TITLE: Reading Data from the 10 DOF IMU

GOAL:

* To read data using I2C from multiple sensors on the IMU device
* To upload this data to a cloud based service online using the ESP8266 Wifi module

DELIVERABLES:

For this project, The data that will be obtained from the 10 Dof IMU will be: The acceleration along the x, y, and z axis, The Heading in which the device is facing along with the magnitude of the earth’s magnetic field, The rate at which the IMU is turning in each of three axes and the temperature and elevation of the IMU.

LITERATURE SURVEY:

This 10 Dof IMU is useful for recording the position of moving objects. For example, if the user was piloting a drone, knowing statistics such as the height of the drone and how fast it is turning may help with controlling the device. It can also be used to measure how quickly it is accelerating in any direction; this can help with knowing how to control the drone in areas where there is a high wind speed. Since the drone will normally accelerate in the direction following the wind, the pilot of the drone would know what to look out for and have an idea of how to maintain control of the device.

Based on the Project presentation materials and a quick literature survey - explain the need, contribution, and sustainability of your project.

COMPONENTS:

* Atmega328p
* Huzzah ESP8266 Wifi Module
* 10DOF IMU from DFRobot
  + Accelerometer – ADXL345
  + Digital Compass – HMC5883L
  + Barometer – BMP085
  + Gyroscope – ITG-3200
* 4-Channel Bi-Directional Level Shifter

The components used for this project were the Atmega328p microcontroller, the Huzzah ESP8266 Wifi module from Adafruit and the 10DOF IMU from DFRobot.

The BSS138 level shifter is used to safely communicate between the Atmega328p and the Wifi Module/10DOF IMU. This is because the Atmega328p outputs 5v for logic and these two modules operate safely at 3.3v. It should be noted that the BSS138 includes pull-up resistors which are necessary for proper communication between the 10DOF IMU and the Atmega328p when using I2C communication.

The Huzzah ESP8266 Wifi module was utilized to transfer the data received from the Atmega328p to the cloud. This wifi module operates at 3.3V for both the Rx and Tx pins and supplied power to it. However, Adafruit designed this module with voltage regulators on the power pins and level shifters on the Rx pins. This allows us to safely provide 5V for the Rx pins and up to 16V on the power pins. Due to this wifi module being the one from Adafruit, it comes preprogrammed with support for the LUA scripting language. This means that the device will boot up and run the items in the init.lua file saved onto it. Programming the device is possible through the use of a program called ESPlorer.

The 10DOF IMU from DFRobot operates at 3.3V for the SDA, SCL and Power pins. This device requires that any inputs provided to it be changed to 3.3V in order to avoid damaging the device. The device has an Accelerometer, A Compass, A Barometer, and a Gyro.

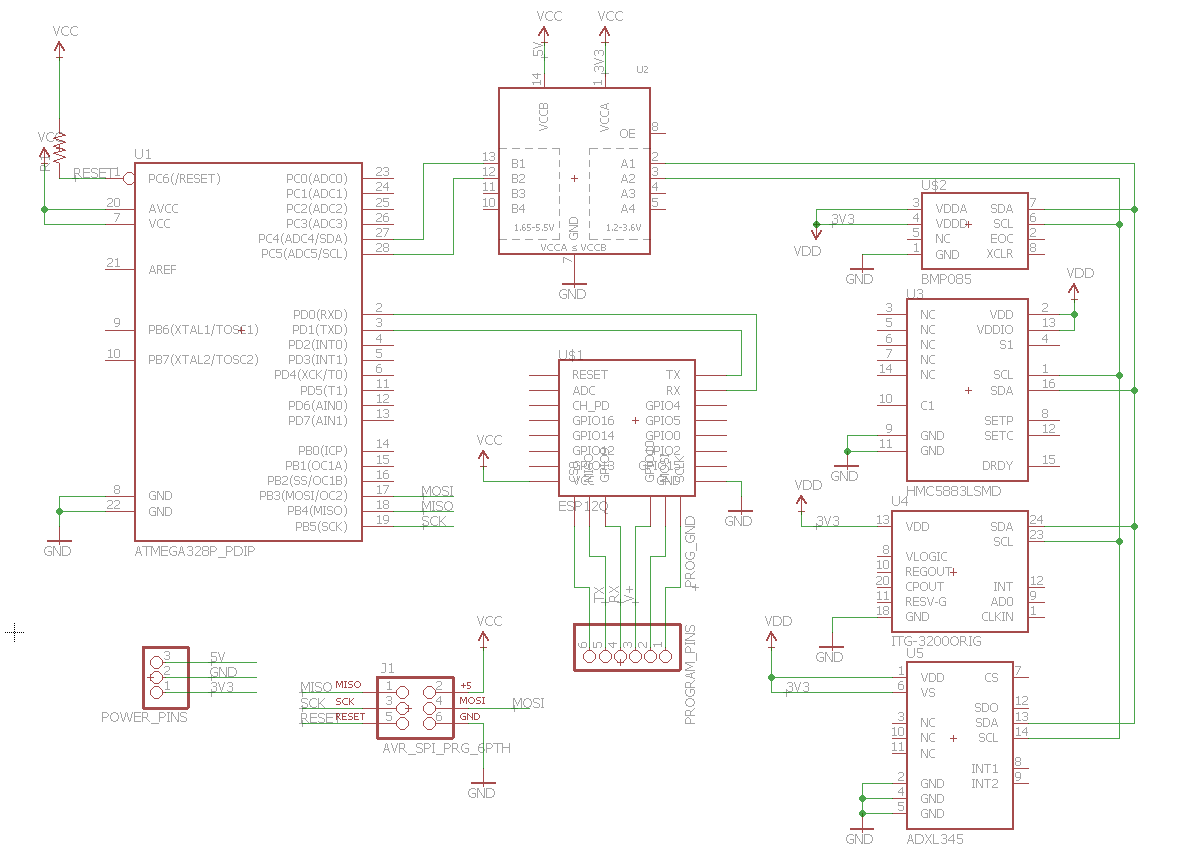
The accelerometer measures the acceleration in the X, Y and Z directions and outputs a number related to the G-Force. The G-force is a value that is related to the strength of the Earth’s gravitational field. 1g is the value of the Earth’s gravitational field and the values being read will be reported in terms of the G-force.

