Flight Director Embedded System and Mobile iOS Application

A Senior Project Report

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

By

Anthony Epshteyn

June 2019

Table of Contents

Abstract

For my senior project, I was asked to assist an Aerospace Engineering professor with the design of his new glider. He needed to create a flight director-type instrument so that his pilot could get the aircraft's positional data during flight. This positional data came from a powerful sensor mounted on the fuselage.

To interface with the sensor, I created an embedded system comprised of a ESP32 micro-controller communicating with the sensor via UART/RS-232. The micro-controller was mounted on a breadboard and connected to the sensor via jumper wires. The ESP32 featured a Bluetooth chip that allowed for communication using the Bluetooth Low Energy protocol. Using this chip, I was able to pass the sensor data to an iPhone application that would serve as a GUI display. I was also responsible for developing this application, which read the sensor data from the ESP32 and displayed a user interface that was similar to flight directors used in industry. With this app installed, the pilot would be able to mount his/her phone on the dashboard of the glider and see the orientation of the aircraft. A preliminary version of this app was created, but now must be tuned for precision and to fit the exact needs of the pilot. Flight testing will have to be done when the glider is ready so that the pilot can give feedback on the adjustments necessary to the UI.

Introduction

Background

In the field of aviation, one of the instruments commonly used by pilots is called a flight director, seen in Figure 1. Its function is to display the proper pitch and bank angles that are required for an aircraft to follow a specific flight path, along with providing any GPS data that the pilot needs. After setting a particular destination, the pilot is able to follow the visual cues given by the flight director with control inputs to keep the aircraft on course. Without a flight director, the pilot is forced to make control decisions based on synthesizing information from the attitude indicator (pitch/roll data) and navigation instrument (heading/altitude data), greatly increasing the workload. While a flight director is a common instrument used on large commercial or cargo planes, many smaller aircraft and gliders do not include this tool on their dashboard.

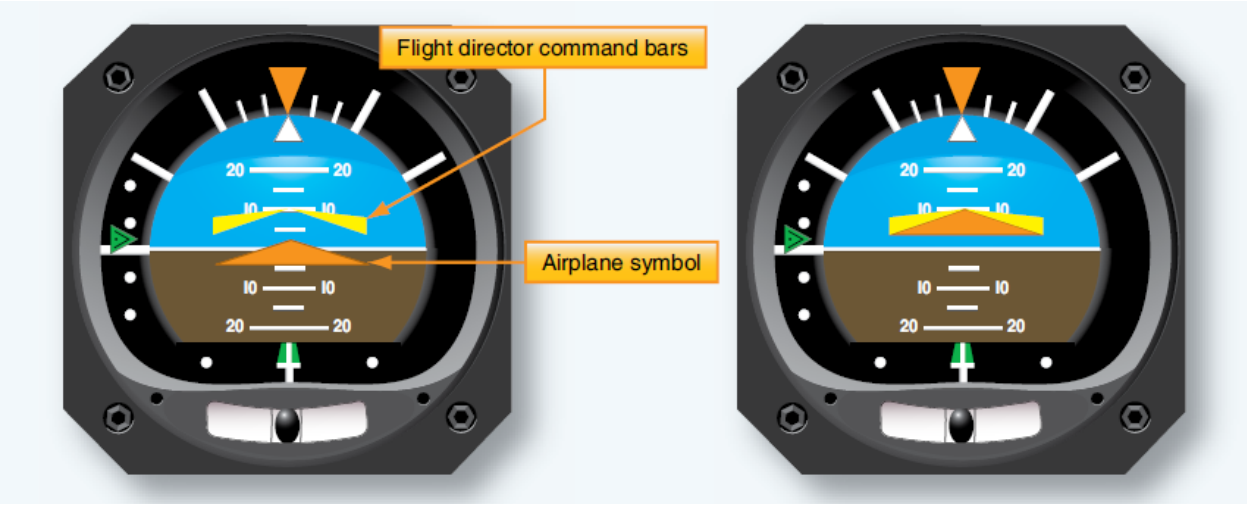

Figure 1 - Example of a Simple Flight Director [1]

A flight director works by reading data from a combination of two sensors: a global positioning system (GPS) sensor and an internal measurement unit (IMU). The GPS sensor is responsible for determining the aircraft's coordinate position, as well as its altitude above sea level. The IMU is an electronic device capable of measuring and reporting its angular rate, as well as the specific force and magnetic field acting on it, by using a combination of accelerometers, gyroscopes, and magnetometers. When properly mounted on the body of an aircraft, the IMU is able to compute an aircraft's principal axis (Figure 2) and Euler angles (pitch, roll and yaw) defined below.

Vertical axis: Originates at the center of gravity and extends towards the bottom of the fuselage, perpendicular to the wings. Motion in this axis is called yaw.
Transverse axis: Originates at the center of gravity and extends to right of the fuselage, perpendicular to the line connecting wingtip to wingtip. Motion in this axis is called pitch.
Longitudinal axis: Originates at the center of gravity and extends forward, perpendicular to the line drawn from tip to tail of the fuselage. Motion in this axis is called roll, while angular displacement in this axis is called bank.

