Senior Design Capstone 2023
uMove

AJ Funari, Ryan Herrington,
Isaac Navarete, Ben Bamidele

May 2nd, 2023

Table of Contents

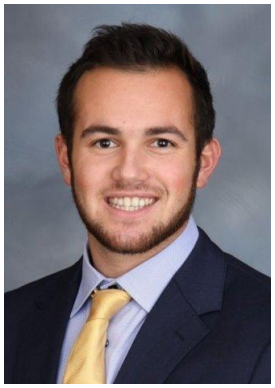
Section 1: Team Introduction

AJ Funari is a senior studying Intelligent Systems Engineering with a concentration in Computer Engineering and Cyber-Physical Systems. He has also earned himself a mathematics and business minor. AJ is working towards a master's degree in Intelligent Systems Engineer with a concentration in Cyber-Physical Systems, planning to graduate in December, 2023. He joined the Vehicle Autonomy and Intelligence Lab (V.A.I.L) his sophomore year. AJ currently researches integrating artificial intelligence with different drive robots to operate autonomously. He is experienced with ROS, Gazebo, and Rviz. Last year, he participated in an IT (Information Technology) and Cybersecurity internship at Curia Global. This summer he will begin his career as an Engineer at the Naval Surface Warfare Center located in Crane, Indiana. AJ will join their Special Forces and Expeditionary Team where he will spend time researching and building Unmanned Surface Vessels (USVs). Finally, he hopes to begin a Ph.D. in Intelligent Systems Engineering in January, 2024.

Ryan Herrington is a senior studying Intelligent Systems Engineering with a concentration in Computer Engineering and Cyber-Physical Systems. He has also earned a mathematics minor. Ryan will be graduating Spring of 2023 with a Bachelors. In high school, Ryan was the lead programmer of his robotics team which helped him learn sensor unification and robotics control. During college he became well versed with embedded systems. He also had an internship with JFW Industries where he learnt to solder among other skills. After graduation Ryan will begin his career at the Naval Surface Warfare Center located in Crane, Indiana. Ryan will be joining their Embedded Systems team where he will spend time reverse engineering embedded systems.

Isaac is also a Cyber-Physical systems concentration that is familiar with machine learning, physiological signal processing, and embedded systems. He is a current research assistant in the Socio Neural Physiological Lab where he creates software for data analysis and data collection from different physiological signals. He will be pursuing a career at NSWC Crane after graduation.

Ben is a Cyber-Physical Systems major with expertise in Embedded Systems, Digital Design, and machine learning. His background involved undergraduate research at Vehicle Autonomy and Intelligence Lab and an internship/job in Electromagnetic Warfare Division at NSWC (Naval Surface Warfare Center) Crane. He will be pursuing a career at Ford Motor Company after graduation.



AJ Funari
Project Coordinator

Ryan Herrington
System Manager

Isaac Navarette
Presentation Manager

Ben Bamidele
Report Manager

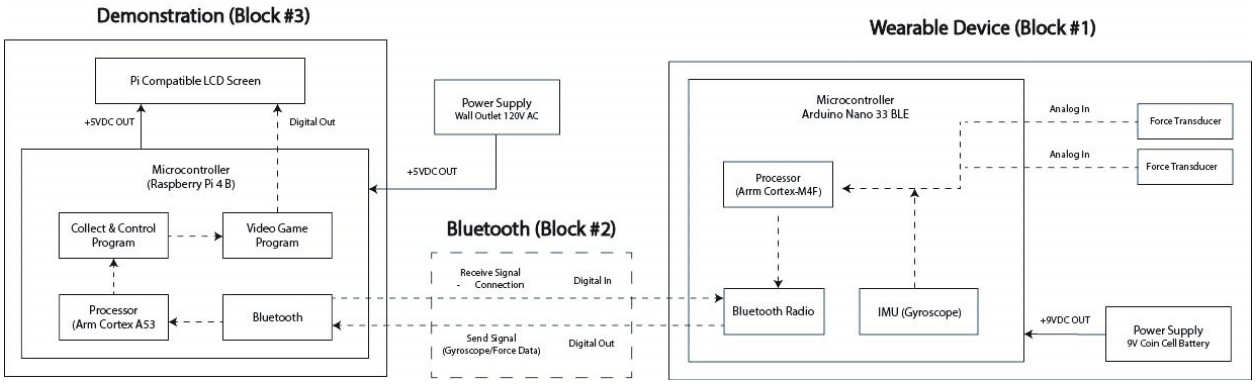
Section 2: Project Introduction

The goal of our project is to create a wearable device to be used by Luddy Outreach to draw in new students at recruiting events, and to also be used at Luddy Education events to help instruct younger students about what Luddy does and get them interested in STEM (Science, Technology, Engineering, and Math) fields. With this goal in mind, we have two major parts of the project. The first major part of our project is the wearable device that will be used to collect and transmit data using Bluetooth Low Energy to control a video game. An Arduino Nano 33 BLE is the microcontroller associated with the wearable device. It includes an on board Inertial Measurement Unit (IMU), multiple analog pins, and Bluetooth Low Energy capability. The IMU is necessary to calculate the orientation of the wearable device. Two analog pins are used to collect data from finger pressure. A custom circuit is integrated with the Arduino Nano to complete the functionality. The circuit includes two force transducers, a battery, and a switch. Bluetooth Low Energy is used to transmit the data and communicate with the Raspberry Pi.

The second portion is the demonstration which includes the display screen and Raspberry Pi. The demonstration part of the project is responsible for receiving data via Bluetooth from the wearable device and converting the input to a control output for the video game to interpret and display on the screen. This section includes the Raspberry Pi for processing and receiving data, along with providing hardware for the video game to run and a screen to display the video game output so the user gets instant feedback on how the wearable is controlling the video game. The demonstration will be built into a case to protect the hardware, keep users safe, and provide easy mobility. The case is responsible for supporting the demonstration and providing a temporary place to store the wearable device and other supplies.

Section 2.1: Block Diagram

The basic block diagram is broken down into four main parts. Block 1 encompasses two force transducers. Block 2 consists of the inertial measurement unit (IMU). Block 3 includes Bluetooth Low Energy. Block 4 is the demonstration, including the control algorithm and video game.



Section 2.2: Project Planning

In project planning, the group decided it would be best to assign separate roles to each member. The Project Coordinator is AJ, System Manager is Ryan, Presentation Manager is Isaac, and the Report Manager is Ben. The Project Coordinator is a task given to the person who owns the MS Excel Project Plan documents, tracks, and reports progress to the project plan, and reports progress to the instructor. Ryan responsibilities as the System Manager includes integrating and populating the Detailed Block design plans, responsibility for the Safety & Compliance Features in the product, and reviews for Completeness ALL Block-Block Interface Requirements. The Presentation Manager roles must submit electronic files for all Prelim Presentations and Final Presentation, keep backup of the master slide set, and own the master MS PPT Presentation slide set for the team. Ben's responsibilities as Report Manager is to own the master MS Word Project Report file for the team, responsible for good visibility and proper spelling in report, and keeps backup copy of master report and integrates weekly progress submissions from ALL team members. The Gantt Chart is shown below, it shows the group's progress for each of the tasks that we must complete. It highlights the owner of each task as well. For example, the design structure of the wearable device belongs to Ben and Isaac.

TASK TITLE	TASK OWNER	PCT OF TASK COMPLETE
Project Conception and Initiation		
Acquire Customer	Isaac	100%
Research	All	100%
Development of Wearable		
Choose Sensors	All	100%
Data Collection	Isaac/Ben	100%
Power Supply Consumption	Isaac	100%
Prototype Design		
Wearable Device	AJ	100%
Case Enclosure	AJ	100%
Software Development		
Bluetooth Integration	AJ	100%
Sensor Integration	Ryan	100%
Control Function	Ryan	100%
Wearable Demo - Luddy ISE		
Video Game Integration	Ryan	100%
Play Testing	Ryan	100%
Documentation		
Presentation	Isaac	100%
Report	Ben	100%

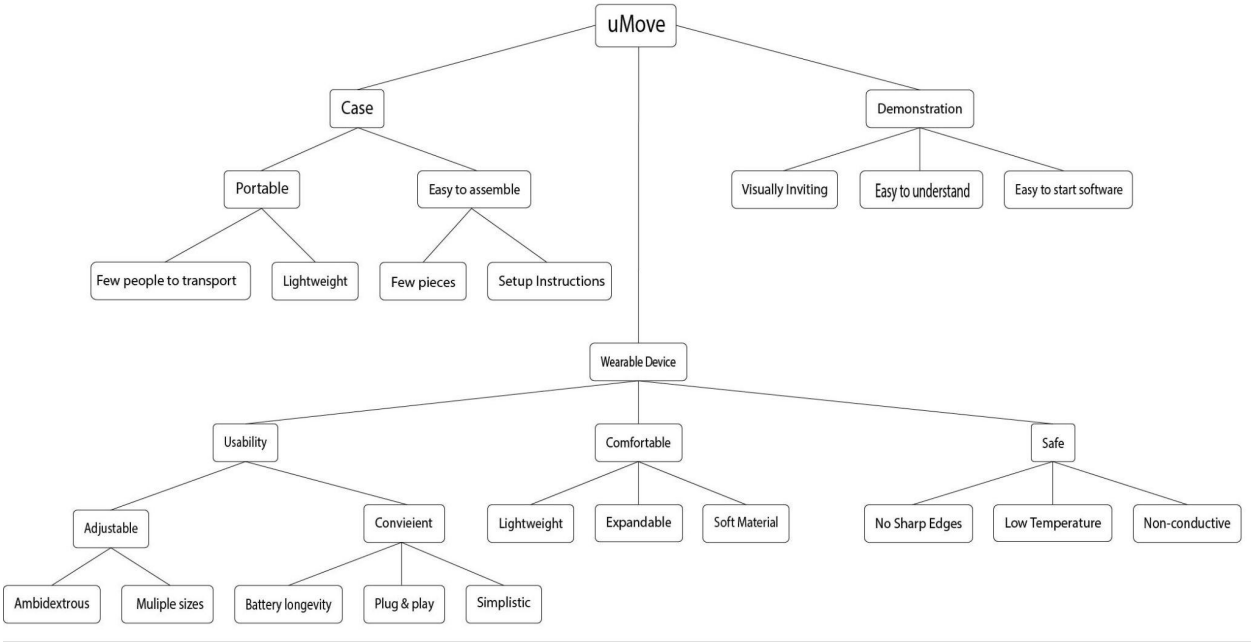
Section 2.3: Patents

Each of the group members produced three “active” patents that would be useful in understanding diverse ways to complete the design. The patents can be used as sources of inspiration for different approaches on products/designs that are similar to the project. Another important reason researching patents is important is to avoid infringing on any existing patents. After the group each searched for three patents, the group collaborated to identify the best three patents that would be worthwhile in creating a mitigation strategy for our project. Patent 1’s Number is US 2011005436A1 and there was no infringement on that because we are using force sensors on the tip of the finger to identify gestures along with the IMU. Patent 2’s Number is 20180311575 and there was also no infringement on that since we do all sensing on board the wearable device. Patent 3’s Number was identified as US 10191544 B2. We did not infringe on this research method through the use of a wristband/glove to control a simulated game, not a robot. Patent 3 uses a robot as the mechanism that is being controlled. The figure below gives greater detail for each patent by describing its purpose and motive.

	Summary
Patent 1	In this patent they describe a device that is used to identify gestures. They used an IMU in this design to aid with classification, however they claim that a device with a signal with a transmitter and receiver (i.e., ultrasonic wave, infrared, or a laser) pointed at the carpal tunnel in your wrist is infringement.
Patent 2	This patent is for a controller used with a VR (Virtual Reality) headset. External sensors detect the LED ring to determine where it is in space relative to the headset to give a position in VR. Uses wireless communication to send over controller input data.
Patent 3	A sensing device incorporated into wristband and having sensors to provide signal inputs relating to the pitch, roll and yaw movements of a user's wrist. A continuous feedback loop with a drift detector receives the sensor signal inputs and a rotation matrix input and uses the sensor signal inputs to calculate a numerical error, A proportional-integral controller receives the calculated numerical error output from the drift detector and outputs a control output to a drift adjuster.

