CG4002 Embedded System Design Project

August 2020 semester

"FitNests"

Initial Design Report

Group 5	Name	Student #	Sub-Team	Specific Contribution
Member #1	Nicholas Lee	A0168783L	Hw Sensors	Section 2.1.3 Section 3.1 Section 6
Member #2	Lincoln Lim	A0184208E	Hw FPGA	Section 3.2 Section 6
Member #3	Rusdi Haizim B Rahim	A0183139B	Comms Internal	Section 2.1.2 Section 4.1 Section 6
Member #4	Claire Chan	A0187908L	Comms External	Section 4.2 Section 6
Member #5	Wang Jiannan	A0172069B	Sw Machine Learning	Section 5.1 Section 6
Member #6	Umar Bin Moiz	A0180913H	Sw Dashboard	Section 1.3 Section 5.2 Section 6

Content Page

Content Page 2

Section 1 System Functionalities 4

Section 1.1 Use Cases 4

Section 1.2 Feature list 5

Section 1.3 User Story of Target Users: Gym Members 5

Section 2 Overall System Architecture 6

Section 2.1: High Level System Architecture 6

Section 2.1.1 Implementations on Beetle and Ultra96 7

Section 2.1.2 Communication Protocols 8

Section 2.1.3 Hardware Components 9

Section 2.2: Final Form of the System 10

Section 2.3: Main Algorithm for Activity Detection 11

Section 3 Hardware Details 12

Section 3.1: Hardware sensors 12

Section 3.1.1: Components 12

Section 3.1.2 Pin table 17

Section 3.1.3 Schematics 17

Section 3.1.4 Operating voltage level and current 18

Section 3.1.5 Algorithms and libraries 19

Section 3.2: Hardware FPGA 19

Section 3.2.1 Neural Network 19

Section 3.2.2 Mapping and Evaluating 20

Section 3.2.3 Potential Optimization 21

Section 4 Firmware & Communications Details 21

Section 4.1 Internal Communications 21

Section 4.1.1 Task Management on the Beetles 22

Section 4.1.2 Configuration and Set-up of BLE Interfaces 22

Section 4.1.3 Communication protocol between Beetles and Laptop 22

Section 4.1.4 Ensuring Reliability of Data 24

Section 4.2 External Communications 25

Section 4.2.1 Secure communications between laptops, Ultra96 and evaluation and dashboard servers 25

Section 4.2.2 Active processes and scheduling on Ultra96 26
Section 4.2.3 Synchronization delay between users 28

Section 5 Software Details 29
Section 5.1 Software Machine Learning 30
Section 5.1.1 Sensor Data Segmentation 30
Section 5.1.2 Data Pre-processing 31
Section 5.1.2 Features Extraction 32
Section 5.1.3 Machine Learning Model Selection and Optimization 32
Section 5.1.4 ML Model Training and Validation 35
Section 5.1.5 ML Model Testing 37
Section 5.2 Software Dashboard 38
Section 5.2.1 Software Dashboard Design Mockup 38
Section 5.2.2 Storing sensor data 39
Section 5.2.3 Real time streaming 40
Section 5.2.3 User-survey 40

Section 6 Project Management Plan 41

References 45

Section 1 System Functionalities

Section 1.1 Use Cases

For all use cases below, unless specified otherwise, the parties involved are as follows:

- User : The person performing the action
- Laptop: The user's personal laptop used in connection with the wearable device
- Wearable: The wearable device worn by the user
- FPGA: The Ultra96 FPGA (running remotely) that's used to run the machine learning algorithm for classifying actions and positions
- Evaluation server: The server that displays the next action/position for the users to execute
- Dashboard: The user interface running on the web that displays any relevant data regarding each user's movements and positions.

1. **Use Case: Classification of positions and actions performed**
   Main Success Scenario (MSS)
   a. Evaluation server shows the respective positions of each user, along with an action to perform.
   b. Users will perform said transition to a new position and/or perform the action as shown.
   c. Wearable will signal to each user's laptop about the predicted action and repositioning executed (along with data about each user's timestamp).
   d. Laptop does error checking of data and then sends properly structured data to the FPGA.
   e. FPGA runs its classification and sends final data to the dashboard. It also sends the predicted actions and positions of each user to the Evaluation server.
   Use Case Ends

Section 1.2 Feature list

1. Easy-to-use
2. Responsive in detecting movements
3. Classifying each action performed accurately
4. Aesthetically-pleasing user interface (UI) for the dashboard
5. Reliable connections between wearable device and laptop/server

Section 1.3 User Story of Target Users: Gym Members

Priority	As a ...	I want to ...	So that I can …
High	Gym member	Have a clear, user-friendly interface	I can easily use the application when I go to gym
High	Gym member	Purchase a cheap product	Afford it
Medium	Gym member	Low power consumption	Have a long lasting and durable device
High	Gym member	Monitor my movements	Make sure I am doing the exercise correctly
Medium	Gym member	Have a lightweight wearable	Exercise comfortably without obstruction
Low	Gym member	Reference a set of workouts to do before I start	Know which workout to do and perform the move correctly
High	Gym member exercising with friends and/or instructor remotely	See what moves other members are doing at the same time	Be in sync with the other members during my workout even while physically apart
High	Gym member	Get real time feedback on my movements	I can correct any wrong movement during the workout
Low	Gym member	Have a set of workout routines available and being monitored for movement accuracy and duration when doing them	Exercise on my own and rest assured that my exercise is optimized for performance and results
Low	Gym instructor	Have a dashboard to monitor movements	Correct the movements for customers and track dangerous maneuvers

Section 2 Overall System Architecture

Section 2.1: High Level System Architecture

The diagram below overviews how the different end systems interface with one another.

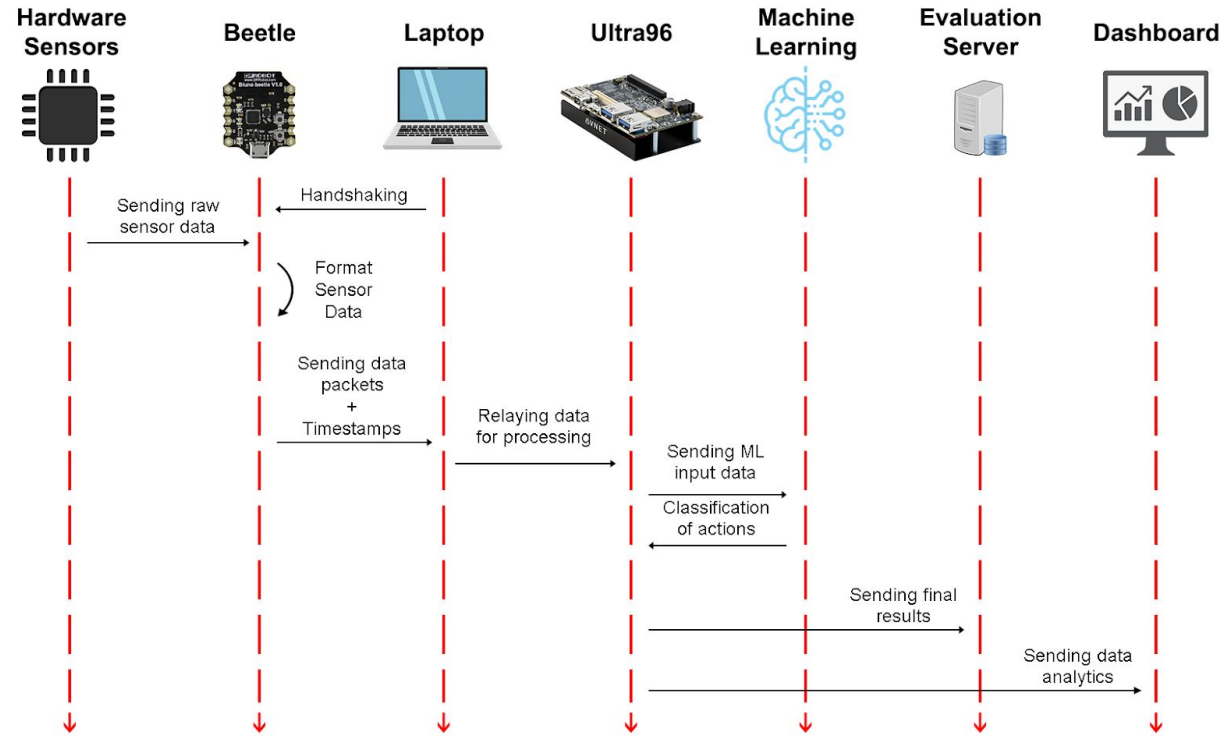

Figure 2.1: High Level Architecture

Section 2.1.1 Implementations on Beetle and Ultra96

The beetle will have three main tasks to carry out:

- SensorReading
- WritingDataToSerial
- ReadingDataFromSerial

The first task will take care of reading the sensor values and storing them into a buffer for data transmission. Once all the required sensor data is read, a flag will be toggled on to signal to the second task that data is ready for transmission. The third task will handle any data sent by the laptop, which mainly concerns the initial handshake stage and any dummy packets sent by the laptop for testing purposes.

Afterwards, the laptop will send the data to Ultra96 for ML processing, which will ultimately be sent to the dashboard as highlighted in the section below.

Section 2.1.2 Communication Protocols

The figure below briefly indicates how the incoming sensor data is handled from the beetle, through the laptop, then the Ultra96, and lastly to both the evaluation server and the dashboard. The in-depth packet details about the formats of the data being sent will be discussed from Sections 4 onwards.

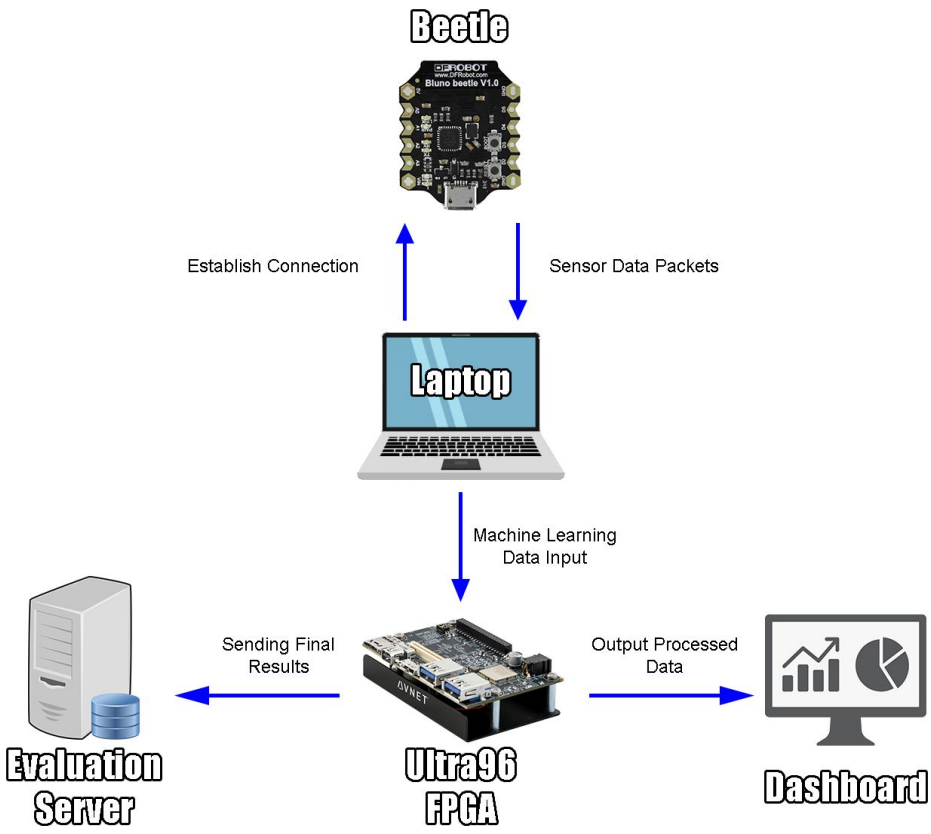

Figure 2.1.2.1: Overall Communication Procedure

The figure below briefly shows how the beetle establishes connection with the laptop, and the subsequent processes that happen thereafter.

