Computer Science Senior Software Engineering Project (CS462)
Winter 2017

MIDTERM PROGRESS REPORT
FEBRUARY 17, 2017

HEAD-UP DISPLAY ALIGNMENT SYSTEM

Group 65:
KRISNA IRAWAN
JIONGCHENG LUO
DREW HAMM

Supervised by:
ROCKWELL COLLINS, INC.

Abstract

TODO: This is a progress report that keeps track of our project progress for the first half of winter term. The document consists of three main sections that each section reflecting each individual group member. The report mainly introduces our current stage of the project, remaining work, the problems we met and their solutions, as well as some pieces of coding and experimental design we have been working on.

I. Signatures

Client	Date
Author 1	Date
Author 2	Date
Author 3	Date

Contents

II. Jiongcheng Luo (Roger)

A. Purpose

The traditional methodology for an airplane Head-UP Display (HUD) to provide flight information is by obtaining data from a mounted device call Inertial Reference Unit on the aircraft. This method outputs precise and aligned data to the HUD through a mechanical alignment system. Yet, the industrial production of this mechanical alignment system is time consuming and costly. In addition, the current system does not compensate for airframe droop during flight. The purpose of this project is to develop a demonstration system to prove that there is such an algorithm or mathematical procedure that can generate precise and aligned data with reduced installation cost utilizing the data from multiple inexpensive IMUs. This algorithm can handle multiple groups of data in near real-time and the outcome (aligned-data) of this algorithm will compensate the alignment error correctly, and the alignment error should be within a range of one milli-radian.

B. Current Stage

Currently, we have completed the hardware setup that using a Metro Mini microcontroller and MPU-9250 IMU, the microcontroller and IMU are hooked up by using I2C protocol, and we can now use Arduino IDE for programming. From the perspective of software, we are able to recognize the address of connected IMU and read the output data from all three sensors of the MPU-9250 including acceleration (accelerator value), gyroscope values and magnetometer value. In addition, we discovered two different filtering algorithms for getting quaternion output based on the raw data of the sensor, which we can generate quaternion value by using sample code and existing libraries.

C. Remaining work

- Add multiple IMUs onto single microcontroller. We still need to figure out how to connect slave IMUs and select the correct address of expected IMU by using I2C protocol.
- Get precise error of all IMUs/ sensors
- Develop an alignment algorithm for aligning a group of quaternion data.
- Develop demonstration interface.

D. Problems and Solutions

- **Problem 1:** Knowing the actual error of the IMU/sensors. It is difficult to measure actual precise error of the sensor data by just looking at it output data since the error could be very minor and unstable.

  Solution: A solution to measure the error is by comparing the actual alter angle with the sensor output data. An actual alter angle can be calculated by using a laser pointer and a mechanical angle adjuster as figure 1 for assistant tool, as well as applying trigonometric algorithm. Look at experimental design description for more detail.

- **Problem 2:** A declination is a factor that would cause error to the yaw data in different region on the earth, the declination needs to be counteract before getting the real yaw data.

