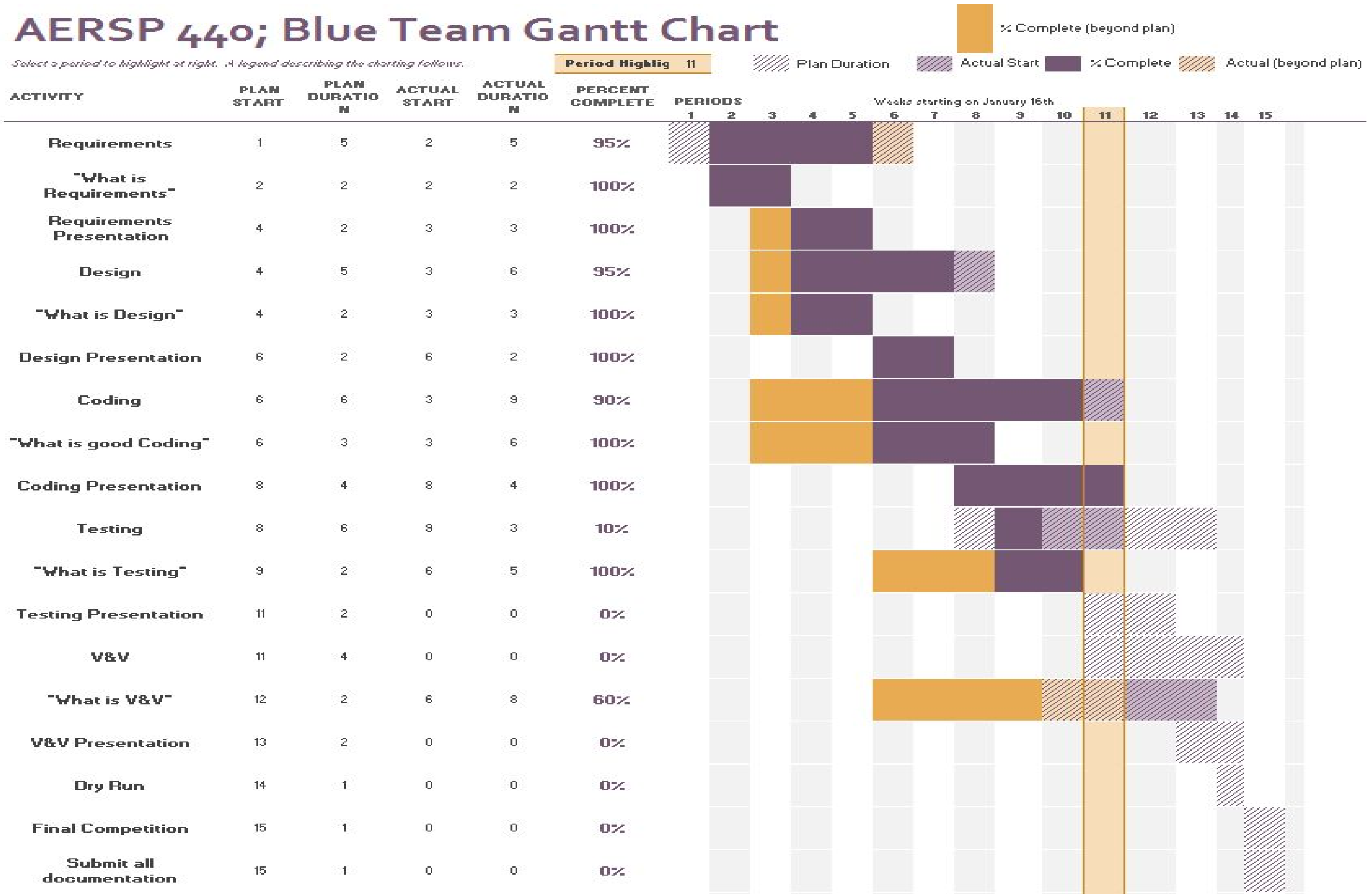
Code Development

Coding Blue Team:
Kevin Parlak
Timothy Sam
Nawaf Abdullah
Evan Kerr
Ali Wahab



AERSP 440; Blue Team Gantt Chart

Select a period to highlight at right. A legend describing the charting follows.

Period Highlig 11

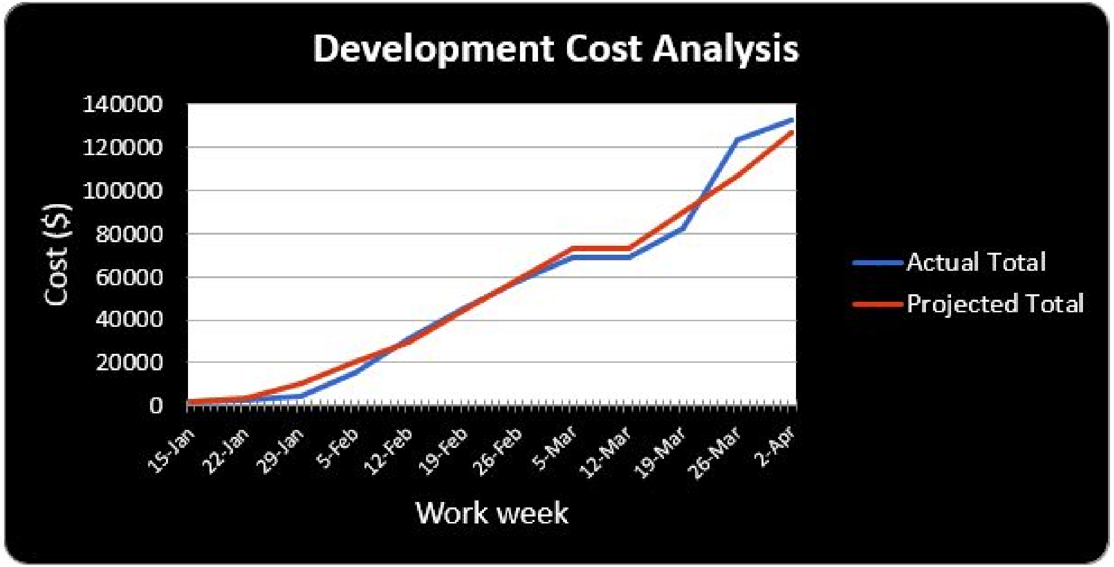
ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Requirements	1	5	2	5	95%
"What is Requirements"	2	2	2	2	100%
Requirements Presentation	4	2	3	3	100%
Design	4	5	3	6	95%
"What is Design"	4	2	3	3	100%
Design Presentation	6	2	6	2	100%
Coding	6	6	3	9	90%
"What is good Coding"	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%
]