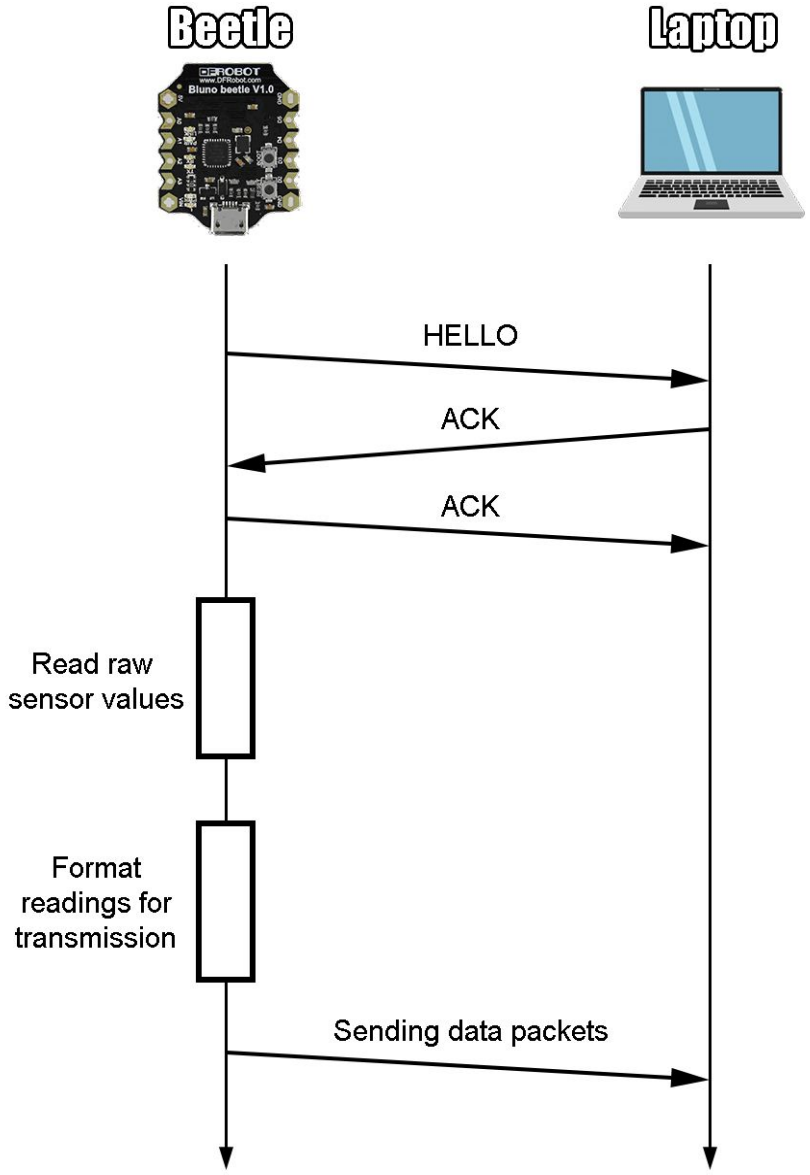

Figure 2.1.2.2: Connection between Beetle and Laptop

Section 2.1.3 Hardware Components

This section provides the overall logic connections for the main hardware components.

The following hardware is used:

- MPU 6050 IMU
- Bluno Beetle V1.1
- Laptop

The overall hardware layout will consist of two accelerometers sending raw data via wired connection to the Bluno Beetle. The Bluno Beetle will process these raw data and convert them into readable values. The Bluno Beetle will then connect to the laptop via bluetooth and send the data over. In depth hardware component descriptions and connections will be elaborated in section 3.

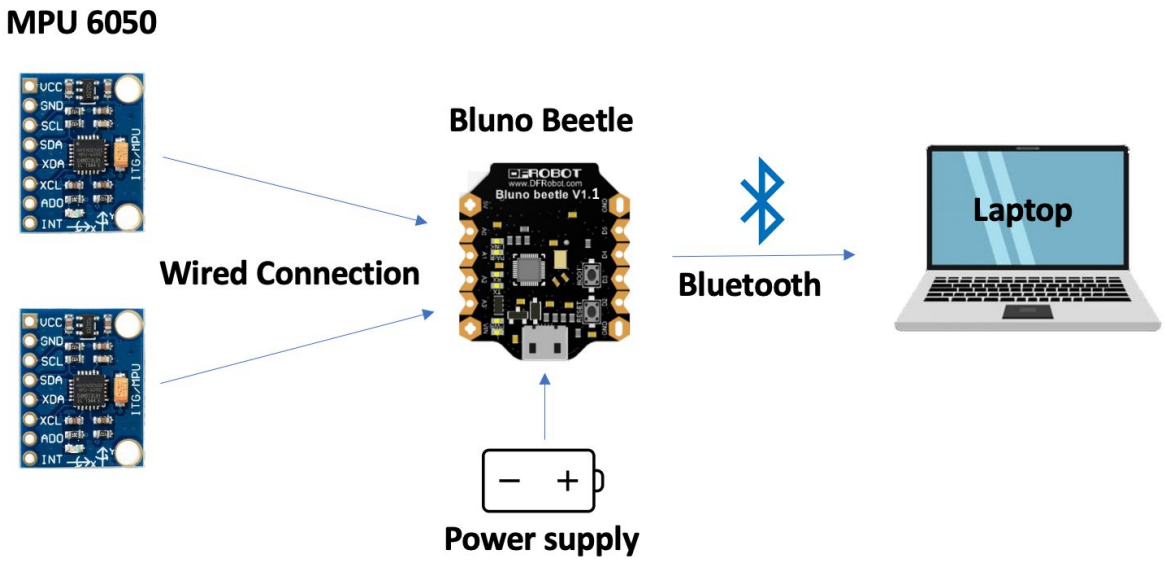

Figure 2.1.3: Connection between Hardware Components

There will be two MPU6050 equipped. One to measure hand movements and one to measure user position. Both MPU6050 will be wired to Bluno Beetle to send relevant data over. Bluno

Beetle is powered by batteries and has bluetooth enabled for connecting and sending data to the laptop.

Section 2.2: Final Form of the System

The figure below illustrates how the MPU sensors and Bluno beetle are going to be integrated on a wearable.

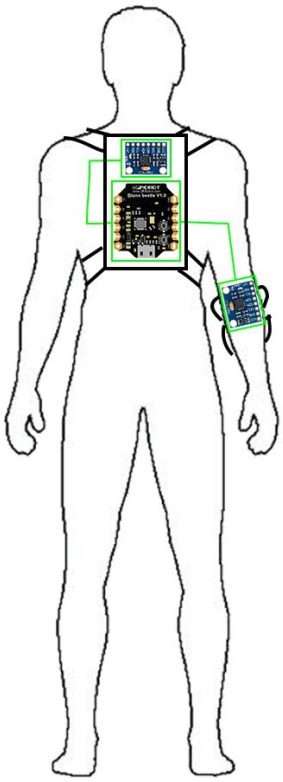

Figure 2.2: Prototype Design of the Final Product

Section 2.3: Main Algorithm for Activity Detection

This section will outline an overview of the key steps performed in our algorithm. They are as follows:

1) Hardware sensors will read body movements and send data to beetle.
2) After initial connection, beetle will send the data read previously as packets to be sent to laptop continuously
3) Laptop receives the data packets and formats it accordingly to be sent to FPGA

4) FPGA will run the Machine Learning algorithm and classify the dance moves and positions accordingly
5) The processed data is then sent to the dashboard from the FPGA for analytics
6) Concurrently, the data is also sent to the evaluation server for logging

Section 3 Hardware Details

Section 3.1: Hardware sensors

The hardware sensors are responsible for all raw data obtained from the wearable device, ensuring proper connection as well as passing data to the laptop wirelessly. This is achieved with accelerometers, gyroscopes and a microcontroller with bluetooth. Apart from the main components, power supply and schematics have been cautiously designed for safety and accurate measures. An in-depth explanation of the components, pin layout, schematics, power supply and libraries with regards to connection will be elaborated in this section.

Section 3.1.1: Components

MPU 6050

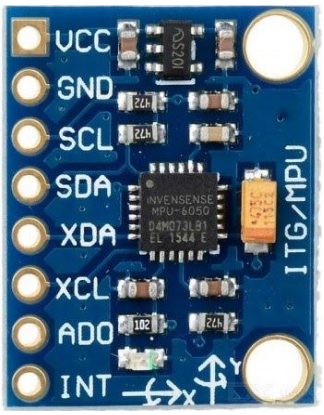

Figure 3.1.1.3: MPU 6050

The MPU6050 is an Inertial Measurement Unit (IMU) which has a 3 axis accelerometer and a 3 axis gyroscope that is integrated on the chip, giving it 6 Degrees Of Freedom (DOF) measurements. It also has an internal temperature sensor and a Digital Motion Processor (DMP)

Accelerometer:

The main component we will be using is the accelerometer. The accelerometer measures acceleration with respect to the change in velocity of an object. It detects acceleration using the 3 dimensional axis (x,y,z).

Gyroscope:

The MPU6050 also has a gyroscope which measures the angular rotation with respect to the X, Y, Z axis the accelerometer detects. The gyroscope consists of 3 sensors, one for each axis. (Roll, Pitch, Yaw) Each of the sensors produces a voltage whenever they are rotated. The voltage is converted using a 16-bit analog to digital converter.

Digital Motion Processor (DMP)

The MPU6050 consists of a DMP. The DMP concept is delivere by Invensense. All data from the accelerometer and gyroscope is sent to the DMP in MPU6050, which then formats it in order for it to be utilised on the I2C bus. The main purpose of the DMP is to process motion data and pass it across other devices.

