Computer Science Senior Software Engineering Project

HEAD-UP DISPLAY ALIGNMENT SYSTEM

FINAL REPORT
SPRING 2017

Group 65:
KRISNA IRAWAN
JIONGCHENG LUO
DREW HAMM

Sponsor:
ROCKWELL COLLINS, INC.

JUNE 12, 2017

Abstract

Currently, a Head-up Display (HUD) obtains flight data from an Inertial Measurement Unit (IMU) on the airplane. However, the current HUD installation process requires specialized equipment and epoxy which is time consuming, costly, and interrupts production line progress for the manufacturer. In order to improve the current HUD alignment systems, Rockwell Collins, seeks a new alignment methodology that utilizes HUD mounted IMUs, aims to reduce the installation cost and time required to precisely align flight information to the HUD. This report contains all detailed documentations and records over the design and implementation of the project.

Contents

I. Introduction

Original Requirements Document

Computer Science Senior Software Engineering Project (CS462)

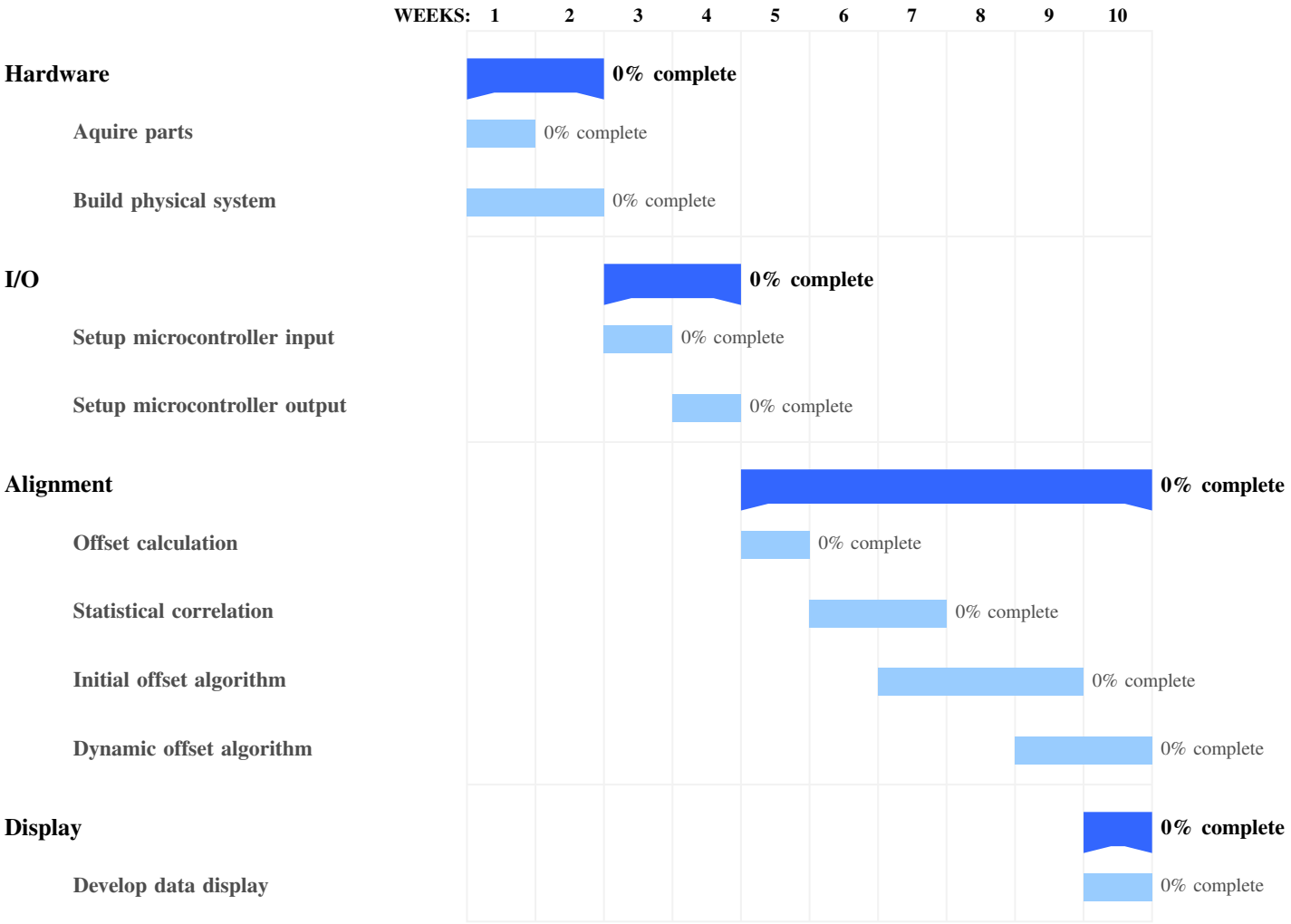
Winter 2017

Abstract

We are working with Rockwell Collins to explore potential technological innovations relating to their Head-Up Display (HUD) systems that present critical flight information to pilots. Our primary objective is to improve Rockwell Collins current HUD systems by reducing the cost and time required to precisely align flight information to the HUD. To meet our objective we will look into using a new alignment methodology in conjunction with the current HUD system as a proof of concept. The product being developed is a demonstration system that looks to include a MEMS IRU mounted onto the HUD and a new alignment algorithm that utilizes this additional sensor to determine accurate HUD alignment. This document will introduce the specific details of the demonstration system and describe the requirements for the development of the product.

Contents

I. Timeline for Project Plan (Gantt Chart)



II. Introduction

A. Purpose

The purpose of this document is to describe a demonstration system that explores using an additional HUD mounted MEMS IRU to meet previous installation standards while reducing costs. This system will further be described by its use of sensor data to dynamically find alignment offset error during flight as a consequence to airframe droop. This document describes the hardware and software requirements along with the functional and non-function requirements of the project. This document is intended to be used by the Head-Up display system engineer team of Rockwell Collins as a proof of concept.

B. Scope

This project aims to develop a demonstration system to work as a proof of concept for Rockwell Collins. This project is called the HUD Alignment System. The primary goal of this project will be to use sensor data to find the initial alignment offset after HUD installation. The alignment offset must be found within the same accuracy of the previous installation standards and will be used within the system as a hard coded value. The secondary goal will be to use this additional sensor to find the alignment offset during flight to be used within the system as a dynamic value. The dynamic alignment offset must also be found within the desired accuracy standards.

Currently, the HUD obtains data from an aircrafts mounted device called inertial reference unit (IRU), this IRU outputs precise and aligned data to the HUD. However, the current alignment process requires specialized equipment and epoxy which is time consuming, costly, and interrupts production line progress for the original equipment manufacturer. In addition, the resulting HUD alignment, while precise, does not compensate for airframe droop during flight. Rockwell Collins looks forward to a new alignment methodology utilizing an inexpensive microelectromechanical systems (MEMS) IRU mounted onto the HUD to infer alignment data from the aircrafts precisely mounted and aligned IRU. This project works on a solution that utilizes the data from both the inexpensive MEMS IRU and the aircraft mounted IRU to develop an algorithm, which aims to output precise and aligned data with reduced installation cost.

The outcome (aligned-data) of this algorithm will compensate the alignment error correctly, and the alignment error should be within a range of one milliradian. The product will make the alignment process more dynamic and less time consuming. The dynamic alignment process also makes this new system compensate for the airframe droop that happens during the flight environment. As well as from the perspective of the industries, this product aims to improve the installation process by reducing cost and time for all parties involved.

C. Definitions, Acronyms, and Abbreviations

- **HUD**
  A Head-Up Display (HUD) is a transparent display placed in front of a pilots head position in the cockpit of the aircraft. A HUD presents critical flight information to the pilots during the flight environment by using graphical, numerical and symbolical data [1].

- **Airframe Droop**
  As the center of the aircraft is pushed up due to lift, the nose of the aircraft is pulled down by its weight. The result of these conflicting forces causes a bend in the aircraft 's frame.

- **Conformal Attitude**
  A method of presenting flight information on the HUD. With conformal attitude presentation, the displayed information on the HUD is presented with respect to the real world (outside scene in front of the aircraft), and the presentation is dependent on the head position of the pilot [2].

- **Real-time**
  In this system, the algorithm should recognize and precisely align the alignment error for IMU output within minimal time (e.g., 500 milliseconds).

- **IRU/IMU**
  An Inertial Reference Unit (IRU), sometimes its called an Inertial Measurement Unit (IMU) is a device consists of inertial sensors typically including MEMS gyroscopes, accelerometers and compasses. These MEMS sensors measure and provide data of an aircraft 's velocity, acceleration and orientation [3].

- **MEMS**

An Inertial Reference Unit (IRU), sometimes its called an Inertial Measurement Unit (IMU) is a device consists of inertial sensors typically including MEMS gyroscopes, accelerometers and compasses. These MEMS sensors measure and provide data of an aircraft 's velocity, acceleration and orientation [4].

- **Accelerometer**
  An electromechanical device that measures the amount of static acceleration due to gravity, in other word, the rate of change in velocity of the mounted object [5].

- **MEMS Gyroscope**
  An electromechanical device that measures rotational motion/angular velocity, in other word, the speed of rotation of the mounted object [6].

- **I2C**
  The Inter-integrated Circuit (I2C) Protocol is a protocol intended to allow multiple slave digital integrated circuits (chips) to communicate with one or more master chips [7].

- **Alignment Algorithm**
  The uses of mathematical and logical equations to align the input data from the IRUs based on the data from previous precisely pre-aligned IRUs.

- **Symbology**
  The use and interpretation of symbols or special characters, in order for representing the corresponding flight data.

- **Milliradian**
  1 Milliradian (MRAD) = 0.001 radian, approximately 0.057296 degrees.

- **Quaternion**
  Four-element vector that is used to encode any rotation in a 3D coordinate system.

- **Altitude**
  A distance measurement (usually in vertical) between ground and the aircraft.

- **Latitude**
  AAn angular distance shown as a horizontal line, in degrees, minutes, and seconds of a point north or south of the Equator. Lines of latitude are often referred to as parallels [8].

- **Longitude**
  An angular distance shown as a vertical line, in degrees, minutes, and seconds, of a point east or west of the Prime (Greenwich) Meridian. Lines of longitude are often referred to as meridians [8].

D. Overview

The second section of this document contains an overall description of the product, including the product perspectives, product functions, user characteristics, constraints, assumptions and dependencies. The third section of this document will specify the projects requirements.

III. Overall Description

A. Product Perspective

Currently, the HUDs position is precisely aligned and determined during installation by a mechanical alignment method. Since the product will be developed as a proof of concept, this product will be a demonstrated system by using two separate MEMS IMUs respectively simulate for the add-on HUD IRU and Aircraft IRU. By including the HUD IRU, this alignment system will be able to determine the HUDs position as well as providing additional benefit to determine the HUDs shifting position during flight as a result of airframe droop.

B. Product Functions

- Both HUD and Aircraft IRUs will measure the acceleration and orientation of the mounted target object.
- The master boards (microcontrollers) will recognize both IRUs measurement data (raw) and output to the alignment algorithm.
- The alignment algorithm will compare the outputs from both IRUs (data from the aircraft IRU is assumed to be precise and aligned) and be able to quantify the alignment error as well as their correlation.
- The alignment algorithm will be able to calculate the target objects precise position based on the acceleration and orientation output from both IRUs.
- The algorithm will account for the target objects position and rotation to ensure that the conformal attitude, flight path information and other displayed data is accurately aligned.
- This algorithm will determine the alignment error to a desired accuracy level as 1.0 milliradian.

See figure 1 as the entire workflow for data transmission and connections between the hardware

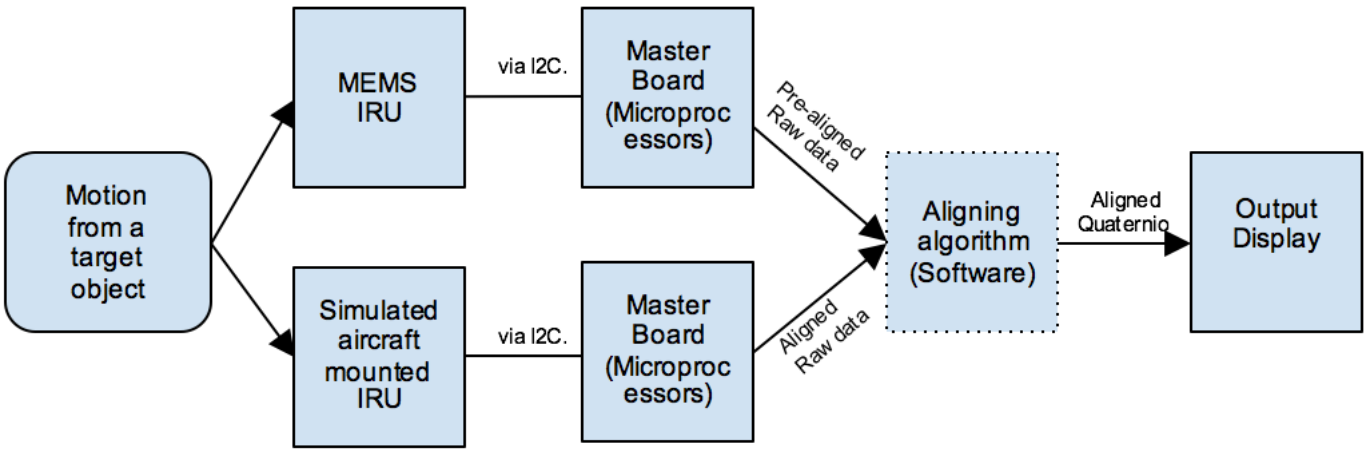


Figure 1: *HUD Alignment System*

C. User Characteristics

This alignment algorithm is intended to be applied to the HUD system, which will be used by Rockwell Collins to determine the availability of having an additional MEMS IRU in the for the new HUD alignment system. The intended users of this product are the technical expertise in this area, such as HUD system engineers in Rockwell Collins. The audience of this document will be familiar with the abbreviations and jargon used to describe the product. This familiarity reduces the requirement for more elaborate explanations of the project. As our expected user maintains technical expertise in the field, they will expect our project to meet a higher standard of competence.

D. Constraints

Hardware Limitations:

- This IMU model will be SparkFun MPU-9250 IMU MEMS sensors.
- The SparkFun MPU-9250 IMU is programmable to output 8,000 samples per second, down to 3.9 samples per second,
- The microcontroller model will used as the Metro Mini 328 chip to set up the sensors.
- The model ATmega328P as the core chip in Metro Mini 328.

- The algorithm will account for the target objects position and rotation to ensure that the conformal attitude, flight path information and other displayed data is accurately aligned.
- This algorithm will determine the alignment error to a desired accuracy level as 1.0 milliradian.

Signal Handshake Protocol:

- I2C protocol is used to communicate between IMU MEMS IMUs.

Higher order language Requirement:

- C/C++ programming language is used for the development of the software.

E. Assumption and Dependencies

- This product will depend on both the microcontroller and the sensors that are being used within the system in terms of hardware limitations.
- We assume that the microcontroller can interact with our sensors as well as to meet any other hardware concerns.
- We assume that the sensors are sufficient to simulate the aircraft IRU and HUD mounted MEMS IRU when building our demonstration system.

IV. Specific Requirements

A. External Interface Requirements

This section describes the detailed breakdown of the entire system. A description of the user, hardware, software and communication interfaces are presented.

1) User Interface: Our user interface will be the physical manipulation of our demonstration system in order to generate movement data.The software user interface for this project is a display that will show the properly aligned symbology of the airplane. A generic HUD symbology picture which could be moved up/down, left/right will be used to illustrate the alignment.

2) Hardware Interface:

- A microcontroller will have access to data from two separate 9-axis sensors that are used to represent the aircraft IRU and HUD mounted MEMS IRU.
- The microcontroller will additionally be connected to an output device.
- The hardwares are still not decided at this point. However, these are the product that we might use for the hardwares.
- Product Name: Metro Mini 328 - 5V 16MHz [9]
- Features:
  - ATmega328P onboad chip in QFN package
  - 16MHz clock rate, 28K FLASH available
- IMU Model: SparkFun IMU Breakout - MPU-9250 [10]
- Features:
  - Digital-output X-, Y-, and Z-axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of 250, 500, 1,000 and 2,000/sec and integrated 16-bit ADCs
  - Digital-output triple-axis accelerometer with a programmable full-scale range of 2g, 4g, 8g and 16g and integrated 16-bit ADCs
  - 3-axis silicon monolithic Hall-effect magnetic sensor with magnetic concentrator
  - Digitally programmable low-pass Gyroscope filter
  - Gyroscope operating current: 3.2mA
  - Accelerometer normal operating current: 450A
  - Magnetometer normal operating current: 280A at 8Hz repetition rate
  - VDD supply voltage range of 2.4 3.6V

3) Software Interface: At this point we are still not sure what additional library/software we will use in this project. We will use C-like language (C/C++) to program this project on Arduino IDE. We will also use the ATmega328P library since we will use the ATmega328P chip as our microcontroller board.

4) Communication Interface: I2C protocol will be used to communicate with MPU-9250 and Metro Mini 328.

B. Functional Requirements

1) Sensors:

a) Sensor recognition: This program will have the capability to recognize sensors. This program shall recognize and identify all sensors provided to the system. This functionality is the core to the input recognition capability.

b) Input recognition: This program will correctly interpret the data provided by the recognized sensors. This program will have the ability not only to recognize the outputs from the sensors, but also to interpret the outputs so it can be used as a proper input for the program. This functionality will be dependent on the success of the sensor recognition functionality.

2) Algorithm:

a) Quaternion Manipulation: This program will have the ability to do quaternion manipulation to the inputs. Inputs that are received from the sensors will be compared and manipulated to determine the alignment error. The desired accuracy level of the alignment error is within one milliradian. This functionality will execute in real-time, within 500 milliseconds.

3) Statistics:

a) Statistical Corelation: This program will have the capability to generate a confidence interval for each alignment error output being generated by the algorithm. Outputs in a form of alignment error will be used to determine the confidence interval of the data. This functionality will execute in real-time, within 500 milliseconds.

4) Simulation:

a) Simulated HUD: This program will have the capability to display the properly aligned symbology of the airplane by using either external LCD display or graphic software (etc., OpenGL). A generic HUD symbology picture could be moved up/down, left/right will be used to illustrate the alignment of the airplane.

C. Performance Requirements

- Number of terminals to be supported: Two terminals for both MEMS sensors.
- Amount of information to be handled: Programmable up to 3.9 to 8000 data per seconds. We will start with 10 data per seconds from both sensors.
- Type of information to be handled: Quaternion
- Valid Range of Accuracy: milliradians
- Timing: Currently not specified
- Destination of Output: Simulated HUD interface

D. Design Constraints

This product is limited to the specification of the hardware. This product will get its input from MEMS IRU and aircraft IRU. The specification of the MEMS IRU and aircraft IRU will alter the quaternion manipulations being done to the data. Different MEMS IRUs will require a different quaternion manipulations to achieve the expected standard of accuracy. This hardware dependability makes this product dependent to one specific hardware. Thus, limiting our design to one specific hardware.

Another design constaint is to precisely get the hardware error from the IMU sensors, since the alignment algorithm requires high accuracy data input, minor error may cause unexpected offset of the final results. Therefore, we will need a feasible experiment to test and measure the sensor error precisely.

E. Software System Attributes

1) Reliability: It is important to make sure the system in both software and hardware will be able to run properly and smoothly without abortion. As a proof a concept, reliability is important, a wrong data will lead to incorrect outcome in the determination of further development that based on this products.

2) Availability: The software piece of the entire system should be able to run without any crashes or other situation lead to system error (this is a critical issue during an real flight situation). Yet, the hardware pieces should be able to detect the failed components at the time when it happens.This product will have the capability to detect error and may restart itself when crashes.

3) Security: This product will be used internally by Rockwell Collins and since this product will be used for a proof of concept, security is not an issue.

4) Maintainability: This system does not work solely as a software neither a hardware product, which it has to be tested onto real flight situation. The current development is for a proof of concepts. Hence, maintainability will not be an issue for this project. This project will be used as a reference and basis of a future project in Rockwell Collins, but not directly used as the core/underlying code for any long-term product.

5) Portability: Portability of this system exists on the hardware level. Since hardware programming is required for this project to manipulate functionalities of the hardware components (e.g., sensors, microprocessors/microcontrollers), specific hardware programming language for this project is host-dependent. It depends on the specific controller boards we are using. In this project, we are using C for the programming language and Arduino IDE as the microcontroller programmer. C is one of the proven portable language for hardware programming for most general microcontrollers/microprocessors, as well as Arduino IDE is commonly used as the developemnt environment for programming AVR chip (Atmega328), therefore this system has high portability.

REFERENCES

[1] "Head-up guidance system," Rockwell Collins. [Online]. Available: https://www.rockwellcollins.com/Data/Products/Displays/Head-Up_Displays-HUD/Head-up_Guidance_System.aspx

[2] D. R. Jones, T. S. Abbott, and J. R. B. II, "Concepts for conformal and body-axis attitude information for spatial awareness presented in a helmet-mounted display," The National Aeronautics and Space Administration, 1993. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930014219.pdf

[3] "Inertial navigation system, a.k.a inertial reference unit," Earth Observing Laboratory, 2005. [Online]. Available: https://www.eol.ucar.edu/instruments/inertial-navigation-system-aka-inertial-reference-unit

[4] "What is mems technology?" MEMS and Nanotechnology Exchange. [Online]. Available: https://www.mems-exchange.org/MEMS/what-is.html

[5] "A beginners guide to accelerometers dimension engineering," Dimension Engineering LLC. [Online]. Available: http://www.dimensionengineering.com/info/accelerometers

[6] A1RONZO, "Gyroscope," SparkFun Electronics. [Online]. Available: https://learn.sparkfun.com/tutorials/gyroscope

[7] SFUPTOWNMAKER, "I2c," SparkFun Electronics. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c

[8] "Latitude and longitude facts," Worldatlas. [Online]. Available: http://www.worldatlas.com/aatlas/imageg.htm

[9] "Adafruit pro trinket - 5v 16mhz," Adafruit. [Online]. Available: https://www.adafruit.com/products/2000

[10] "Sparkfun imu breakout - mpu-9250," SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/13762

[11] "Ieee recommended practice for software requirements specification (ieee std 830-1998)," IEEE Computer Society, 1998. [Online]. Available: https://docs.google.com/a/oregonstate.edu/viewer?a=v&pid=sites&srcid=b3JlZ29uc3RhdGUuZWR1fGNhcHN0b25lLXdyaXRpbmctY29tbXVuaWNhdGlvbnN8Z3g6MzhkN2Q0MjAxN2Y5OGYyYg

[12] C. Rapids, "Head-up guidance system (hgs) for midsize and light business aircraft," Rockwell Collins. [Online]. Available: https://www.rockwellcollins.com/-/media/Files/Unsecure/Products/Product_Brochures/Displays/Head_up_displays/HGS-3500_White-Paper.ashx

III. Change since Original Requirement Document

	Requirement	What happened to it?	Comments
1	Our user interface will be the physical manipulation of our demonstration system in order to generate movement data.	Completed.	We built an object that can be moved as well as manipulated to change the pitch and roll.
2	The software user interface for this project is a display that will show the properly aligned symbology of the airplane.	Completed.	Our display shows the static offset values along with the current positions of both aircraft and HUD sensors via a dynamic GUI using aircraft models.
3	We will use a generic HUD symbology picture which could be moved up/down, left/right which will be used to illustrate the alignment.	Changed.	We created a generic HUD symbology but ended up using aircraft models instead.
4	A microcontroller will have access to data from two separate 9-axis sensors that are used to represent the aircraft IRU and HUD mounted MEMS IRU.	Completed.	The microcontroller is connected to both IMUs via I2C as well as using digital logic pins connected to the IMUs respective AD0 pins.
5	The microcontroller will additionally be connected to an output device which will be used to display the results.	Completed.	The microcontroller can be connected to a laptop via serial communication.
6	The microcontroller hardware has yet to be chosen.	Changed.	We choose to use the Metro Mini 328.
7	The MEMS hardware has yet to be chosen.	Changed.	We choose to use the MPU-9250.
8	We will use C-like language (C/C++) to program this project on Arduino IDE.	Completed.	Our solution was written in the selected language and developed through the Arduino IDE.
9	We will also use the ATmega328P library since we will use the ATmega328P chip as our microcontroller board	Completed.	Our solution uses the ATmega328P library.
10	Additional library/software that we may use has yet to be specified.	Changed.	We used VPython, pySerial along with an open source library for the MPU-9250.
11	I2C protocol will be used to communicate between microcontroller and IMUs.	Completed.	We used the I2C protocol to communicate between microcontroller and IMUs.
12	This program will have the capability to recognize sensors.	Completed.	Using I2C, we can recognize and communicate with available sensors.
13	This program shall recognize and identify all sensors provided to the system.	Completed.	We can recognize and identify all sensors after modifying the magnetometer to act as a slave to its respective IMU.
14	This program will correctly interpret the data provided by the recognized sensors.	Completed.	We are able to retrieve the x, y, and z data from the accelerometer, gyroscope, and magnetometer.

	Requirement	What happened to it?	Comments
15	This program will have the ability to do quaternion manipulation on the inputs.	Completed.	We the magdwick filter to convert our sensor data into quaternions. We can take the difference between two quaternions.
16	Inputs that are received from the sensors will be compared and manipulated to determine the alignment error.	Completed.	We can take the difference between two sets of values to find the alignment offset.
17	Alignment error is found within one milliradian.	Pending.	We have yet to test our algorithmic results against the physical offset.
18	An alignment error reading will be found within 500 milliseconds of starting its function.	Completed.	Not counting initialization, the readings are returned under 500 milliseconds.
19	This program will have the capability to generate a confidence interval for the alignment error output being generated by the algorithm.	Completed.	We calculate the confidence interval for 95.
20	Outputs in the form of alignment error will be used to determine the confidence interval of the data.	Completed.	We take a set of alignment offset samples to determine the confidence interval.
21	The confidence interval will be found in 500 milliseconds of starting its function.	Completed.	We take 80 samples before determining the confidence interval.
22	This program will have the capability to display the properly aligned symbology fo the airplane by using either external LCD display or graphic software (etc., OpenGL).	Completed.	We use VPython to display our results.
23	A generic HUD symbology picture could be moved up/down, left/right and will be used to illustrate the alignment of the airplane.	Changed.	We created a generic HUD symbology but ended up using aircraft models instead.
24	Two terminals will be supported; one for each MEMS sensor.	Completed.	We can access each IMU.
25	We will handle a programmable 3.9 to 8000 data per seconds of information.	Completed.	The rate at which samples are taken can be changed and our device supports our requirements.
26	We will start with 10 data per seconds from both sensors.	Completed.	We started with few readings and increased as our devices allowed.
27	We will handle quaternion information.	Completed.	We convert our sensor values into quaternions and manipulate or display the quaternion values.
28	The valid range of accuracy is milliradians	Completed.	We used milliradians for data output.
29	The destination of output will be the simulated HUD interface.	Changed.	We created a generic HUD symbology but ended up using aircraft models instead.

Original Design Document

Computer Science Senior Software Engineering Project (CS462)

Winter 2017

Abstract

A Head-up Display (HUD) Alignment system is developed as a proof of concept aims to explore a potential technological innovation for the HUD system that presents critical flight information to pilots. The primary objective of this project is to reduce the cost and time required to precisely align flight information to the HUD by introducing an additional sensor component to the system to make the alignment process more dynamic. This document is intended for use by Rockwell Collins and their HUD system development team. This document provides and explains an overall system framework, design viewpoints and specific design description for each viewpoint within the system.

Contents

I. Introduction

A. Purpose

The purpose of this software design document is to provide a description that sufficiently describes the system design. The system design must be specified completely to the point in which is needed for the development to proceed. The description will fully specify what is to be built, how it will be built and what expectations need to be met at completion.

B. Purpose

A HUD or a Head-Up Display provides critical information to the pilots during flight environment. Currently, the HUD obtains data from an aircrafts mounted device called an Inertial Reference Unit (IRU), which an IRU would output precise and aligned data to the HUD through an mechanical alignment system. However, the current alignment process requires specialized equipment and epoxy which is time consuming, costly, and interrupts production line progress for the original equipment manufacturer. In addition, the resulting HUD alignment, while precise, does not compensate for airframe droop during flight. Rockwell Collins looks forward to a new alignment methodology utilizing an inexpensive microelectromechanical systems (MEMS) IRU mounted onto the HUD to infer alignment data from the aircrafts precisely mounted and aligned IRU.

This project is to develop a feasible demonstration system as a proof of concept for Rockwell Collins, that will prove there is an algorithm that is able to output precise and aligned data with reduced installation cost utilizing the data from both the inexpensive MEMS IRU and the aircraft mounted IRU. The outcome (aligned-data) of this algorithm will compensate the alignment error correctly, and the alignment error should be within a range of one milliradian. The product will make the alignment process more dynamic and less time consuming. The dynamic alignment process also makes this new system compensate for the airframe droop that happens during the flight environment. As well as from the perspective of the industries, this product aims to improve the installation process by reducing cost and time for all parties involved.

C. Overview

The following second and third part of this document are glossary and references that we have for this document. The fourth part of this document cover the design and implementation details of this project. The design and implementation details of this document are split into six different viewpoints. These viewpoints contain context view, composition view, dependency view, interface view, interaction view, and algorithm view.

II. Definitions

- **HUD**
  A Head-Up Display (HUD) is a transparent display placed in front of a pilots head position in the cockpit of the aircraft. A HUD presents critical flight information to the pilots during the flight environment by using graphical, numerical and symbolical data [1].

- **Airframe Droop**
  As the center of the aircraft is pushed up due to lift, the nose of the aircraft is pulled down by its weight. The result of these conflicting forces causes a bend in the aircraft 's frame.

- **Conformal Attitude**
  A method of presenting flight information on the HUD. With conformal attitude presentation, the displayed information on the HUD is presented with respect to the real world (outside scene in front of the aircraft), and the presentation is dependent on the head position of the pilot [2].

- **Real-time**
  In this system, the algorithm should recognize and precisely align the alignment error for IMU output within minimal time (e.g., 500 milliseconds).

- **IRU/IMU**
  An Inertial Reference Unit (IRU), sometimes its called an Inertial Measurement Unit (IMU) is a device consists of inertial sensors typically including MEMS gyroscopes, accelerometers and compasses. These MEMS sensors measure and provide data of an aircraft 's velocity, acceleration and orientation [3].

- **MEMS**

An Inertial Reference Unit (IRU), sometimes its called an Inertial Measurement Unit (IMU) is a device consists of inertial sensors typically including MEMS gyroscopes, accelerometers and compasses. These MEMS sensors measure and provide data of an aircraft 's velocity, acceleration and orientation [4].

- **Accelerometer**
  An electromechanical device that measures the amount of static acceleration due to gravity, in other word, the rate of change in velocity of the mounted object [5].

- **MEMS Gyroscope**
  An electromechanical device that measures rotational motion/angular velocity, in other word, the speed of rotation of the mounted object [6].

- **I2C**
  The Inter-integrated Circuit (I2C) Protocol is a protocol intended to allow multiple slave digital integrated circuits (chips) to communicate with one or more master chips [7].

- **SPI**
  Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers [8].

- **UART**
  A universal asynchronous receiver/transmitter (UART) is a block of circuitry responsible for implementing serial communication. On one end of the UART is a bus of eight-or-so data lines (plus some control pins), on the other is the two serial wires - RX and TX [9].

- **Multiplexer**
  A MUX (Multiplexer) is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal [10].

- **DMP**
  A Digital Motion Processor (DMP is a programmable chip within the SparkFun MPU-9250 IMU MEMS sensor. Allows for filtering output as well as converting to quaternion output without the need of an additional microcontroller [11].

- **Alignment Algorithm**
  The uses of mathematical and logical equations to align the input data from the IRUs based on the data from previous precisely pre-aligned IRUs.

- **Symbology**
  The use and interpretation of symbols or special characters, in order for representing the corresponding flight data.

- **Milliradian**
  1 Milliradian (MRAD) = 0.001 radian, approximately 0.057296 degrees.

- **Quaternion**
  Four-element vector that is used to encode any rotation in a 3D coordinate system.

- **Altitude**
  A distance measurement (usually in vertical) between ground and the aircraft.

- **Latitude**
  AAn angular distance shown as a horizontal line, in degrees, minutes, and seconds of a point north or south of the Equator. Lines of latitude are often referred to as parallels [12].

- **Longitude**
  An angular distance shown as a vertical line, in degrees, minutes, and seconds, of a point east or west of the

Prime (Greenwich) Meridian. Lines of longitude are often referred to as meridians [12].

III. Context View

A. Program Design

This section is the black box view for the system. This section covers the high-level, overall design description for this project including design concern, design elements, and HUD alignment system workflow are presented in this section.

1) Design Concern: The primary goal of this project will be to use sensor data to find the initial alignment offset after HUD installation. The alignment offset must be found within the same accuracy of the previous installation standards and will be used within the system as a hard coded value. The secondary goal will be to use this additional sensor to find the alignment offset during flight to be used within the system as a dynamic value. The dynamic alignment offset must also be found within the desired accuracy standards. The outcome of this project is to create an algorithm that will produced the correct alignment data for the HUD alignment system. The aligned-data will compensate the alignment error correctly, and the alignment error should be within a range of one milliradian.

*2) **Design Elements**:*

a) Stakeholders: Rockwell Collins HUD system engineers. This program is a proof of concepts that intended to be applied to the future HUD system. This program will be used by Rockwell Collins to determine the availability of having an additional MEMS IRU in the new HUD alignment system.

b) Design relationships: Users will be able to interact with the program via simulation or physical demonstration system. The users will be able to move and interact with the physical demonstration system to simulate the airframe droop and misalignment in the system.

c) Design constraints: This project is limited to its hardware, signal handshake protocol, and higher order language requirements:

- This IMU model will be *MPU-9250* MEMS sensors.
- The *MPU-9250* is programmable and it outputs 8,000 samples per second, down to 3.9 samples per second.
- The microcontroller model will used as the *Metro Mini 328* to set up the sensors
- The model *ATmega328P* as the core chip in *Metro Mini 328*.
- C/C++ programming language is used for the development of the software based on the selected microcontroller.

