CG4002 Embedded System Design Project

January 2020 semester

"Dance Dance"
Design Report

Group 6	Name	Student Number	Sub-Team	Specific Contributions
Member #1	Lim Yi Sheng	A0168536U	Hw1: sensors	Section 1 Section 2.1.3, 2.3 Section 3.1-3.5 Section 6.3
Member #2	Soh Boon Jun	A0168082B	Hw2: FPGA	Section 2.3 Section 3.4-3.9 Section 6.3
Member #3	Melvin Tan	A0164512M	Comm1: Internal	Section 2.1.1-2.1.2 Section 4.1-4.5 Section 6.3
Member #4	Jin Shuyuan	A0162475B	Comm2: External	Section 2.1.2 Section 4.6-4.10 Section 6
Member #5	Davindran S/O Sukumaran	A0167009E	Sw1: ML	Section 5.1-5.5 Section 6.3
Member #6	Gerald Chua Deng Xiang	A0167371B	Sw2: Dashboard	Section 2.2 Section 5.6-5.10 Section 6.3

Section 1 System Functionalities

Section 1.1 Use Cases

For all the use cases below, unless specified otherwise:

1) The computer used to give the dancers instructions is the Evaluation Server
2) The computer used to show the analytics data is called the Dashboard
3) The dancers are the Users
4) The Ultra96 and Arduino Beetle setup, along with the connected sensors, is the System.

- **Use Case: UC01 - Correct position and dance move**
    - **MSS:**
        1. Server displays positions the users are supposed to be in alongside a random dance move on screen.
        2. Users move to their destined positions and perform the dance move.
        3. System detects and predicts the users' position, dance moves performed by the users and estimates synchronization delay between the users, and sends the information to the dashboard and evaluation server.
        4. Evaluation server checks and store the correctness of the position, dance move prediction and records synchronization delay of the users alongside the correctness of position and dance move in a log file and shows the next set of positions and dance move on screen. Dashboard stores and analyses the sensor data received.
           Use Case ends

- **Use Case: UC02 - Final logout dance action**
    - **MSS:**
        1. Server displays "Logout" and the positions the users are supposed to be in on screen.
        2. Users move to their destined positions and perform the final dance move.
        3. System detects and predicts the users' position and dance moves performed by each user
        4. The system closes the connection with the evaluation server.
           Use Case ends

Section 1.2 Feature List

- Detects and gathers data on the movements of each dancer with the help of the GY-521 (MPU 6050) IMUs.
- Gathers electromyography (EMG) data of 1 dancer to detect flexes of the dancer's muscle group and determine muscle fatigue of that individual.

- Predicts the dance actions of all 3 dancers using machine learning algorithms.
- Obtains relative position of the 3 dancers based on the signals and data obtained from sensors.
- Estimates synchronization delay between the 3 dancers when they are dancing.
- Stores sensor data and output onto the evaluation server.
- Analyzes individual dancers' movements, visualizes the results in a dashboard and displays relevant data analytics through a remote laptop.
- Logs out upon executing a specific dance move.

Section 2 Overall System Architecture

Section 2.1 High-Level System

Section 2.1.1 Implementations on Beetle and Ultra96

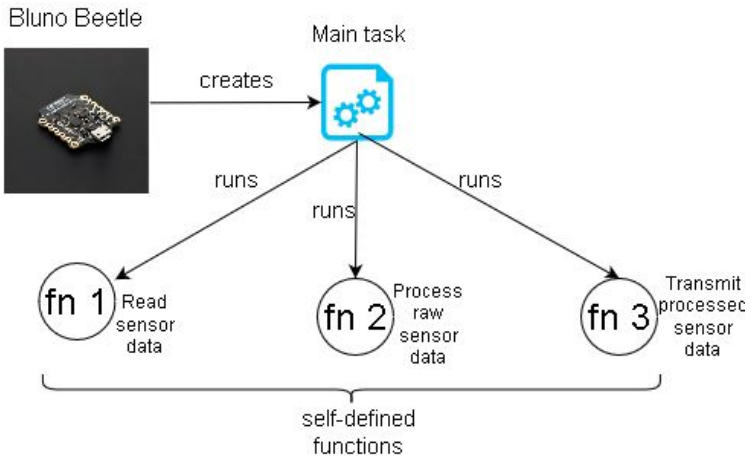

Figure 1: Functions defined in Beetle

The Beetle will have a main task created, which will run mainly three functions. The first function, reading sensor data, will obtain data from sensors when dancers dance. The second function will then do pre-processing on the raw sensor data and be converted to a format suitable to be transmitted to the Ultra96. The third function will transmit properly formatted data to the Ultra96.

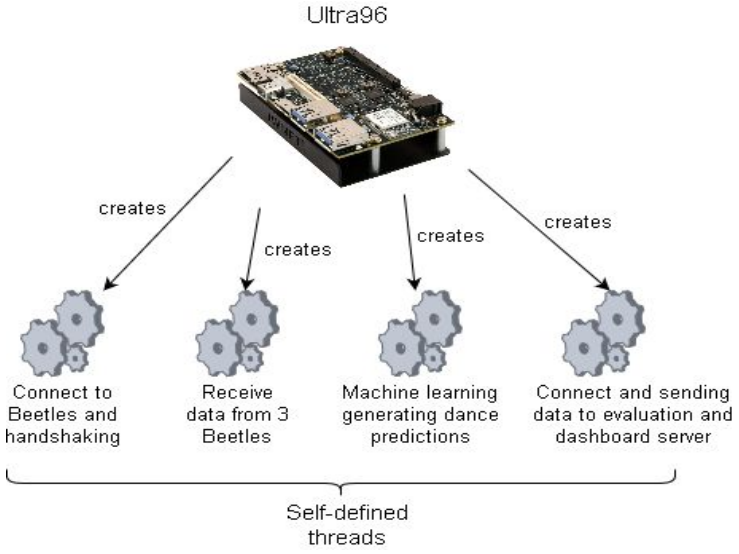

Figure 2: Processes in Ultra96

The Ultra96 will have four threads created as shown in figure 2 above. The order of execution goes from left to right, starting from connection and handshaking with the Beetles and ending when results are being sent to the evaluation and dashboard server.