Section 2.4: Objectives

These are the goals/objectives that we aimed towards in our project and wanted to accomplish; however, it will not be a failure if we do not meet certain objectives. We split this project into three main parts. The case where the whole demonstration will be stored and transported. Then we had our wearable device, and finally our demonstration, which allows us to demonstrate the capabilities of our wearable device.



We want our case to be portable to where one person can hold and transport the complete system. This objective helps us decide what kind of case we want, and whether we would purchase one or make one ourselves. For our final project when ended up building our case from scratch. The main components of the case were laser cut pieces and wood that were not heavy. The case has a small compartment below the screen that could be used as a storage place if a recruiter wanted to store some additional materials inside. Our wearable devices have a lot more objectives that we want to work towards. Analogous to the portability of the case, we wanted our device to be easy to take on and off to the point where anybody can see it and have some intuition on how it fits in your hand. The more people the device can fit on the better. However, we want to make sure that the device is safe and comfortable. Any hard or sharp edges hinder both objectives and any conductive materials must not be exposed to the user. This was accomplished by a strapped wristband that was built into the enclosure. The straps make it easy to adjust to different hand sizes. Finally, our demonstration must be visually stimulating. This was accomplished by the combination

of wood and 3d printed materials on the outside of the case that give the project a vintage but sleek style. We wanted to be able to grab people's attention and retain it for as long as we want. That way there is more curiosity for incoming first-year students about our major. For the user who is setting up the demo, we would want to make it as seamless as possible to do so.

Section 2.5: Constraints

For our project, we were able to determine numerous constraints that would help mold our design decisions and ultimate design. The constraints our team was able to produce are listed in the table below.

System Section	Constraints		
Total System	One person portable	Cannot exceed 15 lbs.	Cannot shock User
Wearable	Battery replaceable	Cannot exceed 2 lbs.	Fit inside case
Case	Rest on table	Safely Secure Glove	Safely contain demonstration
Demonstration	Compatible with wearable	1-page manual	2-minute usability

Section 2.6: Functions

Based on our Objectives, we created a list of functions that each part of the system should accomplish. These functions are specific to their respective systems and were organized in the table below to show the functions of the systems we are creating.

System Section	Functions			
Wearable	Wearable	Communicate	Collect Axis Data	Collect Joint Data
Case	Portable	Protective	...	...
Demonstration	Display Data	Communicate	Interactive	Translate Data

Section 2.7: Design Alternatives

For our design alternatives we created tables to help us determine what else we could do that is separate from our chosen design. Our tables correspond to what part of the project each function and mean pair relate to. Our tables are organized to show what our current design is in the first column, the columns after show what our next choices would be if the first option were unfeasible.

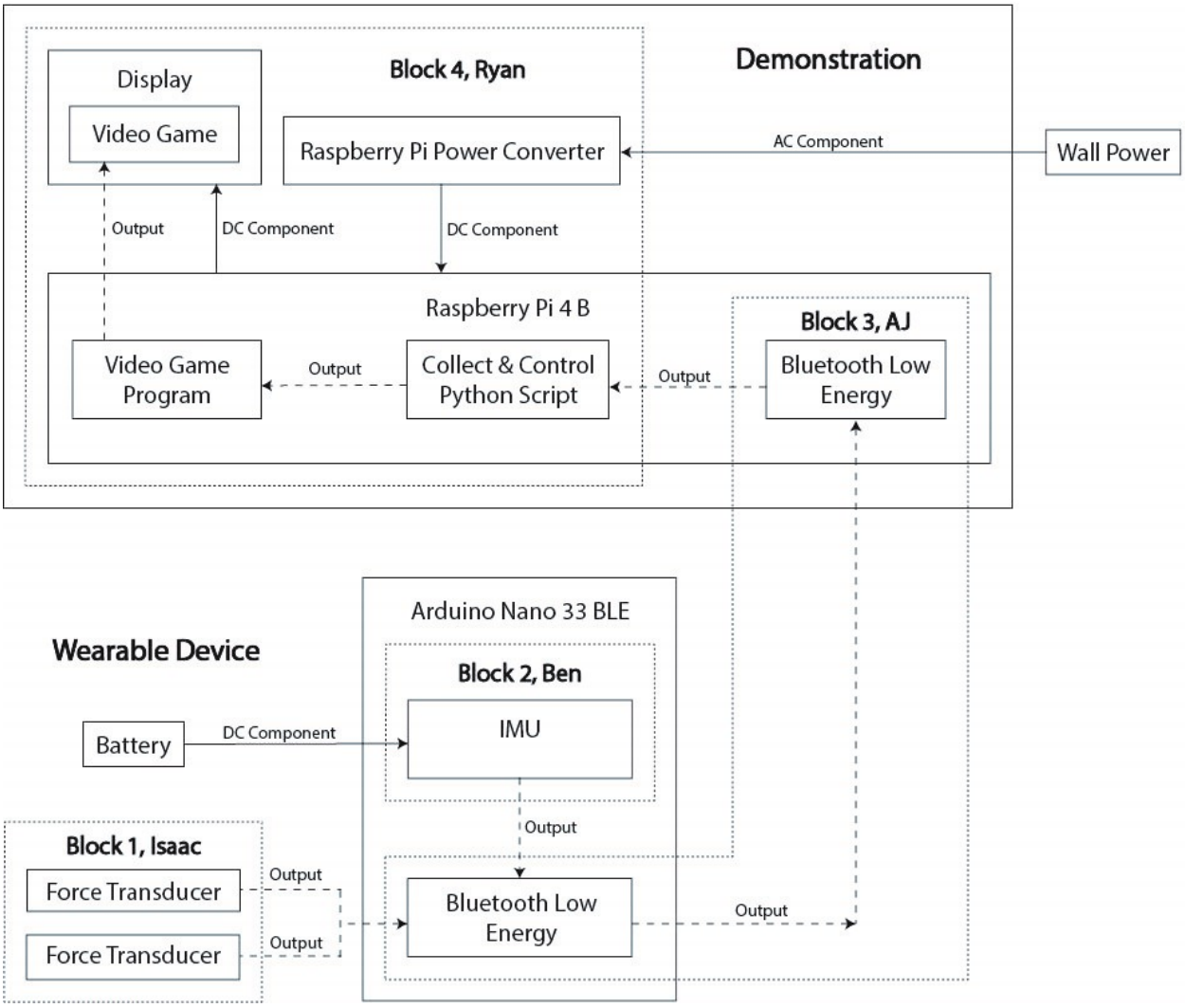
Wearable Device					
Function	**Means**				
Wearable	Velcro strap	Glove	Wristband	Handheld	Gripper
Communication	Bluetooth	Serial	----------	----------	----------
Collect Axis Data	IMU	Gyroscope	----------	----------	----------
Collect Joint Data	Pressure sensor	Conductive fabric	Button	Flex sensor	IMU
Power	Coin Cell	Lithium Ion	Alkaline Battery	----------	----------

Demonstration					
Function	**Means**				
Display Data	Monitor + Pi	Laptop	----------	----------	----------
Interactive	Custom Video Game	Emulation	ROS Simulation	----------	----------
Communication	Bluetooth	USB/Serial	----------	----------	----------
Translate Data	Python	C/C++	----------	----------	----------

Case					
Function	**Means**				
Portable	Custom Case	Briefcase	Antistatic Bag	----------	----------
Protective	Foam	Hard Outer Shell	----------	----------	----------

Section 2.8: Advanced Block Diagram

The detailed block diagram, or advanced block diagram is broken down into 3 distinct parts. Block 1 for the wearable device, Block 2 for Bluetooth, Block 3 for the Demonstration. Isaac owns Block 1, Ben owns Block 2, AJ owns Block 3, Ryan owns Block 4.



The wearable device, Block 1 and Block 2, includes the force transducers and inertial measurement unit. We decided to use a 9V battery to power the wearable system because the source is small, compact, and easily replaceable. The battery provides 9V DC, which will be sufficient enough to power the system. Two force transducers are connected to the microcontroller, one for the user's index and one for the user's middle finger. The microcontroller has seven analog pins, which is plenty for completing our functionality. The IMU is a digital signal that is passed from the electrical component and Bluetooth.

Block 3, Bluetooth, is used to communicate between two microcontrollers and share data. The Arduino Nano 33 BLE transmits the static roll of the microcontroller and voltage values from the transducers.The Raspberry Pi receives Bluetooth packages, unpacks the data, maintains the control algorithm, and hosts the demonstration. Bluetooth is powered via the Arduino Nano and transmits a digital signal.

Block 3, Demonstration, contains the display for the video game, and the Raspberry Pi. The Raspberry Pi will receive and unpack the data from the wearable device. To power the Raspberry Pi, we decided to use a wall outlet since we do not want to worry about the Raspberry Pi losing battery. We decided to use a Raspberry Pi compatible LCD Screen to display our video game because it is easily able to connect and work with the Raspberry Pi. A Raspberry Pi outlet converter will plug into a NEMA 5-15 wall outlet. The Pi can output +5 voltage, which will be used to power the LCD Screen. The demonstration has 2 main pieces of software that run on the hardware. There is the middleware, which collects the Bluetooth data and converts it to keyboard commands, and the video game, which is what the user plays when interacting with our device.

Section 2.9: Block Diagram Description Table

To go along with our detailed block diagram, we created a table that tells what each block does. This includes the block name, Owner of the block, Block Function, Power Interface, and Digital and Analog Inputs and outputs each block has. Below is the table containing more detailed information about the specifics of the detailed block diagram.

Block #	Block Name	Owner	Block Function	Power Interface	Digital Interface	Analog Interface
1	Wearable	N. Isaac, B. Ben	This is the wearable device, it collects real world data, translates to digital and sends it to the communication block	IN: +3VDC	Output: Raw data from sensors	Input: Raw data from sensors
2	Bluetooth	F. AJ	Communication between the demonstration device and the wearable device will be handled via Bluetooth. This block handles both sending and receiving on the required device.	Powered by microcontroller	Input: Raw data from sensors Output: Raw Data from sensors	None
3	Demonstration	H. Ryan	Demonstration receives raw data from the wearable and overseas translating it to command inputs, then sent to the video game, which is displayed.	IN: +5VDC	Input: Raw data from wearable Output: Visual display	None

Section 2.10: Detailed Design Analysis

Our project has 3 major components, the wearable system(Block 1 and Block 2), the Bluetooth connection (Block 3) and the Demonstration (Block 4).

An Arduino Nano 33 BLE is the microcontroller of the design because it includes an inertial measurement unit (IMU), Bluetooth, and seven analog pins. The wearable system is powered via a 9V battery source and the force transducers are soldered onto analog pins. Bluetooth is utilized to communicate between devices. The Bluetooth module features a low energy mode which can help the wearable device last longer and send data quicker over short distances. Another key contributor to our decision was the relatively cheap price of the Arduino.

The sensors we chose for our fingers will be force transducers. These are simple to interface. Although these sensors can be replaced by buttons, the crucial difference will be the amount of force a user is using to press on the transducers. This can affect the control functions of the demonstration depicting how much intensity a function can have. For example, if an object in the demo needs to accelerate, we can make it proportional to the amount of force the user is pressing on the sensor.