IV. Composition View

A. Hardware Configuration Design

This section will cover the specific design plan for assembling all hardware compositions of the system. This section is intended to summarize and explain all the necessary hardware components involved in the system design and their specification.

1) Design Concerns: One of the concerns in developing this demonstration system is to use inexpensive MEMS IMU instead of using the one from the original equipment manufacturer, which is costly. Hence, we chose to use the *Metro Mini 328* as the microcontroller that has a fair price in the market currently. *Metro Mini 328* provides an option that allows to use a 3.3V mode power supply. And since 3.3V is the standard operating power supply our selected IMU sensors model, *MPU-9250*, *Metro Mini 328* make it easy to connect with *MPU-9250* without any external circuit or additional resistors.

2) Design Elements:

- Simulated aircraft object: a remote controlled vehicle/drone
- HUD IMU: Three *MPU-9250*s
- Aircraft Three *MPU-9250*s
- Microcontroller: one *Metro Mini 328*
- Output Display: Graphical output by LCD display/graphical software (for demonstration system only)

a) Design Relationships: Initially, all the hardware components will be mounted onto a breadboard as figure 1. While the breadboard is in a motion that could be a moving action by a remote vehicle, a flying action of a drone or simply by shaking it in hand, the IMU sensors (both HUD and aircraft IMUs) are simultaneously updating these motion data such as acceleration, angular velocity and orientation of the moving object and sending data to the connected microcontroller that is the *Metro Mini 328* though the *I2C* protocols. Here, the input from both IMUs to the microcontroller are raw data, and these two groups of data will be the input for the alignment algorithm. Refer Figure 4 in section **Component Diagram** for more detail about the relationship among all hardware components.

Fig. 1. IMU & Microcontroller Mounted onto a breadboard

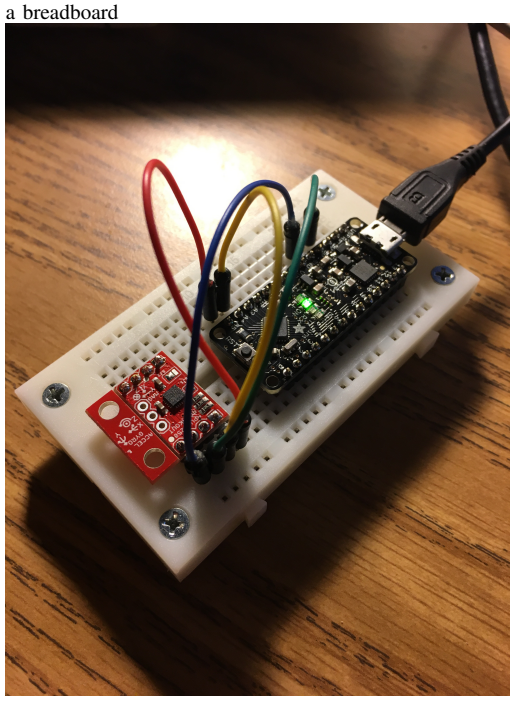
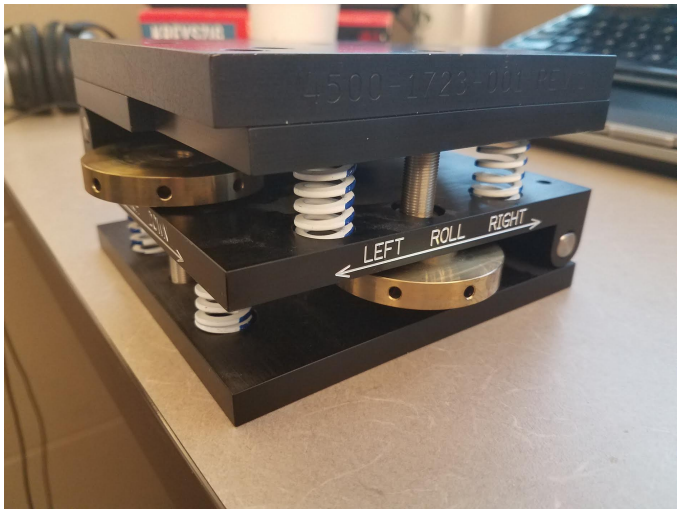


Fig. 2. Mechanical Angle Adjuster



b) Function Attribute: The entire system is constructed based on four major entities including a simulated aircraft object, a microcontroller, a HUD IMU and an aircraft IMU. In addition, an output display is necessary in this demonstration system for testing purpose but it is not part of the original system, that means this display is used for simulating the real HUD output on an aircraft. Respectively, these two IMUs in this demonstration system represent and simulate the functionalities for the real IMU component mounted on a head-up display of an aircraft and the aircraft body itself. Within this demonstration system, both IMUs may consist of one or more *MPU-9250* (e.g., accelerometers). The simulated aircraft object provides motion input to the IMU sensors this object can be any kinds of vehicle or drone that is able to provide physical motion.

c) Experiment Design for IMU Error Measurement: It is vital to know the actual error of all IMU sensors before using the data into the algorithm. 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 2 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

Fig. 3. Draft graph of error measurement experiment

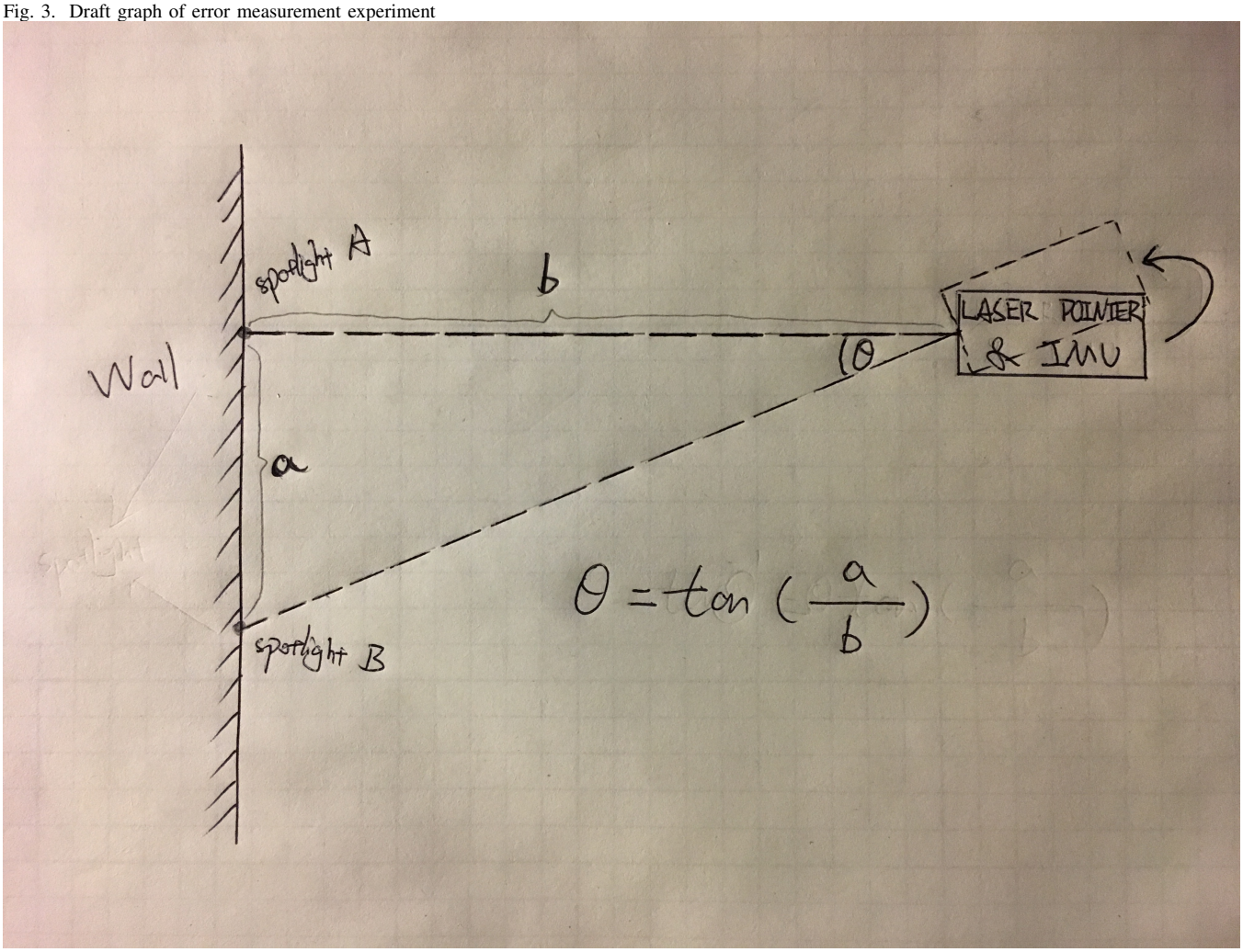
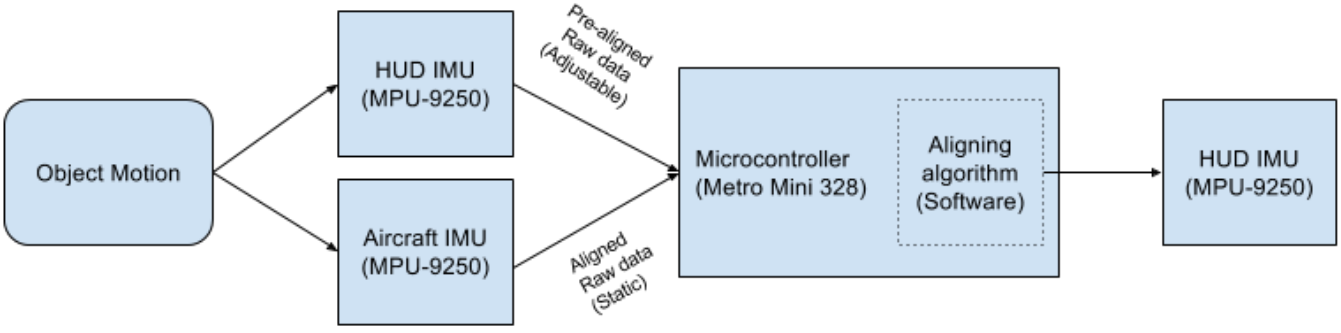


Fig. 4. Data Flow between Hardware Components



between the laser pointer and wall, as well as the distance between two spotlight (b), refer figure 3:

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.

V. Dependency View

A. Dependency Design

This section will cover the interconnection of hardware and software throughout the system. This section is intended to summarize and explain all the necessary components in terms of their dependencies.

1) Design Concern: All components must have clearly defined relationships. All components within the system should be included in the dependency design, refer Figure 5 UML diagram for more detail.

2) Design Elements:

- HUD IMU: *MPU-9250*.
- HUD IMU DMP: Provides on board filtering and 9-axis to quaternion conversion.
- Aircraft IMU: *MPU-9250*.
- Aircraft IMU DMP: Provides on board filtering and 9-axis to quaternion conversion.
- Microcontroller: *Metro Mini 328* .
- Offset Algorithm: Takes input from both IMUs through the microcontroller and returns the offset between each other.
- Statistical Analysis Algorithm for Initial Alignment Offset: Takes multiple offset inputs until a valid initial offset can be returned.
- Statistical Analysis Algorithm for Dynamic Alignment Offset: Takes multiple offset inputs until a valid dynamic offset can be returned.
- Initial Offset: The value found that represents the initial alignment offset.
- Dynamic Offset: The value found that represents the dynamic alignment offset.
- HUD: Graphical output by a computer software (for demonstration system only).

3) Dependency Attributes: See Figure 5.

4) Design Rationale: An overall view of the system is required to understand how each component fits into the system in relation to other components. By providing the dependency design, faults within the system can be determined by analyzing their specific scopes and connections. The dependency view will be used to determine the prioritization for component development to ensure that components are being developed by time of need to avoid blocking.

VI. Interface View

A. Software User Interface Design

This section will cover the specification of the software user interface portion for this project. This section is intended to guide the software development team to interact correctly with the software user interface.

1) Design Concerns: The goal of this project is to have a demonstration system that can be used by Rockwell Collins HUD system engineers to determine the availability of having an additional MEMS IRU in the new HUD alignment system. Graphical user interface will be crucial for our project presentation and demonstration system. The algorithm from our project will output a raw alignment data and it will be hard to understand the output without the help of graphical user interface. The graphical user interface will put context to the data and present the symbology of the alignment data output. A gauge icon is used to define the HUD symbology. The graphical user interface is created in Visual Studio IDE and ready to be connected to its Arduino program counterparts.

2) Design Elements: This section will cover the interface attributes that defines each functionality on the display and a paper prototype as Figure 6.

a) Interface Attributes:

- *Heading:* Display the direction of where the aircraft is pointing
- *Bank Angle:* Display the vertical elevation angle of the aircraft
- *Pitch Angle:* Display the angle between the aircraft and the horizon. Pitch angle shows the horizontal elevation angle of the aircraft
- *Credible Interval:* Display the degree of certainty of the alignment offset
- *Alignment Offset:* Display the calculated alignment offset detected by the algorithm
- *Original Data:* Display the original alignment data from the sensors
- *Aligned Data:* Display the calculated alignment data from the algorithm

Fig. 5. Dependency UML Diagram

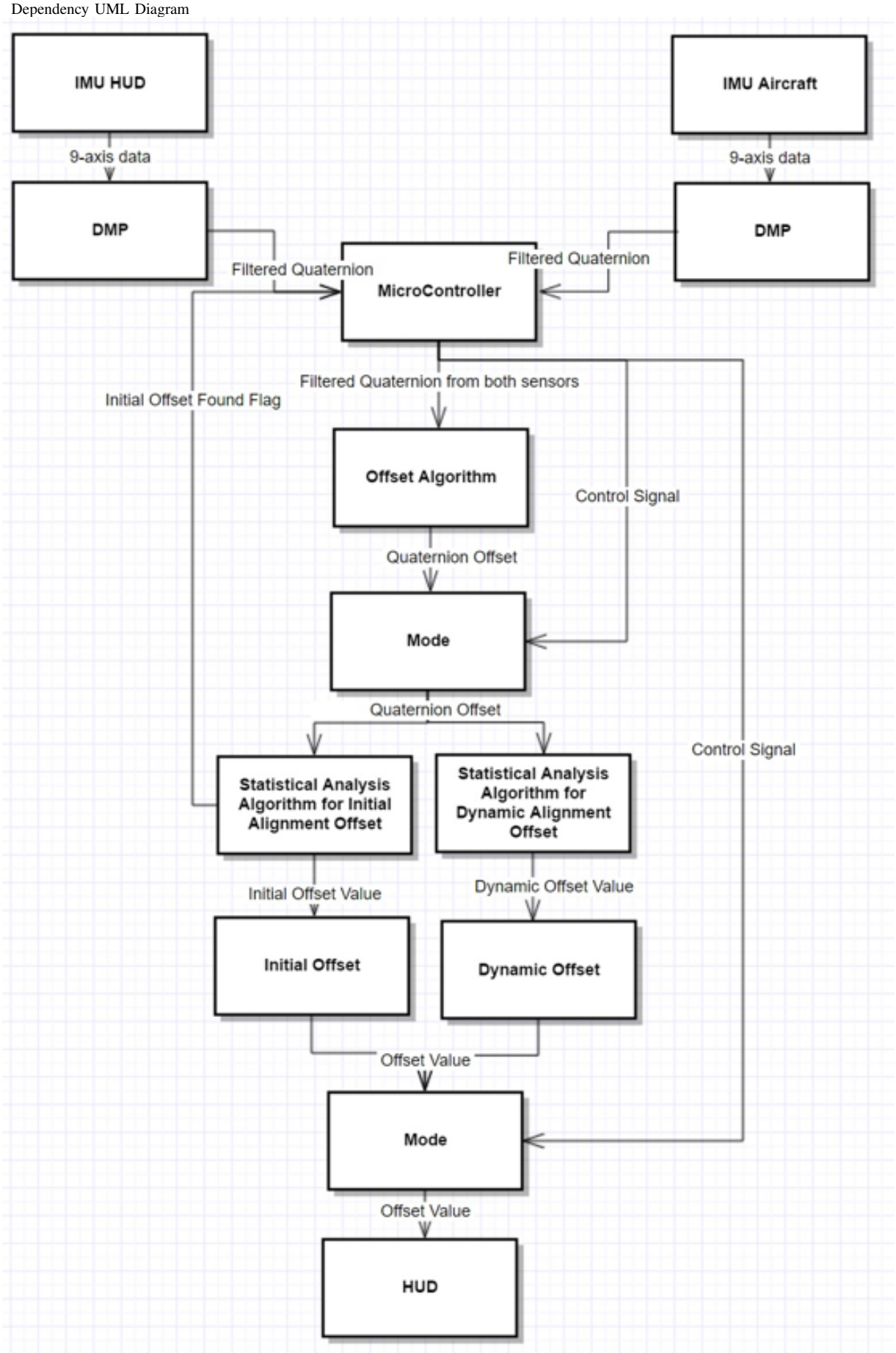
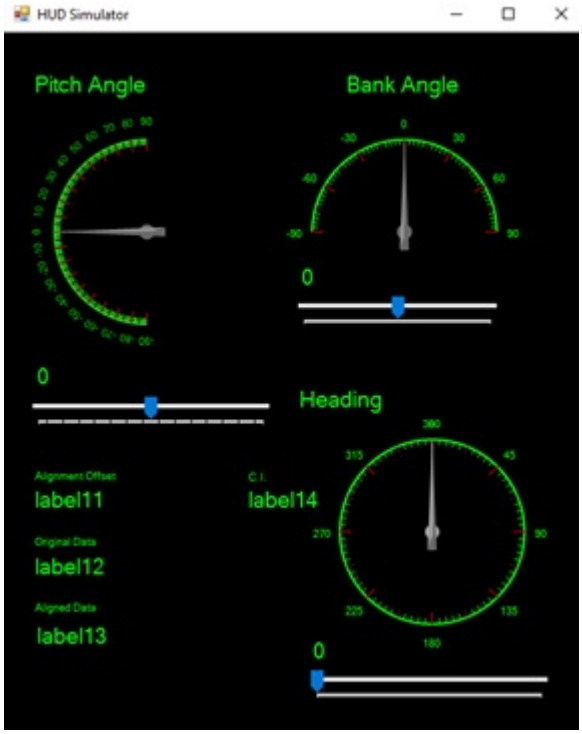


Fig. 6. The HUD Simulator



3) Design Rationale: An important piece of this project is the dynamic alignment algorithm and our clients encouraged us to spend most our time in developing and refining the dynamic alignment algorithm. Learning that Rockwell Collins use a sophisticated software to generate their HUD display, using a full HUD display will also be out of scope for this project. We want to create a graphical user interface that is simple and serve its purpose as a medium for the user to understand the data being generated by the algorithm. We create this graphical user interface with Visual Studio Toolkit. The built in visual studio graphical user interface toolkit will help us to arrange the HUD symbology representation to its designated place and also display the appropriate data to the screen. We use a gauge to represent the HUD symbology. This user interface might not look like a flight simulator HUD, but it still serves its purpose. We also use 0 360 scale as the heading, where 0/360 is north, 90 is east, 180 is south, and 270 is west.

B. Physical User Interface Design

This section will cover the specification of the physical user interface portion for this project. This section is intended to guide the software development team to interact correctly with the physical user interface in the demonstration system.

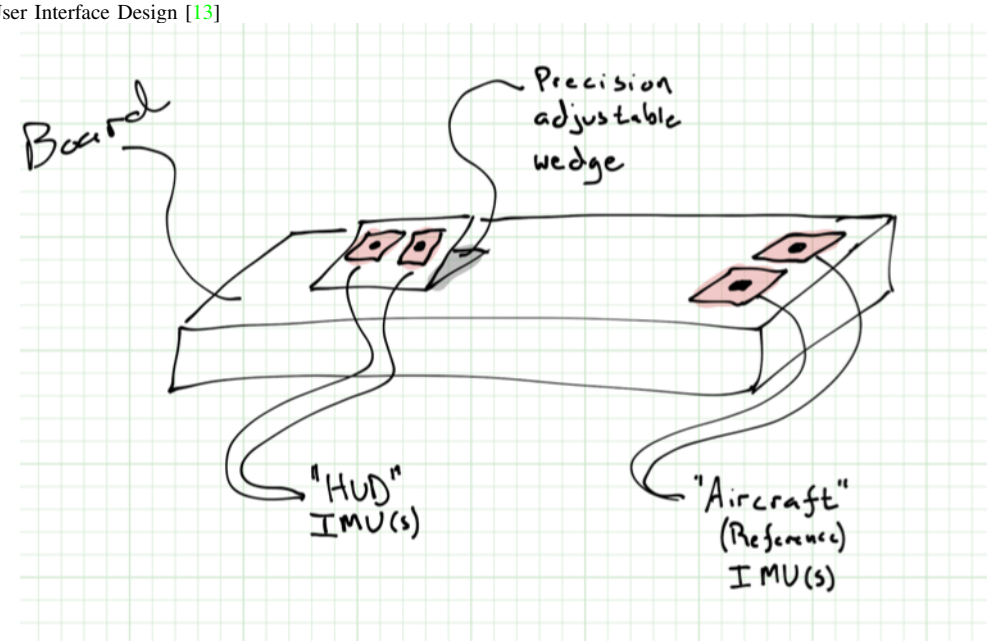
1) Design Concern: The goal of this project is to have a demonstration system that can be used by Rockwell Collins HUD system engineers to determine the availability of having an additional MEMS IRU in the new HUD alignment system. The user of this product will be able to directly interact with the physical user interface of this product. The physical user interface is the most interactive piece of this product. Refer to Figure 7 The user will have the ability to adjust the precision adjustable wedge to simulate the HUD offset during flight. The user will also have the ability to move the board to simulate the airplanes movement. This physical user interface settings is arranged to simulate the flight environment. Rockwell Collins will help us in creating the board with the precision adjustable wedge in it.

2) Design Elements:

a) Interface Attributes:

- Board: A platform that can be moved to simulate the airplane movement
- Precision adjustable wedge: An adjustable platform that can be moved to simulate the HUD alignment offset after initialization
- HUD IMU sensors: The sensors that act as the HUD IMU
- Aircraft IMU sensors: The sensors that act as the precisely aligned aircraft IMU.

Fig. 7. Physical User Interface Design [13]



3) Design Rationale: The physical user interface is a crucial piece of this project. This project algorithm is heavily based on the alignment and the set-up of our physical user interface. The physical user interface is an important part of our demonstration system. Our physical user interface is the most interactive piece of this product. Different than the software user interface, the user will be able to directly interact with our physical user interface. Our clients come up with the design for the physical user interface. We think that this is a great set-up to start our project. Since the wedge can be pre-aligned and adjusted, it eliminates some of our burden to find out the delta angle of the sensors. The precision adjustable wedge will also make our demonstration system more user friendly since it can be adjusted without requiring any additional tools.

VII. Interaction View

1) Hardware Communication Design: This section will describe the communication methods between hardware components within the entire system. This section is intended to list out and describe how each hardware component talks with others including communication protocols and connecting methods.

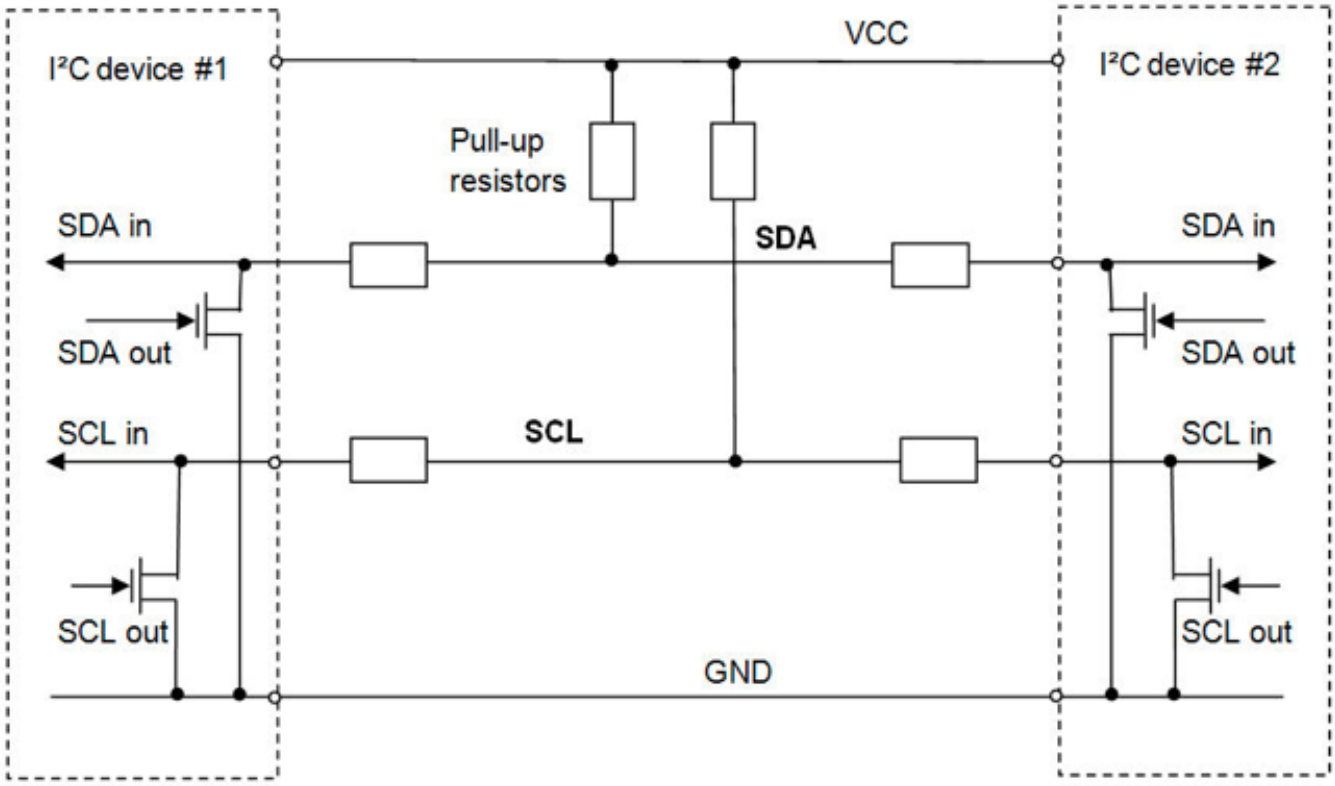
2) Design Concerns: There are two necessary communications/connections for one input-output cycle, one is between the IMU and the microcontroller, the other is between the microcontroller and output display. Refer to the Technology Review for this project, there are three options for choosing the communication protocols between the IMU and the microcontroller: *I2C*, *SPI* and *UART*. Based on our project design, *I2C* protocol is the prior consideration for communicating between IMU and the microcontroller since *Metro Mini 328* has limited pin numbers and *I2C* only takes two pins (*SDA* and *SCL*) for wiring connection and *I2C* provide fair transmission speed. In addition, USB is the appropriate communication method between the microcontroller and output display (computer) since *Metro Mini 328* provides direct USB port and the makes the implementation simple and fast.

3) Design Elements:

a) I2C Protocol: One *I2C* device (IMU) takes totally 4 pins to connect with a master board (microcontroller), besides VDD and GND pins, there are *SDA* and *SCL* which represent ”serial data” and ”serial clock” respectively. Both SDA and SCL are bi-direction wires, that means they can transfer sensor data to the master board, as well as transferring command data from the master board. The data rate has to be chosen between 100 kbps, 400 kbps and 3.4 Mbps, respectively called standard mode, fast mode and high speed mode. Some IC variants include 10 kbps (low speed mode) and 1 Mbps (fast mode +) as valid speeds [14].

In this system, there are more than one IMU sensors (*MPU-9250*) needed to be working together in order to retrieve a higher accuracy data output. Hence, we choose to use *I2C* protocol since it allows multi-devices communication. Additional devices are also known as the slave devices, referring to Figure 8, *I2C* allows *I2C* device #1 and #2 (these represent *MPU-9250* in this project) to communicate with each other by just using one SDA and one SCL wire (besides of VCC and GND), there is no need for extra chip select pins like SPI protocol for extra devices.

Fig. 8. *I2C* Connection between Slave Devices



b) Universal Serial Bus: Metro Mini 328 uses Universal Serial Bus (USB) connection as default communication method between microcontroller and output display (computer). Specifically, *Metro Mini 328* provides a Micro-USB (B) port, the implementation of this connection is simple and fast.

4) Design Constraints: Either the Aircraft or HUD IMU consists of multiple *MPU-9250* sensors units, they can be put parallel together and act as an entity. See as Figure 9, both IMUs will be connected to an *I2C* Multiplexer separately by *I2C* protocol. An *I2C* multiplexer allows the microcontroller to select the particular IMU unit based on the address that the *I2C* Multiplexer assign to that IMU, so that we can easily split two groups of data (from HUD and Aircraft IMU) for the software program. Finally, an output display will retrieve a finalized aligned data from the microcontroller and allows for testing.

VIII. Algorithm View

A. Statistical Analysis Method for Initial Alignment Offset Design

This section will cover the specification of the statistical analysis method for initial alignment of this project. This section is intended to guide the software development team to develop a suitable algorithm. This algorithm is intended to calculate the proper initial alignment data from the sensors under an acceptable degree of certainty.

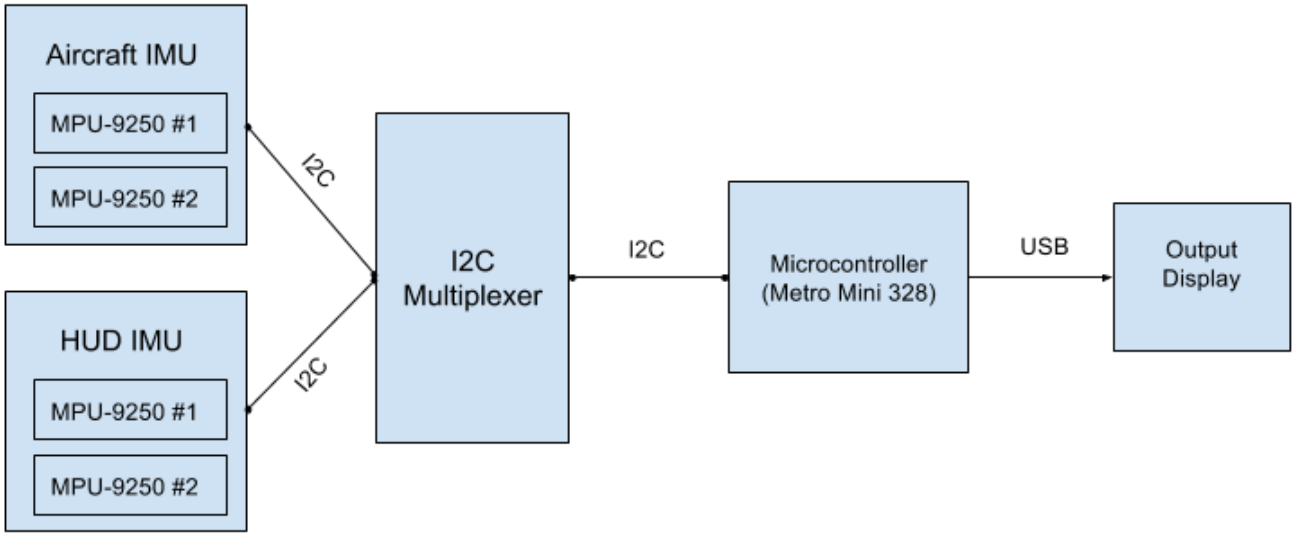
1) Design Concerns: Our primary goal is to find the initial alignment offset. Using a statistical analysis method is desired to ensure an accurate result. This algorithm requires the input of both sensors taken at the same time. A series of inputs will be received by this algorithm until the offset has been found to be within one milliradian of accuracy.

2) Design Elements: This section will cover the algorithm attributes that defines each functionality of the statistical analysis method for initial alignment offset

a) Processing Attributes:

- *Assumption:* The offset algorithm will calculate the correct alignment offset value
- *Prerequisite:* Both sensors must be transmitting data that corresponds to the same time values.
- *Input:* A series of sensor input values.

Fig. 9. Communcation Method between System Hardware Components [14]



- *Output:* An alignment offset value that best represent the alignment error under an acceptable credible interval
- *Acceptable Range:* 95% credible interval
- *Acceptable Accuracy:* unit milliradian

3) Design Rationale: Confidence interval, credible interval, and tolerance interval are three possible options of statistical analysis method that we can use to gain credibility of our alignment data. The confidence interval will give us the frequency of the true value output being generated by the algorithm over a time period. However, until the time of calculation, we are not sure what kind of value that we will get from the algorithm. Hence, leaving us with no parameter to work on confidence interval. The tolerance interval will give us the spread of error in alignment data being generated by the algorithm. This statistical method can be useful for us to further refine our algorithm. Credible interval will give us the credibility to state the certainty of a true value within an interval of data. This method will give us the true value that might be a great representation of our alignment data. We think that credible interval method fits the most with our project. Thus, this makes us decide to do credible interval for our statistical analysis method.

B. Statistical Analysis Method For Dynamic Offset Design

This section will cover the implementation details of the statistical analysis method for the offset error of this project. This section is intended to guide software developer to develop a suitable algorithm. This algorithm is intended to calculate the proper offset error value under an acceptable degree of certainty.