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

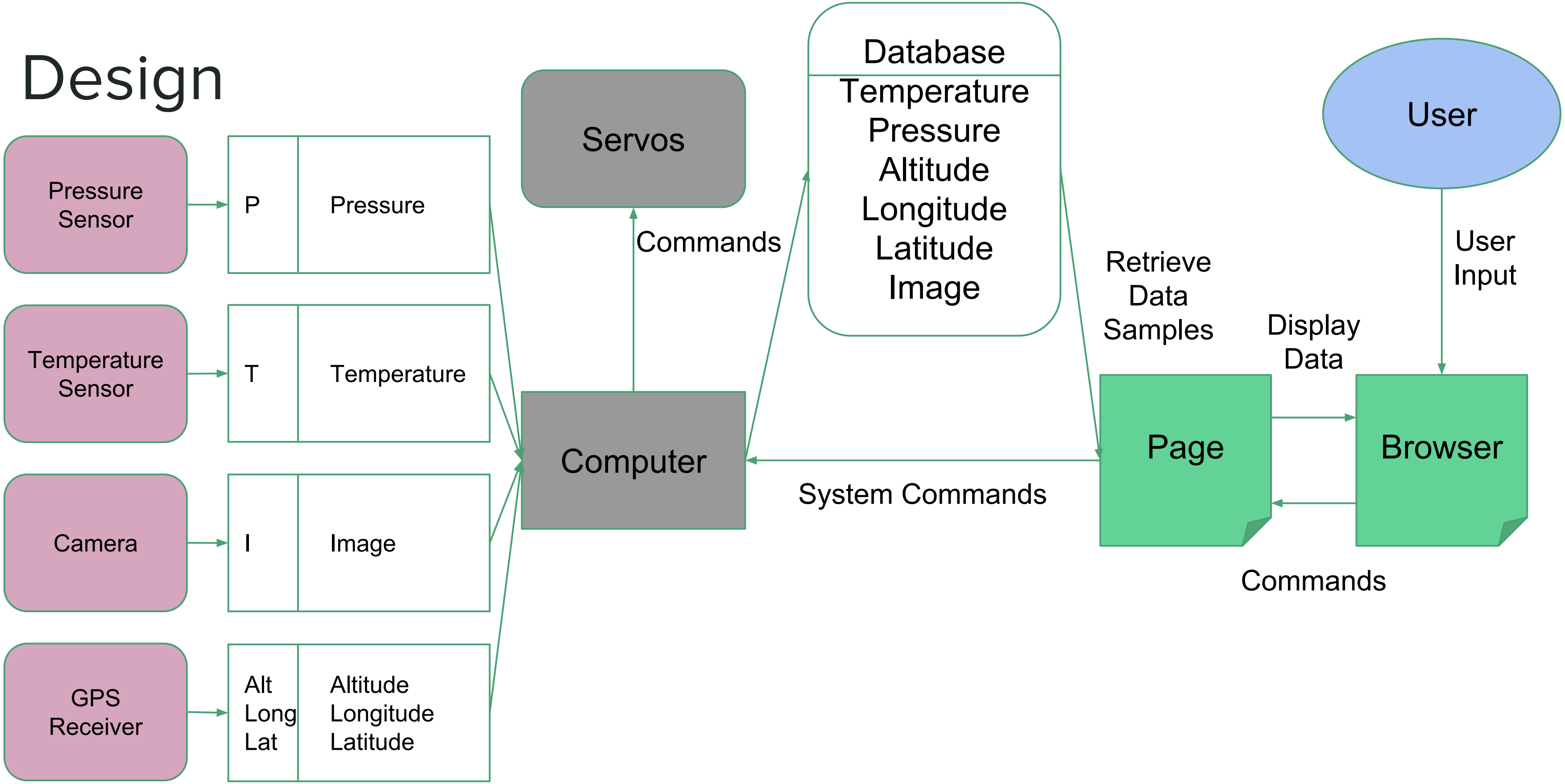


Requirements

System Level Requirements

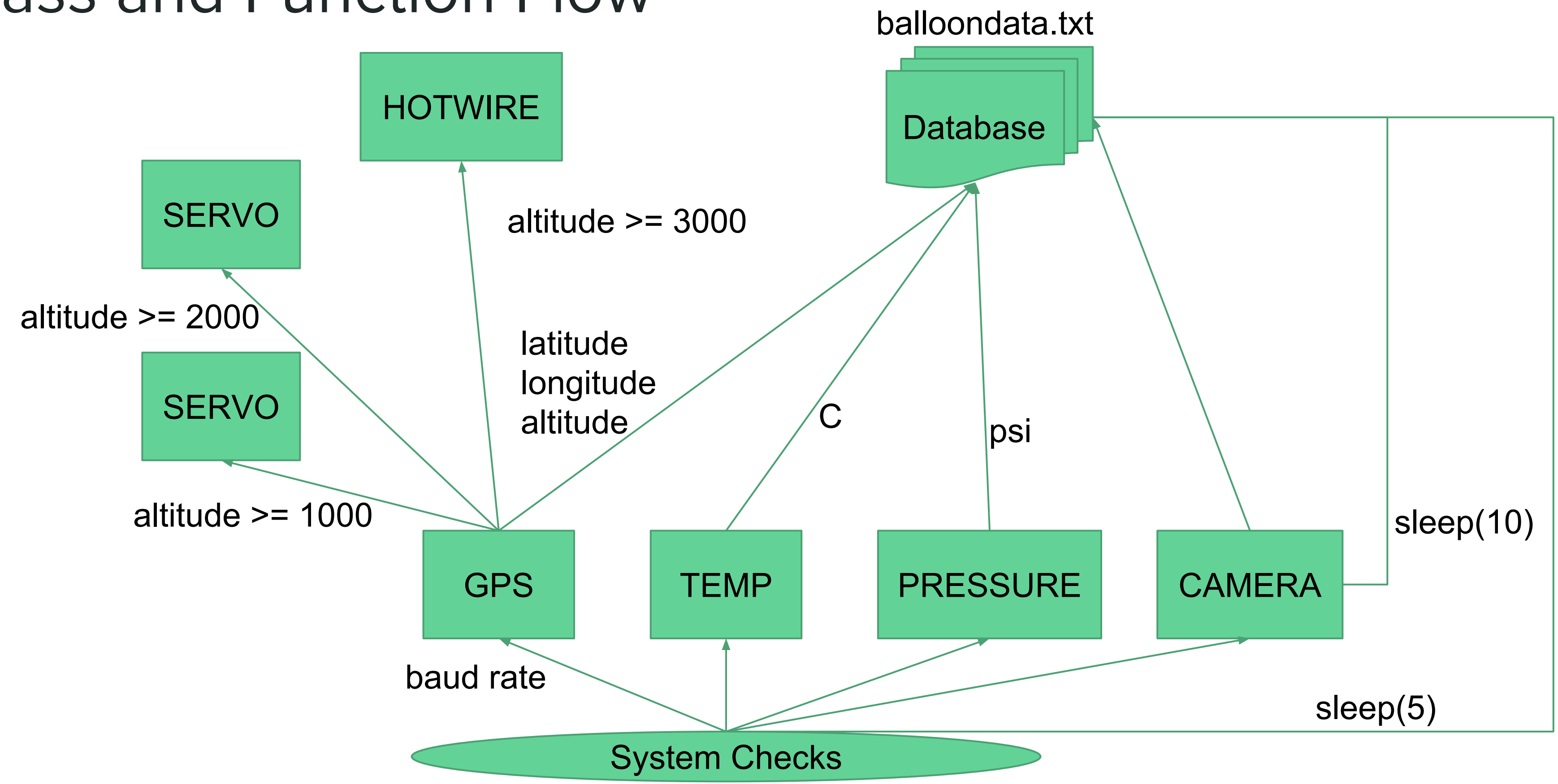
- User Interface
    - **Webhost** - The user interface shall display flight system instrument readings on a laptop within a single window that shows live measurements and graphs data as a function of altitude.
- Flight Payload
    - **Database** - The flight computer shall continuously measure data from onboard instruments.
    - **Webhost** - The flight computer shall transmit data from onboard instruments to the user interface.
    - **GPS, SERVO** - The flight payload shall autonomously unfurl a red ribbon when altitude data reaches 1000 ft.
    - **GPS, SERVO** - The flight payload shall autonomously unfurl a black/white ribbon when altitude data reaches 2000 ft.
    - **GPS, HOTWIRE** - The flight payload shall autonomously release the payload when altitude data reaches 3000 ft.
    - **Executables** - The flight payload shall receive, process, and execute commands from the user interface as a redundant safety measure.
    - **HOTWIRE** - A parachute shall autonomously deploy following payload separation.