Section 2.1.2 Communication Protocols

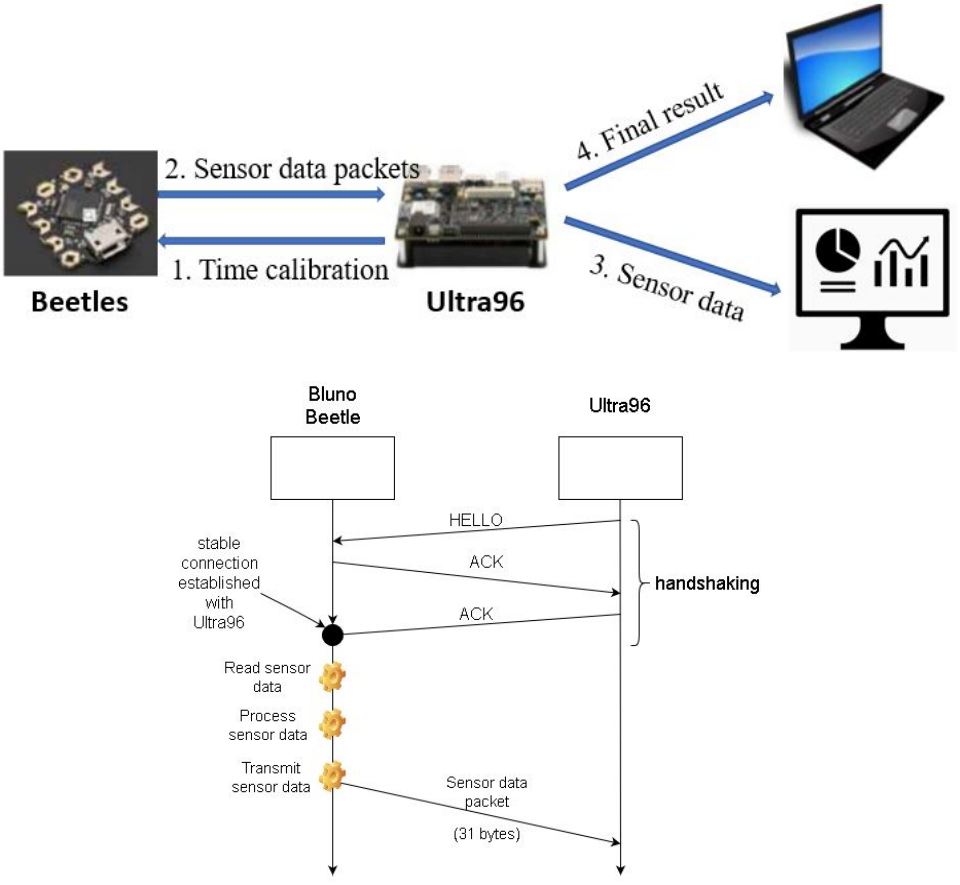

Figure 3: Establishing connection and sending data

1. **Time calibration**

Packet format (Time calibration packet)					
Packet_type (char)	ID (char)	Timestamp (long)	Timestamp (long)	Timestamp (long)	Timestamp (long)
Total size:18 bytes					

2. **Sensor data packet**

Packet format (Sensor data)					
Packet_type (char)	Timestamp (long)	Start (char)	IMU1[6] (float)	IMU2[6] (float)	Checksum[4] (char)
Total size:58 bytes					

3. **Sensor data to dashboard server**

Packet format (Sensor data for dashboard)
Sensor data (str) #data1|data2|data3|...|
Total size:multiple of 16 bytes

4. **Final results sent to evaluation server**

Packet format (Final result)
Final result (str) #position|dance action|synchronization delay|
Total size:multiple of 16 bytes

Section 2.1.3 Hardware components

The hardware components involved in our design are the following:

- Bluno Beetle
- GY-521 (MPU 6050) IMU
- MyoWare Muscle Sensor
- Ultra96

The connections between the various individual hardware components can be seen across the screenshot below.