The Compass outputs the strength of the earth’s magnetic field in milligauss. This strength is measured in the X, Y, and Z direction. The heading which is the direction the compass is facing, can be obtained by using the strength of the magnetic field in the X, and Y directions.

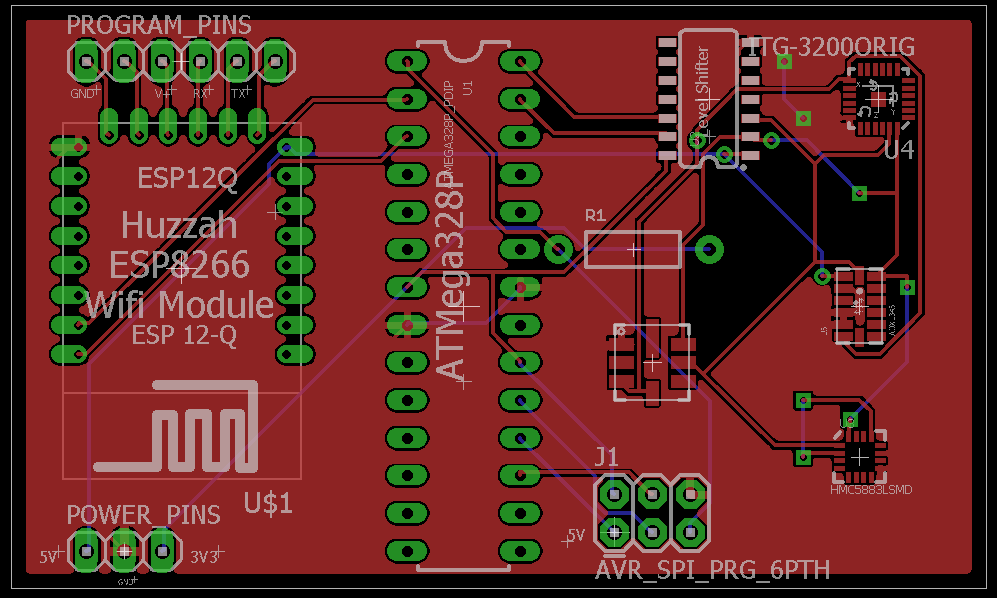
The Barometer reads both the temperature and pressure. The raw values read from the Barometer are uncalibrated and thus must be calibrated using the calibration values stored on the barometer. Once these values are calibrated, the temperature is given in degrees Celsius and the pressure is given in Pascals. It should be noted that the value for the temperature is 10 times larger than the actual temperature, so it becomes necessary to divide this value by 10 in order to obtain the actual temperature.

The Gyroscope measures the rotation of the device in degrees per second. The values range from +-2000 degrees per second and are given in the X, Y and Z directions.