Design



Pressure Sensor
P
Pressure
Temperature Sensor
T
Temperature
Camera
I
Image
GPS Receiver
Alt
Long
Lat
Altitude
Longitude
Latitude
Servos
Commands
Computer
Database
Temperature
Pressure
Altitude
Longitude
Latitude
Image
System Commands
Retrieve Data Samples
Page
Display Data
Commands
Browser
User
User Input

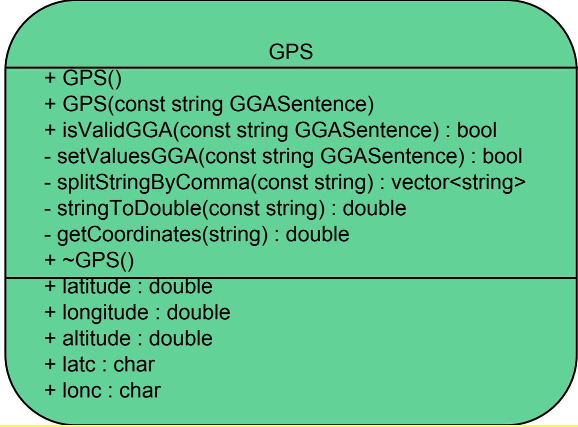
Class and Function Flow



GPS Unit

Use of NMEA Parser algorithm:

- GPGGA data only
  - Latitude, longitude, and altitude
- Parsed by separating data into vectors by comma separation
  - splitStringByComma
- Two functions determine the parse
  - isValidGGA
  - setValuesGGA



GPS

+ GPS()
+ GPS(const string GGASentence)
+ isValidGGA(const string GGASentence) : bool
- setValuesGGA(const string GGASentence) : bool
- splitStringByComma(const string) : vector<string>
- stringToDouble(const string) : double
- getCoordinates(string) : double
+ ~GPS()

+ latitude : double
+ longitude : double
+ altitude : double
+ latc : char
+ lonc : char

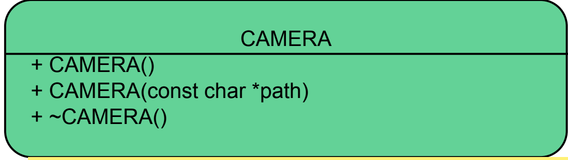
```
//**********Latitude, Longitude, Altitude Data**********
//Call GPS class
f >> nmea;
GPS gps(nmea);
//Is the data GPGGA data only?
if (gps.isValidGGA(nmea))
{
        balloondata << gps.latitude << " " << gps.latc << endl;
        balloondata << gps.longitude << " " << gps.lonc << endl;
        balloondata << gps.altitude << " ft" << endl;
}
```

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

OOP Advantage:

- Parsing scheme hidden within the object

Camera Unit



CAMERA

+ CAMERA()
+ CAMERA(const char *path)
+ ~CAMERA()

```
//**********Camera Data**********
//Call CAMERA class
CAMERA camera("Pictures/raspicam_image2.jpg");
```

Use of open source Raspicam libraries:

- Holds all functions and serial calls to Raspberry Pi Camera port
- https://github.com/cedricve/raspicam.git

OOP Advantage:

- Specify any file path to save the picture
- Easy call in main

Pressure Sensor

Call pressureout() function

Open i2c bus on GPIO

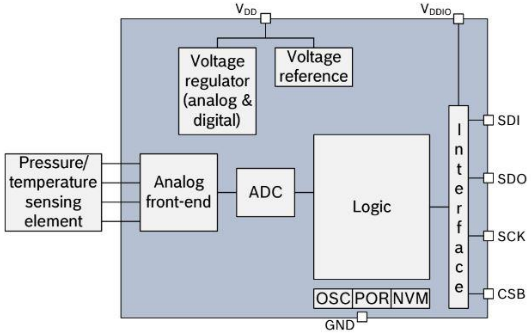
Read 24 bytes of data from the specific address

Convert the data acquired to pressure and temperature coefficients

Convert temperature and pressure data to 19 bits

Add offset to data

Output the data



PRESSURE
+ PRESSURE() - pressureout() : void + ~PRESSURE()
+ pressure_psi : double

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

$$data\_filtered = \frac{data\_filtered\_old \cdot (filter\_coefficient - 1) + data\_ADC}{filter\_coefficient},$$

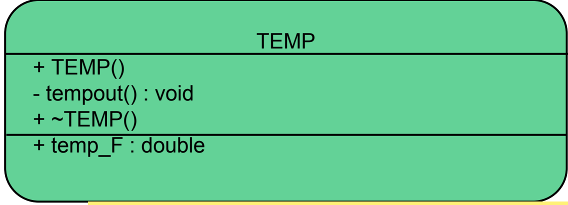
Use of i2C pins on the Pi:

- Changeable slave address

OOP Advantage:

- Algorithm is hidden in the object
- Easy call in main

Temperature Sensor



TEMP

+ TEMP()
- tempout() : void
+ ~TEMP()

