO servo1(fd,1,9600);
        }

        //Altitude = 2000 feet?
        if(gps.altitude==2000)
        {
            //Call servo class with channel from Maestro
            //Set max position for servo2
            SERVO servo2(fd,5,9600);
        }
        //Close port
        close(fd);

        //End timer
        //Clock time elapsed
        clock_t endTimer=clock();
        double duration =(endTimer-startTimer)/(double) CLOCKS_PER_SEC;

        balloondata<<duration<<" ms"<<endl;

        //**********Output**********
        balloondata.close();

        //Wait 5 seconds for new data
        count++;
        //sleep(5);
        durationOverall=duration+5;
        //**********RELEASE**********
        //Ignite nichrome wire and release mechanism at 3000 ft
        if(gps.altitude==3000)
        {
            //Ignite hotwire for 10 seconds
            HOTWIRE hotwire(10);
            hotwire=true;
        }
}
    //-------------------------------------------------------------------------
    //MISSION ACCOMPLISHED
    return 0;
}
```

4.2 GPS

4.2.1 Class Definition

```
#ifndef GPS_H
#define GPS_H

#include <vector>
#include <string>

//Length of minutes data in NMEA
#define MINUTE_LENGTH 9

using namespace std;

//Class definition for GPS
class GPS
{
public:
    //Default constructor
    GPS();
    GPS(const string GGASentence);
    double  latitude;
    double  longitude;
    double  altitude;
    char latc;
    char lonc;
    bool isValidGGA(const string GGASentence);
    //Destructor
    ~GPS();

private:
    void setValuesGGA(const string GGASentence);
    vector<string> splitStringByComma(const string);
    double stringToDouble(const string);
```

```
        double getCoordinates(string);
    };

    double degreesToDecimal(const int Degrees,const double Minutes,const int seconds = 0);

    #endif // GPS_H
```

4.2.2 Class Definition

```
#include <iostream>
#include <string>
#include <vector>
#include <assert.h>
#include <sstream>
#include <stdlib.h>
//GPS header file
#include "gps.h"

    using namespace std;

    //Constructor
    GPS::GPS()
    {
        latitude            =   0;
        longitude           =   0;
        altitude            =   0;
    }

    //Destructor
    GPS::~GPS()
    {
        latitude            =   0;
        longitude           =   0;
        altitude            =   0;
    }

    GPS::GPS(const string GGASentence)
    {
        //If a GPGGA sentence, set output
        if (isValidGGA(GGASentence))
            setValuesGGA(GGASentence);
        else
        {
            latitude    =   latitude;
            longitude   =   longitude;
            altitude    =   altitude;
        }
    }

    //Validation function
    bool GPS::isValidGGA(const string GGASentence)
    {
        //Set data to valid to begin
        bool returnBool = true;
```

```cpp
        //Call split by comma function
        vector<string> elementVector = splitStringByComma(GGASentence);

        //Validating GPGGA data
            //Is the size of the line not equal to 15?
            if (elementVector.size() != 15) returnBool = false;
            else
            {
                //Is header GPGGA?
                if (elementVector[0] != "$GPGGA") returnBool = false;
                //0 indicates invalid data
                if (elementVector[6] == "0") returnBool = false;
                //Is the length of the longitude data less than length of a minute
                string str1;
                str1=elementVector[4];
                if (str1.length() < MINUTE_LENGTH) returnBool = false;
                //Is the length of the latitude data less than length of a minute
                string str2;
                str2=elementVector[2];
                if (str2.length() < MINUTE_LENGTH) returnBool = false;
                if (elementVector[7] == "0") returnBool = false;
            }

        return returnBool;
    }

    //Setter function
    void GPS::setValuesGGA(string GGA)
    {
        //Elevation
        int elevation = 1154;   //ft
        //STL vector calls split function
        vector<string> elementVector = splitStringByComma(GGA);

        // Assert we have a GGA sentence
        assert(elementVector[0] == "$GPGGA");

        //North-South
        //Grab latitude from element vector 2
        this->latitude           = getCoordinates(elementVector[2]);

        //If direction is 'S', set coordinate to south
        if (elementVector[3] == "S")
            latc='S';
        else
            latc='N';

        //East - West
        //Grab longitude from element vector 4
        this->longitude           = getCoordinates(elementVector[4]);
        //If direction is 'W', set coordinate to west
        if (elementVector[5] == "W")
            lonc='W';
        else
            lonc='E';