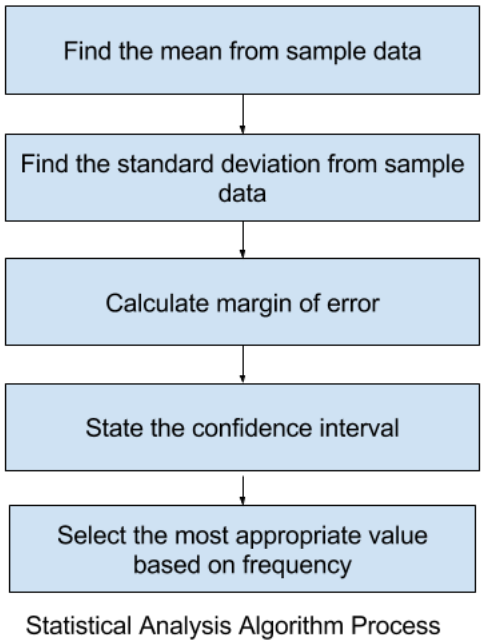
1) Design Concerns: The goal of using a statistical analysis method is to find the credibility of the output that we get from the algorithm. This algorithm will get its input from the offset algorithm. Then, the alignment error data that we get from the offset algorithm will be processed by this algorithm to find the credible interval for a range of offset error values. This algorithm will find the credible interval of the alignment error data and decide which value is best represent the alignment error for that instance. Referring to Figure 10 for specific process details.

2) Design Elements: Elements include assumption, prerequisite, expected input, expected output of this algorithm. This section will also cover the method and formula that we will be using to create this algorithm.

a) Processing Attributes:

- *Assumption:* The offset algorithm will calculate the correct alignment error value
- *Prerequisite:* An alignment error value produced by the offset algorithm
- *Input:* A range of alignment error values
- *Output:* An alignment error value that best represent the alignment error under an acceptable credible interval
- *Acceptable Range:* 95% credible interval
- *Acceptable Accuracy:* unit milliradian

Fig. 10. Statistical Analysis Algorithm Process



3) Design Rationale: Confidence interval, credible interval, and tolerance interval are three possible options of statistical analysis method that we can use to gain credibility of our alignment data. The confidence interval will give us the frequency of the true value output being generated by the algorithm over a time period. However, until the time of calculation, we are not sure what kind of value that we will get from the algorithm. Hence, leaving us with no parameter to work on confidence interval. The tolerance interval will give us the spread of error in alignment data being generated by the algorithm. This statistical method can be useful for us to further refine our algorithm. Credible interval will give us the credibility to state the certainty of a true value within an interval of data. This method will give us the true value that might be a great representation of our alignment data. We think that credible interval method fits the most with our project. Thus, this makes us decide to do credible interval for our statistical analysis method.

C. Offset Algorithm Design

This section will cover the implementation details of the offset algorithm for this project. This section is intended to guide software developer to develop a suitable algorithm. This algorithm is intended to calculate the proper offset between two quaternion inputs.

1) Design Concerns: The resulting output must be the quaternion offset of two quaternion inputs. The algorithm will first take the input of the first input. The next step will be to multiply both inputs against each other. The resulting multiplication will be the desired output.

2) Design Elements:

a) Processing Attributes:

- Quaternion input A: The first of two quaternion inputs
- Quaternion input B: The second of two quaternion inputs
- Quaternion output offset: The resulting offset with the form of quaternion

3) Design Rationale: The design requires an offset in terms of a quaternion output in order to be used within the rest of the system.

IX. Conclusion

This design document has covered six main design views from the perspective of hardware and software, each view contains specific design viewpoints as well as relevant detailed explanations. This document would help and provide necessary plan

information for our project implementation in terms of hardware set-up, algorithm design, testing and debugging. More importantly, this design document allows us to think thoroughly over the entire implementation process, and helps us move forward with in the development of our solutions to the problem.

REFERENCES

[1] "Head-up guidance system," Rockwell Collins. [Online]. Available: https://www.rockwellcollins.com/Data/Products/Displays/Head-Up_Displays-HUD/Head-up_Guidance_System.aspx

[2] D. R. Jones, T. S. Abbott, and J. R. B. II, "Concepts for conformal and body-axis attitude information for spatial awareness presented in a helmet-mounted display," The National Aeronautics and Space Administration, 1993. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930014219.pdf

[3] "Inertial navigation system, a.k.a inertial reference unit," Earth Observing Laboratory, 2005. [Online]. Available: https://www.eol.ucar.edu/instruments/inertial-navigation-system-aka-inertial-reference-unit

[4] "What is mems technology?" MEMS and Nanotechnology Exchange. [Online]. Available: https://www.mems-exchange.org/MEMS/what-is.html

[5] "A beginners guide to accelerometers dimension engineering," Dimension Engineering LLC. [Online]. Available: http://www.dimensionengineering.com/info/accelerometers

[6] A1RONZO, "Gyroscope," SparkFun Electronics. [Online]. Available: https://learn.sparkfun.com/tutorials/gyroscope

[7] SFUPTOWNMAKER, "I2c," SparkFun Electronics. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c

[8] "A brief introduction to the serial peripheral interface (spi)," Arduino. [Online]. Available: https://www.arduino.cc/en/Reference/SPI

[9] Jimbo, "Serial communication: Uarts," SparkFun Electronics. [Online]. Available: https://learn.sparkfun.com/tutorials/serial-communication/uarts

[10] "The multiplexer," ElectronicsTutorials. [Online]. Available: http://www.electronics-tutorials.ws/combination/comb_2.html

[11] "Mpu-9250 product specification revision 1.0," InvenSense. [Online]. Available: https://cdn.sparkfun.com/assets/learn_tutorials/5/5/0/MPU9250REV1.0.pdf

[12] "Latitude and longitude facts," Worldatlas. [Online]. Available: http://www.worldatlas.com/aatlas/imageg.htm

[13] W. Lahr, 2016.

[14] "Introduction to i2c and spi protocols," Byteparadigm. [Online]. Available: http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/

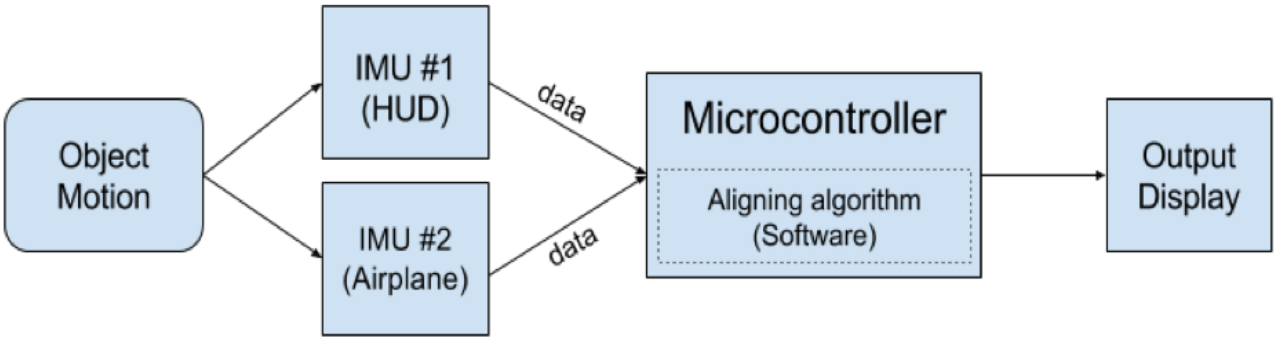
[15] "Ieee recommended practice for software requirements specification (ieee std 830-1998)," IEEE Computer Society, 1998.

[16] C. Rapids, "Head-up guidance system (hgs) for midsize and light business aircraft," Rockwell Collins. [Online]. Available: https://www.rockwellcollins.com/-/media/Files/Unsecure/Products/Product_Brochures/Displays/Head_up_displays/HGS-3500_White-Paper.ashx

[17] "Adafruit pro trinket - 5v 16mhz," Adafruit. [Online]. Available: https://www.adafruit.com/products/2000

[18] "Sparkfun imu breakout - mpu-9250," SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/13762

Fig. 1. New Data Flow diagram (Hardware)



A. Necessary Change Based on Original Design

1) Change in IV.COMSITION VIEW: **Hardware Configuration Design**

The original diagram showed that the Aircraft IMU outputs aligned raw data to the microcontroller; what we did was both IMU would only output raw data and the alignment is measured/calculated by the microcontroller as figure 1.

2) Change In VI.INTERFACE VIEW: **Software User Interface Design**

Our previous GUI have a 2D gauge-like looks to it. It looks more like a car dashboard rather than a real plane HUD. This GUI serve our purpose of showing the data well. However, it is hard to visualize the change on the plane for each angle. Our team is determined to give the best for this project and decided to improve on our GUI. To solve this problem, we create a new GUI as figure 2 that visualize the change of angle to a plane object. We create this GUI by using a python sublanguage called VPython. We also use the pySerial library to connect the GUI to the Arduino part of the project.

3) Change In VIII Algorithm View: **Statistical Analysis Method for Initial Alignment Offset Design & Offset Algorithm Design**

For our statistical analysis, we applied a 95% confidence interval to our offset data to ensure that the resulting offset was precise. By using a confidence interval, we can retrieve a valid offset value as soon as the quaternion value converges. Without using a confidence interval, we could end up with an erroneous value.

Our implementation originally stored 60 samples each for the yaw, pitch and roll offset before using those values to find the confidence interval. Unfortunately, the microcontroller's memory is limited, which makes finding a precise value difficult. We were able to increase the number of samples and increase our performance by taking the offsets of each yaw, pitch and roll sequentially, thus tripling our sample sizes.

Currently, the confidence interval is only being applied to the offsets between each device. Further improvements may be gained by increased use of statistical analysis. One such improvement would be to apply our analysis during calibration. The current process relies on a simple average of a single stream of data.

Original Tech Review

Computer Science Senior Software Engineering Project (CS462)

Winter 2017

Abstract

This project is a proof concept to explore a potential technological innovation for Head-Up Display (HUD) system that present critical flight information to pilots. The primary objective of this project is to reduce the cost and time required to precisely align flight information to the HUD by introducing additional sensor to the system to make the alignment process more dynamic. To achieve this goal, there are eight different main technologies that will be critical for the development of the project. This document will compare three alternative options for each main technologies. This document will also include the option that we choose for each main technologies to develop this project.

Contents

I. Introduction

There are eight different main technologies that will be used or involved in this project. These technologies contain using of both software and hardware. For instance, for hardware, we will compare and discuss about some possible hardware options for building the demonstration system; for software, we will compare and discuss different algorithms or approaches for resolving the problem. Each of us takes an authorship (responsibility) for three technologies:

Jiongcheng (Roger):

- Hardware selection of Microcontroller
- Hardware selection of Represented MEMS IMU
- Hardware selection of Communication Protocol Type

Krisna:

- User Interface Toolkit
- Programming Language
- Statistical Analysis Method

Drew:

- IRU Data Representation
- Filter Technique

II. Technologies

A. Hardware selection of Microcontroller

1) Context: It is critical to choose the most appropriate microcontroller for this system since there are limitations from both hardware and software perspectives. A microcontroller plays an important role in the system, which it will have following functionalities in the system:

- Process input from the IMUs and output to the computer/display
- Process the alignment algorithm
- Debugging and testing purpose

2) Options: We consider to use a board between *Adafruit Pro Mini* [1], *Metro Mini 328* [2] and *InvenSense MPU-9250 CA-SDK Reference Board* [3]. Anyone of these board has its outstanding areas and lack, and we will choose one of them based on the following criteria.

3) Criteria:

1) **Support Language**: We expect to use a general programming language for the microcontroller since using familiar programming language will reduce our time on additional research and the save time on working on the algorithm itself.
2) **Clock Speed**: The alignment algorithm is required to within around 500 milliseconds time from taking the input from the IRUs to the output of aligned data, that require the board has a fast speed and running any processes.
3) **I2C Protocol**: This is necessary to have on the board since the MPU-9250 (our selected IMU) require I2C protocol to communicate.
4) **Connection with PC**: This is necessary to have on the board since we will use computer to any software program.
5) **Size**: Size is less important since this system is for demonstration purpose.
6) **Cost**: This is relatively important but as long as the price is within the expected budget, that will be acceptable.

4) Table of Detailed Comparison:

Model	*Arduino Pro Mini*	*Metro Mini 328*	*InvenseSense MPU-9250 CA-SDK Reference Board*
Core Chip	ATmega328	ATmega328	Texas Instrument MSP430
Support Language	C	C	C/C++
Clock Speed	8MHz (3.3V mode)	16MHz (3V mode)	16MHZ (1.8V 3.6V)
I2C Protocol/Number	Yes/1	Yes/1	Embedded Structure
Connection with PC	USB	FTDI/USB	USB/Bluetooth
Size	33mm x 18mm	18mm x 44mm x 4mm	Unknown
Cost (U.S Dollar)	$9.95	$12.50	$440.00
Advantage	Small size	Faster clock speed	With embedded IMU
Shortage	Slower Clock Speed	Lack of resource of guidance/datasheet	High cost

5) Overall Discussion: By comparing these boards by the above criteria, *Arduino Pro Mini* and *Metro Mini 328* have similar performance and specification, they both have the advantages of small size and low cost. *InvenseSense MPU-9250 CA-SDK Reference Board* is a special case, which its a board as well as the IRU itself since it has embedded on-board sensors. *MPU-9250 CA-SDK is very powerful* for multi-sensor system, other than an embedded MPU-9250 (accelerometer, gyroscope and compass), it also has pressure sensor, UV sensor, humidity and temperature sensor, light and proximity sensor. However, high cost is a noticeable shortage of *MPU-9250 CA-SDK*. In addition, and not all of its functionalities are necessary for this project. Now, we can only look at *Metro Mini 328* and *Arduino Pro Mini*. Based on the comparison and the criteria, *Metro Mini 328* will be more preferable. Because this module has a faster clock speed compares to the *Arduino Pro Mini*, although it has a larger size but this doesnt affect to its actual performance based on our project requirement, other than that, all other concerned criteria are same as the *Arduino Pro Mini*.

B. Hardware selection of Represented MEMS IMU

1) Context: An Inertial Measurement Unit (IMU) is the most significant part of the hardware components of this system. An MEMS IMU will be used to measure the acceleration, velocity position of the aircraft and output data for alignment algorithm. Therefore, we are looking for a model of IMU that is highly accurate, well performed as well as with affordable expense for the demonstration of this project, following chart compare these three model with a list of comparison.

2) Options: We are looking at three models that are considered to be the most likely options for representing MEMS IRU. *9DoF Sensor Stick* [4], *MPU-9250 IMU* [5] and *9DoF IMU* [6] are three models of IMU. All these three IMUs are with 9 degrees of freedom/9 axis sensor, that indicates all of these IMUs contain MEMS sensors of accelerometer, gyroscope and magnetometer(compass), which is a basic requirement for choosing the IMU in this project. Following are some specific criteria that we look and compare for choosing the most proper MEMS IRU.

3) Criteria:

1) **Operating voltage range**: This is determines based on the model we use for the microcontroller, which an operable voltage range of an IMU should be less than the output voltage of its microcontroller.
2) **Support I2C protocol**: This is significant since I2C could be the only protocol for an IMU to communicate with the microcontroller.
3) **Accelerometer output resolution**: This will determine the accuracy of the acceleration data output.
4) **Gyroscope output resolution**: This will determine the accuracy of the angular velocity data output.
5) **Magnetometer output resolution**: This will determine the accuracy of the orientation data output.
6) **Cost**: This is relatively important but as long as the price is within the expected budget, that will be acceptable.

4) Table of Detailed Comparison:

Model	*9DoF Sensor Stick*	*MPU-9250 IMU*	*9DoF IMU*
MEMS Sensors	Accel.: ADXL345 Gyro: ITG-3200 Magn.: HMC5883L	MPU-9250	LSM9DS1
Operating Voltage Range	2.1 — 3.6V	2.4 — 3.6V	1.9 — 3.6V
Output Type	Unknown	16 bits ADC	16 bits ADC
Support I2C Protocol	2g, 4g, 8g, 16g	2g, 4g, 8g, 16g	2g, 4g, 8g, 16g
Gyroscope Output Resolution (Angular Velocity)	Full scale = 2000 degree/s	250, 500, 1000, 2000 degree/s	250, 500, 1000, 2000 degree/s
Magnetometer Output Resolution (gauss)	8 gausses	48 gausses	4, 8, 12, 16 gausses
Size	22.22 X 18.48 mm	Unknown	14.3 mm X 20.5 mm
Cost (U.S. Dollar)	$49.95	$14.95	$24.95
Advantage	Higher output resolution	Least expensive	Wide spectrum of sensor range
Shortage	High expense	Weaker performance	A little expensive

5) Overall Discussion: In overall, these three models have similar performance and specification of out resolution, however, *9DoF Sensor Stick* will be the least preferred option since it has the highest expense and it lacks of resources of its datasheet the relevant guidance. By comparing between *MPU-9250 IMU* and *9DoF IMU*, they both have specific datasheet that can be found, and their prices are both within the acceptable range, so by looking at more specific detail of their hardware performance, *9DoF IMU* has wider sensor range to choose for its output resolution, which is more functional than the *MPU-9250 IMU*. Therefore, the *9DoF IMU* model would be the first preference for selecting the represent MEMS IMU.

C. Hardware selection of Communication Protocol Type

1) Context: There are many different types of protocol for hardware component devices communicating with each other. We chose three different protocol types that are the most common types to use, as well as doable in our hardware system.

2) Options: I2C (Inter-Integrated Circuit), *SPI (Serial Peripheral Interface)* and *UART (Universal Asynchronous Receiver/Transmitter)* are three possible communication protocol types for our project, each of them has its own advantages and lack. In regards to this project, we will consider the following criteria for choosing the most appropriate protocol type for our hardware devices communication [7].

3) Criteria:

1) **High Transmission Speed**: This is one of the most critical criteria to be consider, since our hardware system requires multi-devices to process near synchronously, a fast communication speed between hardware devices is necessary.
2) **Transmission Distance**: This criterion is less important within this system since we consider all of the hardware devices will be implemented as a whole device, that means transmission distance of the protocol type wont be a necessary criterion.
3) **Multi-Devices Support**: Since we may use more than 1 IMU to work together in order to gathering more accurate data, the protocol we choose to use must support multi-devices communication.
4) **Number of wire needed to connect with microcontroller**: This criterion is less important but less number of wires between hardware device provides higher portability of the system, and it also reduces the difficulties on the hardware set-up process.

4) Table of Detailed Comparison:

Protocol Type	*I2C/IIC*	*SPI*	*UART*
Transmission Speed (Standard Speed Mode)	>1 Mbit/s	∼10 Mbit/s	0.3 Kbit/s ∼ 1 M bit/s
Transmission Distance	Short (Within integrated circuit components)	Short (Within integrated circuit components)	Long (wireless)
Multi-Devices Support	Yes	Yes	Yes
Number of Wire Needed	2	3 + 1 for each signal line	none
Advantage	Easy to implement	Fast transmission speed	can be implemented in wireless environment
Shortage	Slow transmission speed	More number of wires needed	Hard to implement

5) Overall Discussion: By comparing the above criteria, we first may eliminate UART because it has the slowest transmission speed within these three types, so its not expected for using in our system. SPI has significant fast speed however since we will use multiple IMUs connecting together in the system, SPI doesnt provide a good portability that it requires 1 more physical wire for each additional devices. Therefore, I2C is the most preferable option, it has relatively high transmission speed, and its easy to implemented for a complicated system.

D. User Interface Toolkit

1) Context: Graphical user interface will be a crucial of for our project presentation and demonstration system. The algorithm from our project will output a raw alignment data and it will be hard to understand the output without the help of graphical user interface. The graphical user interface will put contexts to the data and presents the correct symbology of the alignment data output. User interface toolkits is needed to develop our project user interface. We are looking for a user interface toolkit that will not add more complexity to the project, have minimal impact on the development process of the project, and have a useful library for user interface functionality.

2) Options: There are three options for the user interface toolkit. The first option is the visual studio user interface toolkit. Visual studio user interface toolkit is a built-in toolkit in visual studio that used for programmer to create a window application user interface. The second option is GTK+ toolkit. GTK+ toolkit is an open source cross-platform widget toolkit for creating graphical user interfaces [8]. The last option is the IUP portable user interface. IUP is a computer software development kit that provides a portable, scriptable toolkit to build a graphical user interface [9].

3) Criteria:

1) **Complexity**: Our clients encouraged us to spend most our time in developing and refining the dynamic alignment algorithm. Having a less complex toolkit will give us extra time to work on the alignment algorithm.
2) **Accessibility**: Toolkit and library that is easily accessible is desired in this project. A free and easy to access toolkit is desired.
3) **Adoption Rate**: Toolkit that well adapted in the programmer community will have more resources for debugging and refining our user interface.
4) **Time Commitment**: Using a user interface toolkit that will require huge learning curve and time to learn it will not add much benefits to the overall project.
5) **Library**: More powerful library will have more tool to further refine our graphical user interface.
6) **Overall cost and benefits**: Key strengths and weaknesses of using a particular user interface toolkit.

4) Table of Detailed Comparison:

Toolkit Name	Visual Studio Toolkit	GTK+	IUP
Complexity	Least complex	Less complex	Most complex
Accessibility	Accessible for free. Already have it in our computer.	Accessible for free online.	Accessible for free online.
Adoption Rate	Have a lot of tutorials and references online	The most popular toolkit for graphical user interface.	Less popular than GTK+ toolkit.
Time Commitment	Minimal	Medium	High
Library	Least powerful	Less powerful	Most powerful
Overall Cost and Benefit	Simple and easy to use toolkit. The least powerful toolkit, with the least learning curve.	More powerful option than visual studio, simpler than IUP.	The most powerful option. A complete development tool. Have the most cost and learning curve

5) Overall Discussion: By comparing criteria visual studio toolkit is the simplest and ready to use user interface toolkit option. All of the user interface toolkit options are easily accessible and available for free online. Although, IUP is the most powerful option that we have, it will also bring a lot of complexity to our project. IUP is a development tool that completely different than visual studio, using IUP will change the workflow of the development process. The learning curve and time commitment to learn IUP is huge and it will reduce our time to work on the algorithm of the project. Although GTK+ has the best overall cost and benefits, there is learning curve associated with it and it will also reduce our time to work on the algorithm of the project. Although a user interface is a critical part in our project presentation, this functionality is not a critical part of the project as a whole. Our clients strongly emphasize the quality of our algorithm. The most important piece of this project is the dynamic alignment algorithm and our clients encouraged us to spend most our time in developing and refining the dynamic alignment algorithm. Thus, using the toolkit with the least learning curve is encouraged for our project. This makes us decided to use the built-in visual studio toolkit to create our projects graphical user interface.

E. Programming Language

1) Context: We are looking for a better programming language that will help us to improve the quality of our project in terms of complexity, speed, and memory allocation. The goal of using a particular programming language is to reduce the complexity of writing code for our project, while having an acceptable speed and spaces memory. Programming language that has compatibility with the hardware is also desired.

2) Options: Our clients give us freedom to choose the programming language that we want to use to create this project. There are three options for programming language that we can use for this project. The options for the programming languages are C, C++, and Assembly language.

3) Criteria:

1) **Language Orientation**: Language orientation will change the way we implement the algorithm. More familiar language orientation is desired.
2) **Language Level**: Low-level programming language is a programming language that provides little or no abstraction from a computer's instruction set architecture [10].
3) **Complexity**: This criteria will determine the difficulty and complexity of implementing algorithms in a particular programming language. Less complex programming language is desired.
4) **Speed**: This criteria will determine the running speed of a compiled software written in a particular programming language. A fast running software is desired for this project.
5) **Memory Allocation**: This criteria will determine the memory allocation required to store a compiled software written in a particular programing language. This criteria will also determine the memory usage of a software during running time. More efficient memory allocation is desired for this project.
6) **Hardware Compatibility**: This criteria will determine the communication efficiency between the software and the hardware. More compatible programming language is desired for this project.
7) **Overall Cost and Benefit**: Key strengths and weaknesses of writing software in a particular programming language.

4) Table of Detailed Comparison:

Language Name	C	C++	Assembly
Language Orientation	Object oriented	Object oriented	No orientation
Language Level	High level language	High level Language	Low level language
Complexity	Medium	Medium	Complex
Speed	Fast	Fast	Fastest
Memory Allocation	More efficient	Most efficient	Most efficient
Hardware Compatibility	More Compatible	Compatible	Most Compatible
Overall Cost and Benefit	The best high level programming language to interact with hardware	The less hardware compatibility version of C	The most complex programming language with the best performance to communicate with hardware

5) Overall Discussion: Based from the criteria above, it is clear that assembly language has the best performance and benefits in writing a program that works with hardware. However, the low level language of assembly makes writing code in assembly really complex. With little or no abstraction from a computer's instruction set architecture, this will slow our development time for the algorithm of this project. C++ is a great programming language that is adopted greatly in the programmer community. However, in terms of our project requirement, it is also clear that C++ is the less hardware compatibility version of C. C is arguably one of the best high level programming language to interact with hardware. This makes us decide to use C as our programming language for this project.

F. Statistical Analysis Method

1) Context: The goal of using a statistical analysis method is to find the credibility of the output that we get from the algorithm. There are a lot of options of using statistical method to further refine our alignment data. We are looking for a statistical analysis method that will allow us to find the credibility on our alignment data or error tolerance.

2) Options: Confidence interval, credible interval, and tolerance interval are three possible options of statistical analysis method that we can use to gain credibility of our alignment data. Confidence interval will measure the frequency of repeated events [11]. How much true value that we get from running the test repeatedly. Credible interval will give us the degree of certainty about a values, given the observed data, there is a probability that the true value of falls within the credible region [11]. Tolerance interval will give us the spread of error in our alignment data.

3) Criteria:

1) **Type of statistics**: Each statistics have their own philosophy of finding credibility of their data. We will determine which type of statistics is most suitable for the project.
2) **Parameters**: In our project case, the parameter will be the expected value of alignment data.
3) **Data**: The data will be the alignment data outputs that we get from the algorithm.
4) **Bound**: We are looking to achieve error within an acceptable bound for confidence and credible interval. Tolerance interval will give us bound values of the sample after the calculation.
5) **Result**: The result will vary for each method. We will determine which result will be the most appropriate.

4) Table of Detailed Comparison:

Method Name	Confidence Interval	Credible Interval	Tolerance Interval
Type of Statistics	Frequentist	Bayesian	Can be both
Parameters	Fix	Random	Fix
Data	Random	Random	Fix
Bound	Fix	Fix	Random
Result	Frequency of true value output	Degree of uncertainty about a value	Statistics of acceptable error tolerance

5) Overall Discussion: The confidence interval will give us the frequency of the true value output being generated by the algorithm over a time period. While being run repeatedly, our algorithm will produce the same result, which implies the consistency and the credibility of our algorithm data. However, until the time of calculation, we are not sure what kind of value that we will get from the algorithm. Hence, leaving us with no parameter to work on confidence interval. The tolerance interval will give us the spread of error in alignment data being generated by the algorithm. Tolerance interval can be a great way for us to make sure that we have fulfil the error tolerance requirement for this project. This statistical method can also be useful for us to further refine our algorithm. We can adjust the alignment data by the error interval that tolerance interval gave us to make our alignment data more accurate. Credible interval will give us the credibility to state the certainty of a true value within an interval of data. This method will give us the true value that might be a great representation of our alignment data. We think that credible interval method fits the most with our project. Thus, this makes us decide to do credible interval for our statistical analysis method.

G. IRU Data Representation

1) Context: Our solution requires an accurate reference point to compare against when determining the correct alignment offset. The reference point in an aircraft is the data output from the aircrafts IRU. Since access to an expensive IRU is limited, we will need to find another method that is sufficient to test against.

2) Options: There are three simulation techniques to represent the sensor output from the aircrafts IRU for our demonstration system. The first technique would be to use a cheap MEMS for the sensor data when tracking motion. The second technique would involve redundant MEMS and an algorithm to average across for higher accuracy. The third technique would be to fully simulate the IRU data [12]. This last technique would require the precise input of motion that could be simulated at runtime [13].

3) Criteria:

1) **Accuracy**: As the IRU is used as a reference point, the method to represent the IRU's data must maintain acceptable accuracy.
2) **Cost**: Cost is a driving consideration as the IRU is already too expensive. The chosen method should stay within our budget.
3) **Difficulty**: The chosen method should be one in which we are able to implement within the allotted timeframe.

4) Table of Detailed Comparison:

Method	Single MEMS Sensor Data	Redundant MEMS Improved Sensor Data	Fully Simulated Sensor Data
Accuracy	Low	Medium	High
Cost	Low	Medium	Medium/High
Difficulty	Low	Medium	Low/Medium

5) Discussion: The single MEMS sensor data simulation technique is the most predictable and straightforward. This technique is dependent on the chosen hardware. Although the redundant MEMS improved sensor data simulation technique also depends on the chosen hardware, the accuracy may be higher as a result of overlapping data being used to reduce error. The noise reduction for redundant MEMS is expected to be 1/sqrt(n). Lastly, the fully simulated sensor data technique would ideally provide the highest accuracy. The approach would be to mount the demonstration system to a MEMS testing device. The simulated data would match the scripted motion of the testing device at runtime. This approach could be less difficult but the cost is currently unknown.

6) Selection: At this time we are planning on following the single MEMS sensor data simulation technique as the cost and difficulty are both low. Although the accuracy is less than the other two methods, it can be improved with the selection of hardware.

H. Filter technique

1) Context: In order to find the alignment offset we need data that represents the HUD position in 3d space. We have chosen to use a quaternion output when calculating position as it provides the necessary information. Not only must the output be in a usable form, it is also desired to be of high accuracy. Individual sensor output will have some amount of error as specified by hardware. By combining sensor data, performance issues can be reduced [14].

2) Options: Three options are presented to improve MEMS accuracy. The first option is to the Extended Kalman Filter (EKF). The next option is to use the Mahony Filter. Lastly, we could simply enable the quaternion output mode if supported by hardware such as Invensenses Digital Motion Processor (DMP).

3) Criteria:

1) **Accuracy**: When trying to find the correct alignment offset, accurate positional data is required.
2) **Complexity**: Given that our budget limits the available hardware we have access to, the chosen algorithm must be quick enough to run in realtime.
3) **Availability**: The choice must be available for use in our demonstration system.

4) Table of Detailed Comparison:

Method	**EKF**	**Mahony**	**DMP**
Accuracy	High	Medium	Medium
Complexity	High	Medium	Low
Availability	Open Source	Open Source	Hardware Dependent

5) Discussion: The EKF provides the highest accuracy among the other two options [15]. The high accuracy can be attributed to the high complexity of the algorithm itself. This high complexity is the main drawback to choosing EKF as our hardware must be able calculate the result fast enough to keep up with its output. The EKF is available as open source. Next, similar accuracy is achieved by both the Mahony Filter and Invensense DMP. While the Mahony Filter is also open source, the Invensense DMP is dependent on hardware as the chip is only on select boards [3]. Both the Mahony Filter and Invensense DMP are able to run efficiently on most boards where applicable.

6) Selection: At this time we are considering the MPU-9250 IMU which supports DMP. We will plan on using this feature unless a change of hardware is made. Higher performance equipment will enable us to use the EKF in place of the Mahony Filter.

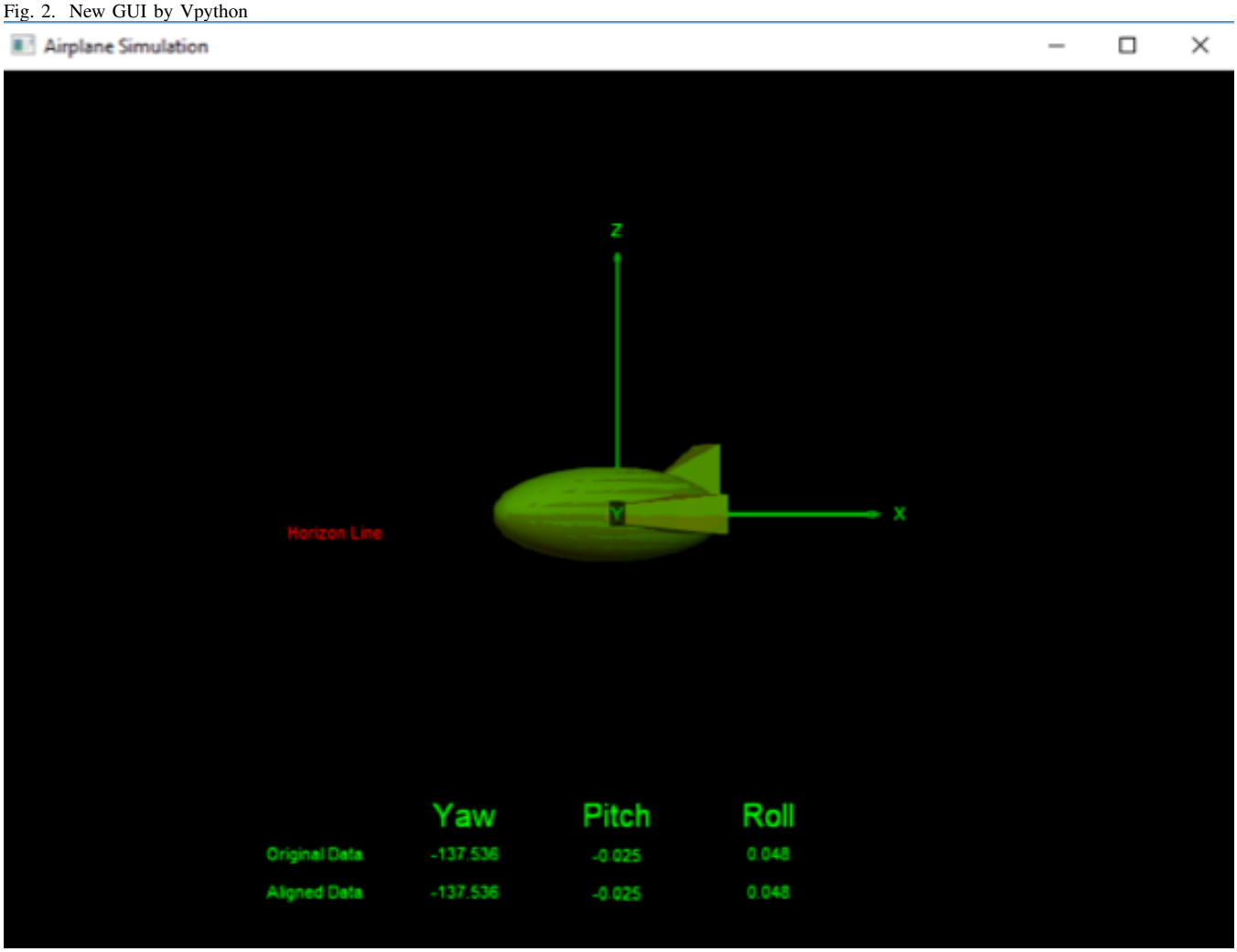
III. Conclusion

Throughout this technology review we covered each of the eight different technologies that were deemed critical to our project. Each technology was provide three potential options to choose between for inclusion in our proposed solution. Before making any decisions, we performed research to effectively compare the options against each other. As a result we have made the appropriate implementation decisions to move forward with in the development of our solution.