+ temp_F : double

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

Option 1 - DHT-11 Sensor

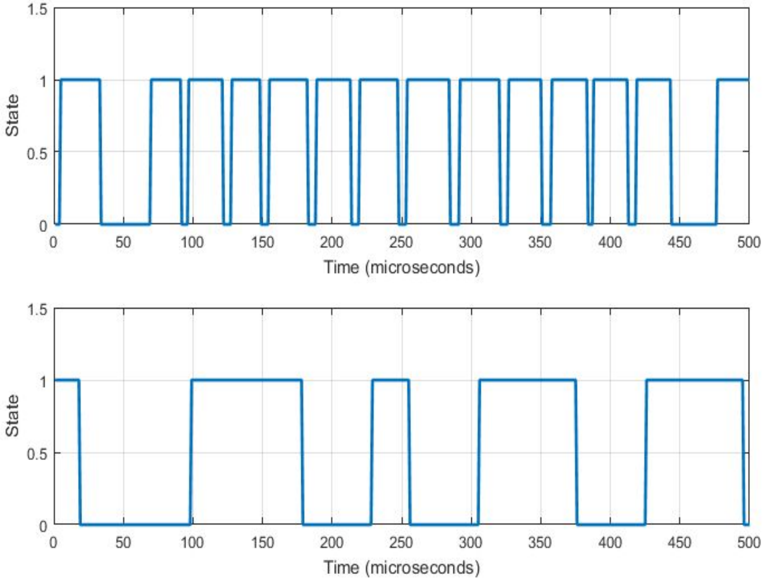
- Noisy data, high error rate
- Complicated read process

Option 2 - BMP280 Pressure Sensor

- Includes high quality temp sensor

OOP Advantage:

- Algorithm is hidden in the object
- Easy call in main



Hot Wire

HOTWIRE

+ HOTWIRE()
+ HOTWIRE(int time)
+ ~HOTWIRE()

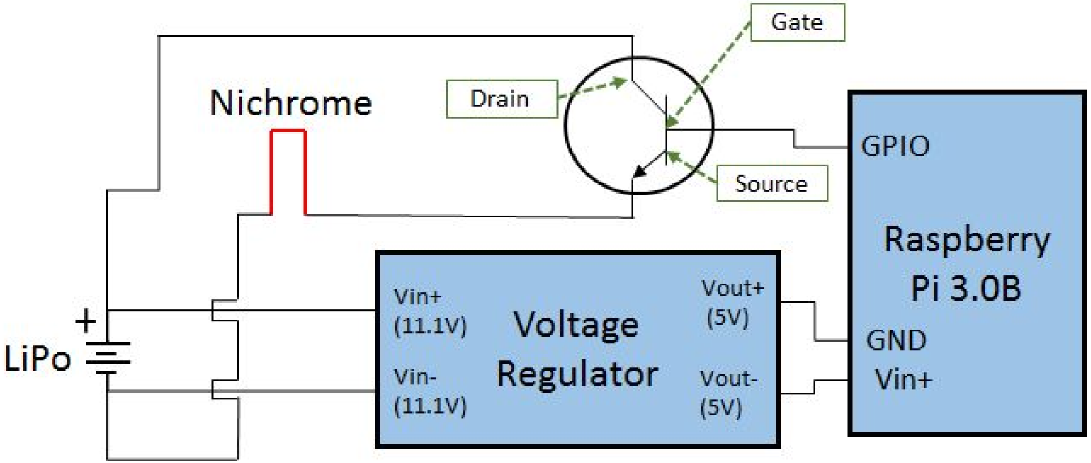
```
//**********RELEASE**********
//Ignite nichrome wire and release mechanism at 3000 ft
if (gps.altitude == 3000)
{
            //Ignite hotwire for 10 seconds
            HOTWIRE hotwire(10);
}
```

Use of open source WiringPi libraries:

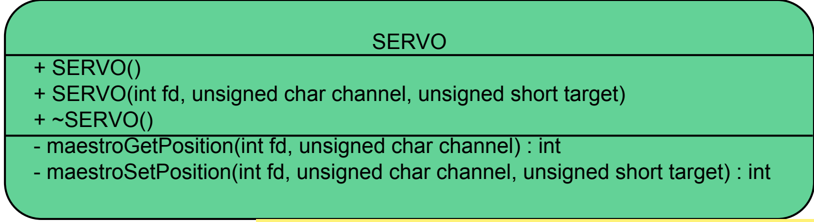
- GPIO output signal
- https://github.com/WiringPi/WiringPi.git
- Transducer to run high current through nichrome wire
  - 5V GPIO to gate
  - Battery + to drain
  - Battery - to Source

OOP Advantage:

- Easy call in main
- Simple, clear function



Servo Motors



SERVO

+ SERVO()
+ SERVO(int fd, unsigned char channel, unsigned short target)
+ ~SERVO()

- maestroGetPosition(int fd, unsigned char channel) : int
- maestroSetPosition(int fd, unsigned char channel, unsigned short target) : int

```
//**********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);
}
```

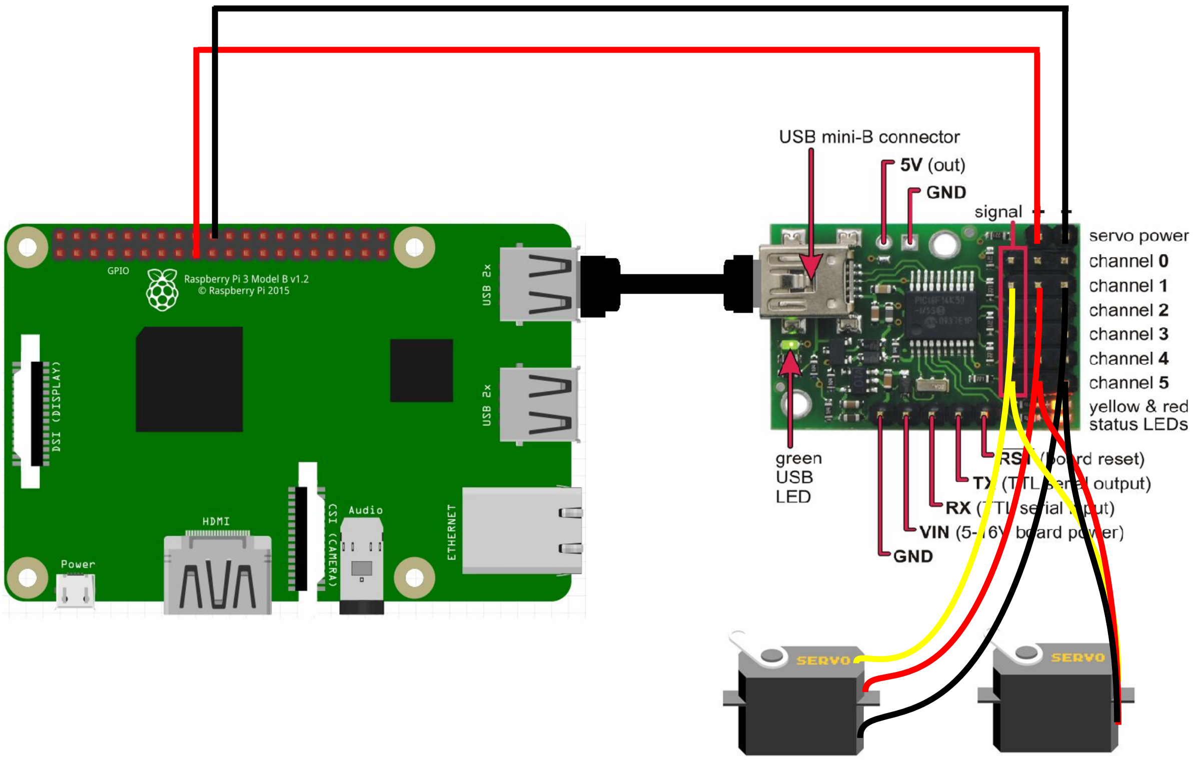
Use of open source Pololu code:

