

**University of Victoria**  
*Faculty of Engineering*  
**Spring 2011 - CENG/ELEC/SENG 499 Report**

## **FLOCS – Flight Location and Orientation Control System**



**Matthew Benstead (matthewb@uvic.ca)**  
**Steven Marks (smarks@uvic.ca)**  
**Jacob Bieling (jbieling@uvic.ca)**  
**Justin Reiher (jreiher@uvic.ca)**  
**Sylvie Foss (fossyl@uvic.ca)**

**April 1, 2011**

**In partial fulfillment of the requirements of the  
B.Eng. Degree**

Matthew Benstead  
4050 Lockehaven Dr  
Victoria, British Columbia  
V8N 4J5

Pan Agathoklis  
Professor, F.E.I.C., P. Eng.  
Faculty of Engineering  
University of Victoria  
PO Box 3055 STN CSC  
Victoria, British Columbia  
V8W 3P6

April 1, 2011

Dear Pan Agathoklis,

Please accept the attached CENG/ELEC/SENG 499 report entitled "FLOCS – Flight Location and Orientation Control System" as partial fulfillment for the CENG/ELEC/SENG 499 course. This report is a result of our work designing an embedded data acquisition system and monitor display for the UVic AERO and AUVic clubs.

We would like to thank you for guiding us through this project and your support. We would also like to thank Dr. Kin Fun Li, 499 Coordinator, as well as the IEEE UVic Student Branch, and the Department of Electrical and Computer Engineering for their support.

Sincerely,

The FLOCS Team

# Table of Contents

List of Tables.....	iii
List of Figures.....	iii
Summary.....	iv
1 Introduction.....	1
1.1 Introduction to FLOCS.....	1
1.2 Project Specifications.....	1
2 Measurement Instrumentation.....	2
2.1 Orientation Measurements.....	2
2.2 Position Measurements.....	3
2.3 Velocity Measurements.....	4
2.4 Proposed Actuators.....	4
3 Microcontroller Selection.....	5
3.1 X86.....	5
3.2 ARM.....	6
3.3 PIC.....	6
3.4 ATMEL.....	6
3.5 Result.....	7
4 Development Platform.....	8
4.1 Recommendations.....	8
5 Sensor Package.....	9
5.1 ADXL345 3-Axis Accelerometer.....	9
5.2 ITG-3200 3-Axis Gyroscope.....	10
5.3 HMC5843 3-Axis Digital Compass.....	11
5.4 SCP-1000 Pressure Sensor.....	12
5.5 Maxsonar EZ4 Ultrasonic Range Finder.....	12
5.6 GPS-Venus634LPx GPS Board.....	13
5.7 Pitot Tube.....	14
6 Communication System.....	15
6.1 On Board Ethernet.....	15
6.2 Wireless Transmitter.....	16
6.3 Wireless Receiver.....	17
6.4 Computer Ethernet Connection.....	18
6.5 Future Communications Development.....	18

7	Printed Circuit Board Design .....	19
8	Telemetry System .....	21
8.1	Design of the Telemetry System .....	21
	Sensor Monitor and Sensors.....	21
8.2	Data Storage and Data.....	22
8.3	Filter Controller and Filter .....	22
8.4	Network.....	22
9	Implementation of Telemetry System.....	23
9.1	FLOC Framework .....	23
9.2	Sensor Drivers .....	23
9.3	Inertial Measurement Unit.....	23
9.4	Transmitting Data.....	25
9.5	Implementing the Design.....	25
10	Monitor.....	26
10.1	Overview.....	26
10.2	Why OpenGL was Chosen .....	26
10.3	The Display .....	27
10.4	The Dashboard .....	27
10.5	The 3D Rendition of the Airplane.....	28
10.6	The 3D Position Graph.....	28
10.7	Code Adaptability.....	29
10.8	Future Work.....	30
11	Conclusions.....	32
12	References.....	33
	Appendix A - Project Data .....	1

# List of Tables

Table 1- Microcontroller Comparison .....	7
---	---

# List of Figures

Figure 1 - Sensor Package .....	1
Figure 2- Image of Aircraft Orientation.....	2
Figure 3- Image of Aircraft Inclination.....	2
Figure 4 - Image of Demo Board (EVK1100) .....	8
Figure 5 - Image of Accelerometer .....	9
Figure 6 - Image of Gyroscope.....	10
Figure 7 - Image of Compass.....	11
Figure 8 - Image of Pressure Sensor .....	12
Figure 9 - Image of GPS .....	13
Figure 10 - Image of Pitot Tube .....	14
Figure 11 - Overall System Workflow.....	15
Figure 12 - Image of Bullet Transmitter .....	16
Figure 13 - Image of AirGrid M5 Receiver.....	17
Figure 14 - PCB Design Image .....	19
Figure 15 - FLOCS UML Design .....	21
Figure 16 - FLOCS Sensor and Sensor Monitor .....	22
Figure 17 - Image of Filter Controller .....	22
Figure 18 - Image of Quaternion Coordinate System .....	24
Figure 19 - Monitor Dashboard .....	26
Figure 20 - FLOCS 3D Display.....	28
Figure 21 - FLOCS Position Graph .....	29

## Summary

The Flight Location and Orientation Control System (FLOCS) project objectives were to design and build a data acquisition system and monitor display program for the UVic AERO and AUVic clubs. The data acquisition system will eventually be expanded by the AERO club to become their Autopilot for a small aircraft they are developing.

The system integrates a number of sensors including a compass, gyroscope, accelerometer, GPS, barometer, pitot tube, and ultrasonic sensors to measure the orientation, position, and velocity data from the aircraft. Managing the whole system will be a real time operating system (FreeRTOS) which is responsible for scheduling concurrent sensor tasks to run on the the onboard AVR32 microcontroller.

The microcontroller retrieves the raw sensor data, filters it to remove noise, and stores the information in dedicated buffers. The data is then be periodically read, converted into Euler angles, and transmitted through a wireless communication system to the monitor software running on a computer.

The communications system consists of an onboard Ethernet port, wireless transmitter, a wireless receiver, and a standard Ethernet port on a computer running the Monitor software. The wireless hardware runs at 5 GHz and can transmit data long distances.

The monitor software interprets the received data, and using Java and OpenGL, displays the information in a user friendly way. It shows the orientation of the aircraft in 3D, diagrams for yaw, pitch, and roll, as well as a series of gauges representing speed and acceleration.

# 1 Introduction

## 1.1 Introduction to FLOCS

The Flight Location and Orientation Control System (FLOCS) is a product designed for the UVic AERO and AUVic clubs to be used as control platforms for autonomous vehicles. The project has been designed around the system requirements provided by the UVic AERO club which plans to use this system to autonomously fly an aircraft.

## 1.2 Project Specifications

The entire system consists of a custom built printed circuit board (PCB), which acts as a sensor and processing package. This package will wirelessly transmit its information to a monitoring system that will graphically display the data in an intuitive manner.

A typical aircraft is equipped with many instruments that provide the pilot with vital information about the aircraft. An autopilot system requires the ability to measure the same information and make real time decisions based on that information to safely navigate an aircraft.

The information that would be needed consists of: airspeed, heading, remaining fuel, altitude, orientation with respect to the ground, engine speed and other related information. In addition pilots use a combination of GPS and maps to find their location by using landmarks or ground beacons to relay their position. The necessary information provided to a pilot has been considered in the design of this autopilot system. This report outlines how the FLOCS platform has been designed so that it can be used as an autopilot system.

In addition to designing the system to be able to fly an aircraft, safety was considered in the design to account for potential failures in the system. Provisions have been put in place to allow the autopilot system to be cut off from the aircraft control if there were a failure in the system. An image of the populated sensor package is shown below.



*Figure 1 - Sensor Package*

## 2 Measurement Instrumentation

### 2.1 Orientation Measurements

One of the most critical concerns when designing an autopilot system is being able to measure the aircraft's orientation in space. Orientation in space for an aircraft is measured by Euler angles which can be determined by yaw, pitch and roll.

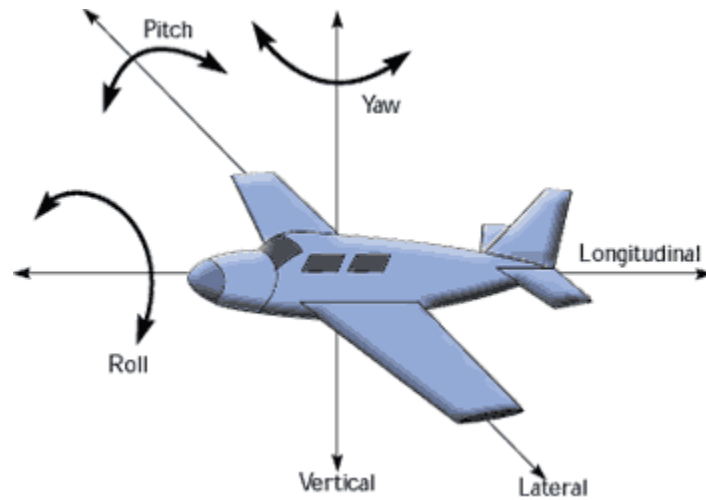


Figure 2- Image of Aircraft Orientation

Several solutions exist, however the most popular is to use an accelerometer, gyroscope and compass fusion. An electronic gyroscope measures the angular rate of change, the accelerometer is used as a reference to gravity, while the compass is used as a reference to the Earth's magnetic North, though It is possible to use an accelerometer to measure inclination in a plane. Figure 3 shows how an accelerometer is used to measure inclination:

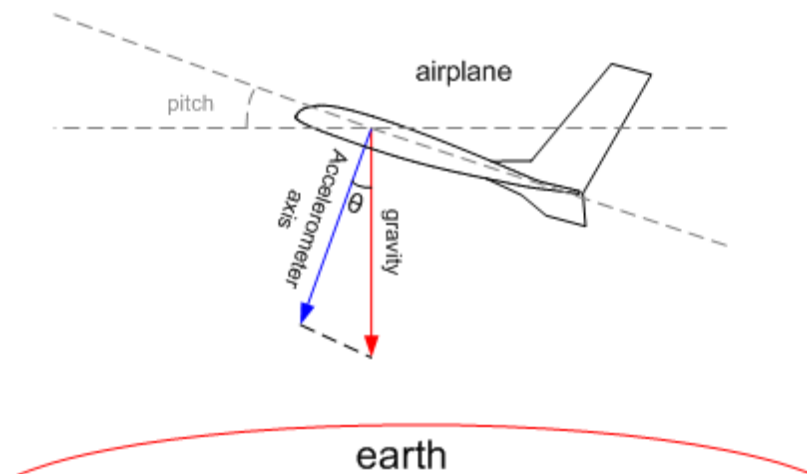


Figure 3- Image of Aircraft Inclination



It should be noted however that using only an accelerometer to measure orientation presents two problems. Accelerations due to turbulence during flight will cause the measurement of gravity to fluctuate. In addition it is not possible to measure yaw with just an accelerometer as rotation about the z-axis does not incur any change in gravity measurements.