REFERENCES

[1] "Arduino pro mini 328 - 3.3v/8mhz," Sparkfun Electronics. [Online]. Available: https://www.sparkfun.com/products/11114
[2] "Adafruit metro mini 328 - 5v 16mhz," Adafruit. [Online]. Available: https://www.adafruit.com/product/2590
[3] "Invensesense mpu-9250 ca-sdk reference board user guide," InvenSense Inc, 2012. [Online]. Available: https://store.invensense.com/datasheets/invensense/MPU-9250CA-SDK.pdf
[4] "https://www.sparkfun.com/products/10724," Sparkfun Electronics. [Online]. Available: SparkFun9DegreesofFreedom-SensorStick
[5] "Sparkfun imu breakout - mpu-9250," Sparkfun Electronics. [Online]. Available: https://www.sparkfun.com/products/13762
[6] "Sparkfun 9dof imu breakout - lsm9ds1," Sparkfun Electronics. [Online]. Available: https://www.sparkfun.com/products/13284
[7] "Introduction to ic and spi protocols." [Online]. Available: http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/
[8] "Gtk+," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/GTK%2B
[9] "Iup (software)," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/IUP_(software)
[10] "Low-level programming language," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Low-level_programming_language
[11] "Confidence vs. credibility intervals." [Online]. Available: http://freakonometrics.hypotheses.org/18117
[12] "Improving accuracy with multiple sensors: Study of redundant mems-imu/gps configurations," Swiss Federal Institute of Technology Lausanne. [Online]. Available: https://www.researchgate.net/profile/Stephane_Guerrier/publication/255962728_Improving_accuracy_with_multiple_sensors_Study_of_redundant_MEMS-IMUGPS_configurations/links/00463520ff0b1c5507000000.pdf?origin=publication_list
[13] "Precision testing for mems accelerometers," Nist. [Online]. Available: https://www.nist.gov/news-events/news/2016/04/precision-testing-mems-accelerometers
[14] "Solutions for mems sensor fusion," Mouser. [Online]. Available: http://www.mouser.com/applications/sensor_solutions_mems/
[15] "Experimental comparison of sensor fusion algorithms for attitude estimation." [Online]. Available: http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2014/media/files/1173.pdf

Fig. 2. New GUI by Vpython



A. *Necessary Change Based on Original Technology*

1) Change in D.User Interface Toolkit: Our previous GUI have a 2D gauge-like looks to it. It looks more like a car dashboard rather than a real plane HUD. This GUI serve our purpose of showing the data well. However, it is hard to visualize the change on the plane for each angle. Our team is determined to give the best for this project and decided to improve on our GUI. To solve this problem, we create a new GUI that visualize the change of angle to a plane object. We create this GUI by using a python library called VPython. We also use the pySerial library to connect the GUI to the Arduino part of the project, an new overall review seen as following:

- **Complexity:** least complex
- **Accessibility:** Free open source
- **Adoption Rate:** Few tutorials or examples
- **Time Commitment:** Minimal
- **Library:** Powerful
- **Overall Cost and Benefit:** Easy but powerful, but too few online tutorials could be found

2) Change in F.Statistical Analysis Method: For our statistical analysis, we ended up applying a 95% confidence interval to our offset data to ensure that the resulting offset was precise. Confidence interval is easier to be programmed but it also works well. In addition, by using a confidence interval, we can retrieve a valid offset value as soon as the quaternion value converges. Without using a confidence interval, we could end up with an erroneous value.

VI. Weekly blog post

A. Fall 2016

1) Jiongcheng Luo (Fall 2016):

Week 3 (Oct. 10 ~ 14, 2016)

- **Plans for the coming week**
  My primary plan for next week is to complete the problem statement that is satisfied by the clients also have them sign on it. The next step is to understand the problem more in depth in the technical perspective, I hope to illustrate the problem by using mathematical and physics way, and be able to translate into CS problem such as what would be software program that we will build for the project.

- **Progress since last week**
  The most important progress I've made was I have built a better relationship with my teammates I have known them better. I also have a more clear understanding on the problem that we are trying to solve for the given project. We completed our first draft of problem statement even though it didn't meet the requirement from our clients, but it helped me comprehend the entire problem more deeply.

- **Any problems I encountered**
  It's hard to follow the agenda of our client, which we always had to wait for an uncertain long time for getting their email back, which may affect to our work progress in the future. In addition, the scheduling within our group is not settled yet, we have not yet set up a fixed time for the group meeting.

Week 4 (Oct. 16 ~ 21, 2016)

- **Plans for the coming week**
  By next week, the primary goal is to get the problem statement fixed as the expectation of the client, that in terms of the consistence of the entire statement. We will include more explanation in plain language to the technical terms. After that, I plan to do more and deeper researches, and start to working on the first design procedures to the problem.

- **Progress since last week**
  My progress is I understood how a team collaboration is so important to the success of this project, we have been getting closer as an entity and I started know the the character of each team member: what each of them good at and lack, that understanding helps me to know how to make better complement for each of us.

- **Any problems I encountered**
  The problem I met at this moment is how to improve the relationship between us and the client, since we had an issue that has led to breakdown of our relationship. Besides, I still have unclear problems for our projects such as the I am still not clear about the real time requirement for our algorithm, or like what kind of data we will get for test.

Week 5 (Oct. 23 ~ 28, 2016)

- **Plans for the coming week**
  The primary goal of next week is to complete the requirement document, which requires us to start thinking and planing for working on the project. I also plan to have one or two meetings with our client in regards to the requirement documents.

- **Progress since last week**
  My biggest progress from last week was that we have completed the problem statement as the expectation from our clients. And by that, I have become more familiar with the terminologies of our project, I have a clear picture in my mind about the problem that we are dealing with.

- **Any problems I encountered**
  I am still not sure about the resolution to our project, such as what kind of software and hardware we are going to

play with. In other word, I am still not clear about our procedures for doing this project.

Week 6 (Oct. 31 ˜ Nov. 5, 2016)

- **Plans for the coming week**
  By next week, I hope finish up the requirement document including all the sections/subsections; and I plan to decided which hardwares (boards) to use for our project and start thinking to purchase. And after decide which board to use, I plan to start looking at more detail of the board and maybe starting do some simple coding simulation.

- **Progress since last week**
  By writing on the requirement document, I know much better about the specific points of the project such as the detail workflow, input/output of the product and restriction, etc. In addition, we have known about what sort of boards that we are using for building the product, which helped me narrow the learning process so that I know what to look up and learn about.

- **Any problems I encountered**
  We are still struggling about some of the detail information about the product when writing on the requirement document, many specific points are unknown until we start the implementation.

Week 7 (Nov. 7 ˜ Nov. 11, 2016)

- **Plans for the coming week**
  My plan for the next week is to finish up the the technology review, that's not only for the writing part, but also plan to list out and truly understand all the technologies that we may use as well as how to use these technologies for our project. Also, I plan to list out all the hardware that we are going to implement on and prepare to purchase for those.

- **Progress since last week**
  We have eventually finished the requirement document by last week and move on to the technologies review, by writing up the requirement document, I have a deeper understanding to the restriction and the problem that we may meet during the real implementation process.

- **Any problems I encountered**
  We still have no solutions or ideas for solving some of the specific problems. For example, we need two groups of input data and one of them represent the correct aligned "Aircraft" IRU data for this project, how do we get the "correct aligned" data, what reference do we take to assume those data we come up is correct?

Week 8 (Nov. 14 ˜ Nov. 18, 2016)

- **Plans for the coming week**
  By this week, I plan to start up and hopefully finish a rough draft by the end of the week. I also nail down the all the hardware that we are going to use and send out the list to Kevin. In addition, I plan to send out an email to our client to report our progress and ask about some questions: 1. How to get correct aligned data as reference? 2. Ask for generic HUD symbology picture 3. GitHub Account.

- **Progress since last week**
  We have finished up our technologies review and we have discuss about the question about how to get correct aligned data as reference, even though we do not if our assumption will be correct or doable or not when doing the real implementation. But we assume we can "make" a group of correct aligned data by manually adjusting it and assume this data is correct, so that we will let the other group of data to be correctly aligned based on this assumption.

- **Any problems I encountered**
  The problem I had so far is for getting the correct aligned data as reference, and we have to figure out the hardware we going to use.

2) Drew Hamm (Fall 2016):

Week 3 (Oct. 10 ˜ 14, 2016)

- **Plans for the coming week**
  I want to start looking into the hardware we might be working with for this project. Specifically I will be reading the specifications for both the Motion Sensor Evaluation Board: MPU-9250CA-SDK and the Ellipse-D: Miniature Dual GPS INS. Besides familiarizing myself with the hardware I want to learn about Quaternions as advised by our client in order to better understand the output data we will be working with. Lastly, I plan on finishing up my section of the problem statement as well as working with my team to finish it as a whole.

- **Progress since last week**
  Met up with team members to work on the problem statement and finish a first draft. Received feedback from clients further specifying the project details.

- **Any problems I encountered**
  I found out that my first understanding of the project was incorrect. At first I thought we were mostly working on a proof of concept. I realized my understanding of hardware error was poor. Since our project requires accurate results, I need to spend some time to understand how much error might be expected from our solution.

Week 4 (Oct. 17 ˜ 21, 2016)

- **Plans for the coming week**
  I'll be working together with the group in order to finish up a new revision of our problem statement. We want to address a couple of our clients concerns in order to get their signed approval. We have a meeting with our client on Wednesday to which I want to prepare questions for. We should also be able to get our problem statement signed so we can move on to working on the requirements document.

- **Progress since last week**
  Met with group to clear up some miscommunication we had with our client. Worked on on the problem statement along with research on both MEMS IRU and aircraft IRU specifications.

- **Any problems I encountered**
  The technical writing required to create the problem statement has been difficult. Most of this difficulty is due to having to learn the specialized terminology as well as hardware that we will be working with.

Week 5 (Oct. 24 ˜ 28, 2016)

- **Plans for the coming week**
  I will be working on the requirements document. This will also involve getting together with the group and doing more research in order to fully explain our solution. We will need to get in touch with our client once we have a working draft of the document.

- **Progress since last week**
  Finished our problem statement and met with our client. Our meeting was productive as it helped to answer a few questions we had as well as to ensure that we are covering everything of importance within our project.

- **Any problems I encountered**
  Schedule conflicts for myself and the group made this week difficult. Our solution was less meeting in person and more work being done online.

Week 6 (Oct. 31 ˜ Nov. 4, 2016)

- **Plans for the coming week**
  Continue working on the requirements document. Hopefully finish by Friday. Send client our finished document.

- **Progress since last week**
  Met with group members and decided what sections we are each responsible for writing within the requirements document. Started writing the my sections of the document.

- **Any problems I encountered**
  Some aspects of the project are quite complex and will require extra time to research.

Week 7 (Nov. 7 ˜ Nov. 11, 2016)

- **Plans for the coming week**
  First, I need to decide what to include in the tech review. Next, the group needs to choose who will be responsible for each item. Lastly, start working on the tech review and finish it by Friday.

- **Progress since last week**
  We finished the requirements document and sent it off to our client.

- **Any problems I encountered**
  Last week was busy with midterms and other assignments. Both group members and myself struggled to find time to meet and finish the requirements document.

Week 8 (Nov. 14 ˜ Nov. 18, 2016)

- **Plans for the coming week**
  Although we submitted the tech review I want to look into our project for other options we might need to decide on later. Looking for additional items will carry over into starting work on the design document. Planning to meet with group so we can decide what sections everyone will be responsible for.

- **Progress since last week**
  Finished the tech review. Researched filter techniques for sensor data. Learned about advancements in MEMS quality assurance testing.

- **Any problems I encountered**
  We originally choose 9 items for the tech review however, one of the items was too straight forward to find alternative solutions. We had a hard time finding an additional item to include so we left the document with only 8. I found that the tech review took more time than expected to complete as the research I had to do was quite complicated.

3) Krisna Irawan (Fall 2016):

Week 3 (Oct. 10 ˜ 14, 2016)

- **Plans for the coming week**
  I am planning to do more research on Quaternion, since we will be working on the data in terms of Quaternion rotation. I also want to finalize our problem statement and have a clear understanding of our project. Lastly, I want to start exploring the possibilities of solution that we comes up with. I tried to learn more about the correlation of the acceleration between two data and the error that the integration gives.

- **Progress since last week**
  My team and I work together on the problem statement this week. We also be able to get more information of this project from our clients and have a greater understanding of this project. We also have meetings that really challenge

us to think more deeply about this project and make sure our team are on the same page.

- **Any problems I encountered**
  Although we have a better understanding than last week, it seems that we are still missing some of the points about this project. I am really grateful with the communication that our clients give to us, it is really help us to get a better understanding about this project. We also have difficulties in finding the perfect meeting time for our group. I am still not aware of my teammates schedule. However, we are successfully held our meeting this week and will improve on the schedule communication.

Week 4 (Oct. 17 ˜ 21, 2016)

- **Plans for the coming week**
  I am planning to finalize our problem statement and get our final problem statement signed by our clients. I will also be working on the requirement documents and see if we have any more question about this project.

- **Progress since last week**
  My team and I work together on clearing the communication breakdown that happens this week between our team and our clients. I now have a clearer understanding on how to deal with a real work environment.

- **Any problems I encountered**
  We have a breakdown in our communication with our clients. We are trying to resolve this problem and got some tips from our teacher regarding this issues. This project going to be a learning curve for me, I have to learn more about some terminology and knowledge about hardware.

Week 5 (Oct. 24 ˜ 28, 2016)

- **Plans for the coming week**
  I am planning to further refine our requirement documents and ask some clarification question to our clients (if any). We are also planning on getting our requirement document to be signed before the end of next week.

- **Progress since last week**
  We have another meetings with our clients this week on Wednesday. We clarify some stuff to move forward for our requirement documents. We also get our problem statement signed by our clients.

- **Any problems I encountered**
  I have a time management problem when working on the requirement documents this week. My schedule for this week is packed with assignment and midterms. I haven't got an optimal time to do the requirement documents this week. However, I will refine our requirement documents during the weekend and hopefully can get feedback from our teacher before we send it to our clients.

Week 6 (Oct. 31 ˜ Nov. 4, 2016)

- **Plans for the coming week**
  We will send our requirement documents to our client at the end of the week and see if we need to further refine our requirement document to be signed. I will also start to work on the Technical Review documents, looking for more options that we can do (or can't do) for this project.

- **Progress since last week**
  We work on the Requirement Document. Our clients has found out the ideal hardware that we will be working with for this project. Creating the requirement document makes me think more deeply about this project and how we going to achieve our goal. I got a clearer understanding on how to implements our project.

- **Any problems I encountered**
  The biggest problem for this week is time management. This week is a midterm week for all of us. This makes it hard

for us to focus on the documents.

Week 7 (Nov. 7 ~ Nov. 11, 2016)

- **Plans for the coming week**
  I will start investing my time in the Tech Review documents. I will ask more clarification question to the teacher about this documents. I will push myself in getting started with the design documents.

- **Progress since last week**
  We got a really good review for our requirement documents. Our clients are really pleased with the requirement documents that we send to them. I am glad that things works out and we pleased our clients with our work.

- **Any problems I encountered**
  The biggest problem that we faced is finding time to meet together and spend our time working together on the documents. We also have some question about our project during the weeks but our clients clarify those stuff and really help us to get the information that we need.

Week 8 (Nov. 14 ~ Nov. 18, 2016)

- **Plans for the coming week**
  We are trying our best to finish our Design Documents before the thanks giving break.

- **Progress since last week**
  We already submitted our signed requirement documents on Monday. We have finished our Tech review documents on Wednesday noon.

- **Any problems I encountered**
  Working on the Tech Review documents makes me think more deeply about the project and how we actually going to build this project. This requires me to do a lot of research and make a design decision.

Week 9 (Nov. 21 ~ Nov. 25, 2016)

- **Plans for the coming week**
  We will be working on the Design documents and finished it before Wednesday. We will be working on the Progress report after we submit the design documents to the clients.

- **Progress since last week**
  We get the foundation for the design documents ready. We have a better idea about the progress report.

- **Any problems I encountered**
  It was hard to find a time to work on the document during thanks giving break.

Week 10 (Nov. 28 ~ Dec. 2, 2016)

- **Plans for the coming week**
  Finished up the progress report document and video.

- **Progress since last week**
  We get the design document submitted.

- **Any problems I encountered**
  Busy schedule for dead week.

B. Winter 2016

1) Jiongcheng Luo (Winter 2016):

Week 1 (Jan. 9 ˜ 13, 2017)

- **Plans for the coming week**
  Getting all the basic hardware components and devices, start working on hardware setup including board hoop-up and sample code testing. Also set up a meeting with our clients to talk about plans and changes of the project progress.

- **Progress since last week**
  We held the first group meeting of the term and we decided to purchase our own board if the hardware not arriving yet.

- **Any problems I encountered**
  Delayed on getting the necessary hardware components and we couldn't start the implementation.

Week 2 (Jan. 16 ˜ 20, 2017)

- **Plans for the coming week**
  Getting all the basic hardware components and devices ASAP, start working on hardware setup including board hoop-up and sample code testing, modify the project schedule that due that to the delayed hardware devices

- **Progress since last week**
  We held the first meeting with the client and talked about our plan and blocks currently.

- **Any problems I encountered**
  Still have not yet gotten the necessary hardware components and we couldn't start the implementation.

2) Drew Hamm (Winter 2016): Miss.

3) Krisna Irawan (Winter 2016):

Week 1 (Jan. 9 ~ 13, 2017)

- **Plans for the coming week**
  We will tried to get all the necessary hardware to start this project. We will have a meeting with our clients on Wednesday next week.

- **Progress since last week**
  We meet as a group this week. We have a solid plan for next week.

- **Any problems I encountered**
  Getting started with all the logistic and the scheduling of the class and the project. Trying to get all the necessary hardware for this project.

Week 2 (Jan. 16 ~ 20, 2017)

- **Plans for the coming week**
  We will tried to get all the necessary hardware to start this project. We are trying our best to meet with our teacher to get the hardware.

- **Progress since last week**
  We meet as a group with our clients last Wednesday. We let them know about our logistic problem.

- **Any problems I encountered**
  Getting started with all the logistic and the scheduling of the class and the project. Trying to get all the necessary hardware for this project.

Week 3 (Jan. 23 ~ 27, 2017)

- **Plans for the coming week**
  We will start on the implementation of the project. We will finished up the hardware set up and a brief user interface.

- **Progress since last week**
  We finally get our hardware last Thursday. We can now really start on the implementation of the project.

- **Any problems I encountered**
  The logistic problem really set us back for a couple week behind the schedule. We will work hard to catch up with that.

Week 4 (Jan. 30 ~ Feb. 3, 2017)

- **Plans for the coming week**
  Continue to develop the user interface.

- **Progress since last week**
  We have a brief user interface.

- **Any problems I encountered**
  Getting the hardware setup.

Week 5 (Feb. 6 ˜ Feb. 10, 2017)

- **Plans for the coming week**
  Finished up the midterm progress report and document revisions.

- **Progress since last week**
  Our user interface is ready and finished.

- **Any problems I encountered**
  There is no generic user interface plug-in that I can use. I have to create the user interface from scratch.

Week 5 (Feb. 6 ˜ Feb. 10, 2017)

- **Plans for the coming week**
  Continue with the User Interface first user study.

- **Progress since last week**
  We finished the midterm progress report and revision for our documents.

- **Any problems I encountered**
  Finishing up the documents and video took most of our time this week.

Week 6 (Feb. 13 ˜ Feb. 17, 2017)

- **Plans for the coming week**
  Continue with the User Interface first user study.

- **Progress since last week**
  We finished the midterm progress report and revision for our documents.

- **Any problems I encountered**
  Finishing up the documents and video took most of our time this week.

Week 7 (Feb. 13 ˜ Feb. 17, 2017)

- **Plans for the coming week**
  Start to work on the statistical analysis portion of this project.

- **Progress since last week**
  We finished the midterm progress report and revision for our documents.

- **Any problems I encountered**
  We do not have a chance to meet this week.

Week 8 (Feb. 27 ˜ Mar. 3, 2017)

- **Plans for the coming week**
  Start to work on the statistical analysis portion of this project.

- **Progress since last week**
  Continue to work on the statistical analysis portion of this project.

- **Any problems I encountered**
  We do not know the structure of this class. We are not sure where we at in the class in terms of grades.

Week 9 (Mar. 3 ˜ Mar. 10, 2017)

- **Plans for the coming week**
  Work on the poster.

- **Progress since last week**
  We have a class meeting this week. We were practicing for the expo

- **Any problems I encountered**
  We have to finished up the project soon.

Week 10 (Mar. 13 ˜ Mar. 17, 2017)

- **Plans for the coming week**
  Finished up the progress report. Connect the user interface and statistical analysis to the Arduino module

- **Progress since last week**
  We finished the poster

- **Any problems I encountered**
  We have to finished up the progress report during finals week.

C. Spring 2016

1) Jiongcheng Luo (Spring 2016):

Week 1 (Apr. 3 ˜ 7, 2017)

- **Plans for the coming week**
  As the first week after break, I plan to have a meeting with the entire group as well as our clients, we will plan to give a short report to them and we plan to move on with our project

- **Progress since last week**
  I briefly took a look at the new schedule of the spring term and start making plan based on our current situation.

- **Any problems I encountered**
  We still found hard to retrieve two sets of data from the two different IMUs, we considered our old GUI program is NOT clearly enough to for our demonstration purpose, we planned to develop a new GUI program.

Week 2 (Apr. 10 ˜ 14, 2017)

- **Plans for the coming week**
  By the coming week, we plan to update and fix our poster as the second draft. We also plan to group up to work on the project together, specifically will focus on the design of the UI of the demonstration system.

- **Progress since last week**
  We held our first meeting since the spring break and we planned out our schedule for the rest of the term.

- **Any problems I encountered**
  We found hard to fix the poster since everything is on a very level terminology, and since we have not done the statical analysis for our final testing result, we won't be able to put a specific result onto the poster.

Week 3 (Apr. 17 ˜ 21, 2017)

- **Plans for the coming week**
  We plan to fix our poster for the final draft and have it send to our client for verification. We also plan to meet up again to continue woking on the project together.

- **Progress since last week**
  We got some feedback from the TA in terms of how to fix and update for our poster. We finally found a Python library called VPython that is easy to implement, and it provides ideal graphical interface (3D model animation) for our project virtual demonstration, we have also figured out the hardware configuration in terms of getting data from two IMUs as different addresses.

- **Any problems I encountered**
  We could not illustrate the best graphical display model for showing off to the audience in the expo, we want it to be straight forward and understandable, but also be able to demonstrate our result and outcome of the complex system.

Week 4 (Apr. 24 ˜ 28, 2017)

- **Plans for the coming week**
  We will try to finish up everything for the project and have a well plan for the demonstration in the expo. We had grouped up again and made some progress on the project.

- **Progress since last week**
  We fixed the refined our poster, we submitted the final draft of poster to our client, everything looks good to them.

- **Any problems I encountered**
  We could not show our result to our client as they expect, we are still trying to improve the accuracy and stability of our output, and we will figure out the best way to show off the detail of the alignment.

Week 5 (May. 1 ˜ 5, 2017)

- **Plans for the coming week**
  We will try to finish up putting the Arduino program with the demonstration UI program together and test for expo demonstration, we will make a detailed plan for the demonstration in the expo, also plan to show our result to our client see if we can improve based on what we have.

- **Progress since last week**
  We finally submitted our poster for printing. We roughly finished all implementation on the software part of the project, and we updated our github repo with the latest version of our project, and we also updated the README file in terms of our project description.

- **Any problems I encountered**
  We could not improve the accuracy for the final output result, we may need to ask for some suggestions from our client, we also have not not yet test for the performance of the GUI, we may need to spend some time to debug.

Week 6 (May. 8 ˜ 12, 2017)

- **Plans for the coming week**
  The Friday of coming week will be the engineering EXPO day, so we plan to finalize our project for the expo demonstration, and we also plan to build our demonstration system.

- **Progress since last week**
  We have finalized our GUI and we found out that our dynamic alignment process makes a not bad result.

- **Any problems I encountered**
  We found out a our result is not accurate enough especially and we believe that is because of something goes wrong with yaw calculation or implementation, but we could not determine if the problem comes from the hardware or our algorithm.

Week 7(May. 15 ˜ 19, 2017)

- **Plans for the coming week**
  We plan to implement our physical test on our current system and we also plan to set up a meeting with our client to report our result.

- **Progress since last week**
  We finally held our EXPO last week, although we could not complete our project in hundred percent, we did demonstrated our project and we did attracted a number of visitors to our expo. And even though our project seems to be hard to understand for general audience, some people were still quite interested in knowing about our project.

- **Any problems I encountered**
  We could not demonstrate our system with the static alignment algorithm since something goes wrong within the algorithm. Another problem I encountered was that I could not explain my project clearly to the audiences, I could not explain well to those who do not have any background knowledge about our project.

Week 8(May. 22 ˜ 26, 2017)

- **Plans for the coming week**
  We plan to do our final physical test to verify our result and also plan to set up a meeting with our client to report our final progress.

- **Progress since last week**
  We discussed and designed for the physical test.

- **Any problems I encountered**
  We did not set up the meeting with our client yet since we have not finished the physical test.

- **If you were to redo the project from Fall term, what would you tell yourself?**
  You should start working on the project as soon as possible discussed with the client mote often.

- **What is the biggest skill you've learned?**
  The biggest skill I learned is how to get into a field that I am unfamiliar with before or even not having any experience, my project is kind of like that, which I did not know anything about my project or any background knowledge before I really worked on it, however, throughout this project, I learned how to get start with a project like this.

- **What skills do you see yourself using in the future?**
  I think I will use the skill of dealing with projects involve in both hardware and software.

- **What did you like about the project, and what did you not?**
  I like that my project was not purely dealing with software or programming, but involve much with hardware, physics and mathematics; One thing I do not like about my project is that we did not have much resources or equipments to test with, we got only some conceptual ideas and thoughts and we did not have the chance to work with the real HUD (equipment).

- **What did you learn from your teammates?**
  From Drew, I learned that we should not rely on other external examples or existed libraries too much during implementing our project, we should design our own one based on other resources.

- **If you were the client for this project, would you be satisfied with the work done?**
  I probably will not be 100% satisfied with the current work done since we did not have a result as expected, however I can understand with that since we had limited time, knowledge and resources for our project implementation.

- **If your project were to be continued next year, what do you think needs to be working on?**
  First we can test with varied models of IMUs and select the one that has the best performance and is the easiest to deal with, then we will try to use more IMUs to work together for achieving better performance, and the most important thing that I consider need to be worked on is to design a complete test procedures for in order to precisely test the output of the system.

2) Drew Hamm (Spring 2016):

Week 1 (Apr. 3 ˜ 7, 2017)

- **Plans for the coming week**
  Meet with our group and work on the micro controller code as well as the GUI.

- **Progress since last week**
- **Any problems I encountered**
  I've been looking for a new place to live and finally moved this week so so my time has been limited.

Week 2 (Apr. 10 ˜ 14, 2017)

- **Plans for the coming week**
  I am planning to work on the poster for expo as well as further improving our code.

- **Progress since last week**
  I met with our group over the weekend to discuss the project and make plans for the term.

- **Any problems I encountered**
  We are using a I2C bus to facilitate communication between devices in our project. However, their is a limit to the length in which the I2C bus may be before negatively effecting its performance. This may be a problem given the realized interconnectivity within an aircraft.

Week 3 (Apr. 17 ˜ 21, 2017)

- **Plans for the coming week**
  I continue making improvements to our code in terms of refactoring. I will work on improving the calibration and static alignment processes.

- **Progress since last week**
  I made changes to our poster to improve readability as well as adding the necessary content required by my part. I started refactoring our code to improve readability as well as to promote further development.

- **Any problems I encountered**
  The magnetometer data seems to be skewed. We are having trouble deciding on the data and format to be sent to the GUI.

Week 4 (Apr. 24 ˜ 28, 2017)

- **Plans for the coming week**
  Improve the calibration process and submit the final version of our project in terms of what will be graded. I need to submit my release form for expo.

- **Progress since last week**
  I met with our group to continue work on our solution and test it with the GUI. I made changes to our poster to improve grammar and adjust positioning of elements. We finished and submitted our poster

- **Any problems I encountered**
  Inconsistent calibration results and limited micro controller memory. Started using the Arduino IDE serial plotter for analyzing individual sensor data which has been insightful. We also need to perform physical tests on our system but we are having a hard time creating a plan of attack.

Week 5 (May. 1 ˜ 5, 2017)

- **Plans for the coming week**
  Research methods for improving data readings and sample selection. Work on the midterm progress report. We will be meeting with another team to review each others posters and give feedback.

- **Progress since last week**
  I was able to push a working version of our alignment solution. Submitted my release form for expo.

- **Any problems I encountered**
  When taking the offset using quaternions I found that the result was inaccurate while the quaternion was converging as a result of the filter algorithm. I tried to remove samples taken during the convergence period by analyzing the change of values over time. More work needs to be done to improve this process. Another problem I faced had to do with the error of individual samples. Even when the device was stationary, I noticed values spiking at times. I attempted to remove these samples by using a hard coded threshold that was determined heuristically. I'm not sure what the best approach would be at this current time.

Week 6 (May. 8 ˜ 12, 2017)

- **Plans for the coming week**
  We will meeting up to make some last minute changes with our project in terms of what will be displayed during expo.

- **Progress since last week**
  We have been having problems with the magnetometer data and poor readings. I'm not sure if I miss configured the device or am not performing sufficient calibration. Since expo is next week, we will explore a 6-axis filter instead of the 9-axis filter.

- **Any problems I encountered**
  Worked with Krisna to add the statistical confidence interval when finding the static offset. Our group met with another group to review each others posters and provide feedback. We started working on our midterm progress report.

Week 7(May. 15 ˜ 19, 2017)

- **Plans for the coming week**
  We will try to make plans to test our system.

- **Progress since last week**
  I finished my part of the midterm progress report. We met up to discuss our project to ensure we were all on the same page for expo. We built a simple demonstration system that could be manipulated during expo to help explain our project. EXPO!

- **Any problems I encountered**
  My audio input is not working so I had to meet up with Roger and use his laptop to record my presentation. Our system has yet to undergo the physical tests that are required before sending our results to RC. Since our system has yet to be tested, I realized late that our solution was not correctly checking for static offset. In order to make our solution work, we would have to move the HUD sensor from the IRU to the HUD and then take the difference. Our error was a result of testing our system on a with an aligned axis. What we were really detection was simple the sensor bias. Our false interpretation was reaffirmed because we were successfully taking the dynamic offset.

Week 8(May. 22 ˜ 26, 2017)

- **Plans for the coming week**
  I am planning on meeting with our group and getting in contact with our client. I'm also planning on working on our final documents.

- **Progress since last week**
  Started working on our final documents.

- **Any problems I encountered**
  We were not able to meet this week so testing the system and contacting our client delayed.

- **If you were to redo the project from Fall term, what would you tell yourself?**
  First of all, I would tell myself to setup version control early and make use of the skills learned in my software development classes to promote collaboration and a sense of direction while building momentum as milestones were reached. Next, I would tell myself to utilize the availability of our clients by asking more questions when referencing the technical challenges of our project. Lastly, I would stress testing early; both physically and programmatically.

- **What is the biggest skill you've learned?**
  I think biggest skill I have learned would have to be the ability to work with sensors such as an accelerometer, gyroscope and magnetometer. I choose this because there was a lot involved. First, I had to learn how to configure devices by reading through the devices register mapping documentation. Next, I had to learn how to calibrate the devices by taking initial readings during setup, comparing them against factory defaults and storing the results in the appropriate registers or using that information to modify future readings. I also had to learn how communication between devices worked and why and how some devices should be configure as slaves or masters.

- **What skills do you see yourself using in the future?**
  Considering the shift towards the Internet of things, I can see myself working with multiple devices while facilitating the necessary communication for those devices in future work or even hobby projects.

- **What did you like about the project, and what did you not?**
  I liked that I was given an opportunity to work in an area I was not familiar with. I did not like how much work was required to catch up in understanding of particular topics when I would have rather been making significant progress towards a solution.

- **What did you learn from your teammates?**
  I learned additional interpersonal skills and strategies to overcome poor first impressions. My team also helped me to learn skills relating to latex and vpython.

- **If you were the client for this project, would you be satisfied with the work done?**
  I would have liked to see a better solution but I believe the documented requirements have been met.