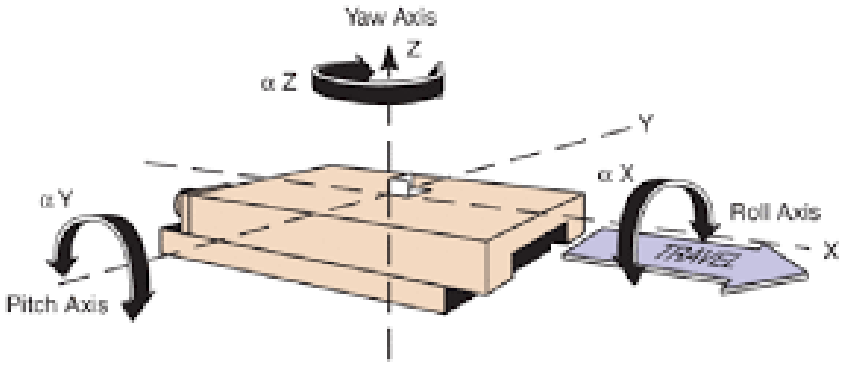
We wanted our demonstration to be a game. This game has the purpose of demonstrating what our wearable device can do. We chose to break up our demonstration into two large portions: a processing unit, and the display. Since we are running a game and we want something small and lightweight, the Raspberry Pi (RPi) fits both criteria. We can use the built-in Bluetooth to connect to the wearable device and process the data packages that are sent over from it. That way we can interface that data into the video game program on the RPi. Since RPi is a well-known brand, there is a display that is compatible with most RPi's. This is the RPi 7" Touch screen. To avoid compatibility issues, we chose this display. Ideally, we would want a slightly bigger one, but this is the biggest they have. This also helps with portability. We created our video game in house to ensure compatibility and to be able to change the level design.

Section 3: Inertial Measurement Unit (IMU) - Ben

This section covers how the IMU was utilized to collect data about the acceleration and angular rotation of the microcontroller, or wearable system. An Inertial Measurement Unit (IMU) is an electronic device that measures and reports an object's orientation, velocity, and gravitational forces using a combination of accelerometers, gyroscopes, and sometimes magnetometers. Using the Arduino_LSM9DS1 library, I analyzed the raw data. This included {x, y, z} values for the accelerometer, gyroscope, and magnetometer.

The terms for describing an object's orientation in 3D space are called yaw, pitch, and roll. Pitch is the rotation of an object about its lateral axis (an axis running from side to side). Positive pitch indicates that the object is moving upwards and negative pitch indicates that it is moving downwards. Roll is the rotation of an object about its longitudinal axis (an axis running from front to back). It describes the tilting or leaning movement of an object, with positive roll indicating that the object is tilting to the right and negative roll indicating that it is tilting to the left. Yaw is the rotation of an object about its vertical axis (an axis running from top to bottom). The figure below represents this analysis.

GitHub branch: https://github.com/aj-funari/capstone_ISE/tree/data_collection



For the scope of our project, we only need the accelerometer to calculate the static roll of the microcontroller. Pitch and yaw were not used since the video game only relies on 1 dimension. This resulted in the reduction of complexity for our system as well as more available processing power due to less resources on the controller being used.

Section 3.1: IMU Initialization

The Arduino LSM9DS1 library allows us to use the Arduino Nano 33 BLE IMU module without having to go into complicated programming. The library takes care of the sensor initialization and sets the values of the Accelerometer's range at [-4, +4]g -/+0.122 mg and its output is set to a data rate of 104 Hz. To include the IMU initialization, the following lines are added to our programming script. Once initialized, the accelerometer can be accessed to read its {x, y, z} axis values.

```
#include <Arduino_LSM9DS1.h>
```

```
void setup() {
  Serial.begin(9600);

  // begin IMU initialization
  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while(1);
  }
```

Section 3.2: Reading IMU Data

An accelerometer is an electromechanical device used to measure acceleration forces. Such forces may be static, like the continuous force of gravity or, as is the case with many mobile devices, dynamic to sense movement or vibrations. The accelerometer's {x, y, z} axes were used for orientation calculations. These axes refer to the three orthogonal axes of a Cartesian coordinate system. The X-axis is perpendicular to the direction of gravity when the accelerometer is at rest on a flat surface. The Y-axis is parallel to the direction of gravity when the accelerometer is at rest on a flat surface. The z-axis is aligned with the depth of the accelerometer. When the accelerometer is in motion, it detects changes in acceleration along these three axes. For example, if an accelerometer is moving up and down (in the direction of the y-axis), it will detect positive acceleration along the y-axis when moving up and negative acceleration along the y-axis when moving down. If the accelerometer is tilted to the side along the x-axis, it will detect positive acceleration along the x-axis in the direction of the tilt and negative acceleration along the x-axis in the opposite direction of the tilt. The same principle applies for movement along the z-axis. By measuring the acceleration along these three axes, an accelerometer can determine the direction and magnitude of the forces acting on it, including the force of gravity. This information can be used to determine the orientation and movement of the device in three-dimensional space.

Below is the set up for setting the X, Y, and Z axes that will be used.

```
// varaibles for roll and pitch calculation
float accelX,           accelY,           accelZ,          // units m/s/s i.e. accelZ if often 9.8 (gravity)
      gyroX,            gyroY,            gyroZ,           // units dps (degrees per second)
      pitch, roll;
```

The image below is the function used to update the values of the axes and a description of the variables.

```
bool readIMU() {
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable() ) {
    IMU.readAcceleration(accelX, accelY, accelZ);
    IMU.readGyroscope(gyroX, gyroY, gyroZ);
    return true;  // return true if IMU is available
  }
  return false;  // return false if IMU is unavailable
}
```

`IMU.accelerationSampleRate()` - reads the sampling rate in Hz.

`IMU.accelerationAvailable()` - checks if there's data available from the IMU.

`IMU.readAcceleration(Ax, Ay, Az)` - reads the accelerometer, and returns the value of the **x** **y** and **z** axis.

`IMU.gyroscopeSampleRate()` - reads the sampling rate in Hz.

`IMU.gyroscopeAvailable()` - checks if there's data available from the IMU.

`IMU.readGyroscope(Gx, Gy, Gz)` - reads the accelerometer, and returns the value of the **x** **y** and **z** axis.

Section 3.3: Calculating Roll

The code below is the main part of the program (void loop) where the readIMU() function is called continuously. Every function call, new {x, y, z} values are used to compute the pitch and roll. Depending on the orientation of the microcontroller inside the closure, either the roll or pitch degree will be sent over Bluetooth.

```
void loop() {
  // poll for Bluetooth® Low Energy events
  BLE.poll();

  // calculate roll
  if (readIMU()) {
    roll = atan2(-accelY, sqrt(accelX * accelX + accelZ * accelZ)) * 180 / M_PI;
    pitch = atan2(-accelX, sqrt(accelY * accelY + accelZ * accelZ)) * 180 / M_PI;
    Serial.print("roll: ");
    Serial.print(roll);
    Serial.print(" ");
    Serial.print("pitch: ");
    Serial.println(pitch);
  }
```

The process of computing Roll

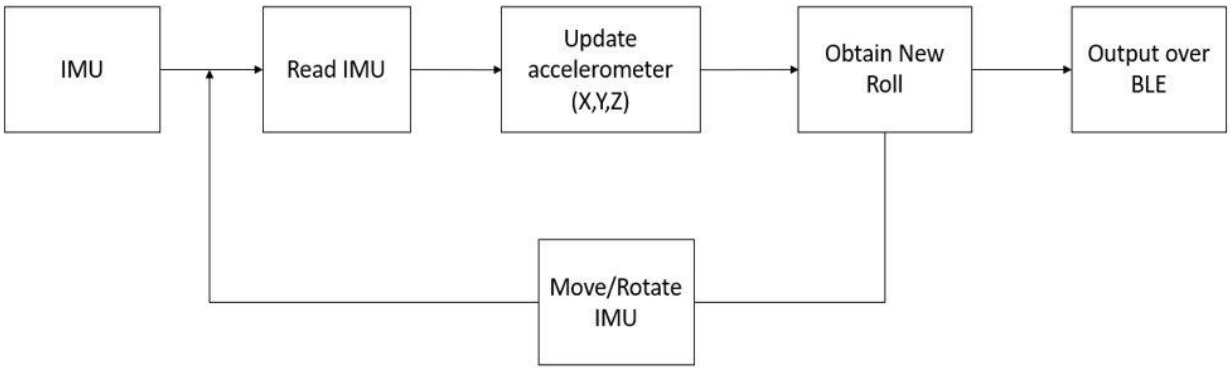
sqrt(accelY * accelY + accelZ* accelZ) calculates the magnitude of the x and z components of the acceleration vector using the Pythagorean theorem. This value represents the total acceleration in the x-z plane, which is perpendicular to the y-axis.

atan2(-accelY, ...) calculates the angle between the x-axis and the y-z plane. The first argument **(-accelY, ...)** is the y-component of the acceleration vector, but it is negated because we want the angle in the opposite direction (i.e., we want the roll angle to be positive when the device is tilted to the right). The second argument is the total acceleration in the x-z plane.

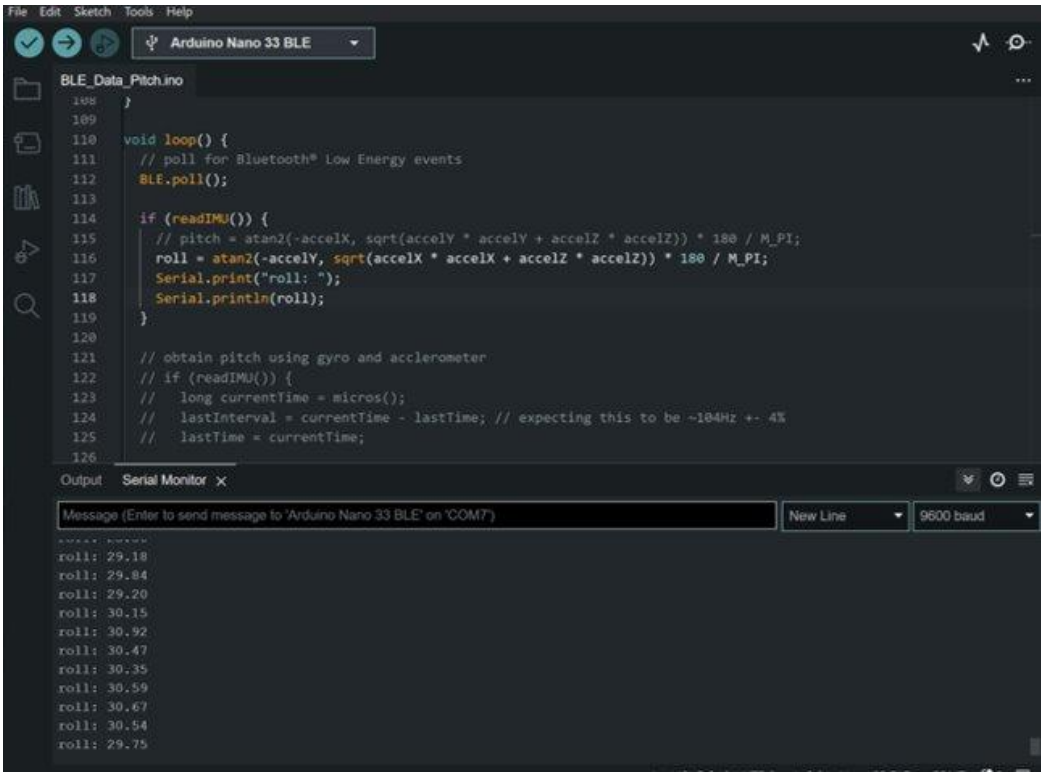
*** 180.0 / PI** converts the result from radians to degrees. The constant 'PI' is defined in the math.h library as the value of pi, which is approximately 3.141592. Multiplying by 180.0 converts radians to degrees, since there are 180 degrees in pi radians.

Overall, this calculation gives the roll angle in degrees, with a range of -90 to 90 degrees. When the device is level, the pitch angle is 0 degrees. When the device is tilted right, the roll angle is positive, and when it is tilted left, the roll angle is negative. The computation for pitch is the same as roll but accelY and accelX are switched because pitch takes a different plane into account.