SCHEMATICS: (exception - include image)



INITIAL PCB: (exception - include image)



IMPLEMENTATION:

* Obtain an ESP8266 wifi module, Atmega328p and 10Dof IMU
* Register for a ThingSpeak
* Program the Huzzah ESP8266 to upload data to a ThingSpeak server whenever data is received through UART
* Program the Atmega328p to communicate to the 10Dof IMU and receive the data from each of the four sensors
* Program the Atmega328p to send these values serially to the wifi module

SNAPSHOTS/SCREENSHOTS:

Video of the Demo: <https://www.youtube.com/watch?v=KVI4KMxK6Yw>

Screenshots of Demo along with Screenshot of finished circuit:

<https://drive.google.com/folderview?id=0B74Hv5bsRp-1ekd1NjVrUkN2LU0&usp=sharing>

CODE: (with comments) [\*final code]

#define F\_CPU 16000000UL // Clock Speed

#include <avr/io.h>

#include <util/delay.h> // Needed for \_delay\_ms()

#include <stdlib.h>

// Necessary Libraries for UART

#include <stdint.h> // needed for uint8\_t

#include <stdio.h> // Needed for printf

#include <math.h> // Needed for arctan

#define BAUD 9600 // Setup the BUAD Rate

#define MYUBRR F\_CPU/16/BAUD -1 // Value being written into UBRR0 Register

// i2c functions

void i2c\_stop**();** // Function to stop i2c\_communication

void i2c\_write**(**unsigned char**);** // Function to send a value via i2c

void i2c\_start**(**void**);** // Function to start i2c\_communication

void i2c\_init**(**void**);** // Function to initialize the micrc controller for i2c

uint8\_t i2c\_ReadACK**(**void**);** // Function to read value via i2c with an acknowledgment

uint8\_t i2c\_ReadNACK**(**void**);** // Function to read value via i2c without an acknowledgment

// Compass Stuff

void i2c\_ReadCompass**(**void**);** // Function to read Heading from compass

void i2c\_CompassSetUp**(**void**);** // Function to initialize the compass

float heading**;** // float value to hold the heading

// Global Variables for Compass Functions

int Xval **=** 0**;**

int Yval **=** 0**;**

int Zval **=** 0**;**

// USART Functions

int USART0SendByte**(**char**,** FILE **\***stream**);** // Declaration of the Method to transmit char

int USART0ReceiveByte**(**FILE **\***stream**);** // Declaration of the Method to Receive a char

void USART0init**(**void**);** // Declaration of USART init function

void delay**();** // Method used to delay for 1s

FILE usart0\_str **=** FDEV\_SETUP\_STREAM**(**USART0SendByte**,** USART0ReceiveByte**,** \_FDEV\_SETUP\_RW**);**

// Accelerometer

void i2c\_ReadAccel**(**void**);** // Method to read values from Accelerometer

void i2c\_AccelSetUp**(**void**);** // Method to initialize the accelerometer

float X\_accel**,** Y\_accel**,** Z\_accel**;** // Converted Values from the Accelerometer

int X\_accel\_Raw**,** Y\_accel\_Raw**,** Z\_accel\_Raw**;** // Raw Values from the Accelerometer

// Barometer/ Temperature Sensor functions

void readBarometer**(**void**);** // Method to read the values from the barometer

void getBarometerCalibrations**(**void**);** // Method to read the calibration values from the Barometer

void readBarometerUP**(**void**);** // Method to read the uncalibrated pressure from the barometer

void readBarometerTemp**(**void**);** // Method to read the uncalibrated temperature from the barometer

void ConvertUP**(**void**);** // Method to convert the uncalibrated pressure to calibrated pressure

void convertTemp**(**void**);** // Method to convert uncalibrated temperature to calibrated temperature

// Calibration Values - These values are set via getBarometerCalibrations()

long oss**;**

long AC1**,** AC2**,** AC3**,** B1**,** B2**,** MB**,** MC**,** MD**;**

unsigned long AC4**,** AC5**,** AC6**;**

long X1**,** X2**,** X3**,** B3**,** B5**,** B6**;**

unsigned long B4**,** B7**;**

// User Values

long UP **=** 0**;** // Uncalibrated Pressure

long BaroTemp **=** 0**;** // Uncalibrated Temperature

float altitude **=** 0.0**;** // The elevation in feet

float TempC **=** 0.0**;** // The temperature in Celsius

float Std\_Pressure **=** 100380**;** // Pascals

// Gyro Functions

void Gyro\_init**(**void**);** // Method to initialize the Gyro

void Gyro\_read**(**void**);** // Method to read values from the Gyro

int X\_Gyro\_Raw**,** Y\_Gyro\_Raw**,** Z\_Gyro\_Raw**;** // Raw Values

float X\_Gyro**,** Y\_Gyro**,** Z\_Gyro**;** // Calibrated Values

void ConvertValues**(**void**);** // Converts raw values to usable real values

int main**(**void**)**

**{**

USART0init**();** // Enable the USART

stdin**=**stdout**=&**usart0\_str**;**

i2c\_init**();**

delay**();**

\_delay\_ms**(**100**);** // Allow the sensor to boot

// Initialize all the sensors

i2c\_CompassSetUp**();**

i2c\_AccelSetUp**();**

Gyro\_init**();**

\_delay\_ms**(**100**);** // Allow for modules to boot

**while** **(**1**)**

**{**

i2c\_ReadCompass**();**

i2c\_ReadAccel**();**

getBarometerCalibrations**();** // This must be done every time since values may change