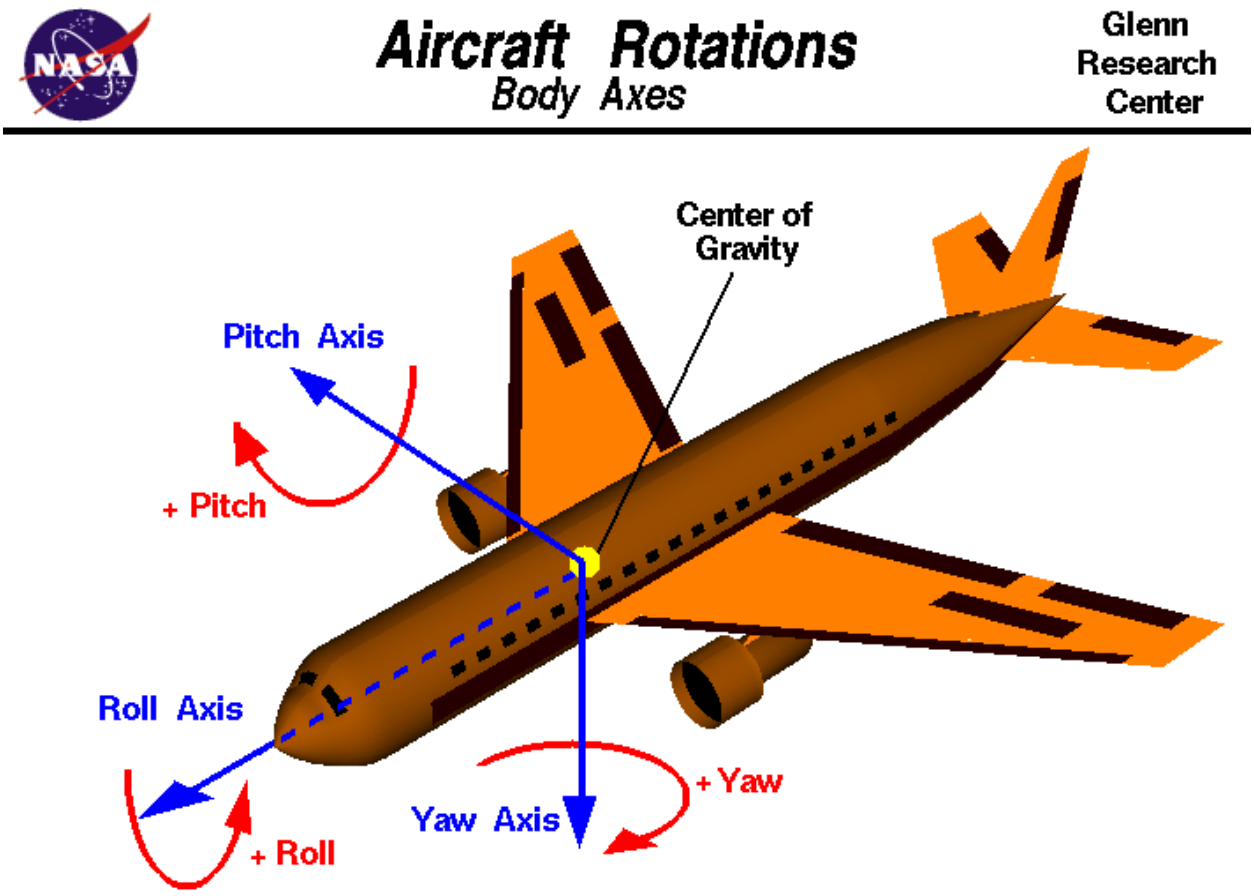

Figure 2 - Principle Axis and Euler Angles [2]

Clients & Stakeholders

The main client for this project is an Aerospace Engineering professor from Cal Poly, Dr. Paulo Iscold. Dr. Iscold is a fairly new faculty member focused in aircraft design, aerodynamics, and flight testing. Before becoming a professor, he was apart of the Center for Aeronautics Studies of the Federal University, in Brazil where he developed many new airplane designs, including the CEA-311 Anequim which set five world records with the International Federation of Aeronautics. Dr. Iscold also worked for a Red Bull Air Race team, where his flight path optimization, aerodynamic modifications, and flight simulation helped the British Team Bonhomme win the World Championship.

Here at Cal Poly, Dr. Iscold is currently working with students to develop a high performance sailplane wing under the title Project Nixus. He has partnered with Jim Payne, a world famous test pilot who recently broke the record for highest altitude sailplane flight with the Perlan Project, who will pilot the glider equipped with the Nixus wings seen in Figure 3.

Figure 3 - Nixus Glider Developed by Dr. Paulo Iscold [3]

Project Ideas and Framed Insights

Because of the compact nature of the glider, its dashboard does not feature a flight director. To assist Jim Payne with control of the aircraft, and to gather important information on the flight path, Dr. Iscold wanted to develop an embedded system to take in IMU and GPS data, and display a flight director-like interface for the pilot. After discussion, Dr. Iscold and I decided

the best way to accomplish this was to use an internet-of-things (IoT) style micro-controller (MCU) communicating with a mobile phone application, which would serve as the graphical user interface (GUI) and look similar to a real flight director as seen in larger commercial planes. Dr. Iscold had access to very high-powered and accurate IMUs that could synthesize accelerometer, gyroscopic, and GPS data and output the principle axis. This sensor could be attached to the MCU, which did not need to be powerful since its only job would be to take in the sensor's data and transmit it to the phone. The phone would use this data to run the flight director. The initial app would simply display the sailplane's current trajectory. After testing the system inflight, the app could be expanded upon to include more features such as assisting the pilot in maintaining a set path.