Pinouts

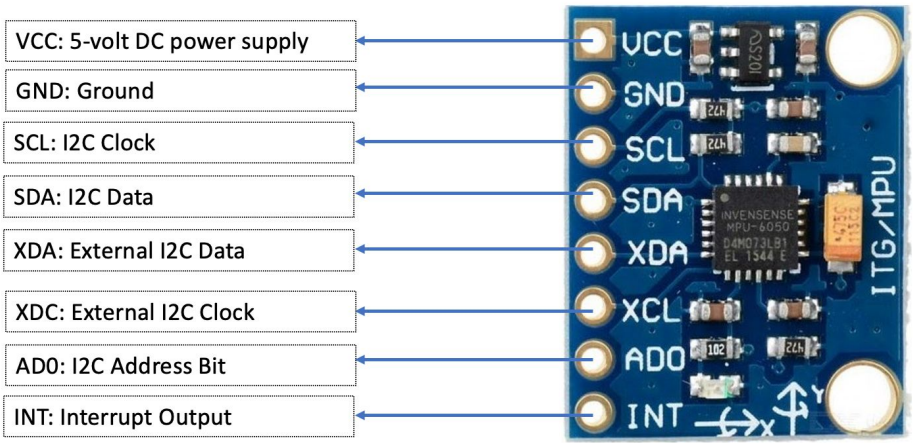

Figure 3.1.1.4: MPU6050 Pinout

Pin description

Pin name	Description
VCC	VCC supplies voltage to the module. The MPU6050 is supplied with a 5-volts power

	but the chip actually uses a 3.3-volt logic. Internally, the MPU6050 has its own resistors to change voltage levels to 3.3-volts
GND	Ground of MPU6050
SCL	Serial Clock - I2C Clock This pin provides the clock pulse for I2C
SDA	Serial Data - I2C Data This pin provides the data for I2C
XDA	Auxiliary Serial Data Apart from the SDA, XDA is an external I2C bus for attaching external sensors
XCL	Auxiliary Serial Clock The I2C clock line for the external I2C bus for XDA
AD0	This pin output is to change the internal I2C address of the I2C address of the MPU6050. Especially if we were to use more than one MPU6050.
INT	Interrupt output to signal when the processor is ready to read

Bluno Beetle V1.1

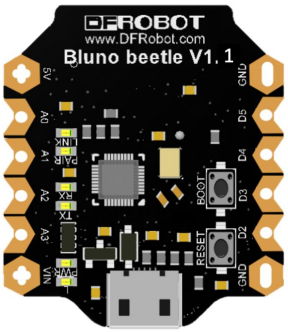

Figure 3.1.1.5: Bluno Beetle V1.1

The Bluno Beetle V1.1 is an Arduino based board with Bluetooth 4.0 (BLE - Bluetooth Low Energy). It uses Arduino IDE to upload code. Code can be uploaded via its inbuilt micro USB port as well as wirelessly.

Pinouts

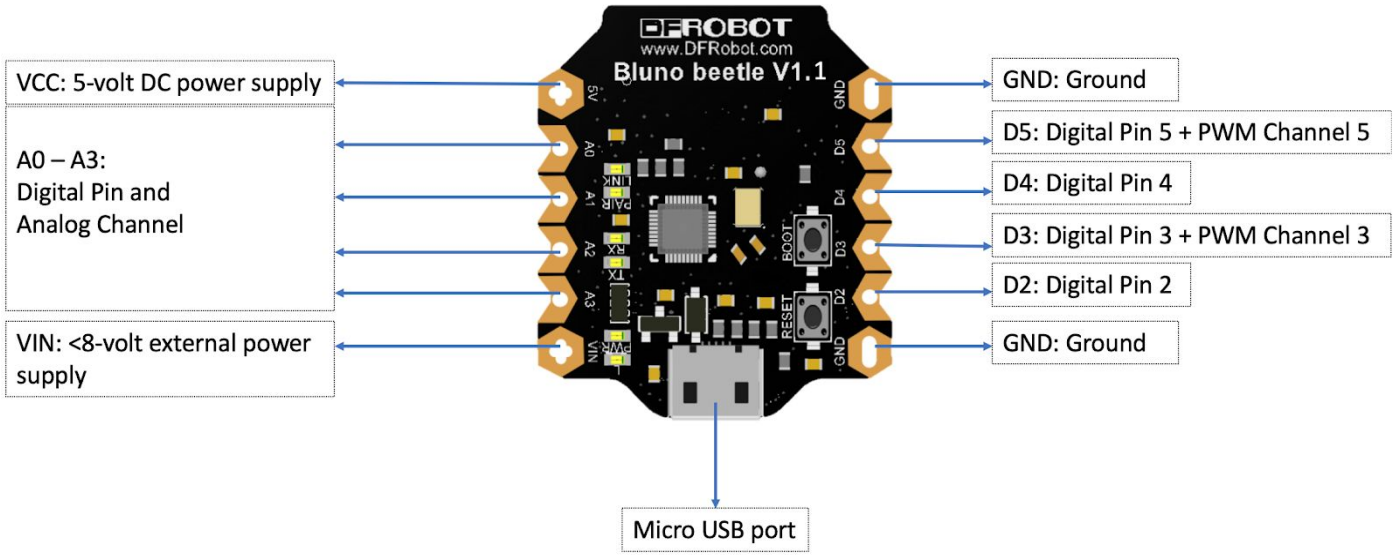

Figure 3.1.1.6: Bluno BeetleV1.1 Pinout

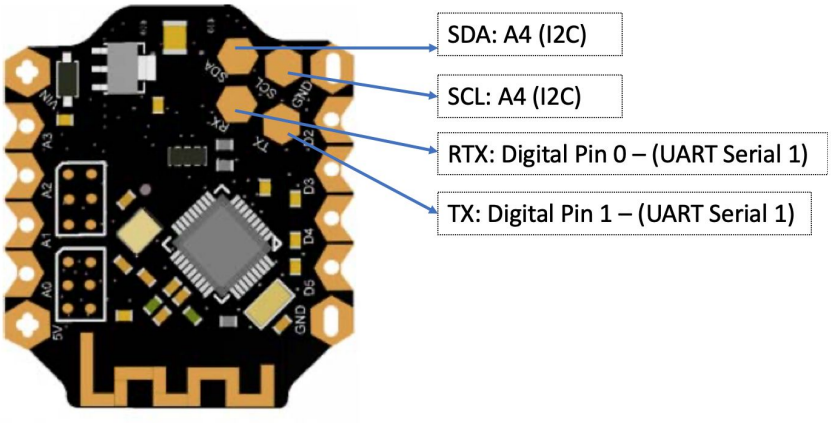

Figure 3.1.1.7: Bluno Beetle V1.1 Pinout

Pinout description

Pin name	Description
VCC	VCC supplies voltage to the module. The Bluno Beetle is supplied with 5 volts when connected.
GND	Ground
VIN	An external power supply that supports less than 8 volts.
A0 - A3	Digital pin A0 to A3 respectively
D2	Digital pin 2
D3	Digital pin 2 that supports PWM (Pulse Width Modulation) channel A0
D4	Digital pin 4
D5	Digital pin 5 that supports PWM (Pulse Width Modulation) channel A0
SDA	Serial Data - I2C Data This pin provides the data for I2C
SCL	Serial Clock - I2C Clock

	This pin provides the clock pulse for I2C
RTX	Digital pin 0 that has UART (Universal Asynchronous Receiver/ Transmitter) with serial 1
TX	Digital pin 1 that has UART (Universal Asynchronous Receiver/ Transmitter) with serial 1

Component details are gathered from the following datasheets:

Beetle BLE V1.1 **[1]**
https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0339_Web.pdf

MPU-6050 **[2]**
https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

Section 3.1.2 Pin table

The following table shows the mapping of pins within the two main components: MPU-6050 and Bluno Beetle V1.1.

Connection	MPU-6050	Bluno Beetle V1.1
a	VCC	VCC
b	GND	GND
c	INT	D4
d	SCL	SCL
e	SDA	SDA

Table 3.1.2: MPU-6050 to Bluno Beetle V1.1 mapping

Logic of connection:

a. VCC: Both MPU-6050 and Bluno Beetle gets their voltage from the same power supply.

b. GND: All grounds should be connected to the same ground together with the power supply.

c. INT / D4: Interrupt from MPU-6050 will be triggered when data needs to be sent to Bluno Beetle

d. SCL: The I2C clock from MPU-6050 and Bluno Beetle should be linked and synchronised to work hand in hand and obtain I2C clock.

e. SDA: MPU-6050 and Bluno Beetle V1.1 need a pin output to send data across each other for I2C.

Section 3.1.3 Schematics

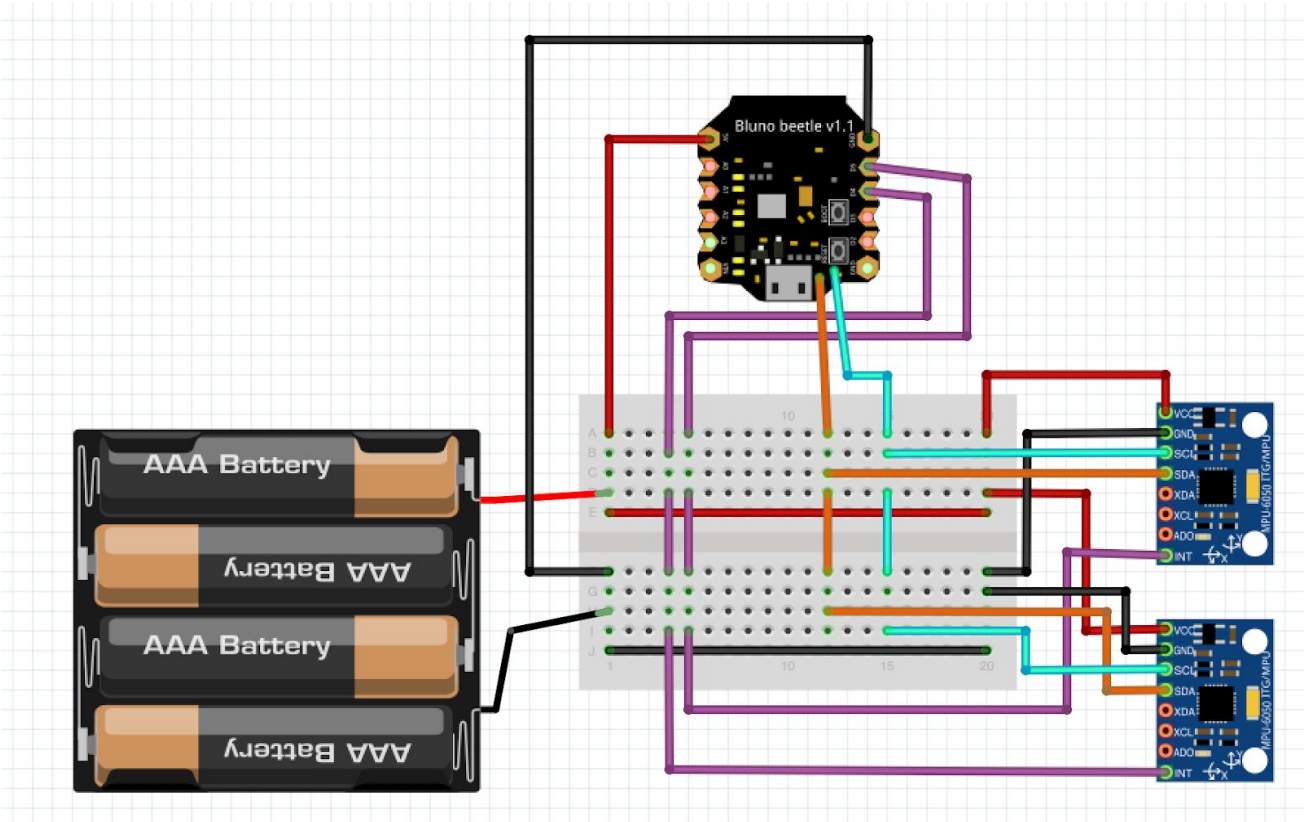

Figure 3.1.3: Full design schematic

Wire legend

Wire colour	Description
Red	VCC
Black	Ground
Purple	INT for 2 x MPU-6050
Orange	SDA for Bluno Beetle and 2 x MPU-6050
Cyan	SCL for Bluno Beetle and 2 x MPU-6050

Section 3.1.4 Operating voltage level and current

AAA Battery

- Voltage: 1.5 volts
- 4 x AAA Battery will be used to supply power to the wearable in series, giving it 6 volts.