readBarometer**();**

Gyro\_read**();**

ConvertValues**();**

// Send the values to the ESP8266

printf**(**"%.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %d\n"**,**

X\_accel**,** Y\_accel**,** Z\_accel**,** altitude**,** TempC**,** X\_Gyro**,** Y\_Gyro**,** Z\_Gyro**,** heading**,** Zval**);**

// Wait one second before reading new values

delay**();**

**}**

**}**

//i2c functions

void i2c\_stop**()**

**{**

TWCR **=** **(**1**<<**TWINT**)|(**1**<<**TWEN**)|(**1**<<**TWSTO**);**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void i2c\_write**(**unsigned char data**)**

**{**

TWDR **=** data**;**

TWCR **=** **(**1**<<**TWINT**)|(**1**<<**TWEN**);**

**while(!(**TWCR **&** **(**1**<<**TWINT**)));**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void i2c\_start**(**void**)**

**{**

TWCR **=** **(**1**<<** TWINT**)** **|** **(**1**<<**TWSTA**)** **|** **(**1**<<** TWEN**);**

**while** **(!(**TWCR **&** **(**1**<<**TWINT**)));**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void i2c\_init**(**void**)**

**{**

TWSR **=** 0x00**;** //set prescaler bits to 0

TWBR **=** 0x0C**;** //0x48; //SCL freq. is 100k for XTAL = 16M

TWCR **=** 0x04**;** //enable TWI module

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint8\_t i2c\_ReadACK**(**void**)**

**{**

TWCR **=** **(**1**<<**TWINT**)|(**1**<<**TWEN**)|(**1**<<**TWEA**);**

**while** **((**TWCR **&** **(**1**<<**TWINT**))** **==** 0**);** // Wait until acknowledment is received

**return** TWDR**;**

**}**

//read byte with NACK

uint8\_t i2c\_ReadNACK**(**void**)**

**{**

TWCR **=** **(**1**<<**TWINT**)|(**1**<<**TWEN**);**

**while** **((**TWCR **&** **(**1**<<**TWINT**))** **==** 0**);**

**return** TWDR**;**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ---------------- Compass Functions -------------------------------

