Spring 2015

Disc Golf Locator

Noah M. Sanor
University of Akron Main Campus, nsanor@gmail.com

Disc Golf Locator

Final Design Report

<u>Design Team 11</u>

Shane Gamble, EE

Brandon Linhart, Cp.E

Noah Sanor, Cp.E

Christian Wallenfelsz, EE

Dr. Tsukerman, Faculty Adviser

4/21/2015

Table of Contents

List of Figures

List of Tables

Abstract (CW)

Disc golf is a game similar to traditional golf where players throw small plastic discs into chain-link nets. Disc golf courses cover several acres containing lakes, small wooded areas, large bushes, and grassy fields. It is not uncommon to accidentally throw a golf disc into the woods or bushes, so it is the goal of this project to create a device to locate the disc and make suggestions for the player to improve performance. A small device will be attached the disc which will track its location and flight characteristics. The device will contain a GPS receiver, an inertial measurement unit (IMU), data storage device, wireless transfer device, and an audio alarm to locate the disc. The GPS will record the flight path of the disc and the IMU will measure flight characteristics which will be stored locally on the disc during flight. After the disc is thrown and recovered, players will be able to use a smartphone app to retrieve the flight data from the tracking device by wireless communication. The smartphone app will plot the flight path on a map and analyze the inertial data to make suggestions for players to improve their throws.

1. Problem Statement

Need Statement (CW)

Disc golf is a game very similar to traditional golf. In disc golf, a player attempts to throw a small plastic disc into a slightly elevated chain-link cage rather than using clubs to hit balls into holes. Disc golf courses consist of numerous holes and can cover a respectably large area similar to a traditional golf course. It is common for disc golf courses to run through wooded areas with large amounts of foliage and brush. It is also common for a hole on the course to not be visible from the throwing location due to buildings, trees, or even elevation (throwing up a hill). These obstructions causes great difficulty in retrieving discs when they are consequently thrown into bushes or other foliage because it may not always be possible to see where the disc lands. Many hours can be spent searching through woods to find a lost disc and players will usually get frustrated and give up searching. Lost discs and time wasted detract from the player's enjoyment of the game. These unfortunate circumstances demonstrate a need to develop a system a player can use to easily and quickly locate a disc after it is thrown.

Objective Statement (CW)

The objective of this project is to create a system a disc golf player can use to track the location of a golf disc after it is thrown. The system will consist of a small devices which can be placed on the disc and software which can map the flight of the disc. The devices on the disc will log the GPS position of the disc and sound an alarm after a short period of time once the disc has been thrown. The alarm can be used to locate the disc audibly. The software will map the flight path of the disc, log throwing statistics, and then display recommendations to adjust the throw for players to improve their performance.

Modification of Project Operation (CW)

Initially, the project was based on RFID. The theory was to use a small passive RFID tag and use multiple readers to calculate and display distance and direction of the tag from a master control station. However, upon more research into RFID systems, passive RFID tags were discovered to operate within a range of a few meters. So research shifted focus into active RFID systems. In active systems, the tag contains a microchip, antenna, RF module, on-board power (usually a battery), and any other electronics for various purposes, whereas a passive tag mainly consists of a microchip (Lahiri). An active tag is capable of communicating over long distances depending on the application. While the range for the project would be satisfied with an active tag, it is not clear whether weight and size of active tag would allow the project to work. Furthermore, the project is meant to locate an object thrown arbitrarily into a wooded area with thick brush, weeds, bushes, etc… The presence of unknown physical objects ranging in size and location could potentially hinder radio based location devices due to multipath, reflections, and other potential interference. Therefore, it was decided to find an alternative method to locate a golf disc. The changes throughout the entire proposal were to eliminate ideas based on an RF device.

The original concept of operation was to use the Friis equation to determine distance from the tag on the disc. The Friis equation,

$$P_R = \frac{P_T G_T G_R \lambda^2}{(4\pi)^2 R^2}, \tag{1}$$

can be used to determine the distance between the transmitter and receiver provided the gain of the transmitting and receiving are known as well as the power transmitted and received (Levis, et al). However, the Friis equation is valid for free-space unobstructed transmission with no noise or interference. There are variations of the Friis formula to include noise, provided the noise parameters are known. For this project, the noise parameters would not be known and estimating distance based on received power would be larger because the power received would be smaller from noise. Therefore, alternative ideas were researched in order to facilitate locating a disc.

Research (CW)

GPS

Today, Global Navigation Satellite Systems (GNSS) are ubiquitous throughout everyday life. GNSS is used in everything from cell phones to cars. However, satellite navigation was developed and used after ground based systems were used. Such ground based systems like the British DECCA and US LORAN (long range navigation) systems were developed during WWII. These early systems used LF radio signals from known locations to geo-locate the position of receiver stations. LORAN receivers were open to public use after WWII and a modified version of LORAN, standardized as LORAN-C, was used into the 1980s. Although, the cheaper and more accurate system GPS took over the commercial market (Chen, et al).

There are a few main satellite systems in use today. The Russian GLONASS constellation consists of 20 working satellites from the late Soviet era. The European Union operates the Galileo constellation and China has recently started to implement their BeiDou constellation. The United States operates the oldest working GNSS which is the Global Positioning System (GPS). The GPS constellation consists of 24 satellites which are in geo-synchronous orbit to provide constant and even coverage across the globe. The GPS system is based on a geo-location method called Time Delay Of Arrival (TDOA). TDOA works by knowing the time and location of a transmitter. Then a hyperbola of possible locations can be calculated from receiving one signal. GPS needs at least three different signals to provide latitude, longitude, and a fourth signal to provide altitude (Petrovski).

Unfortunately, the world is not a nice sphere. The world's actual shape resembles an oblong ellipsoid. The most accurate coordinates system to resemble the Earth is the World Geodetic System of 1984 (WGS84). The reference frame of the WGS84 model is Earth-centered, Earth-fixed (ECEF), meaning the *xyz* position of *(0,0,0)* is the center of mass of the Earth. The z-axis points up through the North Pole. The x-axis points out through the prime meridian at 0˚ longitude. The y-axis points out through 90˚ E longitude. The axes rotate with the Earth as it rotates, so coordinates are constant. The WGS84 ellipsoid has the semi-major axis defined at 6378137.0 m and the semi-minor axis defined at 6356752.3142 m. Other parameters are defined for the WGS84 model regarding flattening and curvature. The GPS system uses the WGS84 model to describe latitude, longitude, and altitude (Acharya). In Figure 1, the relationship between ECEF *xyz* coordinates and latitude, longitude, and altitude are shown. The WGS84 is geodetic, so the latitude is measured from the surface of the Earth.

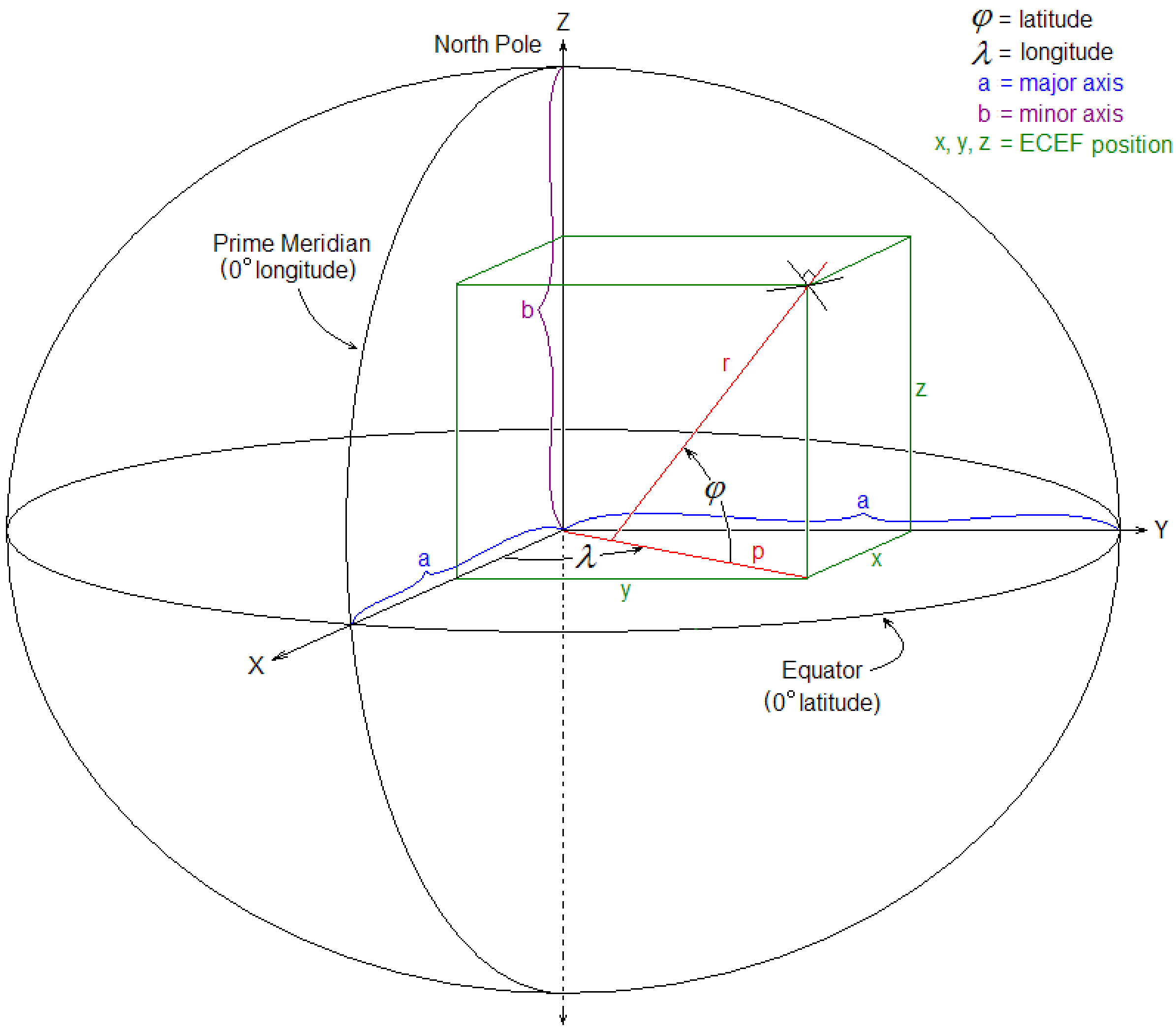


Figure 1: Cartesian Representation of an Earth-Centered Earth-Fixed Coordinate System

Micro Electromechanical Sensors

There are many different electronic sensors. Smaller sensors have become prevalent in many technologies used in everyday life such as phones, game controllers, medical devices, and even car tires. Many of these sensors are based on mechanical principles. The application of these principles in micro-electronics has introduced devices known as micro electromechanical sensors (MEMS). These sensors often utilize silicon structures to replace larger mechanical systems. In some modern MEMS devices, these silicon structures have been fabricated on the scale of 500 microns (500 micrometers).

Accelerometers

Acceleration is defined is defined in Newton's Second Law of Motion. The equation,

$$F = ma \qquad (2)$$

relates the force applied to an object by its mass and acceleration it experiences. If a known mass is used, the acceleration of an object can be calculated my measuring the force applied to it. Rather than using mechanical devices, MEMS technology generally measures a capacitance. By allowing a conductor to move between two fixed parallel plates with a known distance between them, the capacitance between the plates will produce a voltage which can be converted which will proportionally relate to acceleration. The transfer function of a MEMS accelerometer relating voltage to acceleration will take the general form,

$$H(\omega) = K\frac{1}{\omega^2}V\frac{dC}{dx} \qquad (3)$$

where *K* is a constant that will vary with each device and manufacturer, ω is the frequency, V is the voltage produced as the capacitance changes with position of the free conductor (Jones and Nenadic). The important element of Equation (3) is how the movement of the conductor will change a capacitance which can relate to acceleration. The specific formulas for accelerometers are often proprietary and differ with manufacturer.

Gyroscopes

Angular velocity is the rate of change in an angle between two axes. Gyroscopes can measure angular velocity based on torque and angular momentum. Torque is the measure of the force which will cause an object to rotate around an axis. Torque is defined as,

$$\tau = r \times F, \qquad (4)$$

where ***r*** is the distance from a reference where the force ***F*** is being applied. When a force is applied to an object, the resulting torque will rotate the object on a perpendicular axis to the force and distance vectors. Torque will cause angular momentum.

Angular momentum is measure of rotation of an object. It is defined as

$$L = r \times p \qquad (5)$$

where $\boldsymbol{p}$ is the momentum of an object a distance $\boldsymbol{r}$ from reference. Mathematically, angular momentum is very similar to torque. For rotational motional, the angular momentum can be simplified to,

$$L = I_o \omega \tag{6}$$

where I_o is the moment of inertia of the object and ω is the angular velocity.

The final concept for a gyroscope is the rate of precession. As an object spins around its axis, it will tend to rotate the axis. The rotation of the axis of the spinning object is called precession as demonstrated in Figure 2.

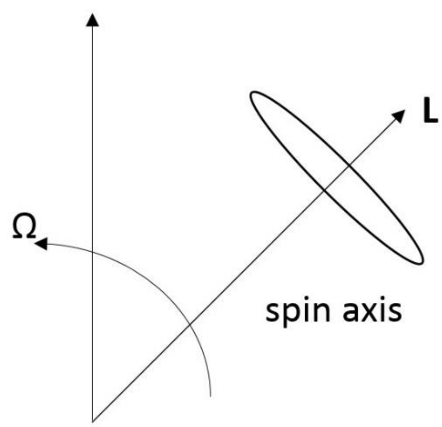


Figure 2: Precession of Rotating Object

Since angular momentum moves with precession and since torque is produced in the same direction as the rate of precession, the rate of precession can be related to angular momentum and torque (Kloppner & Kolenkow). Thus, from Equation 5 and Equation 6, the rate of precession Ω, is,

$$\Omega = \frac{rF}{I_o \omega}. \tag{7}$$

These basic principles guide the operation of a gyroscope. Clearly, by measuring the forces acting on the body, the angular velocity of the body can be determined. MEMS gyroscopes use micro-structures which do not spin, but compress or expand which causes a change in capacitance across the structure. Different manufactures relate this varying capacitance to the angular velocity of an object.

Magnetometer

For navigation purposes, orientation and direction are crucial pieces of information. Conveniently, the Earth produces a magnetic field which is mostly constant in direction. However, as illustrated in Figure 3, Earth's magnetic field does not directly align with geographic north.

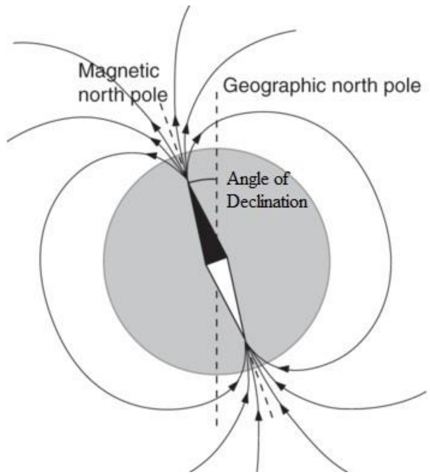


Figure 3: Depiction of magnetic declination

The angle of declination varies depending on the location of the measurement. A traditional compass may use an iron or a magnetic dipole which will align with the field pointing to magnetic north. MEMS magnetometers will measure the magnetic field intensity in different directions which can be used to determine heading from magnetic north.

Android Application (NS)

The Android platform was chosen to be used in this project because all of the team members own an Android smartphone, so the project could be tested by any person on the team. In addition, almost all Android smartphones contain a bluetooth antenna that can interface with the disc tracker. There are various cross-platform interactive development environments (IDEs) such as Android Studio and Eclipse to develop Android applications using an intuitive graphical interface. Another positive factor of Android is that the applications are written in Java. This is a benefit, because Java is one of the most commonly used programming languages around today.

The Java programming language is a high level, object oriented language that is used on various devices such as desktop PCs and smartphones. Java code is compiled to bytecode that is run on a java virtual machine (VM). Java is platform independent, because a VM can be installed on a supported system to run some compiled bytecode. Like many other programming languages, there are many libraries written for Java to greatly expand upon the functionality of the language (Lindholm, et al).

Android is an operating system (OS) that is built and maintained by Google, Inc. Many different types of devices can run Android, but it is most prevalently used as a mobile OS in smartphones. Since Android is built on top of the linux kernel, many of the system level tools available to desktop linux distributions are available to Android as well. Android runs a process virtual machine called Dalvik that utilizes Just-In-Time (JIT) compilation of Java code. Many of the wireless communication modules of the device are accessible through the use of built-in API libraries provided by the Android Software Development Kit (Liu and Yu). One example of an API that will be used in this project is the Bluetooth API that will be used to receive data from the disc tracker and send a signal to the tracker to signal the buzzer to emit a sound.

Flora Microcontroller (NS)

The Flora is an Arduino compatible microcontroller board that runs an Atmel ATMega32u4 at its core. This microcontroller board was designed to be used in wearable electronics. The Flora was chosen for this project for many different reasons. Since the Flora uses an AVR chip that is Arduino compatible, there are many AVR and Arduino libraries available for it. Additionally, this microcontroller is very small (4.445cm in diameter) and lightweight (4.4g), both of which are big constraints for the project (Adafruit).

Serial Interfaces (NS)

There are many different serial interfaces used in embedded systems today that all have different advantages and disadvantages. Based on the modules chosen to be used in the disc finder device, there are three serial interfaces that will be used in the project. The reason that there will be three separate interfaces used in the project is because the modules that were chosen, were primarily decided on based on power consumption, size and price. The serial interface supported by the device was not a major deciding factor. The main features of these interfaces are summarized in the sections below.

Universal Asynchronous Receiver Transmitter (NS)

The Universal Asynchronous Receiver Transmitter (UART) interface is commonly used in embedded systems to communicate between a single master and a single slave node (Mikhaylov & Tervonen). This interface operates in full duplex mode by using two communication lines. The transmit (Tx) pin of the master is connected to the receive (Rx) pin of the slave, while the Rx pin of the master is connected to the Tx pin of the slave.

Serial Peripheral Interface (NS)

The Serial Peripheral Interface (SPI) is a single master, multiple slave interface that provides full duplex communication between the master and a slave (Mikhaylov & Tervonen). Three lines are used across all connected devices. These lines are the clock (SCLK), master input slave output (MISO) and master output slave input (MOSI). Each slave node requires its own separate chip select (CS) line. The CS line needs to be pulled down before communication with a node commences. Since the chip select lines are active low, a pull-up resistor should be used to set the lines high when the slave is not in use.

Inter-Integrated Circuit (NS)

The Inter-Integrated Circuit (I^2C) Interface was created by Philips Semiconductor in 1982 (Mikhaylov & Tervonen). I^2C is a multiple master, multiple slave interface that uses two common lines across all devices: the clock (SCLK) and the data (SDA). A pull-up resistor is used on both of the lines. This interface uses a defined data format shown in Figure 4. An I^2C device first sends a start bit followed by a 7-bit address and then a read/write bit to specify the direction of communication. Next, data is continually transmitted until a stop bit is sent.

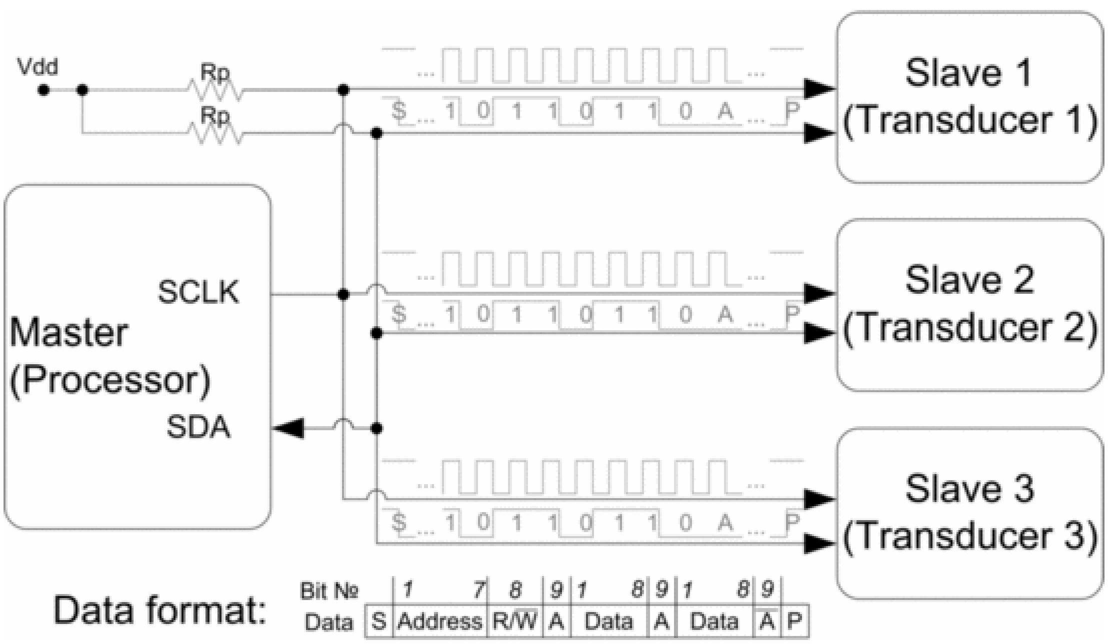


Figure 4: I2C Interface and Data Format

Marketing Requirements (CW, NS, SG, BL)

1. Minimally impact the disc's flight characteristics.
2. The system should be portable.
3. Operation in various temperatures.
4. The system should be simple to use.
5. Interfacing with a smartphone application.
6. The components should be attached directly to the golf disc.
7. The disc's motion should be trackable.
8. Audible within an average throwing range.
9. Electrical components should be very lightweight.
10. The flight path of the disc should be displayable on a virtual map.
11. Should provide recommendations to players for accurate throws.