MPU-6050

- Voltage: 5 volts but the chip uses 3.3 volts logic

Bluno Beetle V1.1

- Voltage: 5 volts

There will be 4 AAA batteries supplying power to the Bluno Beetle as well as 2 MPU-6050. 6 volts is more than sufficient to supply all components since Bluno Beetle requires at least 5 volts and MPU-6050 has its internal resistors to change voltage levels to 3.3 volts. No resistor, capacitor or voltage regulator is required.

Section 3.1.5 Algorithms and libraries

The interface used to code will be Arduino IDE as Bluno Beetle V1.1 is well designed with it. Arduino IDE has inbuilt libraries for simple set up. The relevant additional libraries with regards to the project design would be the I2C library that provides intuitive interfaces to I2C devices.

I2C library for connection to I2C devices:

- Jeff Rowberg has worked on multiple I2C sensors and has libraries on his Github which will be utilised in this project.
- This library will be used to extract the accelerometer X,Y, Z axis as well as the gyroscope Roll, Yaw, Pitch values
- https://github.com/jrowberg/i2cdevlib **[3]**

Section 3.2: Hardware FPGA

Section 3.2.1 Neural Network

Vivado is just an Integrated Developer Environment (IDE), although it does not directly translate to any implementation on the FPGA, it does have a User Interface (UI) that helps developers like us, to visualize mapping using block design.

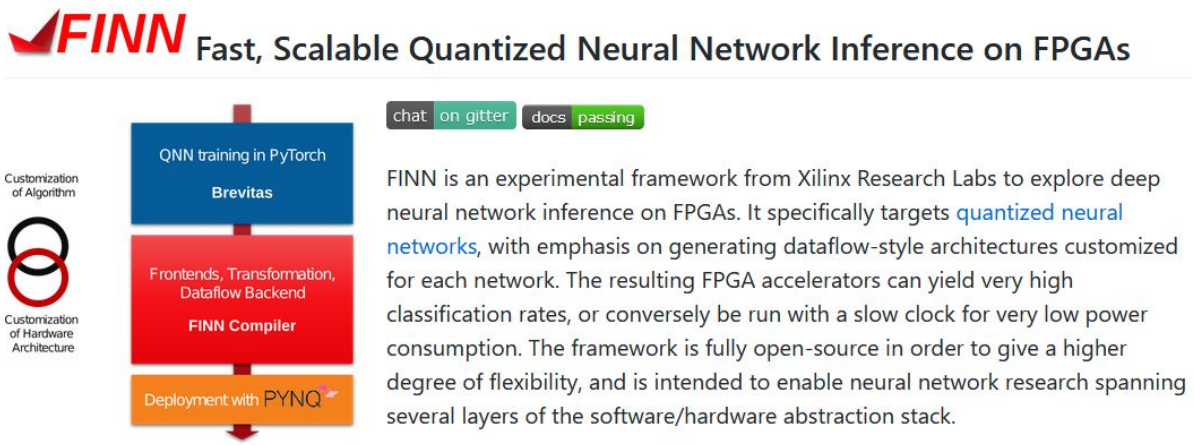

Figure 3.2.1 FINN

We begin with the standard FPGA Neural Network Development Workflow with Brevitas and FINN Compiler. Brevitas is a Pytorch Library for quantization-aware training. FINN Compiler and HLS Library to generate bitstream.

https://github.com/Xilinx/brevitas
https://xilinx.github.io/finn/
https://github.com/Xilinx/finn

Upon considering the possible Machine Learning (ML) Models in Section 5, our job is to implement hardware acceleration using some quantized neural networks. Models include Support Vector Machine (SVM), K Nearest-Neighbor (KNN) and Multilayer Perceptron (MLP).

Section 3.2.2 Mapping and Evaluating

Since the FPGA board is in a remote location, we are unable to run the bitstream directly onto the board. Therefore, we shall use Jupyter Notebook to upload the files and (attempt to) run it from there. The Jupyter Notebook is an interactive computing environment that enables users to author notebook documents.

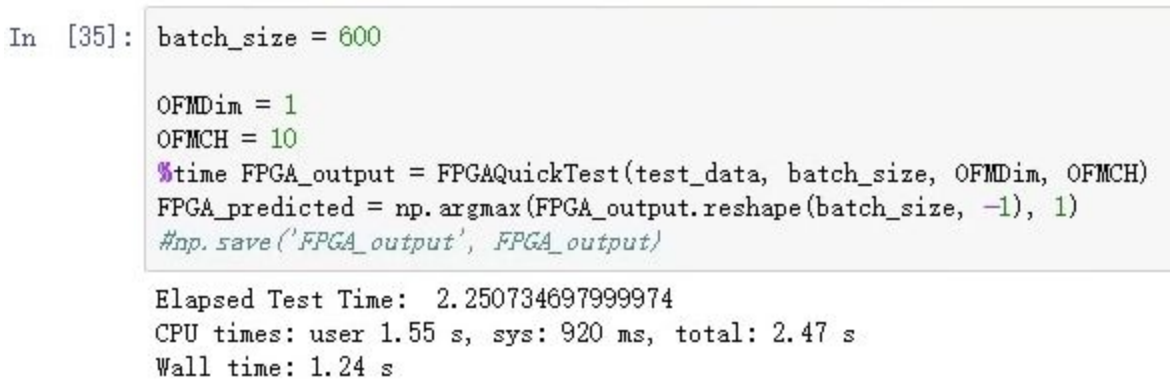

Figure 3.2.2 Jupyter Notebook

Evaluation process will also be done through Jupyter Notebook.

Section 3.2.3 Potential Optimization

Depending on the ML Algorithm, we will attempt both software implementation and hardware accelerated implementation. Afterwards, we will check its performance by checking and comparing the time taken to complete the tasks given, as well as the accuracy of the tasks completed. Depending on the outcome, we shall decide if optimization is ideal, and where can be further improved on. Factors such as parallelization and mapping issues will be considered then.

Furthermore, we can check on the memory utilization and try to reduce the power consumption.

In conclusion, we need to do a lot of research and testing to find the most optimal solution.

We could skip FINN and look into CNN, but this could involve mathematics.

Section 4 Firmware & Communications Details

Section 4.1 Internal Communications

The internal portion of the communications will be mainly responsible for interfacing both beetles and laptops together. This involves ensuring proper stable connections with the beetles, coming up with the format of data being sent, and ensuring the data sent is uncorrupted.

Section 4.1.1 Task Management on the Beetles

We are planning to use the FreeRTOS since it is a readily available and open-source library. It enables us to run multiple tasks simultaneously on the Arduino Beetle, which is critical in scheduling the reading of sensors as well as the sending and receiving of data between the Beetle and the Laptop. Additionally, most of us are familiar with it due to taking the CG2271 (Real-Time Operating Systems) which enable easier collaboration between the Hardware and the Communications team.

Section 4.1.2 Configuration and Set-up of BLE Interfaces

Firstly, each of our laptops require a Linux operating system (OS) to run the necessary bluetooth libraries. We plan to use Ubuntu in VirtualBox since it is lightweight and allows for easy file transfer to-and-fro our main OS. In Ubuntu, our laptops will first pair up with the beetles by first typing the command "*bluetoothctl*" in the terminal, followed by "*connect MAC_ADDRESS*" where *MAC_ADDRESS* refers to the MAC address of the beetle **[6]**. For VirtualBox in particular, there is a need to activate the bluetooth device in the virtual machine itself by clicking "*Devices > USB > BLUETOOTH_DEVICE*".

For bluetooth configuration in Ubuntu, we then need to set the minimum and maximum connection interval of the laptop's bluetooth interface to 10, by typing in "*echo 10 > conn_min_interval*" and "*echo 10 > conn_max_interval*". This is done to allow a higher data throughput by reducing time spent transmitting data via bluetooth which allows for some power-saving.

After doing all of the above, we then run the python script on Ubuntu which will automatically try to establish connection with the beetles via bluetooth.

Section 4.1.3 Communication protocol between Beetles and Laptop

We will utilise the Bluepy library which provides multiple high level functions and classes that help to connect to each Beetle with ease. Using the Bluetooth Serial protocol, we make use of the following service and characteristic UUIDs to read and write the data **[4]**:

- Service: “0000dfb0-0000-1000-8000-00805f9b34fb”
- Characteristic: “0000dfb1-0000-1000-8000-00805f9b34fb”

Should there be a need for multiple Beetles to be connected to the laptop at the same time, we will make use of the Python threading library to run multiple threads to handle data received by each beetle separately.

A 3-way Handshake is first used to establish connection where the laptop sends the ‘H’ byte first, then ‘A’ by Bluno, then ‘A’ by laptop again. The beetle reads the data sent via the Serial.read() function and the laptop will read the bytes sent by reading the data parameter as part of the handleNotification() function provided in the Bluepy library **[5]**. Whenever we send data from the Beetle, we simply use Serial.print(DATA_PACKET) since we are using the Serial protocol.

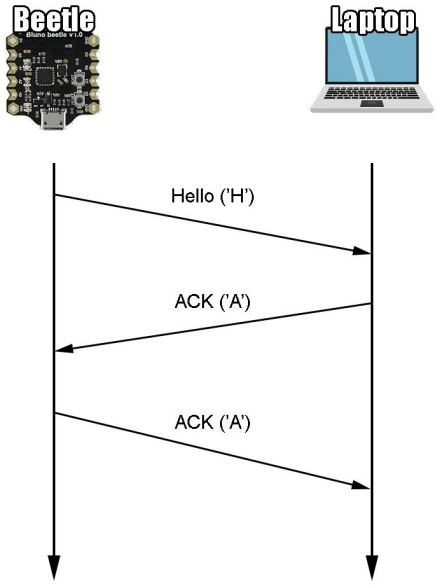

Figure 4.1.3: 3-way Handshake between Laptop and Beetle

The table below shows the packet types for the beetle and laptop to distinguish the type of data being sent. The first two (H, A) are involved in the handshaking process while the last one (R) refers to the sensor data being sent with its respective packet format.