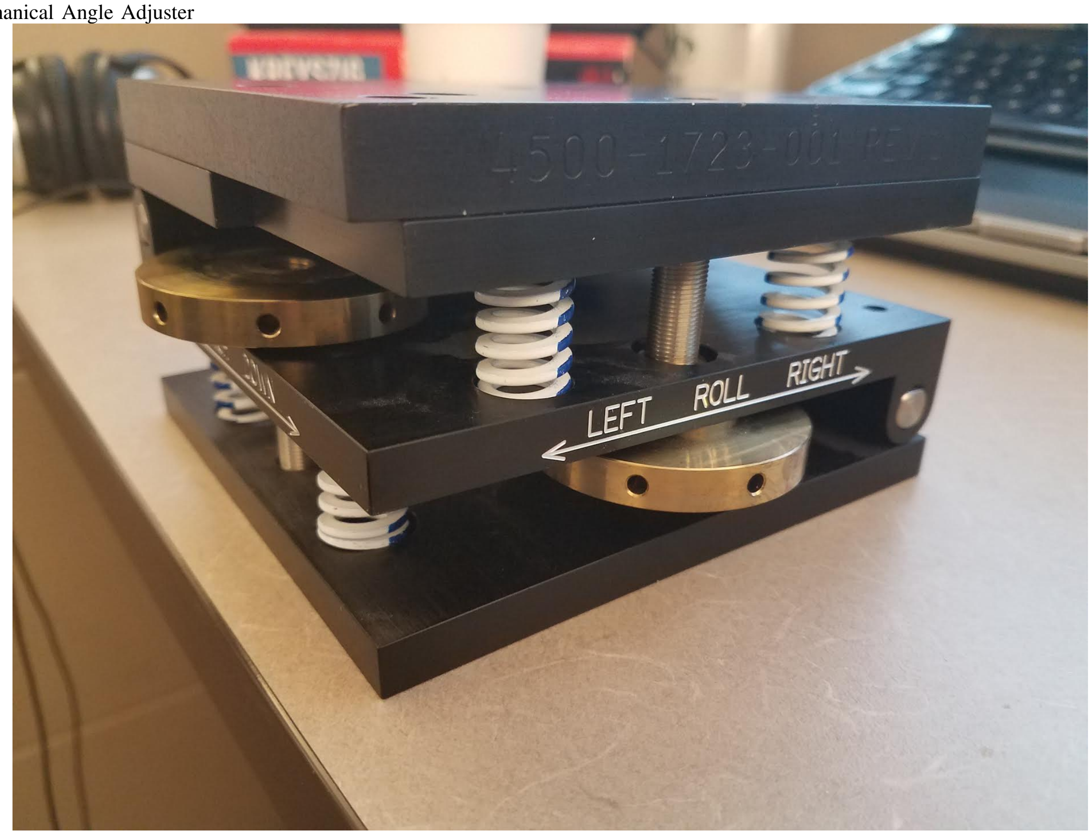
  Solution: A simple solution is to look up the declination value from NOAA (National Centers for Environmental Information) at this region and time. Adding/subtracting the value (in degree) to the raw yaw data.

E. Interesting Code

```
static Quaterniond toQuaternion(double pitch, double roll, double yaw)
{
        Quaterniond q;
        double t0 = cos(yaw * 0.5);
        double t1 = sin(yaw * 0.5);
        double t2 = cos(roll * 0.5);
        double t3 = sin(roll * 0.5);
        double t4 = cos(pitch * 0.5);
        double t5 = sin(pitch * 0.5);

        q.w() = t0 * t2 * t4 + t1 * t3 * t5;
        q.x() = t0 * t3 * t4 - t1 * t2 * t5;
        q.y() = t0 * t2 * t5 + t1 * t3 * t4;
        q.z() = t1 * t2 * t4 - t0 * t3 * t5;
        return q;
}
```

Fig. 1. Mechanical Angle Adjuster



This piece of code is using a mathematical algorithm that take the pitch roll yaw data to calculate quaternion. There are also other ways like Madgwick filter algorithm and Mahony fliter algorithm that can generate more precise quaternion values.

