Spring 2021

# Judo Wearable

Alison Waugh
*The University of Akron*, arw116@uakron.edu

Natalie Masters
*The University of Akron*, nbw9@uakron.edu


# JUDO WEARABLE

By

Natalie Masters

Alison Waugh

Final Report for 4600:471 Senior Design, Spring 2021

Final Report for 4600:497 Honor Design, Spring 2021

Faculty Advisor: Dr. Scott Sawyer

3 May 2021

Project No. 13

## Abstract

This report details the research, design, testing, and fabrication of a judo wearable that is meant to improve one-on-one training in preparation for competitions. The conceptual design phase included critical research and investigation into the feasibility of different clothing bases and microelectronic components to ensure a durable and flexible design. The embodiment design phase elaborates on the advantages of the selected components as well as modifications made due to resource challenges. In the detail design face, the schematics and code were fully fleshed out, leading directly into verification and testing of the design.

# Contents

# 1 Introduction

## 1.1 Background

The sport of judo was founded by Kano Jigoro Shihan, and is meant to strengthen the body by practicing attack and defense. The principles of judo extend beyond physical training, and is meant to train one's mind as well as inspire one to give back to society. While there are different ways to test one's skills, the best way is through competition. In competition, there are multiple ways to win a match. One of those ways is by the tori (thrower) performing a perfect throw (ippon throw) on the uki (person being thrown) which is defined as both shoulders hitting the mat simultaneously and instantaneously ends the match. Another way is by earning enough points by effectively performing a throw (wazari throw), and other such techniques. This project will modify an existing rash guard with a microcontroller and sensors to collect and transfer data to determine whether a throw is ippon, wazari, or ineffective/failed.

## 1.2 Principles of Operation

Traditionally, the quality of a judo throw is determined by a third person such as a referee or coach to judge the position of the uki's shoulders in relation to the mats. The purpose of this project is to eliminate the need for this outside perspective, so four compass sensors will be connected to and powered by an Arduino Nano 33 BLE microcontroller. The Arduino will be positioned at a no-contact point on the rash guard to ensure it is not damaged. This set up collects and transfers acceleration and position data via Bluetooth to a phone app. The data is then processed by the app to determine if the throw was ippon or not. This equipment will aid in learning throws, training for competitions, and understanding the dynamics experienced by the body during a throw.

## 1.3 Expanded Design Brief

There is a need for a modified rash guard outfitted with at least four 6-DOF sensors that can qualify the effectiveness of judo throws. These four sensors must be placed on the front of the rash guard to ensure they are not damaged, destroyed, or disconnected during a throw. Due to the nature of the sport, the requirements that qualify a perfect throw in judo are specific and precise. As such, an ippon throw will be defined as when the velocity of each shoulder reaches 0 m/s within 0.5 seconds of each other. Consequently, a wazari throw will be defined as when the velocity of each shoulder reaches 0 m/s with a delay greater than 0.5 seconds. An ineffective throw will be defined by incorrect orientation of the body or a different part of the body coming to a rest first. All of these criteria are subject to tuning as validated testing demonstrates.

## 1.4 Relevant Standards

Future iterations of the design will need to properly adhere to ASME (American Society of Mechanical Engineers) standards. In particular "Data Acquisition Standard PTC 19.22 - 2007(R2017)" would be applicable. The scope of this standard includes multiplexing and data processing with data acquisition systems that utilize multiple sensors and link to networked computers. The Judo Wearable falls within this category.

# 2 Design

The wearable sensor array is meant to be used during judo practice. Therefore, the design had to be low profile so that components and connections would not interfere with the throws. Judo throws and falls require a high range of motion, necessitating the design to also be very flexible. Another aspect of a low profile design is that it would allow the wearable to be used with or without a judo gi. Since the wearable has to be on a highly active user, it had to be battery powered and able to transfer data to an external recipient such as  a phone.

## 2.1 Conceptual Design

### 2.1.1 Morphological Chart

The team generated several original concepts and methods to meet the required objectives of the design. The main functions of the design are the clothing base, microcontroller system, the type of sensor, connection methods, and means of calculating the results. The team's solutions are seen and compared in the morphological chart in Table 1, as well as the chosen design to progress with.

Table 1 – Morphological Chart

| Garment base | Vest | T-shirt | Rash guard |
|---|---|---|---|
| Microcontroller | Raspberry Pi | Arduino | |
| Sensor | Gyroscopes and accelerometers | 6-Degree of freedom sensors | 9-Degree of freedom sensor |
| Connect microcontroller and sensors | Soldered | Alligator clips | |
| Calculate results | On-board | External | |

The original design the team decided on above evolved from using 9-DOF sensors to 6-DOF sensors as new information was discovered on availability of parts. The 9-DOF sensors that are compatible with the FLORA are a discontinued product, and could not be sourced. Thus, the design was modified to use 6-DOF sensors instead.

### 2.1.2 Objective Tree

In order to create a method to decide on a final design, the team created an objective tree. This broke down the final intended outcome into weighted goals that would be used to evaluate the generated design concepts. The assigned values were agreed upon by the team, and are shown below in Figure 1.

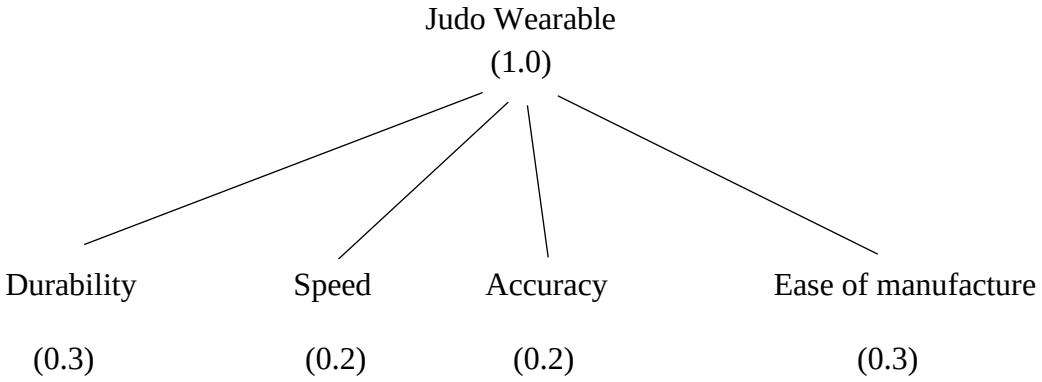



Figure 1 – Objective Tree

### 2.1.3 Weighted Decision Matrix

The weighted decision matrix below in Table 2 compares the different microcontroller options while being rated against the goals established in the objective tree. The values were discussed by the team and agreed upon. The weighted decision matrix established that the best microcontroller available for this application is the Arduino Nano.

Table 2 – Weighted Decision Matrix

| | | Raspberry Pi | | Arduino Nano | | Adafruit Flora | |
|---|---|---|---|---|---|---|---|
| Durability | 0.3 | 2 | 0.6 | 4 | 0.9 | 4 | 1.2 |
| Speed | 0.2 | 3 | 0.6 | 3 | 0.6 | 3 | 0.6 |
| Accuracy | 0.2 | 4 | 0.8 | 4 | 0.8 | 3 | 0.8 |
| Ease of Manufacture | 0.3 | 2 | 0.6 | 4 | 0.9 | 4 | 1.5 |
| | | | 2.6 | | 3.8 | | 3.6 |

## 2.2 Embodiment Design

With the best clothing base, microcontroller system, sensor type, and connection method identified, the team began conducting research on sourcing these components. The team realized that the goal not addressed in the objective tree was low heat production, which is critical for devices that are flush against the body. This led to a focus on components that are specifically designed for wearing on the body, and are therefore robust enough for this project.

### 2.2.1 Sensor Selection

Initial research on sensors brought the team to a company called Adafruit Industries. They had many types of 9-DOF sensors that included an accelerometer, magnetometer, and gyroscope. This was ideal for the precision required, as well as having a clear reference orientation based upon the earth. However, as mentioned previously, the original design had to be modified to use 6-DOF sensors instead of 9-DOF sensors due to sourcing issues. The FLORA Accelerometer/Compass sensor was deemed an acceptable substitution as it still possessed the necessary orientation. As seen in Figure 2, this sensor is also incredibly small which means that it is easier to choose locations on the body to reduce the risk of it sustaining damage during operation.