- Call to USB port microcontroller is connected to
- https://www.pololu.com/docs/0J40/5.h.1

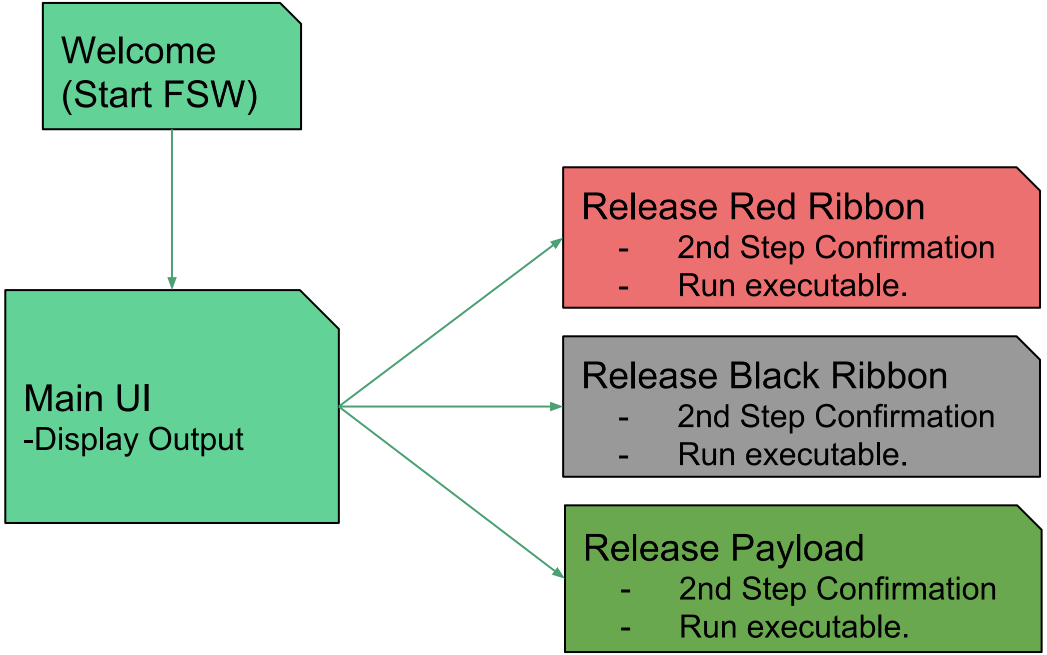
OOP Advantage:

- Switchable channel call in main
- Position setting in main
- Ability to call more than one servo