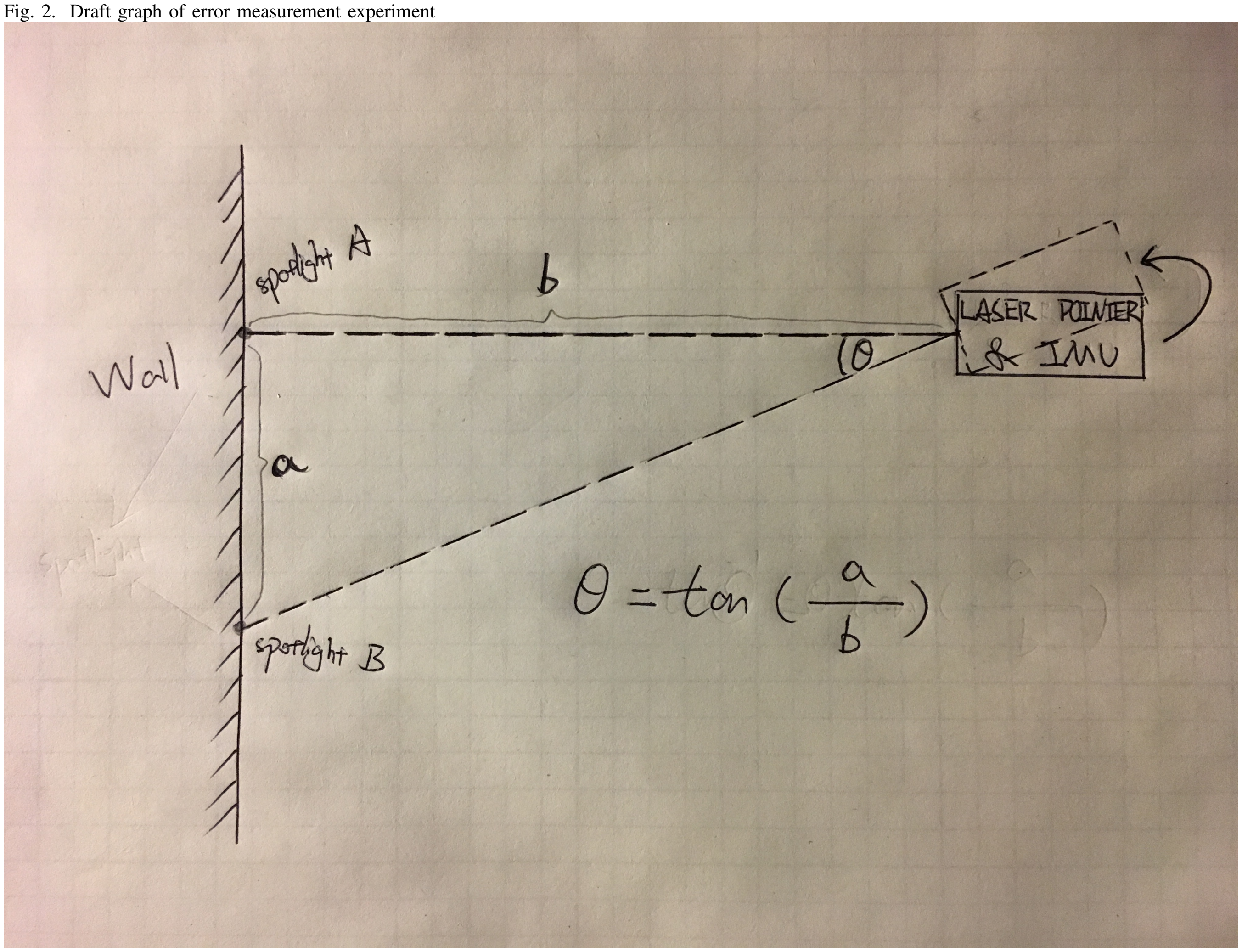
F. Experimental Design Description

This is an experimental design for measuring the actual error of the IMU sensor. A method to precisely measure the error is to compare a known inclined angle with the calculated angle from two sensor output results (before and after incline).

The experiment will be using a laser pointer and a mechanical angle adjuster as figure 1 for assistant tool. A mechanical angle adjuster allows to be manually set for getting a certain degree of angle for simulating yaw, pitch and roll, that will be able to provide a stable motion for the IMU as well as error measurement. A laser pointer is also necessary, that a laser is for constructing a triangle by using the initial distance between the laser pointer and the wall (a), alternative distance between the laser pointer and wall, as well as the distance between two spotlight (b), refer figure 2:

Theta represents the angle between adjusting the angle, and by measuring the distance value of a and b, theta can be calculated by applying trigonometric function. Assume theta is the actual precise angle (in radian), the error of the IMU sensor can be measured by comparing the actual precise angle with the output data from the IMU.