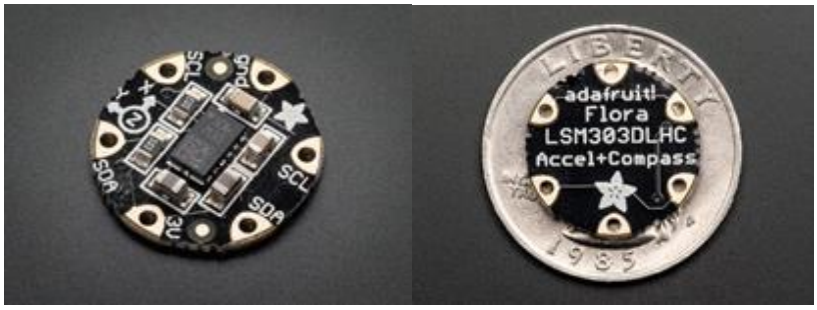



Figure 2 – Adafruit FLORA Accelerometer/Compass Sensor. Retrieved from https://www.adafruit.com/product/1247

### 2.2.2 Clothing Base

When determining an appropriate choice for the clothing base for the microcontroller and sensors, there were two significant contenders: a martial arts rash guard or a judo gi. As seen in Figure 3 below, the rash guard is a light-weight, form fitting garment whereas the judo gi is made from a thicker material and its movements do not follow the body as closely.

Figure 3 – Rash guard (left) and judo gi (right)

While the goal for the final design is to use the rash guard, for the first iteration of the design the team decided to repurpose a long-sleeve cotton shirt. The decision was made with the delicate nature of the rash guard in mind. The skin-tight, elastic properties of the rash guard exist because it is made out of spandex and polyester. The downside of these materials is that they can be subject to extensive tearing or fraying. As the design progressed to the prototyping stage, it became clear that it would be beneficial to create the first iteration using a cotton shirt.

## 2.3 Detail Design

### 2.3.1 Circuit Design

A common problem when using multiples of the same sensor is that they have a fixed I2C address. In order to work around this problem, a multiplexer must be added to the circuit. Multiplexers come in various sizes, and for this project's purposes a 1x8 was used. This capacity was chosen as the minimum number of sensors to be used was four, but additional sensors could be used if desired.

From Adafruit's website, a basic schematic was referenced to create the circuit design seen in Appendix A. This schematic only shows the multiplexer being connected to one compass sensor, but the connections are identical for additional sensors. The only change would be the number associated with the respective SDA or SCL pin.

### 2.3.2 Software Design

The system had to gather data from its sensor array, transfer it to a connected phone, analyze the data, and output the result. Two software systems were used to accomplish these goals; Arduino IDE for the microcontroller and MIT App Inventor for the phone. Since two softwares were being used on two devices, the goals had to be divided. The Arduino on the wearable had to be able to gather sensor data, catalogue it, and send it to the app via Bluetooth. The app had to receive the data, record it to a local file, analyze for an ippon throw, and output a result. Analysis was tasked out to the app in order to reduce processor load on the Arduino. This would reduce power draw, increasing battery life and reducing the chance of the board overheating.

### 2.3.3 Arduino IDE

The microcontroller program needed to connect with the sensors then write their data to a form accessible with Bluetooth. To connect to the sensors, the Arduino needed to have access to a variety of C++ libraries specific to the Adafruit sensors and for the onboard 9 degree-of-freedom inertial measurement unit. It also needed the Wire library to connect to I2C devices and the ArduinoBLE library to use the onboard Bluetooth.

All of the sensors needed to be assigned a unique identifier. As there were two sensors, an accelerometer and a magnetometer, for each LSM303 breakout, there were eight different sensor items. Then the multiplexer port each breakout was physically connected to was assigned to a

variable. This was so the port could be changed without going through the entire program. It was noted in the comments where each sensor was located on the body.

The BLE service with an identifying number was created with float characteristics for each variable. That was an x-, y-, and z-component for each accelerometer, magnetometer, and gyroscope. These would be overwritten with each sensor reading and were set to notify the connected device of the change. Then they could be read by the connected device. For this application, the Arduino was the peripheral Bluetooth device and the phone was the central Bluetooth device.

There were two functions to display sensor details for the LSM303 accelerometers and magnetometers. These were necessary for debugging with the serial monitor. They outputted the sensor's name, driver, unique identifier, maximum and minimum values, and resolution. The Arduino's IMU did not require a separate function as it was unique and the data could be requested directly from the IMU. A third function was defined to select the multiplexer port and switch between reading sensors. The multiplexer's own address was identified, then it utilized the Wire library's functions to read the correct port.

The setup code ran once on startup. It would initialize the sensors, start the serial monitor, activate the Bluetooth, and set outputs. The serial monitor was started on 115200 baud and the Wire library was initialized to start I2C connection. The board's built-in LED was set as an output so it could offer a status indicator when the board was not connected to a laptop. The BLE was started and the local name was set to "JudoWearable," to make it easily identifiable. Then the service previously created was set to be advertised so a central device could read it. All of the characteristics for the data variables were added to this service and this service was added to the BLE, which was also set to advertise. Next the LSM303 breakouts had to be initialized. The multiplexer port selection function was utilized to change between ports so the magnetometer and accelerometer on each board could be set up. The magnetometers had autoRange enabled and used the function to display sensor details in the serial monitor if it was available. The accelerometer range was set to two g's and also had their sensor details displayed. After all the LSM303 sensors were initialized, the Arduino's onboard IMU had to be initialized. This could be done as one operation for the whole IMU rather than three separate operations for the accelerometer, magnetometer, and gyroscope. The sample rates for each of the three sensors in the IMU were outputted to the serial monitor if it was available. The final step of setup was to create the object for the BLE central device, aptly named "central."

Once setup was complete, the program could move on to the loop portion of the code which would run repeatedly. This is where data gathering and transmission occurred. The BLE attempted to connect to a central device. If it could, it would set the LED to high and display the central device's IP address in the serial monitor. Then a new sensor event was created to gather the data from the LSM303 breakouts' accelerometers and magnetometers. The multiplexer port was selected and the magnetometer event was retrieved. The x-, y-, and z-components were read

to unique variables named based upon the direction, sensor type, and which sensor, for later transmission. The accelerometer event data was retrieved and stored much the same way. This process was repeated for the remainder of the LSM303 breakout boards. The onboard IMU was read next and its data was also stored in uniquely identified variables. If the serial monitor was available, all the data was displayed. Then, if the Bluetooth was connected to a central device, the BLE characteristics were overwritten with the updated values of their corresponding variables. If the BLE was not connected to a central device, it would output the IP address of the last connected device to the serial monitor and change the built-in LED to low. There was a delay of 500 milliseconds between each iteration of the loop and therefore between each instance of data.

### 2.3.4 MIT App Inventor

A phone app had to be developed to read the Bluetooth data from the Arduino and sensors, then analyze that data, and output the result. This app was created using the MIT App Inventor, a free online software that uses a block-style coding system. The app design was separated into the front end user interface and the back end programming.

The front end included visible and non-visible components. The visible components were a "connect" button, a "record" switch, and a "status" label. The non-visible components were a Bluetooth Low Energy connection, a notifier, a local file to store data in, and a clock. Each of these components enabled certain programmable blocks on the back end. There were also various aesthetic changes that were made to the user interface to make it easier to read.

The back end was composed of blocks that connected the user interface to routines and outputs. All of the variables that contained the sensor data had to be initialized to zero. For clarity, these were given the same names as those in the Arduino program.