Project Overview

After narrowing down the scale of the project to match that of a senior project, Dr. Iscold and I decided that I was to choose and program a MCU that could read in the IMU's data and send it to an iPhone. I was also responsible for developing an iOS application that could connect to the MCU and display the Euler angle data. Dr. Iscold could then send this system up in the air with Jim Payne on the next test flight and work on enhancing the capabilities of the system from there. A early design draft of the GUI display of the mobile application is seen in Figure 4.

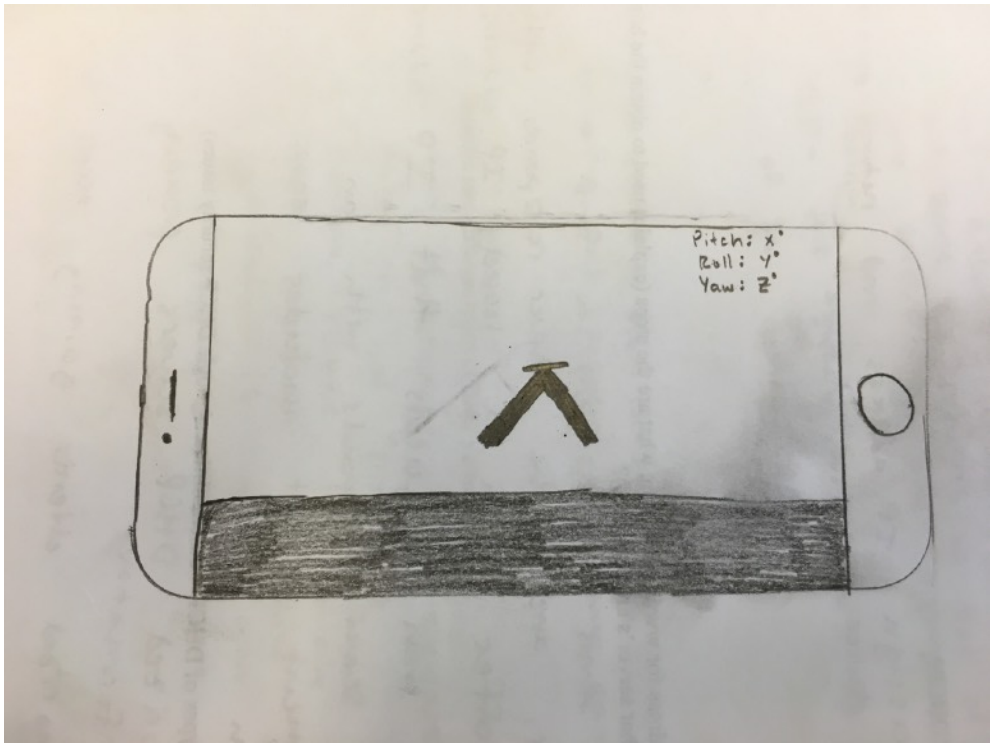

Figure 4 - Preliminary Draft Design for Mobile Phone Application

Project Deliverables

- iOS application to act as GUI display of IMU data
- Micro-controller to transmit sensor data to phone application
- Necessary hardware to connect IMU to MCU, and MCU to phone

Formal Project Definition

Customer Requirements

- Choose an MCU to acquire data from the IMU and transmit it to the phone
- Develop the hardware system necessary to communicate between the IMU and MCU
- Develop the hardware/software system necessary to communicate between the MCU and phone
- Develop an iOS application to process IMU data and display it in a GUI similar to a flight director

Engineering Requirements

This project had very few explicit engineering requirements. As long as it fit in the aircraft and responded to changes in flight orientation in reasonable time, Dr. Iscold was satisfied with the design. With that information, I set some requirements for myself. (Note - reaction time refers to the time it takes for a change in aircraft to appear on the mobile application.)

Table 1 - Engineering Requirements

Specification Number	Parameter Description	Requirement or Target with Units	Tolerance	Risk	Compliance
1	MCU Voltage Necessary	5V	+/- 0.5V	M	A, T
2	Hardware Surface Area	1 sq ft	+/- 0.3 ft in each direction	M	A, T, I
3	Hardware Height	5 in	+/- 1 in	M	A, T, I
4	Hardware Weight	0.5 lbs	+/- 0.1 lbs	H	A, T, I
5	Reaction time	0.1 s	+/- 0.1 s	H	I
6	Battery Life while using App	2 hours from full charge	Min	H	I

User Stories

Assuming the IMU was properly mounted on the aircraft and connected to the MCU, the user would mount his/her phone on the dashboard of the sailplane before takeoff. The pilot would then open the application and be prompted to connect to the MCU. Once connected, he/she would see the orientation of the aircraft and Euler angles displayed on the phone throughout the duration of the flight. Once landed, the pilot could disconnect from the MCU and close the application.