Objective Tree (CW)

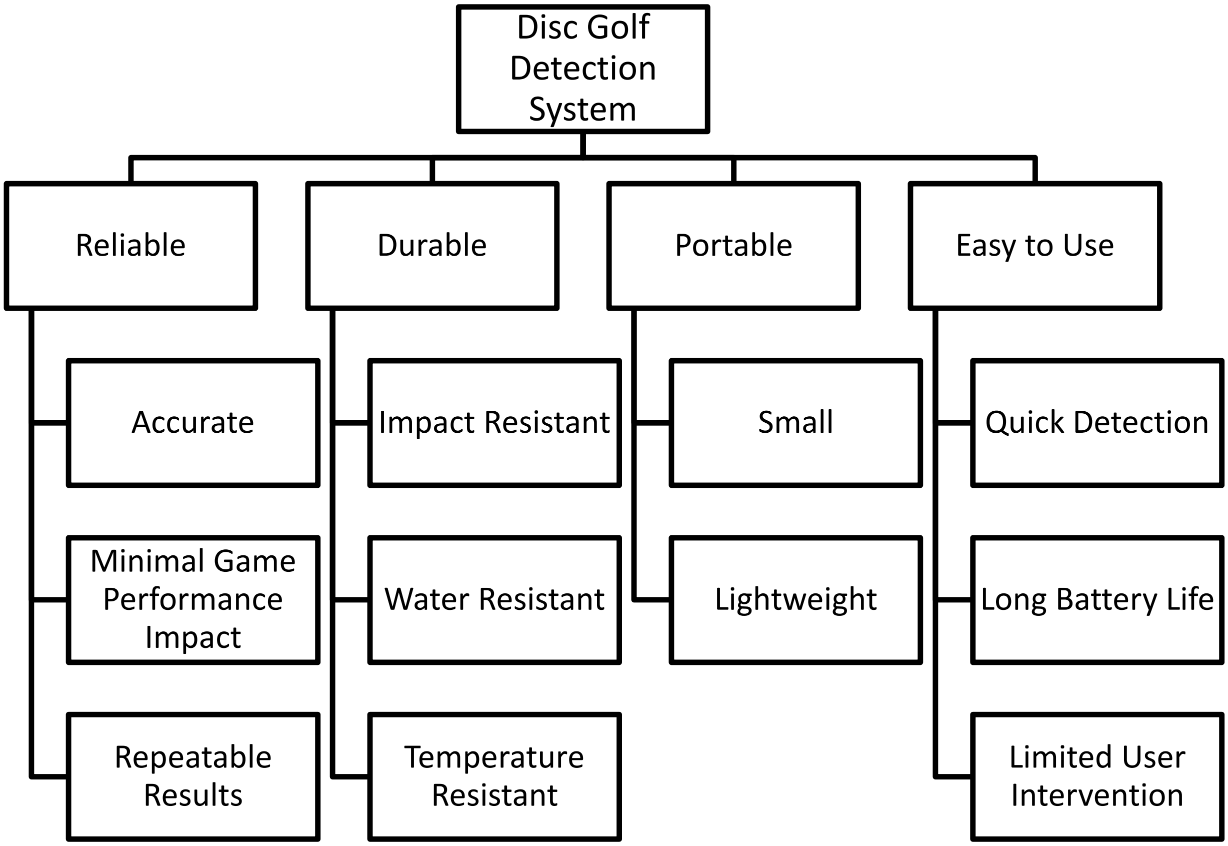


Figure 5: Objective Tree for the Disc Golf Locator

2. Design Requirements Specification (CW, NS, SG, BL)

Table 1: Engineering Requirements Table

Marketing Requirements	Engineering Specifications	Justification
1,2	Tracker will be at most 15.24 centimeters in diameter	This is the maximum size to reasonably fit on a golf disc
1,6,9	Tracker will weigh no more than 100 grams	Device weight added to weight of disc must allow it to glide
1	Tracker components will be mounted to evenly distribute weight	An imbalance in weight of the disc will alter its flight path
3	Tracker must operate within various outdoor temperatures from 0˚C to 40˚C	People may play in cool weather or high heat
5,10	Tracker will wirelessly send data to a smartphone	Limits user interaction
2	Tracker must operate below 5W of power	Maximum power required for sensors, data storage, wireless transmission
7,10	Tracker will use GPS	Record flight path of disc
8	Tracker will be able to produce a sound that can be heard from at least 10 meters away	Player must be able to locate the disc from a distance where it may not be visible
4,5,10	Smartphone application must be compatible with Android 4.3+ on all carriers	App will provide easy access for users
5	Smartphone application must be able to connect and disconnect from the tracker without crashing or disrupting the operation of the tracker	The tracker and application will be connecting and disconnecting multiple times throughout a game
4,11	Smartphone application will process and display flight data and make calculations for improvement	Easily provide feedback about throw to a user and

3. Accepted Technical Design

The system (shown in Figure 6) is centered around an Arduino-compatible microcontroller which runs at 3.3V and is supplied by a battery at 3.7V. This device was chosen for its capabilities in a very small, lightweight package. The controller takes in location and motion data from a GPS unit and an Inertial Measurement Unit (IMU). That information is filtered and parsed and stored in the microcontroller. After flight, the information is retrieved and sent wirelessly to a smart phone application over Bluetooth using a Bluetooth LE breakout module. The controller is also connected to a piezoelectric buzzer and triggers an audible alert for location. Basic flow of hardware connections is shown in Figure 7.

Hardware - Level 0 Block Diagram (CW)

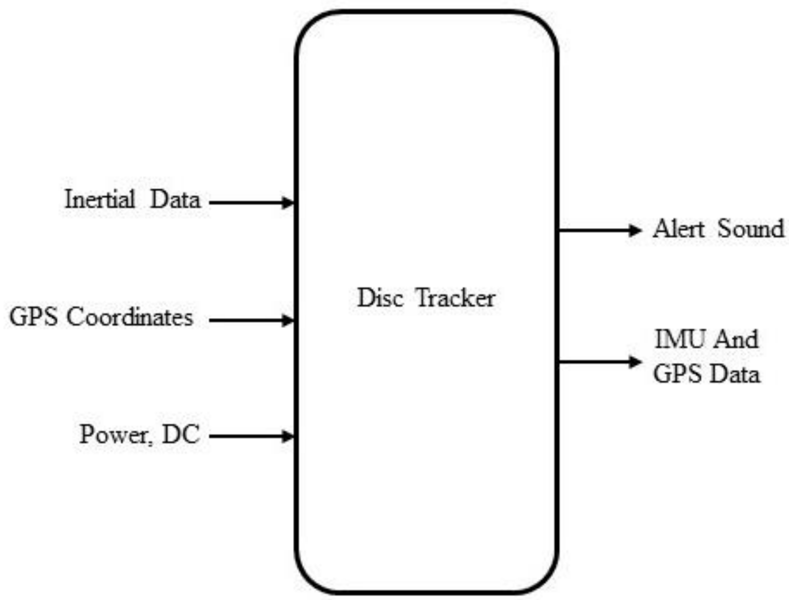


Figure 6: Level 0 Hardware Block Diagram

Hardware - Level 0 Functional Requirement Table (CW)

Table 2: Level 0 Hardware Functional Requirement Table

Module	Microcontroller
Inputs	• Activation • Power, DC • GPS Coordinates • Inertial Data

Outputs	• Alert Sound • IMU and GPS Data
Functionality	The device receives DC power from a battery. Upon activation, after which the disc is thrown, the microcontroller logs inertial data (acceleration, radial velocity, magnetic field intensity) and GPS coordinates. The data is then used in a smartphone application.

Hardware - Level 1 Block Diagram (CW)

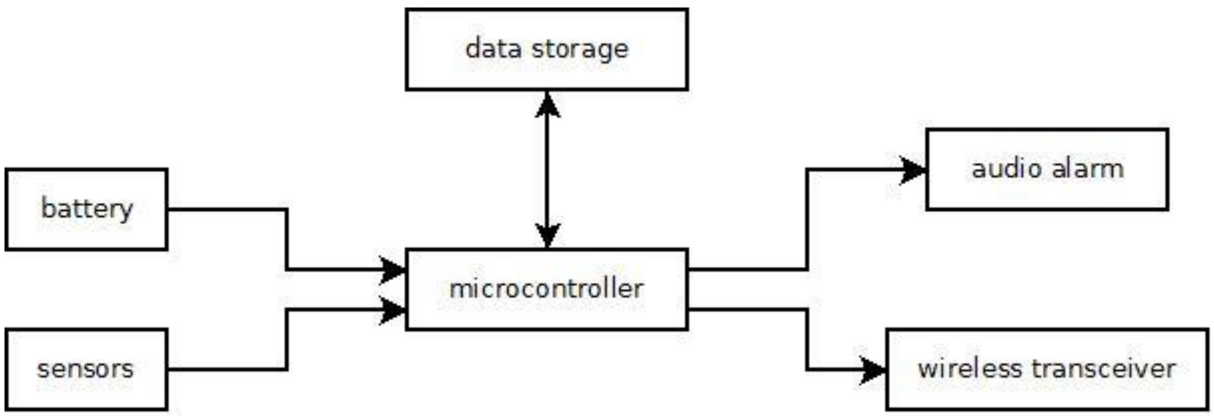


Figure 7: Level 1 Hardware Block Diagram

Hardware - Level 1 Functional Requirement Table (CW)

Table 3: Level 1 Hardware Functional Requirement Tables

Module	Battery
Inputs	• Power, DC
Outputs	• Power, 3.7 VDC
Functionality	The battery supplies power to the microcontroller and all on-board devices. It is recharged by an off-board charger.

Module	Sensors
Inputs	• Inertial Forces • GPS signals • Power, DC
Outputs	• GPS data • Inertial metrics (IMU data)
Functionality	The sensors measure GPS location data, inertial data and supply it to the microcontroller.

Module	Data Storage
Inputs	• IMU data • GPS data • Power, DC
Outputs	• IMU data • GPS data
Functionality	The data storage was intended to log the IMU and GPS data in real time during flight. The microcontroller could then retrieve the data when needed.

Module	Audio Alarm
Inputs	• Power, DC
Outputs	• Sound
Functionality	The audio alarm is triggered when device is ready to be thrown and after the device is thrown for the player to locate the device.

Module	Wireless Transceiver
Inputs	• IMU data • GPS data • Power • Wireless signal from smartphone
Outputs	• Wireless signal with IMU and GPS data
Functionality	The wireless transceiver communicates with a smartphone to transfer the IMU and GPS data stored on the disc.

Module	Microcontroller
Inputs	• IMU data • GPS data • Power, DC
Outputs	• Power, DC • IMU data • GPS data
Functionality	The microcontroller controls every attached device. It directly power each peripheral as well as send and receive data at appropriate times.

Hardware - Level 2 Block Diagram (CW)

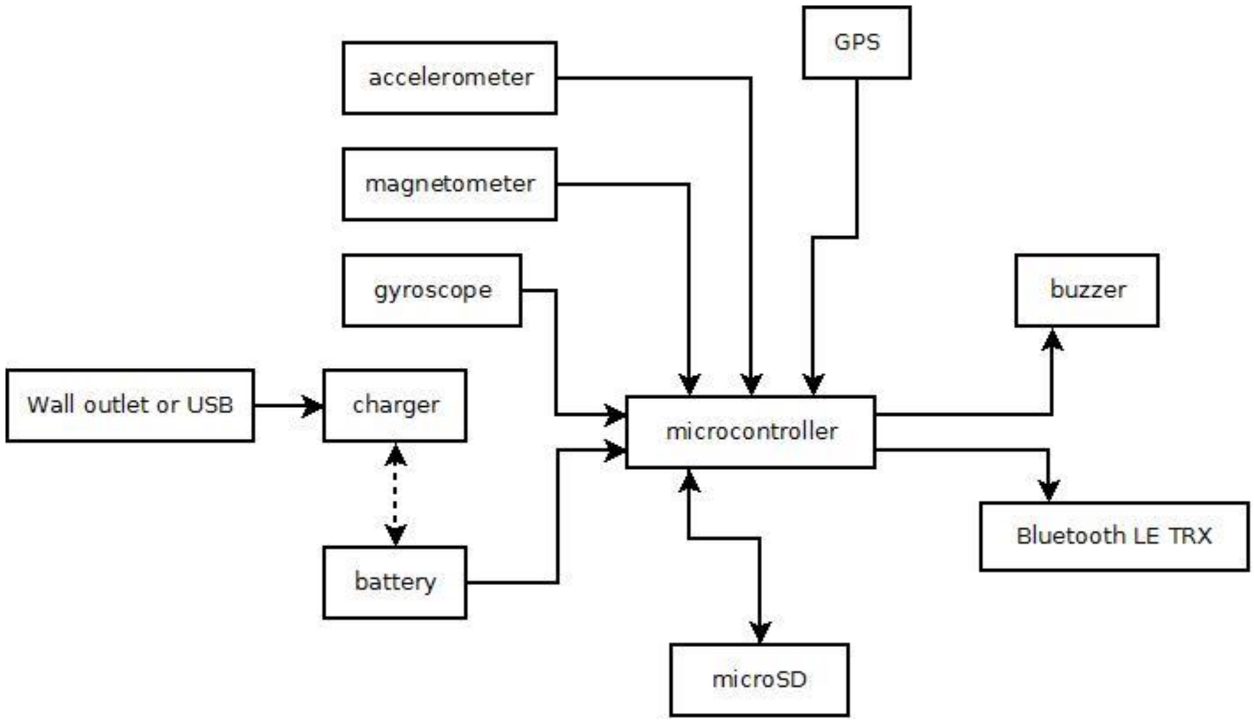


Figure 8: Level 2 Hardware Block Diagram

Hardware - Level 2 Functional Requirement Table (CW & SG)

Table 4: Level 2 Hardware Functional Requirement Tables

Module	Charger
Inputs	• Power, DC from supply • Power, DC from USB
Outputs	• Power, DC
Functionality	The off-board charger charges the battery when it is depleted and disconnected from the golf disc.

Module	Battery
Inputs	• Power, DC
Outputs	• Power, 3.7 VDC
Functionality	The battery supplies power to the microcontroller and all on-board devices. It is recharged by an off-board charger.

Module	Gyroscope
Inputs	• Rotational Force • Power, DC
Outputs	• 3-D Radial Velocity
Functionality	Measures the angular rate at which the device changed from its last position. Angular velocities are measured around the 3 Cartesian axes relative to the device.

Module	Accelerometer
Inputs	• Linear Force • Power, DC
Outputs	• 3-D Linear Acceleration
Functionality	Measures acceleration of the device in three linear directions in Cartesian space relative to the device.

Module	Magnetometer
Inputs	• Magnetic Field Intensity
Outputs	• 3-D Magnetic Field Intensity
Functionality	Measures the magnetic field intensity of Earth's magnetic field in three dimensions of Cartesian space relative to the device.

Module	GPS
Inputs	• RF signals • Power, DC
Outputs	• GPS data
Functionality	The GPS receives signals from satellites to calculate coordinates and other data such as translational speed.

Module	Micro SD
Inputs	• Power, DC • GPS data • IMU data
Outputs	• GPS data • IMU data
Functionality	Micro SD was intended to be used to store GPS information and IMU data in flight. The information could be retrieved when it needs to be sent to the smartphone.

Module	Buzzer
Inputs	• Power, DC
Outputs	• Sound, ~95 dB
Functionality	The buzzer is powered on after the disc hits lands to produce a loud audio signal for location.

Module	Bluetooth LE Transceiver
Inputs	• Power, DC
Outputs	• RF Bluetooth LE signal
Functionality	A Bluetooth LE (Low Energy) transceiver allows communication between the Tracker and a smartphone. The data from the microcontroller is sent to the smartphone via a bluteooth connection.

Module	Microcontroller
Inputs	• Power, DC • GPS data • IMU data
Outputs	• Power, DC • GPS data • IMU data • Alarm Signal
Functionality	The microcontroller powers and communicates with each peripheral when appropriate. It logs the GPS and IMU data during flight, powers the buzzer after it lands, and then sends the data to a smartphone through a Bluetooth LE connection.

Tracking Device Schematic (CW & SG)

The schematic for the tracking device attached to the golf disc is shown in Figure 9. The schematic shows the pin connections between the Flora microcontroller and each module. The connections for the battery to the microcontroller and to the off-board charger are shown in Figure 10 and Figure 11, respectively. The rechargeable battery connects directly to the microcontroller with a JST connecter. Each tracker module connects via appropriate serial communication pins. Some of the modules support different serial communication protocols and some can only connect with a particular protocol because of how the module was constructed. The module operations and connections are explained below.

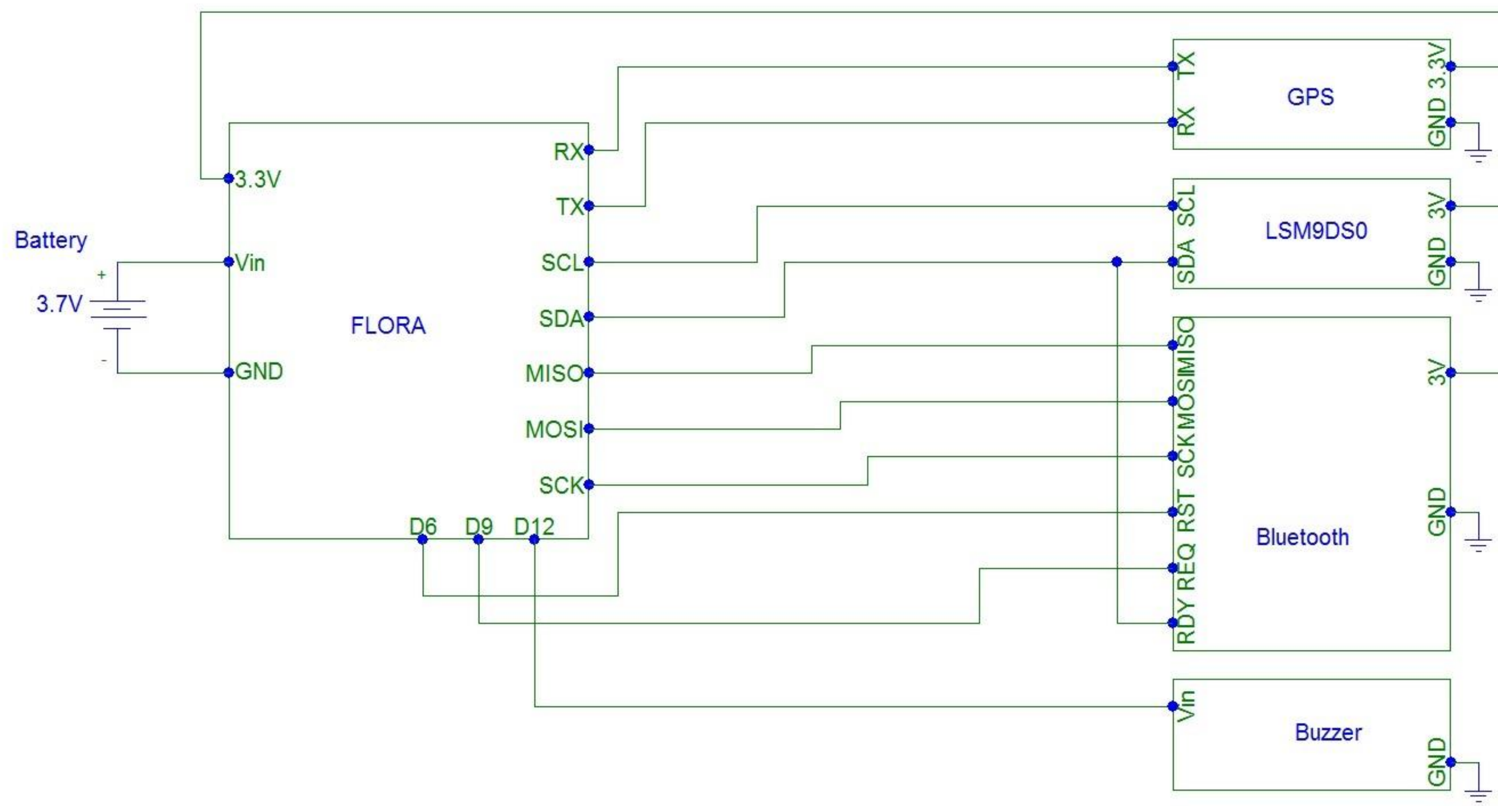

Battery
3.7V
FLORA
3.3V
Vin
GND
RX
TX
SCL
SDA
MISO
MOSI
SCK
D6
D9
D12
GPS
TX
RX
GND 3.3V
LSM9DS0
SDA SCL
GND 3V
Bluetooth
RDY REQ RST SCK MOSI MISO
GND
3V
Buzzer
Vin
GND

Figure 9: Schematic for Disc Tracker

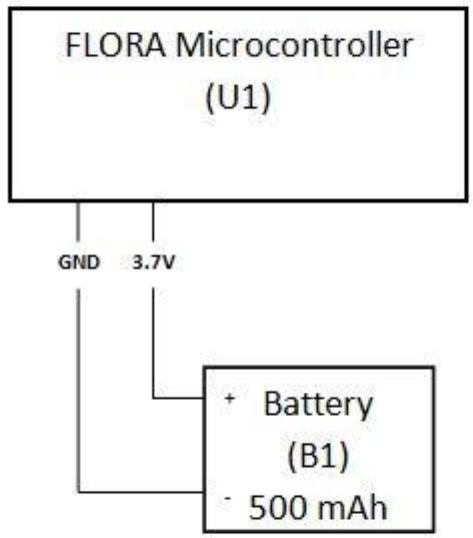


Figure 10: Schematic for battery connection

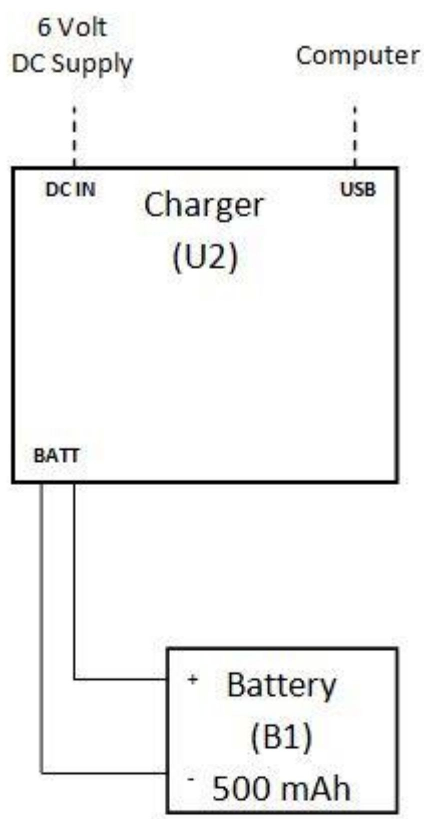


Figure 11: Schematic for off-board battery charger

Battery (SG)

A battery had to be chosen that could be small enough to attach to a golf disc while having at least enough capacity to power the tracker for an average game length and not add an enormous amount of weight. The battery used, therefore, is of lithium ion polymer (LiPoly) construction with a capacity of 500 mAh and a weight of 10.5 grams. The package size was small enough to be affixed to the underside of the golf disc and it was more than capable of powering the tracking device for an average game length. The battery supplies power to the microcontroller which regulates incoming voltage and distributes power to the peripheral components with limited current. When depleted, the battery shuts off at 3 Volts and must be disconnected from the system and connected to the off-board charger.

Off-board Battery Charger (SG)

To charge the LiPoly battery properly and safely, a compatible charger was chosen. Initially, a small solar panel was to be mounted to the golf disc to provide supplemental power during game play. Thus, the charger is capable of accepting power from a solar panel in addition to an external source and is small and light enough to fit on the golf disc. It became apparent that for the additional weight to the disc, the marginal amount of power supplied by a 1 Watt solar panel under best conditions was not enough to warrant mounting it on the disc.

Since the charger is not used in such a fashion as previously mentioned, it simply uses incoming power from a DC supply or USB connection to charge the battery at constant current and constant voltage (CC/CV). To maintain health of the battery and avoid overheating, a resistor was soldered in to set the charging current to a safe limit of 150 mA. Bypass capacitors were also soldered in to stabilize the charging control loops in the absence of a connection. The charger was assembled onto a small proto-board to increase its size for ease of handling as shown in Figure 12. A yellow and a green LED with current-limiting resistors were soldered to the board to increase visibility of "Charging" and "Charge Complete" indicators.