Integrating the angular rate of change gives angular position therefore the use of a gyroscope is optimal for calculating Euler angles.

$$\int d\omega / dt = \theta$$

The issue with just using a gyroscope however is that there is no zero reference, so error accumulates over time due to drift. However using the accelerometer in conjunction with the gyroscope is a reliable way to measure the Euler angles. A weighting of sensor data is performed to mix the angles calculated by the accelerometer and gyroscope to provide more accurate Euler angles.

Since there is drift over time with integrating the gyroscope measurements, the compass sensor is used as a point of reference to Earth's magnetic North which corrects for the yaw measurement. With the addition of a compass sensor, the system now has a fixed reference point which can be used to correct errors in the gyroscope measurements.

A combination of all three sensors provides excellent accuracy and measurement of the Euler angles and thus can be used to measure the aircraft's orientation.

## **2.2 Position Measurements**

The most common tool in today's industry to measure position is the Global Positioning System (GPS). This is not the only solution to position measurement, but is the most prevalent. Absolute positioning can be measured from a starting location, however this is not an accurate solution in the application of flying an aircraft. GPS modules are readily available, and are used in many applications. Many GPS modules have antennas built into them however for the project this is not desirable. In order to get a strong signal the antenna needs to be exposed to the sky. Taking this into consideration, a GPS module with an external antenna was selected.

GPS units are very accurate at measuring latitude and longitude locations on the planet, however GPS units tend to lack accuracy in altitude measurements. For this reason two additional sensors have been selected to provide accurate height measurements.

The first sensor is a barometer, which measures pressure. There is a relation between altitude and atmospheric pressure, which can be used to calculate height above sea level, as described below.

$$h_{alt} = \left(1 - [p_{sta} / 1013.25]^{0.190284}\right) \times 44307.6934$$

The pressure in the above equation needs to be measured in hectopascals. This provides sufficiently accurate height measurement for flight calculations. The uncertainty in the measurement depends on the barometric sensor. Typically this ranges from 0.5m to 3m accuracy depending on the sensor. However during landing and takeoff higher precision is required to ensure the aircraft does not crash. The solution to this problem is to use an ultrasonic sensor which can provide precision in centimeters but only at close range. Thus it is used only for landing and takeoff maneuvers.

### 2.3 Velocity Measurements

In an aircraft there are several key speed and velocity measurements that are important in navigation. Typically these are airspeed, wind speed, and ground speed.

Airspeed is defined as the speed air is moving over your wings. This is critically important because if the airspeed drops below a certain threshold the aircraft will stall and begin to lost altitude. Airspeed is measured by using a digital pitot tube. A pitot tube is a measurement between the dynamic and static pressure.

The wind speed can be roughly estimated by subtracting airspeed from ground speed. Wind speed is defined as how fast the wind is moving in a given direction with respect to ground, which affects flight trajectory. This is especially important during landing and takeoff procedures.

Ground speed is defined as the speed at which you are travelling relative to the ground. An example of low ground speed but high airspeed would be if the aircraft is flying directly into the wind, thus the airspeed will be greater than the ground speed. Ground speed can be calculated and provided by the GPS module. By calculating the change in location between two points in an interval gives a measure of ground speed.

### 2.4 Proposed Actuators

The autopilot system upon completion will provide the aircraft several pulse width modulated (PWM) signals to control the required flight surfaces and propulsion systems. The controls to be operated the autopilot are:

1. Elevator: This flight surface controls pitch of the aircraft
2. Rudder: This flight surface controls the yaw of the aircraft
3. Ailerons: These flight surfaces control roll, an additional feature will be the ability to control the ailerons individually to act as flaps
4. Throttle: This controls the engine speed

### 3 Microcontroller Selection

When deciding which microcontroller to use in FLOCS the main design criterion were:

1. Single-chip solution
2. 32-bit with a Floating Point Unit (FPU)
3. Available Real Time Operating System (RTOS)
4. Sample code for drivers
5. Ease of use
6. Convenience
7. Price

A single-chip solution would be ideal because the system would be put on an aircraft where weight constraints are a limiting factor. A 32-bit microcontroller with a FPU is desirable so that complex calculations can be computed faster. By using an RTOS, there would be no need to implement a task scheduler which would save time on development. When implementing code for a microcontroller it is always convenient to have available driver code. The previous experiences from FLOCS members, and fellow engineering students allowed us to determine which development environment would give us the best ease of use. Convenience and Price were both main components of the selection, because of the small budget and limited time. The shortlist of microcontrollers are listed below:

- X86
- ARM
- PIC
- ATMEL

#### 3.1 X86

The X86 architecture was the most powerful processor considered. This architecture fulfilled many of the requirements listed above; such as the 32-bit (as well as 64-bit) FPU, able to run Linux real-time system, and is POSIX compliant. This system would be difficult to interface with some of the devices described in the Sensor Package Section 5. A possible solution to this would be to add a PC104 stack, but would significantly increase the weight and price of the system and isn't a single-chip solution. Using POSIX would mean access to lots of sample code and portable design. When with the PC104 stack FLOCS would be at the mercy components for driver source code and the Linux kernel. The FLOCS team consists of some members that are proficient in Linux, but this would require a lot of setup when compiling the real-time kernel, ensuring all drivers work, and having the right libraries installed. In the end, X86 wouldn't be convenient because it would require a lot overhead and cost the team a significant amount of money to do so. The X86 architecture wasn't chosen because of the weight limitations and the complexity it would add to the project.

### **3.2 ARM**

The ARM architecture can be used for a range of projects from general purposing computing to embedded systems. All of the ARMs' are 32-bit processors and most have FPU's. With an ARM chip, FLOCS could install a complex or simpler RTOS, for example a real-time version of Linux or FreeRTOS respectively. This would also allow for a single chip solution because different versions of ARM processors come with varying electrical interfaces (i.e SPI, UART, etc.). There is sample code and an IDE, for which it would make ARM easy to develop on. It wouldn't be easy to use when it came to developing a final prototype with a PCB, because the pins are small or in a ball grid array configuration. If using real-time Linux, there would have the same problems as X86, i.e. a hassle to compile kernels and configure drivers. As for the ease of use, it was unfamiliar ground for the team because no member had used an ARM processor for such a project. This option was also impractical because the development board for a typical ARM processor costs \$250. ARM processors weren't chosen due to cost, difficult to implement and could possibly take more time to setup due to lack of experience from the team.

### **3.3 PIC**

PICs have a wide range of range different electrical interfaces, which gives the most customizable integration with electrical systems which would allow for a single chip solution. There are 32-bit versions of the chips but none have a FPU. The options for an RTOS would be limited to only simple ones, like FreeRTOS where the program gets written into the kernel. There is no option to upgrade to a complex RTOS like real-time Linux. There is sample code but it is sparse and not much of it is supplied by the manufacture. As for ease of use, previous members have worked with PICs and have found getting a good compiler to be expensive, but the documentation for the microcontrollers are excellent. When it came to convenience, development would take up a lot of time because choosing the right microcontroller and development board for the project would need to be decided in advance. A PIC microcontroller wasn't chosen because it would be limiting our system in the way of RTOS and convenience.

### **3.4 ATMEL**

ATMEL microcontrollers have a smaller selection of chips, but have many varying internal structures. For example; number of electrical interfaces, RAM and ROM size, computational and architectural units. This would allow for a greater single-chip solution, because if we decide in the future to upgrade the PCB doesn't need many changes. They have a brand of microcontrollers called the AVR32, with a series called UC3 and versions A, B, and C. The two series that interested us were the A and C. The A is 32-bit microcontroller but has no FPU and C is the same chip with a FPU added on. The only problem with C is that is not yet publicly available and are in limited supply. As for the RTOS we could only run a simple RTOS, but the software package it came already included the RTOS. When it came to ease of use, many team members had already used the AVR32 on projects in the past and all were successful experiences. It came with its own IDE as well as all the driver code. It was also very convenient because the IEEE Student Branch had a few Development boards (EVK1100) and programmers (JTAGIC mkII) on loan in their office. We chose to use the AVR32UC3A0512 micro-controller because it was very convenient and fulfilled all the requirements.

### 3.5 Result

After analyzing all of the options, the team decided to use the ATMEL AVR32UC3A0512 microcontroller. It was chosen because in the future we could upgrade to a AVR32UC3C microcontroller which has an FPU as well as it convenience and ease of use. The table below shows the tabulated results.

Micro-Controller	Single-chip	32-bit	RTOS	Sample Code	Ease of Use	Convenience	Price
X86	No	Yes	Complex / Simple	Linux / Dependent	No	None	Expensive
ARM	Yes	Yes	Complex / Simple	Linux / Dependent	Middle	Middle	Medium
PIC	Yes	Yes	Simple	Dependent	Middle	None	Cheap
ATMEL	Yes	Yes	Simple	Yes	Yes	Yes	Cheap

Table 1- Microcontroller Comparison

## 4 Development Platform

Selection of the AVR32 microcontroller allowed prototyping and testing of the sensors to be done on the EVK1100 development board. This development board provided a standard platform that the team was able to use to test each part of the autopilot system. The EVK1100 development board is shown in Figure 4:



Figure 4 - Image of Demo Board (EVK1100)

The I2C and SPI drivers which are part of the AVR32 framework allowed quick development and testing of the sensors making the validation process for each sensor. Learning how the drivers functioned had a steep learning curve, however once sensor communication had been achieved validating the rest of the sensors went smoothly. To further facilitate testing, the sensors were selected based off of what is available at [www.sparkfun.com](http://www.sparkfun.com). The sensors were acquired on breakout boards which made connecting the sensor under test easy.

### 4.1 Recommendations

Restricting the sensor selection from [www.sparkfun.com](http://www.sparkfun.com) inadvertently made selection of sensors that are nearing or are at the end of life cycle. In future revisions new sensors will be needed to be selected to build the autopilot board.

## 5 Sensor Package

The sensor package is defined as the set of sensors integrated on the autopilot board which are used to measure the parameters needed to autonomously fly the aircraft. FLOCS uses an accelerometer, gyroscope, compass, barometer, ultrasonic sensor, GPS unit, and pitot tube which are described below.

### 5.1 ADXL345 3-Axis Accelerometer

The ADXL345 is a 3-axis digital accelerometer which has a very small footprint and low power usage. The sensor is measuring accelerations of  $\pm 4g$  but can be configured for  $\pm 2g$ ,  $\pm 8$ , or  $\pm 16g$  measurements. A block diagram of the sensor is shown below in Figure 5.