Design

Hardware

VectorNav VN-200 Rugged (Figure 5): The IMU provided by Dr. Iscold is called the VN-200 Rugged series, produced by VectorNav Embedded Navigation Solutions. It combines MEMS inertial sensors, a high-sensitivity GPS receiver, and advanced Kalman filtering algorithms to provide optimal estimates of position, velocity, and attitude. This extremely accurate and expensive IMU is used commonly in industry, and features a 10-pin Harwin connection that transmits data via RS-232/UART. Using the VectorNav Control Center, I can choose the data output format and baud rate. I set this sensor to output Euler angle data at a baud rate of 115200. Figure 9 shows the hardware setup to allow for UART communication.

Figure 5 - VN-220 Rugged Series [4]

BNO055 (Figure 6): Because the VN sensor was so expensive, Dr. Iscold had a limited supply of them and had to use them for other projects. Therefore, I used the inexpensive BNO055 IMU produced by Adafruit to test the system. This sensor can be mounted on a breadboard and transmits the same Euler angle data (though less accurate) via an I2C interface. Switching between the two sensor requires switch between UART and I2C, including a few minor tweaks to the hardware setup and a small change in the code for the MCU, but it allows me to test a majority of the system at my convenience. Figure 10 shows the hardwire setup to allow for I2C communication.

Figure 6 - Adafruit BNO055 IMU [5]

ESP32 (Figure 7): The micro-controller I chose for this project is the ESP32, part of the ESP32 DevKit v1 development board. This board can be mounted on a breadboard and has I2C and UART capabilities to interface with both IMUs. The board also includes a Bluetooth chip that allows for Bluetooth Low Energy (BLE) communication with other BLE devices, such as an iPhone. I chose this specific MCU because of its small size, inexpensive cost, and BLE functionality. The blackbox diagram depicting how the MCU communicates with the other hardwire components can be seen in Figure 8.

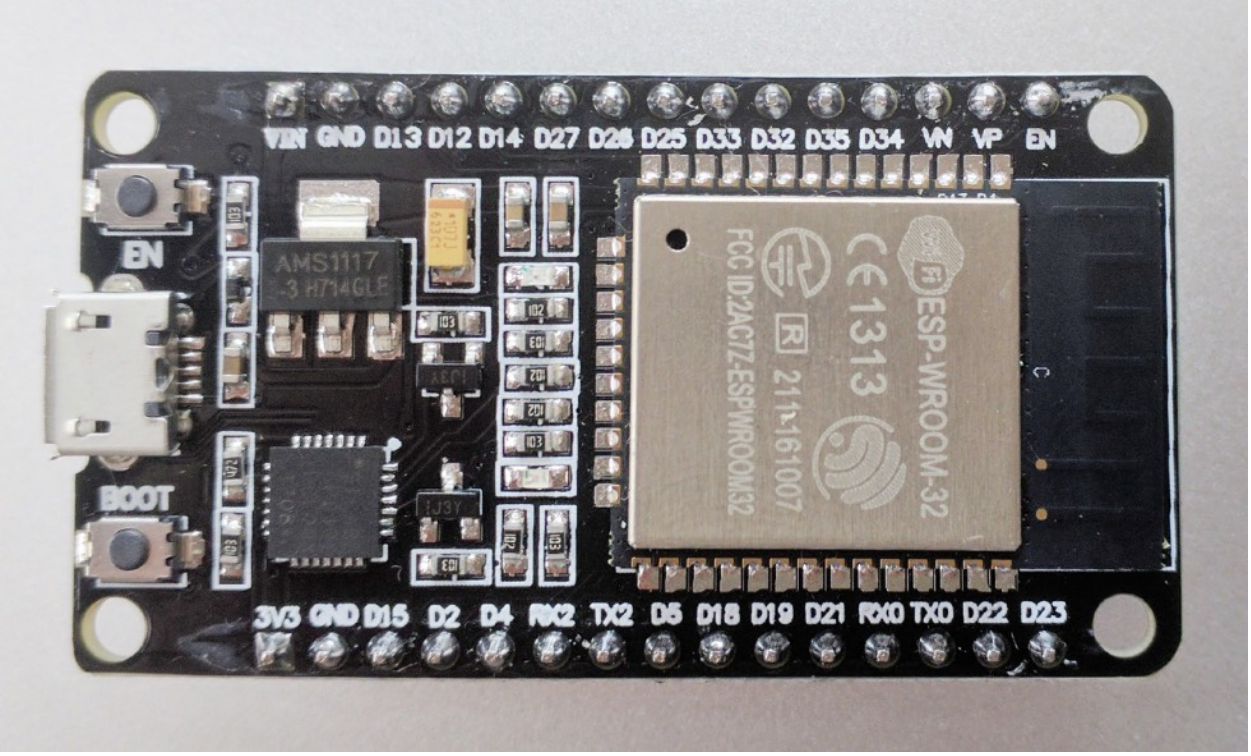

Figure 7 - ESP32 DevKit v1 by Espressif [6]