void i2c\_CompassSetUp**(**void**)**

**{**

i2c\_start**();**

i2c\_write**(**0x3C**);** // Address + Write bit of compass

i2c\_write**(**0x00**);** // Address of configuration register A

i2c\_write**(**0x70**);** // 15Hz rate with 8 Samples

i2c\_stop**();**

i2c\_start**();**

i2c\_write**(**0x3C**);** // Address + Write bit of compass

i2c\_write**(**0x01**);** // Address of configuration register B

i2c\_write**(**0xA0**);** // Gain = 5

i2c\_stop**();**

**}**

void i2c\_ReadCompass**(**void**)**

**{**

// Set up Single Measurement Mode

i2c\_start**();**

i2c\_write**(**0x3C**);** // Address + Write bit of compass

i2c\_write**(**0x02**);** // Address of configuration register B

i2c\_write**(**0x01**);** // Gain = 5

i2c\_stop**();**

\_delay\_ms**(**10**);** // Delay for at least 6 ms

i2c\_start**();**

i2c\_write**(**0x3D**);** // Address + Read bit

// Read the values

// Address pointer automatically increments after

// every read

Xval **=** i2c\_ReadACK**()** **<<** 8**;**

Xval **|=** i2c\_ReadACK**();**

Zval **=** i2c\_ReadACK**()** **<<** 8**;**

Zval **|=** i2c\_ReadACK**();**

Yval **=** i2c\_ReadACK**()** **<<** 8**;**

Yval **|=** i2c\_ReadNACK**();**

i2c\_stop**();**

**}**

// ------------------- USART Functions ------------------------------

// Sends a character via USART

int USART0SendByte**(**char u8Data**,** FILE **\***stream**)**

**{**

**if(**u8Data **==** '\n'**)**

**{**

USART0SendByte**(**'\r'**,** 0**);**

**}**

//wait while previous byte is completed

**while(!(**UCSR0A**&(**1**<<**UDRE0**))){};**

// Transmit data

UDR0 **=** u8Data**;**

**return** 0**;**

**}**

// Receives a character via USART

int USART0ReceiveByte**(**FILE **\***stream**)**

**{**

uint8\_t u8Data**;**

// Wait for byte to be received

**while(!(**UCSR0A**&(**1**<<**RXC0**))){};**

u8Data**=**UDR0**;**

//echo input data

USART0SendByte**(**u8Data**,**stream**);**

// Return received data

**return** u8Data**;**

**}**

void USART0init**(**void**)**

**{**

/\*Set baud rate \*/

UBRR0L **=** MYUBRR**;**

UCSR0B **|=** **(**1 **<<** TXEN0**);** // Enable transmitter

UCSR0C **|=** **(**1 **<<** UCSZ01**)** **|** **(**1 **<<** UCSZ00**);** // Set frame: 8-bit data