        //Grab altitude and convert to ft with - elevation above SSL
        this->altitude             = stringToDouble(elementVector[9]);
        this->altitude             =   (3.28*(this->altitude)-elevation);
    }

    //Split by comma function
    vector<string> GPS::splitStringByComma(string input)
    {
        //Initializing variables
        vector<string>  returnVector;
        stringstream    ss(input);
        string          element;

        //Separate data based on comma seperation
            while(getline(ss, element, ','))
            {
                returnVector.push_back(element);
            }

        return returnVector;
    }

    //Decimal degrees function
    double degreesToDecimal(int degrees, double minutes, int seconds )
    {
        //Initializing variables
        double returnDouble = 0;
            //Calculate coordinate in decimal degrees
            returnDouble = degrees + minutes/60 + seconds/3600.0f;

        return returnDouble;
    }

    //String to double conversion function
    double GPS::stringToDouble(string inputString)
    {
        //If string empty, return 0
        double returnValue = 0;
            //Convert to string
            istringstream istr(inputString);
            istr >> returnValue;

        return (returnValue);
    }

    //Coordinate function
    double GPS::getCoordinates(string array)
    {
        //Initializing variables
        double decimalDegrees = 0;
        string degreeArray;
        string minuteArray;

        //Check for length of latitude and longitude and parse correctly
        if (array.length() > MINUTE_LENGTH)
        {

            degreeArray.assign(array.begin()+1, array.end() - 7);
            minuteArray.assign(array.end() - 7, array.end());
        }
        else
        {
            degreeArray.assign(array.begin(),array.end()-7);
            minuteArray.assign(array.begin()+2,array.end());
        }

            //Convert strings into numbers
            int degrees;
            double minutes;
            degrees = atoi(degreeArray.c_str());
            minutes = stringToDouble(minuteArray);

            //Convert degrees and minutes into decimal
            decimalDegrees = degreesToDecimal(degrees,minutes);

        return decimalDegrees;
    }
```

4.3 Camera

4.3.1 Class Definition

```
#ifndef CAMERA_H
#define CAMERA_H

//Class definition for camera
class CAMERA
{
public:
    //Default constructor
    CAMERA();
        CAMERA(const char *path);
        //Destructor
        ~CAMERA();
    };

    #endif /* CAMERA_H */
```

4.3.2 Class Functions

```
#include <iostream>
#include <fstream>
//Raspicam libraries
#include <raspicam/raspicam.h>
#include <raspicam/raspicamtypes.h>
//Camera header file
#include "camera.h"

using namespace std;

    //Constructor
    CAMERA::CAMERA()
    {

    };

    CAMERA::CAMERA(const char* path)
    {
        //Call Raspicam directory classes
        raspicam::RaspiCam Camera;
        //Error if camera cannot open
        if(!Camera.open())
            cout<<"Error opening camera"<<endl;
        //Grab image
        Camera.grab();
        //Allocate space for picture
        unsigned char *data=new unsigned char
[Camera.getImageTypeSize(raspicam::RASPICAM_FORMAT_RGB)];
        //Retrieve data
        Camera.retrieve(data,raspicam::RASPICAM_FORMAT_RGB);
        //Output to a file path
        ofstream outFile(path,ios::binary);
        outFile<<"P6\n"<<Camera.getWidth()<<"
"<<Camera.getHeight()<<" 255\n";
        outFile.write((char*) data,
Camera.getImageTypeSize(raspicam::RASPICAM_FORMAT_RGB));
        //Destroy pointer
        delete data;
    };

    //Destructor
    CAMERA::~CAMERA()
    {

    };
```

4.4 Pressure Sensor

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();
    };

    #endif /* PRESSURE_H */
```

4.4.2 Class Functions

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

    using namespace std;

    //Constructor
    PRESSURE::PRESSURE()
    {
        pressureout();
    };

    void PRESSURE::pressureout()
    {
        //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);

        //Read 24 bytes of data from address(0x88)
        char reg[1] = {0x88};
        write(file, reg, 1);
        char data[24] = {0};
            //Error check on reading input data
        if(read(file, data, 24) != 24)
        {
                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;
```

```c
    }
    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;
        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;
        this->pressure_psi = pressure * hpapsiconversion;

        //Output data to screen
        printf("%.2f psi \n", this->pressure_psi);
    };