Hardware Diagrams

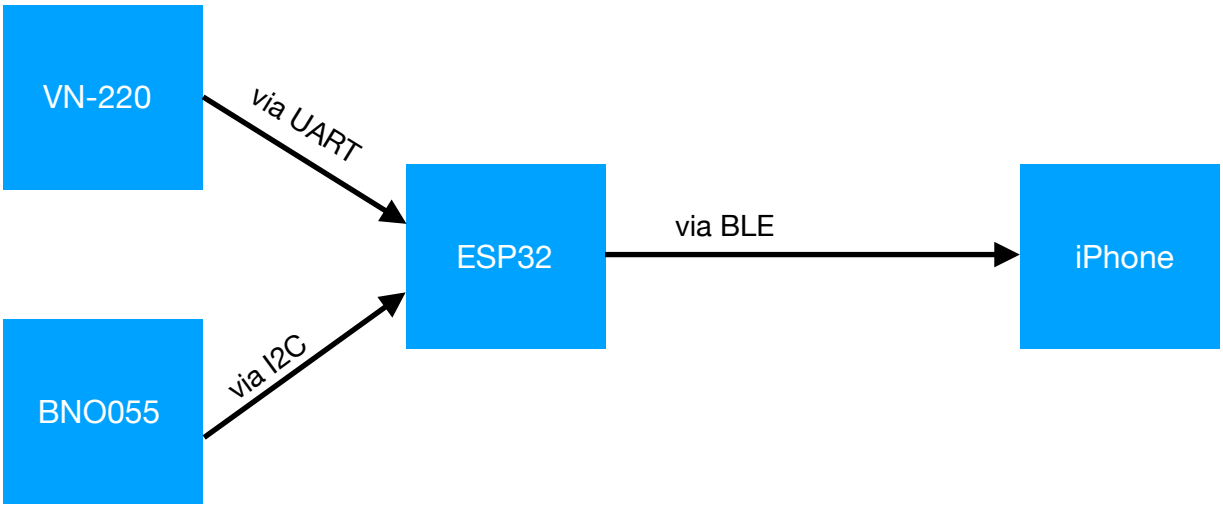

Figure 8 - High Level Hardware Diagram

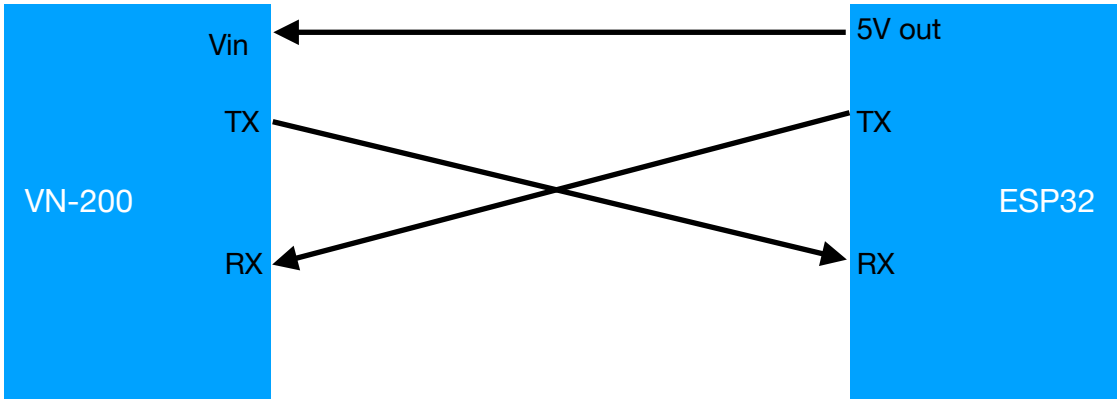

Figure 9 - UART Communication Between MCU and VN-200

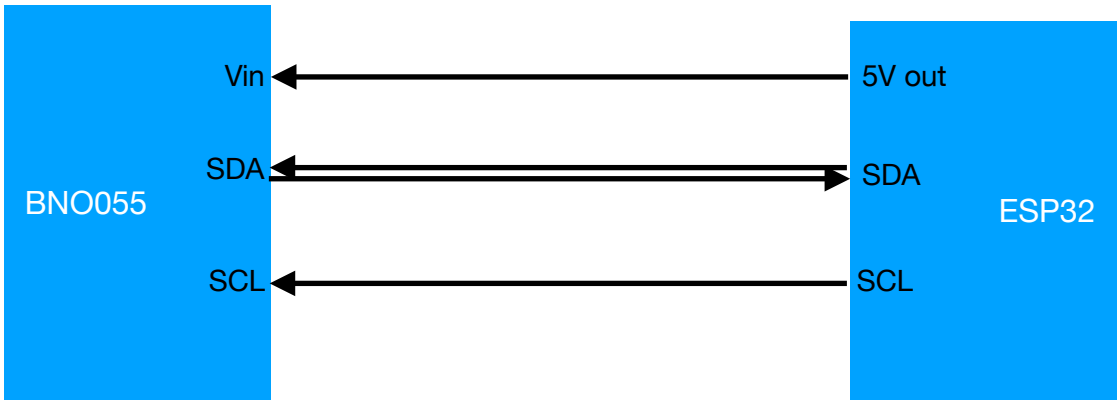

Figure 10 - I2C Communication Between MCU (Master) and BNO055 (Slave)

Bluetooth Low Energy Background

The ESP32 and the iPhone application communicate via BLE. In the BLE model, there are two stages: pre-connection and post-connection. In pre-connection, devices either act as centrals or peripherals. Peripherals (or slaves) advertise themselves and wait for centrals to connect to them, while centrals (or masters) scan for devices. Once connected, one of the devices acts as the client while the other is a server. Servers have a local database of resources or information which it provides to the client. The client can use read/write operations to access this data. The information is split up into characteristics, each with a unique ID so that the client can access data by keyword. When a smartphone application is connecting to an external device, it is common to have the phone be the central/client while the other device is the peripheral/server.