Servo - Pololu - Pi Wire Schematic

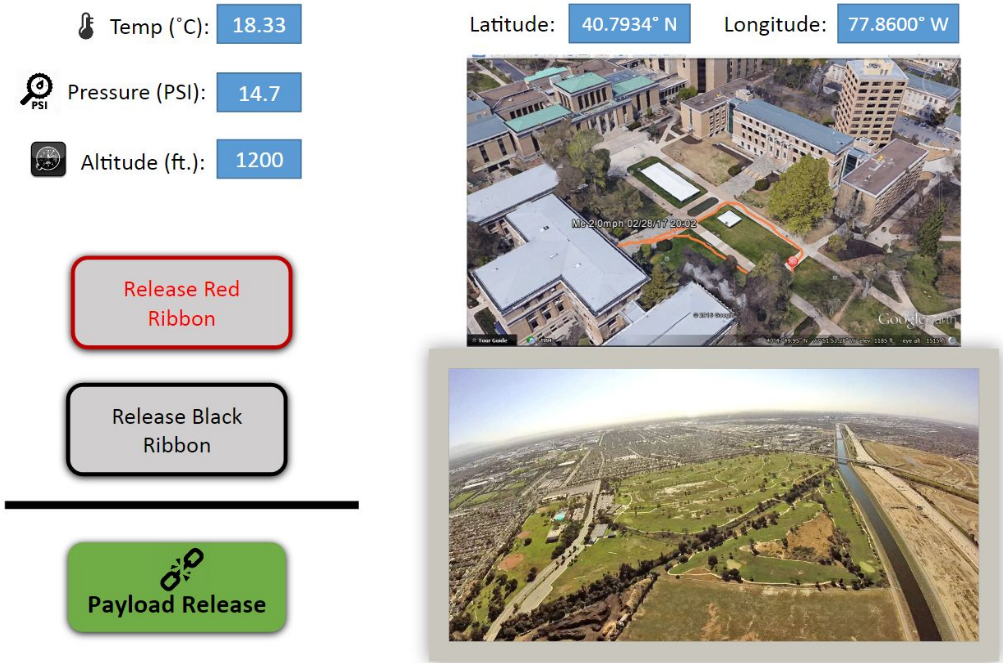


User Interface



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

User Interface Before Development



Temp (˚C): 18.33

Pressure (PSI): 14.7

Altitude (ft.): 1200

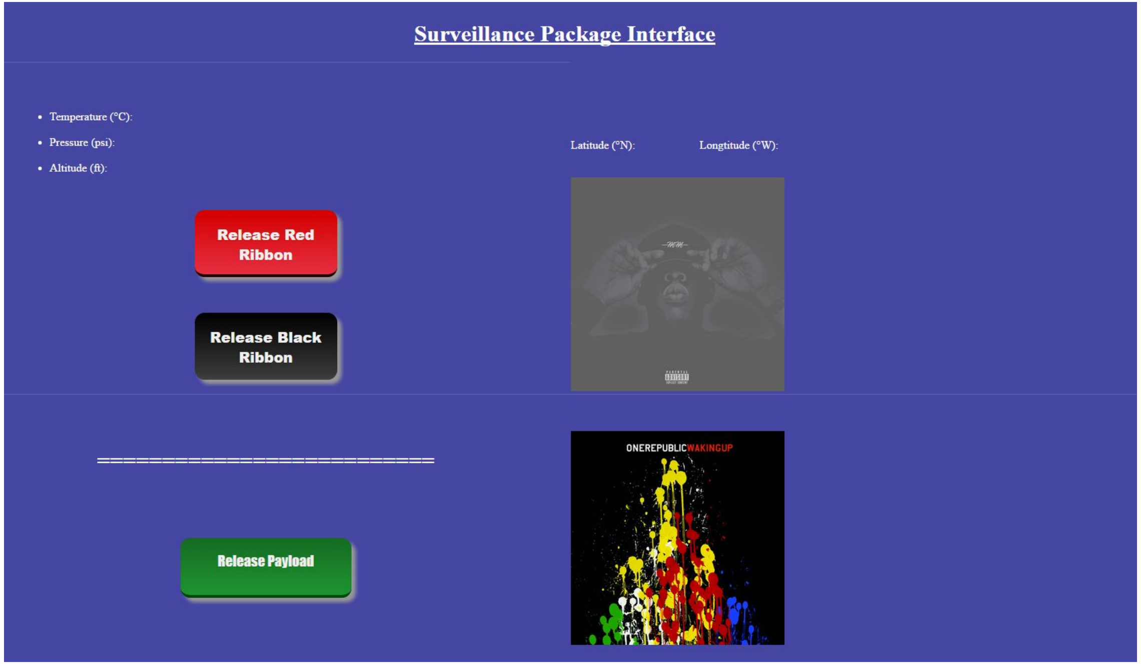
Latitude: 40.7934˚ N

Longitude: 77.8600˚ W

User Interface After Development (Tentative)

Features:

- Written as a webpage in HTML & CSS
- Displays temperature, pressure, altitude, latitude, and longitude.
- Three buttons for redundancy that are linked to executables.
- Displays a map with the package's location.
- Displays images captured with RasPi Camera.
- Refreshes automatically every 5 seconds.



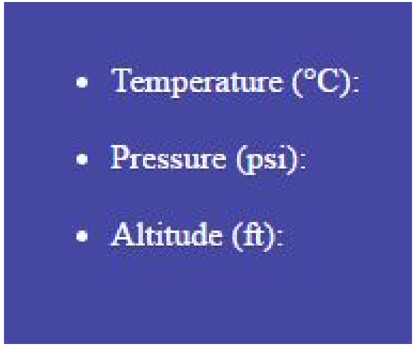
Surveillance Package Interface

- Temperature (°C):
- Pressure (psi):
- Altitude (ft):

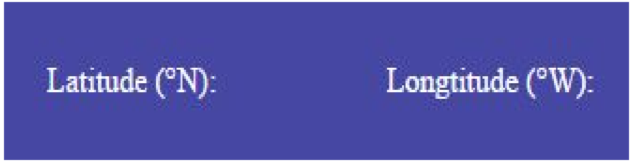
Latitude (°N): Longtitude (°W):

Data

UI

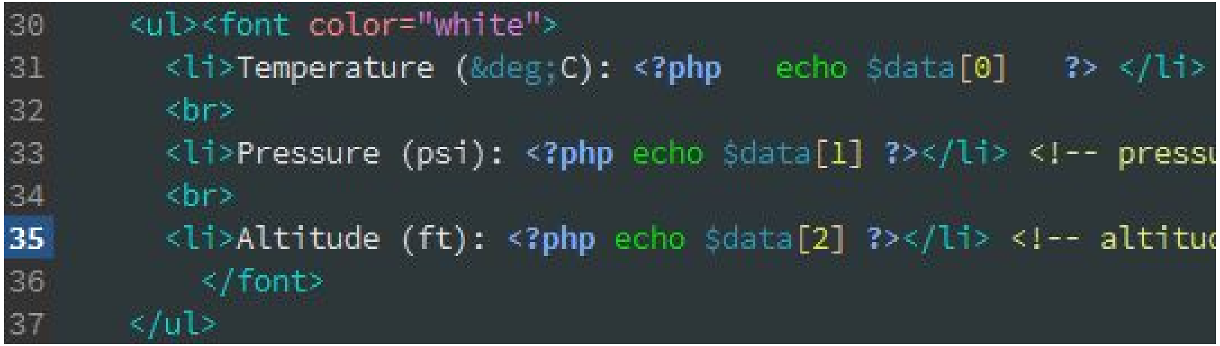


- Temperature (°C):
- Pressure (psi):
- Altitude (ft):



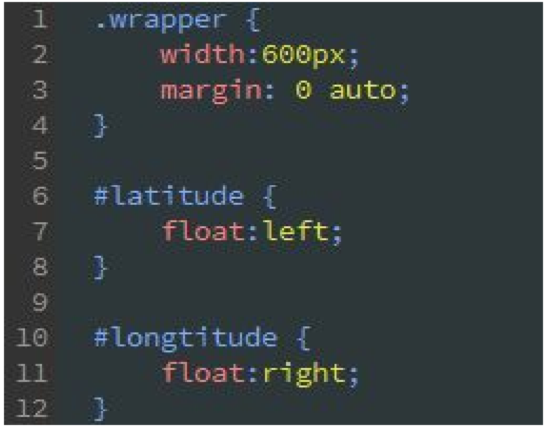
Latitude (°N): Longtitude (°W):

HTML



```
    <ul><font color="white">
      <li>Temperature (&deg;C): <?php   echo $data[0]   ?> </li>
      <br>
      <li>Pressure (psi): <?php echo $data[1] ?></li> <!-- pressu
      <br>
      <li>Altitude (ft): <?php echo $data[2] ?></li> <!-- altitu
        </font>
    </ul>
```