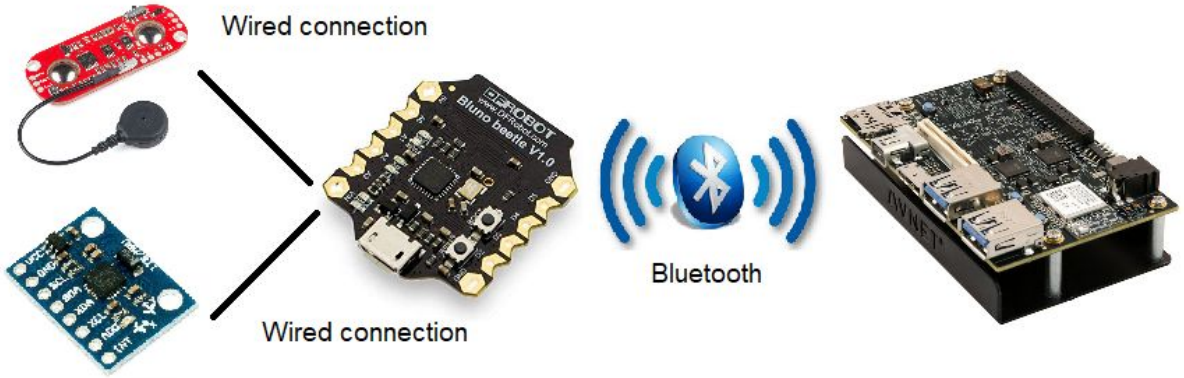

Figure 4: Hardware connection and interfacing

In our design, we will be using wired connections from the MyoWare Muscle Sensor and GY-521 (MPU 6050) IMU to the Bluno Beetle for transferring the sensor data received from the respective sensors to the Bluno Beetle. The Bluno Beetle will then process these raw data received from the sensors and transfer these processed data to the Ultra96 via Bluetooth.

Section 2.2 Final Form of the system

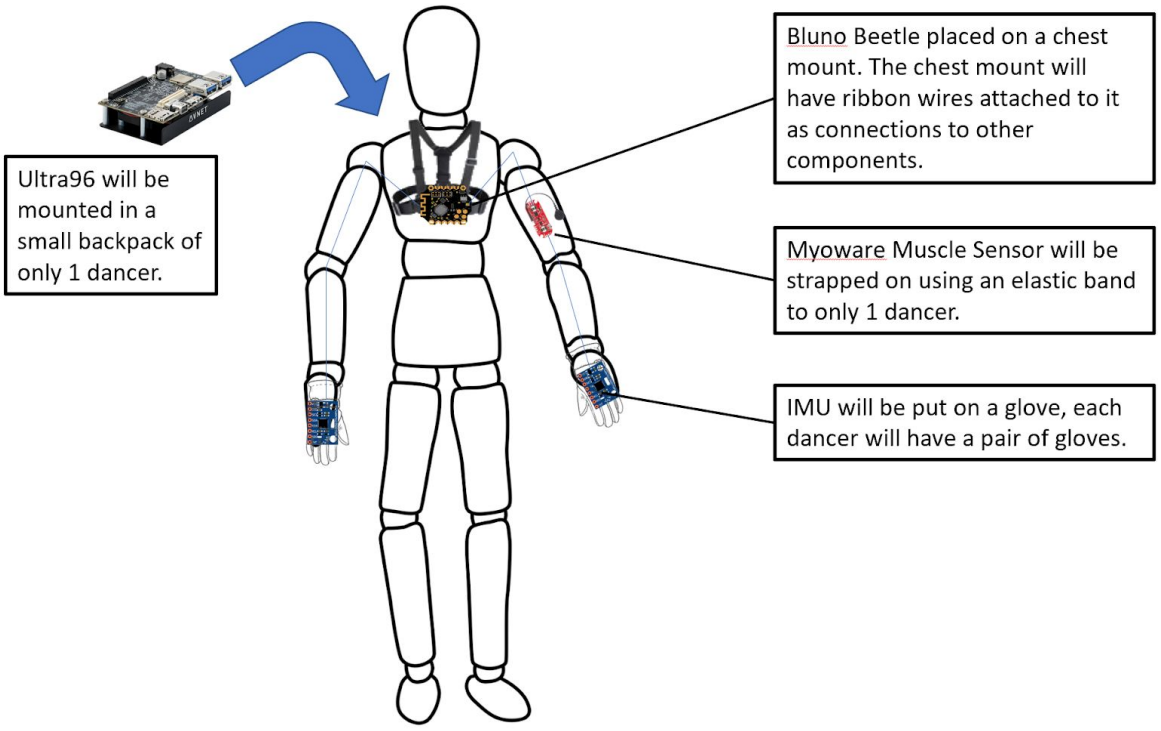

Figure 5: System's design final form

Section 2.3 Main Algorithm for Activity Detection

In this section, we will first provide an overview on the major steps in our algorithm and the steps will be explained briefly. The major steps in our main algorithm for this project are as follows:

1. Detect movement on sensors
2. Bluno Beetle to send a special flag packet alongside data packets to Ultra96
3. Ultra96 will start collecting data from data packets after the special flag packet
4. Ultra96 will use the collected data, pre processed and segmented according to features to the machine learning algorithm and classify the location of dancers and dance move accordingly
5. The predicted dance move alongside locations will be sent to the evaluation server for logging and the sensor data will be sent to the dashboard for analytics

Machine Learning High level Algorithm

1) We will first start off with detecting movements based on the data collected from our sensors at 20Hz and thresholding will be applied on the changes in value to detect for spikes. These spikes would indicate that a new dance move has started which we will then set a special flag on the Bluno Beetle that is to be sent to Ultra96 indicating the start of a new dance move.

2) The data will then be pre-processed via software implementation to minimise noise. Data is then divided into 5 second time windows so that each of them is analysed separately and sequentially. The data will be processed according to selected features, and then passed to the selected machine learning model (SVM/KNN/Neural Network), which will then classify the activity.

3) To train the model, K-fold cross validation will be used to split the data into training and test sets. After training and testing the model, evaluation metrics such as accuracy, precision, recall and f1 score and a confusion matrix will be used to evaluate the model.

4) The results, which consists of the predicted dance move and relative position of the dancers generated by the machine learning algorithm will then be sent to the evaluation and dashboard servers via a Python socket library.