- **If your project were to be continued next year, what do you think needs to be working on?**
  Memory considerations when analyzing sensor output. Improve calibration of sensors then test against documented hardware error tolerance specifications. Magnetometer considerations. Physical tests against filtered quaternion output and the resulting offset with static values. Look into physically testing dynamic values. Research methods to handle drift such as resetting to a known reference then finding ways to physically test against under expected conditions.

3) Krisna Irawan (Spring 2016):

Week 1 (Apr. 3 ~ 7, 2017)

- **Plans for the coming week**
  We are planning to set up a meeting with our clients. We are planning to work together on our project this weekend. Hopefully we can finished up our project before week 3.

- **Progress since last week**
  We know the schedule for the class this term. We know our plan for this term.

- **Any problems I encountered**
  To connect the statistical analysis portion, I need the data from the offset algorithm. Our team still working on the offset algorithm.

Week 2 (Apr. 10 ~ 14, 2017)

- **Plans for the coming week**
  We are planning to finished up the poster. We also planning to meet up with other groups to do the extra credit for this class.

- **Progress since last week**
  We meet with the TA and talk about our current state and where we are heading next.

- **Any problems I encountered**
  We are not able to do much work this week.

Week 3 (Apr. 17 ~ 21, 2017)

- **Plans for the coming week**
  We are planning to finished up the project this week. We are hoping to get the poster signed by our clients this week.

- **Progress since last week**
  We the poster feedback from our TA, we will incorporate that feedback to our poster. We are almost done with the project. We take care of the algorithm and reading from 2 sensors.

- **Any problems I encountered**
  We found a language called vpython that will help us to create the 3d simulation of the HUD. We are still learning about this language.

Week 4 (Apr. 24 ~ 28, 2017)

- **Plans for the coming week**
  Set up an interview for the WIRED style review. Finished up the WIRED style review.

- **Progress since last week**
  I finished making the 3d simulation and convert the statistical analysis portion to python.

- **Any problems I encountered**
  We have to wrap up our project before the code freeze on May 1st. We will be working hard this weekend.

Week 5 (May. 1 ~ 5, 2017)

- **Plans for the coming week**
  Start to work on the midterm progress report. Finishing the hardware user interface setup.

- **Progress since last week**
  I finished the WIRED style review assignment.

- **Any problems I encountered**
  We have a busy schedule this week, we will work on the project on the weekend.

Week 6 (May. 8 ˜ 12, 2017)

- **Plans for the coming week**
  Do our best in the expo. Finished the midterm progress report.

- **Progress since last week**
  We do the extra credit assignment. We are almost done with the midterm progress report.

- **Any problems I encountered**
  We need some time to debug our code, especially, the Yaw value for the calculation.

Week 7(May. 15 ˜ 19, 2017)

- **Plans for the coming week**
  Start to work on the final documentation of the project

- **Progress since last week**
  We successfully finished the expo!

- **Any problems I encountered**
  We work hard to get our project done for the expo. We finally get it done!

Week 8(May. 22 ˜ 26, 2017)

- **Plans for the coming week**
  Start to work on the final documentation of the project. I have to work on the three small writing assignment that our instructor gave to us.

- **Progress since last week**
  We went to the final class this week. We knew what we have to do for the rest of the term.

- **Any problems I encountered**
  We did not have a chance to meet this week except during class time.

- **If you were to redo the project from Fall term, what would you tell yourself?**
  Start early, get closer to the client before hand.

- **What is the biggest skill you've learned?**
  I learn about Arduino IDE and how sensors work.

- **What skills do you see yourself using in the future?**
  Working with IMU.

- **What did you like about the project, and what did you not?**
  We have an amazing clients that really supportive. I like to visualize the plane movement on the user interface. I have

a hard time learning about quaternion.

- **What did you learn from your teammates?**
  I learn a lot about IMU and Arduino IDE from my teammates.

- **If you were the client for this project, would you be satisfied with the work done?**
  Honestly, I think there is some part that we can improved on. Overall, I think we meet the requirement that our clients gave.

- **If your project were to be continued next year, what do you think needs to be working on?**
  Solve the drifting yaw angle. Also, improve on the accuracy of the algorithm and sensors.

Final Poster

HEAD-UP DISPLAY AUTO-ALIGNMENT SYSTEM

The cost of an airplane may be reduced in the future!

WHAT IS A HEAD-UP DISPLAY (HUD)?

A transparent display that is installed in a number of aircraft today. Located in the pilot's forward field of view, a HUD presents flight information by using graphical, numerical and symbolical data. This device eliminates the need for pilots to continually transition from head-down instruments to head-up, out-the-window view during critical phases of flight.

Pilot's forward field of view

CURRENT HUD ALIGNMENT SYSTEM

Currently, a HUD obtains data from an aircraft's Inertial Measurement Unit (IMU). To display precise and accurate information, the HUD must be carefully aligned to the IRU during installation. However, the current HUD installation process requires specialized equipment and epoxy which is time consuming, costly, and interrupts production line progress for the manufacturer.

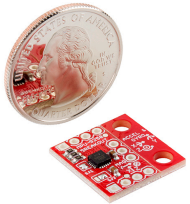
Head-Up Display during flight

NEW ALIGNMENT METHODOLOGY

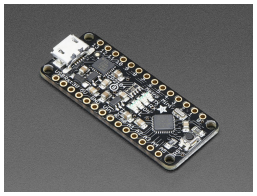
The goal of this project is to improve the current HUD alignment systems by reducing the installation cost and time required to precisely align flight information to the HUD. Our client, Rockwell Collins, seeks a new alignment methodology that utilizes HUD mounted inertial measurement units.

WHAT IS AN INERTIAL MEASUREMENT UNIT (IMU)?

An IMU contains a set of sensors that typically include an accelerometer, gyroscopes and/or magnetometer (compass). In this project, we used an IMU of model MPU-9250 from *Sparkfun Electronics*. *An MPU-9250 IMU* is as tiny as a quarter, yet it contains all three of the sensors mentioned above. A diverse set of sensors increases the degrees of freedom which improves data accuracy. In addition, in order to retrieve sensor data, we also used a microcontroller of model Metro-Mini 328 from *Adafruit Industry* to communicate with the IMU.



IMU - *Sparkfun MPU-9250*

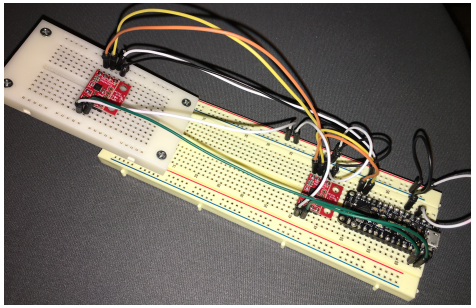


Microcontroller - *Metro-Mini 328*

PROJECT OVERVIEW

Our project serves as a proof of concept for a new alignment methodology. This project is separated into three parts.

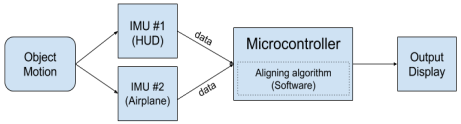
- The configuration of hardware and communication between devices.
- The development of an program that can find the HUD's alignment offset in relation to the aircraft.
- The design and implementation of a demonstration interface that visually presents the alignment results.



Microcontroller wired to two IMUs

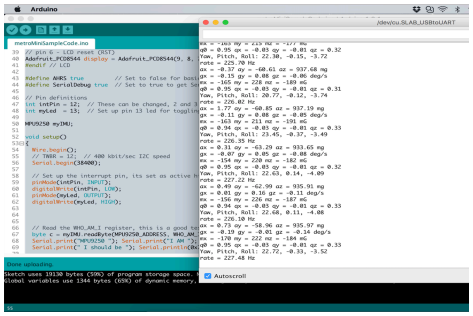
HOW DID WE SOLVE THE PROBLEM?

Our demonstration system made use of two separate IMUs to generate the data that represents the individual points of origin for both the HUD and aircraft. Along with the IMUs, we used a microcontroller to process data and a PC to display the results.



Data flow between hardware components

By using a communication protocol we were able to access each IMU from the microcontroller. We used the Arduino development environment to program the microcontroller as well to debug data output. Data then was retrieved from each sensor and used to determine the alignment offset through statistical analysis. The results were then sent to the PC to be displayed.

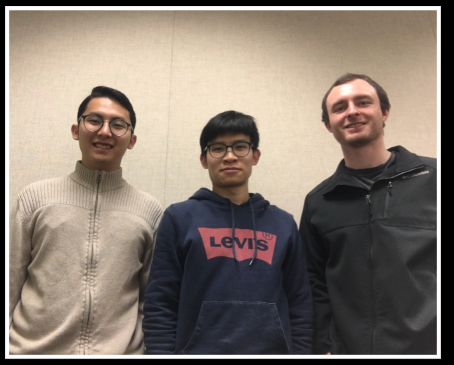


Reading IMU data in Arduino Development Environment

CALIBRATION PROCESS & RESULT

Our project required a two part calibration process. First, each IMU was configured to output its individual sensor data. During IMU configuration, an initial calibration was performed to discover each sensors individual bias. Once the biases were known, we removed them from future readings. Next, the alignment offset discovery required a calibration process to produce a result within an acceptable range. We found the offset between two IMUs (airplane and HUD) after taking a number of samples from each device, the difference between each pair of samples and the average of those differences. Multiple data readings were required in each part of calibration to remove inaccuracies.

Meet the Engineers



Group Members - *Group #65*

- Krisna Irawan - irawank@oregonstate.edu
- Jiongcheng Luo - luojio@oregonstate.edu
- Drew Hamm - hammd@oregonstate.edu

Sponsors - *Rockwell Collins, Inc.*

- Brandon Wilson
- Weston Lahr

Rockwell Collins

HYPOTHETICAL OUTCOME

Although we were not able to test the system using a real HUD on an aircraft, our project demonstrates how an inexpensive IMU can be used to accurately find an offset in alignment. This new alignment methodology points to a departure from the current HUD installation process by utilizing HUD mounted IMUs. Overall, this system proves that there is an opportunity to substantially cut HUD installation costs in the future.

Oregon State UNIVERSITY

VIII. Project Documentation

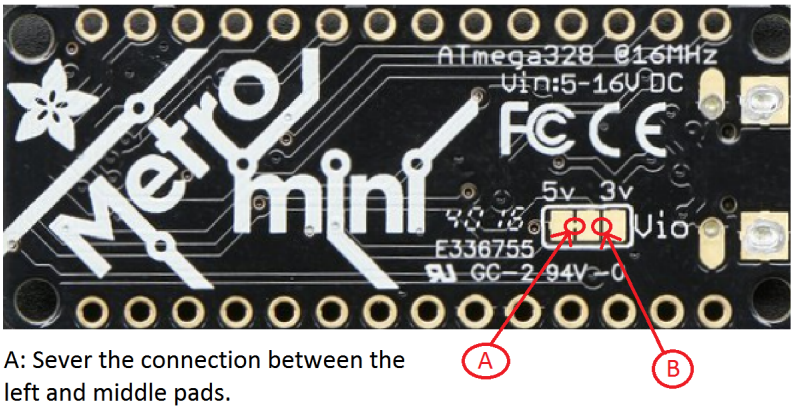
Below you will find everything that is required in order to run our project. First, you will find a list of all the materials used in our project. Next, you will find a list of steps and images describing the process to setup the hardware along with a block diagram explaining the connections between devices. The following two sections will list our projects dependencies along with instructions to program the microcontroller. Lastly, a step has been included for running our project and verifying the results.

Required Hardware

- 2x SparkFun IMU Breakout - MPU-9250
- Adafruit Metro Mini 328 -5V 16MHz
- PC /w display
- USB 2.0 cable - Type-A Male to Mini Type-B Male
- 2x breadboard
- Wires to connect MPUs and Microcontroller
- Solder
- Soldering Iron
- Male Headers to connect devices to breadboard

Hardware setup

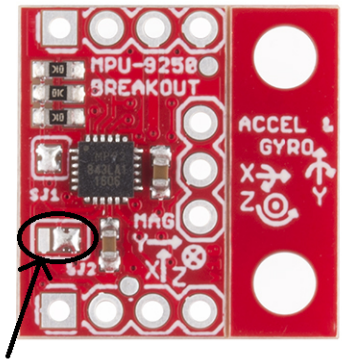
- Modify Microcontroller to output 3.3v on its digital logic pins



A: Sever the connection between the left and middle pads.

B: Solder a new connection between the right and middle pads.

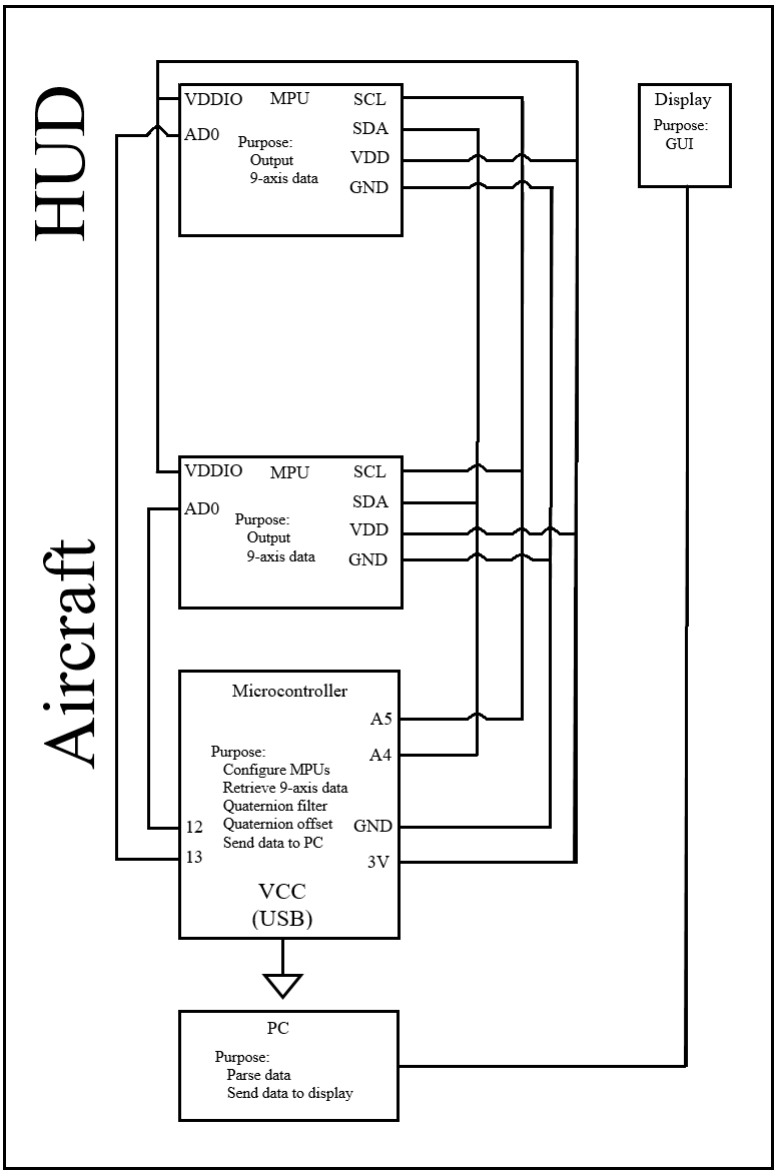
- Modify IMUs to allow their AD0 pins to receive voltage which enables dynamic I2C addresses



A: Sever the connection between the right and middle pads.

B: Solder a new connection between the left and middle pads.

- Solder male headers to the bottom of the microcontroller
- Solder male headers to the bottom of both IMUs
- Attach the microcontroller and one IMU to the first breadboard
- Attach the second IMU to the second breadboard
- Connect devices and pc



Setup Dependencies

- Install the Arduino IDE
- Install Python 2.7.9
- Install VPython
- Install pySerial
- Download our arduino project
- Download our python script

Program Hardware

- Ensure that the microcontroller is connected to the PC
- Load our arduino project on the Arduino IDE
- Program the microcontroller with our project through the Arduino IDE

Run application

- Load our python script
- Run the script and follow instructions
- Verify results by viewing the GUI

PSEUDO CODE

```
// Place HUD in alignment with aircraft IRU
wait()

// Setup:
for each MPU-9250:
        update AD0 pin  // Do this in order to change I2C address
        initialize/configure registers
        for each sensor: // Accelerometer, Gyroscope, and magnetometer
                initialize/configure registers
calibrate device

// Install HUD
wait()

// Static Offset Mode
while statistical confidence insufficient:
        while sample size insufficient:
                for each MPU-9250:
                        read data
                        update stored quaternion value via quaternion filter
                record quaternion offset sample
        perform confidence interval

send static offset via serial port

// Dynamic offset mode
while system is active:
for each MPU-9250:
        read data
        update stored quaternion value via quaternion filter
record quaternion offset

send dynamic offset via serial port
```

IX. How did we learn new Technology

X. What did we learn from all this?

A. Jiongcheng Luo

- **What technical information did you learn?**
  Throughout this course, a new technology I learned is Latex formatting. Throughout the project, I learned a lot about manufacturing an aircraft, especially its Head-up display system and Inertial Measurement Unit. I learned how an IMUs works and how we could apply it into a useful product. Specifically, from the perspective of software, I learned how to program Arduino and using VPython library.

- **What non-technical information did you learn?**
  I learned how to collaborate with a team from everyone did not know each other to everyone made a project together, specifically are the skills of communication, cooperation with co-workers. In addition, we had change to contact with a real company client, I learned a lot how a real industry deal with new technologies. Throughout working on the project, I learned how to compare and choose the tools or technologies to be used for our project.

- **What have you learned about project work?**
  I learned that before doing a project, a team should have well discussion and make sure everyone understands what this project is doing. After that, a detailed, comprehensive plan needs to be made, during design procedures, all documents need to keep track of any changes and new ideas. During software development, using agile methodology is preferred, so that every part of the software can be modified easily.

- **What have you learned about project management?**
  A project first should be designed comprehensively including all details, graphs, diagrams, examples, documents etc. And during the implementation, its necessary to use version control for all software programs especially for a program that with complicated structure. Everyone can work on a different version for implementing different purpose or portion of the project, a completed version should a merge of all completed parts by different version and the merge needs to be agreed by all team members.

- **What have you learned about working in teams?**
  I learned that everyone in a team must has unique background knowledge, interests and everyone has different focus field, therefore, in order to get our project done successfully, we need to know each other well, including the lack and expertise of everyone, so that everyone can work on what they good at and everyone can complement for each other.

- **If you could do it all over, what would you do differently?**
  First, I would get started working on the project earlier, we had spent too much time on the design procedures and researching, then we found that we did not have enough time to accomplish the plans since more problems came up with we moving forward. Second, I would use version control system to keep track of our program, since we had met a problem that we did not save the previous version and we overwrote it, and as a result, we lose the version that was working before, so that we had to redo it all over again. Third, I would meet with our clients more often in regards to technical discussion, we had a lot of issues and problems that we did not know how to solve at all, and we had spent too much time in researching and attempting, that leads to that we could not complete some parts of the project at the end. Fourth, I would try using more IMUs for testing purpose, even though our result showed that working, it did not meet our initial expectation, more IMUs would more accurate.

B. Drew Hamm

- **What technical information did you learn?**
  This project ended up touching on quite a few topics that were previously unfamiliar to me. First, I had to learn about Tait-Bryan angles and how the different conventions worked such as for representing and aircrafts heading, elevation, and bank. Next, I had to learn about quaternions and several different filter algorithms, Madgwick and Mahony, for converting our 9-axis data into a usable form. Since our work focused primarily hardware, I had to learn how to configure and communicate with our devices; which required extensive reading into register mapping and other documentation. Additionally, I learned how to work with accelerometers, gyroscopes, and magnetometers. Lastly, I learned improved technical writing skills along with Latex formatting.

- **What non-technical information did you learn?**
  Primarily, throughout this project I had to learn how to work effectively with a new team. Communication amongst the

team as well as with our client played heavily into our means to achieve a successful outcome. In order to complete tasks on time, especially after falling behind initially, improving my time management skills was crucial. Lastly, the skills required for expo such as communicating ideas from technical topics to diverse audiences.

- **What have you learned about project work?**
  Although I already knew of the importance that communication had in project work, I am now placing even greater emphasis on it for my later project work. I learned of the importance of creating tasks that can be developed independently. Creating tasks that can be effectively delegated amongst team members to play to everyone's strengths is something I would like to strive for in the future. Lastly, I learned how important a frequently updated timeline that includes milestones and well thought out tasks can be.

- **What have you learned about project management?**
  As for project management, I learned that it's important to keep a direct line of communication open between management and the team. Similarly to having a direct line of communication, it's important to maintain an updated timeline of the team's progression.

- **What have you learned about working in teams?**
  Since we made a poor first impression with our client, our team had to come together to overcome that initial problem. I learned how effective a united front could be to resolving issues.

- **If you could do it all over, what would you do differently?**
  First, I would like to setup version control early. Considering our project, there was some ambiguity as to whether we should develop in the open or not. Second, I would try to actively apply the skills I learned in my software development classes to promote collaboration and a sense of direction while building momentum as milestones were reached. Next, it would have been helpful to send more of my questions towards our client. By asking questions often, we could have improved our utilization of their technical experience to better guide my efforts. Unfortunately, we did not test our project soon enough and ended up going down a wrong path. If I had to do it all over, I would have tested early and often and considered following test driven development. Another change that would have greatly improved our results would have been to create an improved timeline that was updated frequently. An improved timeline would consist of tasks that could be developed independently. As with independent tasks, the timeline should consist of whom is responsible for which task; with proper thought put into each others strengths. Lastly, I would have tried to get access to not only the actual IRU that is used within an aircraft, but also the HUD that we were supposed to design our proof of concept solution for.