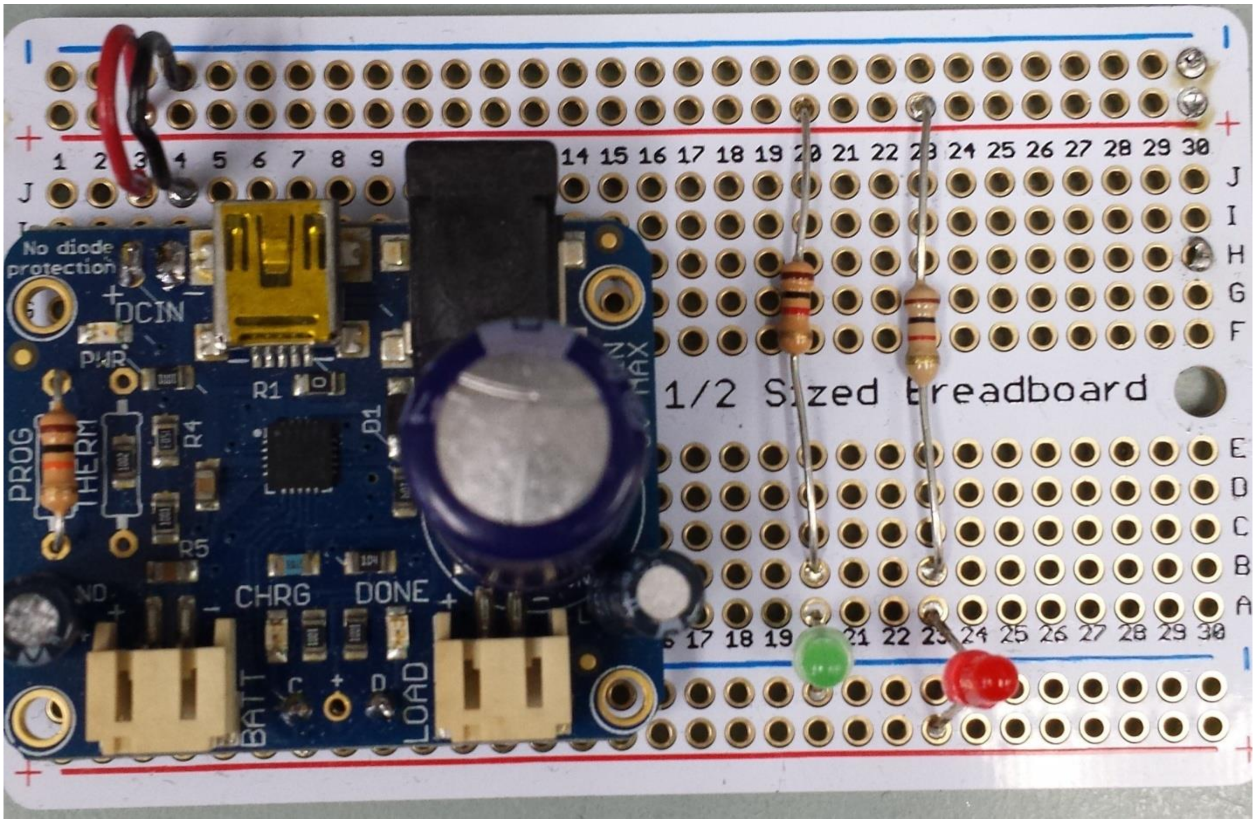


Figure 12: Off-board battery charger mounted to proto board

Piezoelectric Buzzer (SG)

To make an audible alert from the disc that is loud enough at a long distance, a low-power, small buzzer was needed. The Mallory Sonalert MSO206NR piezoelectric buzzer is a small device capable of producing a large amount of sound. It is a solid-state component that requires only a small DC voltage. It works on the principles of piezoelectricity in which voltages applied to materials with a crystalline structure cause deformations of the material and vice-versa. This allows a loud, high frequency (3.5 kHz) sound to be produced using very little electrical power. Within the rated 2 - 6 Volts DC it only draws up to 30 mA of current.

It was decided that the buzzer should be audible at a maximum distance of 100 meters. Based on typical sound pressure levels measured in decibels, dB_{SPL}, the sound from the buzzer at this distance needed to be a minimum of 40 dB. This is roughly the sound level of a quiet conversation at normal talking distance. Since sound intensity follows the inverse square law, the minimum sound pressure level the buzzer needed to produce was calculated working backwards from 40 dB at 100 meters. Considering a roughly 1 meter distance from a buzzer at ground level to the listener's ear, that equates to a 100 times increase in distance. Applying the inverse square law to these numbers yields the amount of change of intensity level,

$$I_1 = \frac{40}{[\frac{1}{100}]^2} = 400{,}000. \tag{8}$$

Converting this to dB_{SPL} yields

$$\frac{400{,}000}{10^4} = 40\ dB_{SPL}. \tag{9}$$

Therefore, the decrease in sound level across 100 meters is 40 dB_{SPL} so the minimum required level from the buzzer was set at 40 + 40 or 80 dB_{SPL}, at 1 meter.

Table 5: Piezo Buzzer Data

Manufacturer:	Mallory	CUI	Kingstate
Operating Voltage:	2 to 6 VDC	3 to 5 VDC	3 to 20 VDC
Max Current Draw:	30 mA	35 mA	10 mA
Loudness:	90 to 99 dB @ 1ft.	95 dB @ 10cm	95 @ 30cm
Normalized Loudness (1m):	79.68 to 88.68 dB	75 dB	85 dB
Weight:	3.5g	1.4g	7.0g

To ensure the minimum sound level of 40 dB_{SPL} at 100 meters, the buzzer needed to be able to produce a level of, at minimum, 80 dB_{SPL} normalized at 1 meter. As power and weight were also concerns, the device needed to be as sensitive as possible. From Table 5, Mallory MSO206NLR in the first column was the best choice among piezoelectric buzzers. During tests, a disc was thrown roughly 30 meters and the buzzer was audible well within that range.

Power Calculations (SG)

Table 6: Component Power Ratings

Component	Voltage (V)	Current (mA)	Power (mW)
Microcontroller	3.6	150	540
GPS	3.3	25	82.5
Piezo Buzzer	3.3	30	99
IMU	3.3	6.45	21.285
microSD Reader	3.3	150	495
Bluetooth LE	3.3	12.5	41.25

The maximum power consumption of the components necessary for this design under continuous operating conditions are given in Table 6. Using these values, the worst-case power consumption of the whole system was calculated. The maximum total power is:

$$540 + 82.5 + 99 + 21.285 + 495 + 41.25 = 1{,}279.04\, mW, \qquad (10)$$

or

$$1.28\, Watts.$$

Since the microSD card breakout module was not able to be successfully integrated in the tracker's operation, its power connection was cut. The worst-case power consumption was then

$$784.04\, miliwatts.$$

Table 7: Battery Ratings

Battery	Voltage (V)	Capacity (mAh)	Weight (g)	Cycle Life (hrs)
Adafruit 258	3.7	1200	25	3.3
SparkFun PRT-00339	3.7	1000	22	2.7
Adafruit 1578	3.7	500	10.5	1.4
Adafruit 1317	3.7	150	4.65	0.4

Table 7 gives data for several of the considered choices of onboard energy storage. All batteries listed are of Lithium Ion (Li-ion) or Lithium Ion Polymer (LiPo) construction. The cycle life is the calculated amount of time that the battery is capable of supplying the system on a full charge. This is based on the battery's capacity and the original 1.28 W system power draw. To convert the battery's capacity rating to a power rating based on a system operating voltage of 3.5 Volts, the calculation is,

$$Battery\ Power\ (Watt - Hours) = \ Capacity\ (mAh) \times\ 1000\ \times\ 3.5\ (V). \qquad (11)$$

Cycle life is,

$$Battery\ Life\ (Hours) = \frac{Battery\ Energy(WH)}{1.28\ W}. \quad (12)$$

Given that the actual power draw of the components during normal use is much less than the worst case scenario, the battery life is actually much greater than the values given especially considering the disconnected microSD breakout module.. Since weight was more of a limiting factor, the 500 mAh Adafruit 1578 was the most appropriate choice of battery as it delivers more than enough lifetime (>1.4 hrs) and adds only 10.5 grams to the disc. Even after several hours of use, the battery was able to maintain sufficient charge.

Global Positioning System (GPS) (CW)

The GPS 3.3 V and GND pins are connected to the 3.3 V and GND output pins on the Flora for power. The GPS is connected with two wires for UART serial communication. The TX and RX pins on the Flora are connected to the RX and TX pins on the GPS, respectively. The GPS will be used to track the location of the disc during its flight. It will be set to calculate a fix at an update rate of 5 Hz. This is the maximum fix rate civilian GPS units can calculate their position at. The baud rate for the UART connection will be set to the default 9600 baud rate for the GPS. The project only needs latitude, longitude, and time to operate. Therefore, as defined by National Marine Electronics Association (NMEA) standard 0183, the GPS will output NMEA RMC sentences which provides the required position relative to the WGS84 ellipsoid. The RMC sentence means the recommended minimum navigation information. The format of an NMEA RMC sentence is shown in Figure 13.

```
                                                            12
       1         2 3         4 5         6 7   8   9    10  11|
       |         | |         | |         | |   |   |    |   | |
$--RMC,hhmmss.ss,A,1111.11,a,yyyyy.yy,a,x.x,x.x,xxxx,x.x,a*hh
```

Figure 13: NMEA RMC sentence structure

The definitions of each field are explained below in Table 8**Error! Reference source not found.**.

Table 8: NMEA RMC sentence description

Field	Description
1	UTC Time
2	Status, A = Active, V = Void

3	Latitude
4	North or South
5	Longitude
6	East or West
7	Ground Speed (knots)
8	Track Good (degrees)
9	Date (ddmmyy)
10	Magnetic Declination (degrees)
11	East or West
12	Checksum

The RMC sentences are provided in a CSV format where each different sentence is on a new line. Since each location fix is calculated every 200 ms, an RMC sentence will be set to output every 200 ms. The Flora will read each sentence from the GPS and then send it to the micro SD card for storage until it is required for transmission over Bluetooth.

A test was conducted to demonstrate relative accuracy of a commercial GPS. An Etrex Venture HC handheld GPS was carried along a walk to emulate a disc throw. The path began in the southwestern end of parking area for an apartment complex. The initial GPS coordinates were recorded from the Etrex GPS as N 41˚ 04.55' (latitude) and W 81˚ 29.832' (longitude). The beginning and end points recorded are the first and last points in Table 9.

Table 9: Coordinates from an Etrex GPS

Latitude			Longitude		
degrees	**minutes**	**decimal degrees**	**degrees**	**minutes**	**decimal degrees**
41	4.543	41.075717	-81	29.833	-81.497217
41	4.543	41.075717	-81	29.832	-81.497200
41	4.543	41.075717	-81	29.831	-81.497183
41	4.543	41.075717	-81	29.83	-81.497167
41	4.543	41.075717	-81	29.829	-81.497150
41	4.543	41.075717	-81	29.828	-81.497133
41	4.543	41.075717	-81	29.827	-81.497117
41	4.543	41.075717	-81	29.826	-81.497100
41	4.543	41.075717	-81	29.825	-81.497083
41	4.543	41.075717	-81	29.824	-81.497067
41	4.543	41.075717	-81	29.823	-81.497050
41	4.543	41.075717	-81	29.822	-81.497033
41	4.543	41.075717	-81	29.821	-81.497017
41	4.543	41.075717	-81	29.82	-81.497000
41	4.543	41.075717	-81	29.819	-81.496983

41	4.543	41.075717	-81	29.818	-81.496967
41	4.543	41.075717	-81	29.817	-81.496950
41	4.543	41.075717	-81	29.816	-81.496933
41	4.543	41.075717	-81	29.815	-81.496917
41	4.543	41.075717	-81	29.814	-81.496900
41	4.543	41.075717	-81	29.813	-81.496883
41	4.543	41.075717	-81	29.812	-81.496867
41	4.544	41.075733	-81	29.811	-81.496850
41	4.545	41.075750	-81	29.81	-81.496833
41	4.546	41.075767	-81	29.809	-81.496817
41	4.547	41.075783	-81	29.808	-81.496800
41	4.548	41.075800	-81	29.807	-81.496783

The data in Table 9 shows the GPS coordinates retrieved from walking an Etrex GPS through a parking lot. The length of the walk is similar to a moderate golf disc throw. The points illustrate the relatively good accuracy of civilian GPS. During this experiment, the horizontal dilution of precision (HDOP) was recorded at ±13 ft. The HDOP value occurs from the large distance between the unit and the GPS satellites. Perhaps more intuitively, HDOP is similar to the error that occurs from the small angle approximation, or comparing arc length to straight distance between two points separated by an angle. However, the received coordinates for this test were confirmed accurate after the coordinates were converted into decimal degrees and plotted on a map of the area in Figure 14.

Figure 14: GPS coordinates plotted on a map

The path can be clearly seen from the image. The image in Figure 14 is from a website tool which plots multiple points onto Google Maps (www.darrinward.com). The tool requires decimal degrees to plot the coordinates which is the reason for the column in Table 9.

Inertial Measurement Unit (IMU) (CW)

The microcontroller will control the IMU and log the data it outputs. The IMU will consist of a 9-DOF (degrees-of-freedom) chip composed of a 3-axis accelerometer, gyroscope, and magnetometer. The accelerometer will provide values of acceleration in m/s^2 based on a Cartesian coordinate system centered on the chip. The gyroscope will provide values of *deg/s* around the axes defined in the Cartesian coordinate system for the accelerometer. Finally, the magnetometer will provide measurements of the magnetic field intensity in gauss along the three Cartesian axes of the accelerometer.

The magnetometer can be used to determine orientation on the surface of the earth. The magnetometer will provide the output of the magnetic field intensity in a horizontal and vertical direction on the surface of the earth. Therefore, the heading can be calculated from the angle between the two given vectors,

$$heading = \tan^{-1}\frac{H_y}{H_x} \tag{13}$$

where magnetic field intensities in the vertical and horizontal positions are given by H_y and H_x, respectively. Magnetic declination can be accounted for after the global position is known.

The accelerometer can be used to determine distance traveled with the acceleration measurements and the elapsed time. The accelerometer measures instantaneous acceleration at given intervals. The time between intervals can be used to calculate distance traveled. Velocity can be obtained from integrating acceleration,

$$v(t) = \int_0^t a dt = at, \tag{14}$$

where *a* is the value of acceleration. Further, position can be calculated as,

$$x(t) = \int_0^t v dt = vt. \tag{15}$$

where *v* is the velocity. Considering initial position and combining Equation **Error! Reference source not found.**(**14**) and Equation (**15**) yield a formula to calculate distance traveled,

$$x(t) = x_o + vt + \frac{1}{2}at^2. \tag{16}$$

The basic kinematic equations can be used to calculate position by integrating the acceleration measurement twice. This calculation can be implemented recursively to calculate total distance traveled by adding the new distance to the previous distance.

Since the gyroscope measures angular velocity, which is the derivative of the angular position, the angle of change for each axis can be calculated. Therefore, the angle is

$$\theta = \int_{t_1}^{t_2} \omega dt = \omega(t_2 - t_1), \tag{17}$$

where ω is the angular velocity output from the gyroscope. Since the angle can be calculated on each axis, yaw, pitch, and roll can be defined for the device attached to the gyroscope.

Hardware Mounting (SG)

A critical part of the design was the mounting of all hardware to the golf disc. Ideally, the hardware should be mounted such that the disc can sustain significant impact at any point which is possible during game play. However, because of the constraints imposed by using separate, interconnected modules, the system was designed to sustain only impact from the top and edge of the disc. Fragile electronics were left exposed on the underside of the disc but since they did not extend beyond the lip of the disc, it was possible for the disc to be dropped at all angles on flat surfaces.

Components were arranged on the disc according to weight distribution and sensor orientation as well as routing of connections. The ideal balance of weight that was symmetric around the center point of the disc was found and then a small compromise was made to facilitate electrical connections by shifting some components to different points on the disc. Though the disc ended up being slightly heavier on the side where the battery (the heaviest component) was mounted, the overall balance was such that it did not noticeably impede the flight characteristics of the disc.

Several methods were utilized to secure components to the disc and make electrical connections between components. Primarily, a clear RTV silicone sealant was used to bond components to the disc. It was chosen for its flexible, adhesive properties. To mount the GPS module, a square hole was cut in the center of the disc to allow the main chip to stick up on the top of the disc so that the antenna could receive an un-attenuated satellite signal during flight, as seen in Figure 15. The IMU chip was adhered to the back of the GPS so that it was aligned to the rotational and mass centers of the disc. Remaining components were sewn to the disc or attached with Velcro.

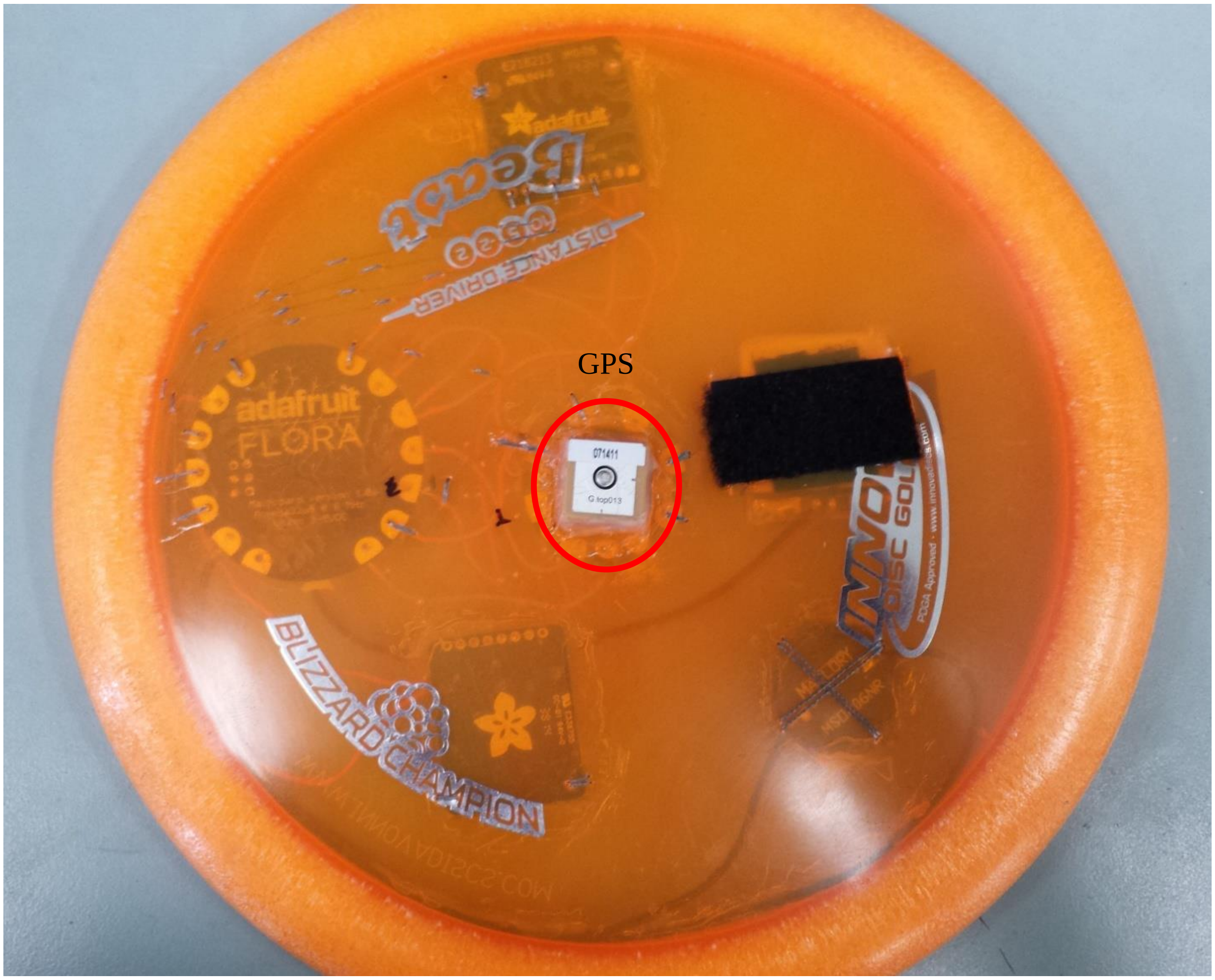


Figure 15: Top view of completed disc

Several connections, mainly those to the Bluetooth module, were made using a conductive thread. The thread was a 3-ply thread made entirely of 316L stainless steel and designed for wearable electronics. It was chosen for its size, tensile strength, and conductivity. At 10 Ohms per foot, the thread served two purposes: make electrical connections, and hold components to the golf disc. Using a standard sewing needle, the thread was sewn into the disc in such a manner to create "traces" in which a majority of the thread was exposed on the underside of the disc. Connections to the modules were made either by pulling the thread through the contact holes and tying a large knot that pulled tight to the contact or by wrapping the thread through the contact holes several times and securing with a knot. It was found that the best connections were those with the thread that had been wrapped several times around the chip contact. Since the thread had a slight tendency to fray, microscopic shorts appeared between a few of the traces. This was rectified by coating each thread trace with a thin lacquer (i.e. clear finger nail polish). The lacquer also safeguarded against human contact and moisture. All connections are mounted are visible in Figure 16.

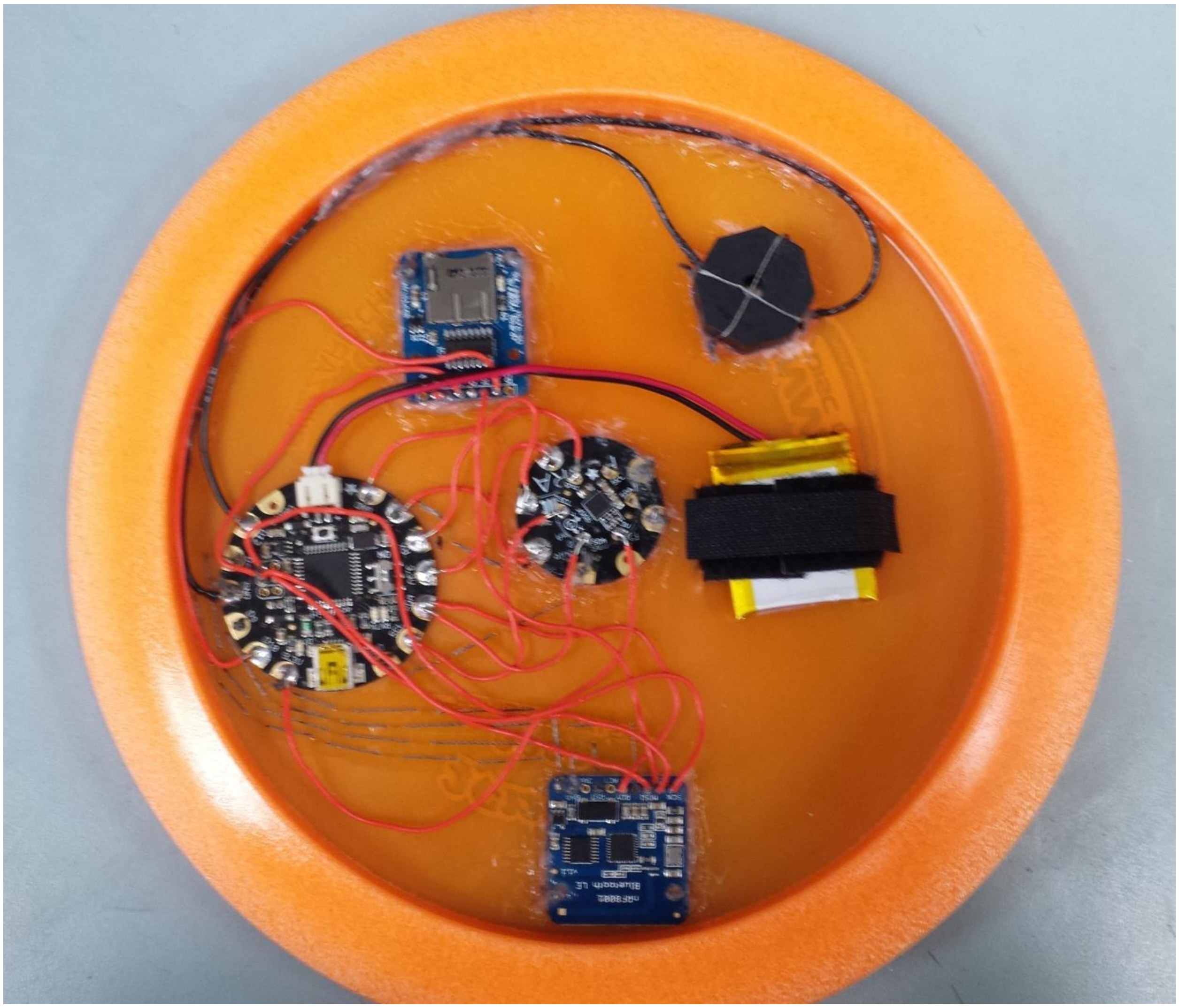


Figure 16: Underside of disc with mounted components

Weight Experiment (CW)

Weight is a serious concern for this project. The final design connect to the disc must not weigh too much or the disc will fall quickly to the ground when it is thrown. An (Saturday, September 13, 2014) experiment was conducted at the Arboretum Disc Golf course in Canton, OH to determine potential weights which may drastically hinder the disc's performance. This experiment was performed by duct taping five quarters into a thin weight shown in Figure 17.