Embedded Software Design

The program running on the ESP32 is a simple BLE peripheral and server written in C++ for the Arduino IDE (for which the ESP is compatible). It uses a unique ID for the connection and each characteristic, polls the sensor, and then sets the characteristic values for each of the Euler angles. A flowchart for this system can be seen in Figure 11.

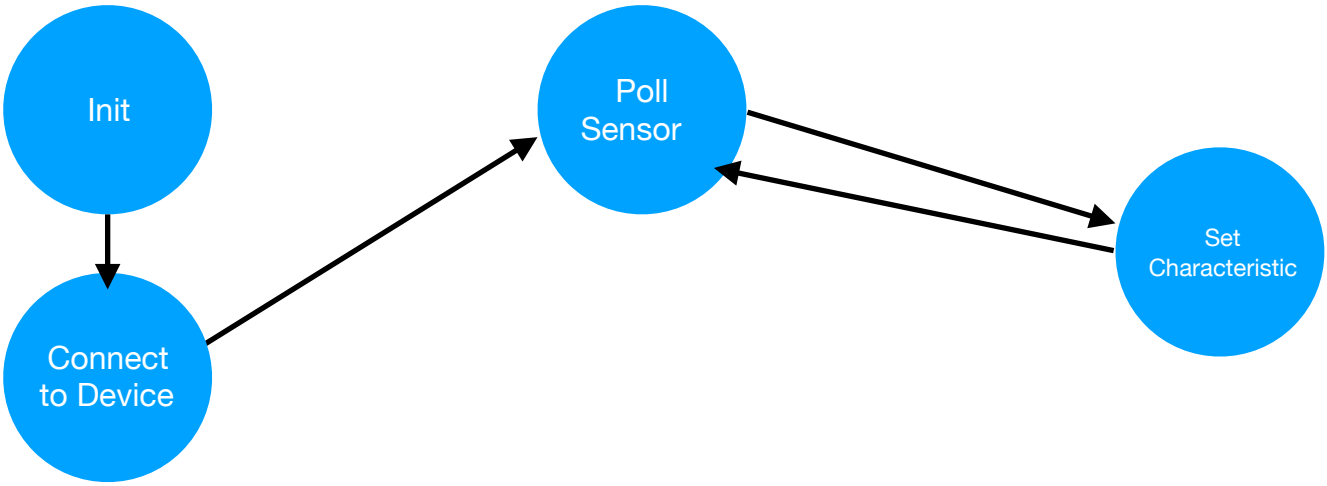

Figure 11 - Flowchart for ESP32 Main Program

iOS Development

The application code is where all of the processing is done on the sensor data, and was by far the most time consuming to write. In this system, the smartphone acts as the central peripheral and uses a Xcode library called CoreBluetooth to handle the BLE connection. The program scans for the specific ESP32 via the ESP32 peripheral ID, connects to it, and then reads characteristic values for each principle axis via their characteristic IDs. It then updates the GUI based on the sensor data. This GUI is shown in Figure 12 at its default position (all Euler angles at zero). The white bar representing the aircraft will move up and down relative to the horizon to display pitch and rotate clockwise or counter-clockwise to represent roll. Yaw is displayed by the displacement of the black bar in the top middle of the screen from true north.

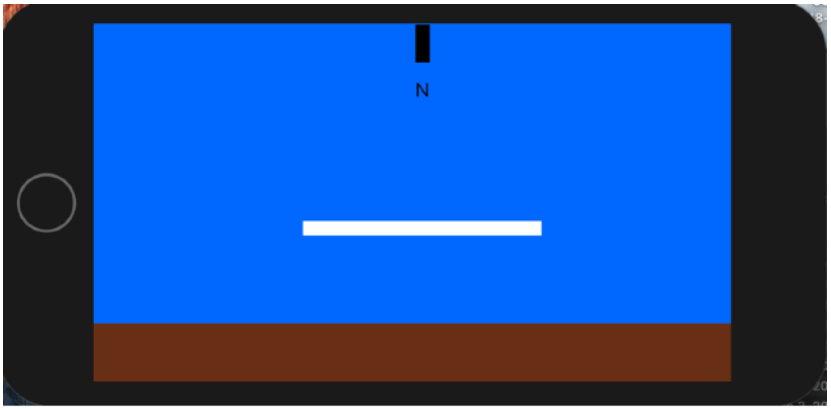

Figure 12 - GUI App Display in Default Position

Testing and Future Work

Embedded System

To test the embedded system before the iPhone application was complete, I used an iPhone app called Blynk. This app is free on the Apple Market place and allows you to easily connect to ESP or similar BLE devices and display data. Using that app and accompanying Arduino library, I was able to receive sensor data via UART and transmit it to the Blynk app, displaying the Euler angles as text on the screen.