C. Krisna Irawan

XI. Appendix (Source Code)

A. *Note for source code*

- There are totoally 12 .cpp code files for microntroller prgoram, 1 .ino code file as the main program for running arduino and 1 Python code file for graphical user interface.
- ”MPU9250.cpp” and ”MPU9250.h” are libraries for the MPU9250 IMU breakout.
- ”quaternionFilters.cpp” and ”quaternionFilters.h” are online credited libraries we found for quaternion convertion calculation.
- ”quaternionFilters.cpp” and ”quaternionFilters.h” are online credited libraries we found for quaternion convertion calculation.

```
# gui.py

        #!/usr/bin/python
        /*
        ——      Oregon State University 2017 Capstone Project ——
        Author:         Jiongcheng Luo, Drew Hamm, Krisna Irawan
        Description:
                graphical display program using vpython (http://vpython.org/)
                and pyserial (https://pythonhosted.org/pyserial/) libraries
        */

        import serial
        import visual as vp
        import math
        import sys
        import time


        PORT = '/dev/tty.SLAB_USBtoUART'          # mac OS
        #PORT = 'COM4'                                          # windows


        BAUD_RATE = 38400
        DtoR = (math.pi/180)
        RtoD = (180/math.pi)

        SERIRAL_INPUT = serial.Serial(PORT, BAUD_RATE)


        def readData(string):
                newSerialInput = []
                for x in string.split(','):
                        newSerialInput.append(x.strip())
                return newSerialInput

        #-------------------------------------------------
        #                                   MAIN PROGRAM
        #-------------------------------------------------
        class IMU():
                def __init__(self,name):
                        self.name = name

                        self.prev_pitch = 0
                        self.prev_roll = 0
                        self.prev_yaw = 0

                        self.cur_pitch = 0
                        self.cur_roll = 0
                        self.cur_yaw = 0

                        self.pitch = 0
                        self.roll = 0
                        self.yaw = 0

                def reset(self,b):
                        if b == True:
                                self.pitch = 0
                                self.roll = 0
                                self.yaw = 0

        def reset(plane, yaw, pitch, roll, color):
                plane.visible = False
```

```
            del plane
            #time.sleep(0.05)
            plane = vp.frame()
            vp.ellipsoid(frame=plane, pos=(0,0,0), length=8, height=2.5, width=2.5, color=color,
                opacity=0.5)
            vp.pyramid(frame=plane, pos=(-1,0,0), size=(4,6,1), color=color, opacity=0.5)
            vp.pyramid(frame=plane, pos=(-3.5,0,1), size=(2,1,2), color=color, opacity=0.5)
            plane.rotate(angle=math.pi/2, axis=(-1,0,0), origin=(0,0,0))

            plane.rotate(angle=-math.pi/2, axis=(0,1,0))
            plane.rotate(angle=roll, axis=(0,0,1))
            plane.rotate(angle=pitch, axis=(1,0,0))
            return plane;

    def text_reset(text, data, x, y, z):
            text.visible = False
            del text
            data = data*RtoD
            text = vp.label(pos=(x,y,z), text=str(data), height=10, border=0, font='sans', color=vp.
                color.green, box=0)
            return text;

    def display():
            # scene and object initilaizing
            scene2 = vp.display(title='Airplane Simulation',
         x=0, y=0, width=800, height=600,
         center=(0,0,8), autoscale = 1, background=vp.color.black)

            #Text
            text1 = vp.label(pos=(-4,-8,0), text='Yaw', height=18, border=0, font='sans', color=vp.
                color.green, box=0)
            text1 = vp.label(pos=(0,-8,0), text='Pitch', height=18, border=0, font='sans', color=vp.
                color.green, box=0)
            text1 = vp.label(pos=(4,-8,0), text='Roll', height=18, border=0, font='sans', color=vp.
                color.green, box=0)
            text1 = vp.label(pos=(-8,-9,0), text='HUD IMU Data:', height=10, border=0, font='sans',
                color=vp.color.green, box=0)
            text1 = vp.label(pos=(-8,-10,0), text='Airplane IMU Data:', height=10, border=0, font='
                sans', color=vp.color.green, box=0)

            #Plane 1
            plane = vp.frame()
            vp.ellipsoid(frame=plane, pos=(0,0,0), length=8, height=2.5, width=2.5, color=vp.color.
                red, opacity=0.5)
            vp.pyramid(frame=plane, pos=(-1,0,0), size=(4,6,1), color=vp.color.red, opacity=0.5)
            vp.pyramid(frame=plane, pos=(-3.5,0,1), size=(2,1,2), color=vp.color.red, opacity=0.5)
            plane.rotate(angle=math.pi/2, axis=(-1,0,0), origin=(0,0,0))
            #Axis Rotation for debugging purpose
            plane.rotate(angle=math.pi/2, axis=(0,-1,0), origin=(0,0,0))
            # plane.rotate(angle=math.pi/2, axis=(0,0,-1), origin=(0,0,0))

            #Plane 2
            plane2 = vp.frame()
            vp.ellipsoid(frame=plane2, pos=(0,0,0), length=8, height=2.5, width=2.5, color=vp.color.
                green, opacity=0.5)
            vp.pyramid(frame=plane2, pos=(-1,0,0), size=(4,6,1), color=vp.color.green, opacity=0.5)
            vp.pyramid(frame=plane2, pos=(-3.5,0,1), size=(2,1,2), color=vp.color.green, opacity=0.5)
            plane2.rotate(angle=math.pi/2, axis=(-1,0,0), origin=(0,0,0))
            #Axis Rotation for debugging purpose
            plane2.rotate(angle=math.pi/2, axis=(0,-1,0), origin=(0,0,0))
            # plane2.rotate(angle=math.pi/2, axis=(0,0,-1), origin=(0,0,0))

            #Axis and Horizon Line
            Axis = vp.label(pos=(0,0,7.5), text='Y', height=10, border=0, font='sans', color=vp.color
                .green, box=0)
            Axis = vp.arrow(pos=(0,0,0), axis=(7,0,0), shaftwidth=0.1, color=vp.color.green, opacity
                =0.5)
            Axis = vp.label(pos=(7.5,0,0), text='X', height=10, border=0, font='sans', color=vp.color
                .green, box=0)
            Axis = vp.arrow(pos=(0,0,0), axis=(0,7,0), shaftwidth=0.1, color=vp.color.green, opacity
                =0.5)
            Axis = vp.label(pos=(0,7.5,0), text='Z', height=10, border=0, font='sans', color=vp.color
                .green, box=0)
            Axis = vp.arrow(pos=(0,0,0), axis=(0,0,7), shaftwidth=0.1, color=vp.color.green, opacity
                =0.5)
```

```
        #Print Original Data
        text2 = vp.label(pos=(-4,-9,0), text='Original Data', height=10, border=0, font='sans',
            color=vp.color.green, box=0)
        text3 = vp.label(pos=(0,-9,0), text='Original Data', height=10, border=0, font='sans',
            color=vp.color.green, box=0)
        text4 = vp.label(pos=(4,-9,0), text='Original Data', height=10, border=0, font='sans',
            color=vp.color.green, box=0)

        #Print Aligned Data
        text5 = vp.label(pos=(-4,-10,0), text='Alignment Error', height=10, border=0, font='sans'
            , color=vp.color.green, box=0)
        text6 = vp.label(pos=(0,-10,0), text='Alignment Error', height=10, border=0, font='sans',
             color=vp.color.green, box=0)
        text7 = vp.label(pos=(4,-10,0), text='Alignment Error', height=10, border=0, font='sans',
             color=vp.color.green, box=0)

#-----------------------------------------------
#                                    GUI PART
#-----------------------------------------------
        def sense_filter(cur,prev):     # adjust animation sensitivity
                __SENSE = 0.005         # higher _SENSE = higher sensitivity
                delta = cur - prev
                if math.fabs(delta) > __SENSE:
                        return delta
                return 0

        # enable which axis to run
        yawAxis = True
        pitchAxis = True
        rollAxis = True

        imu1 = IMU('1')
        imu2 = IMU('2')

        # init = 0
        # loop over for animation
        while True:
                # performed rate for animation
                vp.rate(50)
                while SERIRAL_INPUT.inWaiting() == 0:
                        pass

                serial_line = SERIRAL_INPUT.readline()
                c = readData(serial_line)
                if len(c) == 6: # filter out none numeric data input
                        print c
                        imu1.cur_yaw = float(c[3]) * DtoR
                        imu1.cur_pitch = float(c[1]) * DtoR
                        imu1.cur_roll = float(c[2]) * DtoR

                        imu2.cur_yaw = float(c[3]) * DtoR
                        imu2.cur_pitch = float(c[4]) * DtoR
                        imu2.cur_roll = float(c[5]) * DtoR

                plane = reset(plane, imu1.cur_yaw, imu1.cur_pitch, imu1.cur_roll, vp.color.red)
                plane2 = reset(plane2, imu2.cur_yaw, imu2.cur_pitch, imu2.cur_roll, vp.color.
                    green)
                text2 = text_reset(text2, imu1.cur_yaw, -4, -9, 0)
                text3 = text_reset(text3, imu1.cur_pitch, 0, -9, 0)
                text4 = text_reset(text4, imu1.cur_roll, 4, -9, 0 )
                text5 = text_reset(text5, imu2.cur_yaw, -4, -10, 0)
                text6 = text_reset(text6, imu2.cur_pitch, 0, -10, 0 )
                text7 = text_reset(text7, imu2.cur_roll, 4, -10, 0 )

        SERIRAL_INPUT.close() # Only executes once the loop exits


if __name__ == "__main__":
        yawA = []
        # for loop allows serial reading for couple of seconds
        # to avoid error
        print "initializing..."
```

```
        serial_line = SERIRAL_INPUT.readline().rstrip()

        while (serial_line != 'DONE'):
                serial_line = SERIRAL_INPUT.readline().rstrip()
                print serial_line


        display()
```

```
/*
Drew Hamm
Jiongcheng Luo
Krisna Irawan
Project information:
Auto_Alignment.ino
CS462 CS Senior Capstone
Fall 2017 - Spring 2017
Group 65

Program description:
Initializes two SparkFun MPU-9250 Breakout boards to communicate with a Adafruit Metro Mini 328 -
    5V 16MHz board.
Communication via a single I2C channel.
Initializes both MPU-9250s with onboard error correcting values.
Both MPU-9250 accelerometer, gyroscope and magnetometer is requested by and sent to the Metro
    Mini 328.
A sensor fusion algorithm is applied to data recieved from each MPU-2950.
Data is converted into the quaternion form.
The quaternion data from each sensor is used to find the realtime alignment offset between each
    MPU-9250.

TODO:
Output the error variance along with alignment offset data.
Apply statistical analysis to alignment offset data until error variance is reduced to an
    acceptable range.
Transition from finding and recording the initial alignment offset to finding the dynamic
    alignment offset.
note: dynamic alignment offset is in regards to to airframe droop problem.

Note:
SDA and SCL should have external pull-up resistors (to 3.3V).
10k resistors are on the EMSENSR-9250 breakout board.

Hardware Devices:
Adafruit Metro Mini 328 - 5V 16MHz  => MetroMini328
SparkFun MPU-9250 Breakout          => MPU[0]
SparkFun MPU-9250 Breakout          => MPU[1]

Hardware Modifications:
MetroMini328   -> Convert GPIO Logic level to 3.3V down from 5V
            => Cut and solder closed a jumper on the bottom of the board
MPU[0], MPU[1] -> Enable AD0 to control the I2C address
            => Resolder the SJ2 jumper to connect the middle and left pads.

Hardware Connections:
MetroMini328(3.3v) -> (MPU[0], MPU[1])(VDD)
MetroMini328(GND)  -> (MPU[0], MPU[1])(GND)
MetroMini328(A4)   -> (MPU[0], MPU[1])(SDA)
MetroMini328(A5)   -> (MPU[0], MPU[1])(SCL)
MetroMini328(12)   -> MPU[0](AD0)
MetroMini328(13)   -> MPU[1](AD0)
*/
#include "stdafx.h"

#define YAW 0
#define PITCH 1
#define ROLL 2
#define maxSample 60



void output_mag();
void output_gyro();
void output_acc();
void output_Yaw_Pitch_Roll();
```

```
// The two digital login pins on the MetroMini328 that will be used
// to set the AD0 pins of each MPU9250
int AD0[2] = {12, 13};

// The two mpus connected to the microcontroller
// One should have their AD0 pin set HIGH wheras the other should
// have their AD0 pin set LOW
MPU9250 mpu[2];

int numberOfMPUs = 2;
float samples[maxSample];

void setup()
{
// Begin Wire to enable I2C communication
Wire.begin();
delay(1000);

// Begin Serial to enable recieve output from microcontroller to pc
Serial.begin(38400);

// Set AD0 pins for each MPU in order to change their address
// Store the respective address within each individual IMU
if(SERIAL_DEBUG){
Serial.println("Set ADO pins");
}

for(int i = 0; i < numberOfMPUs; i++){
// Set AD0
pinMode(AD0[i], OUTPUT);
digitalWrite(AD0[i], i);
delay(100);

// Set address for each device per its ADO pin value
if(i){
  mpu[i].set_i2c_address(MPU9250_ADDRESS_ADO_1);
}else{
  mpu[i].set_i2c_address(MPU9250_ADDRESS_ADO_0);
}
}

delay(100);

// Setup both mpus
if(SERIAL_DEBUG){
Serial.println("Setup both MPUS");
}
for(int i = 0; i < numberOfMPUs; i++){
mpu[i].initMPU9250();
}

if(1){
Serial.println("DONE");
}

for(int i = 0; i < maxSample; i++){
samples[i] = 0.0;
}

}
int c_sample=0;

unsigned long last = millis();
unsigned long now = last;
unsigned long change_in_time = 0;
//unsigned long d_t = 100;
unsigned long d_t = 20;

bool gotDataFrom_A = 0;
bool gotDataFrom_B = 0;
bool q_diff_found = 0;
bool static_offset_unknown = true;

float yaw_confidence_level = 0.0;
```

```
float pitch_confidence_level = 0.0;
float roll_confidence_level = 0.0;
float static_yaw_offset = 0.0;
float static_pitch_offset = 0.0;
float static_roll_offset = 0.0;
float dynamic_yaw_offset = 0.0;
float dynamic_pitch_offset = 0.0;
float dynamic_roll_offset = 0.0;

Quaternion *q_diff;
float yaw_offset = 0.0;
float pitch_offset = 0.0;
float roll_offset = 0.0;
float yaw = 0.0;
float pitch = 0.0;
float roll = 0.0;

int axis = YAW;
int j = 0;


float g_x = 0.0;
float g_y = 0.0;
float g_z = 0.0;
void loop()
{
for(int i = 0; i < numberOfMPUs; i++){

// Check if new data as been accumulated
if (mpu[i].hasData()){
  //Serial.print(i);
  //Serial.println(" has data");
  mpu[i].retrieve_data();
  if(i == 0){
    gotDataFrom_A = 1;
  }else{
    gotDataFrom_B = 1;
  }
}

//updates time and the quaternion
mpu[i].updateQuaternion();

if(gotDataFrom_A == 1 && gotDataFrom_B == 1){
  q_diff_found = 1;
  gotDataFrom_A = 0;
  gotDataFrom_B = 0;
  now = millis();
  change_in_time = now - last;
  if(change_in_time > d_t){
    last = millis();
    output_Yaw_Pitch_Roll();
  }
}

/*
if(static_offset_unknown){
  // Take the difference from the latest samples
  if(gotDataFrom_A == 1 && gotDataFrom_B == 1){
    // Update flags
    //q_diff_found = true;
    gotDataFrom_A = 0;
    gotDataFrom_B = 0;
    //if(q_diff){
    //  free(q_diff);
    //  q_diff = NULL;
    //}

    //Get the next sample
    switch(axis){
      case YAW:
        samples[c_sample] = (quaternionToYaw(mpu[0].q_m.q) - quaternionToYaw(mpu[1].q_m.q));
        break;
      case PITCH:
        samples[c_sample] = (quaternionToPitch(mpu[0].q_m.q) - quaternionToYaw(mpu[1].q_m.q));
```

```
            break;
          case ROLL:
            samples[c_sample] = (quaternionToRoll(mpu[0].q_m.q) - quaternionToRoll(mpu[1].q_m.q));
            break;
        }

        //Update current sample count
        c_sample = c_sample + 1;

        //Check if last sample was received
        if(c_sample == maxSample){

          for(int k = 0; k < maxSample; k++){
            Serial.println(samples[k]);
          }
          Serial.print("Max sample reached for ");
          switch(axis){
            case YAW:
              Serial.println("yaw.");
              break;

            case PITCH:
              Serial.println("pitch.");
              break;

            case ROLL:
              Serial.println("roll.");
              break;
          }

          //Get mean
          float mean = getMean(samples, maxSample);
          Serial.print("mean: ");
          Serial.println(mean);

          //Get standard deviation
          float standard_deviation = getStandardDeviation(samples, mean, maxSample);
          Serial.print("standard_deviation: ");
          Serial.println(standard_deviation);
          //Get confidence interval
          float confidence_level = getConfidenceInterval(standard_deviation, maxSample, 1.96);

          //Check confidence interval before continuing
          //Output results when passing
          if(confidence_level < 0.001){
            switch(axis){
              case YAW:
                yaw_confidence_level = confidence_level;
                static_yaw_offset = mean;
                axis = PITCH;
                c_sample = 0;

                Serial.print("Yaw offset: ");
                Serial.print(static_yaw_offset, 5);
                Serial.print(" +- ");
                Serial.println(yaw_confidence_level, 5);
                break;
              case PITCH:
                pitch_confidence_level = confidence_level;
                static_pitch_offset = mean;
                axis = ROLL;
                c_sample = 0;

                Serial.print("Pitch offset: ");
                Serial.print(static_pitch_offset, 5);
                Serial.print(" +- ");
                Serial.println(pitch_confidence_level, 5);
                break;
              case ROLL:
                roll_confidence_level = confidence_level;
                static_roll_offset = mean;
                static_offset_unknown = false;

                Serial.print("Roll offset: ");
                Serial.print(static_roll_offset, 5);
```

```
            Serial.print(" +- ");
            Serial.println(roll_confidence_level, 5);
            Serial.println("DONE");
            numberOfMPUs = 1;
            //free(samples);
            break;
          }
        }else{
          Serial.println("Confidence level insufficient");
          c_sample = 0;
        }//confidence interval
      }//max sample reached
    }//both have data
  }//static offset unknown
  else{
    // Output data
    now = millis();
    change_in_time = now - last;
    if(change_in_time > d_t){
      last = millis();
      yaw = quaternionToYaw(mpu[0].q_m.q);// * PI / 180.0f;
      pitch = quaternionToPitch(mpu[0].q_m.q);// * PI / 180.0f;
      roll = quaternionToRoll(mpu[0].q_m.q);//  * PI / 180.0f;

      //Output 'Airplane data'
      Serial.print(yaw, 5);
      Serial.print(",");
      Serial.print(pitch, 5);
      Serial.print(",");
      Serial.print(roll, 5);
      Serial.print(",");
      //Output 'Corrected data'
      Serial.print(yaw + static_yaw_offset, 5);
      Serial.print(",");
      Serial.print(pitch + static_pitch_offset, 5);
      Serial.print(",");
      Serial.println(roll + static_roll_offset, 5);
    }// d_t
  }
  */
  }//for

  }//loop



  void output_mag(){
  Serial.print(mpu[0].mx);
  Serial.print(",");
  Serial.print(mpu[0].my);
  Serial.print(",");
  Serial.print(mpu[0].mz);
  Serial.print(",");
  Serial.print(mpu[1].mx);
  Serial.print(",");
  Serial.print(mpu[1].my);
  Serial.print(",");
  Serial.println(mpu[1].mz);
  }

  void output_gyro(){
  Serial.print(mpu[0].gy);
  Serial.print(",");
  Serial.println(mpu[1].gy);
  }

  void output_acc(){
  Serial.print(mpu[0].ax);
  Serial.print(",");
  Serial.print(mpu[0].ay);
  Serial.print(",");
  Serial.print(mpu[0].az);
  Serial.print(",");
  Serial.print(mpu[1].ax);
  Serial.print(",");
```

```
Serial.print(mpu[1].ay);
Serial.print(",");
Serial.println(mpu[1].az);
}

void output_Yaw_Pitch_Roll(){
float yaw_0 = quaternionToYaw(mpu[0].q_m.q) * RAD_TO_DEG;
float pitch_0 = quaternionToPitch(mpu[0].q_m.q) * RAD_TO_DEG;
float roll_0 = quaternionToRoll(mpu[0].q_m.q) * RAD_TO_DEG;
float yaw_1 = quaternionToYaw(mpu[1].q_m.q) * RAD_TO_DEG;
float pitch_1 = quaternionToPitch(mpu[1].q_m.q) * RAD_TO_DEG;
float roll_1 = quaternionToRoll(mpu[1].q_m.q) * RAD_TO_DEG;
Serial.print(yaw_0);
Serial.print(",");
Serial.print(pitch_0);
Serial.print(",");
Serial.print(roll_0);
Serial.print(",");
Serial.print(yaw_1);
Serial.print(",");
Serial.print(pitch_1);
Serial.print(",");
Serial.println(roll_1);
}
```

```
/*

        data_utils.h

*/
#ifndef _DATAUTILS_H_
#define _DATAUTILS_H_

float getMean(float *samples, int sampleSize);
float getStandardDeviation(float *samples, float mean, int sampleSize);
float getConfidenceInterval(float standardDeviation, int sampleSize, float z_value);

double quaternionToPhi(Quaternion *q);
double quaternionToTheta(Quaternion *q);
double quaternionToPsi(Quaternion *q);

float quaternionToYaw(Quaternion *q);
float quaternionToPitch(Quaternion *q);
float quaternionToRoll(Quaternion *q);

Quaternion* q_difference(Quaternion *q1, Quaternion *q2);

Quaternion* q_inverse(Quaternion *q);
Quaternion* q_conjugate(Quaternion *q);
float q_dot(Quaternion *q1, Quaternion *q2);
Quaternion* q_mult(Quaternion *q1, Quaternion *q2);
Quaternion* q_div(Quaternion *q, float c);

#endif
```

```
/*

        data_utils.cpp

*/

#include "stdafx.h"

float getMean(float * samples, int sampleSize){
  float mean = 0.0;
  for(int i = 0; i < sampleSize; i++){
    mean += samples[i];
  }
  mean = mean / sampleSize;
  return mean;
}

float getStandardDeviation(float * samples, float mean, int sampleSize){
  float standardDeviation = 0.0;
  for(int i = 0; i < sampleSize; i++){
```

```
    standardDeviation += (samples[i] - mean) * (samples[i] - mean);
  }
  standardDeviation = sqrt(standardDeviation / (float)sampleSize);
  return standardDeviation;
}

float getConfidenceInterval(float standardDeviation, int sampleSize, float z_value){
  float confidenceInterval = 0.0;
  confidenceInterval = z_value * (standardDeviation / sqrt((float)sampleSize));
  return confidenceInterval;
}


// Define output variables from updated quaternion---these are Tait-Bryan
// angles, commonly used in aircraft orientation. In this coordinate system,
// the positive z-axis is down toward Earth. Yaw is the angle between Sensor
// x-axis and Earth magnetic North (or true North if corrected for local
// declination, looking down on the sensor positive yaw is counterclockwise.
// Pitch is angle between sensor x-axis and Earth ground plane, toward the
// Earth is positive, up toward the sky is negative. Roll is angle between
// sensor y-axis and Earth ground plane, y-axis up is positive roll. These
// arise from the definition of the homogeneous rotation matrix constructed
// from quaternions. Tait-Bryan angles as well as Euler angles are
// non-commutative; that is, the get the correct orientation the rotations
// must be applied in the correct order which for this configuration is yaw,
// pitch, and then roll.
// For more see
// http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles
// which has additional links.

//Returns yaw in radians
float quaternionToYaw(Quaternion *q){
  float q2q3 = q->q2 * q->q3;
  float q1q4 = q->q1 * q->q4;
  float q1q1 = q->q1 * q->q1;
  float q2q2 = q->q2 * q->q2;
  float q3q3 = q->q3 * q->q3;
  float q4q4 = q->q4 * q->q4;
  float yaw;
  yaw = atan2f(2.0f * (q2q3 + q1q4), q1q1 + q2q2 - q3q3 - q4q4);

  //Declination of corvallis
  yaw += (15.14 * DEG_TO_RAD);

  return yaw;
}

//Returns pitch in radians
float quaternionToPitch(Quaternion *q){
  float q2q4 = q->q2 * q->q4;
  float q1q3 = q->q1 * q->q3;
  float pitch;
  pitch = -asinf(2.0f * (q2q4 - q1q3));

  return pitch;
}

//Returns roll in radians
float quaternionToRoll(Quaternion *q){
  float q1q2 = q->q1 * q->q2;
  float q3q4 = q->q3 * q->q4;
  float q1q1 = q->q1 * q->q1;
  float q2q2 = q->q2 * q->q2;
  float q3q3 = q->q3 * q->q3;
  float q4q4 = q->q4 * q->q4;

  float roll;
  roll = atan2f(2.0f * (q1q2 + q3q4), q1q1 - q2q2 - q3q3 + q4q4);

  return roll;
}


double quaternionToPhi(Quaternion *q){
  double q0 = q->q1;
```

```
  double q1 = q->q2;
  double q2 = q->q3;
  double q3 = q->q4;

  double R32 = 2.0 * (q2*q3 - q0*q1);
  double R33 = 2.0 * q0*q0 - 1.0 + 2.0 * q3*q3;
  double phi = atan2(R32, R33 );

  return phi;
}

double quaternionToTheta(Quaternion *q){
  double q0 = q->q1;
  double q1 = q->q2;
  double q2 = q->q3;
  double q3 = q->q4;

  double R31 = 2.0 * (q1*q3 + q0*q2);
  double theta = -atan(R31 / sqrt(1.0 - R31*R31));

  return theta;
}

double quaternionToPsi(Quaternion *q){
  double q0 = q->q1;
  double q1 = q->q2;
  double q2 = q->q3;
  double q3 = q->q4;

  double R11 = 2.0 * q0*q0 - 1.0 + 2.0 * q1*q1;
  double R21 = 2.0 * (q1*q2 - q0*q3);
  double psi = atan2(R21, R11 );

  return psi;

}

Quaternion* q_difference(Quaternion *q1, Quaternion *q2){
  Quaternion *q_diff;
  Quaternion *q_inv;
  q_inv = q_inverse(q1);
  q_diff = q_mult(q2, q_inv);

  //Deconstruct
  //q_inv
  if(q_inv){
    free(q_inv);
    q_inv = NULL;
  }

  return q_diff;
}

Quaternion* q_inverse(Quaternion *q){
  Quaternion *q_di;
  Quaternion *q_co;
  float dot = 1.0;
  q_co = q_conjugate(q);
  //dot = q_dot(q, q);
  q_di = q_div(q_co, dot);

  //Deconstruct
  //q_co, q_do
  if(q_co){
    free(q_co);
    q_co = NULL;
  }
  return q_di;
}

Quaternion* q_conjugate(Quaternion *q){
  Quaternion *q_c = new Quaternion();
  float q1 = q->q1;
  float q2 = -(q->q2);
  float q3 = -(q->q2);
```

```
  float q4 = -(q->q2);

  q_c->q1 = q1;
  q_c->q2 = q2;
  q_c->q3 = q3;
  q_c->q4 = q4;

  return q_c;
}

float q_dot(Quaternion *a, Quaternion *b){
  float a1b1 = a->q1 * b->q1;
  float a2b2 = a->q2 * b->q2;
  float a3b3 = a->q3 * b->q3;
  float a4b4 = a->q4 * b->q4;

  float result = a1b1 + a2b2 + a3b3 + a4b4;
  return result;
}

Quaternion* q_mult(Quaternion *a, Quaternion *b){
  Quaternion *q_m = new Quaternion();
  float a1b1 = a->q1 * b->q1;
  float a2b2 = a->q2 * b->q2;
  float a3b3 = a->q3 * b->q3;
  float a4b4 = a->q4 * b->q4;

  float a2b1 = a->q2 * b->q1;
  float a1b2 = a->q1 * b->q2;
  float a4b3 = a->q4 * b->q3;
  float a3b4 = a->q3 * b->q4;

  float a3b1 = a->q3 * b->q1;
  float a4b2 = a->q4 * b->q2;
  float a1b3 = a->q1 * b->q3;
  float a2b4 = a->q2 * b->q4;

  float a4b1 = a->q4 * b->q1;
  float a3b2 = a->q3 * b->q2;
  float a2b3 = a->q2 * b->q3;
  float a1b4 = a->q1 * b->q4;

  q_m->q1 = a1b1 - a2b2 - a3b3 - a4b4;
  q_m->q2 = a2b1 + a1b2 - a4b3 + a3b4;
  q_m->q3 = a3b1 + a4b2 + a1b3 - a2b4;
  q_m->q4 = a4b1 - a3b2 + a2b3 + a1b4;
  return q_m;
}

Quaternion* q_div(Quaternion *q, float c){
  Quaternion *q_d = new Quaternion();
  float q1_c = q->q1 / c;
  float q2_c = q->q2 / c;
  float q3_c = q->q3 / c;
  float q4_c = q->q4 / c;

  q_d->q1 = q1_c;
  q_d->q2 = q2_c;
  q_d->q3 = q3_c;
  q_d->q4 = q4_c;

  return q_d;
}
/*
void q_print(Quaternion *q){
  int precision = 5;
  Serial.print("q1: ");
  Serial.print(q -> q1, precision);
  Serial.print(" q2: ");
  Serial.print(q -> q2, precision);
  Serial.print(" q3: ");
  Serial.print(q -> q3, precision);
  Serial.print(" q4: ");
  Serial.println(q -> q4, precision);
}
```

```
*/

/*

        debug_config.h

*/
#ifndef _DEBUG_CONFIG_H_
#define _DEBUG_CONFIG_H_

#define SERIAL_DEBUG true
#define EXTRA_GYRO_INFO false
#endif

/*

        MPU9250.h

*/


/*
 Note: The MPU9250 is an I2C sensor and uses the Arduino Wire library.
 Because the sensor is not 5V tolerant, we are using a 3.3 V 8 MHz Pro Mini or
 a 3.3 V Teensy 3.1. We have disabled the internal pull-ups used by the Wire
 library in the Wire.h/twi.c utility file. We are also using the 400 kHz fast
 I2C mode by setting the TWI_FREQ  to 400000L /twi.h utility file.
 */
#ifndef _MPU9250_H_
#define _MPU9250_H_

#define X_AXIS 0
#define Y_AXIS 1
#define Z_AXIS 2


// See also MPU-9250 Register Map and Descriptions, Revision 4.0,
// RM-MPU-9250A-00, Rev. 1.4, 9/9/2013 for registers not listed in above
// document; the MPU9250 and MPU9150 are virtually identical but the latter has
// a different register map

//Magnetometer Registers
#define AK8963_ADDRESS   0x0C
#define WHO_AM_I_AK8963  0x00//0x49 // (AKA WIA) should return 0x48
#define INFO             0x01
#define AK8963_ST1       0x02  // data ready status bit 0
#define AK8963_XOUT_L    0x03  // data
#define AK8963_XOUT_H    0x04
#define AK8963_YOUT_L    0x05
#define AK8963_YOUT_H    0x06
#define AK8963_ZOUT_L    0x07
#define AK8963_ZOUT_H    0x08
#define AK8963_ST2       0x09  // Data overflow bit 3 and data read error status bit 2
#define AK8963_CNTL      0x0A  // Power down (0000), single-measurement (0001), self-test (1000)
    and Fuse ROM (1111) modes on bits 3:0
#define AK8963_ASTC      0x0C  // Self test control
#define AK8963_I2CDIS    0x0F  // I2C disable
#define AK8963_ASAX      0x10  // Fuse ROM x-axis sensitivity adjustment value
#define AK8963_ASAY      0x11  // Fuse ROM y-axis sensitivity adjustment value
#define AK8963_ASAZ      0x12  // Fuse ROM z-axis sensitivity adjustment value

#define AK8963_CNTL1      0x0A // Control 1 register
#define AK8963_CNTL2      0x0B // Control 2 register

// AK8963 Control Register Signals
#define CNTL1_16BIT_OUTPUT 0x10
#define CNTL1_CONTINUOUS_MEASUREMENT_MODE_1 0x02
#define CNTL1_CONTINUOUS_MEASUREMENT_MODE_2 0x06 // 100Hz
#define CNTL1_FUSE_ROM_MODE 0x0F
#define CNTL2_SOFT_RESTART 0x01

#define SELF_TEST_X_GYRO 0x00
#define SELF_TEST_Y_GYRO 0x01
#define SELF_TEST_Z_GYRO 0x02
```

```
/*#define X_FINE_GAIN      0x03 // [7:0] fine gain
#define Y_FINE_GAIN       0x04
#define Z_FINE_GAIN       0x05
#define XA_OFFSET_H       0x06 // User-defined trim values for accelerometer
#define XA_OFFSET_L_TC    0x07
#define YA_OFFSET_H       0x08
#define YA_OFFSET_L_TC    0x09
#define ZA_OFFSET_H       0x0A
#define ZA_OFFSET_L_TC    0x0B */

#define SELF_TEST_X_ACCEL 0x0D
#define SELF_TEST_Y_ACCEL 0x0E
#define SELF_TEST_Z_ACCEL 0x0F

#define SELF_TEST_A       0x10

#define XG_OFFSET_H       0x13  // User-defined trim values for gyroscope
#define XG_OFFSET_L       0x14
#define YG_OFFSET_H       0x15
#define YG_OFFSET_L       0x16
#define ZG_OFFSET_H       0x17
#define ZG_OFFSET_L       0x18
#define SMPLRT_DIV        0x19
#define CONFIG            0x1A
#define GYRO_CONFIG       0x1B
#define ACCEL_CONFIG      0x1C
#define ACCEL_CONFIG2     0x1D
#define LP_ACCEL_ODR      0x1E
#define WOM_THR           0x1F

// Duration counter threshold for motion interrupt generation, 1 kHz rate,
// LSB = 1 ms
#define MOT_DUR           0x20
// Zero-motion detection threshold bits [7:0]
#define ZMOT_THR          0x21
// Duration counter threshold for zero motion interrupt generation, 16 Hz rate,
// LSB = 64 ms
#define ZRMOT_DUR         0x22

#define FIFO_EN           0x23
#define I2C_MST_CTRL      0x24
#define I2C_SLV0_ADDR     0x25
#define I2C_SLV0_REG      0x26
#define I2C_SLV0_CTRL     0x27
#define I2C_SLV1_ADDR     0x28
#define I2C_SLV1_REG      0x29
#define I2C_SLV1_CTRL     0x2A
#define I2C_SLV2_ADDR     0x2B
#define I2C_SLV2_REG      0x2C
#define I2C_SLV2_CTRL     0x2D
#define I2C_SLV3_ADDR     0x2E
#define I2C_SLV3_REG      0x2F
#define I2C_SLV3_CTRL     0x30
#define I2C_SLV4_ADDR     0x31
#define I2C_SLV4_REG      0x32
#define I2C_SLV4_DO       0x33
#define I2C_SLV4_CTRL     0x34
#define I2C_SLV4_DI       0x35
#define I2C_MST_STATUS    0x36
#define INT_PIN_CFG       0x37
#define INT_ENABLE        0x38
#define DMP_INT_STATUS    0x39  // Check DMP interrupt
#define INT_STATUS        0x3A
#define ACCEL_XOUT_H      0x3B
#define ACCEL_XOUT_L      0x3C
#define ACCEL_YOUT_H      0x3D
#define ACCEL_YOUT_L      0x3E
#define ACCEL_ZOUT_H      0x3F
#define ACCEL_ZOUT_L      0x40
#define TEMP_OUT_H        0x41
#define TEMP_OUT_L        0x42
#define GYRO_XOUT_H       0x43
#define GYRO_XOUT_L       0x44
#define GYRO_YOUT_H       0x45
#define GYRO_YOUT_L       0x46
```

```
#define GYRO_ZOUT_H         0x47
#define GYRO_ZOUT_L         0x48
#define EXT_SENS_DATA_00    0x49
#define EXT_SENS_DATA_01    0x4A
#define EXT_SENS_DATA_02    0x4B
#define EXT_SENS_DATA_03    0x4C
#define EXT_SENS_DATA_04    0x4D
#define EXT_SENS_DATA_05    0x4E
#define EXT_SENS_DATA_06    0x4F
#define EXT_SENS_DATA_07    0x50
#define EXT_SENS_DATA_08    0x51
#define EXT_SENS_DATA_09    0x52
#define EXT_SENS_DATA_10    0x53
#define EXT_SENS_DATA_11    0x54
#define EXT_SENS_DATA_12    0x55
#define EXT_SENS_DATA_13    0x56
#define EXT_SENS_DATA_14    0x57
#define EXT_SENS_DATA_15    0x58
#define EXT_SENS_DATA_16    0x59
#define EXT_SENS_DATA_17    0x5A
#define EXT_SENS_DATA_18    0x5B
#define EXT_SENS_DATA_19    0x5C
#define EXT_SENS_DATA_20    0x5D
#define EXT_SENS_DATA_21    0x5E
#define EXT_SENS_DATA_22    0x5F
#define EXT_SENS_DATA_23    0x60
#define MOT_DETECT_STATUS   0x61
#define I2C_SLV0_DO         0x63
#define I2C_SLV1_DO         0x64
#define I2C_SLV2_DO         0x65
#define I2C_SLV3_DO         0x66
#define I2C_MST_DELAY_CTRL  0x67
#define SIGNAL_PATH_RESET   0x68
#define MOT_DETECT_CTRL     0x69
#define USER_CTRL           0x6A  // Bit 7 enable DMP, bit 3 reset DMP
#define PWR_MGMT_1          0x6B // Device defaults to the SLEEP mode
#define PWR_MGMT_2          0x6C
#define DMP_BANK            0x6D  // Activates a specific bank in the DMP
#define DMP_RW_PNT          0x6E  // Set read/write pointer to a specific start address in
    specified DMP bank
#define DMP_REG             0x6F  // Register in DMP from which to read or to which to write
#define DMP_REG_1           0x70
#define DMP_REG_2           0x71
#define FIFO_COUNTH         0x72
#define FIFO_COUNTL         0x73
#define FIFO_R_W            0x74
#define WHO_AM_I_MPU9250    0x75 // Should return 0x71