Figure 17: Five quarters wrapped in duct tape serving as a weight

The five quarters were then taped to the underside of a golf disc. The tape was wrapped all the way around the disc in a cross pattern with the quarters at the underside cross section. The US Mint indicates that quarters weigh 5.670 grams. So five quarters had an approximate weight of 28.4 grams. The disc was thrown multiple times with and without the quarters attached.

Two different people threw the disc with and without the weight attached. When the first person threw the weighted disc, there was no discernable difference between flight path or distance thrown from that of the un-weighted disc. Similarly, when the second person threw the weighted it appeared to travel just as far as the un-weighted disc. The specific distances were not measured because there was not a tool available during the test to accurately measure throw distances. Since the theoretical weight of the current device design is estimated around 30 grams and nearly 30 grams did not interfere with the performance of a disc, this test helped lead to the idea this project would be successful.

MicroSD Card Breakout (BL)

The FLORA microcontroller will maintain storage logs of sensor information using the proposed MicroSD card breakout board+ (MicroSD reader) from Adafruit, in addition to a standard MicroSD card formatted using FAT32. The MicroSD reader will be directly connected to the microcontroller as portrayed in Figure 9 from Section 3 above. In this implementation, the CS, CLK, DI, and DO pins of the MicroSD reader will be connected to the SS, SCK, MOSI, and MISO pins of the microcontroller respectively. The DI and DO pins regulate the data inflow and outflow to the slave node (MicroSD reader) from the master node (microcontroller). The MicroSD card will contain disc information collected for each flight from the IMU and GPS sensors. This data will then be transmitted to the smart phone application using the system's Bluetooth feature.

Bluetooth (BL)

As discussed previously, the FLORA microcontroller will communicate flight sensor data stored on the MicroSD card to the smartphone application by means of Bluefruit LE - Bluetooth Low Energy (BLE 4.0) (Bluetooth) device; produced by Adafruit. The smartphone application will communicate with the microcontroller and determine what data to transmit back to the smartphone to synchronize sensor characteristics of sequential flight attempts. This data will then be stored on the smartphone's internal storage for use by the application. Additionally, the user may select data sets corresponding to individual flights and mark them for deletion, removing them from both the phone and disc's storage. The Bluetooth unit will be connected to the microcontroller as depicted in Figure 9 from Section 3 above.

Software - Level 0 Block Diagram (CW)

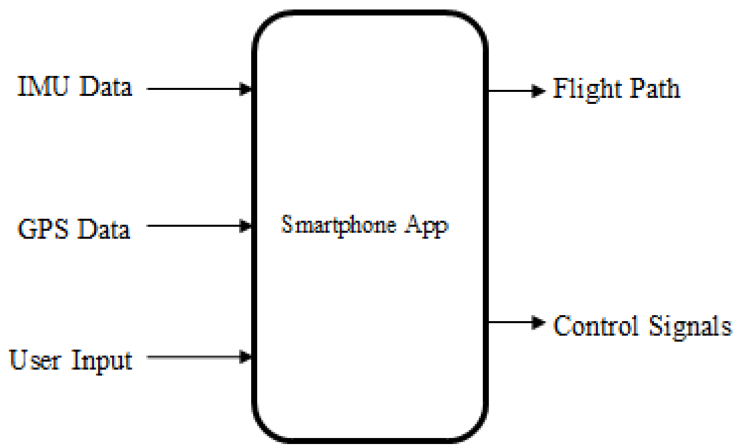


Figure 18: Level 0 Software Block Diagram

Software - Level 0 Functional Requirement Table (NS)

Table 10: Level 0 Software Functional Requirement Table

Module	Smartphone App
Inputs	• IMU Data • GPS Data • User Input
Outputs	• Flight Path • Control Signals

Functionality	The app will use the logged IMU and GPS data to map the flight path of the disc onto a map of the area. It will also plot the best case next throw and send commands to the microcontroller to change the operation of the device.

Software – Level 1 Block Diagram (NS)

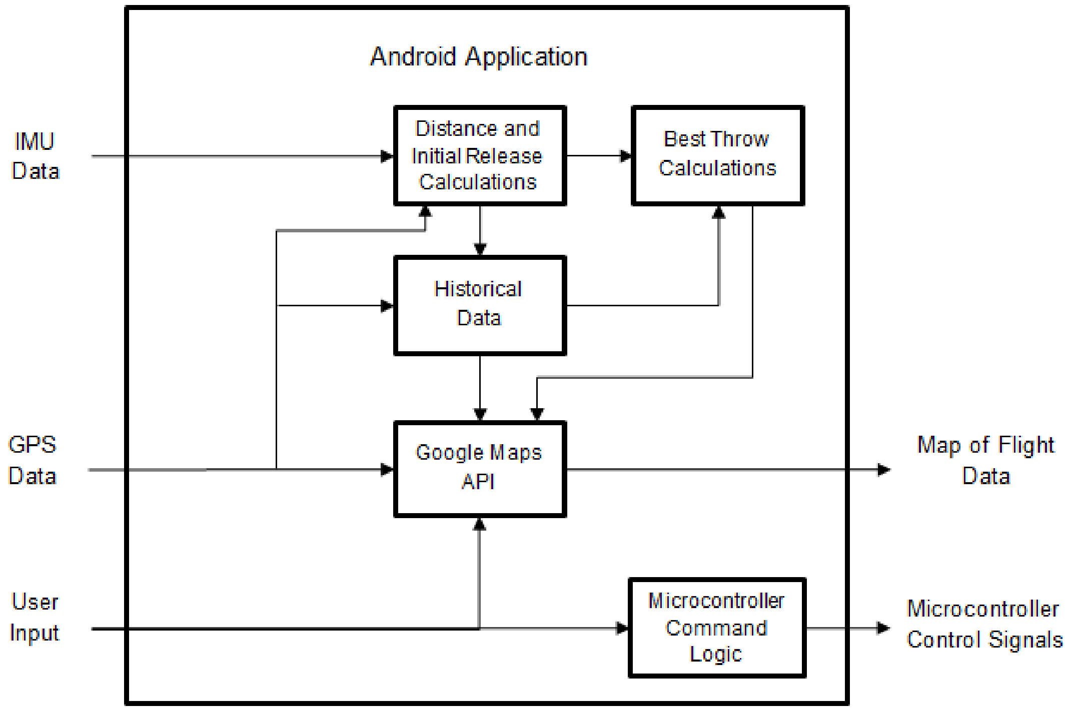


Figure 19: Level 1 Software Block Diagram

Software – Level 1 Functional Requirements Table (NS)

Table 11: Level 1 Software Functional Requirement Tables

Module	Distance and Initial Release Calculations
Input(s)	• IMU Data • GPS Data
Output(s)	• Distance • Direction

Function	This module calculates the distance traveled using the initial and final GPS coordinates and calculates initial release data.

Module	Historical Data
Input(s)	• GPS Data • Flight Distance • Direction.
Output(s)	• Distance traveled for previous throws • Direction for previous throws • GPS coordinates for previous throws
Function	This module stores the flight information for every previous throw.

Module	Best Throw Calculations
Input(s)	• Flight Distance • Direction • Historical Data.
Output(s)	• Best available throw
Function	This module calculates the distance traveled using the initial and final GPS coordinates and calculates the average speed of the throw using the distance and the initial and final timestamps.

Module	Microcontroller Command Logic
Input(s)	• User input
Output(s)	• Microcontroller Control Signals
Function	This module takes user input from the touch screen and sends commands to the microcontroller to control the operation of the disc.

Module	Google Maps API
Input(s)	• GPS Data • Historical Data • Best Throw Calculations • User Input
Output(s)	• Map of flight data
Function	This module uses the location data from all previous throws and puts the flight paths on a map of the golf course. The best throw calculations are used to show how far the disc can be thrown next. The module also takes user input to place intended targets on the map.

Software - Level 2 Block Diagram (NS)

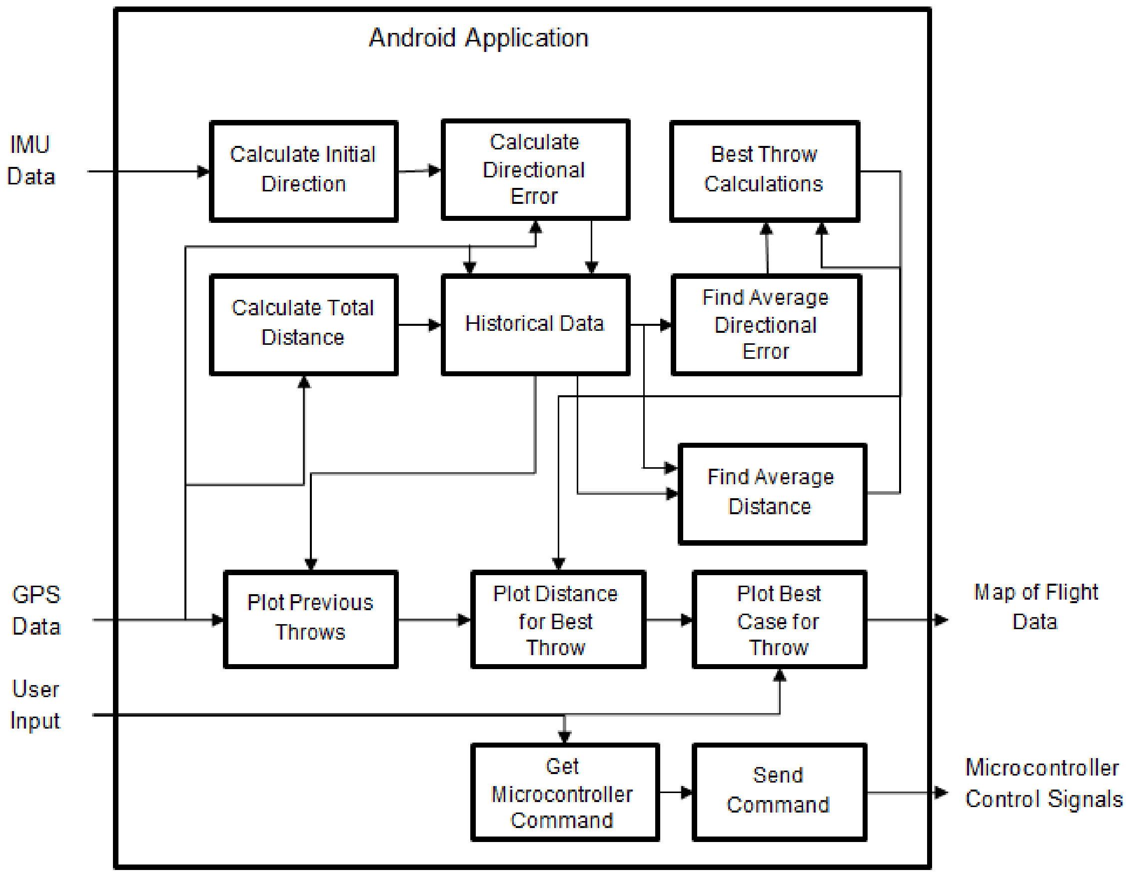


Figure 20: Level 2 Software Block Diagram

Software - Level 2 Functional Requirement Table (NS)

Table 12: Level 2 Software Functional Requirement Tables

Module	Plot Previous Throws
Inputs	• GPS data
Outputs	• Map of course highlighting all previous throws
Function	Plot the initial and final GPS coordinates of every prior throw onto a map of the golf course.

Module	Calculate Total Distance

Inputs	• GPS Data
Outputs	• Distance of throw
Function	Calculate the distance between the initial and final GPS coordinates.

Module	Calculate Directional Error
Inputs	• GPS Data • Initial direction
Outputs	• Directional error
Function	Calculate the difference between the direction of the initial release and the direction the disc actually took using the GPS data.

Module	Historical Data
Inputs	• GPS Data • Distance • Directional error
Outputs	• GPS Data • Distance • Directional error of previous throws
Function	Stores the flight data for every throw and outputs the data from all previous throws.

Module	Find Average Directional Error

Inputs	• Historical directional errors
Outputs	• Average directional error
Function	Calculate average directional error based on all previous directional errors

Module	Find Average Distance
Inputs	• Historical distances • Directional errors
Outputs	• Average distance
Function	Calculate the average distance based on all previous distances and their corresponding directional errors.

Module	Best Throw Calculations
Inputs	• Average distance • Average directional error
Outputs	• Distance of best possible throw
Function	Calculate the distance of the best possible throw using the average flight data.

Module	Plot Distance for Best Throw
Inputs	• Distance of best possible throw • Map of previous throws
Outputs	• Map of course with all previous throws and the distance of the best possible throw

Function	Plot a circle with a radius of the distance of the best possible throw on the map created in the Plot Previous Throws block.

Module	Plot Best Case for Throw
Inputs	• User input • Map of course with all previous throws and the distance of the best possible throw
Outputs	• Map of flight data
Function	User input is used to select the direction of the throw and a line is plotted that shows the best case scenario of the next throw (distance) and an indicator showing any directional compensation that should be considered.

Module	Get Microcontroller Command
Inputs	• User input
Outputs	• Microcontroller command
Function	User input is used to select a mode of operation for the disc and the corresponding command is looked up from storage.

Module	Send Command
Inputs	• Microcontroller command
Outputs	• Microcontroller control signals
Function	Format and send the command over Bluetooth to the disc.

Application Angle Calculation (NS)

In the Android application, the total angle of difference is calculated and saved for every throw. The angle from the starting GPS location to the user-plotted hole is referred to as the hole angle. The angle from the starting GPS location to the final GPS location is referred to as the actual angle. The total angle of difference describes the difference between the hole angle and the actual angle. The angle calculation is shown below.

$$Hole\ Angle = atan2(\sin(longitude2 - longitude1) * \cos(latitude2), \cos(latitude1) * \sin(latitude2) - \sin(latitude1) * \cos(latitude2) * \cos(longitude2 - longitude1)) \quad (18)$$

Where latitude1/longitude1 are the coordinates of the starting GPS location and latitude2/longitude2 are the coordinates of the hole.

$$Actual\ Angle = atan2(\sin(longitude2 - longitude1) * \cos(latitude2), \cos(latitude1) * \sin(latitude2) - \sin(latitude1) * \cos(latitude2) * \cos(longitude2 - longitude1)) \quad (19)$$

Where latitude1/longitude1 are the coordinates of the starting GPS location and latitude2/longitude2 are the coordinates of the final GPS location.

$$Angle\ of\ Difference = Actual\ Angle - Hole\ Angle \quad (20)$$

Application Totals Data (NS)

The Android application keeps track of user data by holding the throw count, average distance and average angle in a totals object that is accessible to the entire application. Every time this object is changed, its data is written to a table in a SQLite database. The database allows for nonvolatile storage of the data. When a new throw is transferred, the throw count gets incremented and the average angle and distance are updated to include the new data.

Application Data Transfer Operation (NS)

Data is transferred from the disc tracker device to the application in 20 byte increments. When data is received it is buffered by saving it into a single string. The end of transmission is signaled by the receipt of the string “$FF”. When the termination string is received, the buffered data string is split on the “$” character into a vector of strings. This vector is sent to a function that parses the latitude/longitude pair out of each string and saves them into a file. The total distance is found by using the built in distanceTo function found in the Google Maps API to return the distance between the first and last GPS points that were transferred. The angle of difference is calculated using the equations shown above. Once these parameters are calculated the totals data is updated using the method described above.

Microcontroller Control Flow (CW)

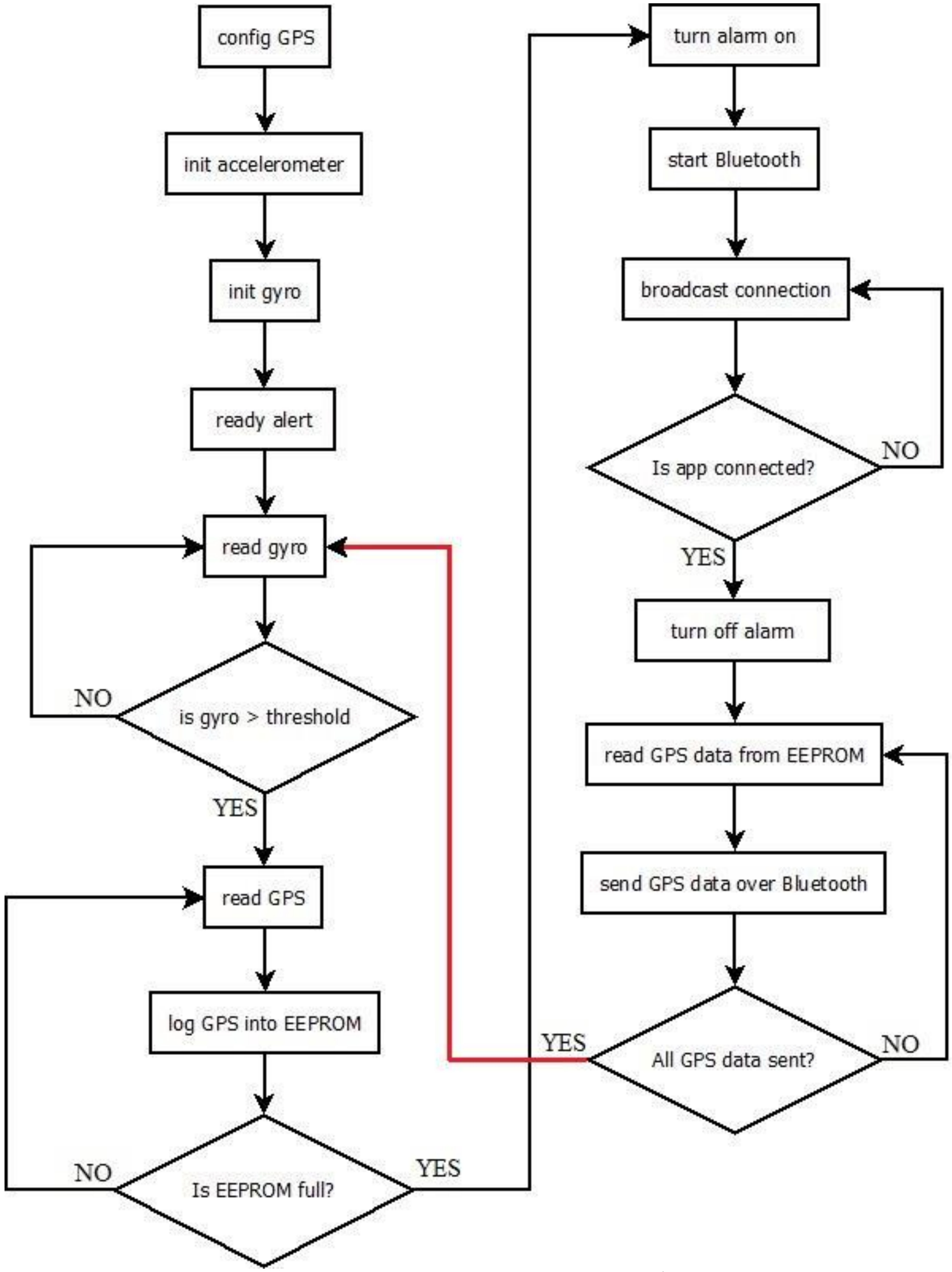


Figure 21: Microcontroller Control Flow[1]

[1] The red line is used to indicate no intersection between flow options.

Figure 21 shows the control flow of the device. The process starts when the microcontroller is turned on. Table 13 below describes each control block in detail.

Table 13: Table of Control Flow

Control Step	**Function**
Config GPS	Sets GPS to calculate a fix 5 Hz and send current fix 2 Hz to FLORA over UART.
Init Accelerometer	Turns on the accelerometer to provide acceleration measurements at 50 Hz with 2g sensitivity through I2C.
Init Gyroscope	Turns on gyro to provide radial velocity measurements at 95 Hz with 2000 dps sensitivity through I2C.
Ready alert	The buzzer will sound to indicate disc can be thrown.
Read gyro	Reads the radial velocity on the axis perpendicular to the disc.
Is Gyro > Threshold	If the radial velocity is greater than 1000 dps, the disc is spinning and process continues to the next step. Otherwise, it reads the radial velocity again.
Read GPS	Reads and parses the NMEA RMC string to attain latitude and longitude in decimal-minute degrees.
Log GPS into EEPROM	Logs the parsed GPS string into the EEPROM.
Is EEPROM full?	If the EEPROM is not full, read the next GPS string. Otherwise, move to the next step.
Turn alarm on	Turn the buzzer on.
Starts Bluetooth	Starts the Bluetooth device.
Broadcast Connection	Starts advertising available Bluetooth connection.
Is app connected?	If the app has connected to Bluetooth, move to the next state. Otherwise, continue broadcasting Bluetooth connection.
Turn off alarm	Turns buzzer off after app connects.
Read GPS data from EEPROM	Reads the parsed GPS string from the EEPROM.
Send GPS data over Bluetooth	Bluetooth sends the GPS string 20 characters per packet.
All GPS data sent?	If all the GPS data has been sent, start control process again. Otherwise, read the next GPS string from EEPROM to send.

4. Operation, Maintenance, and Repair Instructions

Operation Instructions

Disc (CW):

1) Attach battery to Velcro slot on the disc.
2) Plug the male JST end of the battery cable into the JST female port on the disc.
3) On the disc, turn the power switch to the **ON** position.
4) Ensure the top of the disc has *line-of-sight* with the sky.
5) When the disc alarm is heard, it should be thrown.
6) The disc alarm will continue to sound once it lands until the disc connects to the app.

Battery Charger (CW):

1) Plug the USB Type-A male end of the USB to mini-USB cable into a USB Type-A female port on a computer.
2) Plug the mini-USB Type-B male end of the USB to mini-USB cable into the mini-USB Type-B female connector on the charger.
    a. If the red LED is on, the battery is charging.
    b. If the green LED is on, the battery is fully charged and ready to use.

Android Application Installation (NS):

1) Enter into the security settings on the Android device and enable “Installation from unknown sources”.
2) Download the app.apk file and open it.
3) Select “Ok” to accept the required permissions for the application. This will install the application to the device and it will appear in the application drawer as “Where’s My Disc”.

Android Application Operation (NS):

The application has three tabs to separate the different operations. The tabs can be navigated through by selecting each from the action bar at the top or by swiping in the direction of the desired tab. There is also a menu that is accessible by selecting the three dots at the top right of the action bar. The menu contains options for starting a new game, adding the demo throw, viewing the Bluetooth log, viewing the legend for the map, clearing all of the saved data and viewing information about the design team.

The Connect tab is a simple tab that facilitates connecting to the Bluetooth adapter on the disc tracker device. Upon entering the application, if Bluetooth isn't enabled on the Android device, a prompt will appear to ask permission to enable it. Once it is enabled, to search for the tracker device, select "Search for Devices". This will perform a Bluetooth LE scan for compatible adapters. Any devices found will appear in the "Devices Found" list. The disc tracker device will appear as "WMD 4.0". Select the device from the list and the connection status will change from "Device Disconnected" to "Device Connected". To refresh the status of the connection, select "Refresh Connection Status". This will return the status to "Disconnected" if the disc tracker device is out of range, or will remain unchanged if it is still connected.