Full System with App

At the time of writing this report, there are still lingering issues with the app. While it is able to connect to the ESP, updating the GUI real-time with the sensor values proved to be fairly tricky. I am still having trouble making the aircraft marker and yaw ticker go to exactly where I want them to on the screen. After this is complete, I plan on adding more markers so that the pilot can read the angles with precision. Currently, the app only shows relative movement on each axis but does not show the actual value. Notice how Figure 1 in the introduction has measurement tools to read off angles as they change, while the GUI in Figure 12 lacks these markers and thus only shows relative position with no precision. After the GUI is finalized, Dr. Iscold will mount the sensor on his glider and Jim Payne will pilot a test flight while using the app. From Payne's input, I will be able to tweak the GUI and we can iterate these tests from there.

Future Work

I plan to continue working with Dr. Iscold until he is completely satisfied with the app. When the app is finished, I envision it to look much more similar to real flight directors as shown in Figure 1. The user will have the option to preset a flight path and there will be a marker on the GUI that the pilot will line the plane up with to keep that path. Angle and GPS data will be easier to read with visible angle markers. In addition, the iPhone will store some of the angle data, either on the phone or send it to a remote web server, so that flight information can be accessed and reviewed after the flight is over.

Reflection

This project truly allowed me to use all aspects of my computer engineering degree. I was responsible for both hardware and software development which is something I can see myself doing in industry after graduation. I want to work in the field of embedded systems and this project gave me a glimpse as to what that entails. It was also very meaningful to me that the system I have worked on and will continue to work on will be used for a real-world problem. I have completed many projects in my embedded systems and mechatronics courses that were one-and-done; I build them over the course of a couple months, demoed them to a professor, and then tore them down and never used them again. Working on a system that I know will be useful to someone long after I have finished working on it was very rewarding.

When I started the project, I grossly underestimated the difficulty of app development. I came in with a preconceived notion that anyone could build an app in a matter of hours because apps were so commonplace. Not realizing that creating an iPhone application meant learning a new programming language and changing the way I think about programming because of the Xcode development environment, I spend a large portion of the project time going through example application. I first learned how to place and manipulate UI elements and then how to get data from Bluetooth, and am still ironing out how to use that data to interact with the UI.

Bibliography

[1] Guide, Aeronautics. “Flight Director Systems - Aircraft Instrument Systems.” *Aircraft Systems*, Blogger, www.aircraftsystemstech.com/2017/05/flight-director-systems.html.

[2] Hall, Nancy. “Aircraft Rotations.” *NASA*, NASA, 5 May 2015, www.grc.nasa.gov/www/k-12/airplane/rotations.html.

[3] Iscold, Paulo. “Nixus Project.” *Nixus Project*, Facebook, 2019, www.facebook.com/NixusProject/.

[4] “VN-200.” *VectorNav Technologies*, 2019, www.vectornav.com/products/vn-200.

[5]“Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055.” *Amazon*, Amazon, 2019, www.amazon.com/Adafruit-Absolute-Orientation-Fusion-Breakout/dp/B017PEIGIG.

[6] Espressif. “DOIT ESP32 DEVKIT V1 Board · Issue #544 · Espressif/Arduino-esp32.” *GitHub*, 2017, github.com/espressif/arduino-esp32/issues/544.

Appendix

Bill of Materials

Item	Quantity	Cost
Vector Nav VN-200 Rugged Series Sensor	1	$0*
BNO055 IMU	1	$35
Breadboard	1	$8
Male-to-Male Jumper Wires	20	$6
ESP32 DevKit v1	1	$11
	Total Cost	$60.00

**Note - VN-200 sensor estimated cost is around $2500, but it was supplied free of charge by Dr. Iscold*

Analysis of Senior Project Design

Please provide the following information regarding your Senior Project and submit to your advisor along with your final report. Attach additional sheets for your responses to the questions below.

Project Title: _____
_____ Quarter / Year Submitted: _____
Student: (Print Name) _____ (Sign) _____
Advisor: (Print Name) _____ (Initial) _____ Date: _____

• **Summary of Functional Requirements**

Describe the overall capabilities of functions of your project or design. Describe what your project does. (Do not describe how you designed it.)

• **Primary Constraints**

Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors or other issues that impacted your approach? What made your project difficult? What parameters or specifications limited your options or directed your approach?

• **Economic**

o Original estimated cost of component parts (as of the start of your project)
o Actual final cost of component parts (at the end of your project)
o Attach a final bill of materials for all components
o Additional equipment costs (any equipment needed for development?)
o Original estimated development time (as of the start of your project)
o Actual development time (at the end of your project)

• **If manufactured on a commercial basis:**

o Estimated number of devices to be sold per year
o Estimated manufacturing cost for each device
o Estimated purchase price for each device
o Estimated profit per year
o Estimated cost for user to operate device, per unit time (specify time interval)

• **Environmental**

Describe any environmental impact associated with manufacturing or use.