Section 3.4: Passing Data to Bluetooth



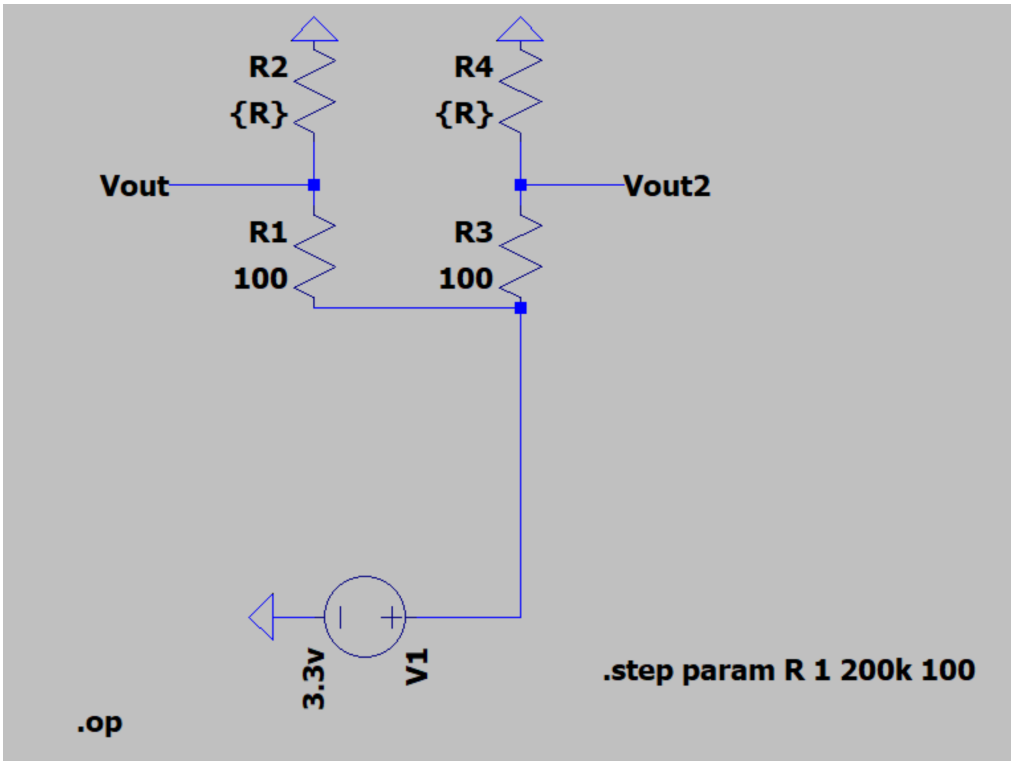
The block diagram above illustrates the entire process of reading IMU to update the {x, y, z} axes of the accelerometer with its current acceleration in order to calculate a new roll that would be used by the Bluetooth to send the hand orientations information to the Raspberry Pi. Since the loop() function in Arduino contains all the code for the IMU, it runs repeatedly in real time. As the user tilts their hand (move/rotating the IMU), the IMU will have new data to update and continuously transmit. Below is the figure of everything working together with roll data printing to the Serial Monitor. This data is passed to Bluetooth.



Section 4: Force Transducers - Isaac

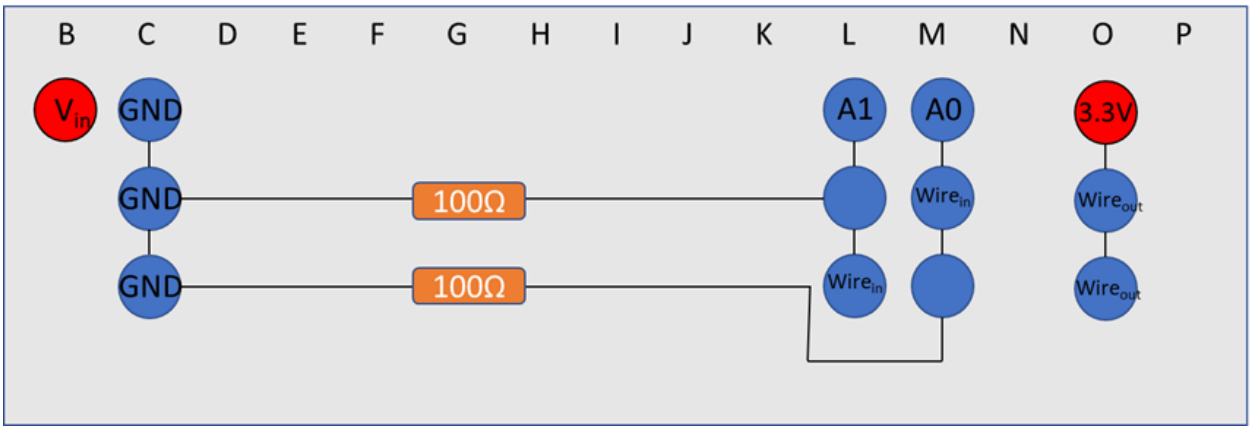
The force transducers have a very simple interface, it is just a variable resistor. What we have to do is read the voltage across each resistor to determine if it is being pressed. Here is a schematic built using LTSpice.

GitHub branch: https://github.com/aj-funari/capstone_ISE/tree/data_collection



The power supply is the microcontroller, and the force transducers are the variable resistors R2 and R4. In this schematic V_{out} and V_{out2} are the two points where we will be reading in data.

After making sure that the schematic simulation didn't cause any adverse effects, we started to prototype the circuit on a PCB prototyping board. This diagram shows exactly how it was connected all together.


B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
V_{in}
GND
GND
GND
100Ω
100Ω
A1
A0
3.3V
$Wire_{in}$
$Wire_{out}$
$Wire_{in}$
$Wire_{out}$

Section 5: Battery - Isaac

Section 5.1: Battery Constraints

We want our wearable device to be independent from the whole demonstration. So, to power it we wanted to use a battery. To figure out what battery would meet our requirements we looked at the voltage needed to power the microcontroller and the standard rating for each battery type. The voltage needed was simply written on the box of the arduino nano, where it could range from 4.5V-21V. The standard rating is based on how many amps can be drawn from a battery over a set amount of time, measured in Amp-hours (Ah). For example, a 100Ah rated battery could draw 1 Amp for 100 hours, or it could theoretically draw 100 Amps for 1 hour.

After uploading our final script to the microcontroller, we measured the *current draw* at 0.03 Amps. We want our wearable to last at least 12 hours. So using the following formula:

$$Hours = \frac{AmpHours}{Current\ Draw}$$

Then rearranging to solve for the minimum needed rating:

$$AmpHours = CurrentDraw \cdot Hours$$

$$AmpHours = 0.03\,A \cdot 12\,h = 0.36\,Ah$$

We got a rating of **0.36 Ah** needed to meet our goal.So our two constraints was finding a battery with a voltage between 4.5V-21V and an estimated rating greater than 0.36 Ah.

Section 5.2: Choosing a Battery

Our search consisted of finding the smallest sized battery with these specifications. We started with a coin cell battery, where we quickly found out that it did meet our rating requirements (estimated at 0.5Ah), but the maximum voltage we could find was at 3V. Our next option was using a lithium ion polymer battery (LiPo). The max rating we found was at 2Ah, which would have exceeded our goal tremendously, but the maximum voltage we found was at 3.7V. Since LiPo batteries are rechargeable, we looked for a boost converter that could boost the voltage to 5V, and keep the rating above 0.36 Ah. We found a boost converter called “PowerBoost 1000 Charger” on Adafruit, which would boost our LiPo voltage to 5V and has the ability to still draw over 1 Amp if needed, along with the ability to recharge our battery with a microUSB. This would have been perfect, but unfortunately it was out of stock on Adafruit and our attempt at purchasing it from China took too long to deliver. So we went with our third option: a 9V Alkaline battery. It met our rating requirement coming in with 0.5 Ah. With this rating we would expect ~16 hours of battery life in our wearable device before we would have to replace it.

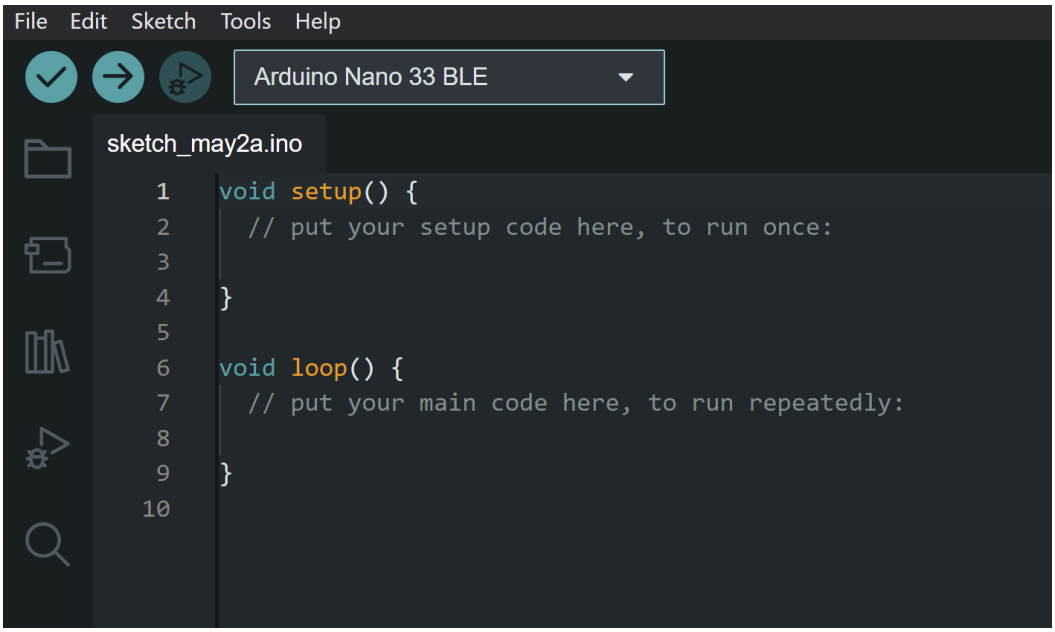
Section 6: Bluetooth Low Energy - AJ

Bluetooth Low Energy, BLE, relies on short-range radio frequency to communicate and share data with other compatible Bluetooth devices. The purpose of Bluetooth for our project is to send Inertial Measurement Unit, or IMU, data and force data between two devices. Data will be transmitted from an Arduino Nano to a Raspberry Pi. This document will walk you through the steps of setting up the Arduino and Raspberry Pi, understanding the GATT profile, writing data to characteristics, connecting devices, testing functionality, reading data from characteristics. Incorporate Bluetooth into the wearable device turns the system into a wireless one-handed controller. When the Arduino is powered via a battery, the user can walk within 10 meters, maintain connection, and continue interacting with the video game.