When clicked, the "connect" button caused the app side BLE to start scanning. It would connect with a peripheral device with the service identifier and name designated in the Arduino program. This would cause the front end "status" label to read "connecting." Once a connection had been established with the peripheral device, the BLE stopped scanning and the "status" label was changed to "connected." Then the app called the BLE to register to receive float-type data from a device with the service identifier given previously and the characteristic identifiers for each characteristic, one for each variable, called out in the Arduino program. This enables the app to receive data from each of those characteristics and read float data from them.

The "record" switch was programmed such that when it is changed, the app will check its status. If it was switched to the "on" position, then it called the non-visible local file. The program initialized the file with a title and a text line and designated the file name and save location. When the file had been saved, the file name was saved to a variable and a Boolean variable "isSaving" was set to true to indicate that the program was to record data.

Upon receiving float data via Bluetooth, the program queries the characteristic identifier. Each characteristic coincides to a counterpart in the Arduino program. If the identifier of the floats received matches the identifier of a particular variable, that variable is overwritten with the new value. Then the data was saved to a local file if the "record" switch had been turned on.

Next the program would have to evaluate the data to determine if a throw had occurred and if it had been ippon. Unfortunately, the app was incomplete at the time of this report due to issues with the prototype.

### 2.3.5 Constraints

The design had several constraints applied for each of its subsystems. These had to be balanced to create an effective final design. The sensors, microcontroller, and circuitry fulfill the data acquisition needs yet be durable enough to safely withstand the punishment of judo. The software had to gather and analyze data fast enough to record a throw without drawing too much power or overheating the microcontroller. The clothing base had to maintain the position of the sensors relative to the body and supply a sturdy foundation for the rest of the components while not interfering with the throw. All components had to be cost-effective and safe. The team had to consider all of these factors in the design.

# 3 Design Verification

There were several steps to verifying the design of the Judo Wearable. This included the initial breadboard prototype, verification of the code with this prototype, and then transferring the breadboard schematic to its permanent location on a PC board.

## 3.1 Breadboard Prototype

The first stage of prototyping began with assembling all components on the breadboard, and then verifying and uploading the code. During this process, the components were connected using nonpermanent connections. This allowed free movement of the compass sensor, allowing confirmation that the code was able to output the position and acceleration data required. At this stage the multiplexer had yet to have headers soldered to the board. This was important as it allowed the verification that all components were fully functional.

## 3.2 Fabrication

For the initial design fabrication, the decision was made to begin with a cotton-based shirt instead of the rash guard. Cotton is a more forgiving material to work with, and due to the team's lack of experience with sewing this was ideal.

To transfer the circuit from the breadboard to the cotton shirt, the first steps were to solder the headers to the multiplexer, the IC sockets to the Adafruit Flex Perma board, and then seat the Arduino and the multiplexer in the IC sockets. From there the wire leads were soldered to the Perma board for positive, ground, SDA, and SCL for both the Arduino and multiplexer. Wire leads were also soldered for pins SC1, SD1, SC2, SD2, SC4, SD4, SC6, and SD6. Each SC- and SD- couple connects to one of the four compass sensors.

The next step was to secure the compass sensors as well as the Perma board to the cotton shirt using normal thread. Then the conductive thread was weaved through the shirt from each sensor to its respective SC- and SD- wire leads. To securely connect the conductive thread to the wire leads, the thread was knotted and then soldered to the wire. To prevent any shorts, this connection was then wrapped in electrical tape so the wire and most of the conductive thread was no longer exposed. See Figure 4 below of initial design iteration.

Figure 4 – First Design Iteration

### 3.3 Iterative Programming

The program had to be adapted to the assembly stage as the additional sensors were incorporated. This served as a method to troubleshoot and resolve issues with both the hardware and the software. It also allowed the team to optimize for the board and the situation. Shifting the analysis of the throw from the Arduino board to the phone app reduced the processor load on the Arduino board. This lessened the power requirements, which extended battery life. It also made it less likely that the board would overheat; a concern with it in such close contact with a person.

# 4 Costs

The budget for this project was limited to $500. Despite some unnecessary expenses in components that were not utilized, the total project cost was $357.75.

## 4.1 Parts

In Table 3 below is the cost breakdown. It is divided into three categories: microelectronics, tools, and connections.

Table 3 – Cost Breakdown

| **Microelectronics** | | **Tools** | | **Connections** | |
|---|---|---|---|---|---|
| Item | Cost | Item | Cost | Item | Cost |
| Arudino Nano 33 BLE Board w/Headers x2 | $46 | Hakko Tip Cleaner | $9.99 | Alligator Clip Test Leads x12 | $3.95 |
| Adafruit FLORA x2 | $29.90 | Weller Helping Hands | $19.99 | Stainless Steel Thin Conductive Thread - 2 ply | $6.95 |
| Adafruit FLORA Accelerometer/Compass Sensor x6 | $89.70 | Hakko Soldering Iron | $32.99 | Adafruit Flex Perma Board x2 | $15.98 |
| USB Battery Pack | $14.95 | Weller Soldering Iron | $19.97 | Solder | $4.99 |
| Adafruit Multiplexer | $6.95 | Multimeter | $44.97 | IC Socket 16 pin | $1.99 |
| Micro-USB Cable | $6.99 | Needles | $0.99 | IC Socket 14 pin | $0.99 |
| Total: | $194 | Total: | $128.90 | Total: | $34.85 |

## 4.2 Labor

Labor is divided into three categories: research and design, coding and debugging, and assembly. Research and design was the most time consuming portion at 140 hours, and this was due to the large knowledge gap to be filled. This time included researching components, materials, coding software and applications, as well as designing the circuit schematic and sensor layout. Coding and debugging totaled 130 hours. This accounts for writing and proofing the code, establishing the sensor criteria, and integrating system checks to identify component issues or failures. Assembling the Judo Wearable took about 80 hours, and was a very intensive process due to the delicate nature of the microelectronic components.

# 5 Conclusion

The team successfully designed, tested, and fabricated the Judo Wearable sensor array. They researched a variety of different options to complete the project. A design was created to fulfill the need to evaluate judo throws for practice purposes. Then, the necessary components were procured. The prototype was created in stages and tested throughout. Programming and debugging were occurring simultaneously with prototype assembly. Then a final prototype was manufactured. However, there were issues with the final prototype. Further troubleshooting led to the team discovering that the multiplexer had been burnt out, likely when the headers were soldered on.

## 5.1 Lessons Learned

Throughout the process, the team learned several valuable lessons. While attempting to complete this project during the COVID-19 pandemic, there were many shipping delays, which caused significant delays in progress. In future work, the team will account for shipping delays and attempt to source components as locally as possible. The team also found the value of open-source software and hardware when working with Arduino. The resources available for these technologies were extremely valuable throughout this project. However, despite the plethora of resources, the team purchased the unnecessary components, proving the importance of extensive research. The team also improved leadership skills by practicing constant communication, a key to an effective group. The team also found that utilizing the strengths of the group members led to a more productivity and efficiency. In addition, they found ways to strengthen the team's weak points by learning and improving other skills, like sewing, soldering, and creating PCB schematics.

## 5.2 Future work

In order to continue this project, the team will need to purchase a new multiplexer. Instead of soldering on the headers themselves, they will get assistance from a more experienced individual. Then, the team can continue with proofing the programs and tuning the Judo Wearable.