Fig. 2. Draft graph of error measurement experiment



III. Krisna Irawan

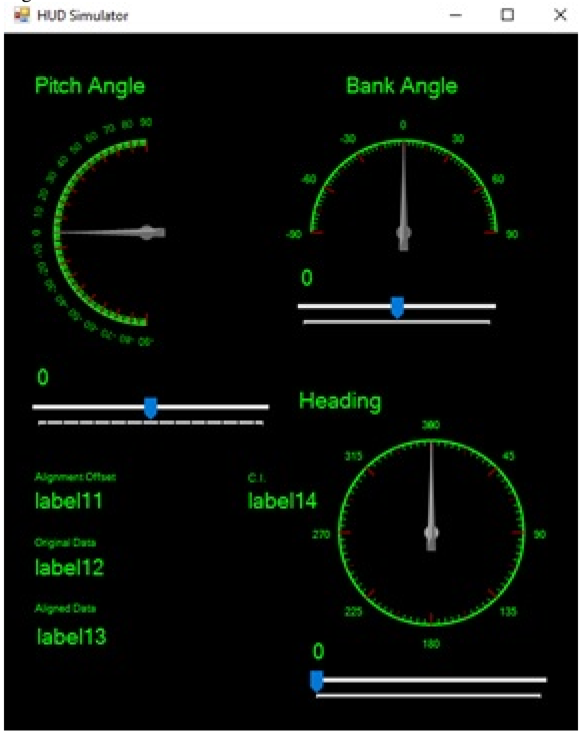
A. Purpose

Our project is a proof of concept that intended to identify the feasibility of a new innovation technology for Rockwell Collins Head-Up Display (HUD) system. Rockwell Collins strive to improve their HUD installment process and accuracy. Currently, the instalment process of the HUD system is costly, time consuming and require specialized equipment and epoxy. This mechanical way of installing the HUD also created an in-flight accuracy problem caused by airframe droop. In this project we are evaluating the feasibility of having an additional MEMS IRU to reduce the instalment cost of the HUD and also eliminate the offset error caused by airframe droop. The end goal of this project is to create a demonstration system that utilize an additional MEMS IRU mounted onto the HUD to infer the alignment data from the aircrafts precisely mounted and aligned IRU. Our software system will produce an aligned data that compensate the alignment error correctly, and the alignment error should be within a range of one milliradian. We hope to find the alignment offset dynamically with this new methodology. By finding the alignment offset dynamically, Rockwell Collins can reduce the installment cost of their HUD and improve the accuracy of their HUD system during flight.

B. Current Stage

I am in charge of the graphical user interface and the statistical analysis portion of the project. Currently, I am at the stage of connecting the graphical user interface that I created in Visual Studio to the offset algorithm part of the software created in Arduino by my teammates. At this stage, I make sure that the graphical user interface works smoothly by running a lot of user interface testing and configuration. I use a slider value in Visual Studio to make sure that the gauge works perfectly. I also put place holders for the alignment offset, original data, and offset data. I believe these place holders will

Fig. 3. The HUD Simulator



reduce the time required for me when connecting the user interface and the offset algorithm in the future. The user interface runs smoothly with the visual studio environment. I am currently waiting for my teammates to make sure that their software portion in Arduino runs smoothly. I also have done research in connecting the Visual Studio graphical user interface with the Arduino environment. I am currently had a brief understanding with that process and probably will start to experiment with Arduino code next week. This week, I conducted a couple user study with my friends. Although the graphical user interface doesnt look like a real HUD from Rockwell Collins, it still serves its purpose. See figure 3 for a better picture. With a little explanation, my friends can grasp the concept and the meaning behind the HUD symbology in the user interface. I believe that at this stage the user interface is ready to go and I can move to the next implementation of the software.

C. Remaining Work

The next step for the user interface is to connect it to the Arduino part of the software. The hardware logistic problem at the beginning of the term has set our team behind the schedule for two weeks. This hinder our team ability to start working on the Arduino portion of the software. Currently, our team is still working on the hardware-software configuration part in Arduino. However, I believe with the current state the user interface, it will take a little time to connect it to the Arduino part of the software. I also would like to do more user study for the graphical user interface. There are significant changes in the user interface and I want to make sure that it still meets the standards of our client. I will start by confirming the user interface to our client this following week.

The next step on the implementation pipeline is the statistical analysis. I am in charged for the statistical analysis for both the pre-aligned data (the original data) and also the aligned data. I am still undecided on which IDE to write my code in. Currently, I am leaning towards to write my code in visual studio since I have been working with the Visual Studio IDE for the user interface. Last term I have done some research for statistical analysis method during the Tech Review. This term, I will start to do more research on the implementation side of the statistical analysis algorithm. The timeline for the statistical analysis portion of the software is tight. Hopefully I can finish the statistical analysis portion of the software on the next two weeks.