Table 4.1.3.1: Packet Types in Beetles

Packet Type	**Packet Value**	**Significance**
HELLO	'H'	To be sent/received when first initialising connection with laptop
ACK	'A'	To be sent/received which signifies end of the handshaking procedure
READY	'R'	To be sent/received as followed by dummy data to be used for debugging and testing

The table below shows the packet format for sensor data being sent with a 'PacketType' of value 'R'. A portion of the packet is dedicated to 'Timestamp' for clock synchronization between the FPGA and Beetle, another portion concerns the various sensor readings from the 2 MPUs. They are split into 'Mpu1_T' and 'Mpu2_T' where the 'T' value signifies x, y, z, pitch, yaw or roll values generated by the MPUs. Lastly the 'Checksum' concerns the error-checking and end-of-packet marker for the data packet overall.

Table 4.1.3.1: Packet Format of Data Received by Laptop

Data Packet Format				
PacketType (char)	Timestamp (long)	Mpu1_x (float)	Mpu1_y (float)	Mpu1_z (float)
Mpu1_pitch (float)	Mpu1_yaw (float)	Mpu1_roll (float)	Mpu2_x (float)	Mpu2_y (float)
Mpu2_z (float)	Mpu2_pitch (float)	Mpu2_yaw (float)	Mpu2_roll (float)	Checksum (char)
Total Size: 54 Bytes				

The baud rate affects how quickly data is sent over serial communication. A baud rate of 115200 bps has been chosen. This is because it strikes the nice balance between having a decently fast transmission speed while also maintaining stable transfer of data packets to minimise losing or having invalid data packets.

Section 4.1.4 Ensuring Reliability of Data

A checksum will be appended at the end of the data packet, as a single char. On both end systems, it is computed by XOR-ing each char in SensorData. On the laptop's end, it will then compare the computed checksum with the checksum provided at the end of the data packet. This ensures that the data sent by the beetles is uncorrupted.

For reliable connection, a reconnect() option will be run on the laptop when an exception is caught which signals a failed connection between the laptop and the beetles. The exceptions are mainly occur when creating the Peripheral object at the start or at the waitForNotifications() portion in the main loop which waits for data to be sent by the beetles.

Section 4.2 External Communications

External communications refer to the connection and transfer of data over the network between clients and servers, notably between the laptop relays and Beetles, the Ultra96, and the evaluation and dashboard servers.This section will discuss how secure communications between clients and servers will be implemented, the threading processes running on the Ultra96, and finally how to estimate the synchronization delay between the different users.

Section 4.2.1 Secure communications between laptops, Ultra96 and evaluation and dashboard servers

The laptop relays sending data from the Beetles, Ultra96 and servers are all in different locations and mostly in different networks, and the Ultra96 can only be accessed via School of Computing's Sunfire network. Thus, to send data across multiple servers, we have decided to use socket communications over the internet utilizing SSH multi-hop tunnelling with port forwarding. Each laptop will bind a local port to a specified port on Sunfire, then the same Sunfire port will be bound to a port on the Ultra96. Two other ports on the Ultra96 will then be used to send processed data and results to the evaluation and dashboard servers respectively.

The laptop client TCP runs on the laptop relays and sends raw data received from the Beetle to the localhost port bound to the Ultra96 through Sunfire for processing. The processed data and results from the Ultra96 will then be sent to the servers via Ultra96's localhost ports. This flow of data from the laptops to the servers will be implemented using the Python socket module.

To ensure secure communications, data sent to the evaluation and dashboard servers will be encrypted via the Advanced Encryption Standard (AES) scheme, available via the Crypto.Cipher.AES Python library. A higher AES encryption standard using a larger key size will result in more secure encryption but slower encryption and decryption as well. Our product design requires fast and continuous transfer and processing of data from device to server, hence we will use the AES-128 scheme, which offers sufficient security without compromising on performance for our purposes **[7]**. A 16-character secret key will be used for the encryption, disclosed only to the evaluation and dashboard servers. The messages for the respective servers will be sent as encrypted plaintext.

The full communications process is detailed below:

1. The SSH port forwarding will be set up for all laptops and the dashboard server through Sunfire to the Ultra96. This can either be done externally as a bash command beforehand or embedded in the server TCP script using Python library sshtunnel.
2. The evaluation server, dashboard and Ultra96 servers will run, create sockets and wait for connection from the clients.
3. Raw data from the Beetles is sent through the laptop relays to the Ultra96.
4. Ultra96 determines the position, action, synchronization delay and other data based on raw data received.
5. Data is encrypted with a secret key, then sent separately to the evaluation and dashboard servers by connecting the client TCP to the server TCP.
6. Servers receive encrypted data and decrypts using the same AES scheme with the same secret key given beforehand.
7. TCP connection between servers and clients are closed when logging out from the respective clients.

Message format between Ultra96 and evaluation server:

```
#[position:str]|[action:str]|[syncdelay:int]|
```

where sync delay is in milliseconds, position is a string where the format is '1 2 3'. The number indicates the identity of the user and the order of the numbers indicate the relative positions of the users.

Message format between Ultra96 and dashboard server:

```
#[data1]|[data2]|[data3]|...|
```

The data to be sent to the dashboard is as yet undecided and will depend on further experimentation and analysis of user needs. For consistency, it will likely be in the format above which is similar to the format for the evaluation server, with the number of fields to be determined.

Section 4.2.2 Active processes and scheduling on Ultra96

There are several processes to be run on the Ultra96 concurrently. We will be using Python's threading module to implement parallelization and synchronization of these processes on the Ultra96.

The processes and their priorities are detailed in the table below, to be run as threads. Except for the first process, all processes will run indefinitely from the start until logout.

Table 4.2.2.1: List of processes on the Ultra96

Process ID	Process	Purpose	Priority
1	Start local server, clock synchronization between Ultra96 and laptops	Create socket connections between the laptops and Ultra96, perform time synchronization between Ultra96 and Beetles	N/A (Runs once at the start, locks all other process threads until it finishes)
2	Receiving data from laptops	Listens for Beetle data from laptops over the network, then forwards data to the machine learning algorithm thread for analysis. There will be one thread running for each Beetle, all at the same priority.	Highest
3	Machine learning algorithm to process sensor data and return results	Determines move performed by and relative positions of users using sensor data sent.	High

4	Send results to evaluation server	Send results of machine learning algorithm to the evaluation server.	Medium
5	Send data to dashboard server	Send data required for dashboard analysis and visualization to the dashboard server.	Medium

Process 1 is to run once before all other processes. To achieve this, process 1 will be assigned a Condition object at the start of runtime, while all other threads wait for the Condition to be set. When the connection is established and time calibration complete, the process notifies the data receiving threads and waits indefinitely. This effectively allows process 1 to run first to establish connection between the Ultra96 and laptops. When the Condition is set, an RLock object is released.

A thread that acquires an RLock object will prevent other threads from running until it has released the RLock object **[8]**. The three receiving data threads (process 2) should acquire the RLock object released by process 1 first. When data collection is completed, the data receiving threads notifies process 3 by releasing the RLock object and waits for the same RLock object. The same mechanism is applied to subsequent processes, where processes 4 and 5 will notify and wake up process 2 instead. As such processes 2 to 5 will run in a circular manner indefinitely until logout.

Section 4.2.3 Synchronization delay between users

To synchronize between users and the Ultra96 as they perform moves remotely, we decided to adapt a simplified version of the Network Time Protocol (NTP), taking the Ultra96's internal clock as a reference clock. NTP is a clock synchronization protocol between computer systems over the network **[9]**.

Synchronization delay between users refers to the relative time difference between the first user starting a move and the last user starting the same move. There are several factors to consider when estimating the delay between users, such as transmission delay and clock offset between each laptop and the Ultra96. First, time calibration will be performed between the Beetle/laptop and Ultra96, then whenever the start of a move is detected, the timestamps in the sensor data packets from the laptops will be used to estimate the synchronization delay between users.

Time calibration will occur at the beginning of connection between Ultra96 and Beetle/laptop. The Ultra96 will send a packet containing an ID uniquely identifying the receiving Beetle and the timestamp t_1 on the Ultra96 right before sending the packet. When the Beetle/laptop receives the packet, it appends the timestamp t_2 as it receives the packet, then appends the timestamp t_3 right before sending the packet back to the Ultra96. The Ultra96 completes the packet with the timestamp t_4 the moment it receives the packet from the Beetle/laptop.

Table 4.2.3.1: Packet format for time calibration

Data Packet Format				
ID (char)	Timestamp t_1 (Ultra96) (long)	Timestamp t_2 (laptop) (long)	Timestamp t_3 (laptop) (long)	Timestamp t_4 (Ultra96) (long)
Total Size: 33 bytes				

With the timestamps obtained above, the below calculations can be made to determine the transmission delay and clock offset between Ultra96 and laptop/Beetle to synchronize their internal clocks. We assume that this calculated transmission delay and clock offset holds for subsequent communications until the connection is closed.

$$\text{Round trip time (RTT)} = (t_2 - t_3) - (t_4 - t_1)$$

$$\text{One-way communication delay} = \text{RTT}/2$$

$$\text{Clock offset} = t_2 - t_1 - \text{RTT}/2$$

When the start of a move is detected, the Beetle timestamp data will be extracted from the packet corresponding to the beginning of a move. The asynchrony between the dancers can then be calculated with the following steps, taking into account the previously calculated clock offset.

1. Take packet timestamps t_A, t_B, t_C, where each timestamp indicates the start of the move of the respective Beetles, and the clock offset for each beetle o_A, o_B, o_C.
2. The start time for each user's move with respect to the Ultra96 can be calculated as: t_A+o_A; t_B+o_B; t_C+o_C.
3. Synchronization delay between the dancers is estimated by: $\max(t_A+o_A; t_B+o_B; t_C+o_C) - \min(t_A+o_A; t_B+o_B; t_C+o_C)$.

Adjustments to the synchronization delay calculation will be made after some experimentation, comparing synchronization delay measured from videos of test moves to the delay estimated by the above calculation of the same test moves.

Section 5 Software Details

Section 5.1 Software Machine Learning

The software Machine Learning component is responsible for collecting data sent over from hardware accelerometers, pre-process the data, apply proper data segmentation techniques to obtain usable data packets, extract features from the data and select suitable machine learning model/algorithm to tackle the classification problem of identifying the dance moves. There are various aspects we need to look at when we select the right techniques and models along the ways, including test accuracy, precision, power management, speed etc. Below sections will go into details on each of the steps in the whole workflow and explain the possible options that we need to explore to obtain the best machine learning solution for this project.