    //Destructor
    PRESSURE::~PRESSURE()
    {

    };
```

4.5 Temperature Sensor

4.5.1 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 TEMP_H
#define TEMP_H

    //Class definition for temperature sensor
    class TEMP
    {
    public:
        //Default constructor
        TEMP();
        double temp_F;
        //Destructor
        ~TEMP();
    private:
        void tempout();
    };

    #endif /* TEMP_H */
```

4.5.2 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;
```

```c
        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);

        //Read 24 bytes of data from address(0x88)
        char reg[1] = {0x88};
        write(file, reg, 1);
        char data[24] = {0};
            //Error check on reading input data
        if(read(file, data, 24) != 24)
        {
                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;
        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;

      //Output data to screen
      printf("%.2f F \n", this->temp_F);
    };

    //Destructor
    TEMP::~TEMP()
    {

    };
```

4.6 Hotwire

4.6.1 Class Definition

```
#ifndef HOTWIRE_H
#define HOTWIRE_H

class HOTWIRE
{
public:
    //Default constructor
    HOTWIRE();
    HOTWIRE(int time);
        //Destructor
        ~HOTWIRE();
      };
```

```

     #endif /* HOTWIRE_H */
```

4.6.2 Class Functions

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

//WiringPi libraries
#include <wiringPi/wiringPi.h>
//Hotwire header file
#include "hotwire.h"

     //Define output GPIO pin
     #define DHTPIN      7

     //Constructor
     HOTWIRE::HOTWIRE()
     {

     };

     HOTWIRE::HOTWIRE(int time)
     {
         digitalWrite( DHTPIN, HIGH );
         delay(time);
         digitalWrite( DHTPIN, LOW );
     };

     //Destructor
     HOTWIRE::~HOTWIRE()
     {

     };
```

4.7 Servo Motors

4.7.1 Class Definition

```
#ifndef SERVO_H
#define SERVO_H

//Class definition for servo motors
class SERVO
{
public:
    //Default constructor
    SERVO();
         SERVO(int fd, unsigned char channel,unsigned short
target);
         //Destructor
         ~SERVO();
     private:
         int maestroGetPosition(int fd, unsigned char channel);
```

```
        int maestroSetTarget(int fd, unsigned char channel, unsigned short target);
    };

    #endif /* SERVO_H */
```

4.7.2 Class Functions

```
#include <iostream>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
//Servo header file
#include "servo.h"

using namespace std;

    //C code
    #ifdef _WIN32
    #define O_NOCTTY 0
    #else
    #include <termios.h>
    #endif

    //Constructor
    SERVO::SERVO()
    {

    };

    //Destructor
    SERVO::~SERVO()
    {

    };

    SERVO::SERVO(int fd, unsigned char channel, unsigned short target)
    {
        //C code
        #ifdef _WIN32
            _setmode(fd, _O_BINARY);
        #else
            struct termios options;
                tcgetattr(fd, &options);
                options.c_iflag &= ~(INLCR | IGNCR | ICRNL | IXON | IXOFF);
                options.c_oflag &= ~(ONLCR | OCRNL);
                options.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG | IEXTEN);
                tcsetattr(fd, TCSANOW, &options);
        #endif

                //Get position of servo
                maestroGetPosition(fd, channel);
                //Set position of servos
                maestroSetTarget(fd,channel,target);
    };

    //Position function
    int SERVO::maestroGetPosition(int fd, unsigned char channel)
    {
        //Find the channel the servo is connected to
        unsigned char command[] = {0x90, channel};
        //Error check for channel
        if(write(fd, command, sizeof(command)) == -1)
        {
            perror("error writing");
            return -1;
        }

        //Response check for channel
        unsigned char response[2];
        if(read(fd,response,2) != 2)
        {
            perror("error reading");
            return -1;
        }

        //Return position of the servo
        return response[0] + 256*response[1];
    }

    //Target function
    int SERVO::maestroSetTarget(int fd, unsigned char channel, unsigned short target)
    {
        //Declare position
        int position = maestroGetPosition(fd, channel);
        //Output current position
        printf("Current position is %d.\n", position);
        //Assign command to move on servo channel
        unsigned char command[] = {0x84, channel, target & 0x7F, target >> 7 & 0x7F};
        //Error check for channel
        if (write(fd, command, sizeof(command)) == -1)
        {
            perror("error writing");
            return -1;
        }

      return 0;
    }
```