#define XA_OFFSET_H         0x77
#define XA_OFFSET_L         0x78
#define YA_OFFSET_H         0x7A
#define YA_OFFSET_L         0x7B
#define ZA_OFFSET_H         0x7D
#define ZA_OFFSET_L         0x7E

//MPU9250 Slave Control Register Signals
#define I2C_SLV0_EN         0x80

// Using the MPU-9250 breakout board, ADO is set to 0
// Seven-bit device address is 110100 for ADO = 0 and 110101 for ADO = 1
#define MPU9250_ADDRESS_ADO_1 0x69  // Device address when ADO = 1
#define MPU9250_ADDRESS_ADO_0 0x68  // Device address when ADO = 0
#define AK8963_ADDRESS  0x0C   // Address of magnetometer

#define READ_FLAG 0x80

class MPU9250
{
  private:
    uint8_t delay_s = 10;
    uint8_t delay_m = 35;
    uint8_t delay_l = 100;

    bool read_Magnetometer();
    bool read_Accelerometer();
    bool read_Gyroscope();
```

```
    bool initMPU9250_Magnetometer();
    bool initMPU9250_Accelerometer();
    bool initMPU9250_Gyroscope();
    bool calibrateMPU9250_Magnetometer();
    bool calibrateMPU9250_Accelerometer();
    bool calibrateMPU9250_Gyroscope();

  protected:
    uint8_t i2c_address = MPU9250_ADDRESS_ADO_0;

    // Set initial input parameters
    enum Ascale
    {
      AFS_2G = 0,
      AFS_4G,
      AFS_8G,
      AFS_16G
    };

    enum Gscale {
      GFS_250DPS = 0,
      GFS_500DPS,
      GFS_1000DPS,
      GFS_2000DPS
    };

    enum Mscale {
      MFS_14BITS = 0, // 0.6 mG per LSB
      MFS_16BITS      // 0.15 mG per LSB
    };

    enum M_MODE {
      M_8HZ = 0x02,  // 8 Hz update
      M_100HZ = 0x06 // 100 Hz continuous magnetometer
    };

    // TODO: Add setter methods for this hard coded stuff
    // Specify sensor full scale
    uint8_t Gscale = GFS_250DPS;
    uint8_t Ascale = AFS_2G;
    // Choose either 14-bit or 16-bit magnetometer resolution
    uint8_t Mscale = MFS_16BITS;

    // 2 for 8 Hz, 6 for 100 Hz continuous magnetometer data read
    uint8_t Mmode = M_100HZ;

  public:
    Quaternion_Module q_m;

    float temperature;   // Stores the real internal chip temperature in Celsius
    int16_t tempCount;   // Temperature raw count output

    // Stores the 16-bit signed sensor outputs
    int16_t accelCount[3], gyroCount[3], magCount[3];

    // Scale resolutions per LSB for the sensors
    float aRes, gRes, mRes;

    // Variables to hold latest sensor data values
    float ax, ay, az, gx, gy, gz, mx, my, mz;

    // Factory mag calibration and mag bias
    float factoryMagCalibration[3] = {0, 0, 0}, factoryMagBias[3] = {0, 0, 0};

    // Bias corrections for gyro, accelerometer, and magnetometer
    float gyroBias[3]  = {0, 0, 0},
          accelBias[3] = {0, 0, 0},
          magBias[3]   = {0, 0, 0},
          magScale[3]  = {0, 0, 0};
    float selfTest[6];

    // Public method declarations
    void getMres();
    void getAres();
    void getGres();
```

```
    uint16_t getSampleCount();
    uint16_t getSampleDelay();
    uint8_t writeByte(uint8_t, uint8_t, uint8_t);
    uint8_t readByte(uint8_t, uint8_t);
    uint8_t readBytes(uint8_t, uint8_t, uint8_t, uint8_t *);
    uint8_t getAscale();
    uint8_t getGscale();
    uint8_t getMscale();
    uint8_t getM_Mode();

    bool set_i2c_address(uint8_t address);
    void initMPU9250();
    bool hasData();
    void retrieve_data();

    void updateQuaternion();
};

#endif // _MPU9250_H_
```

```
/*

        MPU9250.cpp

*/
#include "stdafx.h"

//===================================================================================
//====== Set of useful function to access acceleration. gyroscope, magnetometer,
//====== and temperature data
//===================================================================================

// If intPin goes high, all data registers have new data
// On interrupt, check if data ready interrupt
bool MPU9250::hasData(){
  return (readByte(i2c_address, INT_STATUS) & 0x01);
}

void MPU9250::getMres()
{
  switch (Mscale)
  {
    // Possible magnetometer scales (and their register bit settings) are:
    // 14 bit resolution (0) and 16 bit resolution (1)
    case MFS_14BITS:
      mRes = 10.0f * 4912.0f / 8190.0f; // Proper scale to return milliGauss
      break;
    case MFS_16BITS:
      mRes = 10.0f * 4912.0f / 32760.0f; // Proper scale to return milliGauss
      break;
  }
}

void MPU9250::getGres()
{
  switch (Gscale)
  {
    // Possible gyro scales (and their register bit settings) are:
    // 250 DPS (00), 500 DPS (01), 1000 DPS (10), and 2000 DPS (11).
    // Here's a bit of an algorith to calculate DPS/(ADC tick) based on that
    // 2-bit value:
    case GFS_250DPS:
      gRes = 250.0f / 32768.0f;
      break;
    case GFS_500DPS:
      gRes = 500.0f / 32768.0f;
      break;
    case GFS_1000DPS:
      gRes = 1000.0f / 32768.0f;
      break;
    case GFS_2000DPS:
      gRes = 2000.0f / 32768.0f;
      break;
  }
}
```

```
void MPU9250::getAres()
{
  switch (Ascale)
  {
    // Possible accelerometer scales (and their register bit settings) are:
    // 2 Gs (00), 4 Gs (01), 8 Gs (10), and 16 Gs  (11).
    // Here's a bit of an algorith to calculate DPS/(ADC tick) based on that
    // 2-bit value:
    case AFS_2G:
      aRes = 2.0f / 32768.0f;
      break;
    case AFS_4G:
      aRes = 4.0f / 32768.0f;
      break;
    case AFS_8G:
      aRes = 8.0f / 32768.0f;
      break;
    case AFS_16G:
      aRes = 16.0f / 32768.0f;
      break;
  }
}

uint8_t MPU9250::getAscale(){
  return Ascale;
}

uint8_t MPU9250::getGscale(){
  return Gscale;
}

uint8_t MPU9250::getMscale(){
  return Mscale;
}

uint8_t MPU9250::getM_Mode(){
  return Mmode;
}

uint16_t MPU9250::getSampleCount(){
  uint16_t sample_count = 0;
  // shoot for ~fifteen seconds of mag data
  // at 8 Hz ODR, new mag data is available every 125 ms
  if (Mmode == M_8HZ)
  {
    sample_count = 128;
  }
  // at 100 Hz ODR, new mag data is available every 10 ms
  if (Mmode == M_100HZ)
  {
    sample_count = 1500;
  }

  return sample_count;
}

uint16_t MPU9250::getSampleDelay(){
  uint16_t sample_delay = 0;

  if (Mmode == M_8HZ)
  {
    sample_delay = 135; // At 8 Hz ODR, new mag data is available every 125 ms
  }
  if (Mmode == M_100HZ)
  {
    sample_delay = 12;  // At 100 Hz ODR, new mag data is available every 10 ms
  }

  return sample_delay;
}


bool MPU9250::set_i2c_address(uint8_t address){
  i2c_address = address;
}
```

```
bool MPU9250::initMPU9250_Magnetometer(){
  // First extract the factory calibration for each magnetometer axis
  uint8_t rawData[3];  // x/y/z gyro calibration data stored here

  // Point to magnetometer before writing data
  // Set to write to slave address AK8963_ADDRESS
  writeByte(i2c_address, I2C_SLV0_ADDR, AK8963_ADDRESS);
  delay(delay_s);

  // Now point to a megnetometer register that data will be written to
  // Point slave 0 register at AK8963's control 2 (soft reset) register
  writeByte(i2c_address, I2C_SLV0_REG, AK8963_CNTL2);
  delay(delay_s);

  // Send data
  // Soft restart via AK8963's control 2
  writeByte(i2c_address, I2C_SLV0_DO, CNTL2_SOFT_RESTART);
  delay(delay_s);

  // Point to a different magnetometer register for a seperate write
  // Point slave 0 register at AK8963's control 1 (mode) register
  writeByte(i2c_address, I2C_SLV0_REG, AK8963_CNTL1);
  delay(delay_s);

  // Send data
  // Enter Fuse ROM access mode
  writeByte(i2c_address, I2C_SLV0_DO, CNTL1_FUSE_ROM_MODE);
  delay(delay_s);

  // Enable read from magnometer
  // Set to read from slave address AK8963_ADDRESS
  writeByte(i2c_address, I2C_SLV0_ADDR, AK8963_ADDRESS | READ_FLAG);
  delay(delay_s);

  // Now point to a megnetometer register that data will be read from
  // Point slave 0 register at AK8963's fuse rom x-axis
  writeByte(i2c_address, I2C_SLV0_REG, AK8963_ASAX);
  delay(delay_s);

  // Enable simple 3-byte I2C read from slave 0
  writeByte(i2c_address, I2C_SLV0_CTRL, READ_FLAG | 0x03);
  delay(delay_1);

  // Read the x-, y-, and z-axis calibration values
  readBytes(i2c_address, EXT_SENS_DATA_00, 3, &rawData[0]);
  delay(delay_1);

  if(SERIAL_DEBUG){
    Serial.print("rawData[0]: "); Serial.print(rawData[0]);
    Serial.print(" ");
    Serial.print("rawData[1]: "); Serial.print(rawData[1]);
    Serial.print(" ");
    Serial.print("rawData[2]: "); Serial.print(rawData[2]);
    Serial.println("");
  }

  //Return x-axis sensitivity adjustment values, etc.
  factoryMagCalibration[0] =  (float)(rawData[0] - 128)/256. + 1.;
  factoryMagCalibration[1] =  (float)(rawData[1] - 128)/256. + 1.;
  factoryMagCalibration[2] =  (float)(rawData[2] - 128)/256. + 1.;

  if(SERIAL_DEBUG){
    for(int i = 0; i < 3; i++){
    Serial.print("factoryMagCalibration[");
    Serial.print(i);
    Serial.print("]: ");
    Serial.print(factoryMagCalibration[i]);
    Serial.println();
    }
  }

  // Set to write to slave address AK8963_ADDRESS
  writeByte(i2c_address, I2C_SLV0_ADDR, AK8963_ADDRESS);
  delay(delay_s);
```

```
  // Point slave 0 register at AK8963's control 1 (mode) register
  writeByte(i2c_address, I2C_SLV0_REG, AK8963_CNTL1);
  delay(delay_s);

  // 16-bit continuous measurement mode via AK8963's control 1
  writeByte(i2c_address, I2C_SLV0_DO, Mscale << 4 | Mmode);
  //writeByte(i2c_address, I2C_SLV0_DO, (CNTL1_16BIT_OUTPUT | CNTL1_CONTINUOUS_MEASUREMENT_MODE_2
      ));
  delay(delay_1);

  // Setup for read
  writeByte(i2c_address, I2C_SLV0_ADDR, AK8963_ADDRESS | READ_FLAG);

  // Setup to read actual mag data
  // Point slave to the data regs
  writeByte(i2c_address, I2C_SLV0_REG, AK8963_XOUT_L);
  delay(delay_s);

  // Enable simple 7-byte I2C reads from slave 0
  writeByte(i2c_address, I2C_SLV0_CTRL, READ_FLAG | 0x07);
  delay(delay_s);

}

bool MPU9250::initMPU9250_Accelerometer(){
  // Set accelerometer full-scale range configuration
  // Get current ACCEL_CONFIG register value
  uint8_t c = readByte(i2c_address, ACCEL_CONFIG);
  // c = c & ~0xE0; // Clear self-test bits [7:5]
  c = c & ~0x18;  // Clear AFS bits [4:3]
  c = c | getAscale() << 3; // Set full scale range for the accelerometer
  // Write new ACCEL_CONFIG register value
  writeByte(i2c_address, ACCEL_CONFIG, c);

  // Set accelerometer sample rate configuration
  // It is possible to get a 4 kHz sample rate from the accelerometer by
  // choosing 1 for accel_fchoice_b bit [3]; in this case the bandwidth is
  // 1.13 kHz
  // Get current ACCEL_CONFIG2 register value
  c = readByte(i2c_address, ACCEL_CONFIG2);
  c = c & ~0x0F; // Clear accel_fchoice_b (bit 3) and A_DLPFG (bits [2:0])
  c = c | 0x03;  // Set accelerometer rate to 1 kHz and bandwidth to 41 Hz
  // Write new ACCEL_CONFIG2 register value
  writeByte(i2c_address, ACCEL_CONFIG2, c);
}

bool MPU9250::initMPU9250_Gyroscope(){
  // Configure Gyro and Thermometer
  // Disable FSYNC and set thermometer and gyro bandwidth to 41 and 42 Hz,
  // respectively;
  // minimum delay time for this setting is 5.9 ms, which means sensor fusion
  // update rates cannot be higher than 1 / 0.0059 = 170 Hz
  // DLPF_CFG = bits 2:0 = 011; this limits the sample rate to 1000 Hz for both
  // With the MPU9250, it is possible to get gyro sample rates of 32 kHz (!),
  // 8 kHz, or 1 kHz
  writeByte(i2c_address, CONFIG, 0x03);

  // Set sample rate = gyroscope output rate/(1 + SMPLRT_DIV)
  // Use a 200 Hz rate; a rate consistent with the filter update rate
  // determined inset in CONFIG above.
  writeByte(i2c_address, SMPLRT_DIV, 0x04);

  // Set gyroscope full scale range
  // Range selects FS_SEL and AFS_SEL are 0 - 3, so 2-bit values are
  // left-shifted into positions 4:3

  // get current GYRO_CONFIG register value
  uint8_t c = readByte(i2c_address, GYRO_CONFIG);
  // c = c & ~0xE0; // Clear self-test bits [7:5]
  c = c & ~0x03; // Clear Fchoice bits [1:0]
  c = c & ~0x18; // Clear AFS bits [4:3]
  c = c | getGscale() << 3; // Set full scale range for the gyro
  // Set Fchoice for the gyro to 11 by writing its inverse to bits 1:0 of
  // GYRO_CONFIG
```

```
  // c =| 0x00;
  // Write new GYRO_CONFIG value to register
  writeByte(i2c_address, GYRO_CONFIG, c );

}

//Copied from xxx
bool MPU9250::calibrateMPU9250_Magnetometer(){
  uint16_t ii = 0, sample_count = 0, sample_delay = 0;
  int32_t mag_bias[3]  = {0, 0, 0},
          mag_scale[3] = {0, 0, 0};
  int16_t mag_max[3]  = {0x8000, 0x8000, 0x8000},
          mag_min[3]  = {0x7FFF, 0x7FFF, 0x7FFF};

  // Make sure resolution has been calculated
  getMres();

  // if(SERIAL_DEBUG){
  //   Serial.println(F("Mag Calibration: Wave device in a figure 8 until done!"));
  //   Serial.println(F("  4 seconds to get ready followed by 15 seconds of sampling)"));
  // }
  delay(4000);

  // Determine the number of samples to take
  sample_count = 128;//getSampleCount();

  // Determine the delay between each sample
  sample_delay = getSampleDelay();

  for (ii = 0; ii < sample_count; ii++)
  {
    read_Magnetometer();

    for (int jj = 0; jj < 3; jj++)
    {
      if (magCount[jj] > mag_max[jj])
      {
        mag_max[jj] = magCount[jj];
      }
      if (magCount[jj] < mag_min[jj])
      {
        mag_min[jj] = magCount[jj];
      }
    }

    delay(13);
  }

  if(SERIAL_DEBUG){
    Serial.println("mag x min/max:"); Serial.println(mag_max[0]); Serial.println(mag_min[0]);
    Serial.println("mag y min/max:"); Serial.println(mag_max[1]); Serial.println(mag_min[1]);
    Serial.println("mag z min/max:"); Serial.println(mag_max[2]); Serial.println(mag_min[2]);
  }

  // Get hard iron correction
  // Get 'average' x mag bias in counts
  mag_bias[0]  = (mag_max[0] + mag_min[0]) / 2;
  // Get 'average' y mag bias in counts
  mag_bias[1]  = (mag_max[1] + mag_min[1]) / 2;
  // Get 'average' z mag bias in counts
  mag_bias[2]  = (mag_max[2] + mag_min[2]) / 2;

  // Save mag biases in G for main program
  magBias[0] = (float)mag_bias[0] * mRes * factoryMagCalibration[0];
  magBias[1] = (float)mag_bias[1] * mRes * factoryMagCalibration[1];
  magBias[2] = (float)mag_bias[2] * mRes * factoryMagCalibration[2];

  // Get soft iron correction estimate
  // Get average x axis max chord length in counts
  mag_scale[0]  = (mag_max[0] - mag_min[0]) / 2;
  // Get average y axis max chord length in counts
  mag_scale[1]  = (mag_max[1] - mag_min[1]) / 2;
  // Get average z axis max chord length in counts
  mag_scale[2]  = (mag_max[2] - mag_min[2]) / 2;
```

```
    float avg_rad = mag_scale[0] + mag_scale[1] + mag_scale[2];
    avg_rad /= 3.0;

    magScale[0] = avg_rad / ((float)mag_scale[0]);
    magScale[1] = avg_rad / ((float)mag_scale[1]);
    magScale[2] = avg_rad / ((float)mag_scale[2]);

    if(SERIAL_DEBUG){
      Serial.println(F("Mag Calibration done!"));
    }
  }

  bool MPU9250::calibrateMPU9250_Accelerometer(){
    uint8_t data[6]; // data array to hold accelerometer data
    uint16_t ii, packet_count, fifo_count;
    int32_t accel_bias[3] = {0, 0, 0};
    int16_t acc_max[3]  = {0x8000, 0x8000, 0x8000},
            acc_min[3]  = {0x7FFF, 0x7FFF, 0x7FFF};

    if(SERIAL_DEBUG){
      Serial.println("calibrateMPU9250_Accelerometer");
    }

    // Make sure resolution has been calculated
    getAres();

    // reset device
    // Write a one to bit 7 reset bit; toggle reset device
    writeByte(i2c_address, PWR_MGMT_1, READ_FLAG);
    delay(100);

    // get stable time source; Auto select clock source to be PLL gyroscope
    // reference if ready else use the internal oscillator, bits 2:0 = 001
    writeByte(i2c_address, PWR_MGMT_1, 0x01);
    writeByte(i2c_address, PWR_MGMT_2, 0x00);
    delay(200);

    // Configure device for bias calculation
    // Disable all interrupts
    writeByte(i2c_address, INT_ENABLE, 0x00);
    // Disable FIFO
    writeByte(i2c_address, FIFO_EN, 0x00);
    // Turn on internal clock source
    writeByte(i2c_address, PWR_MGMT_1, 0x00);
    // Disable I2C master
    writeByte(i2c_address, I2C_MST_CTRL, 0x00);
    // Disable FIFO and I2C master modes
    writeByte(i2c_address, USER_CTRL, 0x00);
    // Reset FIFO and DMP
    writeByte(i2c_address, USER_CTRL, 0x0C);
    delay(15);

    // Configure MPU6050 gyro and accelerometer for bias calculation
    // Set low-pass filter to 188 Hz
    writeByte(i2c_address, CONFIG, 0x01);
    // Set sample rate to 1 kHz
    writeByte(i2c_address, SMPLRT_DIV, 0x00);
    // Set accelerometer full-scale to 2 g, maximum sensitivity
    writeByte(i2c_address, ACCEL_CONFIG, 0x00);

    uint16_t  accelsensitivity = 16384; // = 16384 LSB/g

    // Configure FIFO to capture accelerometer data for bias calculation
    writeByte(i2c_address, USER_CTRL, 0x40);  // Enable FIFO
    // Enable accelerometer sensors for FIFO  (max size 512 bytes in
    // MPU-9150)
    writeByte(i2c_address, FIFO_EN, 0x08);
    delay(80);  // accumulate 80 samples in 80 milliseconds = 480 bytes

    // At end of sample accumulation, turn off FIFO sensor read
    // Disable accelerometer sensors for FIFO
    writeByte(i2c_address, FIFO_EN, 0x00);
    // Read FIFO sample count
    readBytes(i2c_address, FIFO_COUNTH, 2, &data[0]);
    fifo_count = ((uint16_t)data[0] << 8) | data[1];
```

```
        // How many sets of full accelerometer data for averaging

        packet_count = fifo_count/6;
        if(SERIAL_DEBUG){
          Serial.print("fifo_count: ");
          Serial.println(fifo_count);
          Serial.print("packet_count: ");
          Serial.println(packet_count);
        }

        for (ii = 0; ii < packet_count; ii++)
        {
          int16_t accel_temp[3] = {0, 0, 0};
          // Read data for averaging
          readBytes(i2c_address, FIFO_R_W, 6, &data[0]);
          // Form signed 16-bit integer for each sample in FIFO
          accel_temp[0] = (int16_t) (((int16_t)data[0] << 8) | data[1]  );
          accel_temp[1] = (int16_t) (((int16_t)data[2] << 8) | data[3]  );
          accel_temp[2] = (int16_t) (((int16_t)data[4] << 8) | data[5]  );

          // Sum individual signed 16-bit biases to get accumulated signed 32-bit
          // biases.
          accel_bias[0] += (int32_t) accel_temp[0];
          accel_bias[1] += (int32_t) accel_temp[1];
          accel_bias[2] += (int32_t) accel_temp[2];

          for (int jj = 0; jj < 3; jj++)
          {
            if (accel_temp[jj] > acc_max[jj])
            {
              acc_max[jj] = accel_temp[jj];
            }
            if (accel_temp[jj] < acc_min[jj])
            {
              acc_min[jj] = accel_temp[jj];
            }
          }
        }

        if(SERIAL_DEBUG){
          Serial.println("acc x min/max:"); Serial.println(acc_max[0]); Serial.println(acc_min[0]);
          Serial.println("acc y min/max:"); Serial.println(acc_max[1]); Serial.println(acc_min[1]);
          Serial.println("acc z min/max:"); Serial.println(acc_max[2]); Serial.println(acc_min[2]);
        }

        // Sum individual signed 16-bit biases to get accumulated signed 32-bit biases
        accel_bias[0] /= (int32_t) packet_count;
        accel_bias[1] /= (int32_t) packet_count;
        accel_bias[2] /= (int32_t) packet_count;

        // Sum individual signed 16-bit biases to get accumulated signed 32-bit biases
        if (accel_bias[2] > 0L)
        {
          accel_bias[2] -= (int32_t) accelsensitivity;
        }
        else
        {
          accel_bias[2] += (int32_t) accelsensitivity;
        }

        // Construct the accelerometer biases for push to the hardware accelerometer
        // bias registers. These registers contain factory trim values which must be
        // added to the calculated accelerometer biases; on boot up these registers
        // will hold non-zero values. In addition, bit 0 of the lower byte must be
        // preserved since it is used for temperature compensation calculations.
        // Accelerometer bias registers expect bias input as 2048 LSB per g, so that
        // the accelerometer biases calculated above must be divided by 8.

        // A place to hold the factory accelerometer trim biases
        int32_t accel_bias_reg[3] = {0, 0, 0};
        // Read factory accelerometer trim values
        readBytes(i2c_address, XA_OFFSET_H, 2, &data[0]);
        accel_bias_reg[0] = (int32_t) (((int16_t)data[0] << 8) | data[1]);
        readBytes(i2c_address, YA_OFFSET_H, 2, &data[0]);
        accel_bias_reg[1] = (int32_t) (((int16_t)data[0] << 8) | data[1]);
```

```
    readBytes(i2c_address, ZA_OFFSET_H, 2, &data[0]);
    accel_bias_reg[2] = (int32_t) (((int16_t)data[0] << 8) | data[1]);

    // Define mask for temperature compensation bit 0 of lower byte of
    // accelerometer bias registers
    uint32_t mask = 1uL;
    // Define array to hold mask bit for each accelerometer bias axis
    uint8_t mask_bit[3] = {0, 0, 0};

    for (ii = 0; ii < 3; ii++)
    {
      // If temperature compensation bit is set, record that fact in mask_bit
      if ((accel_bias_reg[ii] & mask))
      {
        mask_bit[ii] = 0x01;
      }
    }

    // Construct total accelerometer bias, including calculated average
    // accelerometer bias from above
    // Subtract calculated averaged accelerometer bias scaled to 2048 LSB/g
    // (16 g full scale)
    accel_bias_reg[0] -= (accel_bias[0]/8);
    accel_bias_reg[1] -= (accel_bias[1]/8);
    accel_bias_reg[2] -= (accel_bias[2]/8);

    data[0] = (accel_bias_reg[0] >> 8) & 0xFF;
    data[1] = (accel_bias_reg[0])      & 0xFF;
    // preserve temperature compensation bit when writing back to accelerometer
    // bias registers
    data[1] = data[1] | mask_bit[0];
    data[2] = (accel_bias_reg[1] >> 8) & 0xFF;
    data[3] = (accel_bias_reg[1])      & 0xFF;
    // Preserve temperature compensation bit when writing back to accelerometer
    // bias registers
    data[3] = data[3] | mask_bit[1];
    data[4] = (accel_bias_reg[2] >> 8) & 0xFF;
    data[5] = (accel_bias_reg[2])      & 0xFF;
    // Preserve temperature compensation bit when writing back to accelerometer
    // bias registers
    data[5] = data[5] | mask_bit[2];

    // Apparently this is not working for the acceleration biases in the MPU-9250
    // Are we handling the temperature correction bit properly?
    // Push accelerometer biases to hardware registers
    writeByte(i2c_address, XA_OFFSET_H, data[0]);
    writeByte(i2c_address, XA_OFFSET_L, data[1]);
    writeByte(i2c_address, YA_OFFSET_H, data[2]);
    writeByte(i2c_address, YA_OFFSET_L, data[3]);
    writeByte(i2c_address, ZA_OFFSET_H, data[4]);
    writeByte(i2c_address, ZA_OFFSET_L, data[5]);

    // Output scaled accelerometer biases for display in the main program
    accelBias[0] = (float)accel_bias[0]/(float)accelsensitivity;
    accelBias[1] = (float)accel_bias[1]/(float)accelsensitivity;
    accelBias[2] = (float)accel_bias[2]/(float)accelsensitivity;

    if(SERIAL_DEBUG){
      for(int i = 0; i < 3; i++){
        Serial.print("accelBias[");
        Serial.print(i);
        Serial.print("] = ");
        Serial.println(accelBias[i]);
      }
    }
  }
  /*
  bool MPU9250::calibrateMPU9250_Gyroscope(){
    uint8_t data[6]; // data array to hold gyro x, y, z, data
    uint16_t ii, packet_count, fifo_count;
    int32_t gyro_bias[3]  = {0, 0, 0};
    int16_t gyro_max[3]  = {0x8000, 0x8000, 0x8000},
            gyro_min[3]  = {0x7FFF, 0x7FFF, 0x7FFF};

    if(SERIAL_DEBUG){
```

```
  Serial.println("calibrateMPU9250_Gyroscope");
}

// Make sure resolution has been calculated
getGres();

// reset device
// Write a one to bit 7 reset bit; toggle reset device
writeByte(i2c_address, PWR_MGMT_1, READ_FLAG);
delay(100);

// get stable time source; Auto select clock source to be PLL gyroscope
// reference if ready else use the internal oscillator, bits 2:0 = 001
writeByte(i2c_address, PWR_MGMT_1, 0x01);
writeByte(i2c_address, PWR_MGMT_2, 0x00);
delay(200);

// Configure device for bias calculation
// Disable all interrupts
writeByte(i2c_address, INT_ENABLE, 0x00);
// Disable FIFO
writeByte(i2c_address, FIFO_EN, 0x00);
// Turn on internal clock source
writeByte(i2c_address, PWR_MGMT_1, 0x00);
// Disable I2C master
writeByte(i2c_address, I2C_MST_CTRL, 0x00);
// Disable FIFO and I2C master modes
writeByte(i2c_address, USER_CTRL, 0x00);
// Reset FIFO and DMP
writeByte(i2c_address, USER_CTRL, 0x0C);
delay(15);

// Configure MPU6050 gyro and accelerometer for bias calculation
// Set low-pass filter to 188 Hz
writeByte(i2c_address, CONFIG, 0x01);
// Set sample rate to 1 kHz
writeByte(i2c_address, SMPLRT_DIV, 0x00);
// Set gyro full-scale to 250 degrees per second, maximum sensitivity
writeByte(i2c_address, GYRO_CONFIG, 0x00);

uint16_t  gyrosensitivity  = 131;   // = 131 LSB/degrees/sec

// Configure FIFO to capture gyro data for bias calculation
writeByte(i2c_address, USER_CTRL, 0x40);  // Enable FIFO
// Enable gyro and accelerometer sensors for FIFO  (max size 512 bytes in
// MPU-9150)

int calibration_is_imcomplete = 1;
int axis = X_AXIS;
float x_confidence = 0.0;
float y_confidence = 0.0;
float z_confidence = 0.0;
float x_bias = 0.0;
float y_bias = 0.0;
float z_bias = 0.0;

// Enable gyro sensor for FIFO
writeByte(i2c_address, FIFO_EN, 0x70);
while(calibration_is_imcomplete){
  delay(80);  // accumulate 80 samples in 80 milliseconds = 480 bytes
  if(EXTRA_GYRO_INFO){
    Serial.print("Max sample reached for ");
    switch(axis){
      case X_AXIS:
        Serial.println("Gyroscope X.");
        break;

      case Y_AXIS:
        Serial.println("Gyroscope Y.");
        break;

      case Z_AXIS:
        Serial.println("Gyroscope Z.");
        break;
    }
```

```
        }

        // Read FIFO sample count
        readBytes(i2c_address, FIFO_COUNTH, 2, &data[0]);
        fifo_count = ((uint16_t)data[0] << 8) | data[1];

        // How many sets of full gyro
        packet_count = fifo_count/6;

        if(packet_count > maxSample){
          packet_count = maxSample;
        }

        if(EXTRA_GYRO_INFO){
          Serial.print("packet_count: ");
          Serial.println(packet_count);
        }

        // Read data
        for (int i = 0; i < packet_count; i++)
        {

          readBytes(i2c_address, FIFO_R_W, 6, &data[0]);

          // Form signed 16-bit integer for each sample in FIFO
          switch(axis){
            case X_AXIS:
              samples[i] = (float) (((int16_t)data[0] << 8) | data[1]  );
              break;

            case Y_AXIS:
              samples[i] = (float) (((int16_t)data[2] << 8) | data[3]  );
              break;

            case Z_AXIS:
              samples[i] = (float) (((int16_t)data[4] << 8) | data[5]  );
              break;
          }
        }

        //Get mean
        float mean = getMean(samples, packet_count);
        if(EXTRA_GYRO_INFO){
          Serial.print("mean: ");
          Serial.println(mean);
        }

        //Get standard deviation
        float standard_deviation = getStandardDeviation(samples, mean, packet_count);
        if(EXTRA_GYRO_INFO){
          Serial.print("standard_deviation: ");
          Serial.println(standard_deviation);
        }

        //Get confidence interval
        float confidence_level = getConfidenceInterval(standard_deviation, packet_count, 1.960);
        if(EXTRA_GYRO_INFO){
          Serial.print("confidence_level: ");
          Serial.println(confidence_level);
        }
        //Check confidence interval before continuing
        //Output results when passing
        if(confidence_level < 5 && confidence_level > 0.0000001){
          switch(axis){
            case X_AXIS:
              x_confidence = confidence_level;
              x_bias = mean;
              axis = Y_AXIS;
              if(EXTRA_GYRO_INFO){
                Serial.print("Gyro X bias: ");
              }
              break;
            case Y_AXIS:
              y_confidence = confidence_level;
              y_bias = mean;
```

```
            axis = Z_AXIS;
            if(EXTRA_GYRO_INFO){
              Serial.print("Gyro Y bias: ");
            }
            break;
          case Z_AXIS:
            z_confidence = confidence_level;
            z_bias = mean;
            calibration_is_imcomplete = false;
            if(EXTRA_GYRO_INFO){
              Serial.print("Gyro Z bias: ");
            }
            break;
        }

        //Output results
        if(EXTRA_GYRO_INFO){
          Serial.print(mean, 5);
          Serial.print(" +- ");
          Serial.println(confidence_level, 5);
        }
      }else{
        if(EXTRA_GYRO_INFO){
          Serial.println("Confidence level insufficient");
        }
      }
    }

    // At end of sample accumulation, turn off FIFO sensor read
    // Disable gyro sensor for FIFO
    writeByte(i2c_address, FIFO_EN, 0x00);



    // Sum individual signed 16-bit biases to get accumulated signed 32-bit biases
    gyro_bias[0]  = (int32_t) x_bias;
    gyro_bias[1]  = (int32_t) y_bias;
    gyro_bias[2]  = (int32_t) z_bias;
    if(EXTRA_GYRO_INFO){
        for(int i = 0; i < 3; i++){
          Serial.print("gyroBias[");
          Serial.print(i);
          Serial.print("] = ");
          Serial.println((float) gyro_bias[i]/(float) gyrosensitivity, 5);
        }
      }
    // Construct the gyro biases for push to the hardware gyro bias registers,
    // which are reset to zero upon device startup.
    // Divide by 4 to get 32.9 LSB per deg/s to conform to expected bias input
    // format.

    // Biases are additive, so change sign on calculated average gyro biases
    //???
    data[0] = (-gyro_bias[0]/4  >> 8) & 0xFF;
    // Biases are additive, so change sign on calculated average gyro biases
    data[1] = (-gyro_bias[0]/4)       & 0xFF;
    data[2] = (-gyro_bias[1]/4  >> 8) & 0xFF;
    data[3] = (-gyro_bias[1]/4)       & 0xFF;
    data[4] = (-gyro_bias[2]/4  >> 8) & 0xFF;
    data[5] = (-gyro_bias[2]/4)       & 0xFF;

    // Push gyro biases to hardware registers
    writeByte(i2c_address, XG_OFFSET_H, data[0]);
    writeByte(i2c_address, XG_OFFSET_L, data[1]);
    writeByte(i2c_address, YG_OFFSET_H, data[2]);
    writeByte(i2c_address, YG_OFFSET_L, data[3]);
    writeByte(i2c_address, ZG_OFFSET_H, data[4]);
    writeByte(i2c_address, ZG_OFFSET_L, data[5]);

    // Output scaled gyro biases for display in the main program
    gyroBias[0] = (float) gyro_bias[0]/(float) gyrosensitivity;
    gyroBias[1] = (float) gyro_bias[1]/(float) gyrosensitivity;
    gyroBias[2] = (float) gyro_bias[2]/(float) gyrosensitivity;
    if(EXTRA_GYRO_INFO){
      for(int i = 0; i < 3; i++){
```

```
        Serial.print("gyroBias[");
        Serial.print(i);
        Serial.print("] = ");
        Serial.println(gyroBias[i], 5);
      }
    }

  }


  */
  bool MPU9250::calibrateMPU9250_Gyroscope(){
    uint8_t data[6]; // data array to hold gyro x, y, z, data
    uint16_t ii, packet_count, fifo_count;
    int32_t gyro_bias[3]  = {0, 0, 0};
    int16_t gyro_max[3]  = {0x8000, 0x8000, 0x8000},
            gyro_min[3]  = {0x7FFF, 0x7FFF, 0x7FFF};

    if(SERIAL_DEBUG){
      Serial.println("calibrateMPU9250_Gyroscope");
    }

    // Make sure resolution has been calculated
    getGres();

    // reset device
    // Write a one to bit 7 reset bit; toggle reset device
    writeByte(i2c_address, PWR_MGMT_1, READ_FLAG);
    delay(100);

    // get stable time source; Auto select clock source to be PLL gyroscope
    // reference if ready else use the internal oscillator, bits 2:0 = 001
    writeByte(i2c_address, PWR_MGMT_1, 0x01);
    writeByte(i2c_address, PWR_MGMT_2, 0x00);
    delay(200);

    // Configure device for bias calculation
    // Disable all interrupts
    writeByte(i2c_address, INT_ENABLE, 0x00);
    // Disable FIFO
    writeByte(i2c_address, FIFO_EN, 0x00);
    // Turn on internal clock source
    writeByte(i2c_address, PWR_MGMT_1, 0x00);
    // Disable I2C master
    writeByte(i2c_address, I2C_MST_CTRL, 0x00);
    // Disable FIFO and I2C master modes
    writeByte(i2c_address, USER_CTRL, 0x00);
    // Reset FIFO and DMP
    writeByte(i2c_address, USER_CTRL, 0x0C);
    delay(15);

    // Configure MPU6050 gyro and accelerometer for bias calculation
    // Set low-pass filter to 188 Hz
    writeByte(i2c_address, CONFIG, 0x01);
    // Set sample rate to 1 kHz
    writeByte(i2c_address, SMPLRT_DIV, 0x00);
    // Set gyro full-scale to 250 degrees per second, maximum sensitivity
    writeByte(i2c_address, GYRO_CONFIG, 0x00);

    uint16_t  gyrosensitivity  = 131;   // = 131 LSB/degrees/sec

    // Configure FIFO to capture gyro data for bias calculation
    writeByte(i2c_address, USER_CTRL, 0x40);  // Enable FIFO
    // Enable gyro and accelerometer sensors for FIFO  (max size 512 bytes in
    // MPU-9150)

    // Enable gyro sensor for FIFO
    writeByte(i2c_address, FIFO_EN, 0x70);
    delay(80);  // accumulate 80 samples in 80 milliseconds = 480 bytes

    // At end of sample accumulation, turn off FIFO sensor read
    // Disable gyro sensor for FIFO
    writeByte(i2c_address, FIFO_EN, 0x00);
    // Read FIFO sample count
    readBytes(i2c_address, FIFO_COUNTH, 2, &data[0]);
```

```
        fifo_count = ((uint16_t)data[0] << 8) | data[1];
        // How many sets of full gyro and accelerometer data for averaging
        packet_count = fifo_count/6;

        packet_count = fifo_count/6;
        if(SERIAL_DEBUG){
          Serial.print("fifo_count: ");
          Serial.println(fifo_count);
          Serial.print("packet_count: ");
          Serial.println(packet_count);
        }
        for (ii = 0; ii < packet_count; ii++)
        {
          int16_t gyro_temp[3] = {0, 0, 0};
          // Read data for averaging
          readBytes(i2c_address, FIFO_R_W, 6, &data[0]);