GitHub Branch: https://github.com/aj-funari/capstone_ISE/tree/bluetooth

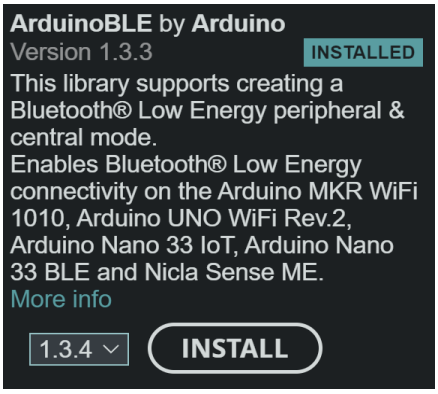
Section 6.1: Setting up Arduino

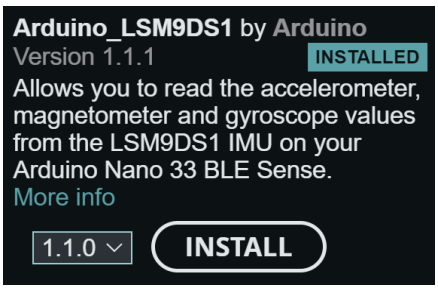
Install Arduino IDE 2.0.3



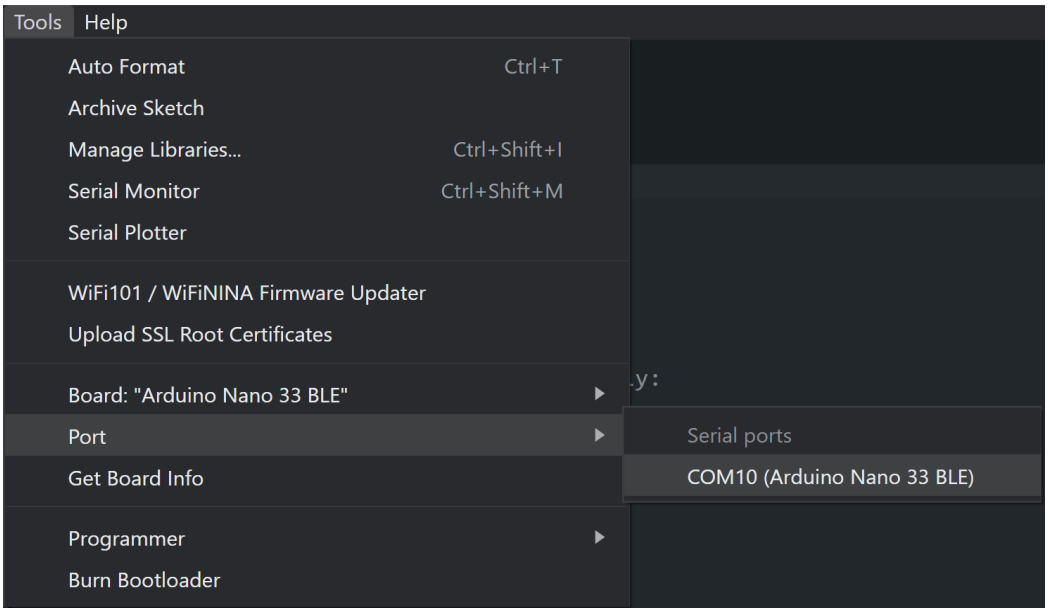
Install the following libraries:

The Arduino AVR Boards package is necessary to run Arduino code. ArduinoBLE is needed to use Arduino's Bluetooth stack. Arduino_LSM9DS1 is the library associated with the IMU.





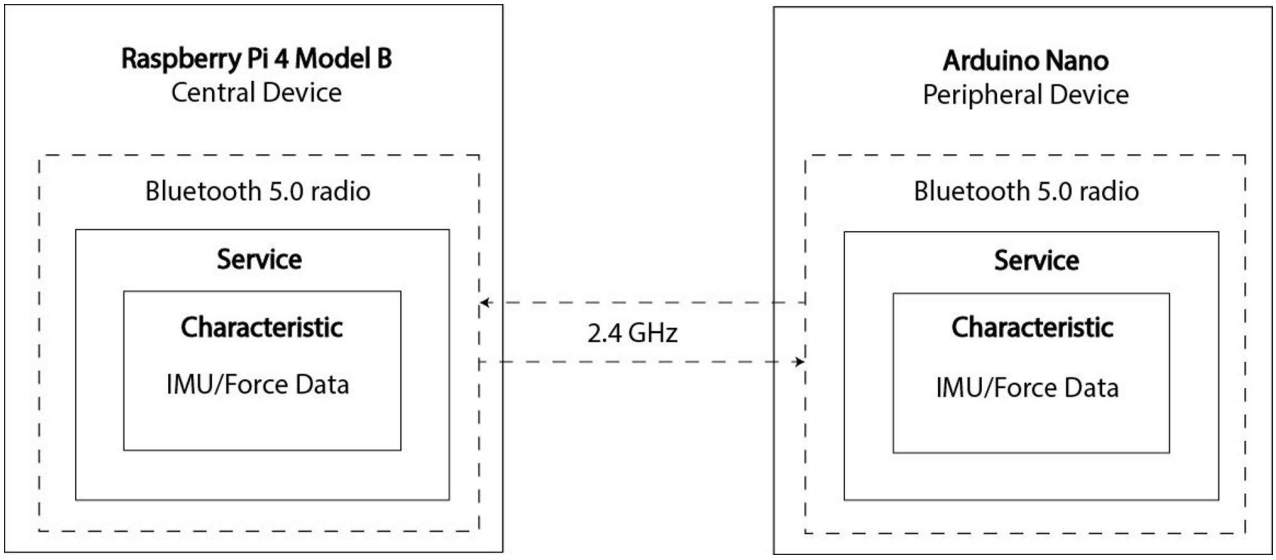
Connect to the Arduino:



Include statements associated with installed libraries:

```
#include <ArduinoBLE.h>
#include <Arduino_LSM9DS1.h>
```

Section 6.2: GATT Profile



General Attribute Profile (GATT) represents the structure of the Bluetooth stack. Each microcontroller, the Arduino Nano and Raspberry Pi have Bluetooth peripherals using Bluetooth 5.0 radio. Once the Arduino Bluetooth library is imported, you have full control to build a custom Bluetooth service. Every Bluetooth service created must include a 16-bit or 32-bit unique identification code called UUID. This code is passed to both microcontrollers to create a seamless connection.

```
// Bluetooth service
BLEService dataService("19B10000-E8F2-537E-4F6C-D104768A1214"); // create service
```

While the service is used to create a Bluetooth connection, characteristics are used to transfer data between two devices. A character is created by passing the UUID and potential extra parameters. These parameters are not exclusive to but include BLERead, BLEWrite, and BLENotify. BLERead allows the developer to read values from the characteristic. BLEWrite allows the developer to write values to the characteristic. BLENotify allows the developer to continuously transmit values through the Bluetooth pipeline. The read and write functionalities are purposed for single purpose tasks. Since our project intends to continuously transmit and read IMU and force data, it is important to provide BLENotify and BLERead. Declaration of our characteristic is shown in the image below.

```
// create gyro characteristic and allow remote device to read and notify
BLECharacteristic dataCharacteristic("19B10001-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify, 50);
```

Build the Bluetooth function schematic with the following steps:

1. Set the local name of the peripheral advertises
2. Set the UUID for the service
3. Add the characteristic to the Bluetooth service
4. Add the service to the Bluetooth pipeline

To complete the tasks listed above, use the following code:

```
// set the local name peripheral advertises
BLE.setLocalName("ArduinoData");

// set the UUID for the service
BLE.setAdvertisedService(dataService);

// add characteristic to Bluetooth service;
dataService.addCharacteristic(dataCharacteristic);

// add service
BLE.addService(dataService);
```

Section 6.3: Writing Data

The video game demonstration relies on three values: the roll angle of the IMU and two force values. It is feasible to create three unique characteristics for each float value. This would cause more latency from sending and receiving data than is necessary. This would cause more latency from sending and receiving data than is necessary. I came to the conclusion that it would be more efficient to copy three floats into one string, then pass that value through the Bluetooth pipeline. Only one Bluetooth characteristic is necessary for this solution; however, extra parsing is required before the control algorithm. The tradeoff between number of characteristics and extra parsing results in the wearable being able to send out data at the same speed, but the Raspberry Pi has to do slightly more processing to get usable data. The wearable is capable of transmitting data at the same speed, but the Raspberry Pi has to do slightly more processing.

```
// Pass data to characteristic using array
char buffer[20];  // maximim string length: "-xxx xxxx xxxx\0"
sprintf(buffer, "%.0f %.0f %.0f", roll, voltageOne, voltageTwo);
```

Finally, data is written to the Bluetooth characteristic using the following code.

```
// write data to characteristic
dataCharacteristic.writeValue(buffer);
```

Section 6.4: Connecting Devices

Once a connection is created between devices, the central can read from the characteristic and write to the characteristic. Begin a Bluetooth instance with the following code:

```
// begin BLE initialization
if (!BLE.begin()) {
  Serial.println("starting Bluetooth® Low Energy module failed!");
  while (1);
}
```

Connecting devices is the action of sharing specific information between two Bluetooth peripherals to create a successful connection. The device broadcasting advertisement packets (GAP) is called the peripheral and the device scanning for packets is called the central. In our situation, the Arduino Nano 33 is the peripheral and the Raspberry Pi is the central. The peripheral transmits the desired pitch of the IMU and forces data as a string. Start advertising using the following code:

```
// start advertising
BLE.advertise();
```

To create a connection between the two devices, there are two event handlers to handle connections and disconnections. The following code is below:

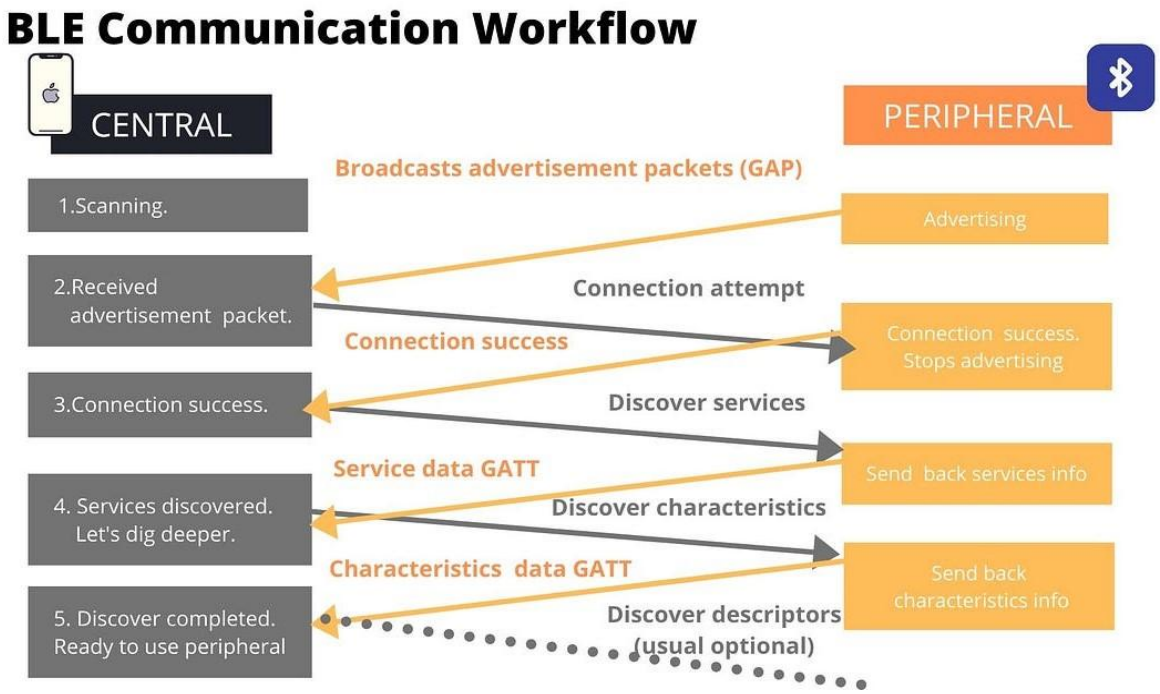
```
// central connected event handler
void blePeripheralConnectHandler(BLEDevice central) {
  Serial.print("Connected event, central: ");
  Serial.println(central.address());
}

// central disconnected event handler
void blePeripheralDisconnectHandler(BLEDevice central) {
  Serial.print("Disconnected event, central: ");
  Serial.println(central.address());
}
```

Inside the main Arduino loop, include the following poll to continuously check for connections and disconnections:

```
void loop() {
  // poll for Bluetooth® Low Energy events
  BLE.poll();
```

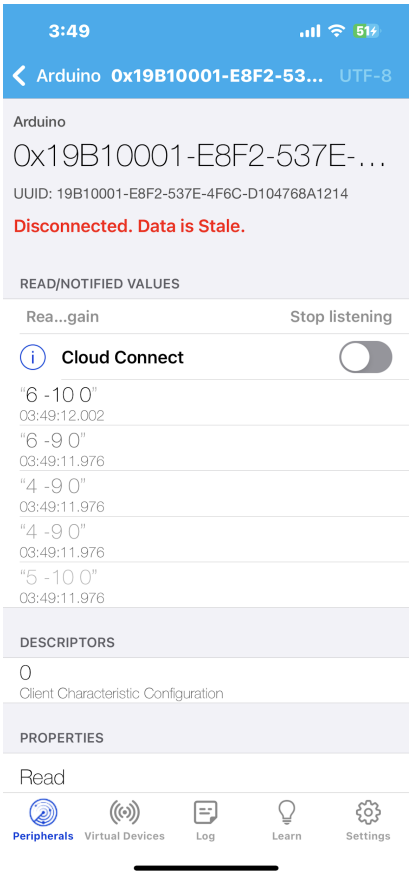
Below is an image that represents the communication between the peripheral and central devices.