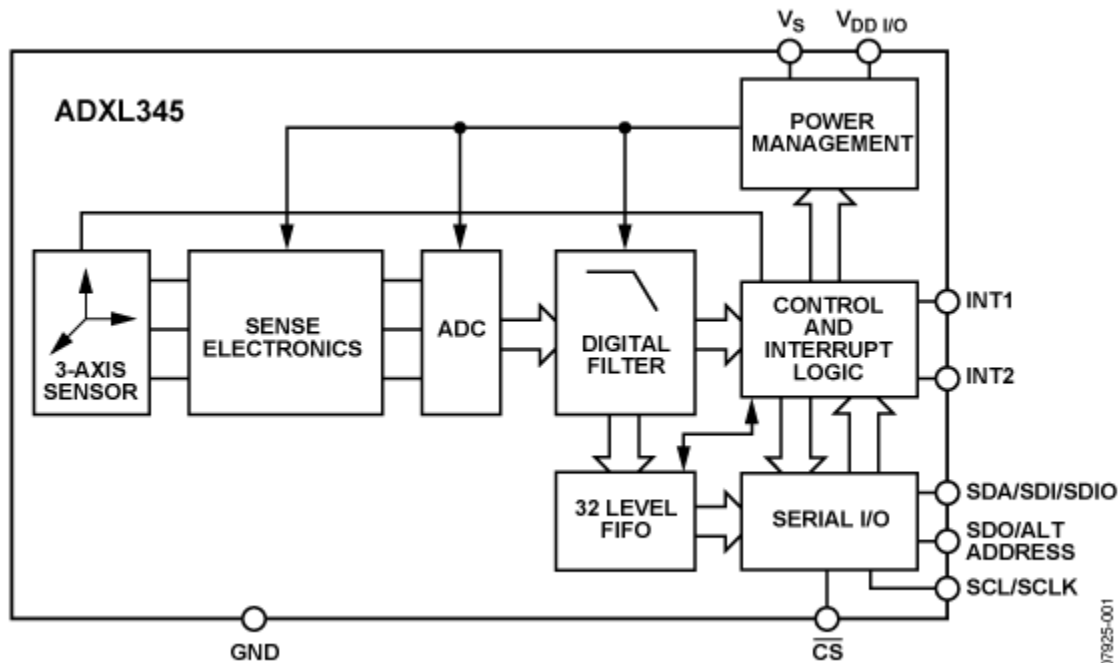


Figure 5 - Image of Accelerometer

This chip can operate at up to 3200Hz but can run in lower speed modes. In low power modes the speed at which accurate data can be measured is reduced. However even at the lowest settings a refresh rate of 12.5Hz is still achieved. The chip also has sleep and standby modes for times when infrequent sampling and very low power use is needed. The data is then processed by the Real Time operating system.

The chip communicates through either Serial Peripheral Interface (SPI) or I2C, and always acts as a slave device. The autopilot board uses 4 wire SPI not only because it is simpler to implement than I2C, but several other devices use I2C exclusively.

## 5.2 ITG-3200 3-Axis Gyroscope

The InvenSense ITG-3200 is a MEMS integrated triple-Axis gyroscope. It has a shock tolerance of 10,000g which makes it extremely resistant to damage from dropping which is a very common occurrence in consumer peripheral electronics. In the ITG-3200 there are three very small mechanical gyroscopes each one recording one axis of motion. Each one of these generates an analog signal which is fed into a 16 bit D/A converter. These digital signals are then fed through a user controlled low pass filter. This chip also contains an internal temperature sensor which is used in reducing thermal noise in the readings from the gyroscopes. This temperature value can also be accessed by external hardware. The ITG-3200 communicates through an I2C interface with a speed of up to 400kHz.

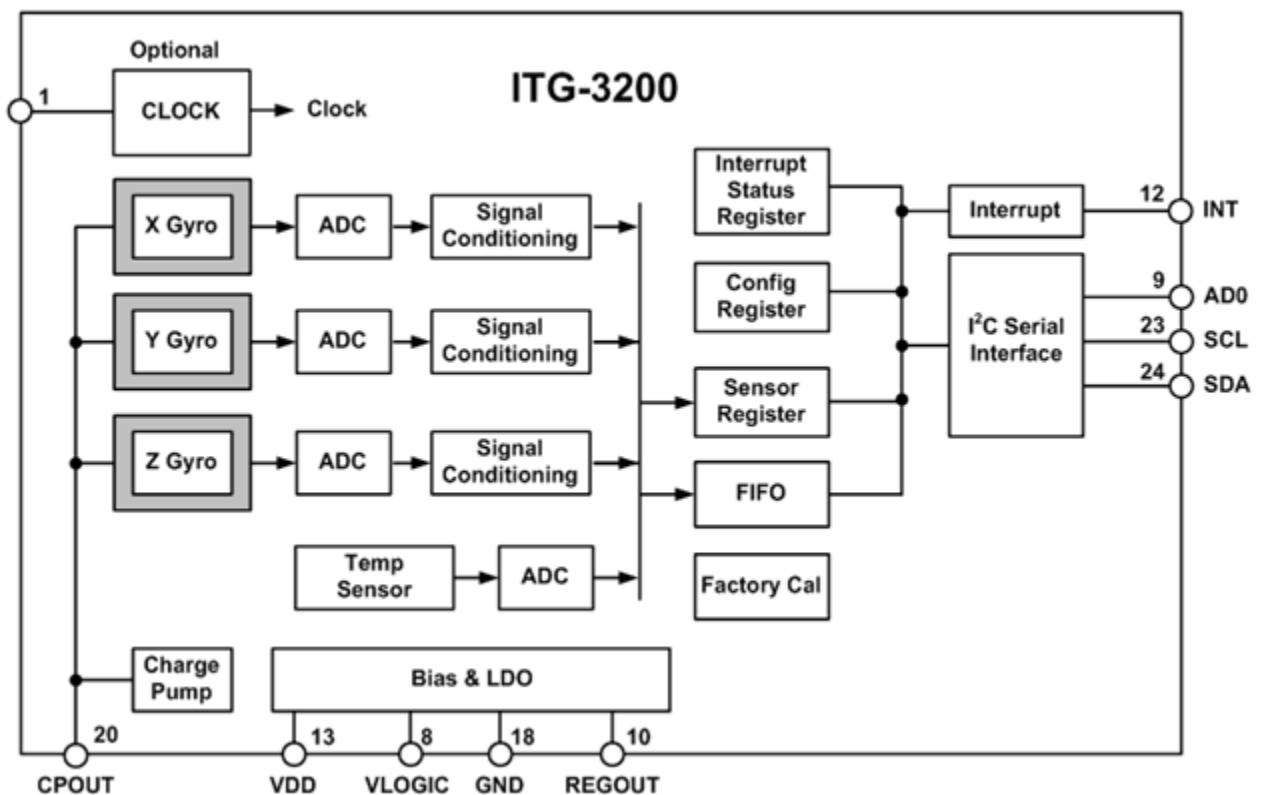


Figure 6 - Image of Gyroscope



### 5.3 HMC5843 3-Axis Digital Compass

The Honeywell HMC5843 is a 3-axis digital compass designed for low field magnetic sensitivity. It possess a digital interface and communicates through the I2C protocol and communicates I2C slave. It allows very precise measurements of the direction and magnitudes of the earth's magnetic field. Using three Anisotropic Magnetoresistive (AMR) sensors, one for each axis, the HMC5843 can very accurately determine the orientation of the chip with 16 bit precision.

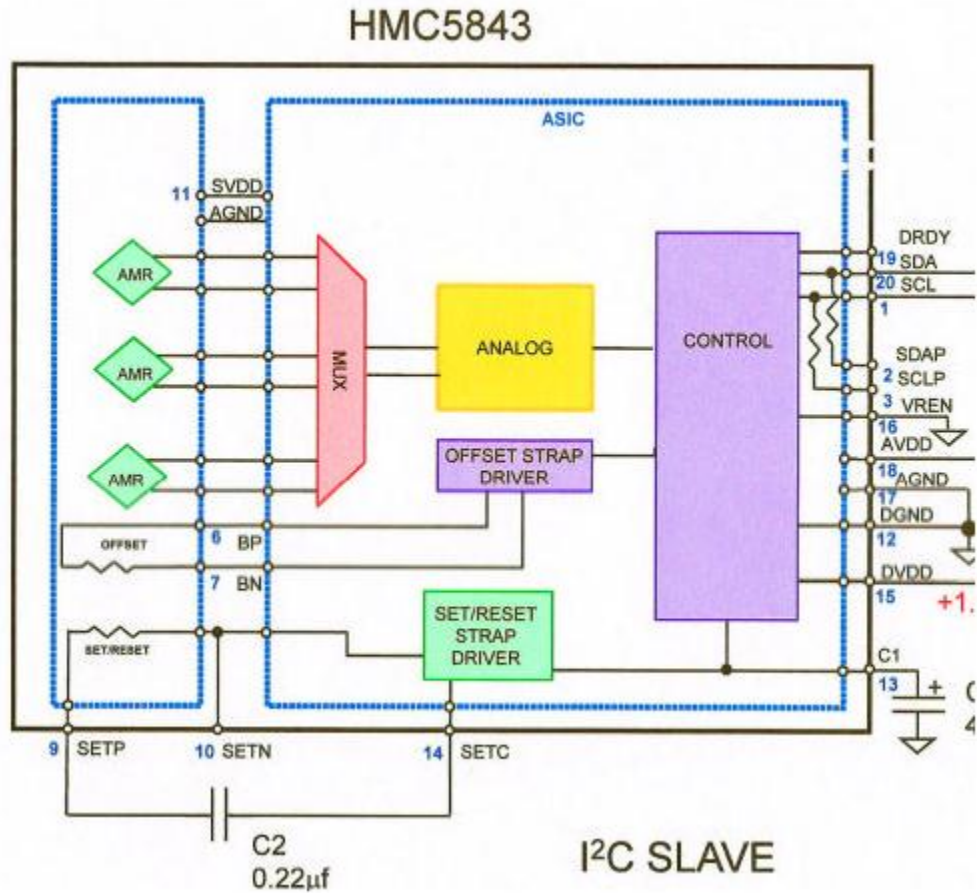


Figure 7 - Image of Compass

## 5.4 SCP-1000 Pressure Sensor

The SCP-1000 is a pressure sensor that measures pressure relative to vacuum and completes all data processing on the chip itself. The chip possess an internal temperature sensor which allows it to adjust and calibrate data internally. The only processing done by the autopilot system is the conversion of the temperature and pressure readings from integer to decimal values.