**}**

// Function delays for 1 second

void delay**(){**

**for(**int i **=** 500**;** i **>** 0**;** i**--)**

\_delay\_ms**(**2**);**

**}**

void i2c\_ReadAccel**(**void**)**

**{**

////// X /////////

i2c\_start**();**

i2c\_write**(**0xA6**);** // Address to write to Accelo

i2c\_write**(**0x33**);** // X msb Address

i2c\_start**();**

i2c\_write**(**0xA7**);** // Address to read from Accelo

X\_accel\_Raw **=** i2c\_ReadACK**()** **<<** 8**;**

i2c\_start**();**

i2c\_write**(**0xA6**);** // Address to write to Accelo

i2c\_write**(**0x32**);** // X lsb Address

i2c\_start**();**

i2c\_write**(**0xA7**);** // Address to read from Accelo

X\_accel\_Raw **|=** i2c\_ReadACK**();**

////// Y ////////

i2c\_start**();**

i2c\_write**(**0xA6**);** // Address to write to Accelo

i2c\_write**(**0x35**);** // Y msb Address

i2c\_start**();**

i2c\_write**(**0xA7**);** // Address to read from Accelo

Y\_accel\_Raw **=** i2c\_ReadACK**()** **<<** 8**;**

i2c\_start**();**

i2c\_write**(**0xA6**);** // Address to write to Accelo

i2c\_write**(**0x34**);** // Y lsb Address

i2c\_start**();**

i2c\_write**(**0xA7**);** // Address to read from Accelo

Y\_accel\_Raw **|=** i2c\_ReadACK**();**

////// Z ////////

i2c\_start**();**

i2c\_write**(**0xA6**);** // Address to write to Accelo

i2c\_write**(**0x37**);** // Y msb Address

i2c\_start**();**

i2c\_write**(**0xA7**);** // Address to read from Accelo

Z\_accel\_Raw **=** i2c\_ReadACK**()** **<<** 8**;**

i2c\_start**();**

i2c\_write**(**0xA6**);** // Address to write to Accelo

i2c\_write**(**0x36**);** // Y lsb Address

i2c\_start**();**

i2c\_write**(**0xA7**);** // Address to read from Accelo

Z\_accel\_Raw **|=** i2c\_ReadNACK**();**

i2c\_stop**();**

**}**

void i2c\_AccelSetUp**(**void**)**

**{**

i2c\_start**();**

i2c\_write**(**0xA6**);**

i2c\_write**(**0x2D**);** // Power Ctl register

i2c\_write**(**0x08**);** // Enable measurement mode

i2c\_stop**();**

**}**

// This function converts the raw pressure read from the barometer to

// the actual pressure value. These conversions were obtained from

// the datasheet

void ConvertUP**(**void**)**

**{**

B6 **=** B5 **-** 4000**;**

X1 **=** B6**\***B6**;**

X1 **=** X1 **/** 4096**;**

X1 **=** X1 **\*** B2**;**

X1 **=** X1 **/** 2048**;**

X2 **=** AC2 **\*** B6**;**

X2 **=** X2 **/** 2048**;**

X3 **=** X1 **+** X2**;**

B3 **=** AC1 **\*** 4**;**

B3 **=** B3 **+** X3**;**

B3 **=** B3 **<<** oss**;**

B3 **=** B3 **+** 2**;**

B3 **=** B3 **/** 4**;**

X1 **=** AC3 **\*** B6**;**

X1 **=** X1 **/** 8192**;**

X2 **=** B6 **\*** B6**;**

X2 **=** X2 **/** 4096**;**

X2 **=** X2 **\*** B1**;**

X2 **=** X2 **/** 65536**;**

X3 **=** X1 **+** X2**;**

X3 **=** X3 **+** 2**;**

X3 **=** X3 **/** 4**;**

B4 **=** **((**unsigned long**)(**X3 **+** 32768**));**

B4 **=** B4 **\*** AC4**;**

B4 **=** B4 **/** 32768**;**

B7 **=** **(**unsigned long**)**UP **-** B3**;**

B7 **=** B7 **\*** **(**50000 **>>** oss**);**

**if(**B7 **<** 0x80000000**)** **{**

UP **=** B7 **\*** 2**;**

UP **=** UP **/** B4**;**

**}**

**else** **{**

X1 **=** UP**/**256**;**

X2 **=** X1**;**

X1 **=** X1 **\*** X2**;**

X1 **=** X1 **\*** 3038**;**

X1 **=** X1 **/** 65536**;**

X2 **=** **(-**7357 **\*** UP**);**

X2 **=** X2 **/** 65536**;**

X3 **=** X1 **+** X2 **+** 3791**;**

X3 **=** X3 **/** 4**;**

UP **=** UP **+** X3**;**

**}**

**}**

// Reads the Raw Barometer Pressure and stores it into UP

void readBarometerUP**(**void**)**

**{**

// Set up for UP Sampling...

i2c\_start**();**

i2c\_write**(**0xEE**);** // Barometer address + write

i2c\_write**(**0xF4**);** // Config register for temp/UP

i2c\_write**(**0x34**);** // Oversampling Resolution 0 \*4.5 ms

i2c\_stop**();**

oss **=** 0**;**

\_delay\_ms**(**10**);** // Delay at least 4.5 ms for UP reading

// Begin Reading...

i2c\_start**();**

i2c\_write**(**0xEE**);** // Barometer address + write

i2c\_write**(**0xF6**);** // MSB of data register

i2c\_start**();** // Restart

i2c\_write**(**0xEF**);** // Barometer address + Read

UP **=** i2c\_ReadACK**();**

UP **=** UP **<<** 8**;**

UP **=** UP **+** i2c\_ReadACK**();**

UP **=** UP **<<** 8**;**

UP **=** UP **+** i2c\_ReadNACK**();**

UP **=** UP **>>** **(**8 **-** oss**);**

i2c\_stop**();**

**}**

// Obtains all the Calibration values needed for converting the raw

// values.

void getBarometerCalibrations**(**void**)**

**{**

i2c\_start**();**

i2c\_write**(**0xEE**);** // Barometer Address + write

i2c\_write**(**0xAA**);** // go to MSB of AC1

i2c\_start**();** // Restart

i2c\_write**(**0xEF**);** // Barometer Address + Read

AC1 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

AC1 **|=** i2c\_ReadACK**();** // LSB

AC2 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

AC2 **|=** i2c\_ReadACK**();** // LSB

AC3 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

AC3 **|=** i2c\_ReadACK**();** // LSB

AC4 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

AC4 **|=** i2c\_ReadACK**();** // LSB

AC5 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

AC5 **|=** i2c\_ReadACK**();** // LSB

AC6 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

AC6 **|=** i2c\_ReadACK**();** // LSB

B1 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

B1 **|=** i2c\_ReadACK**();** // LSB

B2 **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

B2 **|=** i2c\_ReadACK**();** // LSB

MB **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

MB **|=** i2c\_ReadACK**();** // LSB

MC **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

MC **|=** i2c\_ReadACK**();** // LSB

MD **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

MD **|=** i2c\_ReadNACK**();** // LSB

i2c\_stop**();**

**}**

// Reads temperature

void readBarometerTemp**(**void**)**

**{**

// Set up for UP Sampling...

i2c\_start**();**

i2c\_write**(**0xEE**);** // Barometer address + write

i2c\_write**(**0xF4**);** // Config register for temp/UP

i2c\_write**(**0x2E**);** // Temperature \*4.5ms time

i2c\_stop**();**

\_delay\_ms**(**7**);** // Delay at least 4.5ms for temp reading...

// Begin Reading...

i2c\_start**();**

i2c\_write**(**0xEE**);** // Barometer address + write

i2c\_write**(**0xF6**);** // MSB of data register

i2c\_start**();** // Restart

i2c\_write**(**0xEF**);** // Barometer address + Read

BaroTemp **=** i2c\_ReadACK**()** **<<** 8**;** // MSB

BaroTemp**|=** i2c\_ReadNACK**();** // LSB

i2c\_stop**();**

**}**

// This function converts the raw temperature read from the barometer

// into a temperature displayed in degrees Celsius. This conversion was

// obtained from the datasheet for the barometer.

void convertTemp**(**void**)**

**{**

X1 **=** **(**BaroTemp **-** AC6**);**

X1 **=** X1 **\*** AC5**;**

X1 **=** X1 **/** 32768**;**

X2 **=** MC **\*** 2048**;**

X3 **=** X1 **+** MD**;**

X2 **=** X2 **/** X3**;**

B5 **=** X1 **+** X2**;**

BaroTemp **=** **(**B5 **+** 8**);**

BaroTemp **=** BaroTemp**/**16**;**

**}**

// Initializes the Gyro to sample data.