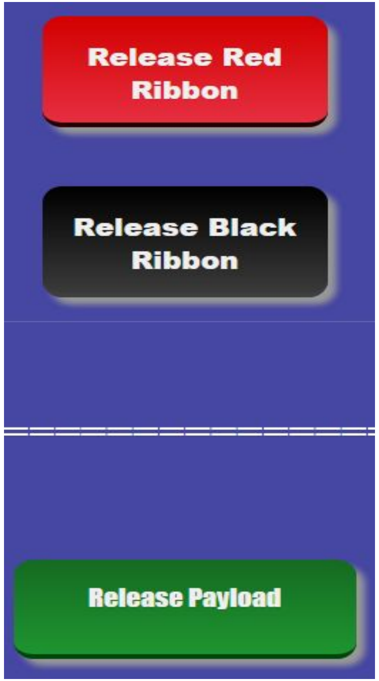
CSS



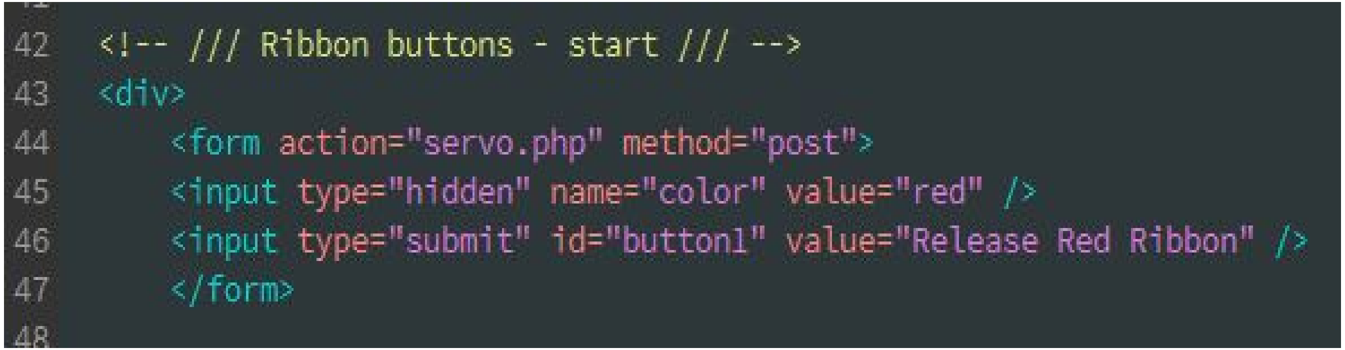
```
.wrapper {
    width:600px;
    margin: 0 auto;
}

#latitude {
    float:left;
}

#longtitude {
    float:right;
}
```

Data

UI

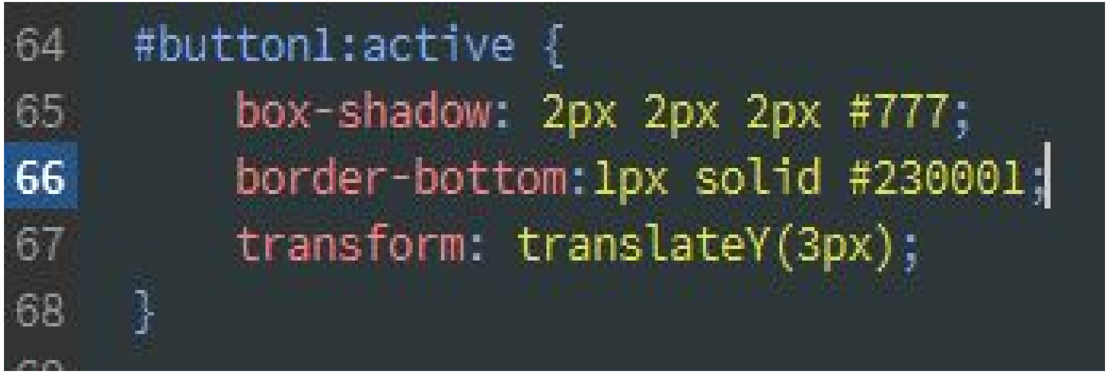


HTML



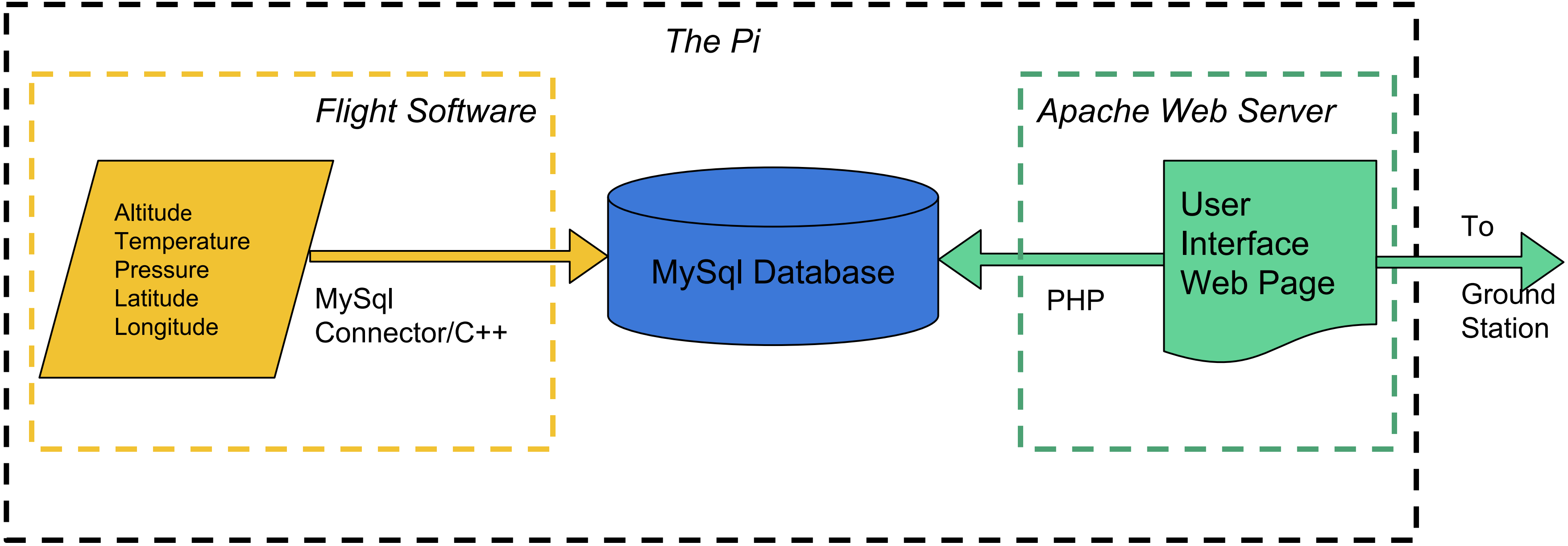
```
<!-- /// Ribbon buttons - start /// -->
<div>
    <form action="servo.php" method="post">
    <input type="hidden" name="color" value="red" />
    <input type="submit" id="button1" value="Release Red Ribbon" />
    </form>
```