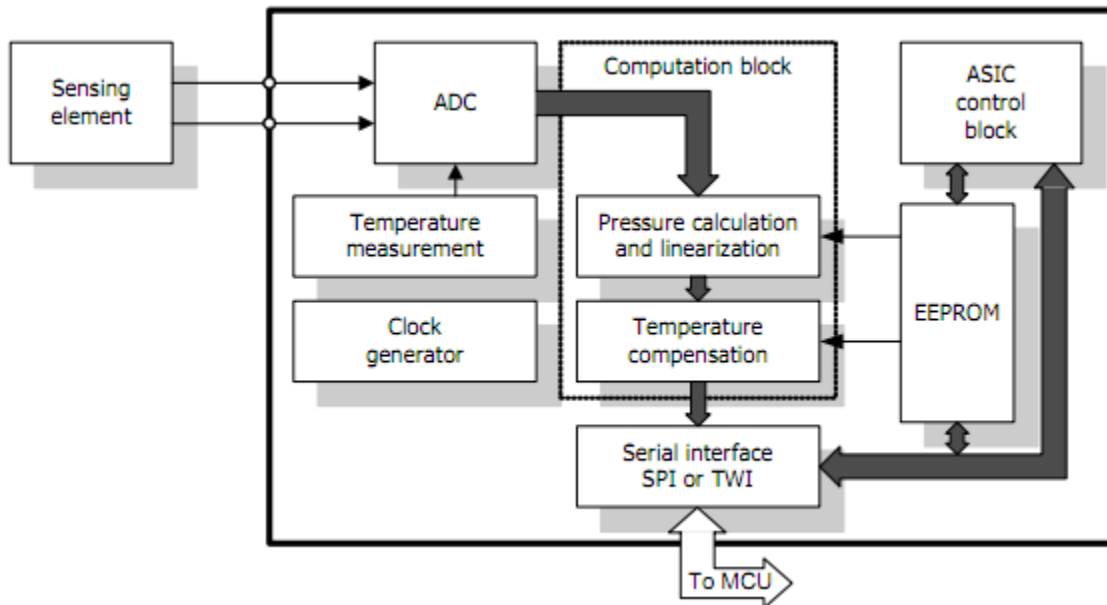


Figure 8 - Image of Pressure Sensor

The SCP-1000 communicates through either SPI or I2C. It can be configured to throw interrupts for a variety of pre-programmed and user defined conditions. The chip can also be configured to simply store data and provide it to another chip on request.

## 5.5 Maxsonar EZ4 Ultrasonic Range Finder

The XL-Maxsonar EZ4 is a ultrasonic range finder that will be incorporated into the project at a later date. The EZ4 generates a high power, narrow beam of sonic energy, and then measures the returning echo. This narrow beam width allows it to accurately measure the distance to even small objects and the high power allows the object to be detected at a greater distance. It's maximum range is 765cm with a resolution of 1cm. The EZ4 operates at up to 42KHz and communicates serially at a speed of 9600 baud using RS232 protocols. Each time the EZ4 takes a range reading it calibrates itself. The sensor then uses this data to properly range the distance from the ground.

This sensor will be employed to give very accurate information about the elevation of the aircraft when it is coming in to land and taking off since the GPS can only give a elevation to about 10 meters of accuracy. When trying to land an aircraft moving at high speed on a possibly uneven surface a much more accurate system was needed and the EZ4 fills this role perfectly.

## 5.6 GPS-Venus634LPx GPS Board

The Venus634LPx board is a complete GPS reviver system. It is capable of 51 Channel acquisitions and 14 channel tracking. The board is equipped with an antenna connection plug and is capable of measuring the location of a connected antenna to within 2.5m. It is designed to be operated at up to 18 km in altitude and up to 515 m/s. It communicates using the R232 communication protocol using the NMEA-0183 protocol. The baud rate can be set to 4800, 9600, 38400, 115200. When started it is able to lock onto GPS satellites and usually can provide a fix a position within less than a minute.

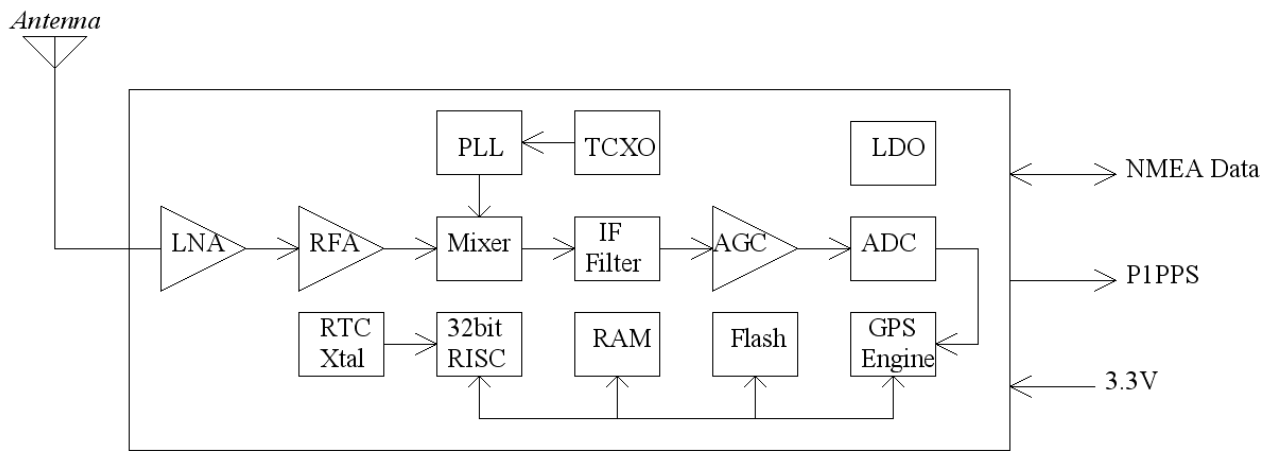


Figure 9 - Image of GPS

The Venus module is connected to the main autopilot system power via 3.3V power, the return ground and a serial transmission and a serial receiver line. In order to speed up the time to get an initial fix from the GPS system, the module also has an auxiliary power pin which allows it to be connected to either a battery or a backup power system. When this pin is used the time it takes to get an initial fix is reduced from a few minutes to about 1 second. The backup power keeps the internal flash memory powered and so when main power is restored it uses the stored information to regain GPS lock. This is called hot starting.

Before the transmission of a data stream the Venus outputs a header which allows the autopilot system to determine which data group it is looking at and parse the data accordingly. The GGA character string gives the autopilot system sufficient data to know it's latitude, longitude, header, altitude as well as the Universal Time and the number of satellites being tracked.

## 5.7 Pitot Tube

The Eagle Tree Systems Airspeed MicroSensor V3 is an instrument that uses a Prandtl style pitot-static tube to measure airspeed. As with the EZ4 sonar sensor the pitot tube is intended to be incorporated into the project at a later date. The measurement apparatus of the V3 consists of a long metal tube with a pressure sensor at the end. As air enters the tube it changes the pressure in the tube and exerts a force on the sensor. The pressure sensor then outputs a differential voltage which can be used to determine the velocity of the plane relative to the air. It is capable of measuring velocities from 3 to 564 KM/h.

The pitot tube will be incorporated into the plane to give a very accurate air speed. Although the GPS can give a velocity relative to ground this is not as useful because a plane flying into a strong head wind could appear motionless with respect to ground, yet still be moving at a high speed.

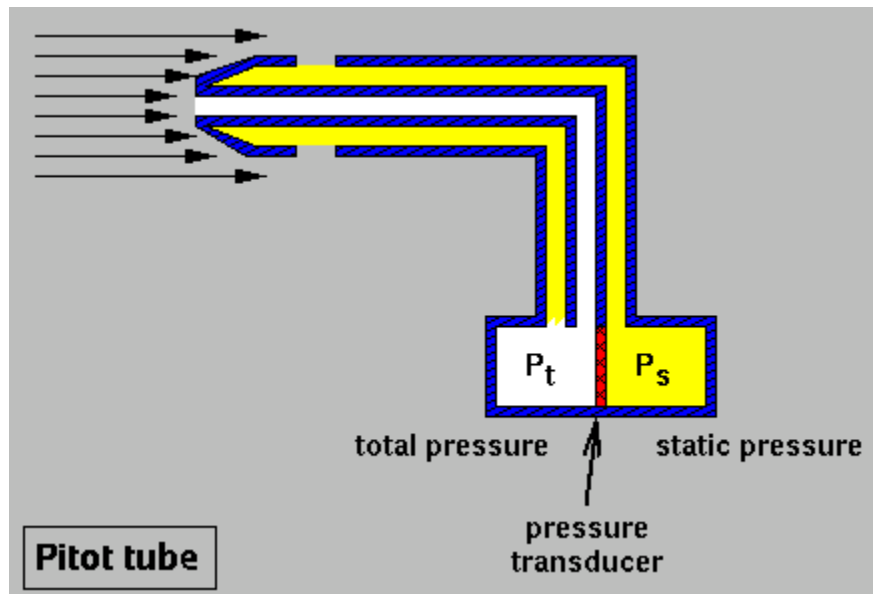


Figure 10 - Image of Pitot Tube

## 6 Communication System

In order for the FLOCS system to display orientation and position data back to the user, a rugged and flexible communications system was needed. This system consists of a four key components

- On board Ethernet connection
- Wireless Transmitter (Ubiquiti Bullet)
- Wireless Receiver (Ubiquiti AirGrid M5)
- Computer with Ethernet connection

Shown below is an overview of how the components are connected together in the final project. The section in orange is the portion on the transmit side, and the green section represents the receive side of the system.