D. Problems and Solutions

There was a logistical problem at the beginning of the term. Our team were not able to get the hardware required to get started with this project. This problem was frustrating for our team since our project rely heavily with the hardware. Without the hardware it is not possible to get started with the bone and the core of this project. Our hardware is the most important part of the demonstration system. The hardware is where our team get the alignment data as the original data and data to be manipulated with the software.

To solve this problem, our team pro-actively contact the instructor and the TA to get the hardware. Our team sent a bunch of email reminding our instructor to get the hardware. Our team also tried to get in touch with the instructor in person. We came to the instructor office hour to talk about the hardware problem. We also talked about the hardware problem with the TA during meetings to see what our TA can do to help us. Our team also set up a meeting with the clients to let them know about the logistical problem that we were having. The meeting with the clients is important because it eliminates any misunderstanding that might happens between our team and our clients. This pro-active communication has allowed our team to get the necessary hardware faster and maintain our good relationship with the clients.

The logistical problem that happens at the beginning of the term has set us behind the schedule for 2 weeks. Our team got the necessary hardware at the beginning of week 3 and had to start with the implementation as soon as possible. To save time, our team decided to work in parallel. I was in charge of the graphical user interface portion of the software and my teammates were in charge in the hardware-software configuration. Because our team was working in parallel, everybody can start with their portion of the software. Hence, it saves our team a lot of implementation time.

At the beginning of the term, I thought that our clients will provide us with the HUD symbology that I can be imported to the graphical user interface. However, this wasnt the case, our clients have a different method of generating the user interface on the HUD. Our clients used a software that will dynamically generate the HUD display. The Parameters come into the HUD software hosted on a computer via ethernet or Arinc- 429 inputs. Then software draws a distorted picture for the windshield and sent up to the HUD. This is significantly different than what I have in mind. There are no generic HUD symbology that I can directly import to the user interface. Thus, I have to create the HUD display from scratch.

Creating a HUD display is not an easy task. Rockwell Collins has put a lot of time and energy with their HUD display. Their current HUD display required a sophisticated software to generate it. Hence, creating a full HUD symbology will be out of scope of our project. I tried to be creative and use a gauge to act as the HUD symbology of the user interface. Although the gauge makes the user interface looks more like a car dashboard than an aircraft HUD, the user interface still serve it purpose in describing the data that we will get from our auto alignment system.

E. Interesting Code

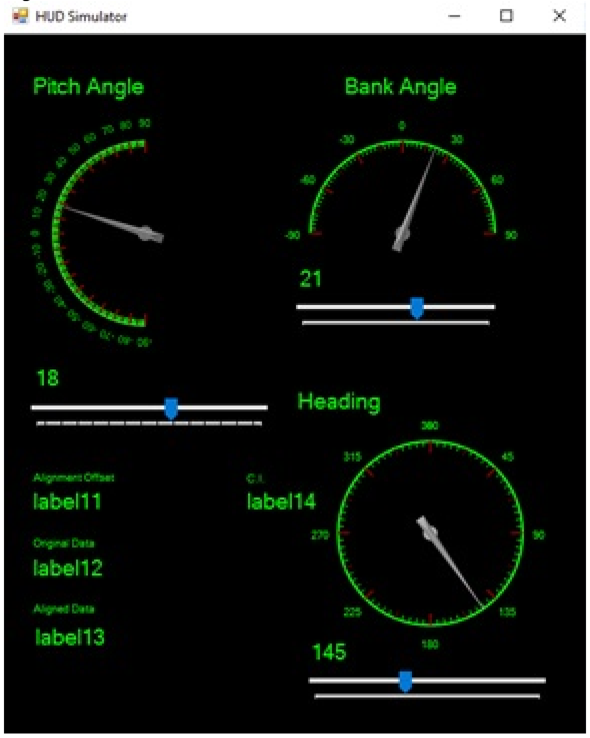
I use Visual Studio Toolkit to create the graphical user interface of the software. I wrote the code in C to make sure that it is compatible with Arduino. Because our team did not have the Arduino code ready, I decided to use the track bar as a method of debugging the graphical user interface. The track bar can be scrolled side by side to change the value of the gauge in the graphical user interface. To be able to use the gauge icon on the user interface, I imported a dll file called aGauge to my program.

```
public Form1()
{
    InitializeComponent();
    label9.Text = "0";
    aGauge3.Value = 0;
    label10.Text = "0";
    aGauge2.Value = 0;
    label8.Text = "0";
    aGauge1.Value = 0;
}

private void Form1_Load(object sender, EventArgs e)
{
    trackBar1.Minimum = -90;
    trackBar1.Maximum = 90;
    trackBar1.TickStyle = TickStyle.BottomRight;
    trackBar1.TickFrequency = 1;
    trackBar3.Minimum = 0;
    trackBar3.Maximum = 360;
    trackBar3.TickStyle = TickStyle.BottomRight;
    trackBar3.TickFrequency = 1;
    trackBar2.Minimum = -90;
    trackBar2.Maximum = 90;
    trackBar2.TickStyle = TickStyle.BottomRight;
    trackBar2.TickFrequency = 1;
}

private void trackBar1_Scroll(object sender, EventArgs e)
```