Section 3 Hardware Details

Section 3.1 Hardware Components/Devices

In this section, we will be listing the hardware components/devices required of our wearable device. The supporting components required of each hardware component/device will be listed alongside the datasheet of the hardware component/device itself.

1. Bluno Beetle V1.1:

The Bluno Beetle is an Arduino Uno based board with bluetooth functionalities. The purpose of the Bluno Beetle is to gather the output from the connected sensors as input and process these inputs into output required of our machine learning models to classify the information. Bluetooth will be used for communication between the Bluno Beetle and Ultra96. [2]

Supporting Components: NIL
Datasheet:

- https://wiki.dfrobot.com/Bluno_Beetle_SKU_DFR0339
- https://datasheet4u.com/datasheet-parts/ATmega328P-datasheet.php?id=1057332

2. GY-521 (MPU 6050) IMU:

The GY-521 (MPU 6050) IMU enables us to get the relative position of the object attached to it from its initial position which in our design would be the dancer's hand, more specifically the wrist. It uses an accelerometer and a gyroscope to gather the data in the 3-dimensional space. The changes in the value of the data in the 3-dimensional space will enable us to determine which move has been performed by the dancer.

Supporting Components: NIL
Datasheet:

- https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
- https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

3. MyoWare Muscle Sensor:

The MyoWare Muscle Sensor enables us to measure muscle activation via Electromyography (EMG). [3] It allows us to gather data regarding an individual's muscle activity and we will then be able to tell the fatigue level of the individual's muscle alongside the improvement, if any, across the weeks.

Supporting Components: NIL
Datasheet:

- https://cdn.sparkfun.com/datasheets/Sensors/Biometric/MyowareUserManualAT-04-001.pdf

4. Ultra96-V2:

The Ultra96 contains the Xilinz Zynq UltraScale + MPSoC ZU3EG. This FPGA is used to perform machine learning algorithms on sensor data. The bluetooth feature of the Ultra96 will be used to interact with the Bluno Beetles to obtain sensor data to be used in the FPGA. The Linux environment in the Ultra96 will enable information to be sent through wifi from the Ultra96 to a remote dashboard for data analytics.

Supporting Components: NIL
Product Brief: http://zedboard.org/sites/default/files/product_briefs/5354-pb-ultra96-v3b.pdf

Section 3.2 Pin Table

In this section, we will be providing the pinout diagram of each individual hardware component/device used in our wearable design. The connections of the pins between the individual hardware components will also be listed in this section. Each wearable set will consist of two GY-521 (MPU 6050) IMU alongside a Bluno Beetle and a special wearable set will come with the MyoWare Muscle Sensor to gather EMG data of an individual.

1. Bluno Beetle to GY-521 (MPU 6050) IMU:
The following are screenshots to the pinout diagrams of the Bluno Beetle and GY-521 (MPU 6050) IMU and a table to show the connection from Bluno Beetle to GY-521 (MPU 6050). We will be using two GY-521 (MPU 6050) per Bluno Beetle and the connections will be shown across the table.

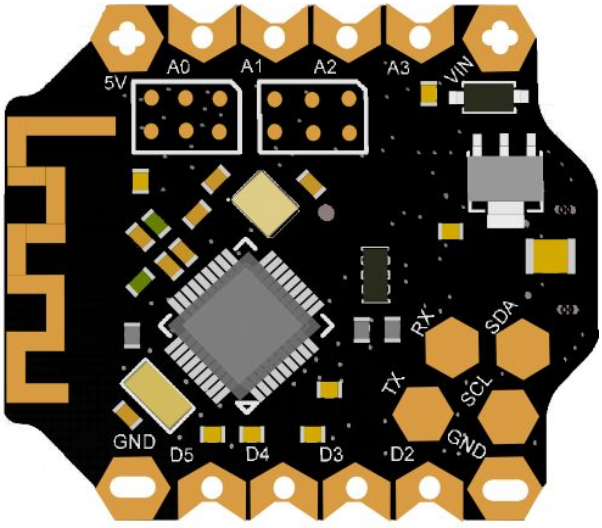

Figure 6: Bluno Beetle Pinout diagram (Adapted from https://wiki.dfrobot.com/Bluno_Beetle_SKU_DFR0339)

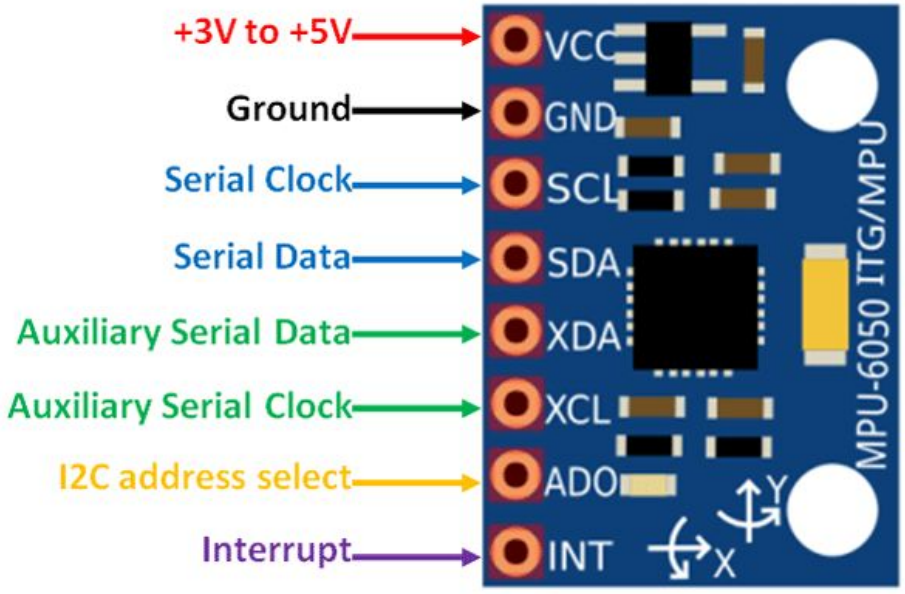

Figure 7: MPU6050 Pinout diagram (Adapted from https://components101.com/sensors/mpu6050-module)