// The range of values sampled is +- 2000 degrees

void Gyro\_init**(**void**)**

**{**

i2c\_start**();**

i2c\_write**(**0xD0**);** // +- 2000 degrees/sec

i2c\_write**(**0x16**);**

i2c\_write**(**0x18**);**

i2c\_stop**();**

**}**

void Gyro\_read**(**void**)**

**{**

i2c\_start**();**

i2c\_write**(**0xD0**);** // Write

i2c\_write**(**0x1D**);** // Address of X MSB

i2c\_start**();**

i2c\_write**(**0xD1**);**

// Read the raw values into their respective variables

// The address pointer increments automatically after

// each read.

X\_Gyro\_Raw **=** i2c\_ReadACK**()** **<<** 8**;**

X\_Gyro\_Raw **|=** i2c\_ReadACK**();**

Y\_Gyro\_Raw **=** i2c\_ReadACK**()** **<<** 8**;**

Y\_Gyro\_Raw **|=** i2c\_ReadACK**();**

Z\_Gyro\_Raw **=** i2c\_ReadACK**()** **<<** 8**;**

Z\_Gyro\_Raw **|=** i2c\_ReadNACK**();**

i2c\_stop**();**

**}**

// Function to read the values from the barometer

// The values must be read in this order since some of the

// temperature values are used in the barometer conversion.

void readBarometer**(**void**)**

**{**

readBarometerTemp**();**

readBarometerUP**();**

**}**

// Converts all the raw values into tangible data that can be used.

void ConvertValues**(**void**)**

**{**

// Heading from compass

heading **=** atan2**((**double**)**Yval**,** **(**double**)**Xval**);** // Returns a value in radians

// Next three lines convert that radian value to a degree value more

// commonly seen on a compass

heading **=** heading **\*** 180**;**

heading **=** heading **/**3.14159**;**

heading **+=** 180**;**

// G force from accelerometer

// 0.004 is used since that is the value provided by the datasheet

// Each bit represents 0.004g

X\_accel **=** X\_accel\_Raw **\*** 0.004**;**

Y\_accel **=** Y\_accel\_Raw **\*** 0.004**;**

Z\_accel **=** Z\_accel\_Raw **\*** 0.004**;**

// Convert Barometer Values

// Values must be converted in this order since the pressure relies

// on a few temperature variables.

convertTemp**();**

ConvertUP**();**

// BaroTemp is 10x larger than actual Celsius value

TempC **=** BaroTemp **/** 10.0**;**

// Altitude formula

altitude **=** pow**(**UP**/** Std\_Pressure**,** 0.190284**);**

altitude **=** 1 **-** altitude**;**

altitude **=** altitude **\*** 44330**;**

altitude **=** altitude **\*** 3.28084**;**

// Gyro Values

// Value to divide by is given in the datsheet for the conversion

X\_Gyro **=** X\_Gyro\_Raw **/** 14.375**;**

Y\_Gyro **=** Y\_Gyro\_Raw **/** 14.375**;**

Z\_Gyro **=** Z\_Gyro\_Raw **/** 14.375**;**

**}**

**LUA SCRIPT**

-- Connect to an Access Point

wifi.setmode**(**wifi.STATION**);**

-- In this case I set up my phone as a wifi hotspot

wifi.sta.config**(**"Nexus 5" **,**"tatertime"**);**

-- Variables

X\_acc **=** 0**;**

Y\_acc **=** 0**;**

Z\_acc **=** 0**;**

alt **=** 0**;**

TmpC **=** 0**;**

X\_Gy **=** 0**;**

Y\_Gy **=** 0**;**

Z\_Gy **=** 0**;**

Heading **=** 0**;**

Z\_Comp **=** 0**;**

-- Code to initalize the UART on the ESP8266 also disables the interpreter

-- This is necessary to ensure proper reading of the received values

-- Code compares the received values whenever a linefeed character (\n) is received

-- These values are then stored in their respective variables

uart.on**(**"data"**,** "\n"**,**

**function(**data**)**

**if** **(string.match(**data**,** "quit"**))** **then**

**print(**"Quitting..."**)**

uart.on**(**"data"**)**

**end**

X\_acc**,** Y\_acc**,** Z\_acc**,** alt**,** TmpC**,** X\_Gy**,** Y\_Gy**,** Z\_Gy**,** Heading**,** Z\_Comp **=** data**:match(**"(-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+.%d+) (-\*%d+)"**)**