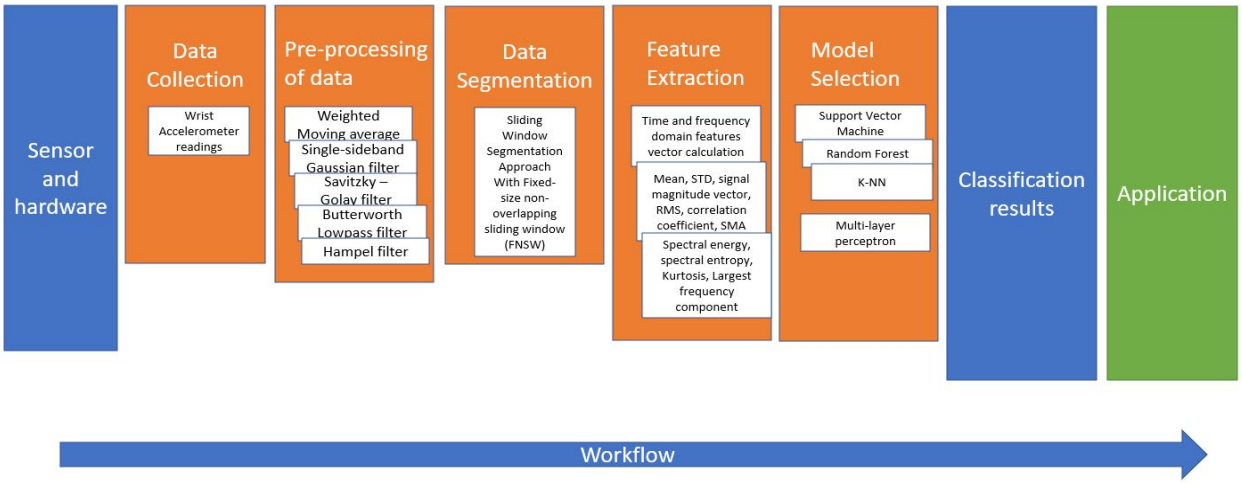

Figure 5.0.1 An overview of the ML workflow

Section 5.1.1 Sensor Data Segmentation

As mentioned in earlier sections, in Fitnests we are deploying two accelerometers in every set. One of which will be placed at the front chest to track relative positions of the user and the other will be placed at the user's wrist, collecting data for activity recognition (AR). It is known that raw data from accelerometers come in as a continuous flow as accelerometers are designed to measure instant metrics of the monitored body **[10]**. However, in order to deploy machine learning algorithms to recognize and detect activities, we need to be able to further divide the data stream into smaller groups, each representing an instance of a valid move to be recognized. This process is called Sensor Data Segmentation.

Existing literature has recommended the sliding-window based segmentation approach to segment time-series data streams generated by accelerometers in the area of activity recognition **[10]**. With this approach, the incoming continuous accelerometer data stream will be cut into windows with either fixed or flexible sizes (time duration). There are commonly two approaches to the fixed-sized sliding window: one is Fixed-sized Non-overlapping Sliding Window(FNSW) and the other is Fixed-sized Overlapping Sliding Window(FOSW). As the names suggest, the former deals with non-overlapping instances and thus data windows can be discrete and atomic. FOSW on the other hand deals with possibly overlapping data between adjacent windows. In FOSW, the study focuses on what is the appropriate window overlapping value given the data's overlapping percentages(window shifts). Dynamic sliding window approach on the contrary, enables the window size to be flexible based on different sensor features or typical activity duration. Window sizes are adjusted for static, dynamic and transitional activities**[10]**.

Given the specific application scenario of the project, we have decided to go with the simple yet effective FNSW approach as the dance moves will be non-overlapping and continue for roughly the same amount of time, giving rise to the possibility of implementing a fixed-size window to capture data.

As for sampling rates, according to Niall etal. in their paper "A Comprehensive Study of Activity Recognition Using Accelerometers"**[11]**, human physical activities generally "produce acceleration signals with most of the energy below 15HZ", it has also mentioned about a study on sampling rate to Machine Learning (ML) classification model performance using a bi-axial accelerometer with a rather limited subset of features and it turned out that the optimal sampling rate is between 15 HZ to 20 HZ. In the article we can also see a table about publicly available datasets for activity recognition based on body-worn accelerometers and we can see that the Samsung Galaxy S2 with its tri-axial smartphone accelerometer, decided to go with 50HZ as its sampling rate**[11]**. Given all the above information and the realistic constraint of our

power-limited design, we decide to set our sampling rate to 20 HZ and then also experiment 50 HZ to compare the classification performance and the power consumption trade-off.

Section 5.1.2 Data Pre-processing

After segmenting the continuous data stream from the accelerometer, we then need to pre-process the data packets to mitigate the influence of noise. There are various noise filtering techniques both in the time domain and the frequency domain. In the time domain, we have possibly moving average filter and median filtering, Single-Sideband Gaussian(SSG) which applies a convolution to smooth the curve and Savitzky–Golay filter used to fit a continuous subset of adjacent data points. Hampel filters can also be applied to remove outliers. In the frequency domain, we have Butterworth low-pass filters that are widely used to remove high frequency noises. We will experiment these filters to find out the best combination to remove noise.**[12]**

Section 5.1.2 Features Extraction

After preprocessing the data, we are ready to extract useful features from the data packets that might help us to uniquely identify and classify each dance move. Features will come from both time and frequency domain, explaining the fact that we need to mitigate noise on both domains. Referring to current literature and previously done studies, we have primarily decided to extract the following features:

In the time domain, we may look at statistical and non-statistical features such as mean, standard deviation (STD), root mean square (RMS), signal magnitude vector (SMV), signal magnitude area (SMA), Correlation coefficient, etc. In the frequency domain, we need to apply Fourier Transform to obtain frequency domain data and thus focus on features such as spectral energy, spectral entropy, Kurtosis and Frequency Signal Weighted Average**[10]**. Fourier Transforms are going to be implemented using Python's Numpy library with its existing support for Fourier transform(https://numpy.org/doc/stable/reference/routines.fft.html).

Section 5.1.3 Machine Learning Model Selection and Optimization

Given the classification nature of the project, there are a few models and algorithms that we find particularly interesting to explore: Support Vector Machine(SVM), K Nearest-Neighbour(KNN) and Multilayer Perceptron(MLP).

Support Vector Machine is a supervised machine learning model that could be used for classification or regression purposes. An SVM model represents data as data points in a

multi-dimensional space where the dimensions are the features selected. The model attempts to draw a hyperplane in the space to clear separate the points and thus categorize the points accordingly, new data points are classified according to which portion of the space do they fall into. Figure 5.1.3.1 below shows an illustration of how SVM works. Note that there is a Margin between the two support vectors and we need to choose the hyperplane such that margin is always maximized.

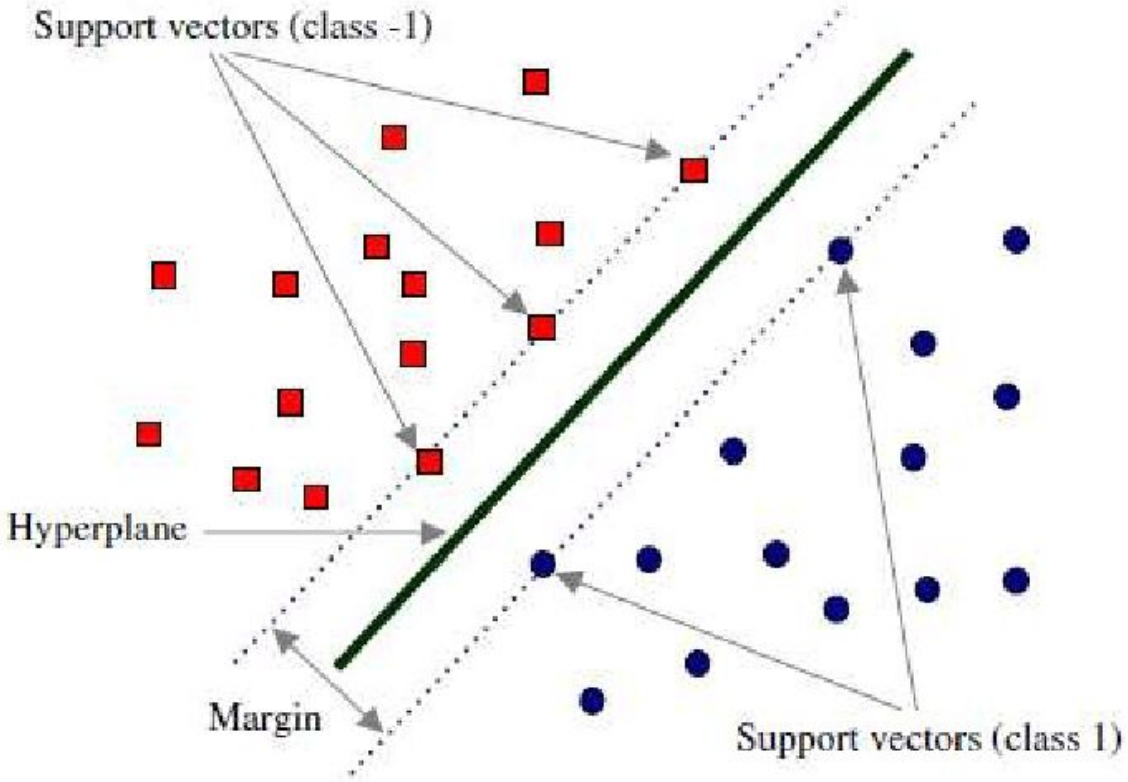

Figure 5.1.3.1 SVM illustration

By default, SVM is used as a linear classification model. However, there are cases when we need the hyperplane to be non-linear in order to clearly separate the data points. In such cases we need to choose the kernel functions properly. Figure 5.1.3.2 below shows a list of possible kernel functions. For our case we could probably use sigmoid kernel or the Leplace Radial Basis Function (RBF). Experiments have to be carried out to decide based on accuracy results.

Kernel Function Examples

Name	Function	Type problem
Polynomial Kernel	$(x_i^t x_j + 1)^q$ q is degree of polynomial	Best for Image processing
Sigmoid Kernel	$\tanh(a x_i^t x_j + k)$ k is offset value	Very similar to neural network
Gaussian Kernel	$\exp(\|x_i - x_j\|^2 / 2\sigma^2)$	No prior knowledge on data
Linear Kernel	$\left(1 + x_i x_j min(x_i, x_j) - \frac{(x_i + x_j)}{2} min(x_i, x_j)^2 + \frac{min(x_i, x_j)^3}{3}\right)$	Text Classification
Laplace Radial Basis Function (RBF)	$\left(e^{(-\lambda\|x_i - x_j\|, \lambda > = 0)}\right)$	No prior knowledge on data

There are many more kernel functions.