• **Manufacturability**

Describe any issues or challenges associated with manufacturing.

• **Sustainability**

o Describe any issues or challenges associated with maintaining the completed device or system.
o Describe how the project impacts the sustainable use of resources.
o Describe any upgrades that would improve the design of the project.
o Describe any issues or challenges associated with upgrading the design.

• **Ethical**

Describe ethical implications relating to the design, manufacture, use or misuse of the project.

• **Health and Safety**

Describe any health and safety concerns associated with design, manufacture or use.

• **Social and Political**

Describe any social and political concerns associated with design, manufacture or use.

• **Development**

Describe any new tools or techniques used for either development or analysis that you learned independently during the course of your project.

Summary of Functional Requirements

The project was to design a GUI system for an iPhone to view position data from a IMU/ GPS sensor. The app was to look similar in style and functionality to a flight director as commonly seen in commercial aviation and be able to track the motion of an aircraft in real time.

Primary Constraints

The most limiting factor in this project was the fact that the app had to be used by a pilot while in flight. A flight director type application would have been much easier to design if it was done on a full-sized computer/laptop instead of a mobile phone. This meant that instead of directly connecting the MCU to the computer, I had to design a Bluetooth system to transfer data between the phone and the MCU.

Economic Information

The estimated cost of this project before starting work on it was fairly inexpensive, around $25. An ESP32 module with Bluetooth was only around $10, and I factored in another $15 for a breadboard, wires, and any other small hardware I would need. The final price was $60 total, with the extra money being spent on an additional IMU, the BNO055. I decided to purchase the extra sensor when I realized it would be beneficial to have some way of testing the system when I didn't have access to the VN-200.

The estimated time to complete this project was only around two months, with about 5 hours of work a week. I believed that the GUI I had to create was fairly simple, and since the MCU simply had to transmit rather than manipulate data, the code for the ESP would be fairly simple. I drastically underestimated the time it would take to create an iOS application. I had never done app development before, and I thought it would be easy because of my previous coding experience and the multitude of resources available online. However, Xcode took a while to get the hang of, and I had to do a lot of experimentation with basic apps to get the functionality down. The final project took about seventy to eighty hours to complete.

If Manufactured on a Commercial Basis:

This product would have a small niche market. The only consumers that would be interested in this are people who pilot for leisure and own small planes that do not include a flight director. Anyone interested must be willing to spend the large sum of money necessary to acquire the VectorNav sensor because the BNO055 is not accurate enough for use in aircraft. While the app can be sold for free (leaving the option to add advertisements), the hardware required to transmit the sensor data would cost around $25, and it would be relatively inexpensive to use, only requiring a 5V power source that would be readily available on an aircraft.

Environmental Impact & Sustainability

The environmental impact of this embedded system is fairly minimal. It does not require much power, but there are still some inherent sustainability issues in producing the raw materials necessary for the sensors and circuitry. The operating costs in terms of energy usage for the embedded systems and app are small and can be disregarded in the bigger picture of flying an aircraft.

Manufactoring

One issue with the hardware is that a breadboard connection is usually not reliable. This can be fixed by developing a wire that has a female 10-pin Harwin on one side and a male microUSB connection on the other. That would allow for direct connection between the MCU and VN-200, thus removing the breadboard. The kit for the product would then include this cord and an ESP32, letting the customer purchase the sensor on his/her own.

Ethical, Health, Safety, Social, and Political Issues

The largest concern for this system is security and someone connecting to the app with nefarious intent. While designing the system, I was not worried about security threats and assumed only people involved in the flight would be connecting to the system. Issues arise because anyone can copy the Bluetooth protocol used to communicate between the MCU and app. This would allow them to create “fake” sensor data and feed this false information to the pilot via the app. While the app does not control the aircraft directly, it would disrupt the pilot and cause confusion, which could lead to problems in the air.

In real flight directors used in industry, the director has access to the same information as the autopilot. The pilot can then use it to confirm the computer navigation system is working as expected. If this type of system was infiltrated, the attacker could easily manipulate the autopilot to take the plane wherever he/she desired. In the commercial aviation industry, this would cause issues on a much larger scale since the aircraft involved carry more passengers or cargo. This kind of attack could also be used to disrupt military aircraft, though security in the military is much more heavily enforced.

Development

During the development of this project, I had to teach myself iOS development. Though proficient in coding from my classwork at Cal Poly, I had never used Xcode or been exposed to the Swift or Objective-C languages. I started by following a few introductory projects provided by Apple where they took me through how to build a basic UI and how to make UI objects move on the screen. I started in Objective-C, thinking my prior C experience would make this language easier to jump into, but quickly found Swift to be easier to read and quicker to code. I also had to learn Bluetooth and BLE technologies, and how a client/server Bluetooth connection was established. From the MCU side of things, a BLE server was fairly easy to setup. I was able to build from the BLE example code provided for the ESP32. Getting the app to act as a client and connect to the ESP proved to be the most difficult aspect of the project, but I was able to answer most of my questions through forum posts on websites like Stack Overflow or the Apple Development Community.