Light Blue Application

The Light Blue Application is an iPhone application that can be used to test functionality of connecting with advertising peripherals, reading data from characteristics, and writing data characteristics. Follow these steps to connect with the Arduino:

1. Run the Arduino script in the IDE
2. Click on the peripheral name
    a. The string passed to BLE.setLocalName
3. Click on the characteristic UUID
4. Change the data format from Hex to String



The Light Blue Application was very useful for testing if the Arduino was capable of communicating with another device. During the process of creating a prototype, we added a switch to our circuit. When the Arduino was not plugged into a computer and connected to the Arduino Serial Monitor, testing functionality with this application was critical.

Section 6.5: Setting up the Raspberry Pi

The Raspberry Pi B 4 comes installed with Bluetooth 5.0. The main command for configuring Bluetooth devices on Linux is using the built in protocol bluetoothctl; however, since our project requires the ability to read and write from service characteristics, we needed a compatible Bluetooth python library equipped with GATT profiles. After digging around the internet, I found a library called bluepy that when installed gave full functionality for Bluetooth connections. The image below shows the library successfully imported into a python script.

```
from bluepy import btle
```

Connecting to Arduino over Bluetooth

```
# Must match with Peripheral
#AJ Arduino MAC a8:8f:15:63:27:d2
#Actual Arduino MAC 66:46:6d:83:a5:c8
MAC = "66:46:6d:83:a5:c8"
SERVICE_UUID = "19B10000-E8F2-537E-4F6C-D104768A1214"
CHARACTERISTIC_UUID = "19B10001-E8F2-537E-4F6C-D104768A1214"
```

The image above represents the Bluetooth MAC address of the Arduino and both UUIDs associated with the GATT profile. Determining the Bluetooth MAC address of the microcontroller can be achieved by running the following code in the Arduino IDE:

```
// Serial.print("Peripheral MAC address: ");
// Serial.println(BLE.address());
```

Building the bridge between devices is fairly simple. Create an instance of the Bluetooth connection by passing the associated Bluetooth MAC address of the Arduino to the bluepy function.

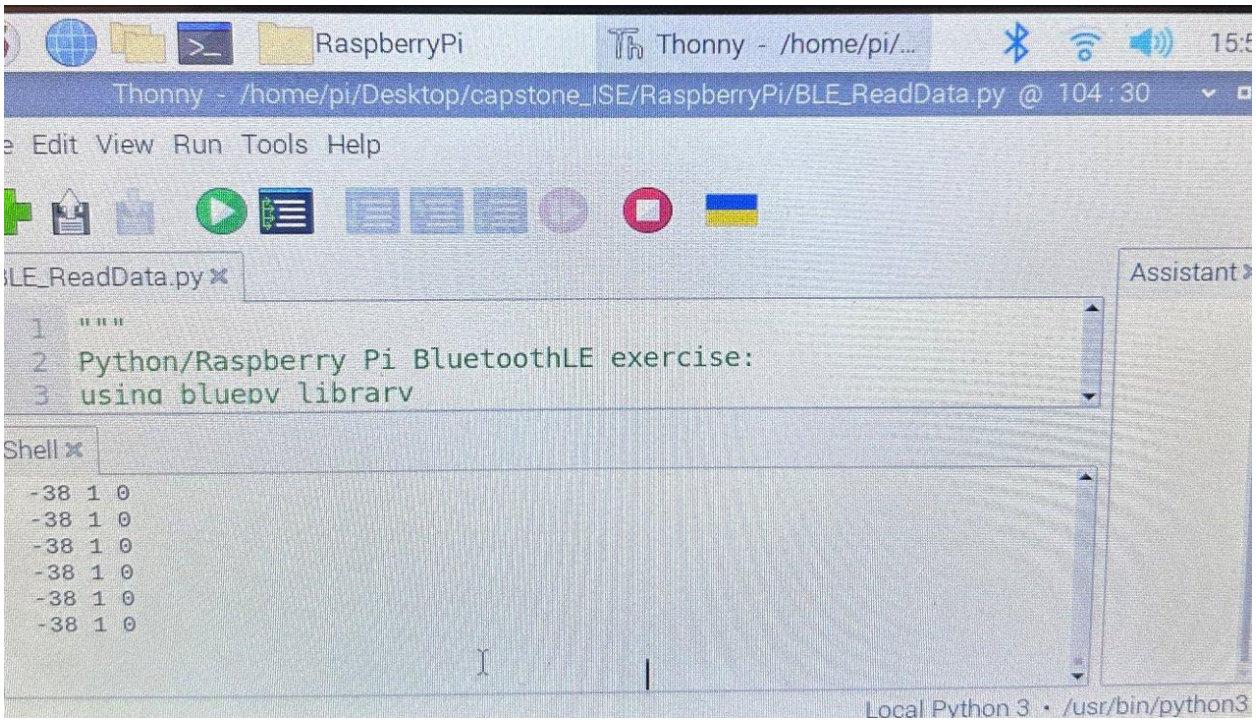
```
# Connect to Peripheral
print("Connect to:" + MAC)
dev = btle.Peripheral(MAC)
print("\n--- dev ----------------------------")
print(type(dev))
print(dev)
```

Reading Data

Reading data from characteristics is also fairly trivial. Once connected to the Bluetooth service, the developer can continuously print the data string using the following code:

```
while(dev):
    for char in characteristics:
        if(char.uuid == CHARACTERISTIC_UUID ):
            # (roll) (jump) (reset)
            data = (char.read()).decode("utf-8")
```

The string is formatted as roll located in the first position, voltage value one located in the second position, and voltage value two location in the third position. The first voltage value correlates to the action jump in the video game and the second voltage value correlates to reset in the video game. This data is passed to the control algorithm, which moves the generated video game player.



Section 7: Demonstration - Ryan

The demonstration for our project refers to the section that the user looks at while using the wearable device. This section is responsible for parsing the data sent from the Bluetooth block, sending the parsed data into a control loop to determine appropriate keyboard commands, and running the video game where the user sees their actions translated to character movement. For these functions, the middleware covers parsing, the control loop, and generating keyboard commands. The video game is solely responsible for creating an environment that the user can easily understand and controlling the character within the environment. Lastly, this section is also responsible for the startup scripts that are on the demonstration device that launch both the video game and middleware.

GitHub branch: https://github.com/aj-funari/capstone_ISE/tree/demonstration

Section 7.1: Hardware

For our specific implementation of the demonstration piece, we chose to have a custom made video game for the wearable to control. This video game requires a computer for it to run separately from the wearable. The computer this runs on must include enough processing power to run the video game, and also run the middleware that is responsible for collecting the data from the Bluetooth Block, parsing it, then sending keyboard inputs to the video game.

For our purposes, a Raspberry Pi 4 Model B was sufficient and up to the task. We chose this specifically because it has plenty of processing power and ram, but it also has Bluetooth capability built into the board. The one thing this device lacks of course is a screen for output. In order to combat this we decided to purchase the Raspberry Pi 7” Touch Screen. We chose this screen in particular because it would be natively supported by the Raspberry Pi and no further development time would need to be spent on connecting a screen to the Raspberry Pi. This screen was also chosen due to its touch screen feature, which allows us to ease the required items to run our demonstration as a mouse and keyboard is not required to come along with the system.

Section 7.2: Middleware

For our project, we determined a layer of code would need to exist between the controller and our video game, as the game engine we were using seemed to lack the ability to create a custom controller like we are doing. Knowing this we set out to create a middleware layer that would parse incoming Bluetooth data, run it through a control loop and generate appropriate keyboard commands based on the received data.

For parsing we use the fact that the Bluetooth data is sent over as a series of character bytes. Knowing this we can easily bring all sent characters into a string. Our data is sent to the middleware layer in a specified and consistent way, such that we can always expect the data to be in the form of gyro data, jump data, reset data. Gyro data is the angle our wearable is currently at, jump data is a 0-1 value to tell us to jump or not and reset data is another 0-1 value to tell us to reset the level or not. All these values are pulled from the Bluetooth data variable and then cast to integers to be used in the control loop.

```
#(pitch) (reset) (jump)
data = (char.read()).decode("utf-8")

# Offset by 2 to bypass the b'
i = 0
#Run through a while loop to get all the pitch data, This is necessary since it is variable in size
pitch = ""
while(data[i] != " "):
    pitch += data[i]
    i += 1
# We now have all of pitch, this is variable however, ranging from a size of 1 to 4
# Now we need to grab the jump and reset button, these can only be size 1
# Offset by 1 as if we don't we'll grab the whitespace
jump = data[i+1]
# NOTE an offset of 2 from the jump, this is to account for whitespace
reset = data[i+3]

# These values are turned into ints
pitch_i = int(pitch)
jump_i = int(jump)
reset_i = int(reset)
```

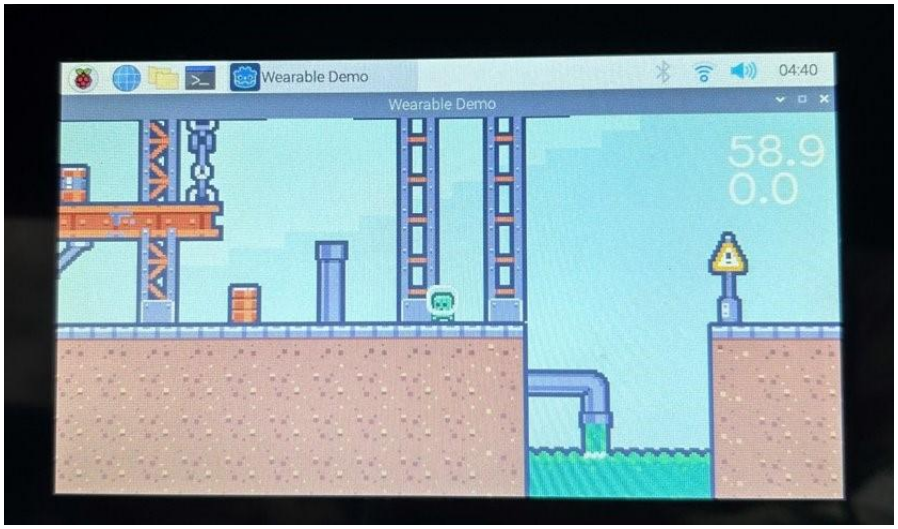
The control loop takes the parsed data that is now in integer form and determines the correct keyboard commands to generate. It does this through if statements; as our game is somewhat basic in controls, it doesn't need any axis inputs which simplify our control scheme. It is important to note that the keyboard library we use should NOT be substituted for another library unless necessary as most other keyboard libraries create a software keyboard to send inputs. This method works excellent for other uses, but video games tend to ignore software keyboards and a library that mimics hardware must be used for this specific purpose. The library we use (pyKey) does this and works well for this intended purpose.

```
#Control loop
if(pitch_i > 20):
    #Use pressKey ONLY for moving laterally
    # You MUST releaseKey if you use pressKey
    pyKey.pressKey("RIGHT")
elif(pitch_i < -20):
    pyKey.pressKey("LEFT")
else:
    pyKey.releaseKey("RIGHT")
    pyKey.releaseKey("LEFT")
if(jump_i > 0):
    #press does a quick "tap" of the key, use for buttons
    pyKey.press("SPACEBAR")
if(reset_i > 0):
    pyKey.press("r")
```