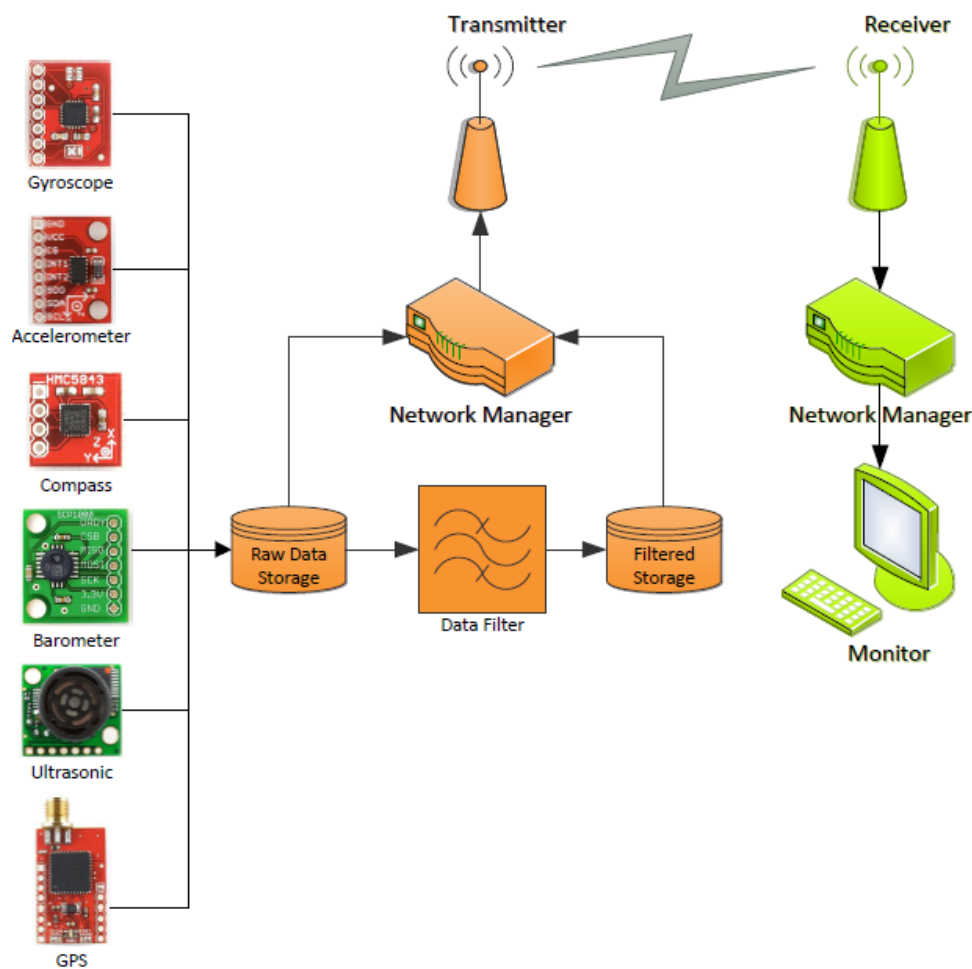


Figure 11 - Overall System Workflow

### 6.1 On Board Ethernet

The on board Ethernet port is used to send out data packets from the board to the rest of the system. As discussed in the sensor package section 5, the Ethernet port is mounted directly on the board and contains its own oscillator. Since the Ethernet port configuration was identical to the EVK1100 Demo board system, we were able to make use of the lwIP port to the AVR32.

The lwIP system is a lightweight implementation of a TCP/IP stack that has been ported to the AVR32. The AVR32 Studio software contains a demo application that makes use of the lwIP stack to implement a basic TFTP server, so this demo was used as a base to develop the on board communication system. The FreeRTOS scheduling system was used to create a separate task that was capable capturing the position and orientation data, encapsulate that data into a C structure, and passing that information out over the network. In order to provide maximum flexibility for the system, the data packets were sent to the broadcast address on the network so that any computer on the same subnet could receive the data packets, thus allowing multiple users to view the data simultaneously.

The decision was made early in the project to make all network communications work over UDP rather than TCP or other communication protocol. UDP was chosen for its lightweight characteristics and flexibility. Since the FLOCS system will ultimately be used as an autopilot having the most up to date information available all the time is paramount to the communications system. Since UDP is connectionless and doesn't retransmit dropped packets the data coming into the monitor system is guaranteed to be the most recent. This decision also made sense from a resource utilization perspective, since no network session needs to remain open (unlike TCP connections) and needlessly use extra CPU cycles.

## 6.2 Wireless Transmitter

The wireless transmitter used in the system was the “Bullet M5” transmitter from Ubiquiti Networks that was purchased by the UVic AERO club. This 5 GHz transmitter was powered using Power Over Ethernet (POE) and has an average transmit power of 22 dBm and a maximum power consumption of 4 Watts. With a rugged plastic cover and a weight of only 0.18 kg this transmitter was ideal for long range outdoor communication. We chose to use this transmitter mostly out of convenience. The transmission specifications met our needs and the equipment was already available. An image of the transmitter is shown below in Figure 12.



*Figure 12 - Image of Bullet Transmitter*

The transmitter was connected to the receiver using a wireless bridge network. The transmitter was configured as a “WDS Station” and the receiver as a “WDS Access Point”. WDS stands for Wireless Distribution System and is a type of wireless network that allows a network to be shared between multiple access points without needing to connect them with a physical link. This network



topology was chosen so that data packets could be sent over the wireless link without any modification and so that the board MAC address was preserved while sending packets.

Another important factor we considered when designing the communication system is the transmission range. The transmitter is rated at being able to transmit at 54 Mbps at over 50 KM, however with the small antenna mounted on our transmitter we were hoping to achieve a few KM transmission range. We were able to test the affective range of the communication system by positioning the receiver on the top of Mount Doug and the transmitter approximately 1.3 KM away on Blenkinsop road. By pointing the receiver at the transmitter we were able to receive data packets while monitoring the connection strength using the built in receiver web-interface. We were able to achieve a result of approximately 15 dB above noise at that range and considered the test a success.

### **6.3 Wireless Receiver**

The 5 GHz wireless receiver used for the project was the Ubiquiti AirGrid M5 receiver. This transmitter/receiver makes use of an 11" x 17" antenna and can transmit data at a maximum power of 25 dBm. Similar to the transmitter, the receiver is powered using POE and come standard with mounting brackets that allow the unit to be easily mounted and positioned on a variety of platforms. For the scope of the FLOCS project, the receiver was mounted on a tripod that was manually positioned to achieve a connection. In future implementations, the receiver will be mounted on a pan-tilt platform that will automatically track the boards position while mounted on an aircraft. An image of the receiver is shown below.



*Figure 13 - Image of AirGrid M5 Receiver*

As mentioned as in the previous section the receiver was configured as a WDS Access Point. This configuration allowed us to connect a computer directly to the receiver and receive packets

regardless of computer platform or configuration. The receiver also used the 5 GHz spectrum and we were able to configure it using the on-board AirOS software to tune the connection parameters.

#### **6.4 Computer Ethernet Connection**

The final step in the communications system is the computer connected to the receiver via a standard Ethernet port. By ensuring that the computer was on the same subnet as the board and transmitter we were able to receive data packets from the board using the broadcast address. Through testing we were able to connect to Windows and Linux based computers and receive data packets, though some firewall configurations were required depending on the Operating System.

In order to confirm that packets were arriving at the destination computer a custom C sockets program was written to receive broadcast packets on a specific port and print out the results. By running this program while the communications system was running, we were able to view data packets arriving on the host computer after they were sent. Once we had confirmed that data packets were reaching the host computer, we were able to allow the monitor to read the packets and display the results.

#### **6.5 Future Communications Development**

When developing this project in the future expanding the communications system will become very important. Currently the communication is uni-directional from the board to the monitor. However the communication hardware supports bi-directional communication. Since the end goal of this project is to build an autopilot, being able to send the system commands would be very valuable. Sending messages to add new way-points, update filter parameters, check the status of onboard components, and send emergency “Call Home” instructions could ultimately allow the system to be used successfully as the UVic AERO’s autopilot.

In order to allow two way communication a new communication “Receive” task would need to be created. Since the positional data is being sent out over UDP it would be simple to use the same protocol to receive messages as well, however this could cause reliability issues. With UDP there is no re-transmit mechanism so if any instructions were lost there would be no way for either side to know that they didn’t arrive. The receive system would most likely need to use a TCP/IP based communication system that supports message retransmission to prevent data loss.



## 7 Printed Circuit Board Design

The layout of the autopilot board has some additional functional design criteria outside of the project specifications to meet mechanical constraints and proposed power distribution of the overall aircraft system to interface with the autopilot.

The autopilot board is powered by a 5V DC power supply which is then linearly regulated to 3.3V and distributed across the board. Because of the complexity of the board, a 4 layer PCB was designed. The top and bottom layers are signal layers and the internal layers are ground and power layers. The board has dimensions of 4" x 6" to allow ample amount of room for all the required connectors. An image of the PCB design is shown below.

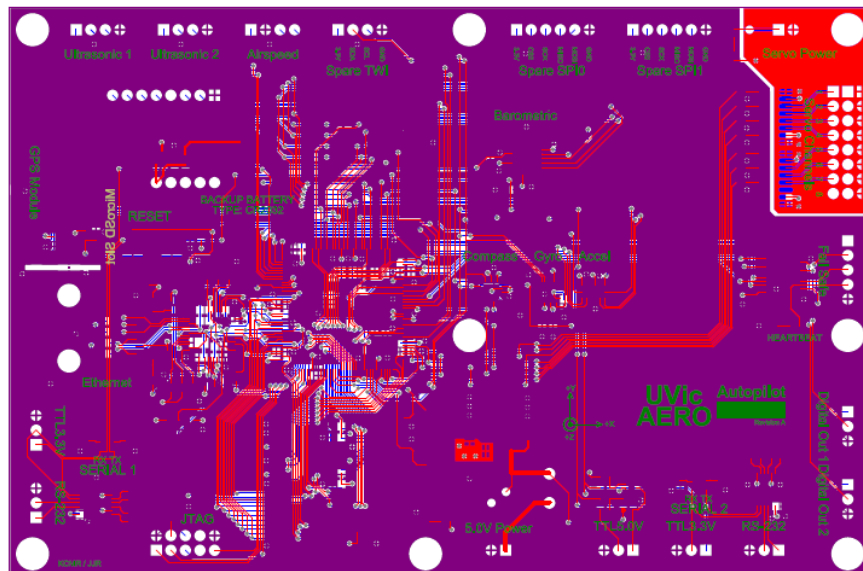


Figure 14 - PCB Design Image

An 800mA fuse was placed on the input power side to prevent damage to the components on the board in the even of a short circuit or power surge. In addition to this fuse, a protection transient absorbing diode is in place to protect the board against large input transients on the power rail as well as a means to protect against reverse polarization of input power damaging the board.

The autopilot board is designed to accept the GPS module as a breakout board to avoid the need to carefully design the autopilot board to have proper impedance matching for the external antenna used for the GPS module. The autopilot board also has provisions to accept a lithium coin cell battery to provide backup power to the GPS module to allow for hot starting.

Serial communication ports have been made available for general purpose use if desired. a 3.3V TTL logic serial port, a 5V TTL logic serial port and RS232 logic serial port is made available to be used in general purpose use.

An additional I2C port and SPI communication port is made available to accommodate additional external sensors if needed. Note due to the rarity of the barometric sensor and its extreme sensitivity to heat one of the spare ports is used to communicate to the barometric sensor. This was a design oversight.

The ultrasonic sensor and pitot tube sensor are external to the aircraft and thus have designated ports to allow these sensors to be placed in the aircraft where convenient.

The main sensors that measure orientation for the aircraft have been placed as close to one another as possible and on a similar axis to minimize the complexity required in programming the autopilot system. All the sensors have been placed such that their orientation agree, in other words the x-axis of the compass, gyroscope and accelerometer all point in the same direction. The same holds true for y and z-axis. The chosen forward direction is along the x axis, pointing towards the front of the aircraft, y towards left (if you consider yourself placed inside the aircraft facing forward) and z towards the sky or out of the aircraft.