## References


[1] Ada, L. (2015, September 11). Adafruit tca9548a 1-TO-8 I2C Multiplexer Breakout. Retrieved May 03, 2021, from https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/wiring-and-test

[2] Arduino nano 33 BLE with headers. (2021). Retrieved May 03, 2021, from https://store.arduino.cc/usa/nano-33-ble-with-headers

[3] ArduinoBLE. (2019, December 25). Retrieved May 03, 2021, from https://www.arduino.cc/en/Reference/ArduinoBLE

[4] ASME. (2008). Data acquisition systems. Retrieved May 03, 2021, from https://www.asme.org/codes-standards/find-codes-standards/ptc-19-22-data-acquisition-systems?productKey=C0540Q%3AC0540Q

[5] Brittain, C. (2020, June 14). Getting started with Bluetooth LE on the Arduino Nano 33 Sense. Retrieved May 03, 2021, from https://ladvien.com/arduino-nano-33-bluetooth-low-energy-setup/

[6] Earl, B. (2013, March 10). LSM303 accelerometer + COMPASS BREAKOUT. Retrieved May 03, 2021, from https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/coding

[7] Jithin, K. (2021, April 20). Arduino BLE ACCELEROMETER tutorial for beginners. Retrieved May 03, 2021, from https://rootsaid.com/arduino-ble-accelerometer-tutorial/

[8] Jithin, K. (2021, March 07). Arduino BLE example Explained step by step. Retrieved May 03, 2021, from https://rootsaid.com/arduino-ble-example/

[9] Mit app inventor + internet of things. (2018). Retrieved May 03, 2021, from http://iot.appinventor.mit.edu/#/bluetoothle/bluetoothleintro

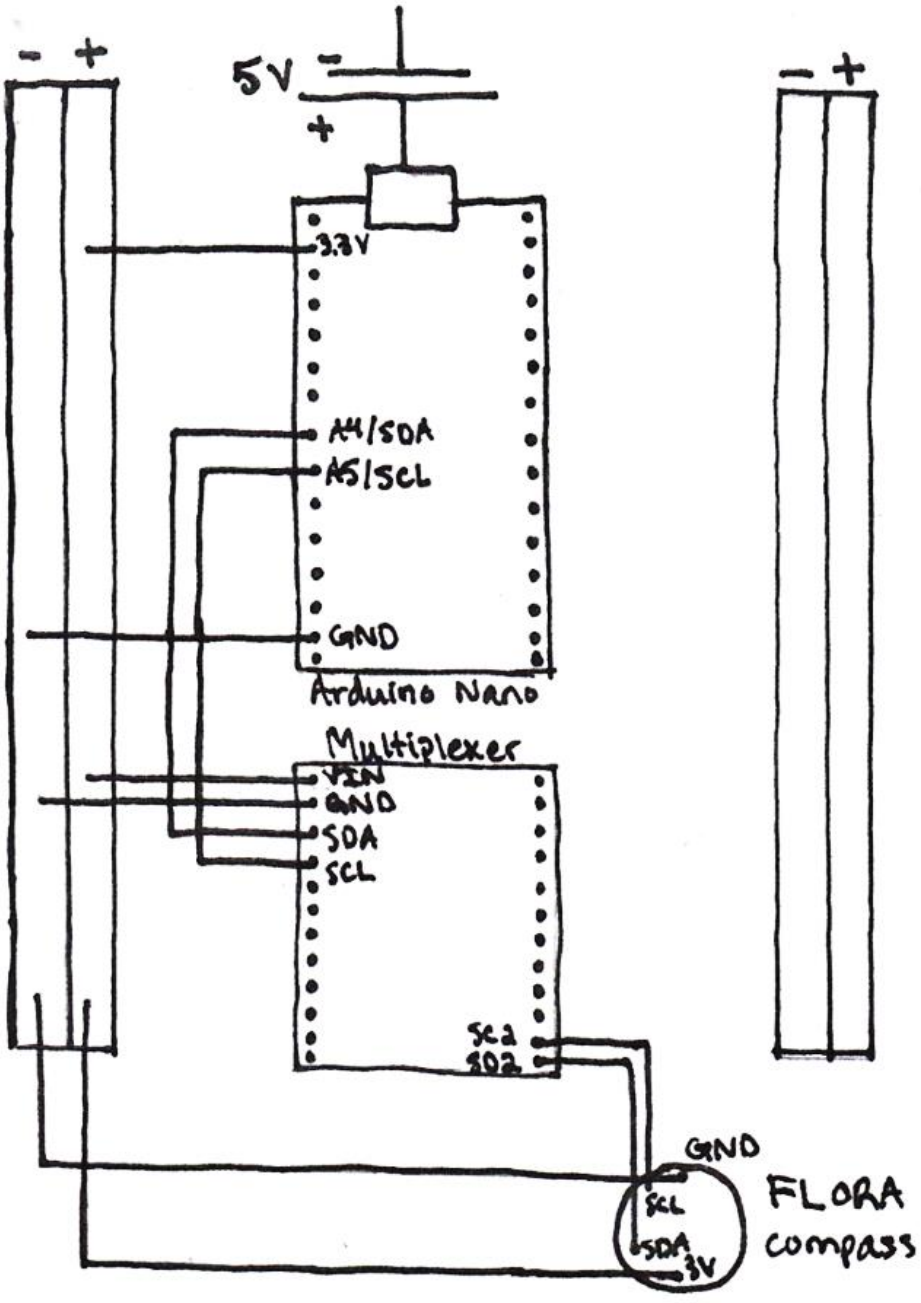
[10] Stern, B. (2013, February 5). Flora accelerometer. Retrieved May 03, 2021, from https://learn.adafruit.com/flora-accelerometer/wiring-with-conductive-thread

[11] Svec, C. (2017, September 06). Ble: Master central slave peripheral client server, we didn't start the fire... Retrieved May 03, 2021, from https://embedded.fm/blog/ble-roles

[12] Tarantula3, & Instructables. (2018, November 10). TCA9548A I2C Multiplexer module - with Arduino And NodeMCU. Retrieved May 03, 2021, from https://www.instructables.com/TCA9548A-I2C-Multiplexer-Module-With-Arduino-and-N/

[13] Townsend, K. (2013, February 28). Using the ADAFRUIT Unified SENSOR DRIVER. Retrieved May 03, 2021, from https://learn.adafruit.com/using-the-adafruit-unified-sensor-driver/how-does-it-work