Bluno Beetle	GY-521 (MPU 6050) - 1	GY-521 (MPU 6050) - 2
5V	VCC	VCC
GND	GND	GND
D2	INT	-
D3	-	INT
SCL	SCL	SCL
SDA	SDA	SDA

Pin table for connections between Bluno Beetle and two GY-521 (MPU 6050), this connection will be consistent across the three sets of wearables

2. Bluno Beetle to MyoWare Muscle Sensor:

The following is a screenshot to the pinout diagram of the MyoWare Muscle Sensor and a table to show the connection of the Bluno Beetle to the MyoWare Muscle Sensor. The MyoWare Muscle Sensor will only be attached to one of the three sets of wearables.

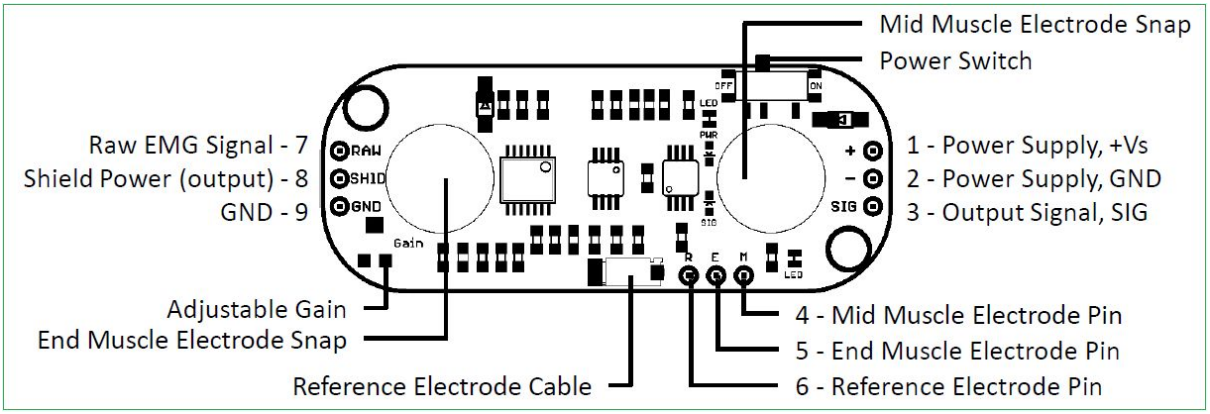

Figure 8: MyoWare Muscle Sensor Pinout diagram (Adapted from http://www.theorycircuit.com/myoware-muscle-sensor-interfacing-arduino/)

Bluno Beetle	MyoWare Muscle Sensor
5V	1 - Power Supply, +Vs
GND	2 - Power Supply, GND
A3	3 - Output Signal, SIG

Pin table for connections between Bluno Beetle and MyoWare Muscle Sensor, this connection will only be applicable to one of the three sets of wearables

Section 3.3 Schematics

In this section, we will be providing the schematics of the special wearable set that includes the MyoWare Muscle Sensor connected to the Bluno Beetle alongside two GY-521 (MPU 6050) IMU. The other wearable sets will be connected similar to this special wearable set but without the MyoWare Muscle Sensor. The following is a screenshot of the proposed schematics for the special wearable set.