Section 7.3: Video Game

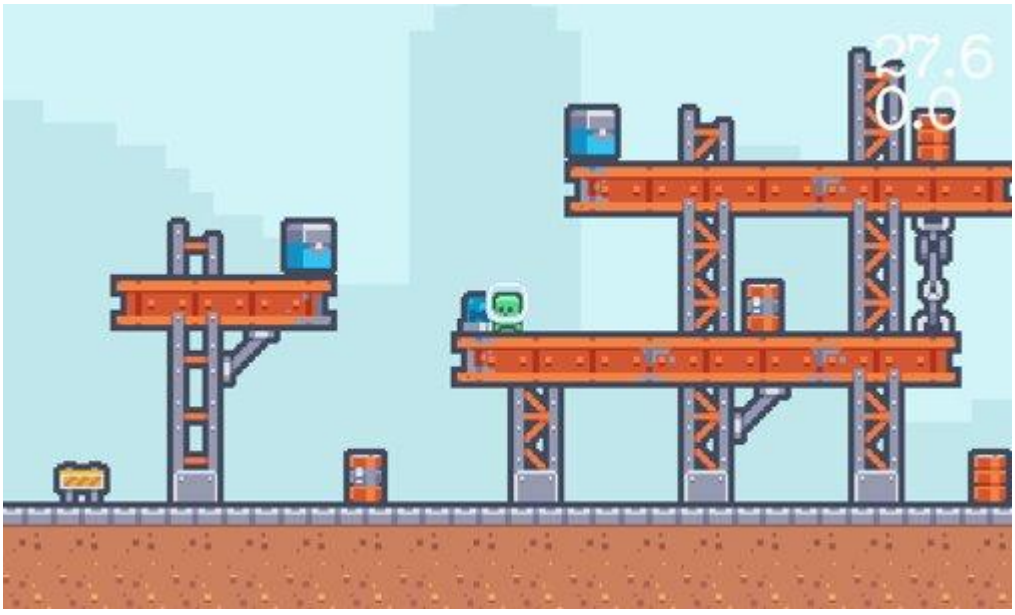
For the demonstration, we created our own video game to suit our specific needs and to be able to adjust level design if needed to better suit our wearable device. The video game we made was created using the game engine Godot version 3.5.1. **It is important to note that GLES2 must be used to export the game, as GLES2 is the correct graphics driver for the Raspberry Pi**. **When exporting, the special export file included in the git repo must be used to configure certain export settings for the Raspberry Pi.**

The video game is a 2D platformer which makes the controls extremely easy to maintain. Middleware will still need to be used, as a way to implement the wearable as a controller through Godot was not found. The middleware is described in Section 7.2. All art assets were acquired from Kenney.nl, which is a website that offers free art assets to be used in video games. We also received consultation on the creation of the video game from Vincent Langlois, a Computer Science major and game design minor at Indiana University. They helped us through the difficulties of starting the project and with various parts of the game as needed. Shown below is a picture of the video game running on the Raspberry Pi.

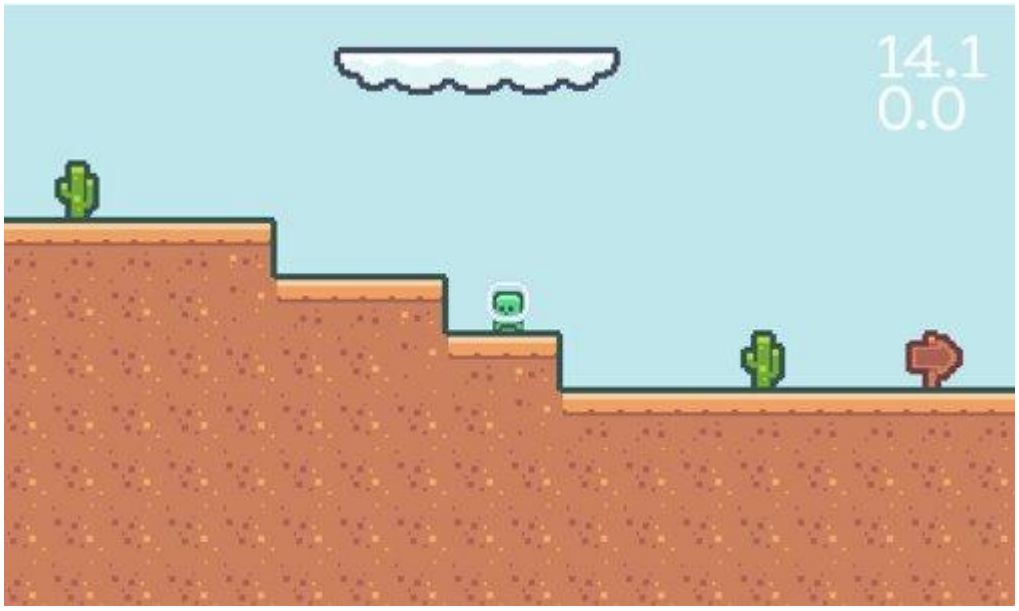


The game is currently controlled using four keyboard inputs. These are left and right arrow keys to respectively move left and right, spacebar to jump, and r to reset the character back to the start of the level. The player is tasked with making it to the end of the level as quickly as possible, with their time being shown in the top right of the screen as they progress through the level. Below this is the best time currently held for the level. This is remembered between level resets, but restarting the game will cause this to go away. This allows for different best times to be achieved at different events, so that the best time better pertains to the skill of the player group rather than everyone who has ever played the demonstration game.

There are currently two versions of the video game on the Raspberry Pi. The first version, shown below, is currently titled as Wearable_Demo1_1.rpi4. This version is played through a nature and industrial themed level. This version offers multiple dangerous obstacles that the player must navigate, or they will be reset back to the start. In this version, the player is not able to jump onto the clouds to ease navigation, as more emphasis is placed on obstacle navigation and skillful avoidance to get a good time. It should be noted that the level design has been tweaked slightly since its first debut, to make certain areas slightly easier to increase completion rate. This is considered our medium difficulty level and should be used for an expected audience range of middle schoolers to young adults. In practice this audience usually has the reflexes to deal with the dangerous obstacles but still be provided enough challenge to keep them interested and competitive with one another.



The second version, shown below, is titled Wearable_Demo_Desert.rpi4. This version is played through a desert themed level. The player has fewer dangerous obstacles to navigate that could cause the player to reset back to the start, with more emphasis being placed on level navigation, and as such is considered to be the easy difficulty level that we currently offer. The player is also offered the ability to jump onto the clouds above the level. This in practice offers more challenge to the player, but rewards a faster time typically since the player spends less time stuck on terrain while traversing the level below. This level should be used for younger audiences, as the level was designed to be somewhat easier to complete with less dangerous obstacles. In practice this audience typically has slower reaction times and benefits from their being less dangerous obstacles. This also allows for higher retention rates as their completion rates will be higher keeping them more interested in beating the best time.



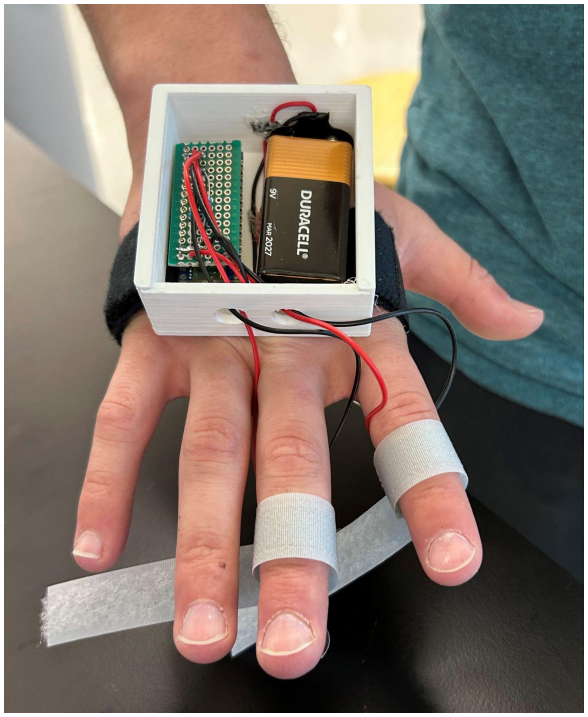
Section 7.4: Startup Scripts

When attempting to start both the video game and middleware, we noticed that this could cause some confusion for involved parties that were not familiar with using a shell terminal. We also concluded that in order to eliminate the need for a mouse and keyboard to be present to launch the required software, that a shell script should be written to automate the launch of both softwares. This script should also be on the Raspberry Pi Desktop to allow for a user to tap on the script to launch all required software to play the demonstration of choice.

There are two scripts on the Raspberry Pi, both of which are located in the bottom left corner of the screen on the Raspberry Pi desktop. These scripts are called easy_start.sh and normal_start.sh. To launch the scripts the user should tap the screen on the script they would like to run. In practice we have found it most easy to launch them by tapping the desired script in rapid succession, as this typically makes the Raspberry Pi want to execute the program rather than rename it. These scripts are titled such that when running easy_start it opens the easier level as described in Section 7.3, and the middleware program that is described in Section 7.2. **It should be noted that when launching these scripts, the user should select the EXECUTE IN TERMINAL option**. This way if the middleware encounters a fatal error it cannot handle, it also closes the video game that is running. This should help the player, as the video game closing indicates that something has gone wrong quicker and the person running the demonstration is able to quickly restart the demonstration rather than the player being unaware for a while that the middleware is no longer running and wondering why their inputs are no longer being accepted.

Section 7.5 Controlling the Video Game with the Wearable

Once the video game and middleware are up and running as described in Section 7.4, demonstration is ready to be played. To control the character, the wearable should be worn on top of the user's hand, with the force transducers placed on the pad of the index and middle finger. Shown below is the proper way to wear the wearable for use.



Now that the wearable is on the user, flip the switch on the rear of the case to turn on the wearable. The switch should be flipped to the left side to power on the wearable device. Once the wearable is on and the middleware is already running, hold the hand as flat as possible during boot to ensure that absolute zero is properly set. After a few seconds the wearable should be booted and transmitting data to the Raspberry Pi. To test this, tilt the hand to either the left or right side and watch the screen to see if the character moves. Once the character is moving, a good connection has been established and the user is ready to play.

To move the character left and right, the user should tilt their hand either to the left or right side, causing the character to move in the direction the user decided to tilt their hand. This action is mapped via the control function to the keyboard's left and right arrow keys should the wearable be inoperable, a keyboard could be plugged into the Raspberry Pi and the video game still played via these keys.

To make the character jump, the user should squeeze their thumb and index finger together, making sure that the force transducer on the pad of their index finger is in between their thumb and index finger. This action is mapped to the keyboard's spacebar and allows the character to jump over obstacles.

To reset the character back to the start of the level, and to reset the current time, the user should squeeze their thumb and middle finger together, again ensuring that the force transducer is between the thumb and middle finger. This action is mapped to the keyboards r key. Below is a diagram of a keyboard, with details on how each key is pressed by the wearable.