Figure 5.1.3.2 SVM Kernel Function Examples

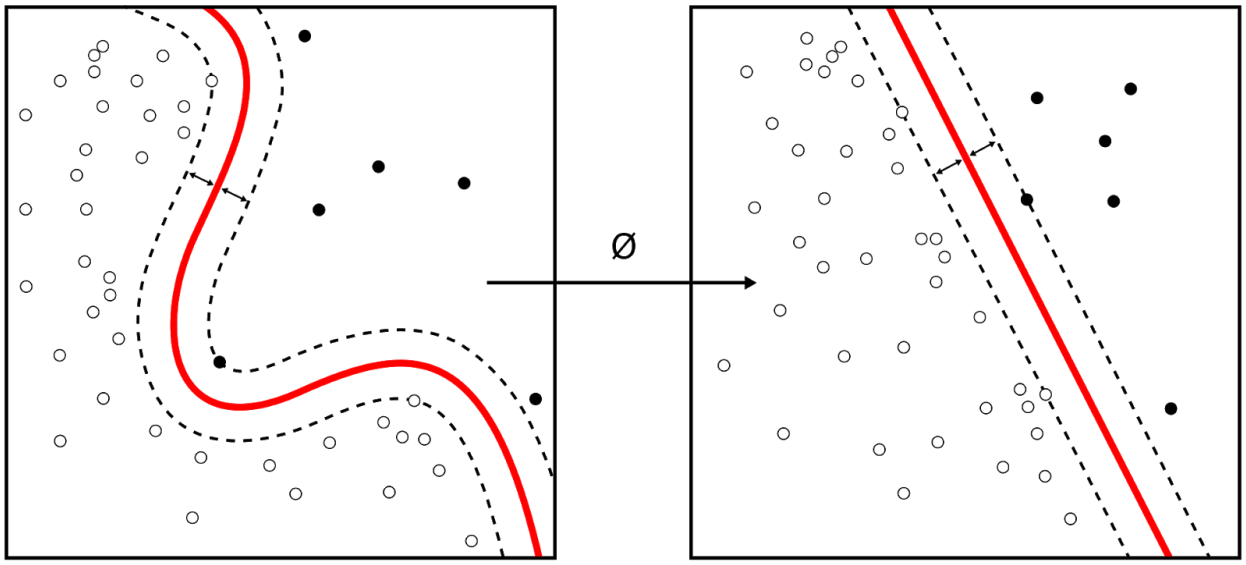

Figure 5.1.3.3 An illustration of how different SVM kernel functions work in non-linear cases

Another possible classification algorithm that we can try is the K-Nearest Neighbors algorithm. The input will be k closest training examples and the output in our case will be the class that a particular point belongs to where it is decided by a plurality vote of the point's neighbours, and the class of the object will be the most common class among its k nearest neighbours. K here will be a positive integer and typically is set to be small. Figure 5.1.3.4 illustrate how KNN works and the importance of choose a suitable K value. In the example, if k = 3, the new data point will be classified as Class B but if we change k to 7, then the new data point will then be classified to belong to Class A.

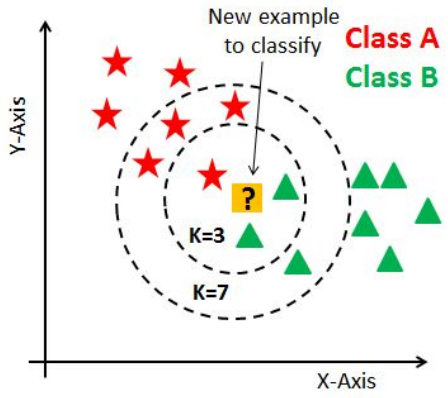

Figure 5.1.1.4 KNN example with illustration of different k values

The next possible Machine Learning algorithm we can try is the Multilayer Perceptron. It is actually a class of feedforward artificial Neural Network (ANN). Feedforward Artificial Neural Networks are distinguished from other Neural Networks as they have connections that do not form a cycle. It is therefore different from the recurrent Neural Networks where connections are fed back and form cycles. Information flows in one direction from the input nodes through the hidden layer nodes to the output nodes. Figure 5.1.1.5 shows an illustration.

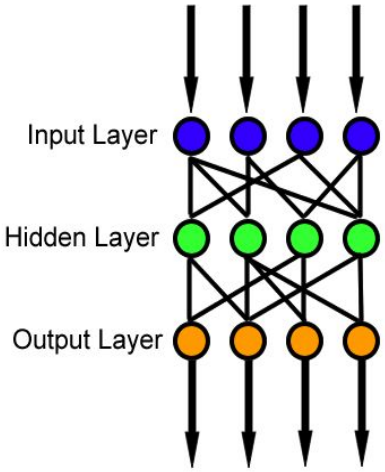

Figure 5.1.1.5 Feedforward Neural Network

Specifically, in multilayer perceptron, there are multiple layers of computational units called perceptrons, each with an activation function to be defined. These perceptrons are interconnected in a feedforward way and each perceptron in one layer only has directed connections from it to those in the subsequent layer. Exact structure details will be determined after training and comparison so we can decide on how many hidden layers to have, etc.

After trying out different machine learning models, we should be able to decide on the one that produces the highest test accuracy in terms of classification and also we need to take into consideration the power consumption issue.

Section 5.1.4 ML Model Training and Validation

When it comes to model training and validation, there are a few steps that we need to follow and there are certain variations and different methods are created to tackle different focuses. To begin with, the first step will be to gather data and split the datasets. Data can be obtained online first by looking for similar datasets on Kaggle or assemble the kit and obtain real data from our sensors. After gathering datasets, we then split the datasets into training dataset, validation dataset and testing dataset, usually as shown in Figure 5.1.4.1 below.

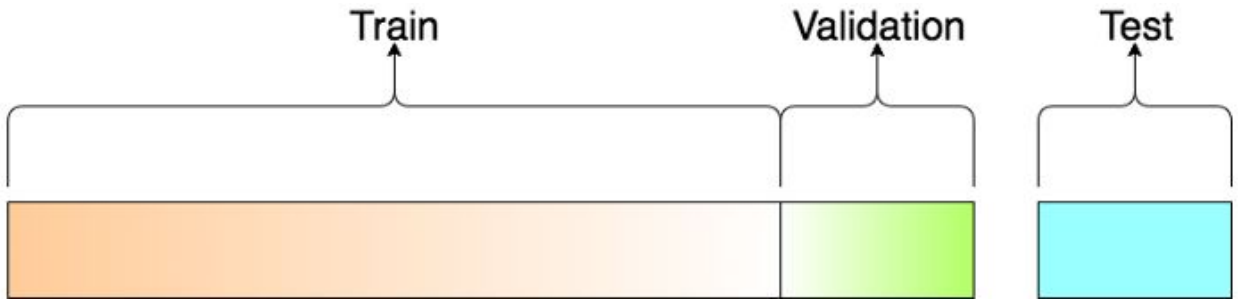

Figure 5.1.4.1 Split of dataset into Training, validation and test datasets

Training dataset is the dataset that we first feed to the model for it to learn from. Validation datasets are introduced later on as an unbiased evaluation of how the model fit on the training dataset and also to tune the hyperparameters. The model will occasionally run into these data but will not learn from them. We use results from the validation dataset to fine-tune the model. At last, we will have the completed isolated test datasets to test the completely-trained model for its accuracy on new data samples.

Another important concept to bring up would be cross-validation. This term comes into place when we only split the dataset into training and test set. In order to test the performance of the model before introducing the test dataset, we can do cross-validation by utilizing the training dataset itself. A popular cross-validation method is 10-fold cross validation, illustrated below in figure 5.1.4.2. Essentially, we split the training dataset into 10 subsets called folds. We first train our data on 9 of the 10 folds and hold out one fold to use on evaluation of the model. We can repeat this process 10 times, each using a different subset and the hold-out set. In the end we average the performance of the 10 rounds as the final performance estimate. K-fold cross validation is proved to give reliable estimate of model performance.

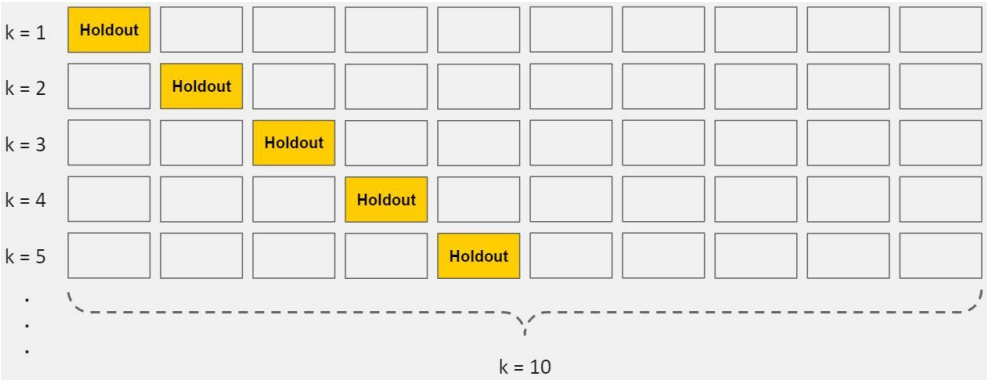

Figure 5.1.4.2 an illustration of K-fold cross validation

Section 5.1.5 ML Model Testing

After training and validating the model, we now need to test our model on the test datasets in order to make sure that our model are precise and accurate, in other words, it does not overfit to learn biases from the training dataset too much and also not underfit to be unable to classify correctly. Test datasets should be kept isolated until this phase to ensure test integrity.

There are various performance metrics that we can use to evaluate our model. Previously we mentioned precision and accuracy. Precision is defined as the fraction of true positives against predicated positives and accuracy is defined as the fraction of correct predictions over the total number of predictions. In simple language, Accuracy measures how well our model behaves in terms of making correct predictions and precision measures how consistent is our model in terms of providing true positives.

Another very important performance metric we would like to bring up here would be the Area under the Receiver Operating Characteristics Curve(AUROC), as illustrated in Figure 5.1.5.1. This is a performance measurement for classification problems thus fit our context of usage. Receiver Operating Characteristics (ROC) is a probability curve and Area Under Curve (AUC) tells us how clear is the model in terms of separating different classes. The higher AUC is, the better the model performs in terms of distinguishing different classes.

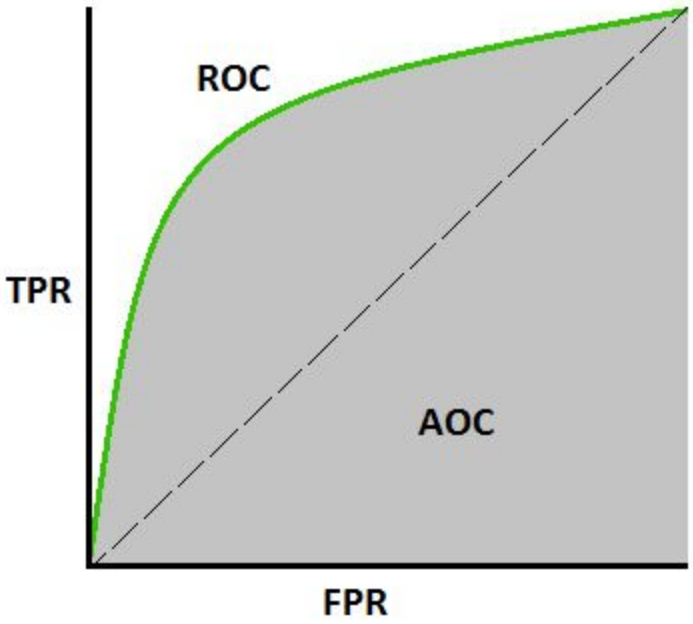

Figure 5.1.5.1 An illustration of the AUROC

The ROC curve is plotted with True Positive Rate(TPR) on the y-axis and False Positive Rate(FPR) on the x-axis. TPR is calculated by the number of True Positives over all Positive cases. FPR is calculated by the number of False Positives over all negative cases. An excellent model will have AUROC near 1 which means it has a very good separation of classes and a poor model will have AUROC near 0 means the separability between classes is negligible. A point to note is that in our case, we have a multi-class scenario where there are more than two classes possible. In such case, we can draw N ROCs where N = number of classes. In each ROC, it will be one class classified against the other classes as a whole**[13]**.