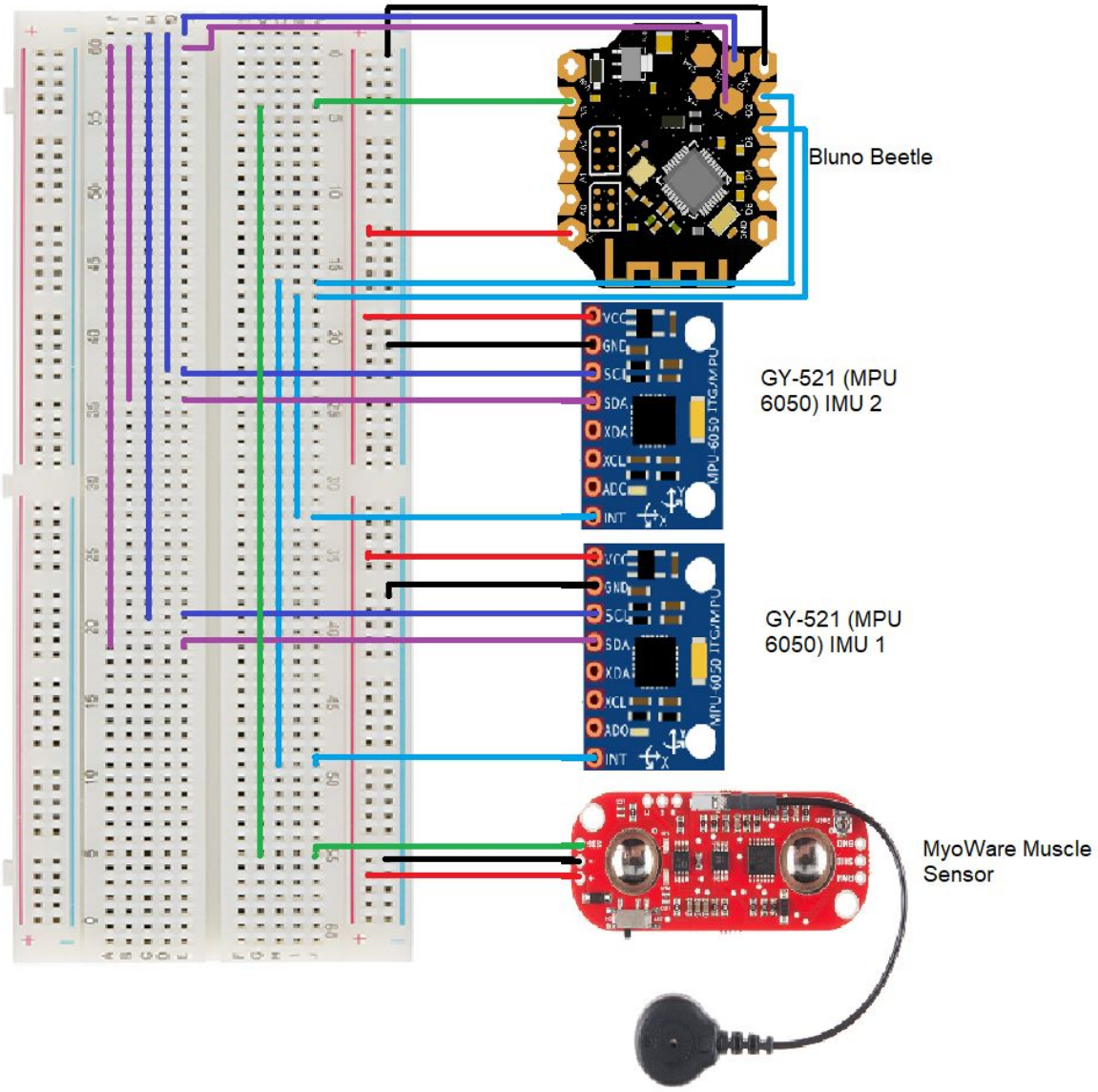

Figure 9: Schematics of the special wearable set that includes the MyoWare Muscle Sensor

Section 3.4 Operational voltage and current

In this section, the operational voltage and current of each individual hardware component/device will be listed.

1. Bluno Beetle:
   - Operational Voltage: 5V
   - Operational Current: 0.2mA

2. GY-521 (MPU 6050) IMU:
   - Operational Voltage: 2.375V to 3.46V
   - Operational Current: 3.8mA

3. MyoWare Muscle Sensor:
   - Operational Voltage: 3.3V or 5V
   - Operational Current: 9mA

4. Ultra96-V2 Development Board:
   - Operational Voltage: 12V
   - Operational Current: 2A - 4A

We decided to use capacitors to ensure that the hardware components will be provided their operational voltage. Additionally, we will be using current regulator circuits to regulate the currents flowing through the components to ensure no surges in the current will occur.

Section 3.5 Algorithms and Libraries

In this section, we will be listing the algorithms and/or libraries we intend to use on Bluno Beetle to gather data from each individual sensor.

1. GY-521 (MPU 6050) IMU:
The main library intended for use for the GY-521 (MPU 6050) IMU will be the I2Cdevlib by jrowberg which can be obtained from https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050. This library will be used in our codes to gather data from the GY-521 (MPU 6050) IMU.

2. MyoWare Muscle Sensor:
The library intended for use for the MyoWare Muscle Sensor will be the Arduino's <Servo.h> library which is part of Arduino's library packages. The information regarding this package can be obtained from https://www.arduino.cc/en/reference/servo.

Section 3.6 Hardware Acceleration of Quantized Neural Network

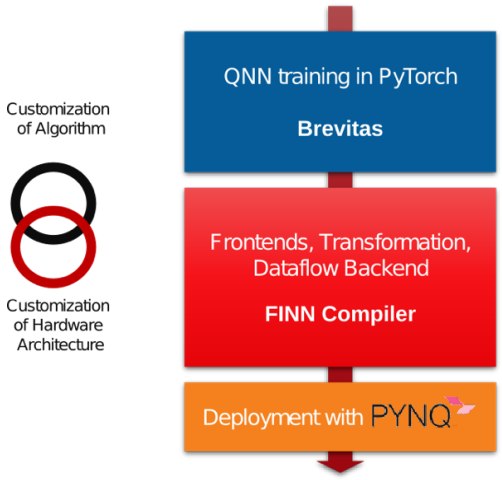

Figure 10: FPGA Neural Network Development Workflow with Brevitas and FINN Compiler