          // Form signed 16-bit integer for each sample in FIFO
          gyro_temp[0] = (int16_t) (((int16_t)data[0] << 8) | data[1]  );
          gyro_temp[1] = (int16_t) (((int16_t)data[2] << 8) | data[3]  );
          gyro_temp[2] = (int16_t) (((int16_t)data[4] << 8) | data[5]  );

          // Sum individual signed 16-bit biases to get accumulated signed 32-bit
          // biases.
          gyro_bias[0]  += (int32_t) gyro_temp[0];
          gyro_bias[1]  += (int32_t) gyro_temp[1];
          gyro_bias[2]  += (int32_t) gyro_temp[2];

        for (int jj = 0; jj < 3; jj++)
          {
            if (gyro_temp[jj] > gyro_max[jj])
            {
              gyro_max[jj] = gyro_temp[jj];
            }
            if (gyro_temp[jj] < gyro_min[jj])
            {
              gyro_min[jj] = gyro_temp[jj];
            }
          }
        }

        if(SERIAL_DEBUG){
          Serial.println("gyro x min/max:"); Serial.println(gyro_max[0]); Serial.println(gyro_min[0]);
          Serial.println("gyro y min/max:"); Serial.println(gyro_max[1]); Serial.println(gyro_min[1]);
          Serial.println("gyro z min/max:"); Serial.println(gyro_max[2]); Serial.println(gyro_min[2]);
        }

        // Sum individual signed 16-bit biases to get accumulated signed 32-bit biases
        gyro_bias[0]  /= (int32_t) packet_count;
        gyro_bias[1]  /= (int32_t) packet_count;
        gyro_bias[2]  /= (int32_t) packet_count;

        // Construct the gyro biases for push to the hardware gyro bias registers,
        // which are reset to zero upon device startup.
        // Divide by 4 to get 32.9 LSB per deg/s to conform to expected bias input
        // format.
        data[0] = (-gyro_bias[0]/4  >> 8) & 0xFF;
        // Biases are additive, so change sign on calculated average gyro biases
        data[1] = (-gyro_bias[0]/4)       & 0xFF;
        data[2] = (-gyro_bias[1]/4  >> 8) & 0xFF;
        data[3] = (-gyro_bias[1]/4)       & 0xFF;
        data[4] = (-gyro_bias[2]/4  >> 8) & 0xFF;
        data[5] = (-gyro_bias[2]/4)       & 0xFF;

        // Push gyro biases to hardware registers
        writeByte(i2c_address, XG_OFFSET_H, data[0]);
        writeByte(i2c_address, XG_OFFSET_L, data[1]);
        writeByte(i2c_address, YG_OFFSET_H, data[2]);
        writeByte(i2c_address, YG_OFFSET_L, data[3]);
        writeByte(i2c_address, ZG_OFFSET_H, data[4]);
        writeByte(i2c_address, ZG_OFFSET_L, data[5]);

        // Output scaled gyro biases for display in the main program
        gyroBias[0] = (float) gyro_bias[0]/(float) gyrosensitivity;
        gyroBias[1] = (float) gyro_bias[1]/(float) gyrosensitivity;
        gyroBias[2] = (float) gyro_bias[2]/(float) gyrosensitivity;
```

```
    if(SERIAL_DEBUG){
      for(int i = 0; i < 3; i++){
        Serial.print("gyroBias[");
        Serial.print(i);
        Serial.print("] = ");
        Serial.println(gyroBias[i]);
      }
    }
  }

  void MPU9250::initMPU9250()
  {
    // Initialize IMU
    // Reset registers
    writeByte(i2c_address, PWR_MGMT_1, 0x80);
    delay(100);

    // Disable sensors
    // Disabled by reset?
    writeByte(i2c_address, PWR_MGMT_2, 0x1F);
    delay(100);

    // Disable slaves
    writeByte(i2c_address, I2C_SLV0_CTRL, 0x00);
    delay(100);

    // Disable bypass mode
    // 0x22?
    writeByte(i2c_address, INT_PIN_CFG, 0x00);
    delay(100);

    // Get stable time source
    // Auto select clock source to be PLL gyroscope reference if ready else
    writeByte(i2c_address, PWR_MGMT_1, 0x00);
    delay(200);

    // Delays the data ready interrupt until external sensor data is loaded
    // Stop between reads when transitioning from one slave read to the next
    // I2C master clock speed = 400 kHz
    // 8MHz Clock Divider = 20
    writeByte(i2c_address, I2C_MST_CTRL, 0x0D);
    delay(100);

    // Enable I2C master
    writeByte(i2c_address, USER_CTRL, 0x20);
    delay(100);

    // Calibrate gyroscope
    calibrateMPU9250_Gyroscope();
    delay(100);

    // Calibrate accelerometer
    calibrateMPU9250_Accelerometer();
    delay(100);

    // Initialize gyroscope
    initMPU9250_Gyroscope();
    delay(100);

    // Temp to ensure regs are still set...
    // Initialize accelerometer
    initMPU9250_Accelerometer();
    delay(100);

    // Disable bypass mode
    // 0x22?
    writeByte(i2c_address, INT_PIN_CFG, 0x00);
    delay(100);

    // Delays the data ready interrupt until external sensor data is loaded
    // Stop between reads when transitioning from one slave read to the next
    // I2C master clock speed = 400 kHz
    // 8MHz Clock Divider = 20
    writeByte(i2c_address, I2C_MST_CTRL, 0x0D);
    delay(100);
```

```
  // Enable I2C master
  writeByte(i2c_address, USER_CTRL, 0x20);
  delay(100);

  // Initialize magnetometer
  initMPU9250_Magnetometer();
  delay(100);

  // Calibrate magnetometer
  calibrateMPU9250_Magnetometer();
  delay(100);
}


bool MPU9250::read_Magnetometer()
{
  // x/y/z gyro register data, ST2 register stored here, must read ST2 at end
  // of data acquisition

  uint8_t rawData[7];

  // Read the six raw data and ST2 registers sequentially into data array
  readBytes(i2c_address, EXT_SENS_DATA_00, 7, &rawData[0]);
  //delay(delay_1);
  uint8_t c = rawData[6]; // End data read by reading ST2 register
  // Check if magnetic sensor overflow set, if not then report data
  if (!(c & 0x08))
  {
    // Turn the MSB and LSB into a signed 16-bit value
    // Might want to filter out erroneous values
    magCount[0] = ((int16_t)rawData[1] << 8) | rawData[0];
    magCount[1] = ((int16_t)rawData[3] << 8) | rawData[2];
    magCount[2] = ((int16_t)rawData[5] << 8) | rawData[4];
  }

  //TODO: update on read success
  return true;
}

bool MPU9250::read_Accelerometer(){
  uint8_t rawData[6];  // x/y/z accel register data stored here
  // Read the six raw data registers into data array
  readBytes(i2c_address, ACCEL_XOUT_H, 6, &rawData[0]);

  // Turn the MSB and LSB into a signed 16-bit value
  accelCount[0] = ((int16_t)rawData[0] << 8) | rawData[1] ;
  accelCount[1] = ((int16_t)rawData[2] << 8) | rawData[3] ;
  accelCount[2] = ((int16_t)rawData[4] << 8) | rawData[5] ;

  //TODO: update on read success
  return true;
}

bool MPU9250::read_Gyroscope(){
  uint8_t rawData[6];  // x/y/z gyro register data stored here
  // Read the six raw data registers sequentially into data array
  readBytes(i2c_address, GYRO_XOUT_H, 6, &rawData[0]);

  // Turn the MSB and LSB into a signed 16-bit value
  gyroCount[0] = ((int16_t)rawData[0] << 8) | rawData[1] ;
  gyroCount[1] = ((int16_t)rawData[2] << 8) | rawData[3] ;
  gyroCount[2] = ((int16_t)rawData[4] << 8) | rawData[5] ;

  //TODO: update on read success
  return true;
}


void MPU9250::updateQuaternion(){
  q_m.update(ax, ay, az, gx, gy, gz, mx, my, mz);
}

void MPU9250::retrieve_data(){
  // Read data from each sensor
```

```
  read_Accelerometer();
  read_Gyroscope();
  read_Magnetometer();

  // Retrieve the modified data
  // Accelerometer
  ax = (float)accelCount[0] * aRes;
  ay = (float)accelCount[1] * aRes;
  az = (float)accelCount[2] * aRes;

  // Gyroscope
  gx = (float)gyroCount[0] * gRes;
  gy = (float)gyroCount[1] * gRes;
  gz = (float)gyroCount[2] * gRes;

  // Magnetometer
  mx = (float)magCount[0] * mRes;// * factoryMagCalibration[0] - magBias[0];
  my = (float)magCount[1] * mRes;// * factoryMagCalibration[1] - magBias[1];
  mz = (float)magCount[2] * mRes;// * factoryMagCalibration[2] - magBias[2];

  /*
  float ratio_acc = 4.0f/32767.0f;
  float ratio_gyro = (1000.0f/32767.0f) * (PI / 180.0f);
  float ratio_mag = 48.0f / 32767.0f;

  ax = (float)accelCount[0] * ratio_acc;
  ay = (float)accelCount[1] * ratio_acc;
  az = (float)accelCount[2] * ratio_acc;

  // Gyroscope
  gx = ((float)gyroCount[0] - 48.4827) * ratio_gyro;
  gy = ((float)gyroCount[1] + 76.3552) * ratio_gyro;
  gz = ((float)gyroCount[2] + 64.3234) * ratio_gyro;

  // Magnetometer
  mx = (float)magCount[0] * ratio_mag;
  my = (float)magCount[1] * ratio_mag;
  mz = (float)magCount[2] * ratio_mag;
  */
}

// Wire.h read and write protocols
uint8_t MPU9250::writeByte(uint8_t deviceAddress, uint8_t registerAddress,
                           uint8_t data)
{
  Wire.beginTransmission(deviceAddress);  // Initialize the Tx buffer
  Wire.write(registerAddress);       // Put slave register address in Tx buffer
  Wire.write(data);                  // Put data in Tx buffer
  Wire.endTransmission();            // Send the Tx buffer
  // TODO: Fix this to return something meaningful
  return 0;
}

// Read a byte from the given register address from device using I2C
uint8_t MPU9250::readByte(uint8_t deviceAddress, uint8_t registerAddress)
{
  uint8_t data; // `data` will store the register data

  // Initialize the Tx buffer
  Wire.beginTransmission(deviceAddress);
  // Put slave register address in Tx buffer
  Wire.write(registerAddress);
  // Send the Tx buffer, but send a restart to keep connection alive
  Wire.endTransmission(false);
  // Read one byte from slave register address
  Wire.requestFrom(deviceAddress, (uint8_t) 1);
  // Fill Rx buffer with result
  data = Wire.read();
  // Return data read from slave register
  return data;
}

// Read 1 or more bytes from given register and device using I2C
uint8_t MPU9250::readBytes(uint8_t deviceAddress, uint8_t registerAddress,
                           uint8_t count, uint8_t * dest)
```

```
{
  // Initialize the Tx buffer
  Wire.beginTransmission(deviceAddress);
  // Put slave register address in Tx buffer
  Wire.write(registerAddress);
  // Send the Tx buffer, but send a restart to keep connection alive
  Wire.endTransmission(false);

  uint8_t i = 0;
  // Read bytes from slave register address
  Wire.requestFrom(deviceAddress, count);
  while (Wire.available())
  {
    // Put read results in the Rx buffer
    dest[i++] = Wire.read();
  }

  return i; // Return number of bytes written
}
```

```
/*

        quaternion.h

*/

#ifndef _QUATERNION_H_
#define _QUATERNION_H_

class Quaternion {
        public:
                float q1 = 1.0f;
                float q2 = 0.0f;
                float q3 = 0.0f;
                float q4 = 0.0f;
};

#endif
```

```
/*

        quaternion_module.cpp

*/

#ifndef _QUATERNION_MODULE_H_
#define _QUATERNION_MODULE_H_

class Quaternion_Module
{
        public:
                // Vector to hold integral error for Mahony method
            float eInt[3] = {0.0f, 0.0f, 0.0f};
            Quaternion *q;

            // used to calculate integration interval
            float deltat = 0.0f;
        uint32_t lastUpdate = 0;
        uint32_t now = 0;
                void update(float ax, float ay, float az, float gx, float gy, float gz, float mx,
                     float my, float mz);
                Quaternion_Module();
};

#endif
```

```
/*

        quaternion_module.cpp

*/

#include "stdafx.h"

// Sensors x (y)-axis of the accelerometer is aligned with the y (x)-axis of
```

```
        // the magnetometer; the magnetometer z-axis (+ down) is opposite to z-axis
        // (+ up) of accelerometer and gyro! We have to make some allowance for this
        // orientationmismatch in feeding the output to the quaternion filter. For the
        // MPU-9250, we have chosen a magnetic rotation that keeps the sensor forward
        // along the x-axis just like in the LSM9DS0 sensor. This rotation can be
        // modified to allow any convenient orientation convention. This is ok by
        // aircraft orientation standards! Pass gyro rate as rad/s
        void Quaternion_Module::update(float ax, float ay, float az, float gx, float gy, float gz, float
            mx, float my, float mz){
                // Get current time
                now = micros();

                // Set integration time by time elapsed since last filter update
                deltat = ((now - lastUpdate) / 1000000.0f);
                lastUpdate = now;



                //MahonyQuaternionUpdate(q, eInt, ax, ay, az, gx*DEG_TO_RAD, gy*DEG_TO_RAD, gz*DEG_TO_RAD
                    , my, mx, mz, deltat);

                //MadgwickQuaternionUpdate(q, ax, ay, az, gx*DEG_TO_RAD, gy*DEG_TO_RAD, gz*DEG_TO_RAD, my
                    , mx, mz, deltat);
                MadgwickQuaternionUpdate_v2(q, ax, ay, az, gx*DEG_TO_RAD, gy*DEG_TO_RAD, gz*DEG_TO_RAD,
                    deltat);
        }

        Quaternion_Module::Quaternion_Module(){
                q = new Quaternion();
        }
```

```
        /*

                quaternionFilter.h

        */

        #ifndef _QUATERNIONFILTERS_H_
        #define _QUATERNIONFILTERS_H_

        #define sampleFreq 512.0f    // sample frequency in Hz
        #define betaDef   0.1f    // 2 * proportional gain
        //
            ---------------------------------------------------------------------------------------------


        // Variable definitions

        //volatile float beta = betaDef;                // 2 * proportional gain (Kp)
        //volatile float q0 = 1.0f, q1 = 0.0f, q2 = 0.0f, q3 = 0.0f;  // quaternion of sensor frame
            relative to auxiliary frame

        // These are the free parameters in the Mahony filter and fusion scheme, Kp
        // for proportional feedback, Ki for integral
        #define Kp 2.0f * 5.0f
        #define Ki 0.0f

        static float GyroMeasError = PI * (40.0f / 180.0f);
        // gyroscope measurement drift in rad/s/s (start at 0.0 deg/s/s)
        static float GyroMeasDrift = PI * (0.0f  / 180.0f);
        // There is a tradeoff in the beta parameter between accuracy and response
        // speed. In the original Madgwick study, beta of 0.041 (corresponding to
        // GyroMeasError of 2.7 degrees/s) was found to give optimal accuracy.
        // However, with this value, the LSM9SD0 response time is about 10 seconds
        // to a stable initial quaternion. Subsequent changes also require a
        // longish lag time to a stable output, not fast enough for a quadcopter or
        // robot car! By increasing beta (GyroMeasError) by about a factor of
        // fifteen, the response time constant is reduced to ~2 sec. I haven't
        // noticed any reduction in solution accuracy. This is essentially the I
        // coefficient in a PID control sense; the bigger the feedback coefficient,
        // the faster the solution converges, usually at the expense of accuracy.
        // In any case, this is the free parameter in the Madgwick filtering and
        // fusion scheme.
        static float beta = sqrt(3.0f / 4.0f) * GyroMeasError;   // Compute beta
        // Compute zeta, the other free parameter in the Madgwick scheme usually
        // set to a small or zero value
        static float zeta = sqrt(3.0f / 4.0f) * GyroMeasDrift;
```

```
//---------------------------------------------------------------------------------------------------

// Function declarations

float invSqrt(float x);


void MadgwickQuaternionUpdate(Quaternion *q, float ax, float ay, float az, float gx, float gy,
                              float gz, float mx, float my, float mz,
                              float deltat);
void MahonyQuaternionUpdate(Quaternion *q, float eInt[], float ax, float ay, float az, float gx,
    float gy,
                            float gz, float mx, float my, float mz,
                            float deltat);

void MadgwickQuaternionUpdate_v2(Quaternion *q, float ax, float ay, float az, float gx, float gy,
     float gz, float deltat);

#endif
```

```
/*

        quaternionFilter.cpp

*/

// Implementation of Sebastian Madgwick's "...efficient orientation filter
// for... inertial/magnetic sensor arrays"
// (see http://www.x-io.co.uk/category/open-source/ for examples & more details)
// which fuses acceleration, rotation rate, and magnetic moments to produce a
// quaternion-based estimate of absolute device orientation -- which can be
// converted to yaw, pitch, and roll. Useful for stabilizing quadcopters, etc.
// The performance of the orientation filter is at least as good as conventional
// Kalman-based filtering algorithms but is much less computationally
// intensive---it can be performed on a 3.3 V Pro Mini operating at 8 MHz!

#include "stdafx.h"

void MadgwickQuaternionUpdate(Quaternion *q, float ax, float ay, float az, float gx, float gy,
    float gz, float mx, float my, float mz, float deltat)
{
  // short name local variable for readability
  float q1 = q->q1;
  float q2 = q->q2;
  float q3 = q->q3;
  float q4 = q->q4;

  /*
  float norm;
  float hx, hy, _2bx, _2bz;
  float s1, s2, s3, s4;
  float qDot1, qDot2, qDot3, qDot4;

  // Auxiliary variables to avoid repeated arithmetic
  float _2q1mx;
  float _2q1my;
  float _2q1mz;
  float _2q2mx;
  float _4bx;
  float _4bz;
  float _2q1 = 2.0f * q1;
  float _2q2 = 2.0f * q2;
  float _2q3 = 2.0f * q3;
  float _2q4 = 2.0f * q4;
  float _2q1q3 = 2.0f * q1 * q3;
  float _2q3q4 = 2.0f * q3 * q4;
  float q1q1 = q1 * q1;
  float q1q2 = q1 * q2;
  float q1q3 = q1 * q3;
  float q1q4 = q1 * q4;
  float q2q2 = q2 * q2;
  float q2q3 = q2 * q3;
  float q2q4 = q2 * q4;
  float q3q3 = q3 * q3;
```

```
        float q3q4 = q3 * q4;
        float q4q4 = q4 * q4;

        // Normalise accelerometer measurement
        norm = sqrtf(ax * ax + ay * ay + az * az);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f/norm;
        ax *= norm;
        ay *= norm;
        az *= norm;

        // Normalise magnetometer measurement
        norm = sqrtf(mx * mx + my * my + mz * mz);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f/norm;
        mx *= norm;
        my *= norm;
        mz *= norm;

        // Reference direction of Earth's magnetic field
        _2q1mx = 2.0f * q1 * mx;
        _2q1my = 2.0f * q1 * my;
        _2q1mz = 2.0f * q1 * mz;
        _2q2mx = 2.0f * q2 * mx;
        hx = mx * q1q1 - _2q1my * q4 + _2q1mz * q3 + mx * q2q2 + _2q2 * my * q3 + _2q2 * mz * q4 - mx *
             q3q3 - mx * q4q4;
        hy = _2q1mx * q4 + my * q1q1 - _2q1mz * q2 + _2q2mx * q3 - my * q2q2 + my * q3q3 + _2q3 * mz *
            q4 - my * q4q4;
        _2bx = sqrtf(hx * hx + hy * hy);
        _2bz = -_2q1mx * q3 + _2q1my * q2 + mz * q1q1 + _2q2mx * q4 - mz * q2q2 + _2q3 * my * q4 - mz *
             q3q3 + mz * q4q4;
        _4bx = 2.0f * _2bx;
        _4bz = 2.0f * _2bz;

        // Gradient decent algorithm corrective step
        s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax) + _2q2 * (2.0f * q1q2 + _2q3q4 - ay) - _2bz * q3 * (
            _2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q4 + _2bz * q2) * (_2bx
             * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q3 * (_2bx * (q1q3 + q2q4) + _2bz *
            (0.5f - q2q2 - q3q3) - mz);
        s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) + _2q1 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q2 * (1.0
            f - 2.0f * q2q2 - 2.0f * q3q3 - az) + _2bz * q4 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (
            q2q4 - q1q3) - mx) + (_2bx * q3 + _2bz * q1) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4)
             - my) + (_2bx * q4 - _4bz * q2) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz
            );
        s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax) + _2q4 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q3 *
            (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + (-_4bx * q3 - _2bz * q1) * (_2bx * (0.5f - q3q3 -
             q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q2 + _2bz * q4) * (_2bx * (q2q3 - q1q4) +
            _2bz * (q1q2 + q3q4) - my) + (_2bx * q1 - _4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f
             - q2q2 - q3q3) - mz);
        s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) + _2q3 * (2.0f * q1q2 + _2q3q4 - ay) + (-_4bx * q4 +
            _2bz * q2) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz
             * q3) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q2 * (_2bx * (q1q3 +
            q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
        norm = sqrtf(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);    // normalise step magnitude
        norm = 1.0f/norm;
        s1 *= norm;
        s2 *= norm;
        s3 *= norm;
        s4 *= norm;

        // Compute rate of change of quaternion
        qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
        qDot2 = 0.5f * (q1 * gx + q3 * gz - q4 * gy) - beta * s2;
        qDot3 = 0.5f * (q1 * gy - q2 * gz + q4 * gx) - beta * s3;
        qDot4 = 0.5f * (q1 * gz + q2 * gy - q3 * gx) - beta * s4;

        // Integrate to yield quaternion
        q1 += qDot1 * deltat;
        q2 += qDot2 * deltat;
        q3 += qDot3 * deltat;
        q4 += qDot4 * deltat;
        norm = sqrtf(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise quaternion
        norm = 1.0f/norm;
        */
        // short name local variable for readability
```

```
        float norm;
        float hx, hy, _2bx, _2bz;
        float s1, s2, s3, s4;
        float qDot1, qDot2, qDot3, qDot4;

        // Auxiliary variables to avoid repeated arithmetic
        float _2q1mx;
        float _2q1my;
        float _2q1mz;
        float _2q2mx;
        float _4bx;
        float _4bz;
        float _2q1 = 2.0f * q1;
        float _2q2 = 2.0f * q2;
        float _2q3 = 2.0f * q3;
        float _2q4 = 2.0f * q4;
        float _2q1q3 = 2.0f * q1 * q3;
        float _2q3q4 = 2.0f * q3 * q4;
        float q1q1 = q1 * q1;
        float q1q2 = q1 * q2;
        float q1q3 = q1 * q3;
        float q1q4 = q1 * q4;
        float q2q2 = q2 * q2;
        float q2q3 = q2 * q3;
        float q2q4 = q2 * q4;
        float q3q3 = q3 * q3;
        float q3q4 = q3 * q4;
        float q4q4 = q4 * q4;

        // Normalise accelerometer measurement
        norm = sqrt(ax * ax + ay * ay + az * az);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f/norm;
        ax *= norm;
        ay *= norm;
        az *= norm;

        // Normalise magnetometer measurement
        norm = sqrt(mx * mx + my * my + mz * mz);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f/norm;
        mx *= norm;
        my *= norm;
        mz *= norm;

        // Reference direction of Earth's magnetic field
        _2q1mx = 2.0f * q1 * mx;
        _2q1my = 2.0f * q1 * my;
        _2q1mz = 2.0f * q1 * mz;
        _2q2mx = 2.0f * q2 * mx;
        hx = mx * q1q1 - _2q1my * q4 + _2q1mz * q3 + mx * q2q2 + _2q2 * my * q3 +
             _2q2 * mz * q4 - mx * q3q3 - mx * q4q4;
        hy = _2q1mx * q4 + my * q1q1 - _2q1mz * q2 + _2q2mx * q3 - my * q2q2 + my * q3q3 + _2q3 * mz *
             q4 - my * q4q4;
        _2bx = sqrt(hx * hx + hy * hy);
        _2bz = -_2q1mx * q3 + _2q1my * q2 + mz * q1q1 + _2q2mx * q4 - mz * q2q2 + _2q3 * my * q4 - mz *
             q3q3 + mz * q4q4;
        _4bx = 2.0f * _2bx;
        _4bz = 2.0f * _2bz;

        // Gradient decent algorithm corrective step
        s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax) + _2q2 * (2.0f * q1q2 + _2q3q4 - ay) - _2bz * q3 * (
             _2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q4 + _2bz * q2) * (_2bx
              * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q3 * (_2bx * (q1q3 + q2q4) + _2bz *
             (0.5f - q2q2 - q3q3) - mz);
        s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) + _2q1 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q2 * (1.0
             f - 2.0f * q2q2 - 2.0f * q3q3 - az) + _2bz * q4 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (
             q2q4 - q1q3) - mx) + (_2bx * q3 + _2bz * q1) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4)
              - my) + (_2bx * q4 - _4bz * q2) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz
             );
        s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax) + _2q4 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q3 *
             (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + (-_4bx * q3 - _2bz * q1) * (_2bx * (0.5f - q3q3 -
              q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q2 + _2bz * q4) * (_2bx * (q2q3 - q1q4) +
             _2bz * (q1q2 + q3q4) - my) + (_2bx * q1 - _4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f
              - q2q2 - q3q3) - mz);
```

```
    s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) + _2q3 * (2.0f * q1q2 + _2q3q4 - ay) + (-_4bx * q4 +
        _2bz * q2) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz
         * q3) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q2 * (_2bx * (q1q3 +
        q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
    norm = sqrt(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);     // normalise step magnitude
    norm = 1.0f/norm;
    s1 *= norm;
    s2 *= norm;
    s3 *= norm;
    s4 *= norm;

    // Compute rate of change of quaternion
    qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
    qDot2 = 0.5f * (q1 * gx + q3 * gz - q4 * gy) - beta * s2;
    qDot3 = 0.5f * (q1 * gy - q2 * gz + q4 * gx) - beta * s3;
    qDot4 = 0.5f * (q1 * gz + q2 * gy - q3 * gx) - beta * s4;

    // Integrate to yield quaternion
    q1 += qDot1 * deltat;
    q2 += qDot2 * deltat;
    q3 += qDot3 * deltat;
    q4 += qDot4 * deltat;
    norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);     // normalise quaternion
    norm = 1.0f/norm;

    q -> q1 = q1 * norm;
    q -> q2 = q2 * norm;
    q -> q3 = q3 * norm;
    q -> q4 = q4 * norm;
}

// Similar to Madgwick scheme but uses proportional and integral filtering on
// the error between estimated reference vectors and measured ones.
void MahonyQuaternionUpdate(Quaternion *q, float eInt[], float ax, float ay, float az, float gx,
    float gy, float gz, float mx, float my, float mz, float deltat)
{
    // short name local variable for readability
    float q1 = q -> q1;
    float q2 = q -> q2;
    float q3 = q -> q3;
    float q4 = q -> q4;
    float norm;
    float hx, hy, bx, bz;
    float vx, vy, vz, wx, wy, wz;
    float ex, ey, ez;
    float pa, pb, pc;

    // Auxiliary variables to avoid repeated arithmetic
    float q1q1 = q1 * q1;
    float q1q2 = q1 * q2;
    float q1q3 = q1 * q3;
    float q1q4 = q1 * q4;
    float q2q2 = q2 * q2;
    float q2q3 = q2 * q3;
    float q2q4 = q2 * q4;
    float q3q3 = q3 * q3;
    float q3q4 = q3 * q4;
    float q4q4 = q4 * q4;

    // Normalise accelerometer measurement
    norm = sqrtf(ax * ax + ay * ay + az * az);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f / norm;        // use reciprocal for division
    ax *= norm;
    ay *= norm;
    az *= norm;

    // Normalise magnetometer measurement
    norm = sqrtf(mx * mx + my * my + mz * mz);
    if (norm == 0.0f) return; // handle NaN
    norm = 1.0f / norm;        // use reciprocal for division
    mx *= norm;
    my *= norm;
    mz *= norm;
```

```
    // Reference direction of Earth's magnetic field
    hx = 2.0f * mx * (0.5f - q3q3 - q4q4) + 2.0f * my * (q2q3 - q1q4) + 2.0f * mz * (q2q4 + q1q3);
    hy = 2.0f * mx * (q2q3 + q1q4) + 2.0f * my * (0.5f - q2q2 - q4q4) + 2.0f * mz * (q3q4 - q1q2);
    bx = sqrtf((hx * hx) + (hy * hy));
    bz = 2.0f * mx * (q2q4 - q1q3) + 2.0f * my * (q3q4 + q1q2) + 2.0f * mz * (0.5f - q2q2 - q3q3);

    // Estimated direction of gravity and magnetic field
    vx = 2.0f * (q2q4 - q1q3);
    vy = 2.0f * (q1q2 + q3q4);
    vz = q1q1 - q2q2 - q3q3 + q4q4;
    wx = 2.0f * bx * (0.5f - q3q3 - q4q4) + 2.0f * bz * (q2q4 - q1q3);
    wy = 2.0f * bx * (q2q3 - q1q4) + 2.0f * bz * (q1q2 + q3q4);
    wz = 2.0f * bx * (q1q3 + q2q4) + 2.0f * bz * (0.5f - q2q2 - q3q3);

    // Error is cross product between estimated direction and measured direction of gravity
    ex = (ay * vz - az * vy) + (my * wz - mz * wy);
    ey = (az * vx - ax * vz) + (mz * wx - mx * wz);
    ez = (ax * vy - ay * vx) + (mx * wy - my * wx);
    if (Ki > 0.0f)
    {
        eInt[0] += ex;      // accumulate integral error
        eInt[1] += ey;
        eInt[2] += ez;
    }
    else
    {
        eInt[0] = 0.0f;     // prevent integral wind up
        eInt[1] = 0.0f;
        eInt[2] = 0.0f;
    }

    // Apply feedback terms
    gx = gx + Kp * ex + Ki * eInt[0];
    gy = gy + Kp * ey + Ki * eInt[1];
    gz = gz + Kp * ez + Ki * eInt[2];

    // Integrate rate of change of quaternion
    pa = q2;
    pb = q3;
    pc = q4;
    q1 = q1 + (-q2 * gx - q3 * gy - q4 * gz) * (0.5f * deltat);
    q2 = pa + (q1 * gx + pb * gz - pc * gy) * (0.5f * deltat);
    q3 = pb + (q1 * gy - pa * gz + pc * gx) * (0.5f * deltat);
    q4 = pc + (q1 * gz + pa * gy - pb * gx) * (0.5f * deltat);

    // Normalise quaternion
    norm = sqrtf(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);
    norm = 1.0f / norm;
    q -> q1 = q1 * norm;
    q -> q2 = q2 * norm;
    q -> q3 = q3 * norm;
    q -> q4 = q4 * norm;
}

//---------------------------------------------------------------------------------------------------

// Fast inverse square-root
// See: http://en.wikipedia.org/wiki/Fast_inverse_square_root

float invSqrt(float x) {
    float halfx = 0.5f * x;
    float y = x;
    long i = *(long*)&y;
    i = 0x5f3759df - (i>>1);
    y = *(float*)&i;
    y = y * (1.5f - (halfx * y * y));
    return y;
}


void MadgwickQuaternionUpdate_v2(Quaternion *q, float ax, float ay, float az, float gx, float gy,
    float gz, float deltat)
{
    float q1 = q -> q1;
```

```
        float q2 = q -> q2;
        float q3 = q -> q3;
        float q4 = q -> q4;

        float norm;                                                 // vector norm
        float f1, f2, f3;                                           // objetive funcyion elements
        float J_11or24, J_12or23, J_13or22, J_14or21, J_32, J_33; // objective function Jacobian
            elements
        float qDot1, qDot2, qDot3, qDot4;
        float hatDot1, hatDot2, hatDot3, hatDot4;
        float gerrx, gerry, gerrz, gbiasx, gbiasy, gbiasz;          // gyro bias error

        // Auxiliary variables to avoid repeated arithmetic
        float _halfq1 = 0.5f * q1;
        float _halfq2 = 0.5f * q2;
        float _halfq3 = 0.5f * q3;
        float _halfq4 = 0.5f * q4;
        float _2q1 = 2.0f * q1;
        float _2q2 = 2.0f * q2;
        float _2q3 = 2.0f * q3;
        float _2q4 = 2.0f * q4;
        float _2q1q3 = 2.0f * q1 * q3;
        float _2q3q4 = 2.0f * q3 * q4;

        // Normalise accelerometer measurement
        norm = sqrt(ax * ax + ay * ay + az * az);
        if (norm == 0.0f) return; // handle NaN
        norm = 1.0f/norm;
        ax *= norm;
        ay *= norm;
        az *= norm;

        // Compute the objective function and Jacobian
        f1 = _2q2 * q4 - _2q1 * q3 - ax;
        f2 = _2q1 * q2 + _2q3 * q4 - ay;
        f3 = 1.0f - _2q2 * q2 - _2q3 * q3 - az;
        J_11or24 = _2q3;
        J_12or23 = _2q4;
        J_13or22 = _2q1;
        J_14or21 = _2q2;
        J_32 = 2.0f * J_14or21;
        J_33 = 2.0f * J_11or24;

        // Compute the gradient (matrix multiplication)
        hatDot1 = J_14or21 * f2 - J_11or24 * f1;
        hatDot2 = J_12or23 * f1 + J_13or22 * f2 - J_32 * f3;
        hatDot3 = J_12or23 * f2 - J_33 *f3 - J_13or22 * f1;
        hatDot4 = J_14or21 * f1 + J_11or24 * f2;

        // Normalize the gradient
        norm = sqrt(hatDot1 * hatDot1 + hatDot2 * hatDot2 + hatDot3 * hatDot3 + hatDot4 * hatDot4);
        hatDot1 /= norm;
        hatDot2 /= norm;
        hatDot3 /= norm;
        hatDot4 /= norm;

        // Compute estimated gyroscope biases
        gerrx = _2q1 * hatDot2 - _2q2 * hatDot1 - _2q3 * hatDot4 + _2q4 * hatDot3;
        gerry = _2q1 * hatDot3 + _2q2 * hatDot4 - _2q3 * hatDot1 - _2q4 * hatDot2;
        gerrz = _2q1 * hatDot4 - _2q2 * hatDot3 + _2q3 * hatDot2 - _2q4 * hatDot1;

        // Compute and remove gyroscope biases
        gbiasx += gerrx * deltat * zeta;
        gbiasy += gerry * deltat * zeta;
        gbiasz += gerrz * deltat * zeta;
        gx -= gbiasx;
        gy -= gbiasy;
        gz -= gbiasz;

        // Compute the quaternion derivative
        qDot1 = -_halfq2 * gx - _halfq3 * gy - _halfq4 * gz;
        qDot2 =  _halfq1 * gx + _halfq3 * gz - _halfq4 * gy;
        qDot3 =  _halfq1 * gy - _halfq2 * gz + _halfq4 * gx;
        qDot4 =  _halfq1 * gz + _halfq2 * gy - _halfq3 * gx;
```

```
    // Compute then integrate estimated quaternion derivative
    q1 += (qDot1 -(beta * hatDot1)) * deltat;
    q2 += (qDot2 -(beta * hatDot2)) * deltat;
    q3 += (qDot3 -(beta * hatDot3)) * deltat;
    q4 += (qDot4 -(beta * hatDot4)) * deltat;

    // Normalize the quaternion
    norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise quaternion
    norm = 1.0f/norm;

    q -> q1 = q1 * norm;
    q -> q2 = q2 * norm;
    q -> q3 = q3 * norm;
    q -> q4 = q4 * norm;
}
```

```
/*

        stdafx.h

*/

#ifndef _STDAFX_H_
#define _STDAFX_H_
/*
        File: stdafx.h

        include file for standard system include files,
        or project specific include files that are used frequently, but
        are changed infrequently
*/

#include <stdint.h>
#include <Arduino.h>
#include <Wire.h>

// TODO: reference additional headers your program requires here
#include "debug_config.h"
#include "quaternion.h"
#include "data_utils.h"
#include "quaternionFilters.h"
#include "quaternion_module.h"
#include "MPU9250.h"

#endif
```

```
/*
        File: stdafx.cpp

        source file that includes just the standard includes
*/
#include "stdafx.h"
```