**end,**

0**)**

-- Function to post the received data to thingspeak

-- Note the code is copied twice in this block, each set is to send

-- data to each webpage. Since thingspeak can only support up to eight

-- sets of data on each page, it becomes necessary for us to open and close

-- a connection to each webpage we are going to be posting the data to

**function** postThingSpeak**(**level**)**

connout **=** **nil**

connout **=** net.createConnection**(**net.TCP**,** 0**)**

-- If the data posts correctly, output that the data posted correctly

-- to the UART

-- Needed mostly for debugging since that is the only time you will

-- have the ESP8266 connected to a PC

connout**:**on**(**"receive"**,** **function(**connout**,** payloadout**)**

**if** **(string.find(**payloadout**,** "Status: 200 OK"**)** **~=** **nil)** **then**

**print(**"Posted Set 1 OK"**);**

**end**

**end)**

-- Open the connection and post the data

connout**:**on**(**"connection"**,** **function(**connout**,** payloadout**)**

**print** **(**"Posting Set 1..."**);**

connout**:**send**(**"GET /update?api\_key=571L2JBQYIO43484&field1=" **..** X\_acc

**..** "GET /update?api\_key=571L2JBQYIO43484&field2=" **..** Y\_acc

**..** "GET /update?api\_key=571L2JBQYIO43484&field3=" **..** Z\_acc

**..** "GET /update?api\_key=571L2JBQYIO43484&field4=" **..** alt

**..** "GET /update?api\_key=571L2JBQYIO43484&field5=" **..** TmpC

**..** "GET /update?api\_key=571L2JBQYIO43484&field6=" **..** X\_Gy

**..** "GET /update?api\_key=571L2JBQYIO43484&field7=" **..** Y\_Gy

**..** "GET /update?api\_key=571L2JBQYIO43484&field8=" **..** Z\_Gy

**..** " HTTP/1.1\r\n"

**..** "Host: api.thingspeak.com\r\n"

**..** "Connection: close\r\n"

**..** "Accept: \*/\*\r\n"

**..** "User-Agent: Mozilla/4.0 (compatible; esp8266 Lua; Windows NT 5.1)\r\n"

**..** "\r\n"**)**

**end)**

-- Close the connection

connout**:**on**(**"disconnection"**,** **function(**connout**,** payloadout**)**

connout**:*close*();**

**collectgarbage();**

**end)**

-- Address of where the device is connecting

connout**:**connect**(**80**,**'api.thingspeak.com'**)**

-- Reinitialize all the values for the second batch of data values

connout **=** **nil**

connout **=** net.createConnection**(**net.TCP**,** 0**)**

connout**:**on**(**"receive"**,** **function(**connout**,** payloadout**)**

**if** **(string.find(**payloadout**,** "Status: 200 OK"**)** **~=** **nil)** **then**

**print(**"Posted Set 2 OK"**);**

**end**

**end)**

-- Open the connection to the second webpage and post the other two values

connout**:**on**(**"connection"**,** **function(**connout**,** payloadout**)**

**print** **(**"Posting Set 2..."**);**

connout**:**send**(**"GET /update?api\_key=TSE4R56WPOW0A4TM&field1=" **..** Heading

**..** "GET /update?api\_key=TSE4R56WPOW0A4TM&field2=" **..** Z\_Comp

**..** " HTTP/1.1\r\n"

**..** "Host: api.thingspeak.com\r\n"

**..** "Connection: close\r\n"

**..** "Accept: \*/\*\r\n"

**..** "User-Agent: Mozilla/4.0 (compatible; esp8266 Lua; Windows NT 5.1)\r\n"

**..** "\r\n"**)**

**end)**

-- Disconnect from the webpage

connout**:**on**(**"disconnection"**,** **function(**connout**,** payloadout**)**

connout**:*close*();**

**collectgarbage();**

**end)**

connout**:**connect**(**80**,**'api.thingspeak.com'**)**

**end**

-- Run the above function on a loop that happens every 10 seconds

-- Thingspeak only accepts values at around this frequency so this time

-- period is acceptable

tmr.alarm**(**1**,** 10000**,** 1**,** **function()** postThingSpeak**(**0**)** **end)**

REFERENCES:

10DOF IMU Webpage [http://www.dfrobot.com/index.php?route=product/product&product\_id=818#.VzJogvkrKUk](http://www.dfrobot.com/index.php?route=product/product&product_id=818%23.VzJogvkrKUk)

Referenced LUA Script

<http://captain-slow.dk/2015/04/16/posting-to-thingspeak-with-esp8266-and-nodemcu/>