Section 5.2 Software Dashboard

Section 5.2.1 Software Dashboard Design Mockup

The software dashboard portion will be mainly responsible for displaying the analytics behind the users move and will provide feedback for the user,

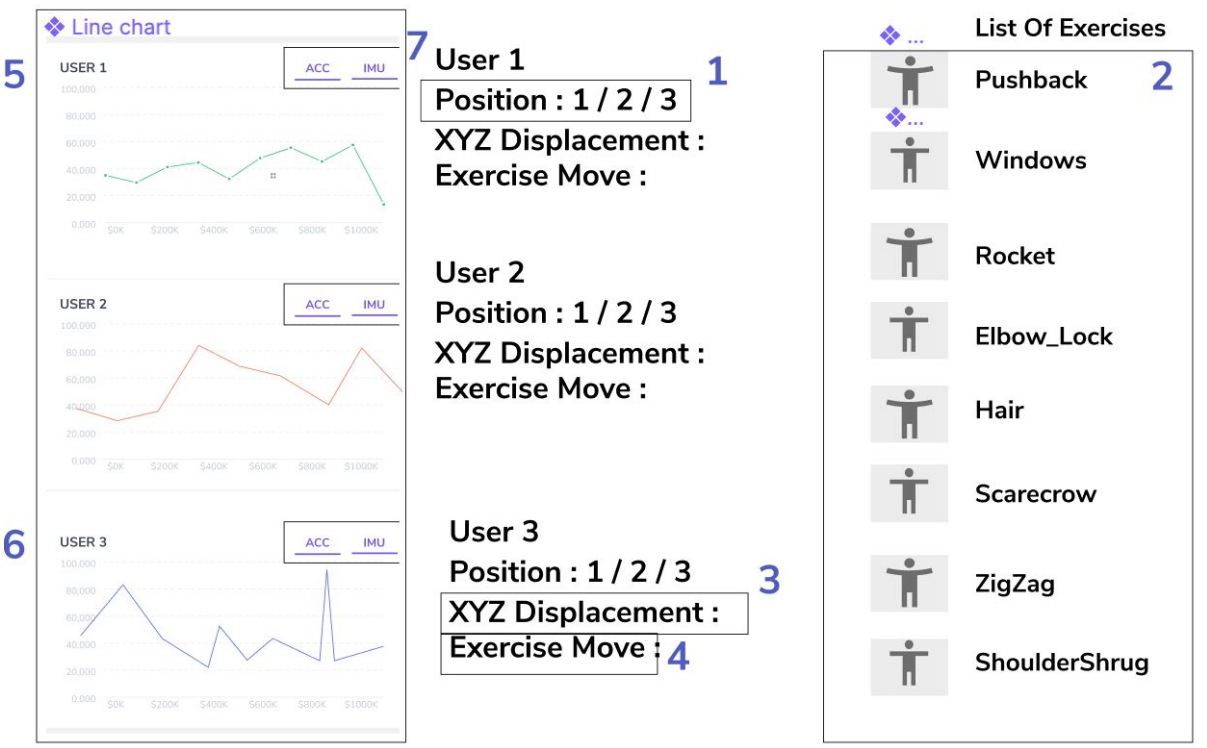

Figure 5.2.1.1 : Figma Mockup of DashBoard design

No	Feature	Details
1	User's Position	This will indicate which position the user is currently at. The position display will be update according to the user's actions(whether they move left or right)
2	List Of Exercises	Displays the list of exercises that a user may execute
3	XYZ Displacement	Displays the accelerometers reading
4	Exercise Move	This feature will show the exercise that is currently being executed by the user

5	Accelerometer Graph	Displays the readings obtained from the accelerometer via a graph
6	Inertial Measurement Unit Graph	Displays the readings obtained by the IMU via a graph
7	Toggle Button - ACC/IMU	Switch used to toggle between the ACC and IMU Graph

Table 5.2.1.2 : Dashboard Feature details

Section 5.2.2 Storing sensor data

For the dashboard backend, we will be using MongoDB as our database to store incoming data received from the Ultra96. A server script will be utilised to process
the data packets and upload the relevant data onto the MongoDb Database. The data will then be retrieved and used to provide analytics for the dashboard display.

Section 5.2.3 Real time streaming

We will be using the MERN stack(MongoDb, Express, React, NodeJs) for our dashboard application. For real time streaming, we will be using setTimeout function to periodically call the mondoDb api (adjusted to suit our latency delay) in a poll like manner to update our React Components. We will then be using D3 to provide data visualisations.

Section 5.2.3 User-survey

For our project, our end users are gym members who are looking to utilize our product in order to ensure that they are conducting their exercises with proper form. Hence in order to understand the needs and preferences of our users we will be first conducting a survey via google forms. This survey will consist of both open ended and close ended questions.

Sample Size of Survey: 50
Sample type : Convenience Sampling
Study Method : Online Survey

Reason for Study Method : With the online survey we can have a mix of open and closed ended questions that the gym members can answer at their own leisure. Hence this method will be the easiest to get a large sample size. The alternative would be via contextual enquiry which would allow us to understand the possible usage of our application first hand, however this method has been ruled out since it is not feasible considering current Covid19 restrictions. To allow us to better understand the user pain points, we could leave the open ended portion to address this issue. In addition we will also be using video conferencing to allow users to try out our Figma prototype. This will allow us to see the relevant pain points that a user might experience and at the same time receive valuable feedback for our product.
Examples of Survey Questions:

Functionality:
What do you think an exercise movement tracker can do?
What type of analytics would you like to know for your exercise tracker?

Perceptions:
Do you prefer having more detailed or a simplified user interface for your analytics dashboard?
Do you think an exercise movement tracker is beneficial for your gym routine?

Section 6 Project Management Plan

Week	Milestone	HW1	HW2	COMS1	COMS2	SW1	SW2
3	Research	Research on Individual Components					
4	Complete design report	Research more on components as well as I2C communication. Standardise schematic design of circuit.	Familiarization with Vivado, FINN and Jupyter Notebook	Set up stable connection between the laptop and one beetle	Set up ssh tunnelling process to send data between laptop, sunfire, Ultra96 and eval & dashboard servers	Research on possible models to use. Look for sample data from the Internet to test out the models	Set up database backend and core dashboard functionalities(basic frontend). Draft user survey Draft wireframe for dashboard
5		Set up initial circuit for both MPU6050 and beetle ble Initial MPU6050 start and calculations	Testing with sample algorithms, such as the existing trained models.	Set up stable connections between the laptop and three beetles	Data encryption (secure socket comms) with dummy data	Test possible models and gather data from our sensors	Update wireframe based on user feedback, implement design change. Test user Interface
6		Work on MPU6050 accelerometer and gyroscope code, setting offsets and calculations Design and attach elastic	Reflect and refinement of the hardware design	Create handshaking protocol and packet format to send data between laptop and beetles	Determine format of data packet to send to server and dashboard, create proper sample data for	Decide on the model to use and refine algorithm according to input data	Decide on the type of data are needed for the dashboard Test realtime feedback

		straps for hardware to make it wearable			dashboard use		using dummy data provided by comms
R		Debugging accelerometer and gyroscope code, setting offsets and calculations Calibrations and soldering after finalising schematic and hardware build	Liaising with SW1 & others	Testing for errors and edge cases in the case of connection being lost or timed out	Estimating user sync delay with dummy data	Test the models on online datasets and measure performanc e	Implement extended front-end features using react and integrate with mongoDb database backend
7	Individual sub-component assessment	More calibrations and testing Set up the rest of the sets for the rest of the team	Implementa tion of hardware designs based on the ML model	Ensure connections are working properly between 3 beetles and the laptop with the specified packet format containing dummy data	Data communicat ion between bluno/lapto p and Ultra96, process threading	Run models on online datasets and compare them based on certain performanc e metrics	Test data sent from beetles and ensure that that all the relevant information are displayed accordingly
8		Feedback from individual assessment Finalising testing and calibrations. Improvising and touching up to make	Finalization and Optimizatio n Part 1	Integrate sensor data received from HW1 to be sent to the laptop, and implement COMS2's protocol to send data to	Finalizing data format for dashboard, integrating comms components for 1 user	Feedback from individual assessment. Get data from our own devices and train models with it. Modify	Update UI/UX and add any extended features Integrate D3.js for Graph Visualisatio n

		hardware neat and more comfortable to wear.		FPGA/Dash board.		accordingly for test accuracy	
9	First evaluation test	Testing the whole product on a single user with 3 moves					
10		Take feedback from evaluation and improve on what is necessary. Check calibrations and fix bugs for all sets	Compariso n of speed and accuracy	Bug fixing and perform rigorous testing on the communicat ion from beetles, to laptop, to FPGA and Dashboard.	Bug fixing and optimizatio n on sync delay estimation	Bug fixing and optimizatio n on test accuracy	Conduct user testing on the dashboard and any possible bug fixing.
11	Second evaluation test	Testing the whole product on 3 users with 3 moves, and tracking location					
12		Final comments from evaluation test Improvising and touching up to make hardware neat and more comfortable to wear.	Finalization and Optimizatio n Part 2	Fine-tuning the entire communicat ion protocol	Finalization on communicat ion protocol	Finalization on ML models	Test the product
13	Final evaluation	The whole product should be working on 3 users with 8 moves, and tracking location					

Table 6: Weekly Timeline of Deliverables

References

[1] DFRobot. Bluno Beetle Datasheet. [online] Available at: https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0339_Web.pdf

[2] Invensense. MPU-6000 and MPU-6050 Product Specification Revision 3.4 [online] Available at: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf https://github.com/jrowberg/i2cdevlib

[3] Jeff Rowberg. I2C device library collection for AVR/Arduino or other C++ based MCUs. [online] Available at: https://github.com/jrowberg/i2cdevlib

[4] DFRobot. *BlunoBasicDemo*. [online] Available at: https://github.com/DFRobot/BlunoBasicDemo

[5] Ian Harvey. *Bluepy - a Bluetooth LE interface for Python*. [online] Available at: https://ianharvey.github.io/bluepy-doc/index.html

[6] Bruce Byfield. *Configuring Bluetooth devices with bluetoothctl.* [online] Available at: https://www.linux-magazine.com/Issues/2017/197/Command-Line-bluetoothctl

[7] Josh Lake. *What is AES encryption and how does it work?*. [online] Available at: https://www.comparitech.com/blog/information-security/what-is-aes-encryption/

[8] PyMOTW. *threading - Managing concurrent threads*. [online] Available at: https://pymotw.com/2/threading/

[9] *What is NTP?* [online] Available at: http://www.ntp.org/ntpfaq/NTP-s-def.htm

[10] Qin Ni ,etal. *Dynamic detection of window starting positions and its implementation within an activity recognition framework* [online] Available at: https://www.sciencedirect.com/science/article/pii/S1532046416300594#:~:text=The%20sliding%20window%20segmentation%20approach,either%20static%20or%20dynamic%20sizes.

[11] Niall, etal. *A Comprehensive Study of Activity Recognition Using Accelerometers* [online] Available at: https://www.mdpi.com/2227-9709/5/2/27/htm

[12] Jiao Liu, etal. *Human Activity Sensing with Wireless Signals: A Survey* [online] Available at: https://www.mdpi.com/1424-8220/20/4/1210/htm

[13] Sarang Narkehede, *Understanding AUC - ROC Curve* [online] Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5