The Data tab consists of a list of throws. Each entry in the list represents a single throw and displays the throw ID, angle of difference and total distance of the throw. The throws are selectable and selecting a throw brings up a more detailed view of the statistics of that throw. In addition to the three fields mentioned earlier, the game id and the sync time of the throw are shown. To return to the data tab from the individual throw details view, select the back arrow at the top of the screen or use the android system back button. If new data is transferred, the list of throws can be refreshed by pulling down on the list, until a white circle fully appears at the top, and then releasing.

The final tab is the Map tab, which uses the Google Maps API to plot GPS coordinates onto a map of the disc golf course. The map will automatically default to the location of the user's android device. When data is transferred from the disc tracker device it is automatically parsed and plotted on the map. The GPS module on the tracker device can sometimes collect a few bad GPS points. To remedy this issue, if a set of coordinates is transferred the application checks if they are within 50 meters of the Android device or 5 meters of the previous throw. If they are not, those points are not plotted or processed. The map tab has a planning feature that allows a user to plot out the path they will take to avoid obstacles and reach the hole in the most efficient way. First the user will plot where the hole on the course is by pressing the "Plot Hole" button. After pushing the button, a flag icon can be placed on the map where the actual hole is located. Next, the player icon will have a circle surrounding it that represents the maximum distance that the user can throw the disc based on previous throws. The user will select a spot within this hole for the first throw to land. Once the first spot has been selected, the circle will move to surround this new point. This process repeats for all subsequent selections, until the user has plotted a full path leading to the hole. There are many different icons used on the map tab, and the all are defined on the map legend shown below.

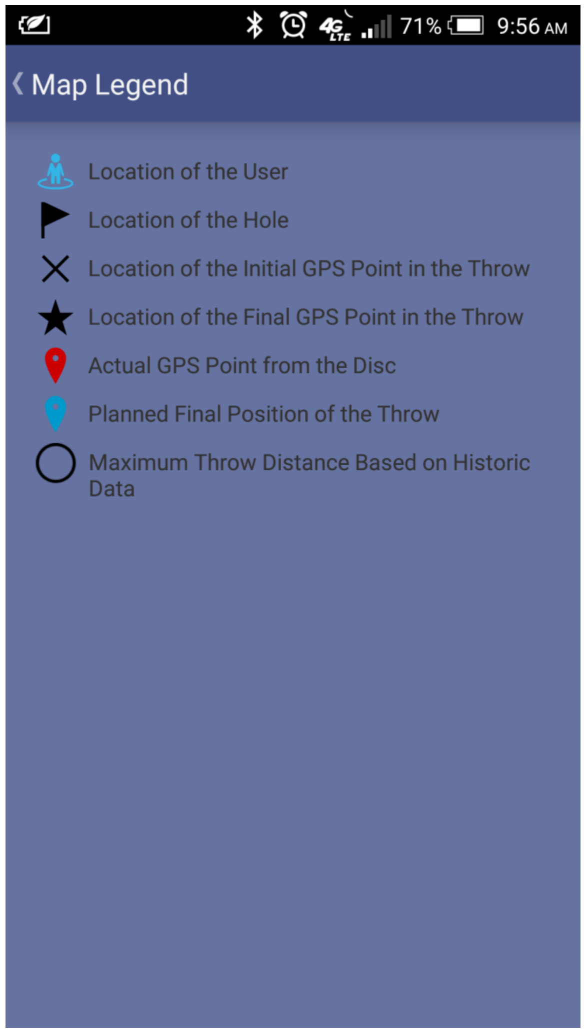
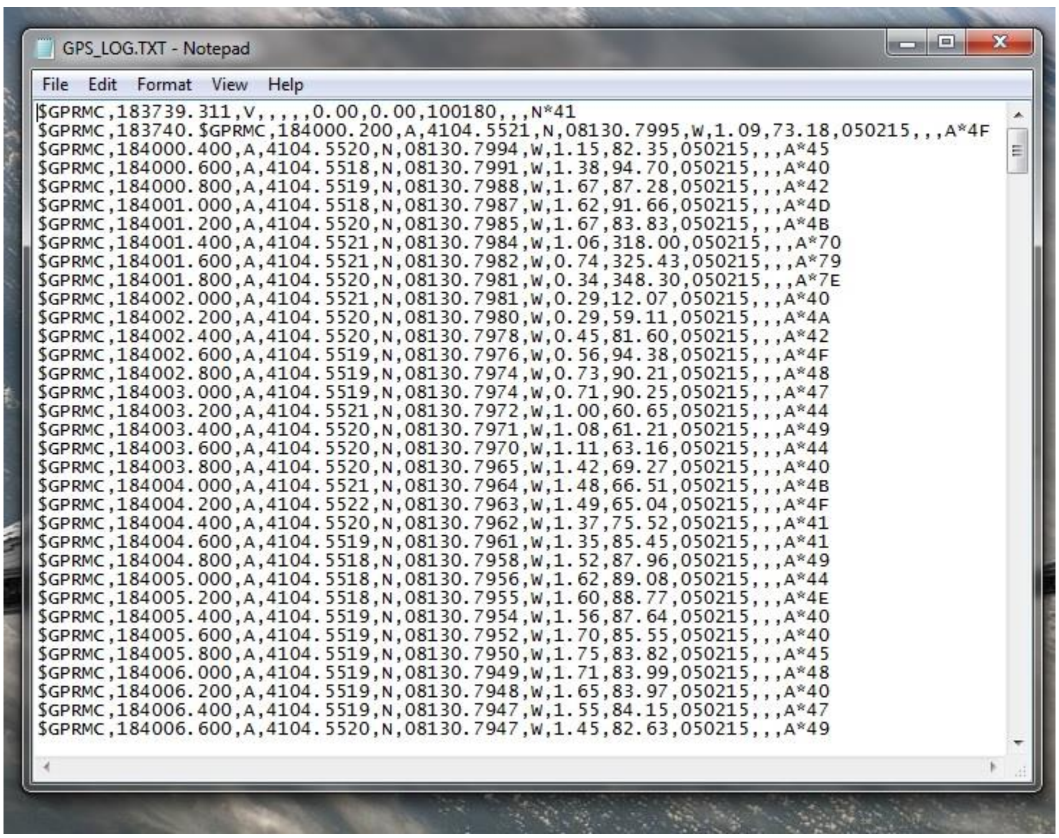


Figure 20: Android Application Map Legend

5. Testing Procedures

GPS and SD card (CW)

Arduino code was developed to use GPS and the SD card reader. Although, the SD card reader was not implemented in the final design because of power issues, it was tested in the development phase of the project since the intention was to use it. The Arduino sketch *midterm_GPS_demo* was written to configure the GPS and write the GPS data into a file on the SD card. The two devices were successfully implemented and tested together. The Arduino sketch is located in the *Appendix*. A screenshot of the output file saved on the SD card is shown in Figure 22*: Raw GPS data logged on SD card*. The specific GPS configuration is explained in Table 13.




```
$GPRMC,183739.311,V,,,,,0.00,0.00,100180,,,N*41
$GPRMC,183740.$GPRMC,184000.200,A,4104.5521,N,08130.7995,W,1.09,73.18,050215,,,A*4F
$GPRMC,184000.400,A,4104.5520,N,08130.7994,W,1.15,82.35,050215,,,A*45
$GPRMC,184000.600,A,4104.5518,N,08130.7991,W,1.38,94.70,050215,,,A*40
$GPRMC,184000.800,A,4104.5519,N,08130.7988,W,1.67,87.28,050215,,,A*42
$GPRMC,184001.000,A,4104.5518,N,08130.7987,W,1.62,91.66,050215,,,A*4D
$GPRMC,184001.200,A,4104.5520,N,08130.7985,W,1.67,83.83,050215,,,A*4B
$GPRMC,184001.400,A,4104.5521,N,08130.7984,W,1.06,318.00,050215,,,A*70
$GPRMC,184001.600,A,4104.5521,N,08130.7982,W,0.74,325.43,050215,,,A*79
$GPRMC,184001.800,A,4104.5520,N,08130.7981,W,0.34,348.30,050215,,,A*7E
$GPRMC,184002.000,A,4104.5521,N,08130.7981,W,0.29,12.07,050215,,,A*40
$GPRMC,184002.200,A,4104.5520,N,08130.7980,W,0.29,59.11,050215,,,A*4A
$GPRMC,184002.400,A,4104.5520,N,08130.7978,W,0.45,81.60,050215,,,A*42
$GPRMC,184002.600,A,4104.5519,N,08130.7976,W,0.56,94.38,050215,,,A*4F
$GPRMC,184002.800,A,4104.5519,N,08130.7974,W,0.73,90.21,050215,,,A*48
$GPRMC,184003.000,A,4104.5519,N,08130.7974,W,0.71,90.25,050215,,,A*47
$GPRMC,184003.200,A,4104.5521,N,08130.7972,W,1.00,60.65,050215,,,A*44
$GPRMC,184003.400,A,4104.5520,N,08130.7971,W,1.08,61.21,050215,,,A*49
$GPRMC,184003.600,A,4104.5520,N,08130.7970,W,1.11,63.16,050215,,,A*44
$GPRMC,184003.800,A,4104.5520,N,08130.7965,W,1.42,69.27,050215,,,A*40
$GPRMC,184004.000,A,4104.5521,N,08130.7964,W,1.48,66.51,050215,,,A*4B
$GPRMC,184004.200,A,4104.5522,N,08130.7963,W,1.49,65.04,050215,,,A*4F
$GPRMC,184004.400,A,4104.5520,N,08130.7962,W,1.37,75.52,050215,,,A*41
$GPRMC,184004.600,A,4104.5519,N,08130.7961,W,1.35,85.45,050215,,,A*41
$GPRMC,184004.800,A,4104.5518,N,08130.7958,W,1.52,87.96,050215,,,A*49
$GPRMC,184005.000,A,4104.5518,N,08130.7956,W,1.62,89.08,050215,,,A*44
$GPRMC,184005.200,A,4104.5518,N,08130.7955,W,1.60,88.77,050215,,,A*4E
$GPRMC,184005.400,A,4104.5519,N,08130.7954,W,1.56,87.64,050215,,,A*40
$GPRMC,184005.600,A,4104.5519,N,08130.7952,W,1.70,85.55,050215,,,A*40
$GPRMC,184005.800,A,4104.5519,N,08130.7950,W,1.75,83.82,050215,,,A*45
$GPRMC,184006.000,A,4104.5519,N,08130.7949,W,1.71,83.99,050215,,,A*48
$GPRMC,184006.200,A,4104.5519,N,08130.7948,W,1.65,83.97,050215,,,A*40
$GPRMC,184006.400,A,4104.5519,N,08130.7947,W,1.55,84.15,050215,,,A*47
$GPRMC,184006.600,A,4104.5520,N,08130.7947,W,1.45,82.63,050215,,,A*49
```

Figure 22: Raw GPS data logged on SD card

IMU (CW)

Similarly, Arduino code was developed to use the IMU. While the accelerometer and magnetometer were, also, not implemented the final process, they were tested along with the gyroscope. The Arduino sketch *midterm_IMU_demo* was written to configure and test the IMU.

The sketch displays the 3-D vector for each IMU device on the serial monitor. The gyroscope configuration is explained in Table 13. The Arduino sketch is located in the *Appendix*.

Bluetooth (BL)

The nRF8001 Bluetooth breakout was tested using an Arduino Uno board as recommended by Adafuit's "Getting Started with the nRF8001 Bluefruit LE Breakout" instructions located on their website. For android users, a nRF UART v2.0 application is available on the android marketplace for connecting to this device. The Bluetooth breakout's UUID is not supported by standard Bluetooth applications, so it must be included in the application which will connect to it.

Adafruit has provided an "Adafruit_BLE_UART" library with sample code "echoDemo," which has been included at the conclusion of this report. The echoDemo provides the capability of sending and receiving hex characters over the Bluetooth connection, translating the data to readable text upon arrival. Once tested, the nRF UART application was used to receive GPS data from the disc, which is portrayed in *Figure **23*** below. The final implementation of the project immediately sends the GPS data, saved in EEPROM, to developed Android Application, which will be discussed later.

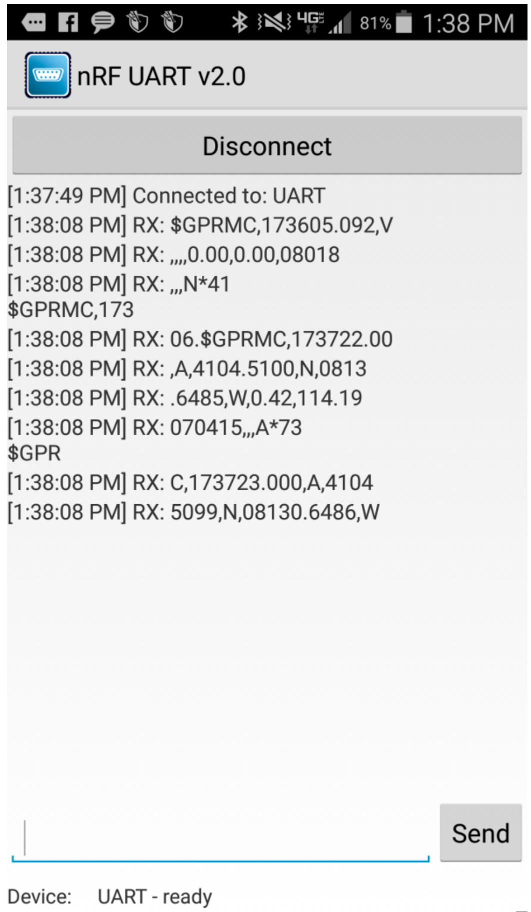


Figure 23: GPS test data sent over Bluetooth

Android Application (NS)

The application was tested thoroughly using GPS data that was collected from the disc tracking device. Once the GPS data string format was decided on, we gathered data from test throws on the device. That data was saved into a text document so it could be tested in the application. The data was split into separate, twenty character long, strings as this length was a limitation of the Bluetooth module. The shortened strings were run through the data parsing functions and the resulting latitude/longitude pairs were examined for accuracy.

In addition to the data transfer and parsing tests, functional testing was performed extensively. The operation of each tab was explored and tested for cosmetic and functional issues. The map tab was the most complex tab, and therefore endured the most thorough testing. The route planning feature, plotting transferred points, and saving the hole location were all inspected in code and in operation.

Disc (CW)

When everything was assembled on the disc, it was tested by throwing it outside on the east side of the Student Union. The assembled disc is shown in <insert figure of disc>. The disc was tested multiple times. The first test resulted in the buzzer breaking off, but everything else worked and data was sent to the app. The buzzer was reattached and the disc was tested again. Everything worked and parts remained on the disc. The app would parse data when it reads an end of transmission string "*FF*" which was accidentally left out of the Arduino code. The ending code was added into the Arduino sketch and the disc was tested again. When it landed, the battery cable broke. A replacement battery was obtained and the parts on the disc were reinforced by tying steel thread around components into the disc. The disc was tested again and it successfully landed, transferred data, and plotted the GPS data.

6. Financial Budget (SG & BL)

Table 14: Proposed Parts List

Ref. Des.	Part Name	Manufacturer	Part Number	Price	Weight	Qty	Website
U1	Controller	Adafruit	659	$24.95	4.40 g	1	http://www.adafruit.com/product/659
U2	GPS Chip	Adafruit	1059	$39.95	5.43 g	1	http://www.adafruit.com/product/1059
U3	IMU Chip	Adafruit	2020	$19.95	2.00 g	1	http://www.adafruit.com/product/2020
P1	Piezo Buzzer	Mallory	MSO206NR	$8.75	3.50 g	1	http://www.digikey.com/product-detail/en/MSO206NR/458-1163-ND/2442606
U4	microSD Reader	Adafruit	254	$14.95	3.43 g	1	http://www.adafruit.com/product/254
U5	Bluetooth LE Chip	Adafruit	1697	$19.95	1.80 g	1	http://www.adafruit.com/product/1697
	Solar Panel	Adafruit	1485	$24.95	-	1	http://www.adafruit.com/product/1485
U6	Battery Charger	Adafruit	390	$17.50	n/a	1	http://www.adafruit.com/products/390
B1	Battery	Adafruit	1578	$7.95	10.5 g	1	http://www.adafruit.com/product/1578
			TOTAL	$178.90	31.06 g		

Table 15: Parts Request 1

Qty.	Part Num.	Description	Cost	Cost
1	659	FLORA Microcontroller	$19.95	$19.95
1	1059	Flora GPS Module	39.95	39.95
1	2020	Flora Accelerometer/Gyroscope/Magnetometer	19.95	19.95
1	MSO206NR	BUZZ PIEZO CIRC 23MM RADIAL	8.75	8.75
1	254	MicroSD card breakout board+	14.95	14.95
1	1697	Bluefruit LE - Bluetooth Low Energy (BLE 4.0)	19.95	19.95
1	1485	Flexible 6V 1W Solar Panel	24.95	24.95
1	390	USB / DC / Solar Lithium Ion/Polymer charger - v2	17.50	17.50
1	1578	Lithium Ion Polymer Battery - 3.7v 500mAh	7.95	7.95

Table 16: Parts Request 2

Qty.	Part Num.	Description	Cost	Cost
2	MSO206NR	BUZZ PIEZO CIRC 23MM RADIAL	$8.75	$17.50
2	102	SD / MicroSD Memory Card	7.95	15.90
1	254	MicroSD card breakout board+	14.95	14.95
1	1697	Bluefruit LE - Bluetooth Low Energy (BLE 4.0)	19.95	19.95
1	1578	Lithium Ion Polymer Battery - 3.7v 500mAh	7.95	7.95
1	641	Conductive Thread	6.95	6.95
2	O-135	Blizzard Champion Disc Golf - Orange, weight 135	13.99	27.98

Table 17: Master Budget

Date	Item	Amount
11/17/2014	Initial Team Budget of $400	$ 400.00
11/18/2014	Parts Request Form 1	$ (173.90)
1/20/2015	Parts Request Form 2	$ (111.18)

Remaining balance:	**$ 114.92**

7. Project Schedules (BL)

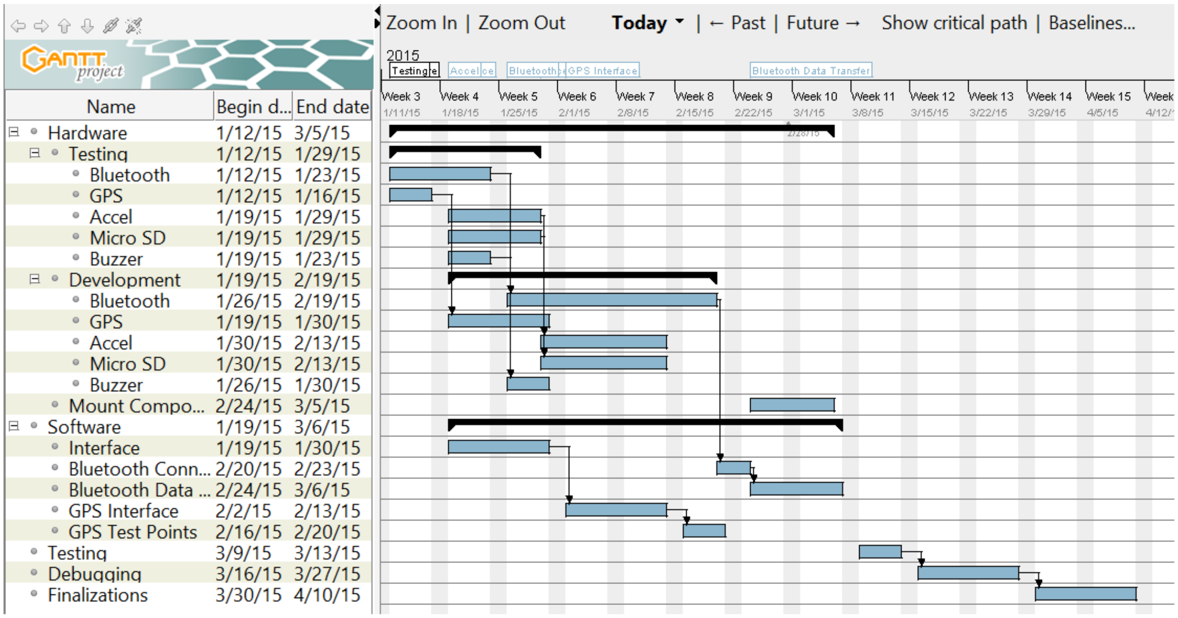
Midterm Report Gantt Chart

Name	Dur...	Begin date	End date	Members
Midterm Report	40	8/25/14	10/17/14	ALL
1. Problem Statement	35	8/25/14	10/10/14	ALL
Need Statement	6	8/29/14	9/5/14	CW
Objective Statement	31	8/29/14	10/10/14	CW
Research	26	8/29/14	10/3/14	CW
GPS	26	8/29/14	10/3/14	
Micro Electromech...	26	8/29/14	10/3/14	
Accelerometers	26	8/29/14	10/3/14	
Gyroscopes	26	8/29/14	10/3/14	
Magnetometer	26	8/29/14	10/3/14	
Android Application	16	9/12/14	10/5/14	NS
Flora Microcontroller	25	9/1/14	10/5/14	NS
Marketing Requiremen...	6	9/5/14	9/12/14	ALL
Objective Tree	1	8/25/14	8/25/14	CW
2. Design Requirements ...	6	9/5/14	9/12/14	ALL
Engineering Requirem...	6	9/5/14	9/12/14	ALL
3. Accepted Technical Des...	22	9/15/14	10/14/14	
HW Level 0 Block Diag...	22	9/15/14	10/14/14	CW
HW Level 0 Functional ...	22	9/15/14	10/14/14	CW
HW Level 1 Block Diag...	22	9/15/14	10/14/14	CW
HW Level 1 Functional ...	22	9/15/14	10/14/14	CW
HW Level 2 Block Diag...	22	9/15/14	10/14/14	CW
HW Level 2 Functional ...	22	9/15/14	10/14/14	SG, CW
SW Level 0 Block Diag...	9	9/29/14	10/9/14	CW
SW Level 0 Functional ...	9	9/29/14	10/9/14	NS
SW Level 1 Block Diag...	9	9/29/14	10/9/14	NS
SW Level 1 Functional ...	10	9/29/14	10/12/14	NS
SW Level 2 Block Diag...	10	9/29/14	10/12/14	NS
SW Level 2 Functional ...	10	9/29/14	10/12/14	NS
4. Project Schedules	37	8/25/14	10/14/14	BL
5. Design Team Information	1	8/25/14	8/25/14	ALL
Midterm Design Presentat...	4	10/14/14	10/17/14	ALL
Slide Show	3	10/14/14	10/16/14	ALL
Presentation 2:15-4:00	1	10/17/14	10/17/14	ALL