## Appendix A: Wiring Diagram



## Appendix B: Arduino Program

```
/*
 * FOR RUNNING OFF NANO W/ NO FLORA MICROCONTROLLER
 * Connect with MIT App inventor
 * Add characteristics for each output
 * Make serial connection optional
 * Add 2 more compass sensors
Judo Wearable
Mechanical Engineering Senior Design Project
Natalie Masters and Alison Waugh
University of Akron
Spring 2021
Evaluate the quality of a judo throw for training purposes
*/
//
//
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_Accel.h>
#include <Adafruit_LSM303DLH_Mag.h>
#include <Arduino_LSM9DS1.h>
#include <ArduinoBLE.h>
//
/* Assign a unique ID to the sensors at the same time */
Adafruit_LSM303_Accel_Unified accel1 = Adafruit_LSM303_Accel_Unified(00001);
Adafruit_LSM303DLH_Mag_Unified mag1 = Adafruit_LSM303DLH_Mag_Unified(00002);
//
Adafruit_LSM303_Accel_Unified accel2 = Adafruit_LSM303_Accel_Unified(00003);
Adafruit_LSM303DLH_Mag_Unified mag2 = Adafruit_LSM303DLH_Mag_Unified(00004);
//
Adafruit_LSM303_Accel_Unified accel3 = Adafruit_LSM303_Accel_Unified(00005);
Adafruit_LSM303DLH_Mag_Unified mag3 = Adafruit_LSM303DLH_Mag_Unified(00006);
//
Adafruit_LSM303_Accel_Unified accel4 = Adafruit_LSM303_Accel_Unified(00007);
Adafruit_LSM303DLH_Mag_Unified mag4 = Adafruit_LSM303DLH_Mag_Unified(10009);
//
/* Set which multiplexer port where each FLORA Compass is connected */
int sp1 = 1;  // Compass 1 (right hip) connected to multiplexer port 1
int sp2 = 2;  // Compass 2 (left hip) connected to multiplexer port 2
int sp3 = 4;  // Compass 3 (left shoulder) connected to multiplexer port 4
int sp4 = 6;  // Compass 4 (right shoulder) connected to multiplexer port 6
//
/* BLE service */
  BLEService judoService("1101");
  BLEFloatCharacteristic charXMagS1("2101", BLERead | BLENotify);
  BLEFloatCharacteristic charYMagS1("2102", BLERead | BLENotify);
```

```
  BLEFloatCharacteristic charZMagS1("2103", BLERead | BLENotify);
  BLEFloatCharacteristic charXAccelS1("2104", BLERead | BLENotify);
  BLEFloatCharacteristic charYAccelS1("2104", BLERead | BLENotify);
  BLEFloatCharacteristic charZAccelS1("2106", BLERead | BLENotify);
  //
  BLEFloatCharacteristic charXMagS2("2107", BLERead | BLENotify);
  BLEFloatCharacteristic charYMagS2("2108", BLERead | BLENotify);
  BLEFloatCharacteristic charZMagS2("2109", BLERead | BLENotify);
  BLEFloatCharacteristic charXAccelS2("2110", BLERead | BLENotify);
  BLEFloatCharacteristic charYAccelS2("2111", BLERead | BLENotify);
  BLEFloatCharacteristic charZAccelS2("2112", BLERead | BLENotify);
  //
  BLEFloatCharacteristic charXMagIMU("2113", BLERead | BLENotify);
  BLEFloatCharacteristic charYMagIMU("2114", BLERead | BLENotify);
  BLEFloatCharacteristic charZMagIMU("2115", BLERead | BLENotify);
  BLEFloatCharacteristic charXAccelIMU("2116", BLERead | BLENotify);
  BLEFloatCharacteristic charYAccelIMU("2117", BLERead | BLENotify);
  BLEFloatCharacteristic charZAccelIMU("2118", BLERead | BLENotify);
  BLEFloatCharacteristic charXGyroIMU("2119", BLERead | BLENotify);
  BLEFloatCharacteristic charYGyroIMU("2120", BLERead | BLENotify);
  BLEFloatCharacteristic charZGyroIMU("2121", BLERead | BLENotify);
  //
  BLEFloatCharacteristic charXMagS3("2122", BLERead | BLENotify);
  BLEFloatCharacteristic charYMagS3("2123", BLERead | BLENotify);
  BLEFloatCharacteristic charZMagS3("2124", BLERead | BLENotify);
  BLEFloatCharacteristic charXAccelS3("2125", BLERead | BLENotify);
  BLEFloatCharacteristic charYAccelS3("2126", BLERead | BLENotify);
  BLEFloatCharacteristic charZAccelS3("2127", BLERead | BLENotify);
  //
  BLEFloatCharacteristic charXMagS4("2128", BLERead | BLENotify);
  BLEFloatCharacteristic charYMagS4("2129", BLERead | BLENotify);
  BLEFloatCharacteristic charZMagS4("2130", BLERead | BLENotify);
  BLEFloatCharacteristic charXAccelS4("2131", BLERead | BLENotify);
  BLEFloatCharacteristic charYAccelS4("2132", BLERead | BLENotify);
  BLEFloatCharacteristic charZAccelS4("2133", BLERead | BLENotify);
  //
/* Function to display the details of the sensors in the serial monitor */
void displaySensorDetails(Adafruit_LSM303_Accel_Unified *accel)
{
  sensor_t sensor;
  accel->getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" m/s^2");
```

```
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" m/s^2");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" m/s^2");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}
void displaySensorDetails(Adafruit_LSM303DLH_Mag_Unified *mag)
{
  sensor_t sensor;
  mag->getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" uT");
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" uT");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" uT");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}
//
/* Tool to select which multiplexer port to call */
#define TCAADDR 0x70 //Mulitplexer addresss
void tcaselect(uint8_t i) {
  if (i > 7) return;
  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << i);
  Wire.endTransmission();
}
//
//
/* Setup code to run once. Initialize sensors, open serial monitor, check connections */
void setup() {
  Serial.begin(115200);
  Wire.begin();
  //
  pinMode(LED_BUILTIN, OUTPUT); // Sets board built-in LED as an output
  /* Begin BLE */
  if (!BLE.begin())
  {
    if (Serial)
    {
      Serial.println("starting BLE failed!");
    }
    while (1);
```

```
}
/* BLE setup: set name, add characteristics, add service */
  BLE.setLocalName("JudoWearable");
  BLE.setAdvertisedService(judoService);
  judoService.addCharacteristic(charXMagS1);
  judoService.addCharacteristic(charYMagS1);
  judoService.addCharacteristic(charZMagS1);
  judoService.addCharacteristic(charXAccelS1);
  judoService.addCharacteristic(charYAccelS1);
  judoService.addCharacteristic(charZAccelS1);
  judoService.addCharacteristic(charXMagS2);
  judoService.addCharacteristic(charYMagS2);
  judoService.addCharacteristic(charZMagS2);
  judoService.addCharacteristic(charXAccelS2);
  judoService.addCharacteristic(charYAccelS2);
  judoService.addCharacteristic(charZAccelS2);
  judoService.addCharacteristic(charXMagIMU);
  judoService.addCharacteristic(charYMagIMU);
  judoService.addCharacteristic(charZMagIMU);
  judoService.addCharacteristic(charXAccelIMU);
  judoService.addCharacteristic(charYAccelIMU);
  judoService.addCharacteristic(charZAccelIMU);
  judoService.addCharacteristic(charXGyroIMU);
  judoService.addCharacteristic(charYGyroIMU);
  judoService.addCharacteristic(charZGyroIMU);
  judoService.addCharacteristic(charXMagS3);
  judoService.addCharacteristic(charYMagS3);
  judoService.addCharacteristic(charZMagS3);
  judoService.addCharacteristic(charXAccelS3);
  judoService.addCharacteristic(charYAccelS3);
  judoService.addCharacteristic(charZAccelS3);
  judoService.addCharacteristic(charXMagS4);
  judoService.addCharacteristic(charYMagS4);
  judoService.addCharacteristic(charZMagS4);
  judoService.addCharacteristic(charXAccelS4);
  judoService.addCharacteristic(charYAccelS4);
  judoService.addCharacteristic(charZAccelS4);
  BLE.addService(judoService);
/* Advertise BLE service to devices */
BLE.advertise();
//
/* Initialize compasses. Output basic sensor information */
// Compass Sensor 1
tcaselect(1);
mag1.enableAutoRange(true);
accel1.setRange(LSM303_RANGE_2G);
```

```
accel1.begin();
mag1.begin();
if(Serial)
{
  Serial.println("Basic sensor information:");
  displaySensorDetails(&mag1);
  displaySensorDetails(&accel1);
}
//
// Compass Sensor 2
tcaselect(2);
mag2.enableAutoRange(true);
accel2.setRange(LSM303_RANGE_2G);
accel2.begin();
mag2.begin();
if (Serial)
{
  displaySensorDetails(&mag2);
  displaySensorDetails(&accel2);
}
//
// Compass Sensor 3
tcaselect(4);
mag3.enableAutoRange(true);
accel3.setRange(LSM303_RANGE_2G);
accel3.begin();
mag3.begin();
if (Serial)
{
  displaySensorDetails(&mag3);
  displaySensorDetails(&accel3);
}
//
// Compass Sensor 4
tcaselect(6);
mag4.enableAutoRange(true);
accel4.setRange(LSM303_RANGE_2G);
accel4.begin();
mag4.begin();
if (Serial)
{
  displaySensorDetails(&mag4);
  displaySensorDetails(&accel4);
}
//
/* Initialize Nano 9DOF IMU */
```

```
  if (!IMU.begin())
  {
    if (Serial.available())
    {
      Serial.println("Failed to initialize IMU!");
    }
    while (2);
  }
  //
  if (Serial)
  {
    Serial.println("Nano 9DOF IMU test");
    Serial.println("Nano 9DOF basic information");
    Serial.print("Accelerometer sample rate = ");
    Serial.print(IMU.accelerationSampleRate());
    Serial.println(" Hz");
    Serial.println();
    //
    Serial.print("Gyroscope sample rate = ");
    Serial.print(IMU.gyroscopeSampleRate());
    Serial.println(" Hz");
    Serial.println();
    //
    Serial.print("Magnetic field sample rate = ");
    Serial.print(IMU.magneticFieldSampleRate());
    Serial.println(" uT");
    Serial.println();
  }

  BLEDevice central = BLE.central();
  //while(!central.connected());
}
//
//
/* Now shit gets real. Time to start pulling numbers and evaluating */
void loop() {
  /* Connect BLE to central device*/
  BLEDevice central = BLE.central();
  if (central)
  {
    if (Serial)
    {
      Serial.print("Connected to central: ");
      Serial.println(central.address());
    }
    digitalWrite(LED_BUILTIN, HIGH);
```

```
}
/* Get new sensor event */
sensors_event_t event;
//
/* Call Compass 1 sensor */
  tcaselect(1);
  float xMagS1, yMagS1, zMagS1, xAccelS1, yAccelS1, zAccelS1;
  if (mag1.begin())
  {
    mag1.getEvent(&event);
    xMagS1 = event.magnetic.x;
    yMagS1 = event.magnetic.y;
    zMagS1 = event.magnetic.z;
  } else {
    xMagS1 = 42;
    yMagS1 = 42;
    zMagS1 = 42;
  }
  //
  if (accel1.begin())
  {
    accel1.getEvent(&event);
    xAccelS1 = event.acceleration.x;
    yAccelS1 = event.acceleration.y;
    zAccelS1 = event.acceleration.z;
  } else {
    xAccelS1 = 42;
    yAccelS1 = 42;
    zAccelS1 = 42;
  }
//
/* Call Compass 2 sensor */
  tcaselect(2);
  float xMagS2, yMagS2, zMagS2, xAccelS2, yAccelS2, zAccelS2;
  mag2.getEvent(&event);
    xMagS2 = event.magnetic.x;
    yMagS2 = event.magnetic.y;
    zMagS2 = event.magnetic.z;
    /*
    xMagS2 = 42;
    yMagS2 = 42;
    zMagS2 = 42;
    */
  //
  if (accel2.begin())
  {
```

```
    accel2.getEvent(&event);
    xAccelS2 = event.acceleration.x;
    yAccelS2 = event.acceleration.y;
    zAccelS2 = event.acceleration.z;
   } else {
    xAccelS2 = 42;
    yAccelS2 = 42;
    zAccelS2 = 42;
   }
//
/* Call Compass 3 sensor */
  tcaselect(4);
  float xMagS3, yMagS3, zMagS3, xAccelS3, yAccelS3, zAccelS3;
  if (mag3.begin())
  {
    mag3.getEvent(&event);
    xMagS3 = event.magnetic.x;
    yMagS3 = event.magnetic.y;
    zMagS3 = event.magnetic.z;
  } else {
    xMagS3 = 42;
    yMagS3 = 42;
    zMagS3 = 42;
  }
  //
  if (accel3.begin())
  {
  accel3.getEvent(&event);
  xAccelS3 = event.acceleration.x;
  yAccelS3 = event.acceleration.y;
  zAccelS3 = event.acceleration.z;
  } else {
    xAccelS3 = 42;
    yAccelS3 = 42;
    zAccelS3 = 42;
  }
//
/* Call Compass 4 sensor */
  tcaselect(6);
  float xMagS4, yMagS4, zMagS4, xAccelS4, yAccelS4, zAccelS4;
  if (mag4.begin())
  {
  mag4.getEvent(&event);
  xMagS4 = event.magnetic.x;
  yMagS4 = event.magnetic.y;
  zMagS4 = event.magnetic.z;
```