4.8 User Interface

4.8.1 PHP

```
<!doctype html>
```

```
<html>
<head>
<!--Header (note: page refreshes every 5 sec) -->
<meta charset="utf-8">
<meta http-equiv="refresh" content="5" />
<title>User Interface</title>
<link rel="stylesheet" href="style.css">
</head>

    <body style="background-color:navy;">

    <!--************************ Column 1 ************************ -->


    <div>
        <h1><center><u><font color="white">Surveillance Package Interface</font></u></center></h1>
    </div>

    <div class="content col-1-2">
    <!-- /// Temp, Pres, and Alt reader block - start /// -->
    <?php
        //Load DATA file
        $filename = "/home/timothy/Documents/testResource.txt";
        $fileContents = file_get_contents($filename);
        $data = explode("\n", $fileContents);
    ?>

    <div>
      <ul><font color="white">
        <li>Temperature (&deg;C): <?php   echo $data[0]   ?></li> <!-- temperature input from cpp code -->
        <br>
        <li>Pressure (psi): <?php echo $data[1] ?></li> <!-- pressure input from cpp code -->
        <br>
        <li>Altitude (ft): <?php echo $data[1] ?></li> <!-- altitude input from cpp code -->
          </font>
      </ul>
    </div>
    <!--/// End /// -->


    <!-- /// Ribbon buttons - start /// -->
    <div>
        <form action="servo.php" method="post">
        <input type="hidden" name="color" value="red" />
        <input type="submit" id="button1" value="Release Red Ribbon" />
        </form>

        <form action="servo.php" method="post">
```

```html
        <input type="hidden" name="color" value="black" />
        <input type="submit"  id="button2" value="Release Black Ribbon"/>
        </form>
    </div>
    <!--/// End /// -->

    <br>
    <br>
    <br>
    <h1><center>
      <font color="white">==========================</font>
    </center></h1>

    <!-- /// Payload Button - start /// -->
    <div>
        <form action="action.php">
        <input type="submit" id="button3" value="Release Payload"/>
        </form>
    </div>
    <!--/// End /// -->
    </div>


    <!--************************ Column 2 ************************ -->

    <br>
    <br>
    <br>
    <br>
    <br>
    <br>

    <div>
    <!-- /// Latitude and Longtitude block - start /// -->
    <div class="wrapper" >
    <font color="white">
        <div id="latitude">
            Latitude (&deg;N): <?php echo $data[3] ?> <!-- Latitude input from cpp code -->
        </div>
        <div id="longtitude">
            Longtitude (&deg;W): <?php echo $data[4] ?> <!-- Longtitude input from cpp code -->
        </div>
        </font>
    </div>
    <!--/// End /// -->

    <br>
    <br>
    <br>
```

```

<!-- /// placeholder 1 - start /// -->
<div>
<img src="placeholder1.jpg">
</div>
<!--/// End /// -->


<!-- /// placeholder 2 - start /// -->
<div class="content">
<img src="placeholder2.jpg">
</div>
<!--/// End /// -->
</div>




</body>


</html>
```

4.8.2 CSS

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

#latitude {
    float:left;
}

    #longtitude {
        float:right;
    }

    #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);
        }
        .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;
        }
```

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 base terminal of the HW transistor to Pin 32 on Pi

5.1.5 Connect GPS Receiver to any open USB port

5.1.6 Connect WiFi adapter to any open USB port

5.1.7 Connect Battery Assembly to Pi

- 5.1.7.1 Connect cable jack into pins 4 and 6
- 5.1.7.2 Ensure Red cable (PWR) is connected to Pin 4
- 5.1.7.3 Ensure Black cable (GND) is connected to Pin 6

5.1.8 Hardware assembly should now be complete

- 5.1.8.1 Raspberry Pi circuit card should have illuminated red LED and blinking green LED
- 5.1.8.2 Maestro Micro-controller should have steady illuminated green LED and blinking orange LED

5.2 Startup

5.2.1 From User Workstation open web browser
5.2.2 Type IP Address: 192.168.1.xxx
5.2.3 Browser will display Welcome screen, press "Start"
5.2.3.1 Read out will display System Check output from main Flight Software (FSW)
5.2.4 Go to main User Interface by pressing "UI" Button
5.2.4.1 User Interface will display current data readout

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 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 Secure Shell to remote into the Pi (SSH)
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
5.4.4.1 Insert SD card into Adapter then into target machine
5.4.4.2 Access file location "/…."
5.4.4.3 Move/Delete/Save files from SD card to target machine memory
5.4.4.4 Remove SD Card

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.

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.

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.

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.

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.

6.6 Interface

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