Final Report Gantt Chart

Name	Dur...	Begin date	End date	Members
Final Report	108	8/25/14	1/21/15	ALL
1. Problem Statement	60	8/25/14	11/15/14	ALL
Need Statement	6	8/29/14	9/5/14	CW
Objective Statement	31	8/29/14	10/10/14	CW
Modification of Projec...	11	10/17/14	10/31/14	CW
Research	56	8/29/14	11/15/14	CW
GPS	26	8/29/14	10/3/14	
Micro Electromec...	26	8/29/14	10/3/14	
Accelerometers	26	8/29/14	10/3/14	
Gyroscopes	26	8/29/14	10/3/14	
Magnetometer	26	8/29/14	10/3/14	
Android Application	16	9/12/14	10/3/14	NS
Flora Microcontrol...	25	9/1/14	10/3/14	NS
Serial Interfaces	1	10/17/14	10/17/14	NS
Universal Asynch....	1	10/17/14	10/17/14	NS
Serial Peripheral ...	2	11/13/14	11/15/14	NS
Inter-Integrated C...	1	11/13/14	11/13/14	NS
Marketing Requirem...	6	9/5/14	9/12/14	ALL
Objective Tree	1	8/25/14	8/25/14	CW
2. Design Requirements...	6	9/5/14	9/12/14	ALL
3. Accepted Technical D...	99	9/5/14	1/21/15	ALL
HW Level 0 Block Dia...	22	9/15/14	10/14/14	CW
HW Level 0 Function...	22	9/15/14	10/14/14	CW
HW Level 1 Block Dia...	22	9/15/14	10/14/14	CW
HW Level 1 Function...	22	9/15/14	10/14/14	CW
HW Level 2 Block Dia...	22	9/15/14	10/14/14	CW
HW Level 2 Function...	22	9/15/14	10/14/14	SG, CW
Tracking Device Sch...	5	11/24/14	11/28/14	SG, CW
Piezoelectric Buzzer	1	11/24/14	11/24/14	SG
Power Calculations	3	10/15/14	10/17/14	SG
Global Positioning Sy...	2	11/25/14	11/26/14	CW
Inertial Measurment ...	6	9/5/14	9/12/14	CW
Engineering Req...	6	9/5/14	9/12/14	ALL
Hardware Mounting	1	11/24/14	11/24/14	SG
Weight Experiment	2	11/13/14	11/15/14	CW
MicroSD Card Break...	11	11/17/14	12/1/14	BL
Bluetooth	11	11/17/14	12/1/14	BL
SW Level 0 Block Dia...	9	9/29/14	10/9/14	CW
SW Level 0 Function...	9	9/29/14	10/9/14	NS
SW Level 1 Block Dia...	9	9/29/14	10/9/14	NS
SW Level 1 Function...	10	9/29/14	10/10/14	NS
SW Level 2 Block Dia...	10	9/29/14	10/10/14	NS
SW Level 2 Function...	10	9/29/14	10/10/14	NS
Microsontroller Contr...	1	1/21/15	1/21/15	NS
4. Parts List	12	11/3/14	11/18/14	SG, BL
5. Project Schedules	67	9/1/14	12/2/14	BL
Midterm Report	1	9/1/14	9/1/14	BL
Final Report	2	12/1/14	12/2/14	BL
6. Design Team Informat...	1	8/25/14	8/25/14	ALL
7. Conclusions & Recom...	2	11/27/14	11/28/14	SG
8. References	1	11/27/14	11/27/14	ALL
9. Appendices	2	11/27/14	11/28/14	ALL
Midterm Design Present...	4	10/14/14	10/17/14	ALL
Final Design Presentation	4	12/1/14	12/4/14	ALL
Slide Show	3	12/1/14	12/3/14	ALL
Presentation 2:15-4:00	1	12/4/14	12/4/14	ALL

Project Design Gantt Chart



8. Design Team Information (SG, BL, NS, CW)

Team Member	Position	Major
Shane Gamble	Hardware Manager	Electrical Engineering
Brandon Linhart	Archivist	Computer Engineering
Noah Sanor	Software Manager	Computer Engineering
Christian Wallenfelsz	Project Leader	Electrical Engineering

9. Conclusions & Recommendations (CW, BL)

The goal of the project was to design a device which could be attached to a disc and help locate it after it was thrown. The final implementation could locate the disc and it could display the flight path of the disc on the app. The project was a complete success. There were power issues which did not allow every component to be powered, write to the SD card, and read from the GPS. The power issue limited the amount of data which could be recorded during a flight. Therefore, metrics about a throw were trimmed down to determine total throw distance and angle of throw relative to the hole.

The team members involved in this project have decided on few recommendations which could improve on implementation; given additional budget or desire to market this product. For instance, the electrical components could have been eliminated and replaced with a custom designed component containing each of their required functions. This would help in eliminating cost, as well as the need to distribute the weight evenly over the disc. Additionally, this would eliminate the exposed wires between components on the underside of the disc. Further, the single component could be enclosed under a protective layer, which would increase durability from landing shock and defend against moisture.

10. References

Acharya, R. (2014). 1.3. Referencing A Position. In *Understanding Satellite Navigation*. Academic Press.

Adafruit Industries, "Getting Started with FLORA," Adafruit Flora datasheet, June 2014.

Bartlett, D. (2013). *Essentials of Positioning and Location Technology*. Cambride University Press.

Chen, X., Parini, C., Collins, B., Yao, Y., & Rehmen, M. (2012). History of GNSS. In *Antennas for Global Navigation Satellite Systems*. John Wiley & Sons.

J. Liu and J. Yu, "Research on Development of Android Applications," in *Intelligent*

Jones, T., & Nenadic, N. (2013). *Electromechanics and MEMS*. Cambride University Press.

Kleppner, D., & Kolenkow, R. (2013). 8.3 Gyroscopes. In *An Introduction to Mechanics* (2nd ed.). Cambride University Press.

Networks and Intelligent Systems (ICINIS), 2011 4th International Conference on, pp.69-72, 1-3 Nov. 2011. doi: 10.1109/ICINIS.2011.40

T. Lindholm, F. Yellin, G. Bracha and A. Buckley, *The Java® Virtual Machine Specification*, 7th ed., Redwood City, California: Oracle America, Inc., 2013, p. 1-2.

Petrovski, I. (2014). GNSS ground and space segments. In *GPS, GLONASS, Galileo, and BeiDou for Mobile Devices*. Cambride University Press.

Untitled diagram of ECEF coordinate system. Retrieved October 1, 2014 from http://upload.wikimedia.org/wikipedia/commons/6/6b/ECEF.png

11. Appendices

Table 18: Component Datasheet Links

Ref. Des.	Part Name	Datasheet Link
U1	Controller	https://learn.adafruit.com/downloads/pdf/getting-started-with-flora.pdf
U2	GPS Chip	http://www.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf
U3	IMU Chip	http://www.adafruit.com/datasheets/LSM9DS0.pdf
P1	Piezo Buzzer	http://www.mallory-sonalert.com/specifications/MSO206NR.PDF
U6	Battery Charger	http://www.adafruit.com/datasheets/MCP73871.pdf
B1	Battery	https://www.adafruit.com/images/product-files/1578/C1854%20PKCell%20Datasheet%20Li-Polymer%20503035%20500mAh%203.7V%20with%20PCM.pdf

GPS and SD card test Arduino sketch code (*midterm_GPS_demo) (CW)*

```
#include <SD.h>
#include <SPI.h>
#include <Wire.h>

void notify(void);

//NMEA command sentences
#define PMTK_SET_NMEA_OUTPUT_RMCONLY "$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n"
#define PMTK_SET_NMEA_UPDATE_5HZ  "$PMTK220,200*2C\r\n"
#define PMTK_SET_NMEA_BAUDRATE "$PMTK251,115200*1F\r\n"
#define PMTK_API_SET_FIX_CTL_5HZ  "$PMTK300,200,0,0,0,0*2F\r\n"

// SD card utilities
Sd2Card card;
SdVolume volume;
SdFile root;

const int CS = 10;  // use 10 for Adafruit product
File fp;

volatile unsigned int loops = 0;

void setup()
{
  // NOTE: RECONNECT GPS EACH test time.
  // the baudrate needs to reset on the GPS

  pinMode(7,OUTPUT);
```

```
  digitalWrite(7,LOW);


  char c; // this char is for reading GPS

  Serial.begin(115200);
  delay(5000);
  Serial1.begin(9600);
  delay(5000);

  // Delay 120 seconds for GPS to get a fix
  delay(120000);

  // configure the GPS
  Serial1.write(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  Serial.println("\nGPS set to RMC sentences only");
  delay(4000);
  Serial1.write(PMTK_SET_NMEA_BAUDRATE);
  Serial.println("\nGPS baudrate set to 115200");
  delay(4000);
  Serial1.end();
  delay(4000);
  Serial1.begin(115200);
  delay(5000);
  Serial1.write(PMTK_API_SET_FIX_CTL_5HZ);
  Serial.println("\nGPS Fix rate changed to 5 Hz");
  delay(4000);
  Serial1.write(PMTK_SET_NMEA_UPDATE_5HZ);
  Serial.println("\nGPS set to send location at 5 Hz");
  delay(4000);

  notify();

//  SD.begin(CS); -------------------------------------------
//  //create GPS log on the SD card
//  if (!SD.begin(CS))
//    Serial.println("\nSD card init failure");
//  else
//    Serial.println("\nSD card init success");

  delay(3000);

//  // checks for GPS log, removes and creates new if it exists
//  if (SD.exists("GPS_LOG.txt"))
//  {
//    SD.remove("GPS_LOG.txt");
//    Serial.println("\nremoved old GPS_LOG.txt");
//    delay(100);
//    fp = SD.open("GPS_LOG.txt", FILE_WRITE);
//  }
//  else
//    fp = SD.open("GPS_LOG.txt", FILE_WRITE);
//
//  delay(100);
//  Serial.println("\ncreated GPS_LOG.txt");
//
```

```
//  delay(1000);

  // writes 8000 characters to SD card
  //Serial.println("\nabout to read GPS...");
  while(loops < 30000)
  {
    if (Serial1.available())
    {
      c = Serial1.read();
      //fp.write(c);
      loops++;
      Serial.write(c);
    }
  }

  delay(100);
  //fp.close();
  Serial.println("\ndone reading GPS");

  digitalWrite(7,HIGH);

}  // end setup

void loop()
{

}

void notify(void)
{
  int limit = 0;

  while(limit < 6)
  {
    digitalWrite(7,HIGH);
    delay(300);
    digitalWrite(7,LOW);
    delay(300);
    limit++;
  }
}
```

IMU Arduino sketch test code (*midterm_IMU_demo) (CW)*

```
#include <Wire.h>

#define lsm_accmag   (0x1D)  // accelerometer and magnetometer have same address
#define lsm_gyro     (0x6B)  // gyro address

// accelerometer registers
#define WHO_AM_I     (0x0F)
#define CTRL_REG0_XM (0x1F)
#define CTRL_REG1_XM (0x20)
#define CTRL_REG2_XM (0x21)
```

```
#define OUT_X_L_A    (0x28)
#define OUT_X_H_A    (0x29)
#define OUT_Y_L_A    (0x2A)
#define OUT_Y_H_A    (0x2B)
#define OUT_Z_L_A    (0x2C)
#define OUT_Z_H_A    (0x2D)

// magnetometer registers
#define CTRL_REG5_XM (0x24)
#define CTRL_REG6_XM (0x25)
#define CTRL_REG7_XM (0x26)
#define OUT_X_L_M    (0x08)
#define OUT_X_H_M    (0x09)
#define OUT_Y_L_M    (0x0A)
#define OUT_Y_H_M    (0x0B)
#define OUT_Z_L_M    (0x0C)
#define OUT_Z_H_M    (0x0D)

// gyro registers
#define CTRL_REG1_G  (0x20)
#define CTRL_REG4_G  (0x23)
#define OUT_X_L_G    (0x28)
#define OUT_X_H_G    (0x29)
#define OUT_Y_L_G    (0x2A)
#define OUT_Y_H_G    (0x2B)
#define OUT_Z_L_G    (0x2C)
#define OUT_Z_H_G    (0x2D)


// prototypes for IMU init
void initGYRO(void);
void initACCEL(void);
void initMAG(void);

int led = 7;  // FLORA pin 7 is connected to LED (red)

// sensitivity characteristics from Table 3 of LSM9DS0 datasheet
float sensitivity_A_2G = 0.061;
float sensitivity_A_4G = 0.122;
float sensitivity_A_6G = 0.183;

float sensitivity_M_2G = 0.08;
float sensitivity_M_4G = 0.16;
float sensitivity_M_8G = 0.32;
float sensitivity_M_12G = 0.48;

float sensitivity_G_245 = 8.75;
float sensitivity_G_500 = 17.5;
float sensitivity_G_2K = 70;

//--------------------------------------------------
//--------------------------------------------------

void setup() {

  Serial.begin(115200);
```

```
  initACCEL();
  delay(1000);
  initMAG();
  delay(1000);
  initGYRO();
  delay(1000);

  // confirm successful init
  pinMode(led, OUTPUT);
  for (int i=0;i<10;i++) {
    digitalWrite(led, HIGH);
    delay(100);
    digitalWrite(led, LOW);
    delay(100);
  }

  // identify device
  unsigned int who = 0;
  Wire.beginTransmission(lsm_gyro);
  Wire.write(WHO_AM_I);
  Wire.endTransmission();
  Wire.requestFrom(lsm_gyro, 1);
  who = Wire.read();
  Serial.println(who);

  digitalWrite(led, HIGH);

  delay(2000);

}  // end setup ------------------------------------

//---------------------------------------------------
//---------------------------------------------------

void loop() {

  unsigned int xl = 0;
  int          xh = 0;
  unsigned int yl = 0;
  int          yh = 0;
  unsigned int zl = 0;
  int          zh = 0;
  float x = 0;
  float y = 0;
  float z = 0;

//  // read all the bytes from each accel register
//  Wire.beginTransmission(lsm_gyro);
//  Wire.write(OUT_X_L_G);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_gyro, 1);
//  xl = Wire.read();
//  //Serial.println(xl);
//
//  Wire.beginTransmission(lsm_gyro);
```

```
//  Wire.write(OUT_X_H_G);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_gyro, 1);
//  xh = Wire.read();
//  //Serial.println(xh);
//
//  Wire.beginTransmission(lsm_gyro);
//  Wire.write(OUT_Y_L_G);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_gyro, 1);
//  yl = Wire.read();
//
//  Wire.beginTransmission(lsm_gyro);
//  Wire.write(OUT_Y_H_G);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_gyro, 1);
//  yh = Wire.read();
//
//  Wire.beginTransmission(lsm_gyro);
//  Wire.write(OUT_Z_L_G);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_gyro, 1);
//  zl = Wire.read();
//
//  Wire.beginTransmission(lsm_gyro);
//  Wire.write(OUT_Z_H_G);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_gyro, 1);
//  zh = Wire.read();
//
//  // form all the measurements from 2's complement
//  xh <<= 8;
//  xh |= xl;
//  x = xh * sensitivity_G_245;
//  x /= 1000;
//  //x *= 9.81;
//
//  yh <<= 8;
//  yh |= yl;
//  y = yh * sensitivity_G_245;
//  y /= 1000;
//  //y *= 9.81;
//
//  zh <<= 8;
//  zh |= zl;
//  z = zh * sensitivity_G_245;
//  z /= 1000;
//  //z *= 9.81;
//
//  Serial.print(x);
//  Serial.print(",");
//  Serial.print(y);
//  Serial.print(",");
//  Serial.println(z);

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

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

//  // read all the bytes from each mag register
//  Wire.beginTransmission(lsm_accmag);
//  Wire.write(OUT_X_L_M);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_accmag, 1);
//  xl = Wire.read();
//
//  Wire.beginTransmission(lsm_accmag);
//  Wire.write(OUT_X_H_M);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_accmag, 1);
//  xh = Wire.read();
//
//  Wire.beginTransmission(lsm_accmag);
//  Wire.write(OUT_Y_L_M);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_accmag, 1);
//  yl = Wire.read();
//
//  Wire.beginTransmission(lsm_accmag);
//  Wire.write(OUT_Y_H_M);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_accmag, 1);
//  yh = Wire.read();
//
//  Wire.beginTransmission(lsm_accmag);
//  Wire.write(OUT_Z_L_M);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_accmag, 1);
//  zl = Wire.read();
//
//  Wire.beginTransmission(lsm_accmag);
//  Wire.write(OUT_Z_H_M);
//  Wire.endTransmission();
//  Wire.requestFrom(lsm_accmag, 1);
//  zh = Wire.read();
//
//  // form all the measurements from 2's complement
//  xh <<= 8;
//  xh |= xl;
//  x = xh * sensitivity_M_2G;
//  x /= 1000;
//
//  yh <<= 8;
//  yh |= yl;
//  y = yh * sensitivity_M_2G;
//  y /= 1000;
//
//  zh <<= 8;
//  zh |= zl;
//  z = zh * sensitivity_M_2G;
//  z /= 1000;
//
//  Serial.print(x);
```

```
//  Serial.print(",");
//  Serial.print(y);
//  Serial.print(",");
//  Serial.println(z);

//------------------------------------------------
//------------------------------------------------

  // read all the bytes from each accel register
  Wire.beginTransmission(lsm_accmag);
  Wire.write(OUT_X_L_A);
  Wire.endTransmission();
  Wire.requestFrom(lsm_accmag, 1);
  xl = Wire.read();

  Wire.beginTransmission(lsm_accmag);
  Wire.write(OUT_X_H_A);
  Wire.endTransmission();
  Wire.requestFrom(lsm_accmag, 1);
  xh = Wire.read();

  Wire.beginTransmission(lsm_accmag);
  Wire.write(OUT_Y_L_A);
  Wire.endTransmission();
  Wire.requestFrom(lsm_accmag, 1);
  yl = Wire.read();

  Wire.beginTransmission(lsm_accmag);
  Wire.write(OUT_Y_H_A);
  Wire.endTransmission();
  Wire.requestFrom(lsm_accmag, 1);
  yh = Wire.read();

  Wire.beginTransmission(lsm_accmag);
  Wire.write(OUT_Z_L_A);
  Wire.endTransmission();
  Wire.requestFrom(lsm_accmag, 1);
  zl = Wire.read();

  Wire.beginTransmission(lsm_accmag);
  Wire.write(OUT_Z_H_A);
  Wire.endTransmission();
  Wire.requestFrom(lsm_accmag, 1);
  zh = Wire.read();

  // form all the measurements from 2's complement
  xh <<= 8;
  xh |= xl;
  x = xh * sensitivity_A_2G;
  x /= 1000;
  x *= 9.81;

  yh <<= 8;
  yh |= yl;
  y = yh * sensitivity_A_2G;
  y /= 1000;
```

```
  y *= 9.81;

  zh <<= 8;
  zh |= zl;
  z = zh * sensitivity_A_2G;
  z /= 1000;
  z *= 9.81;

  Serial.print(x);
  Serial.print(",");
  Serial.print(y);
  Serial.print(",");
  Serial.println(z);

  //delay(10);
}  // end loop -------------------------------------

//--------------------------------------------------
//--------------------------------------------------

void initMAG(void) {

  // set default magnetometer settings
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG7_XM);
  Wire.write(0);  // continuous conversion mode
  Wire.endTransmission();

  // set magnetic sensitivity
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG6_XM);
  Wire.write(0);  // 2g
  // Wire.write(0x20);  // 4g
  // Wire.write(0x40);  // 8g
  // Wire.write(0x60);  // 12g
  Wire.endTransmission();

  // set mag refresh rate
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG5_XM);
  Wire.write(0xC);  // 25 Hz
  // Wire.write(0x10);  // 50 Hz
  Wire.endTransmission();
}

//--------------------------------------------------
//--------------------------------------------------

void initACCEL(void) {

  // set accelerometer to default use
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG0_XM);
  Wire.write(0);
  Wire.endTransmission();
```

```
  // set accelerometer to output at 50 Hz
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG1_XM);
  Wire.write(0x57);
  Wire.endTransmission();

  // set accelerometer to 2g scale
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG2_XM);
  Wire.write(0);
  Wire.endTransmission();
}

//-------------------------------------------------
//-------------------------------------------------

void initGYRO(void) {

  // set gyro to default
  Wire.beginTransmission(lsm_gyro);
  Wire.write(CTRL_REG1_G);
  Wire.write(0x0F);  // 95 Hz
  //Wire.write(0x67);  // 190 Hz
  Wire.endTransmission();

  // set gyro sensitivity
  Wire.beginTransmission(lsm_gyro);
  Wire.write(CTRL_REG4_G);
  Wire.write(0);  // 245 dps
  //Wire.write(0x08);  // 500 dps
  //Wire.write(0x10);  // 2K dps
  Wire.endTransmission();
}
```

Bluetooth Arduino sketch test code (BL)

```
#include <SPI.h>
#include "Adafruit_BLE_UART.h"

// Connect CLK/MISO/MOSI to hardware SPI
// e.g. On UNO & compatible: CLK = 13, MISO = 12, MOSI = 11
#define ADAFRUITBLE_REQ 9
#define ADAFRUITBLE_RDY 2     // This should be an interrupt pin
#define ADAFRUITBLE_RST 6

Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ,
ADAFRUITBLE_RDY, ADAFRUITBLE_RST);
/**************************************************************************/
/*!
   Configure the Arduino and start advertising with the radio
*/
/**************************************************************************/
```

```
void setup(void)
{
  Serial.begin(9600);
  while(!Serial); // Leonardo/Micro should wait for serial init
  Serial.println(F("Adafruit Bluefruit Low Energy nRF8001 Print echo demo"));

  // BTLEserial.setDeviceName("NEWNAME"); /* 7 characters max! */

  BTLEserial.begin();
}

aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;

void loop()
{
  // Tell the nRF8001 to do whatever it should be working on.
  BTLEserial.pollACI();

  // Ask what is our current status
  aci_evt_opcode_t status = BTLEserial.getState();
  // If the status changed....
  if (status != laststatus) {
    // print it out!
    if (status == ACI_EVT_DEVICE_STARTED) {
        Serial.println(F("* Advertising started"));
    }
    if (status == ACI_EVT_CONNECTED) {
        Serial.println(F("* Connected!"));
    }
    if (status == ACI_EVT_DISCONNECTED) {
        Serial.println(F("* Disconnected or advertising timed out"));
    }
    // OK set the last status change to this one
    laststatus = status;
  }

  if (status == ACI_EVT_CONNECTED) {
    // Lets see if there's any data for us!
    if (BTLEserial.available()) {
      Serial.print("* "); Serial.print(BTLEserial.available()); Serial.println(F(" bytes available from
BTLE"));
    }
    // OK while we still have something to read, get a character and print it out
    while (BTLEserial.available()) {
      char c = BTLEserial.read();
      Serial.print(c);
```

```
    }

    // Next up, see if we have any data to get from the Serial console

    if (Serial.available()) {
      // Read a line from Serial
      Serial.setTimeout(100); // 100 millisecond timeout
      String s = Serial.readString();

      // We need to convert the line to bytes, no more than 20 at this time
      uint8_t sendbuffer[20];
      s.getBytes(sendbuffer, 20);
      char sendbuffersize = min(20, s.length());

      Serial.print(F("\n* Sending -> \"")); Serial.print((char *)sendbuffer); Serial.println("\"");

      // write the data
      BTLEserial.write(sendbuffer, sendbuffersize);
    }
  }
}
```