Fig. 4. The HUD Simulator in Action



```
{
    label9.Text = trackBar1.Value.ToString();
    aGauge3.Value = trackBar1.Value;
    aGauge3.Refresh();
}

private void trackBar2_Scroll(object sender, EventArgs e)
{
    label8.Text = trackBar2.Value.ToString();
    aGauge1.Value = trackBar2.Value;
    aGauge1.Refresh();
}

private void trackBar3_Scroll(object sender, EventArgs e)
{
    label10.Text = trackBar3.Value.ToString();
    aGauge2.Value = trackBar3.Value;
    aGauge2.Refresh();
}
```

F. First User Study

I conduct the user study with my friends. Although the graphical user interface doesnt look like a real HUD from Rockwell Collins, it still serves its purpose. With a little explanation, my friends can grasp the concept and the meaning behind the HUD symbology in the user interface. I used gauge as the HUD symbology of the user interface. At first, it was slightly confusing since it is not a typical flight simulator HUD, as figure 4 described. My friends told me that it looks more like a car dashboard. I will conduct the user study with our clients next week.

IV. Drew

A. Purpose

We have made progress towards our demonstration system of aligning data to a HUD in realtime. The purpose of this section will be to describe our current stage of progress, the remaining work that needs to be completed as well as problems we have come across and the solutions that were implemented.

B. *Current Stage*

Our primary focus at this point in our project has been on the hardware required to build the demonstration system, communication between devices and data manipulation. Due to a logistical error we were late to start digging into the meat of our project, however we have made significant progress in these last few weeks. The description of the current stage of our project will follow the timeline in which progress was made. First, a description of acquiring and modifying hardware at the physical level. Next, the configuration required for communication between devices. Lastly, data manipulation in terms of improvements, conversions and use.

We have successfully acquired three Metro Mini microcontrollers to meet the one Metro Mini requirement. The additional microcontrollers allow us to speed up development as each group member may work independently to solve and test independent tasks. Additionally, these extra microcontrollers provide tolerance in case of single hardware failure or project redesign. Second, we have successfully acquired three MPU-9250 IMUs to meet the two IMU requirement. While these additional IMUs allow for independent development by group members to some extent, our demonstration system requires at least two per system. Again, we have some leeway in case of hardware failure as well as allowing us to combine additional MEMs to reduce error.

Although the hardware acquired are capable of meeting our projects requirements, some modifications were needed whereas some were made for ease of use. An ease of use modification was carried out by soldering pins to the bottom of both IMUs and microcontroller to reduce development time by allowing rapid prototyping via breadboard. A single similar modification was needed on each IMU and the microcontroller. To allow the IMU's I2C address to be changed dynamically from 0x68 to 0x69 via its AD0 pin, a jumper on the bottom of the MEM had to be resoldered. Initially, the IMU's I2C address was preset to 0x68 as the jumper connected the AD0 pin to ground. After the modification was made, the IMU could have its address set to 0x69 by supplying 3.3v to the AD0 pin. To convert the microcontroller's digital logic pins to 3.3v down from 5v we had to cut and solder a jumper on the bottom of the microcontroller.