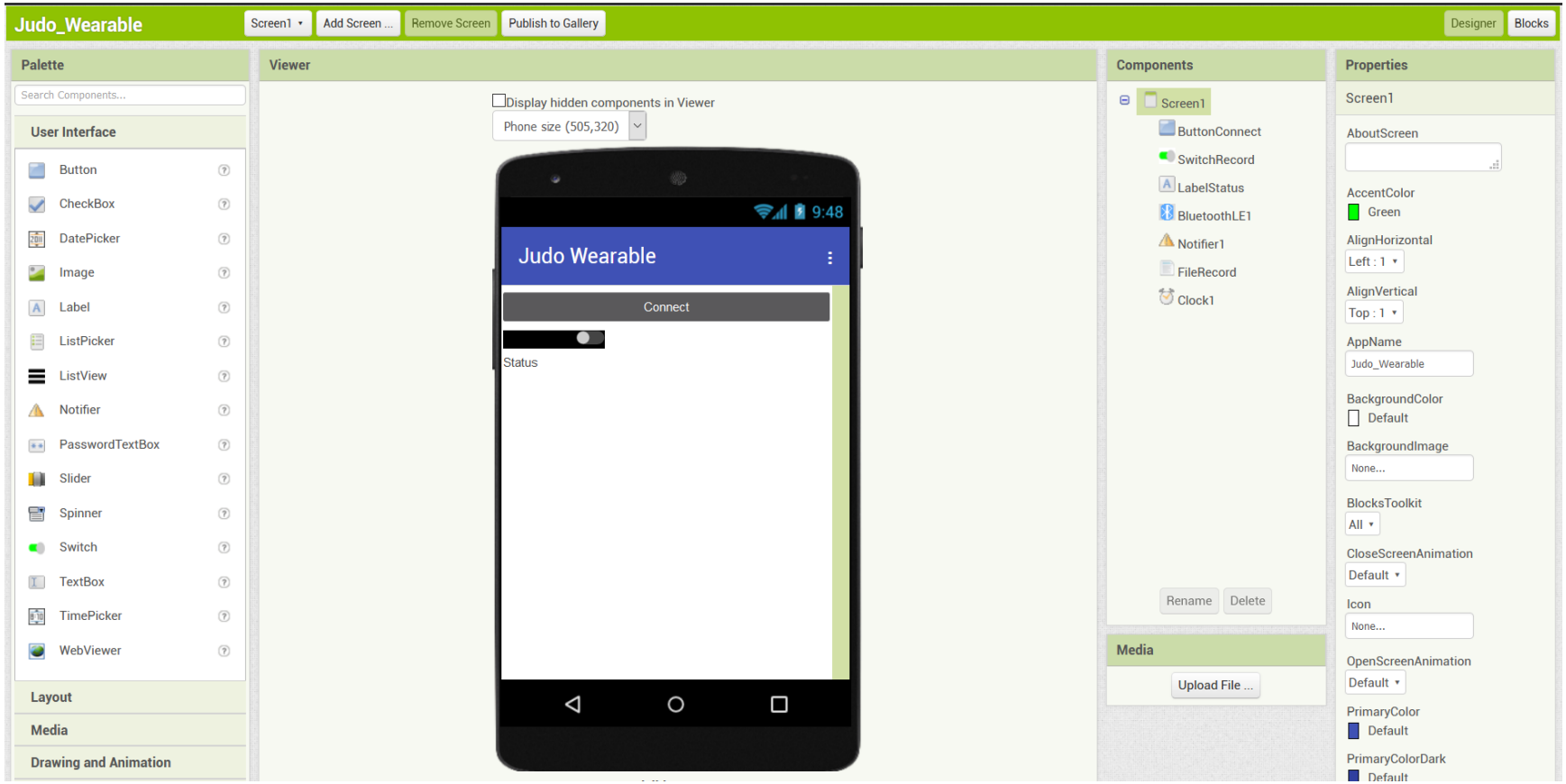
```
  } else {
  xMagS4 = 42;
  yMagS4 = 42;
  zMagS4 = 42;

  }
  //
  if (accel4.begin())
  {
  accel4.getEvent(&event);
  xAccelS4 = event.acceleration.x;
  yAccelS4 = event.acceleration.y;
  zAccelS4 = event.acceleration.z;
  } else {
    xAccelS4 = 42;
    yAccelS4 = 42;
    zAccelS4 = 42;
  }
 //
 /* Call Nano 33BLE IMU 9DOF sensor */
 float xAccelIMU, yAccelIMU, zAccelIMU, xMagIMU, yMagIMU, zMagIMU, xGyroIMU,
yGyroIMU, zGyroIMU;
 if (IMU.accelerationAvailable()) {
  IMU.readAcceleration(xAccelIMU, yAccelIMU, zAccelIMU);
 }//
 if (IMU.magneticFieldAvailable()) {
  IMU.readMagneticField(xMagIMU, yMagIMU, zMagIMU);
 }//
 if (IMU.gyroscopeAvailable()) {
  IMU.readGyroscope(xGyroIMU, yGyroIMU, zGyroIMU);
 }
 //
 /* Write the results to the serial if available */
 if (Serial)
 {
  /* Display the results (magnetic vector values are in micro-Tesla (uT)) */
  Serial.println("Sensor #1 Magnetometer - ");
  Serial.print("X: "); Serial.print(xMagS1); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yMagS1); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zMagS1); Serial.print("  ");Serial.println("uT");
  Serial.println("Sensor #1 Accelerometer - ");
  Serial.print("X: "); Serial.print(xAccelS1); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yAccelS1); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zAccelS1); Serial.print("  ");Serial.println("m/s^2");
  //
  /* Display the results (magnetic vector values are in micro-Tesla (uT)) */
```