The EVK1100 dev board has an external RAM chip to provide extra memory when running the program, it was decided to include this RAM chip on the PCB because it is anticipated that when programming the autopilot to fly the aircraft autonomously that extra memory will be needed.

As standard per PCB design all integrated circuit (IC) components have appropriate de-coupling capacitors. The Ethernet jack used on the board has internal transformers to provide isolation between the board and the devices it connects to.

A micro SD card slot is included and can be written to over the SPI bus to locally store and log flight and GPS information if needed.

Optocouplers are used to buffer the outputs of the PWM signals to the aircraft control servos to prevent noise feedback from the servo motors creating noise issues.

A reset button is provided to allow restarting the system without the need to power cycle the entire board. During the construction phase of this PCB system a few issues were encountered with faulty pin soldering. Namely in first testing the the functionality of the PCB the external oscillator was not properly soldered to the AVR32 microprocessor.

In first programming the AVR32, it is required to erase the chip to unlock it. Once erased, the AVR32 can be properly programmed. In a second revision of laying out the autopilot board miniaturization will be the focus to allow for further flexibility in placing the autopilot system in smaller aircraft where space is limited. Additionally, provision will be made to accommodate a voltage and current consumption system to provide remaining flight time before battery life runs out. A fuel and estimated range calculation will be performed in the next revision.

# 8 Telemetry System

The telemetry system is where data is processed to determine the state variable of the system. The data is received by the sensors then filtered and computed, after it stores on the micro-controller and periodically sent to the monitoring system using UDP.

## 8.1 Design of the Telemetry System

In Figure 15 is the design of the micro-controller software. It shows how objects in the Telemetry System interact. In general, the design flows from the left to right. Initially the Sensor Monitor will create the Sensors, then keep track of the sensors status. Once sensors are running it will store its data in the Raw Data Storage using the Raw Data class. When the Filter Controller runs periodically, it will get the Raw Data from the Raw Data Storage and pass it to the current filter. Once finished computing the State Data it will store it in the State Data Storage. Periodically the Network task will access the Raw Data Storage and State Data Storage to get all the information, which will be packaged up and sent to the monitor.

FLOCS – Simple UML of the Object Layout and Communication model

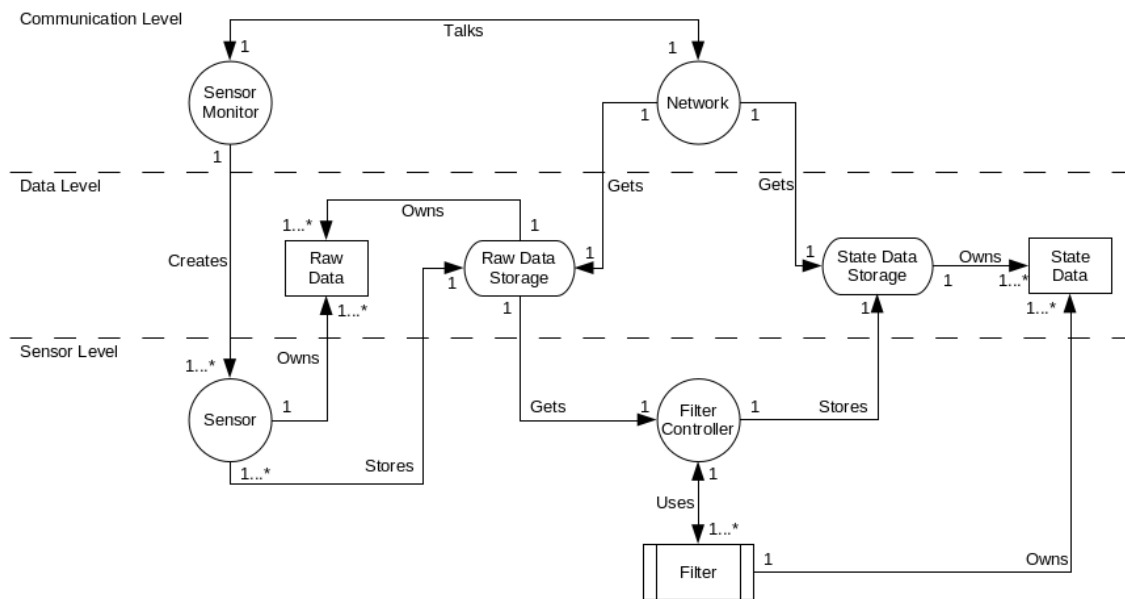


Figure 15 - FLOCS UML Design

### Sensor Monitor and Sensors

Inside the Sensor Monitor is a structure that holds all the sensors. To add a sensor to the Sensor Monitor you must call the add function and pass an instance of the sensor and the periodic task that will poll the sensor. When all sensors are loaded into the Sensor Monitor a function can be called to start all the sensor's tasks. Once the all sensors are started then the Sensor Monitor will monitor the sensors for any problems.

The sensors implement a Sensor interface that allows the Sensor Monitor to interact with it. In this interface there are functions for determining who it is and what its status is. Once implemented there a periodic task created to get the sensor data and store it in the Raw Data Storage.

#### FLOC – Sensor Monitor

#### FLOC – Sensor

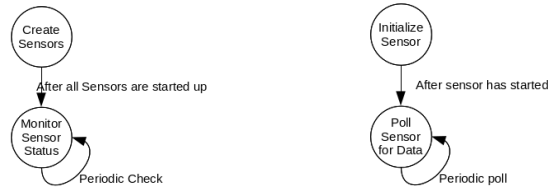


Figure 16 - FLOCS Sensor and Sensor Monitor

## 8.2 Data Storage and Data

The Data Storage and Data are both static classes for raw and state data. They are called by other functions and are used purely storing information for other objects to use. They will use mutexes to keep track of data, so that no one task can corrupt data or read it at inappropriate times.

## 8.3 Filter Controller and Filter

The Filter Controller will have the ability to switch which filter we are using on the fly. This is useful for when testing and not needing to reprogram the entire unit. If a switch happens it will allow the new Filter to reach steady state before allowing it to run. The Filter Controller will have a periodic task that will call the static Filter's step function that will run the filter for that period. There will be another function reset, which will reset a selected filter. This will be particularly useful so that when testing they system, the filters can be set to default values with out having to physically pressing a button.

#### FLOC – Filter Controller

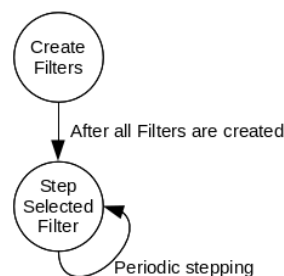


Figure 17 - Image of Filter Controller

## 8.4 Network

Network is a task that runs periodically and will take all the data from the system and pass it to a monitoring system or logging system via UDP. In the future, it will be able to received information and make system changes so that modifications can be made on the fly.

## 9 Implementation of Telemetry System

When creating the telemetry system it was chosen for it to be programmed in C++, because of the ability to easily implement the diagram above and easy for implementing and reading the code. The telemetry system was implemented in stages:

1. Setting up the FLOC framework
2. Creating Sensor Drivers
3. Creating Inertial Measurement Unit
4. Transmitting Data

### 9.1 FLOC Framework

To setup the FLOC framework AVR32 Studio was used as IDE, it came with the demo board and was made for developing code for AVR32 microcontrollers. It took some setup time to learn how they configured their settings for each project. Once the project was setup, using the frameworks included with AVR32 Studio, the FreeRTOS and the AVR32 drivers were added. This was a simple process since it was an added feature from the ATMEL, and improved development time. The lwIP library was added to communicate over Ethernet.

### 9.2 Sensor Drivers

Now that there was a framework setup, the sensors need drivers to run them. When creating the sensor drivers, the AVR32 drivers were used for the electrical interfaces. Once the sensor drivers were created, it was time to see if they concurrently with FreeRTOS. When the sensors were running, they stored data in the Data Storage that was implemented.

The sensor drivers that were created were the gyroscope, accelerometer, barometer, and compass. Problems arised when both the compass and gyroscope used the TWI. This was because the TWI port wasn't implemented to run on FreeRTOS and having two tasks using the TWI interfaces simultaneously caused errors. The gyroscope was chosen to run instead of the compass because it was more vital in determining orientation. The other sensor drivers weren't implemented because of time constraints. The accelerometer and gyroscope were the most important because they would best describe our orientation. Having the compass would of improved the output data because it would have reduced yaw error.

### 9.3 Inertial Measurement Unit

The Inertial Measurement Unit (IMU) is responsible for calculating the Euler angles of the aircraft. The implementation used during the demo came from the open source mbed IMU cookbook. This IMU implementation uses only the accelerometer and gyroscope data to compute Euler angles. There is a discussion about expanding the code to using a compass module to yield better results. That option makes use of a different coordinate system known as a Quaternion coordinate system.

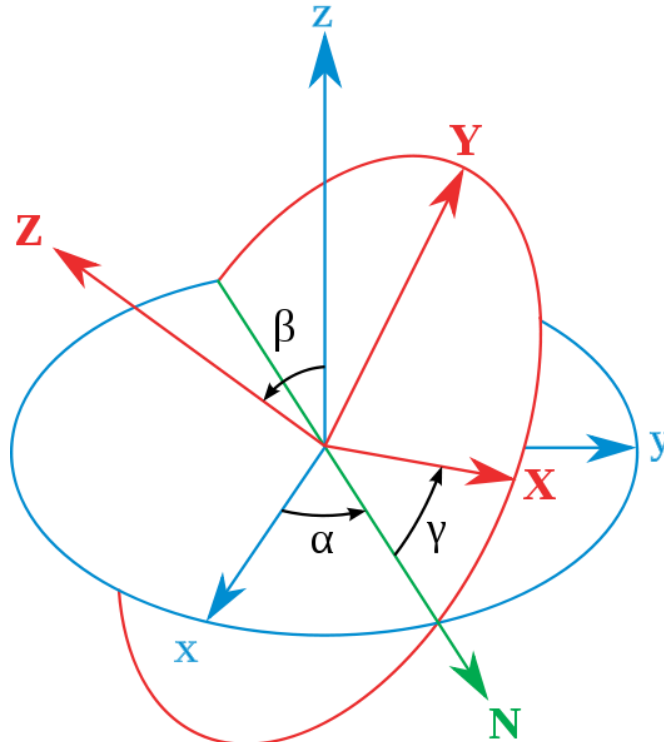


Figure 18 - Image of Quaternion Coordinate System

The Quaternion coordinate system is a 4 dimensional coordinate system which has a relationship to Euler angles in 3 dimensions. Since the gyroscope measures the angular rotational rate taking a gradient of the Quaternion variables maps to this information. Also using the accelerometer inclination data it too can be mapped to the Quaternion variables. By using a Jacobian Matrix to change the data from the 3 dimensional mapping to the Quaternion mapping allows the IMU to appropriately weight the sensor measurements.