CSS

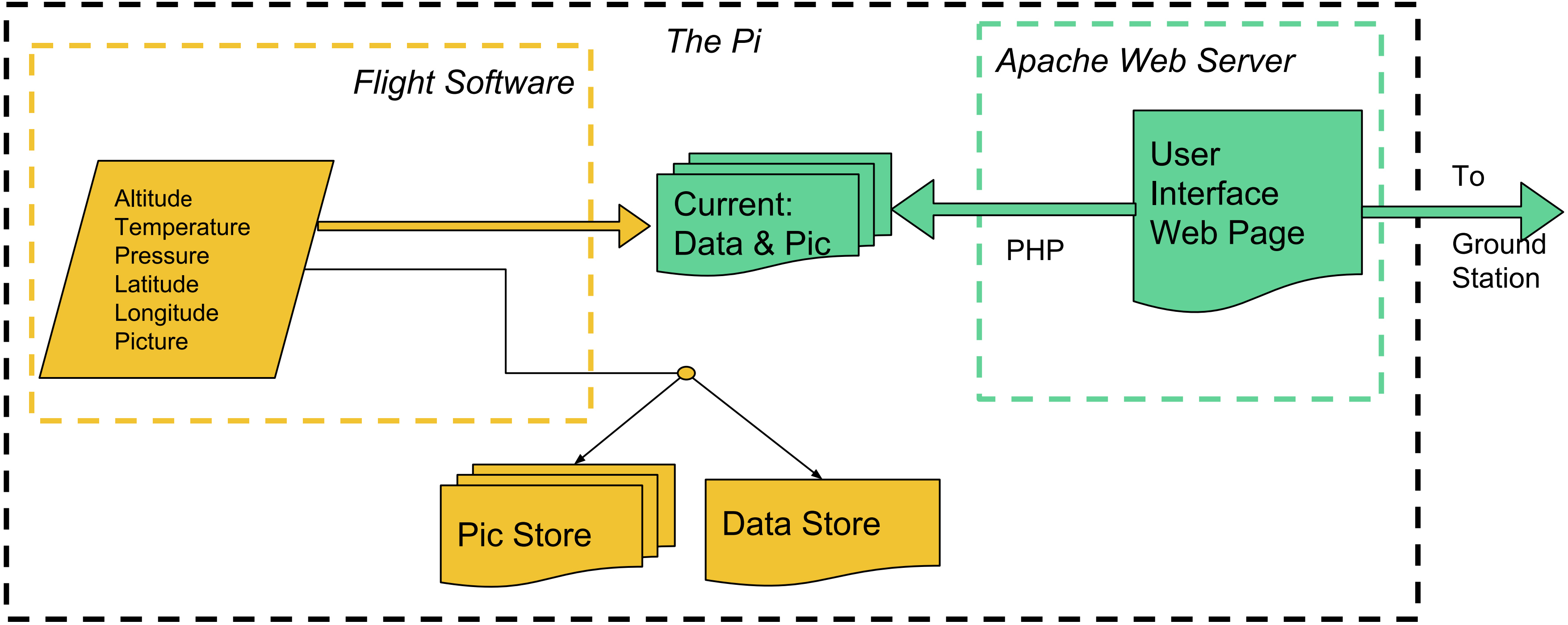


```
#button1:active {
        box-shadow: 2px 2px 2px #777;
        border-bottom:1px solid #230001;
        transform: translateY(3px);
}
```

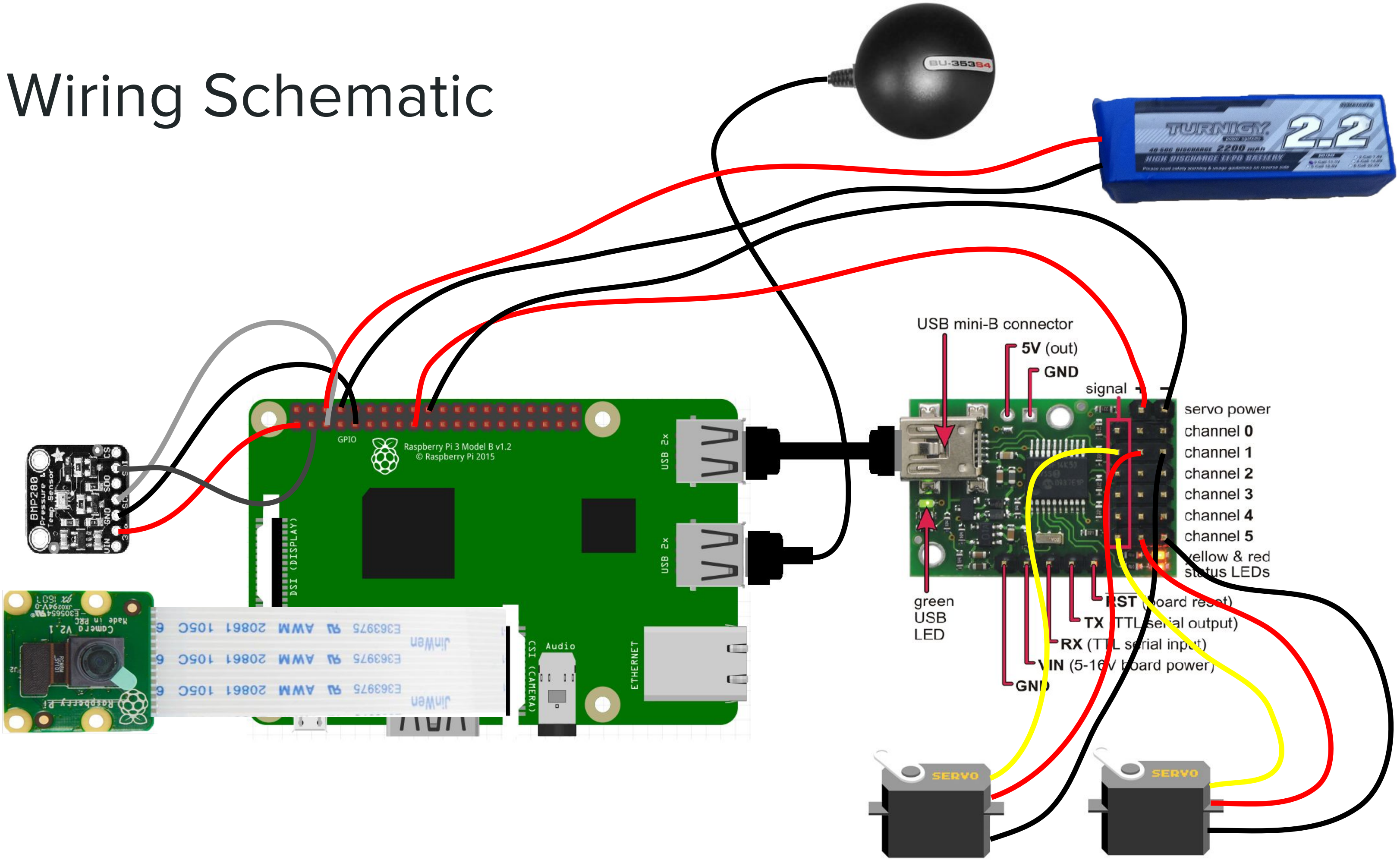
Database - Proposed/In-progress Implementation



Database - Current Implementation



Wiring Schematic



Using the Software

Startup:

- Just need IP Address for the Pi
- Welcome page
  - Start the main software
  - See any command line outputs during System Check
  - Navigate to main UI

Operation:

- Main UI
  - Display last current data values and image
  - Manually issue commands to release ribbons or payload

Guidance to Testing

GPS

- Determine margin of error for altitude readings

Camera

- Begin Unit Tests on Camera Class

Sensors

- Begin Unit Tests on Pressure and Temperature Classes

Hotwire

- Hardware Tests
- Unit test code with hardware

Servos

- Unit test Servo Class to set servo position
- Implement with ribbon deployment system

UI

- Access UI from PC
- Create small executables to be called from button clicks (i.e. stubs)

Questions?

010100010111010101100101011100110111010
00110100101101111011011100111001100111111