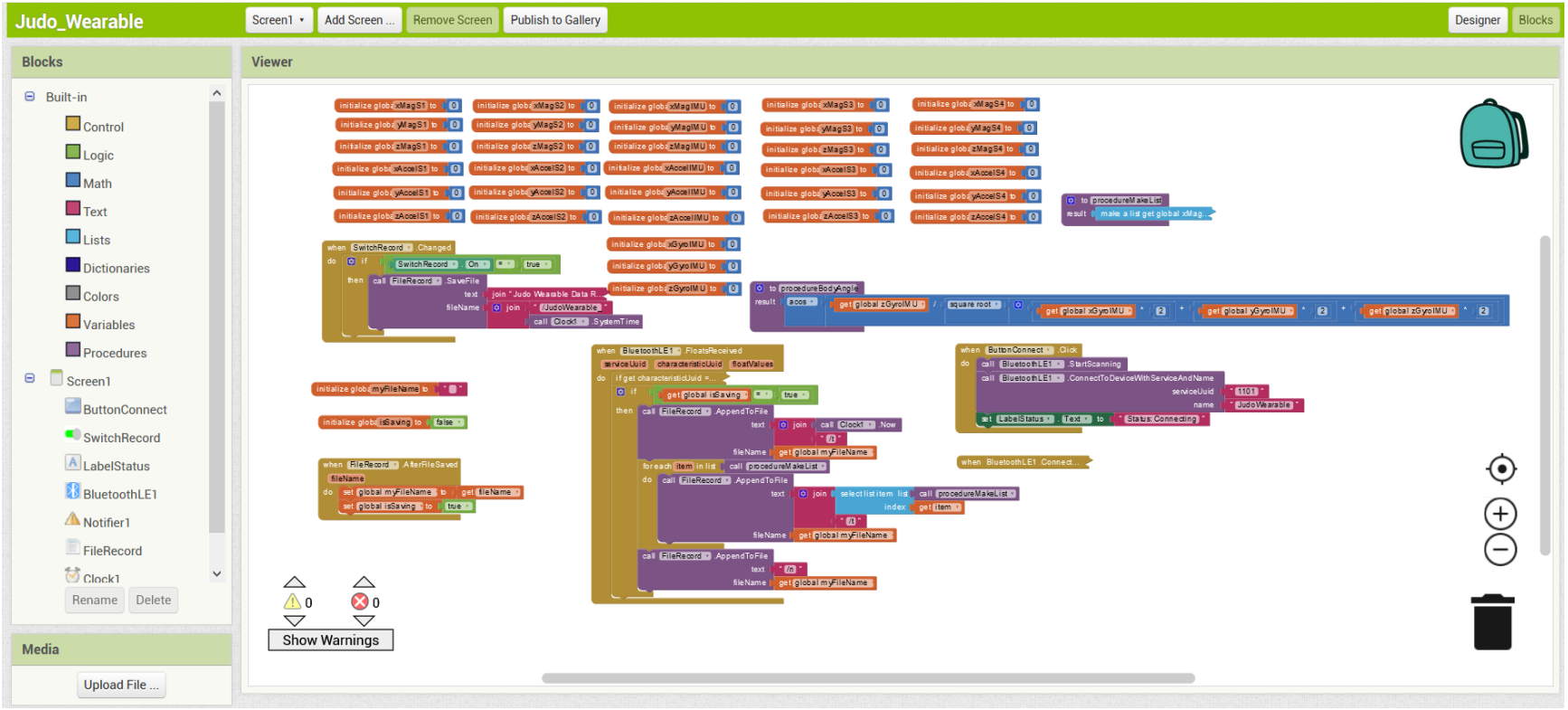
```
  Serial.println("Sensor #2 Magnetometer - ");
  Serial.print("X: "); Serial.print(xMagS2); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yMagS2); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zMagS2); Serial.print("  ");Serial.println("uT");
  Serial.println("Sensor #2 Accelerometer - ");
  accel2.getEvent(&event);
  Serial.print("X: "); Serial.print(xAccelS2); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yAccelS2); Serial.print("  ");
  Serial.print("Z: "); Serial.print(xAccelS2); Serial.print("  ");Serial.println("m/s^2");
  //
  Serial.println("Sensor #3 Magnetometer - ");
  Serial.print("X: "); Serial.print(xMagS3); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yMagS3); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zMagS3); Serial.print("  ");Serial.println("uT");
  Serial.println("Sensor #3 Accelerometer - ");
  Serial.print("X: "); Serial.print(xAccelS3); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yAccelS3); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zAccelS3); Serial.print("  ");Serial.println("m/s^2");
  //
  Serial.println("Sensor #4 Magnetometer - ");
  Serial.print("X: "); Serial.print(xMagS4); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yMagS4); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zMagS4); Serial.print("  ");Serial.println("uT");
  Serial.println("Sensor #4 Accelerometer - ");
  Serial.print("X: "); Serial.print(xAccelS4); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yAccelS4); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zAccelS4); Serial.print("  ");Serial.println("m/s^2");
  //
  Serial.println("Nano 9DOF IMU");
  Serial.println("Accelerometer - ");
  Serial.print("X: "); Serial.print(xAccelIMU); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yAccelIMU); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zAccelIMU); Serial.print("  ");Serial.println("m/s^2");
  //
  Serial.println("Magnetometer - ");
  Serial.print("X: "); Serial.print(xMagIMU); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yMagIMU); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zMagIMU); Serial.print("  ");Serial.println("uT");
  //
  Serial.println("Gyroscope - ");
  Serial.print("X: "); Serial.print(xGyroIMU); Serial.print("  ");
  Serial.print("Y: "); Serial.print(yGyroIMU); Serial.print("  ");
  Serial.print("Z: "); Serial.print(zGyroIMU); Serial.print("  ");Serial.println("degrees/second");
}
//
/* Write the BLE characteristic */
```