After wiring the devices together some configuration was required before communication via I2C could work appropriately. In order to access magnetometer data from individual IMUs we had to configure the magnetometer to act as a slave to the MEM while disabling the default pass through mode as defined by the MEM. Without, this modification the separate magnetometers would be indistinguishable on the I2C bus as they use the same address. When changing the magnetometer to act as a slave to the IMU, the accelerometer and gyroscope were also set this way. Although all sensors were configured in much the same way, the magnetometer may only be set to return one byte at a time and thus required a slightly different configuration.

Now that the communication between devices has been supported, we were able to retrieve and manipulate IMU data. To improve the accuracy of data being generated by the IMUs, each IMU needed to be initialized and calibrated at bootup. First, we took a sampling of at-rest readings from both gyroscope and accelerometer after device initialization. Next, we used these readings to calculate the average offset for each sensor. Lastly, we stored these offsets in accelerometer and gyro bias registers.

With the initial calibration taken care of we were able to output accelerometer, gyroscope and magnetometer data for each MEM via the single I2C bus. While we are only using two MEMs now, by dynamically changing addresses via each MEMs AD0 pin, we are able to access data from as many MEMs as our microcontroller has digital logic pins running at 3.3v. To improve accuracy of the data we are able to access, a filtering technique was used. In our case we chose the Madgwick filter fusion algorithm to create a quaternion-based estimate of device orientation by fusing sensor data.

Lastly, now that the IMU's data has been converted to quaternions, we have the ability to use that data for our implementation. We are able to find the difference between IMUs with a straightforward formula. To find the difference between IMUs, we take the quaternion data from one IMU and multiplying it by the inverse of the other IMU's quaternion data. With quaternion manipulation achieved, we are now able to continue moving towards our solution of aligning data.

C. *Remaining work*

- **Improve offset algorithm**
  While we are able to determine the difference between IMUs, we must modify the algorithm to account for both the initial alignment error and the alignment error that would occur during the flight environment.

- **Create statistical analysis methods**
  By creating statistical analysis methods we will be able to improve the offset algorithm.

- **Test individual MEM error**
  To ensure we are able to meet the maximum alignment error, we will need to test the IMUs. We may find that additional IMUs will be required to reduce alignment error further.

- **Calculate and include sufficient number of MEMS to meet accuracy requirements**
  After testing IMU error and comparing it to the described hardware specifications we will use this information and apply formula to determine the minimum number of IMUs that would be required without changing the specific IMU model being used.

- **Create demonstration UI showing appropriate symbology and alignment corrections**
  Our demonstration system tries to mimic the real world representation by including a US with the appropriate symbology.

- **Create physical demonstration system**
  Our current progress has left us with an implementation in which both IMUs are on a single breadboard. Although this may be sufficient for initial alignment, dynamic alignment would require the IMUs to move independently from one another.

D. Problems and Solutions

- **Problem 1:** Using I2C with both IMUs would not work when the IMUs shared the same address.

  Solution: Apply 3.3v to the IMU's AD0 pin to set its address to 0x69 or ground for 0x68.

- **Problem 2:** To dynamically change the address of each IMU we needed the microcontroller to output 3.3v from its digital logic pins but our microcontroller was set to 5v.

  Solution: To change the digital logic pins output to 3.3v down from 5v we had to cut a jumper and solder a new connection on the underside of the microcontroller.

- **Problem 3:** Setting 3.3v to the AD0 pin of an IMU was not changing its address to 0x69.

  Solution: By soldering a jumper on the IMU we were able to enable the expected AD0 functionality once it's short to ground was removed.

- **Problem 4:** Setting 3.3v to the AD0 pin of an IMU was not changing its address to 0x69.

  Solution: By soldering a jumper on the IMU we were able to enable the expected AD0 functionality once it's short to ground was removed.

E. Experimental Design Description

We may find that in order to meet the maximum alignment error tolerance without changing the model of IMU being used, we would need to combine additional IMUs. It is known that by averaging output of multiple IMUs, that precision can be improved by the square root of n, where n is the number of independent measurements. Optimal precision may be reached with IMUs have equal and opposite biases. Although we have little control over the individual IMU bias in one direction or another, we will may find it useful to increase the number of IMUs in our demonstration system.