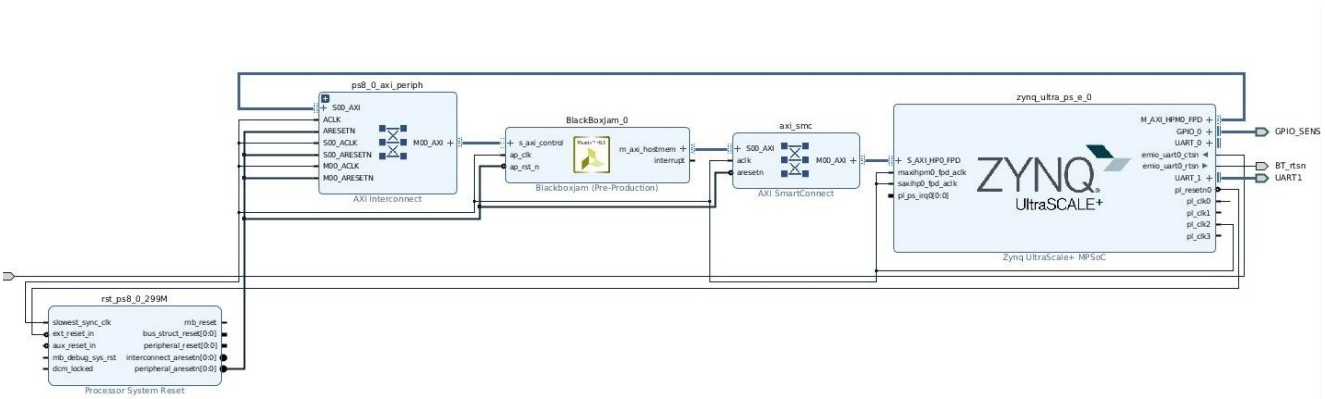

Figure 11: An example Block Design generated by FINN Compiler

We will be implementing a basic feed-forward neural network with backpropagation for activity recognition. The implementation details of the neural network will be explained in further detail under section 5. For our implementation of a hardware acceleration we will be experimenting with a quantized neural network which comes with advantages such as the utilization of fixed point arithmetic, which is faster than floating point arithmetic, reduction in power consumption and an improvement in memory usage due to the lower requirements to store the data.

The FPGA mapping of the neural network will be generated with Brevitas, a Pytorch library for quantization aware training and the FINN Compiler to generate the HLS Library calls which can then be used to generate a bitstream to map the neural network onto the FPGA.

Section 3.7 Hardware Acceleration of Support Vector Machine(SVM)

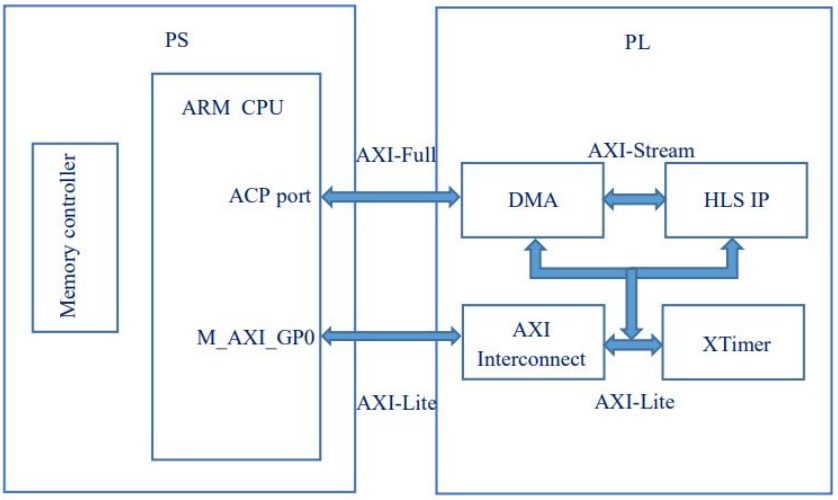

Figure 12: Block Diagram for SVM on a Zynq Device [18]

Our plans to implement a hardware accelerated SVM is based on this research paper[18] titled “Hardware Acceleration of SVM-based multiplier for Melanoma Images”, and the above image is a proposed block diagram for SVM on a Zynq Device described within the paper. Within this research paper, it is stated that the most time-consuming part of the classification algorithm of SVM is the summation of the multiplication of vectors that exists in the main decision function, and the focus of their hardware acceleration is on the speeding up of this computation. Therefore, we will also be exploring in this direction for our implementation of hardware acceleration on SVM.

Section 3.8 Hardware Acceleration Evaluation Metrics

When evaluating the design of our machine learning algorithms on the FPGA board, we will be looking at the speed and accuracy of our implementation. We will make a comparison between both machine learning algorithms on the FPGA board to determine which is a better algorithm for our use case, and we will also compare both the software and hardware implementation for both algorithms to verify the improvements we can get from hardware acceleration.

To measure the speed of execution of the machine learning algorithms, we will simply put a basic harness around the relevant code describing the machine learning algorithm so as to get the execution time for the retrieval of the inference result. We will be getting the total execution time across a fixed set of inputs, and then we will use that to get the improvements in execution time with and without hardware acceleration. Our initial datasets will be retrieved online from websites such as kaggle before moving on to our own datasets collected with our sensors.

For the accuracy of a hardware accelerated machine learning algorithm, we will be using the accuracy of each software implementation of the respective algorithm as a baseline and then compare it with our hardware accelerated implementation. For the implementation of a quantized neural network, several research papers have indicated that utilizing a quantized neural network with the appropriate bitwidth will not result in a greate degradation in accuracy of predictions. Therefore, we will also be expecting a similar result during our implementation.

However we will still need to verify that quantization works on our neural network design and one of our areas of interest will be how much we can quantize the weights and activations in each layer of our neural network while maintaining a reasonable accuracy during the validation phase.

Section 3.9 Hardware Acceleration potential issue and optimization

In this section, we will be discussing the potential issue that we might face during the implementation of hardware acceleration and the potential optimizations to help us resolve these problems.