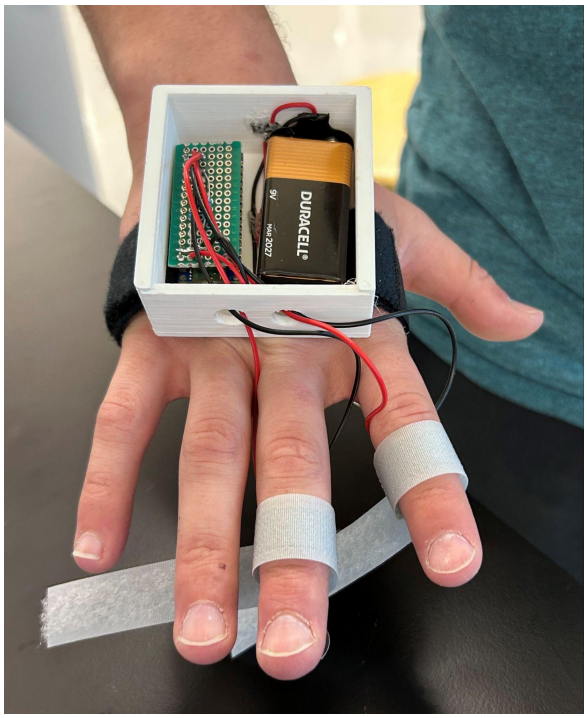
1. Reset the user to start
Squeeze middle finger and thumb together
2. Make the character jump
Squeeze index finger and thumb together
3. Move character left
Tilt hand left
4. Move character right
Tilt hand right

(Keyboard Picture)

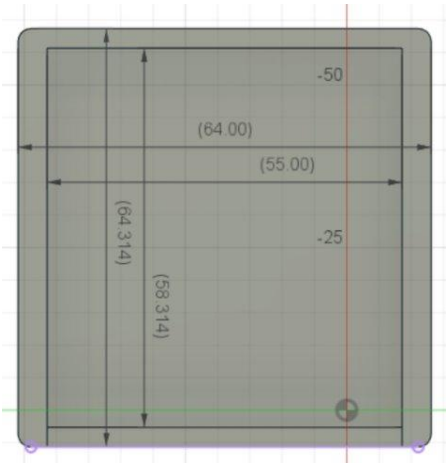
Section 8: Enclosure - AJ

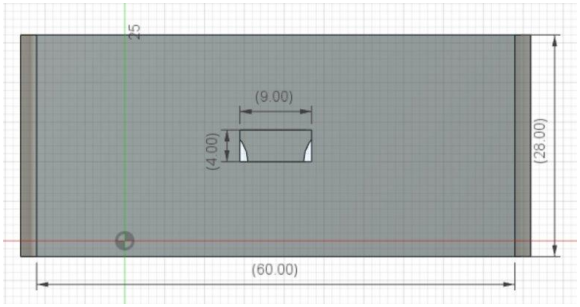
Demonstration

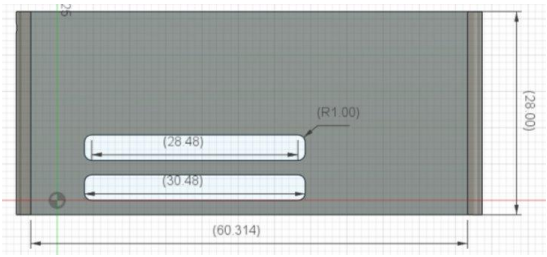
Wearable

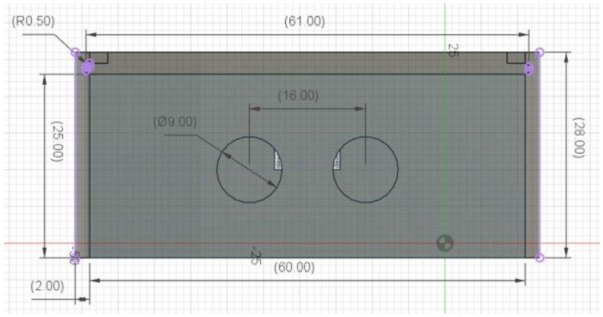


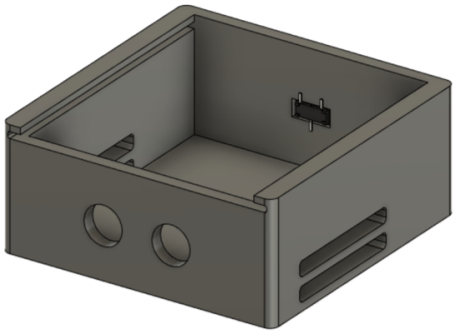
Section 8.1: Wearable Device

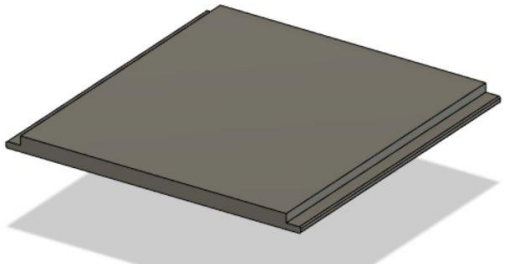




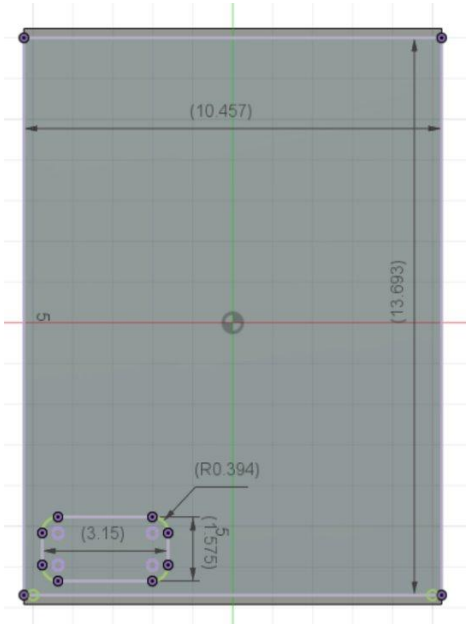


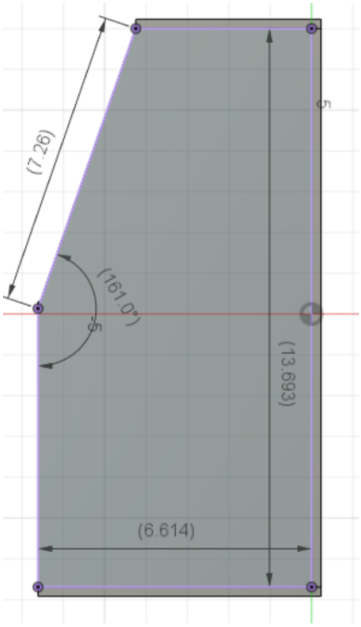


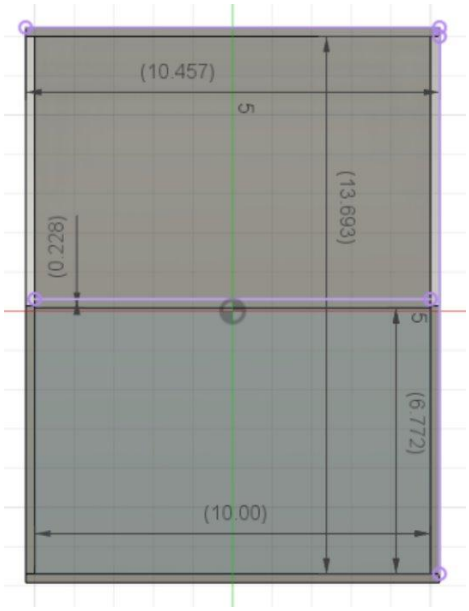


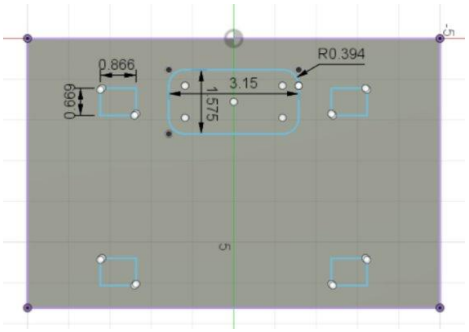


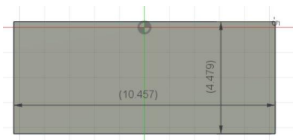
Section 8.2: Demonstration











Section 9: Sustainability - Isaac

Our main concern when it came to the sustainability of our project was battery disposal. Since the alkaline battery is a single use battery we looked into how to properly dispose of them after replacing them. The United States Environmental Protection Agency said that alkaline batteries are commonly disposed of by simply throwing them in the trash. However, it depended on where you lived. So we checked the Bloomington city website and found a “Household Hazardous Waste” page where it gave locations where batteries should be disposed of. Luckily, Luddy has a battery disposal bucket in the fab lab, which all ISE students have access to. So we would strongly suggest that a person hold on to old batteries and throw it away in the bucket.

Maintaining the wearable device is simple. If someone wanted to upload their own scripts to the microcontroller they could simply use the Arduino IDE. Arduino is a very large and well documented platform that allows this feature to be. The same can be said for the Raspberry Pi and the beginner friendly game engine Godot. All of our supplies can be easily replaced, like resistors and wires, which are available in the fab lab, and all other large components, like the arduino nano, raspberry pi LCD screen, and batteries are all easily accessible. Although the case for this project is custom built, it is made out of wood and would take a lot of brute force to break apart. Worst case scenario, someone could 3D print a copy of it to replace it.

Section 10: Future Steps - AJ/Ryan

uMove has high potential for future development! One main issue brought to our attention that was not addressed was latency issues. During play testing, when rotating your hand left or right this moves the video character horizontally and the latency is low. Pressing your thumb and pointer finger together causes the video character to jump. There is noticeable latency with the force transducers that causes the jump to occur. It is unknown the sampling rate at which the Arduino is processing instructions or sending Bluetooth packets. It is also unknown the processing sampling rate of the Raspberry Pi. We did not have time to investigate this issue; however, is it possible to improve the responsiveness of the video game by investigating this and determining where the latency could be coming from.

One aspect of the project we did not implement is using a lithium-ion battery to power the Arduino. This will allow the developer or user to charge the system. The current design uses a 9V battery, which needs to be replaced when the charge is empty. This was mentioned in Section 9 The lithium-ion battery is also significantly smaller, which would shrink the overall size of the wearable. This should allow for a better form factor for the end user, and a more sustainable product. If choosing to work on this specific aspect, our team already purchased and received a lithium-ion battery and a boost converter board has already been purchased but at the time of writing has not been delivered.

One advancement that could be implemented is building a custom printed circuit board. Design an electronic schematic design that consists of an IMU, Bluetooth Low Energy, multiple analog pins, charging pins, and the custom circuit designed for our force transducers. The size of the wearable device can be greatly improved if implemented properly. The footprint of this board should coincide with the size of the lithium-ion battery to further minimize the space the wearable device takes, allowing for a much more comfortable end user experience.

Finally, the wearable device can be easily integrated into separate projects. As long as the communicating device has Bluetooth 5.0, a developer can seamlessly use our wearable system as a control. The one major caveat for our current implementation is the reliance on the middleware program to understand the output from the wearable. So long as the developer is able to get the middleware onto the target or implement some other form of control on the target device, the wearable should work with any Bluetooth 5.0 device. For example, the wearable can be used to fly a drone or drive a car. The limitations are endless!

Section 11: References

IMU details
https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-imu-basics

Arduino_LSM9DS1 library
https://github.com/arduino-libraries/Arduino_LSM6DSOX/blob/master/docs/api.md#readacceleration

Arduino Nano 33 BLE Documentation
https://docs.arduino.cc/resources/datasheets/ABX00030-datasheet.pdf

Bluetooth Central and Peripheral Image
https://medium.com/macoclock/core-bluetooth-ble-swift-d2e7b84ea98e

Arduino to Raspberry Pi Communication Over BLE
https://forum.arduino.cc/t/arduino-to-raspberry-pi-communication-over-ble/874858

Raspberry Pi Bluetooth Library
https://github.com/IanHarvey/bluepy

Keystroke Simulator Library
https://github.com/gauthsvenkat/pyKey

Keyboard picture
https://www.logitech.com/en-us/products/keyboards/k120-usb-standard-computer.920-002478.html

EPA Used Household Batteries
https://www.epa.gov/recycle/used-household-batteries#single

Bloomington Household Hazard Waste
https://bloomington.in.gov/utilities/pretreatment/residential/household-hazardous-waste