Final Project Arduino Sketch (CW, BL)

```
#include <Wire.h>
#include <SPI.h>
#include <EEPROM.h>
#include "Adafruit_BLE_UART.h"

//NMEA command sentences
#define PMTK_SET_NMEA_OUTPUT_RMCONLY "$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n"
#define PMTK_SET_NMEA_UPDATE_5HZ  "$PMTK220,200*2C\r\n"
#define PMTK_SET_NMEA_UPDATE_2HZ  "$PMTK220,500*2B\r\n"
#define PMTK_SET_NMEA_UPDATE_1HZ  "$PMTK220,1000*1F\r\n"
#define PMTK_SET_NMEA_BAUDRATE "$PMTK251,115200*1F\r\n"
#define PMTK_API_SET_FIX_CTL_5HZ  "$PMTK300,200,0,0,0,0*2F\r\n"

#define lsm_accmag   (0x1D)  // accelerometer and magnetometer have same address
#define lsm_gyro     (0x6B)  // gyro address
// accelerometer registers
#define WHO_AM_I     (0x0F)
#define CTRL_REG0_XM (0x1F)
#define CTRL_REG1_XM (0x20)
#define CTRL_REG2_XM (0x21)
#define OUT_X_L_A    (0x28)
#define OUT_X_H_A    (0x29)
#define OUT_Y_L_A    (0x2A)
#define OUT_Y_H_A    (0x2B)
#define OUT_Z_L_A    (0x2C)
```

```
#define OUT_Z_H_A    (0x2D)
// gyro registers
#define CTRL_REG1_G  (0x20)
#define CTRL_REG4_G  (0x23)
#define OUT_X_L_G    (0x28)
#define OUT_X_H_G    (0x29)
#define OUT_Y_L_G    (0x2A)
#define OUT_Y_H_G    (0x2B)
#define OUT_Z_L_G    (0x2C)
#define OUT_Z_H_G    (0x2D)



// prototypes for IMU init
void initGYRO(void);
void initACCEL(void);
void bluetooth(void);
float getValue(int device, int reg_low, int reg_high, float scale);

int led = 7;  // FLORA pin 7 is connected to LED (red)

// sensitivity characteristics from Table 3 of LSM9DS0 datasheet
float sensitivity_A_2G = 0.00059841;  // 0.061 / 1000 * 9.81
//float sensitivity_A_4G = 0.00119682;  // 0.122 / 1000 * 9.81
//float sensitivity_A_6G = 0.00179523;  // 0.183 / 1000 * 9.81

//float sensitivity_G_245 = 0.00875; // 8.75 / 1000
//float sensitivity_G_500 = 0.0175;  // 17.5 / 1000
float sensitivity_G_2K = 0.070;    // 70 / 1000

volatile char c;
volatile unsigned int loops = 0;
volatile float ax = 0;
volatile float ay = 0;
volatile float az = 0;
volatile float gx = 0;
volatile float gy = 0;
volatile float gz = 0;

volatile char gps[80];
volatile int limit;
volatile int done;
volatile int blue;
volatile int index;
volatile int datadone;

volatile int index2;
volatile int count;

// Connect CLK/MISO/MOSI to hardware SPI
// e.g. On UNO & compatible: CLK = 13, MISO = 12, MOSI = 11
#define ADAFRUITBLE_REQ 9
#define ADAFRUITBLE_RDY 2     // This should be an interrupt pin, on Uno thats #2 or #3
#define ADAFRUITBLE_RST 6
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY,
ADAFRUITBLE_RST);
```

```
void setup() {

  Serial.begin(115200);
  delay(5000);
  Serial1.begin(9600);
  delay(5000);
  done = 0;
  // configure GPS
  Serial1.write(PMTK_SET_NMEA_OUTPUT_RMCONLY);
//  Serial.println("\nGPS set to RMC sentences only");
  delay(4000);
  Serial1.write(PMTK_SET_NMEA_BAUDRATE);
//  Serial.println("\nGPS baudrate set to 115200");
  delay(4000);
  Serial1.end();
  delay(4000);
  Serial1.begin(115200);
  delay(5000);
  Serial1.write(PMTK_API_SET_FIX_CTL_5HZ);
//  Serial.println("\nGPS Fix rate changed to 5 Hz");
  delay(4000);
  Serial1.write(PMTK_SET_NMEA_UPDATE_1HZ);
//  Serial.println("\nGPS set to send location at 2 Hz");
  delay(4000);

  // configure IMU
  delay(1000);
  initACCEL();
//  Serial.println("accelerometer initialized");
  delay(1000);
  initGYRO();
//  Serial.println("gyro initialized");
  delay(1000);

  delay(90000);

  index=0;
  blue=0;
  datadone=0;
//  Serial.println("Entering loop");
  for(int i=0;i<3;i++)
  {
    analogWrite(12,250);
    delay(100);
    analogWrite(12,0);
    delay(100);
  }

//digitalWrite(led, HIGH);

}
```

```
void loop()
{
  gz = getValue(lsm_gyro, OUT_Z_L_G, OUT_Z_H_G, sensitivity_G_2K);  // perpendicular axis
    //Serial.println(gy);
    if ((gz < -1000) || (gz > 1000))
    {  // trigger threshold
//      Serial.println("Throwing");
      Serial.flush();
      index2 = 0;
      while(index2 < 750)
      {
        if (Serial1.available())
        {
          c = Serial1.read();

          if (c == '$')
          {
            loops++;
            count = 0;
            while (count < 45)
            {
              gps[count] = c;
              if (Serial1.available())
              {
                c = Serial1.read();
                count++;
                gps[count] = c;
              }
            }
            //Serial.println(gps);
            if ((gps[0]=='$') && (gps[1]=='G') && (gps[2]=='P') && (gps[3]=='R') && (gps[4]=='M') && (gps[5]=='C')
&& (gps[18]=='A'))
            {
              //Serial.println(gps);
              count = 20;
              EEPROM.write(index2, gps[0]);
              index2++;
              while (count < 44)
              {
                EEPROM.write(index2,gps[count]);
                //Serial.print(gps[count]);
                count++;
                index2++;
              }
            }
            else if ((gps[0]=='$') && (gps[1]=='G') && (gps[2]=='P') && (gps[3]=='R') && (gps[4]=='M') &&
(gps[5]=='C') && (gps[18]=='V'))
            {
              //Serial.println(gps);
              count = 20;
              EEPROM.write(index2, gps[0]);
              index2++;
              while (count < 44)
              {
                EEPROM.write(index2, gps[count]);
                //Serial.print(gps[count]);
```

```
                count++;
                index2++;
              }
            }
//            else
//              delay(1);
             //Serial.println();
           //Serial.println();
         }
    //      else
    //        Serial.println(c);
        }//------------------------------------------------------------------
      }
      delay(5000);

        delay(100);
//        Serial.println("Setting bluetoothRDY to true");
        blue=1;

        delay(100);

        if(blue==1)
        {
//          Serial.println("BluetoothRDY is true");
          bluetooth();
          done=0;
        }
  }  // continue to check trigger variable
  //
  //
  index = 0;
}


//--------------------------------------------------
//--------------------------------------------------

void bluetooth(void)
{
  analogWrite(12,250);
  delay(1000);
  analogWrite(12,0);
  delay(1000);

//  Serial.println("Now in the Bluetooth function");
  String r="";
  aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;

  delay(1000);
  Serial.println("Starting the Bluetooth");
  delay(500);
//  Serial.flush();
  BTLEserial.setDeviceName("WMD3.0");
  BTLEserial.begin();
```

```
  while(true)
  {
     //Serial.println("In while done==false stage");

     BTLEserial.pollACI();
     // Ask what is our current status
     aci_evt_opcode_t status = BTLEserial.getState();
     // If the status changed....
     if (status != laststatus)
     {
      // print it out!
      if (status == ACI_EVT_DEVICE_STARTED) {
//          Serial.println(F("* Advertising started"));
      }
      if (status == ACI_EVT_CONNECTED) {
//          Serial.println(F("* Connected!"));
      }
      if (status == ACI_EVT_DISCONNECTED) {
//          Serial.println(F("* Disconnected or advertising timed out"));
      }
      // OK set the last status change to this one
      laststatus = status;
     }



     if (status == ACI_EVT_CONNECTED)
     {

      if(datadone==0)
      {
       for(int i=0;i<15;i++)
       {
        char w = char(EEPROM.read(index));
        r = r+w;
        index++;
       }
       if(index>750)
       {
        datadone=1;
        r = "$FF";
       }
      String s = r;
      uint8_t sendbuffer[30];
      s.getBytes(sendbuffer, 30);
      char sendbuffersize = min(20, s.length());
//          Serial.print(F("\n* Sending -> \""));
//          Serial.print((char *)sendbuffer); Serial.println("\"");
      BTLEserial.write(sendbuffer, sendbuffersize);
      r="";
      }
     if(datadone==1)
     {
      analogWrite(12,250);
```

```
          delay(200);
          analogWrite(12,0);
          delay(10000);
        }

      }
      else
      {
        analogWrite(12,250);
        delay(500);
        analogWrite(12,0);
        delay(500);
      }

 } //end while(done==false)
} //end bluetooth() function

void initACCEL(void) {

  // set accelerometer to default use
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG0_XM);
  Wire.write(0);
  Wire.endTransmission();

  // set accelerometer to output at 50 Hz
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG1_XM);
  Wire.write(0x57);
  Wire.endTransmission();

  // set accelerometer to 2g scale
  Wire.beginTransmission(lsm_accmag);
  Wire.write(CTRL_REG2_XM);
  Wire.write(0);
  Wire.endTransmission();
}

//--------------------------------------------------
//--------------------------------------------------

void initGYRO(void) {

  // set gyro to default
  Wire.beginTransmission(lsm_gyro);
  Wire.write(CTRL_REG1_G);
  Wire.write(0x0F);  // 95 Hz
  //Wire.write(0x67);  // 190 Hz
  Wire.endTransmission();

  // set gyro sensitivity
  Wire.beginTransmission(lsm_gyro);
  Wire.write(CTRL_REG4_G);
  //Wire.write(0);  // 245 dps
  //Wire.write(0x08);  // 500 dps
  Wire.write(0x10);  // 2K dps
```

```
  Wire.endTransmission();
}

//--------------------------------------------------
//--------------------------------------------------

float getValue(int device, int reg_low, int reg_high, float scale) {
  unsigned int low = 0;
  int high = 0;
  float value = 0;

  Wire.beginTransmission(device);
  Wire.write(reg_low);
  Wire.endTransmission();
  Wire.requestFrom(device, 1);
  low = Wire.read();

  Wire.beginTransmission(device);
  Wire.write(reg_high);
  Wire.endTransmission();
  Wire.requestFrom(device, 1);
  high = Wire.read();

  high <<= 8;
  high |= low;
  value = high * scale;
  return value;
}
```

Final Project App Code (NS)

File - C:\Users\BLinhart\Documents\GitHub\WMD\WMD\app\src\main\java\com\example\sdp11\wmd\DBHelper.java

```
package com.example.sdp11.wmd;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by Student on 1/27/2015.
 */
public class DBHelper extends SQLiteOpenHelper {
    public static final String TABLE_THROWS = "throws";

    public static final String COLUMN_THROW_ID = "throw_id";
    public static final String COLUMN_HOLE_ID = "hole_id";
    public static final String COLUMN_GAME_ID = "game_id";
    public static final String COLUMN_START_LAT = "starting_latitude";
    public static final String COLUMN_START_LONG = "starting_longitude";
    public static final String COLUMN_END_LAT = "ending_latitude";
    public static final String COLUMN_END_LONG = "ending_longitude";
    public static final String COLUMN_START_ACCEL_X = "starting_x_acceleration";
    public static final String COLUMN_START_ACCEL_Y = "starting_y_acceleration";
    public static final String COLUMN_START_TIME = "starting_time";
    public static final String COLUMN_END_TIME = "ending_time";
    public static final String COLUMN_SYNC_TIME = "sync_time";

    public static final String TABLE_CALC = "calc_data";

    public static final String COLUMN_INITIAL_DIRECTION = "initial_direction";
    public static final String COLUMN_FINAL_DIRECTION = "final_direction";
    public static final String COLUMN_TOTAL_DISTANCE = "total_distance";
    public static final String COLUMN_THROW_INTEGRITY = "throw_integrity";
    public static final String COLUMN_TOTAL_TIME = "total_time";


    private static final String DATABASE_NAME = "throw_data.db";
    private static final int DATABASE_VERSION = 1;

    private static final String THROWS_DATABASE_CREATE = "create table "
            + TABLE_THROWS + " ( "
            + COLUMN_THROW_ID
            + " integer primary key autoincrement, "
            + COLUMN_HOLE_ID + " integer not null, "
            + COLUMN_GAME_ID+ " integer not null, "
            + COLUMN_START_LAT + " double not null, "
            + COLUMN_START_LONG + " double not null, "
            + COLUMN_END_LAT + " double not null, "
            + COLUMN_END_LONG + " double not null, "
            + COLUMN_START_ACCEL_X + " double not null, "
```

```java
            + COLUMN_START_ACCEL_Y + " double not null, "
            + COLUMN_START_TIME + " integer not null, "
            + COLUMN_END_TIME + " integer not null, "
            + COLUMN_SYNC_TIME + " integer not null"
            + ");";

    private static final String CALC_DATABASE_CREATE = "create table "
            + TABLE_CALC + " ( "
            + COLUMN_THROW_ID
            + " integer primary key autoincrement, "
            + COLUMN_INITIAL_DIRECTION + " integer not null, "
            + COLUMN_FINAL_DIRECTION + " integer not null, "
            + COLUMN_TOTAL_DISTANCE + " double not null, "
            + COLUMN_THROW_INTEGRITY + " double not null, "
            + COLUMN_TOTAL_TIME + " double not null"
            + ");";

    public DBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_THROWS);
        db.execSQL(THROWS_DATABASE_CREATE);
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_CALC);
        db.execSQL(CALC_DATABASE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_THROWS);
        onCreate(db);
    }

}
```

File - C:\Users\BLinhart\Documents\GitHub\WMD\WMD\app\src\main\java\com\example\sdp11\wmd\TotalsData.java

```java
package com.example.sdp11.wmd;

/**
 * Created by nsanor on 2/10/2015.
 */
public class TotalsData {
    //Average distance, angle, sync counter to differentiate throws, etc.
    //Used as a global variable.
    //Create all as static variables.
    static private double averageDistance;
    static private double averageAngle;
    static private int syncCount;

    public static void loadTotalsData() {
        averageDistance = 10;
    }

    public static double getAverageDistance() {
        return averageDistance;
    }

    public static void setAverageDistance(double averageDistance) {
        TotalsData.averageDistance = averageDistance;
    }

    public static double getAverageAngle() {
        return averageAngle;
    }

    public static void setAverageAngle(double averageAngle) {
        TotalsData.averageAngle = averageAngle;
    }

    public static int getSyncCount() {
        return syncCount;
    }

    public static void setSyncCount(int syncCount) {
        TotalsData.syncCount = syncCount;
    }
}
```

File - C:\Users\BLinhart\Documents\GitHub\WMD\WMD\app\src\main\java\com\example\sdp11\wmd\MapFragment.java

```java
package com.example.sdp11.wmd;


import android.location.Location;
import android.os.Bundle;
import android.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import com.google.android.gms.common.GooglePlayServicesNotAvailableException;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapView;
import com.google.android.gms.maps.MapsInitializer;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.CameraPosition;
import com.google.android.gms.maps.model.Circle;
import com.google.android.gms.maps.model.CircleOptions;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.LatLngBounds;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import java.util.Stack;


/**
 * A simple {@link Fragment} subclass.
 */
public class MapFragment extends Fragment {

    double latitude = 41.13747;
    double longitude = -81.4743070000001;
    LatLngBounds bounds;
    private Circle circle;

    private MapView mapView;
    private GoogleMap googleMap;
    private CameraPosition cp;
    private Button mode;
    private Button undo;
    private TextView label;

    private boolean planning = false;

```

```java
    public MapFragment() {
        // Required empty public constructor
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_map, container, false);
        // Gets the MapView from the XML layout and creates it
        mapView = (MapView) view.findViewById(R.id.mapview);
        mapView.onCreate(savedInstanceState);
        mapView.onResume();

        final Stack<Marker> markerStack = new Stack<Marker>();

        mode = (Button) view.findViewById(R.id.button_mode);
        undo = (Button) view.findViewById(R.id.button_undo);
        label = (TextView) view.findViewById(R.id.mode_label);

        if(planning) {
            undo.setVisibility(View.VISIBLE);
            label.setText("Planning Mode");
        }

        mode.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(!planning) {
                    undo.setVisibility(View.VISIBLE);
                    label.setText("Planning Mode");
                    planning = true;
                }
                else {
                    undo.setVisibility(View.INVISIBLE);
                    label.setText("Normal Mode");
                    planning = false;
                }
            }
        });

        undo.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
```

```java
                if (!markerStack.empty()) {

                    //Remove last marker that was placed
                    Marker marker = markerStack.pop();
                    marker.remove();
                    //Remove the circle for that marker
                    if(circle != null) circle.remove();
                    //Get the next to last marker and re-add circle
                    if (!markerStack.empty()) {
                        marker = markerStack.pop();
                        plotRadius(marker.getPosition(), TotalsData.getAverageDistance());
                        markerStack.push(marker);
                    }
                }
            }
        });

        try {
            MapsInitializer.initialize(getActivity().getApplicationContext());
        } catch (Exception e) {
            e.printStackTrace();
        }

        googleMap = mapView.getMap();
        Location mCurrentLocation = MainActivity.getmCurrentLocation();

        // latitude and longitude
        if(mCurrentLocation != null) {
            latitude = mCurrentLocation.getLatitude();
            longitude = mCurrentLocation.getLongitude();
        }

        //calculateBounds();
        googleMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {

            @Override
            public void onMapClick(LatLng point) {
                //lstLatLngs.add(point);
                if(planning) {
                    Marker marker = plotUserPoint(point);
                    markerStack.push(marker);
                }
            }
        });

//        plotPoint(new LatLng(41.075850, -81.513317), false);
//        plotPoint(new LatLng(41.075867, -81.513300), false);
//        plotPoint(new LatLng(41.075867, -81.513300), false);
//        plotPoint(new LatLng(41.075850, -81.513283), false);
```

```
//        plotPoint(new LatLng(41.075867, -81.513267), false);
//        plotPoint(new LatLng(41.075850, -81.513267), false);
//        plotPoint(new LatLng(41.075850, -81.513250), false);
//        plotPoint(new LatLng(41.075850, -81.513233), false);
//        plotPoint(new LatLng(41.075867, -81.513233), false);
//        plotPoint(new LatLng(41.075867, -81.513233), false);

//        41.075850, -81.513233
//        41.075850, -81.513233
//        41.075867, -81.513250
//        41.075850, -81.513233
//        41.075850, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513250
//        41.075867, -81.513267
//        41.075867, -81.513267
//        41.075867, -81.513283
//        41.075867, -81.513283
//        41.075867, -81.513300
//        41.075867, -81.513317
//        41.075850, -81.513317
//        41.075850, -81.513333
//        41.075850, -81.513350
//        41.075850, -81.513367
//        41.075850, -81.513383
//        41.075850, -81.513383
//        41.075850, -81.513400
//        41.075850, -81.513417
//        41.075833, -81.513433
//        41.075833, -81.513450
//        41.075850, -81.513467
//        41.075850, -81.513467
//        41.075850, -81.513483
//        41.075850, -81.513500
//        41.075850, -81.513517
//        41.075850, -81.513517
//        41.075850, -81.513533
//        41.075850, -81.513550
//        41.075850, -81.513550
//        41.075850, -81.513567
//        41.075850, -81.513567
//        41.075850, -81.513583
```

```
//        41.075833, -81.513583
//        41.075850, -81.513600
//        41.075850, -81.513600
//        41.075850, -81.513617
//        41.075833, -81.513617
//        41.075850, -81.513633
//        41.075850, -81.513633
//        41.075850, -81.513633
//        41.075850, -81.513633
//        41.075867, -81.513650
//        41.075850, -81.513650
//        41.075850, -81.513667
//        41.075833, -81.513667
//        41.075817, -81.513667
//        41.075817, -81.513667
//        41.075800, -81.513667
//        41.075783, -81.513667
//        41.075767, -81.513667
//        41.075750, -81.513667
//        41.075750, -81.513683
//        41.075733, -81.513683
//        41.075717, -81.513683
//        41.075700, -81.513683

        googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
        //CameraUpdate cu = CameraUpdateFactory.newLatLngBounds(bounds, 0);
//        CameraPosition cameraPosition = new CameraPosition.Builder()
//                .target(new LatLng(latitude, longitude)).zoom(14).build();
        CameraPosition cameraPosition = new CameraPosition.Builder()
                .target(new LatLng(41.075850, -81.513317)).zoom(googleMap.getMaxZoomLevel()).build();
        googleMap.animateCamera(CameraUpdateFactory
                .newCameraPosition(cameraPosition));

        return view;
    }

    @Override
    public void onResume() {

        super.onResume();
        mapView.onResume();

        MapsInitializer.initialize(getActivity());

        if (cp != null) {
            googleMap.moveCamera(CameraUpdateFactory.newCameraPosition(cp));
            cp = null;
        }
        else Log.e("", "Not saved");
```

```java
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
    }

    @Override
    public void onPause() {
        mapView.onPause();
        super.onPause();

        cp = googleMap.getCameraPosition();
//        googleMap = null;


    }

    private void calculateBounds() {
        //Add formula to calculate distance between lat and long lines at current location
        //Want ~5 mile bounds (8.4 km)
        //About 69km between lines on average
        //About .12 degrees for bounds, 0.6 on either side
        LatLng northeast = new LatLng(latitude + 0.06, longitude + 0.06);
        LatLng southwest = new LatLng(latitude - 0.06, longitude - 0.06);
        LatLngBounds.Builder builder = new LatLngBounds.Builder();
        builder.include(northeast);
        builder.include(southwest);
        bounds = builder.build();
    }

    private Marker plotPoint(LatLng point, boolean user) {
        // create marker
        MarkerOptions marker = new MarkerOptions().position(point);

        if(user) {
            // Changing marker icon
            marker.icon(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_BLUE));
        }
        else marker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED));

        Marker newMarker = googleMap.addMarker(marker);
        return newMarker;
    }

    private Marker plotUserPoint(LatLng point) {
        Marker newMarker = plotPoint(point, true);

```

```java
        if(circle != null) circle.remove();

        plotRadius(point, TotalsData.getAverageDistance());

        return newMarker;
    }

    private void plotRadius(LatLng point, double radius) {
        // Instantiates a new CircleOptions object and defines the center and radius
        CircleOptions circleOptions = new CircleOptions()
                .center(point)
                .radius(radius); // In meters

// Get back the mutable Circle
        circle = googleMap.addCircle(circleOptions);
    }
}
```

```java
package com.example.sdp11.wmd;


import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.List;


public class DataFragment extends Fragment {

    View view;
    ThrowsDataSource dataSource;
    ArrayAdapter adapter;

    public DataFragment() {
        // Required empty public constructor
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.fragment_data, container, false);

        dataSource = new ThrowsDataSource(getActivity());
        dataSource.open();
        dataSource.deleteAllThrows();
        for (int i = 0; i < 25; i++) dataSource.createThrow(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
//        dataSource.createThrow(4, 5, 6, 1, 1, 1, 1, 1);
//        dataSource.createThrow(7, 8, 9, 1, 1, 1, 1, 1);

        ListView lis = (ListView)view.findViewById(R.id.list);

        List<RawThrowData> values = dataSource.getAllThrows();

        adapter = new ArrayAdapter<RawThrowData>(getActivity(),
                android.R.layout.simple_list_item_1, values);
        lis.setAdapter(adapter);
        return view;
```

```
    }

    @Override
    public void onResume() {
        dataSource.open();
        super.onResume();
    }

    @Override
    public void onPause() {
        dataSource.close();
        super.onPause();
    }

}
```

```
package com.example.sdp11.wmd;

/**
 * Created by nsanor on 2/10/2015.
 */
public class GPSDataPoint {
    //Implement using throw id, lat, long, time, etc.
}
```

File - C:\Users\BLinhart\Documents\GitHub\WMD\WMD\app\src\main\java\com\example\sdp11\wmd\IMUDataPoint.java

```
package com.example.sdp11.wmd;

/**
 * Created by nsanor on 2/10/2015.
 */
public class IMUDataPoint {
    //Implement using throw id, and 9 necessary parameters
}