Using the Quaternion to Euler angle equations:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) \\ \arcsin(2(q_0q_2 - q_3q_1)) \\ \text{atan2}(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

The IMU computes roll, pitch and yaw. The issue with the above equations is that there exists singularities as pitch approaches  $\pm 90^\circ$ . This was mitigated by applying an if state that limits the possibility of that singularity. The way the IMU deals with these singularities is it sets the pitch to  $\pm 90^\circ$  accordingly and the yaw to 0 degrees. Intuitively this makes sense as if the aircraft is point straight up there is no yaw component. The roll calculation is left untouched as an aircraft still has the ability to roll even when point straight up. As soon as some yaw is present the aircraft is no longer point straight up and thus the pitch is no longer at 90 degrees.

One of the issues with the open source IMU code that was implemented is to figure out what orientation is expected. During testing and the demo it was found that some of the axis were inverted and thus the system appears to be overly sensitive. The code also has provision for measurement error incurred from the gyroscope unit and uses this error definition as a means of changing the weighting on each sensor. It was found that approximately an error of 1 degree provides optimal results. Further investigation in using this filter code is required to better understand how it performs.

Before any of the raw sensor data is passed to the the IMU filter, the data is averaged over 4 sample values. This averaging reduces the sampling time of the IMU filter. However it should be noted that aircraft and mechanical systems are slow responding systems. For this reason it is assumed that an update rate of 40Hz is considered sufficiently fast enough for an aircraft. Validation of this assumption is required.

Other solutions to implementing the IMU exist such as using Kalman filtering to accurately predict what the Euler angles should be or MATLAB IMU implementation exists. However time has factored in the final decisions that were used for the demo.

Investigation and real flight logging needs to be performed to evaluate how well the sensors behave in real life situation. Further filtering may be required to reduce erroneous measurements.

#### **9.4 Transmitting Data**

Once the IMU was working, the Network task would poll the IMU state data and put transmit it to the monitor. This was done with the lwIP libraries initial added to the frame work. Please see Section 6 for more information.

#### **9.5 Implementing the Design**

Not all features were created in the design from Section 8 because of time constraints. The Sensor Monitor doesn't do any monitoring and there is no Filter Controller. All other structures exist as shown in Figure 15. Since there was no Filter Controller, the IMU filter described above was put into its own task and ran at periodic intervals.

# 10 Monitor

## 10.1 Overview

The monitor system was custom built from the ground up so that it could be completely tailored to the needs of the project. It is written in Java using OpenGL for the graphics and Javolution to parse incoming packets from the network. The system consists of a networking component to handle packets and a graphical component which the user sees. The graphical component includes displays to show the aircraft's orientation and directional velocity in 3D space, a 3D graph of its position, and all its orientation data in an intuitive dashboard display.

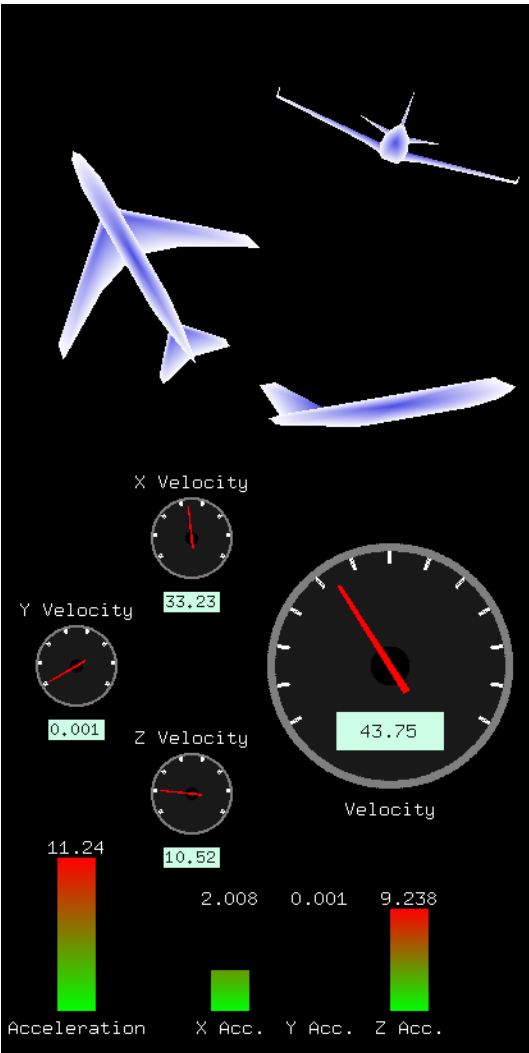


Figure 19 - Monitor Dashboard

## 10.2 Why OpenGL was Chosen

OpenGL was chosen because it is widely used in the industry and can be implemented in a variety of languages. The option to develop in Java was a major advantage for the developer, as the developer



is most comfortable with the language and has previous industry experience with it. OpenGL is open source, so one can trace through code as needed to gain a full understanding of what is happening. OpenGL does have a significant learning curve, particularly when it comes to emulating camera views, but this is inherent to any 3D graphics library other than those interfaced via GUI development tools. This was well worth the effort, however, as a lack of abstraction means fewer limits to options and customizability. It also ensures a complete understanding of how the visuals are set up, reducing the need to fix bugs later on. OpenGL is a very powerful development library offering the developer the ability to create nearly any effect needed, and as the monitor is intended to be adaptable to future project work, this sort of freedom is highly advantageous.

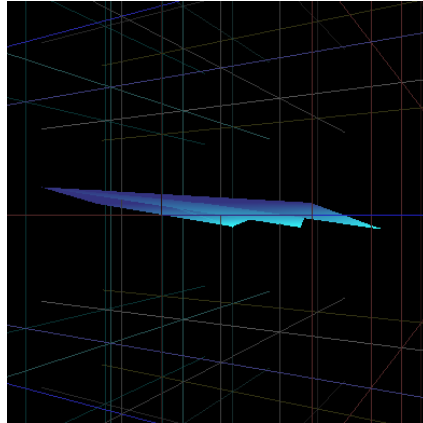
### **10.3 The Display**

The monitor is designed to display all available location and orientation information on the airplane in a most intuitive and visually uncluttered manner. It consists of a simple rendering of an airplane in 3D space, represented by 3D grid lines, a 3D position chart, and a large dashboard display. This allows for the user to look at whichever visualization they find most helpful. The dashboard has numerical data and intuitive ways to read them, while the position chart allows one to view the airplane's path. The 3D view combines some of the material seen on the dashboard so that the user can see how component orientation data combines to make up airplane's overall orientation.

### **10.4 The Dashboard**

The dashboard display is perhaps the most important part of the display as it provides the user with numerical data. As it is intended to show separate components of the airplane's characteristics, it is drawn with the 3D depth of OpenGL disabled. This results in a 2D display that the viewer can easily understand. It includes indicators for yaw, pitch, and roll, velocity, acceleration, wind speed and air speed. The yaw, pitch, and roll indicators are visual representations of an aircraft from the perspectives required to properly show their respective data. For example, the roll indicator consists of a visual rendition of an airplane viewed from the front, so roll can be indicated by rotating the plane. The pitch indicator is an airplane as viewed from the side, and the yaw indicator is an airplane viewed from above. Velocity is indicated in speedometer-style gauges, and acceleration in bar displays. Wind speed and air speed are currently displayed simply as numbers in boxes. Each component of the dashboard, except yaw, pitch, and roll, includes such a box so that the user can see the numerical data as well as just the visual indicators.

## 10.5 The 3D Rendition of the Airplane

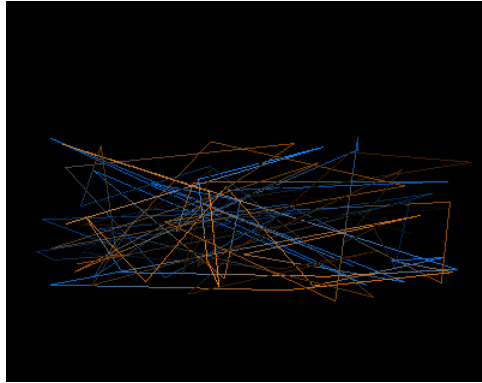


*Figure 20 - FLOCS 3D Display*

While the dashboard shows the user component data such as yaw or velocity on the x-axis, the 3D view of the airplane allows the user to see the airplane's orientation as a combination of all these components. As such, this is the best representation of how the airplane actually looks at a given time with respect to its orientation in 3D space. Yaw, pitch, and roll are shown together on one 3D model, where on the dashboard, they are shown as individual components. This 3D representation also allows the user to see velocity as the airplane actually flying in its window. This effect of movement is created by moving the gridlines in the graph past the airplane such that it looks as if it is flying. There are gridlines in each of the three axis directions, so velocity in any direction can be observed. As all component velocities are observed at once, the airplane's overall direction can be observed in this manner.

## 10.6 The 3D Position Graph

The 3D position graph provides the user with a way to get a sense of the shape of the overall path the airplane is following. The position graph is generated by plotting a line path joining every stored position data point that has been received over the network. Position data points are stored in a queue so that the points the airplane has visited can be graphed in order. The queue is set to a fixed size and wraps around so that there is a limit to how many position data points are stored. This is to reduce memory usage. The scaling of the graph is such that it does not respect an aspect ratio of the axes, but it could be easily modified to do this. Depth is indicated by the color of the graph. A more orange line means the airplane is closer to the user, while a more blue line means the airplane is farther away from the viewer.



*Figure 21 - FLOCS Position Graph*

## **10.7 Code Adaptability**

The monitor is intended to be highly adaptable to a particular use and so it has been implemented in a manner that supports this. This is accomplished via the use of constants and a selection of custom built display components.

One such constant is a scaling constant applied to the 3D space representation of velocity. Velocity of the airplane, as represented in the 3D rendition of the airplane, is not so much necessarily to scale with the plane as it is a good representation to show relative component velocities. For example, if the airplane has high forward velocity and minor upward velocity, it will look as if the airplane is slowly climbing. The 3D display shows this sort of relative motion well. Pure speed of movement, however, would have to be represented relative to the size of the airplane. This can vary and so there is a scaling factor, defined as a constant, applied to the velocity. The speed at which the grid moves can also come down to preference, and so having an adjustable scaling constant allows for this. Because the scaling factor is defined as a constant at the top of the file, it is easy to change and experiment to find what looks best for the application at hand.