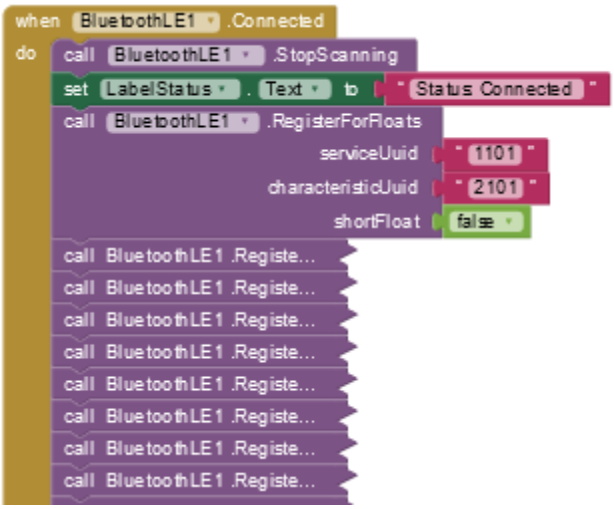
```
if (central.connected())
{
   charXMagS1.writeValue(xMagS1);
   charYMagS1.writeValue(yMagS1);
   charZMagS1.writeValue(zMagS1);
   charXAccelS1.writeValue(xAccelS1);
   charYAccelS1.writeValue(yAccelS1);
   charZAccelS1.writeValue(zAccelS1);
   charXMagS2.writeValue(xMagS2);
   charYMagS2.writeValue(yMagS2);
   charZMagS2.writeValue(zMagS2);
   charXAccelS2.writeValue(xAccelS2);
   charYAccelS2.writeValue(yAccelS2);
   charZAccelS2.writeValue(zAccelS2);
   charXMagIMU.writeValue(xMagIMU);
   charYMagIMU.writeValue(yMagIMU);
   charZMagIMU.writeValue(zMagIMU);
   charXAccelIMU.writeValue(xAccelIMU);
   charYAccelIMU.writeValue(yAccelIMU);
   charZAccelIMU.writeValue(zAccelIMU);
   charXGyroIMU.writeValue(xGyroIMU);
   charYGyroIMU.writeValue(yGyroIMU);
   charZGyroIMU.writeValue(zGyroIMU);
   charXMagS3.writeValue(xMagS3);
   charYMagS3.writeValue(yMagS3);
   charZMagS3.writeValue(zMagS3);
   charXAccelS3.writeValue(xAccelS3);
   charYAccelS3.writeValue(yAccelS3);
   charZAccelS3.writeValue(zAccelS3);
   charXMagS4.writeValue(xMagS4);
   charYMagS4.writeValue(yMagS4);
   charZMagS4.writeValue(zMagS4);
   charXAccelS4.writeValue(xAccelS4);
   charYAccelS4.writeValue(yAccelS4);
   charZAccelS4.writeValue(zAccelS4);
   delay(100);
}
if (!central.connected())
{
  if (Serial)
  {
    Serial.println("");
    Serial.print("Disconnected from central: ");
    Serial.println(central.address());
  }
  digitalWrite(LED_BUILTIN, LOW);
```

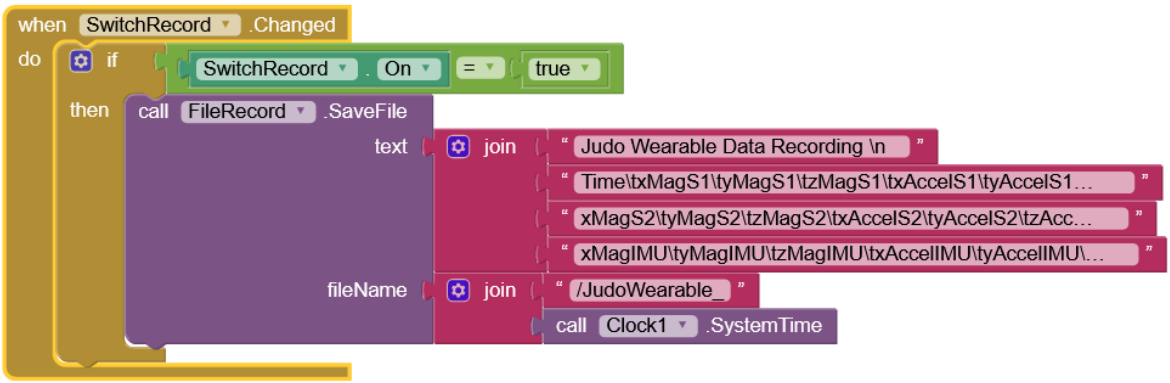
```
  }
  //
  delay(500);
}
```

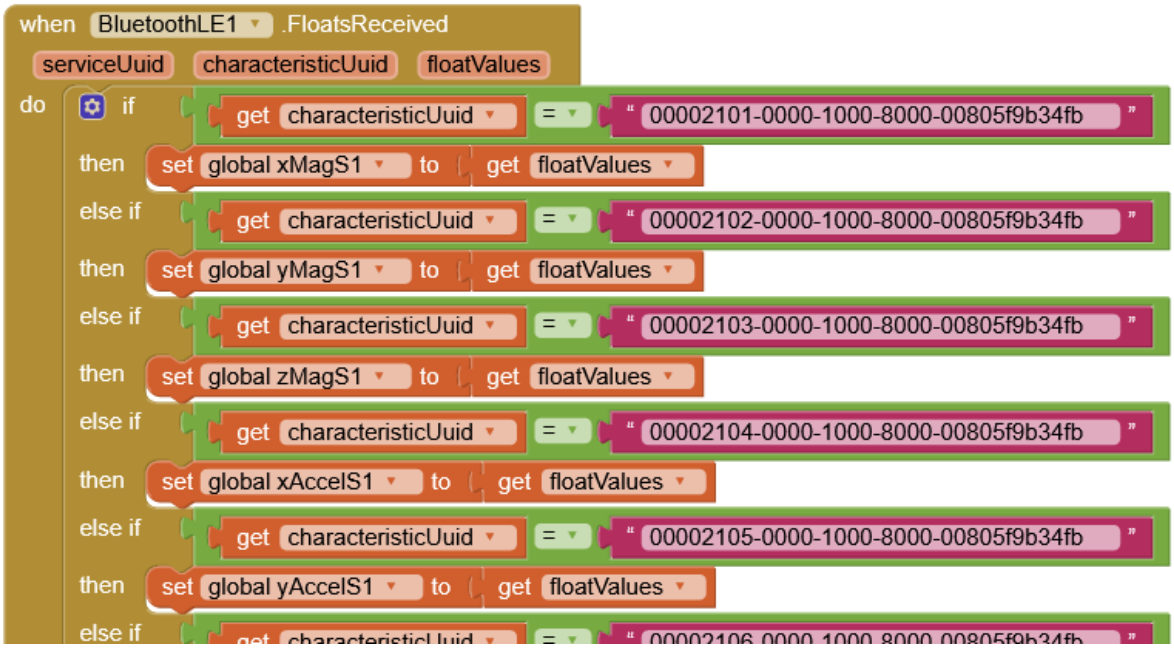
# Appendix C: MIT App Inventor




Judo_Wearable
Screen1
Add Screen ...
Remove Screen
Publish to Gallery
Designer
Blocks
Blocks
Built-in
Control
Logic
Math
Text
Lists
Dictionaries
Colors
Variables
Procedures
Screen1
ButtonConnect
SwitchRecord
LabelStatus
BluetoothLE1
Notifier1
FileRecord
Clock1
Rename
Delete
Media
Upload File ...
Viewer
Show Warnings


when BluetoothLE1 .Connected
do call BluetoothLE1 .StopScanning
set LabelStatus . Text to " Status: Connected "
call BluetoothLE1 .RegisterForFloats
serviceUuid " 1101 "
characteristicUuid " 2101 "
shortFloat false
call BluetoothLE1 .Registe...
call BluetoothLE1 .Registe...
call BluetoothLE1 .Registe...
call BluetoothLE1 .Registe...
call BluetoothLE1 .Registe...
call BluetoothLE1 .Registe...
call BluetoothLE1 .Registe...
call BluetoothLE1 .Registe...


when SwitchRecord .Changed
do if SwitchRecord . On = true
then call FileRecord .SaveFile
text join " Judo Wearable Data Recording \n "
" Time\txMagS1\tyMagS1\tzMagS1\txAccelS1\tyAccelS1... "
" xMagS2\tyMagS2\tzMagS2\txAccelS2\tyAccelS2\tzAcc... "
" xMagIMU\tyMagIMU\tzMagIMU\txAccelIMU\tyAccelIMU\... "
fileName join " /JudoWearable_ "
call Clock1 .SystemTime


when BluetoothLE1 .FloatsReceived
serviceUuid characteristicUuid floatValues
do if get characteristicUuid = " 00002101-0000-1000-8000-00805f9b34fb "
then set global xMagS1 to get floatValues
else if get characteristicUuid = " 00002102-0000-1000-8000-00805f9b34fb "
then set global yMagS1 to get floatValues
else if get characteristicUuid = " 00002103-0000-1000-8000-00805f9b34fb "
then set global zMagS1 to get floatValues
else if get characteristicUuid = " 00002104-0000-1000-8000-00805f9b34fb "
then set global xAccelS1 to get floatValues
else if get characteristicUuid = " 00002105-0000-1000-8000-00805f9b34fb "
then set global yAccelS1 to get floatValues
else if