```

```java
package com.example.sdp11.wmd;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import android.app.Activity;
import android.app.ActionBar;
import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.location.Location;
import android.location.LocationListener;
import android.support.v13.app.FragmentPagerAdapter;
import android.os.Bundle;
import android.support.v4.view.ViewPager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;


public class MainActivity extends Activity implements ActionBar.TabListener,GoogleApiClient.
ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener, LocationListener {

    SectionsPagerAdapter mSectionsPagerAdapter;
    ViewPager mViewPager;
    GoogleApiClient mGoogleApiClient;
    static Location mCurrentLocation;
    String mLastUpdateTime;
    Boolean mRequestingLocationUpdates = true;
    LocationRequest mLocationRequest;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

//        if( savedInstanceState != null ){
//            getActionBar().selectTab(savedInstanceState.getInt(tabState));
//        }

        buildGoogleApiClient();
```

```
        createLocationRequest();

        TotalsData.loadTotalsData();

        final ActionBar actionBar = getActionBar();
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

        mSectionsPagerAdapter = new SectionsPagerAdapter(getFragmentManager());

        mViewPager = (ViewPager) findViewById(R.id.pager);
        mViewPager.setAdapter(mSectionsPagerAdapter);

        // When swiping between different sections, select the corresponding
        // tab. We can also use ActionBar.Tab#select() to do this if we have
        // a reference to the Tab.
        mViewPager.setOnPageChangeListener(new ViewPager.SimpleOnPageChangeListener() {
            @Override
            public void onPageSelected(int position) {
                actionBar.setSelectedNavigationItem(position);
            }
        });

        // For each of the sections in the app, add a tab to the action bar.
        for (int i = 0; i < mSectionsPagerAdapter.getCount(); i++) {
            // Create a tab with text corresponding to the page title defined by
            // the adapter. Also specify this Activity object, which implements
            // the TabListener interface, as the callback (listener) for when
            // this tab is selected.
            actionBar.addTab(
                    actionBar.newTab()
                            .setText(mSectionsPagerAdapter.getPageTitle(i))
                            .setTabListener(this));
        }

    }

    @Override
    protected void onStart() {
        super.onStart();
        //Log.e("Connected?", String.valueOf(mGoogleApiClient.isConnected()));
        mGoogleApiClient.connect();
    }

//    public int getSelectedTab() {
//        return getActionBar().getSelectedTab().getPosition();
//    }
//
//
//    private void updateValuesFromBundle(Bundle savedInstanceState) {
```

```java
//        if (savedInstanceState != null) {
//            if (savedInstanceState.keySet().contains("tabState")) {
//                //getActionBar().set(savedInstanceState.getInt("tabState"));
//            }
//
//
//            //updateUI();
//        }
//    }

//    @Override
//    protected void onPause() {
//        super.onPause();
//        stopLocationUpdates();
//    }

    @Override
    public void onResume() {
        super.onResume();
        if (mGoogleApiClient.isConnected() && !mRequestingLocationUpdates) {
            startLocationUpdates();
        }
    }

    protected void stopLocationUpdates() {
        LocationServices.FusedLocationApi.removeLocationUpdates(
                mGoogleApiClient, (com.google.android.gms.location.LocationListener) this);
    }

    public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        //savedInstanceState.putInt("tabState", getSelectedTab());
    }


    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {

    }

    @Override
    public void onLocationChanged(Location location) {
        mCurrentLocation = location;
        mLastUpdateTime = DateFormat.getTimeInstance().format(new Date());
    }

    @Override
    public void onStatusChanged(String s, int i, Bundle bundle) {

```

```java
}

@Override
public void onProviderEnabled(String s) {

}

@Override
public void onProviderDisabled(String s) {

}

public class SectionsPagerAdapter extends FragmentPagerAdapter {

    public SectionsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int position) {
        // getItem is called to instantiate the fragment for the given page.
        Fragment fragment = null;
        if(position==0) fragment = new ConnectFragment();
        if(position==1) fragment = new DataFragment();
        if(position==2) fragment = new MapFragment();
        return fragment;
    }

    @Override
    public int getCount() {
        // Setup and Map tabs
        return 3;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        Locale l = Locale.getDefault();
        switch (position) {
            case 0:
                return getString(R.string.title_section1).toUpperCase(l);
            case 1:
                return getString(R.string.title_section2).toUpperCase(l);
            case 2:
                return getString(R.string.title_section3).toUpperCase(l);
        }
        return null;
    }
}

```

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   // Inflate the menu; this adds items to the action bar if it is present.
   getMenuInflater().inflate(R.menu.menu_main, menu);
   return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
   // Handle action bar item clicks here. The action bar will
   // automatically handle clicks toggle the Home/Up button, so long
   // as you specify a parent activity in AndroidManifest.xml.
   int id = item.getItemId();

   switch (id){
      case R.id.action_settings:
         Toast.makeText(getApplicationContext(), "Settings",
               Toast.LENGTH_SHORT).show();
         return true;
      case R.id.about:
         Toast.makeText(getApplicationContext(), "About Us",
               Toast.LENGTH_SHORT).show();
         return true;
      default:
         return super.onOptionsItemSelected(item);
   }
}

@Override
public void onTabSelected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {
   // When the given tab is selected, switch to the corresponding page in
   // the ViewPager.
   mViewPager.setCurrentItem(tab.getPosition());
}

@Override
public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {
}

@Override
public void onTabReselected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {
}

protected synchronized void buildGoogleApiClient() {
  mGoogleApiClient = new GoogleApiClient.Builder(this)
         .addConnectionCallbacks((GoogleApiClient.ConnectionCallbacks) this)
         .addOnConnectionFailedListener((GoogleApiClient.OnConnectionFailedListener) this)
         .addApi(LocationServices.API)
         .build();
```

```
    }

    @Override
    public void onConnected(Bundle bundle) {
        mCurrentLocation = LocationServices.FusedLocationApi.getLastLocation(
                mGoogleApiClient);
    }

    public static Location getmCurrentLocation() {
        return mCurrentLocation;
    }

    @Override
    public void onConnectionSuspended(int i) {

    }

    protected void createLocationRequest() {
        mLocationRequest = new LocationRequest();
        mLocationRequest.setInterval(10000);
        mLocationRequest.setFastestInterval(5000);
        mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);
    }

    protected void startLocationUpdates() {
        LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, mLocationRequest, (com.
google.android.gms.location.LocationListener) this);
    }
}
```

File - C:\Users\BLinhart\Documents\GitHub\WMD\WMD\app\src\main\java\com\example\sdp11\wmd\RawThrowData.java

```java
package com.example.sdp11.wmd;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;

/**
 * Created by Student on 1/27/2015.
 */
public class RawThrowData {
    private long throwId;
    private long holeId;
    private long gameId;
    private double startLat;
    private double startLong;
    private double endLat;
    private double endLong;
    private double startXAccel;
    private double startYAccel;
    private long startTime;
    private long endTime;
    private long syncTime;

    public RawThrowData() {}

    public RawThrowData(long throwId, long holeId, long gameId, double startLat, double startLong, double endLat, double endLong, double startXAccel, double startYAccel, long startTime, long endTime, long syncTime ) {
        this.throwId = throwId;
        this.holeId = holeId;
        this.gameId = gameId;
        this.startLat = startLat;
        this.startLong = startLong;
        this.endLat = endLat;
        this.endLong = endLong;
        this.startXAccel = startXAccel;
        this.startYAccel = startYAccel;
        this.startTime = startTime;
        this.endTime = endTime;
        this.syncTime = syncTime;
    }


    public long getEndTime() {
        return endTime;
    }

    public void setEndTime(long endTime) {
```

```java
        this.endTime = endTime;
    }

    public long getStartTime() {
        return startTime;
    }

    public void setStartTime(long startTime) {
        this.startTime = startTime;
    }

    public long getThrowId() {
        return throwId;
    }

    public void setThrowId(long throw_id) {
        this.throwId = throw_id;
    }

    public long getHoleId() {
        return holeId;
    }

    public void setHoleId(long hole_id) {
        this.holeId = hole_id;
    }

    public long getGameId() {
        return gameId;
    }

    public void setGameId(long game_id) {
        this.gameId = game_id;
    }

    public double getStartLat() {
        return startLat;
    }

    public void setStartLat(double start_lat) {
        this.startLat = start_lat;
    }

    public double getStartLong() {
        return startLong;
    }

    public void setStartLong(double start_long) {
        this.startLong = start_long;
```

```java
    }

    public double getEndLat() {
        return endLat;
    }

    public void setEndLat(double end_lat) {
        this.endLat = end_lat;
    }

    public double getEndLong() {
        return endLong;
    }

    public void setEndLong(double end_long) {
        this.endLong = end_long;
    }

    public double getStartXAccel() {
        return startXAccel;
    }

    public void setStartXAccel(double start_x_accel) {
        this.startXAccel = start_x_accel;
    }

    public double getStartYAccel() {
        return startYAccel;
    }

    public void setStartYAccel(double start_y_accel) {
        this.startYAccel = start_y_accel;
    }

    public long getSyncTime() {
        return syncTime;
    }

    public void setSyncTime(long syncTime) {
        this.syncTime = syncTime;
    }

    //Convert from epoch to string
    public String convertDate(long d) {
        Date date = new Date(d);
        DateFormat format = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");
        format.setTimeZone(TimeZone.getTimeZone("America/New_York"));
        String formatted = format.format(date);
        return formatted;
```

```
    }

    public String getAllFields() {
        return String.valueOf(syncTime) + ", "  + convertDate(syncTime);//throwId + ", " + holeId + ", " + gameId
 + ", " + startLat + ", " + startLong + ", " + startXAccel + ", " + startYAccel + ", " + endLat + ", " + endLong;
    }

    //Will be used by the ArrayAdapter in the ListView
    @Override
    public String toString() {
        return getAllFields();
    }

}
```

```java
package com.example.sdp11.wmd;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import java.util.ArrayList;

/**
 * Created by nsanor on 2/10/2015.
 */
public class ThrowAdapter extends ArrayAdapter<CalculatedThrowData> {
    private ArrayList<CalculatedThrowData> items;
    private Context context;

    public ThrowAdapter(Context context, int textViewResourceId, ArrayList<CalculatedThrowData> items) {
        super(context, textViewResourceId, items);
        this.items = items;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = convertView;
        if (v == null) {
            LayoutInflater vi = (LayoutInflater)context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            v = vi.inflate(R.layout.row, null);
        }
        CalculatedThrowData ctd = items.get(position);
        if (ctd != null) {
            TextView id = (TextView) v.findViewById(R.id.throw_id);
            TextView dist = (TextView) v.findViewById(R.id.total_distance);
            TextView tintegrity = (TextView) v.findViewById(R.id.throw_integrity);
            if (id != null) {
                id.setText(String.valueOf(ctd.getThrowId()));
            }
            if(dist != null){
                dist.setText(String.valueOf(ctd.getTotalDistance()));
            }
            if(tintegrity != null){
                dist.setText(String.valueOf(ctd.getThrowIntegrity()));
            }
        }
        return v;
    }

}
```

File - C:\Users\BLinhart\Documents\GitHub\WMD\WMD\app\src\main\java\com\example\sdp11\wmd\ConnectFragment.java

```java
package com.example.sdp11.wmd;


import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothManager;
import android.bluetooth.BluetoothProfile;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.app.Fragment;
import android.os.Handler;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;


/**
 * A simple {@link Fragment} subclass.
 */
public class ConnectFragment extends Fragment{

    private Button on, off, search, paired;
    View view;

    private BluetoothAdapter BA;
    private ListView lv;
    private ArrayAdapter listAdapter;
    private List values;

    private boolean mScanning;
    private Handler mHandler;

    // Stops scanning after 10 seconds.
    private static final long SCAN_PERIOD = 10000;

    public ConnectFragment() {
```

```java
        // Required empty public constructor
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.fragment_connect, container, false);
        final BluetoothManager bluetoothManager =
                (BluetoothManager) getActivity().getSystemService(Context.BLUETOOTH_SERVICE);
        BA = bluetoothManager.getAdapter();

        on = (Button)view.findViewById(R.id.Toggle);
        on.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                toggle(view);
            }
        });

        paired = (Button)view.findViewById(R.id.Paired);
        paired.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(BA.getState() == BluetoothAdapter.STATE_ON) scanLeDevice(true);
            }
        });

        lv = (ListView)view.findViewById(R.id.devices);

        //BA.startLeScan(mScanCallback);

        listAdapter = new ArrayAdapter<String>(getActivity(),android.R.layout.simple_list_item_1, 0);
        lv.setAdapter(listAdapter);

        return view;
    }

    private void scanLeDevice(final boolean enable) {
        if (enable) {
            // Stops scanning after a pre-defined scan period.
            mHandler = new Handler();
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    mScanning = false;
                    BA.stopLeScan(mScanCallback);
                    Log.e("", "Stop Scanning");
                }
```

```java
            }, SCAN_PERIOD);

            Log.e("", "Now Scanning");
            mScanning = true;
            BA.startLeScan(mScanCallback);
        } else {
            mScanning = false;
            BA.stopLeScan(mScanCallback);
        }

    }

    private BluetoothAdapter.LeScanCallback mScanCallback = new BluetoothAdapter.LeScanCallback() {
        @Override
        public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord) {
            getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    listAdapter.add(device.getName());
                    listAdapter.notifyDataSetChanged();
                    Log.e("", device.getName());
                }
            });
        }
    };

//    private BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
//        @Override
//        public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
//            //Connection established
//            if (status == BluetoothGatt.GATT_SUCCESS
//                    && newState == BluetoothProfile.STATE_CONNECTED) {
//
//                //Discover services
//                gatt.discoverServices();
//
//            } else if (status == BluetoothGatt.GATT_SUCCESS
//                    && newState == BluetoothProfile.STATE_DISCONNECTED) {
//
//                //Handle a disconnect event
//
//            }
//        }
//
//        @Override
//        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
//            if (status == BluetoothGatt.GATT_SUCCESS) {
//
//                Log.i("ConnectFragment", "Connected to: " + gatt.getDevice());
```

```
//          }
//      }
//   };



    public void toggle(View view){
        if(!BA.isEnabled()){
            //Intent TurnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            //startActivityForResult(TurnOn, 0);
            BA.enable();
            while (BA.getState() != BluetoothAdapter.STATE_ON);
            Toast.makeText(getActivity(), "Bluetooth is now on!", Toast.LENGTH_SHORT).show();
        }
        else {
            //Intent TurnOn = new Intent(BluetoothAdapter.ACTION_);
            //startActivityForResult(TurnOn, 0);
            BA.disable();
            while (BA.getState() != BluetoothAdapter.STATE_OFF);
            Toast.makeText(getActivity(), "Bluetooth is now off!", Toast.LENGTH_SHORT).show();
        }
    }
}
```

```java
package com.example.sdp11.wmd;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by Student on 1/27/2015.
 */
public class ThrowsDataSource {
    private SQLiteDatabase database;
    private DBHelper dbHelper;
    private String[] allColumns = { DBHelper.COLUMN_THROW_ID,
            DBHelper.COLUMN_HOLE_ID,
            DBHelper.COLUMN_GAME_ID,
            DBHelper.COLUMN_START_LAT,
            DBHelper.COLUMN_START_LONG,
            DBHelper.COLUMN_END_LAT,
            DBHelper.COLUMN_END_LONG,
            DBHelper.COLUMN_START_ACCEL_X,
            DBHelper.COLUMN_START_ACCEL_Y,
            DBHelper.COLUMN_START_TIME,
            DBHelper.COLUMN_END_TIME,
            DBHelper.COLUMN_SYNC_TIME};

    private String[] mainColumns = { DBHelper.COLUMN_THROW_ID,
            DBHelper.COLUMN_INITIAL_DIRECTION,
            DBHelper.COLUMN_FINAL_DIRECTION,
            DBHelper.COLUMN_TOTAL_DISTANCE,
            DBHelper.COLUMN_THROW_INTEGRITY,
            DBHelper.COLUMN_TOTAL_TIME};

    public ThrowsDataSource(Context context) {
        dbHelper = new DBHelper(context);
    }

    public void open() throws SQLException {
        database = dbHelper.getWritableDatabase();
    }

    public void close() {
        dbHelper.close();
    }
```

```java

    public void createThrow(long hole_id, long game_id, double start_lat, double start_long, double end_lat, double end_long, double start_x_accel, double start_y_accel, long startTime, long endTime) {

        ContentValues values = new ContentValues();
        values.put(DBHelper.COLUMN_HOLE_ID, 1);
        values.put(DBHelper.COLUMN_GAME_ID, 1);
        values.put(DBHelper.COLUMN_START_LAT, start_lat);
        values.put(DBHelper.COLUMN_START_LONG, start_long);
        values.put(DBHelper.COLUMN_END_LAT, end_lat);
        values.put(DBHelper.COLUMN_END_LONG, end_long);
        values.put(DBHelper.COLUMN_START_ACCEL_X, start_x_accel);
        values.put(DBHelper.COLUMN_START_ACCEL_Y, start_y_accel);
        values.put(DBHelper.COLUMN_START_TIME, startTime);
        values.put(DBHelper.COLUMN_END_TIME, endTime);

        //Calculate unix time from current time
        long now = System.currentTimeMillis();
        values.put(DBHelper.COLUMN_SYNC_TIME, now);

        long insertId = database.insert(DBHelper.TABLE_THROWS, null,values);

        //Need to get new max throw id to create calc data
//        Log.e("TEST", database.toString());
//        Log.e("ID Test", String.valueOf(getMaxThrowId(database)));
        //Create new entry in CalculatedThrowData here.
        //CalculatedThrowData calc = new CalculatedThrowData(throwId, start_lat, double start_long, double end_lat, double end_long, double start_x_accel, double start_y_accel, long startTime, long endTime);

        //Update globals in TotalsData here.

//        Cursor cursor = database.query(DBHelper.TABLE_THROWS,
//                allColumns, DBHelper.COLUMN_ID + " = " + insertId, null,
//                null, null, null);
//        cursor.moveToFirst();
//        Throw newThrow = cursorToRawThrow(cursor);
//        cursor.close();
//        return newThrow;

    }



    public void deleteThrow(RawThrowData t) {
        long id = t.getThrowId();
        System.out.println("Comment deleted with id: " + id);
        database.delete(DBHelper.TABLE_THROWS, DBHelper.COLUMN_THROW_ID
                + " = " + id, null);
    }
```

```java

    public void deleteAllThrows() {
        //database.rawQuery("delete from sqlite_sequence where name = 'throws'", null);
        database.delete(DBHelper.TABLE_THROWS, null,null);
    }

    public List<RawThrowData> getAllThrows() {
        List<RawThrowData> ts = new ArrayList<RawThrowData>();

        Cursor cursor = database.query(DBHelper.TABLE_THROWS,
                allColumns, null, null, null, null, null);

        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            RawThrowData t = cursorToRawThrow(cursor);
            ts.add(t);
            cursor.moveToNext();
        }
        // make sure to close the cursor
        cursor.close();
        return ts;
    }

    public List<CalculatedThrowData> getAllThrowsShort() {
        List<CalculatedThrowData> ts = new ArrayList<CalculatedThrowData>();

        Cursor cursor = database.query(DBHelper.TABLE_CALC,
                mainColumns, null, null, null, null, null);

        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            CalculatedThrowData t = cursorToThrow(cursor);
            ts.add(t);
            cursor.moveToNext();
        }
        // make sure to close the cursor
        cursor.close();
        return ts;
    }

    private RawThrowData cursorToRawThrow(Cursor cursor) {
        RawThrowData t = new RawThrowData();
        t.setThrowId(cursor.getLong(0));
        t.setHoleId(cursor.getLong(1));
        t.setGameId(cursor.getLong(2));
        t.setStartLat(cursor.getDouble(3));
        t.setStartLong(cursor.getDouble(4));
        t.setEndLat(cursor.getDouble(5));
        t.setEndLong(cursor.getDouble(6));
```

```
        t.setStartXAccel(cursor.getDouble(7));
        t.setStartYAccel(cursor.getDouble(8));
        t.setStartTime(cursor.getLong(9));
        t.setEndTime(cursor.getLong(10));
        t.setSyncTime(cursor.getLong(11));
        return t;
    }

    private CalculatedThrowData cursorToThrow(Cursor cursor) {
        CalculatedThrowData t = new CalculatedThrowData();
        t.setThrowId(cursor.getLong(0));
        t.setInitialDirection(cursor.getDouble(1));
        t.setFinalDirection(cursor.getDouble(2));
        t.setThrowIntegrity(cursor.getDouble(3));
        t.setTotalDistance(cursor.getDouble(4));
        t.setTotalTime(cursor.getInt(5));
        return t;
    }

    public long getMaxThrowId(SQLiteDatabase db) {
        Cursor c = db.rawQuery("SELECT MAX(?) FROM " + DBHelper.TABLE_THROWS, new String[] {"throw_id"
});
        c.moveToFirst();
        //int index = c.getColumnIndex("throw_id");
       // Log.e("TEST", String.valueOf(index));
        return c.getInt(0);
    }
}
```

```java
package com.example.sdp11.wmd;

/**
 * Created by Student on 2/5/2015.
 */
public class CalculatedThrowData {
    private long throwId;
    private double initialDirection;
    private double finalDirection;
    private double throwIntegrity;
    private double totalDistance;
    private double totalTime;

    public CalculatedThrowData() {

    }

    public CalculatedThrowData(long throwId, double start_lat, double start_long, double end_lat, double end_long, double start_x_accel, double start_y_accel, long startTime, long endTime) {
        this.throwId =throwId;
        this.totalDistance = calculateDistance(start_lat, start_long, end_lat, end_long);
        this.totalTime = endTime - startTime;
        this.throwIntegrity = 1;
    }

    public CalculatedThrowData(RawThrowData t) {
        this.throwId = t.getThrowId();
        this.totalDistance = calculateDistance(t.getStartLat(), t.getStartLong(), t.getEndLat(), t.getEndLong());
        this.totalTime = t.getEndTime() - t.getStartTime();
        this.throwIntegrity = 1;
    }

    public long getThrowId() {
        return throwId;
    }

    public void setThrowId(long throwId) {
        this.throwId = throwId;
    }

    public double getInitialDirection() {
        return initialDirection;
    }

    public void setInitialDirection(double initialDirection) {
        this.initialDirection = initialDirection;
    }

    public double getFinalDirection() {
```

```
        return finalDirection;
    }

    public void setFinalDirection(double finalDirection) {
        this.finalDirection = finalDirection;
    }

    public double getTotalDistance() {
        return totalDistance;
    }

    public void setTotalDistance(double totalDistance) {
        this.totalDistance = totalDistance;
    }

    public double getTotalTime() {
        return totalTime;
    }

    public void setTotalTime(double totalTime) {
        this.totalTime = totalTime;
    }

    public double getThrowIntegrity() {
        return throwIntegrity;
    }

    public void setThrowIntegrity(double throwIntegrity) {
        this.throwIntegrity = throwIntegrity;
    }

    public String getMainFields() {
        return throwId + " | " + totalDistance + " | " + throwIntegrity;
    }

    double degreesToRadians(double degrees) {
        return degrees*(Math.PI/180);
    }

    private double calculateDistance(double lat1, double lng1, double lat2, double lng2) {
        double r = 3963.191;
        double lat1rad = degreesToRadians(lat1);
        double lat2rad = degreesToRadians(lat2);
        double long1rad = degreesToRadians(lng1);
        double long2rad = degreesToRadians(lng2);
        double diffLong = long2rad - long1rad;
        double e = Math.acos(Math.sin(lat1rad)*Math.sin(lat2rad) + Math.cos(lat1rad)*Math.cos(lat2rad)*Math.cos(diffLong));
        return r * e;
```

```
    }

    //Will be used by the ArrayAdapter in the ListView
    @Override
    public String toString() {
        return getMainFields();
    }
}
```