There are also constants defining how components are displayed on the dashboard. These include position offsets defining where the various dashboard components are displayed in the GUI. There are also maximum value constants for velocity and acceleration. They represent the maximum values anticipated from the airplane. This is to ensure the displayed amounts do not “saturate” their respective displays. Acceleration is displayed as a rising and falling bar. The bar height is fixed to the maximum acceleration constant. If the actual acceleration exceeds the maximum, then the bar will simply not draw higher than this point. With well-chosen maximum constants, saturation should not occur often enough to make the display visually confusing. If the bar still frequently saturates, the actual numerical value is displayed at the top of the bar so the user can still see how large the value is. For velocity, values are displayed as gauges similar to a speedometers. Saturation in their case would be the needle all the way to the right.

Gauges, bars, and numerical display boxes, the currently available display formats for the dashboard, can be added to the display with ease. They can be initialized with values defining their size, display name, maximum anticipated data values, and whether or not negative values are to be displayed in the numerical display box (gauges and bars also have a numerical display box). If

negative values are not to be shown, the absolute value of the number will be shown instead. This can be valuable when the user simply wishes to know the magnitude of the x-component of the airplane's velocity but not the x-component of the airplane's direction, for example. With this freedom, these components can be used to represent any type of numerical data. For example, bars are not limited to representing acceleration, despite this currently being their sole use in the FLOCS monitor.

Initial scaling of components is implemented such that visual information is not lost when the component is too small. For example, a large velocity gauge has its numerical display box located on the face of the gauge, similar to some speedometers. However, if a velocity gauge is initialized at or below a certain size, the numerical display remains large and is placed below the gauge. Additionally, gauges are drawn such that there is an appropriate number of hash marks on the face with regards to maximum anticipated value the gauge will display. For example, if the maximum value is 50, there might be 50 hash marks so the user can see where the needle is pointing. If the maximum value is 2000, however, 2000 hash marks would simply look like a solid line around the perimeter, so it might look better to have hash marks per every 50 units rather than every 1 unit of velocity. This hash mark scaling is not quite fully implemented yet; it does not look correct at some settings.

## **10.8 Future Work**

There are many opportunities for future work on the FLOCS monitor. The developer had no prior experience using OpenGL, and it certainly takes time to develop a proper mental model of how 3D space is emulated, as well as to learn the typical conventions that make code easier to understand and evolve. With more time to learn OpenGL, the developer would certainly have a better grasp of the sorts of bugs that were encountered during development and be in a better position to maintain and add to the code. Additionally, due to the nature of software it is always possible to improve the system, making it more adaptable, optimizing, and refactoring where appropriate, among others. For this system, there are some areas in particular that could be improved.

The frame containing the 3D rendition of the airplane could use some minor improvements. Currently, the user can rotate the view, effectively orbiting around the scene, by use of the arrow keys on the keyboard. The user can orbit laterally around the plane using the left and right arrow keys and orbit over or under it using the up and down keys. This becomes counter-intuitive, however, as the current implementation causes the everything to rotate including the user's reference axes. This results in the user pressing the arrow keys and seeing the camera orbit around different axes than expected. The desired solution would be for the user's reference axes to reset as soon as the user releases all arrow keys. This would mean an intuitive rotation the next time an arrow key is pressed. Another helpful addition would be to include a compass indicator in one corner of this frame. As the user orbited the camera around the scene, this compass would rotate accordingly to indicate the current camera orientation. This compass would also be highly valuable in indicating the airplane's geographic orientation.

The 3D position graph is implemented such that it always scales to fit within the visible frame. The aspect ratio of the graph is not maintained during scaling, however. Because this is a position graph,

it does not make sense to stretch or compress the graph on either axis independent of the other. This results in the illusion of the airplane moving faster or slower depending on which axis it is travelling on, similar to the effect obtained by watching a 4:3 aspect video stretched to fit a 16:9 display. Ensuring aspect ratio is maintained when resizing the graph might be a simple, single parameter change, or it may require rewriting the code that fits the graph within the viewing area. With a fundamental understanding of OpenGL, which the developer now has, the fix should be trivial though the developer may run into issues not anticipated due to newness to graphics programming.

The 3D position graph could also be significantly improved by drawing an airplane at the leading vertex of the path. This would make it very clear as to where the airplane is, as sometimes it is not always easy to see where a path is drawing itself. The drawn airplane should also scale depending on how far away it is from the user (i.e. how blue the path is), to provide the user with even more depth information. This would be a very intuitive way to see how far away the airplane is. This has not yet been implemented due to the complications resulting from scaling the graph. If the graph is scaled, the drawn airplane will scale too. The developer would need to learn more about OpenGL in order to fully understand what happens in this scenario. An attempt was made to draw the airplane, but it appeared to be flashing somewhat and being scaled as the graph scaled. With more research, the developer could learn what is making this occur and how to resolve the issue.

The developer would like to enable the user to resize the monitor window such that it scales appropriately to the new size. The user would also be able to make the monitor viewable in full screen mode. Resizing the 3D rendition and 3D position graph displays is already fully implemented; these frames scale properly and maintain their aspect ratios. Resizing the dashboard would require the developer to alter how frames are set up and how components are arranged on the screen. Some brief investigation would be required to clarify differences in coordinate space between Java windowing and OpenGL, and how these may interfere. With this knowledge, some refactoring might be required to accommodate any changes in how the components are positioned on the dashboard. There might be calculations required to manually resize some components and position them appropriately as the overall dimensions of the dashboard change.

The developer was initially interested in graphing all data as well as displaying it in the monitor. The graph would appear as a separate window that can be resized or displayed in full screen mode. The developer has done research into what open source Java graphing libraries are available. Some are not well suited to graphing live data, as performance issues will be encountered in such scenarios. JFreeChart is one such example. The developer decided that GRAL, which simply stands for GRaphing Library, would be best suited to the needs of FLOCS. The library includes numerous examples, including a live graphing demo, and can perform a wide variety of graphing tasks. Graphs created with this library are also highly customizable. Some graphs were created successfully but no applicable in-depth work was completed in time for this project. The developer also considered simply building a brand new graphing library using OpenGL in 2D, and now has enough experience to accomplish this, but it would have been too great an undertaking given the time constraints.

## 11 Conclusions

The FLOCS project was a success. By performing the sensor and microcontroller selection we were able to determine the optimal configuration for the telemetry system. The microcontroller chosen was an ATMEL AVR32UC3A0512 which was soldered onto our custom build PCB board. By using an Accelerometer, Gyroscope, Barometer, Compass, GPS, Ultrasonics, and Pitot tube we could measure the orientation, position, and velocity of the aircraft. We were able to successfully measure the Euler Angles using just the accelerometer and gyroscope.

The communications system was comprised of the onboard Ethernet port, wireless transmitter and receiver, as well as a computer with a standard Ethernet port. The system was able to achieve a high data rate at 1.3 km with a 15 dB noise to signal ratio with a handmade antenna. We were able to successfully send data packets from the board to the monitoring system through the communications system.

The monitor was able to successfully decode the data packets and incorporate the telemetry data into a user friendly graphical user interface. The monitor was able to show the 3D position of the aircraft, speed and velocity figures, as well as aircraft orientation.

## 12 References

“Pressure Altitude.” National Weather Service Western Region Headquarters.  
<http://www.wrh.noaa.gov/slc/projects/wxcalc/formulas/pressureAltitude.pdf>

Intel Embedded. April 1, 2011. Intel Corporation.  
[http://www.intel.com/p/en\\_US/embedded/hwsw/hardware/atom](http://www.intel.com/p/en_US/embedded/hwsw/hardware/atom)

ARM. April 1, 2011. ARM Ltd. <http://www.arm.com/products/processors/index.php>

PIC32. April 1, 2011. Microchip Technology Inc. [http://www.microchip.com/en\\_US/family/pic32/](http://www.microchip.com/en_US/family/pic32/)

32-bit AVR UC3. April 1, 2011. Atmel Corporation.  
[http://www.atmel.com/dyn/products/devices.asp?category\\_id=163&family\\_id=607&subfamily\\_id=2138](http://www.atmel.com/dyn/products/devices.asp?category_id=163&family_id=607&subfamily_id=2138)

Atmel Corporation, “AT32UC3A,” AT32UC3A Data Sheet, Sept. 2009, Rev I.

Atmel Corporation, “EVK1100 Getting Started,” User Manual, May 2005, Rev. A.

Analog Devices, “3-Axis,  $\pm 2$  g/ $\pm 4$  g/ $\pm 8$  g/ $\pm 16$  g Digital Accelerometer,” ADXL345 Data sheet Oct. 2010, Rev B

InvenSense Inc, “ITG-3200 Product Specification,” PS-ITG-3200A-00-01.4, Revision 4, March 2010

Honeywell, “3-Axis Digital Compass IC HMC5843,” HMC5843 Datasheet, Feb. 2009

VTI Technologies “SCP1000 Series Absolute Pressure Sensor,” SCP1000 Series Product Family Specification Datasheet, Rev 0.08

MaxBotix Inc “XL- MaxSonar®- EZ4™ (MB1240)XL- MaxSonar®- AE4™ (MB1340) Sonar Range Finder with High Power Output, Noise Rejection, Auto Calibration & Short-Range Narrow Detection Zone,” MaxSonar Series Datasheet, Aug. 2009

SkyTRAQ “VENUS634LPx 65 Channel Ultra Low Power GPS Receiver Datasheet,” Rev 0.5 Jan 2009

Eagle Tree Systems “Instruction Manual for the Airspeed MicroSensor V3,” Rev 1.7, 2008

“lwIP UDP Echo Broadcaster Example using Raw API, Socket or Netconn approaches.”  
UltimaSerial. 04/08/10. [http://www.ultimaserial.com/avr\\_lwip\\_udp.html](http://www.ultimaserial.com/avr_lwip_udp.html)

Ubiquiti Networks. April 1, 2011. Ubiquiti Networks, Inc. <http://www.ubnt.com/>

Barry, Richard. The FreeRTOS Reference Manual. Real Time Engineers Ltd, Version 1.1.0, 2011.

Barry, Richard. Using the FreeRTOS Real Time Kernel. Real Time Engineers Ltd, Version 1.3.2, 2010.

Aaron Berk. "IMU." mbed.org. Aaron Berk. 27 Nov 2010. mbed.org. April 1, 2011.  
<http://mbed.org/cookbook/IMU>

"Maths - Conversion Quaternion to Euler." EuclideanSpace. Martin John Baker. 2010. Martin John Baker. April 1, 2011.  
<http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/index.htm>

"Jogl 2.X API." Java.net. Rev. 2.0.0, July 2009. April 1, 2011.  
<http://download.java.net/media/jogl/jogl-2.x-docs/>

"JOGL Demos." Java.net. April 1, 2011. <http://download.java.net/media/jogl/demos/www/>



## **Appendix A – Project Data**

All project data is stored on the accompanying DVD.