One potential issue that we considered is the limited amount of hardware resources available to us on the FPGA. For example, we might face a problem with the mapping of the whole neural network onto the FPGA, especially if our Neural Network is large with many nodes and layers, or if we are unable to achieve good quantization with reasonable accuracy. If such issues arises, then we might need to prioritize the mapping of certain neural network computations over the others onto the FPGA, allowing part of the computation to be done on the FPGA and allowing the less important computations to be done within the ARM processor on the Ultra96-V2 development board. This might not be a trivial problem to solve as we will need to do a in depth analysis of the neural network architecture to find out what kind of computations can be accelerated more effectively than the others. This will require us to find out what is the most time and resource consuming part of the algorithms, whose acceleration will provide us with the greatest improvements in speed.

The above mentioned issue is however just a hypothesis and more research and testing needs to be done to determine if its actually ideal to split an machine learning algorithm to run on both ARM processor and FPGA. One potential issue we foresee with the splitting a machine learning algorithm is the extra latency added from the transaction of data from the ARM processor to the FPGA and vice versa, which could also make the improvements in computational speed by the FPGA minimal.

On the flip side, if there are sufficient hardware resources, we are also considering the possibility of mapping multiple of the same machine algorithm onto the FPGA to parallelize the inference of sensor data. However, this will also require further study as the bottleneck in

the speed of our system might not exist on machine learning algorithms but in other components of the system and parallelizing the inference phase by the Neural Network on the FPGA will not provide any improvements in speed, and we are not sure if this functionality can be easily supported in PYNQ.

To conclude this section, the above mentioned issues and optimization are all hypotheses and our team has limited understanding on whether the provided Ultra96-V2 development board has sufficient configurable components to handle the needs of our system. As such, more testing and research needs to be done to determine if any of the above issues will arise and whether there will even be a need to perform optimizations on the mapping of the machine learning algorithms onto the FPGA fabric.

Furthermore, we are already utilizing the Brevitas library and FINN Compiler to help us generate the Quantized Neural Network onto the FPGA, and so there might not be much room for us to improve the design of the neural network except at the algorithmic level since that will require us to delve deep into how Brevitas and the FINN Compiler is implemented.

Section 4 Firmware & Communication Details

Section 4.1 Real-time scheduling: FreeRTOS

To make sure that the main task of reading sensor data, processing them and sending them to Ultra96 can be scheduled to run at the intervals that we have decided upon, we have decided to use the FreeRTOS library which is included in the Arduino sketches.

Section 4.2 Processes

Communication Processes on Bluno Beetles			
Number	Name	Purpose	Priority
1	Handshake and clock synchronization	To create a connection between the Beetle and Ultra96 and calibrate time on beetles to synchronize with Ultra96	N/A (only once at the start)
2	Read, process and transmit sensor data to Ultra96	To read data produced by the sensors whenever a dance move has started. To ensure correct formatting of packet structure into a standard that can be replicated in Ultra96's side for a smooth transfer of data	Highest

Handshaking and clock synchronization processes are to be initialized in the setup stage and only once before the main task is created and running. Both of these processes will be suspended and not be run by the task scheduler once they have been completed.

The task in which the reading of sensor data, processing of sensor data and sending of data to Ultra96 is to be done, will be set to run once every fixed interval of time (50 ms) to capture sufficient amounts of sensor data during the whole duration of the dance move for accurate machine learning prediction. The gaps of time between running of the task when the task is not running will lead to a reduction of power consumed by the Beetles.

Communication Processes on Ultra96			
Number	Name	Purpose	Priority
1	Connecting to Beetles, handshaking and clock synchronization	To establish a stable connection between Ultra96 and calibrate time on beetles to synchronize with Ultra96	N/A (only once at the start)
2	Receive data	To initialize the process of receiving Beetle	Highest

	from 3 Beetles (3 similar threads with same priority will be created)	packets. The data will then be extracted from the packets and fed into the machine learning algorithms.	
3	Use data for machine learning algorithms	To feed all relevant data into machine learning models and compute the most likely dance move and relative positions.	High
4	Connect to evaluation server and dashboard server	Send our results to the evaluation server for evaluation. And to analyze dancers movement and dashboard visualization, data should also be transmitted to dashboard server.	Medium

Section 4.3 Multithreading vs Multiprocessing

A consideration has been made between choosing whether to go down the multithreading or multiprocessing route when it comes to the design of processes on Ultra96. Certainly, with the presence of multiple processing cores on Ultra96, overheads spent on processing python code would be greatly reduced. However, there are practical constraints to be considered, and these constraints ultimately make it hard to justify the usage of multiprocessing over multithreading. For example, the sections of code (e.g functions) to multiprocess would depend on whether or not it relies on data processed by other code sections. Also, since Ultra96 would need to receive data from the Beetles before they can be passed to the machine learning algorithm and the results need to be generated from the machine learning algorithm before they can be sent to the evaluation and dashboard servers, the processes detailed above would need to be run in an ordered manner. Thus, the multiprocessing framework would break down in this context. It is also a consequence of the fact that in multiprocessing, there is no guarantee that the first process to be created will be the first to start or complete.

For synchronization between processes in Ultra96, we have decided to use Python's threading module. Since the initial process of handshaking and successful establishing of connections must be done before any transfer of data can take place between Ultra96 and the Beetles, we have decided to use the condition object synchronization primitive. It uses an underlying RLock (re-entrant lock) primitive as an object during its construction as RLock allows the handshake thread to recursively call itself in the event that stable connections cannot be established between Ultra96 and the Beetles on the first try.

As for the process of receiving data from the 3 Beetles, we will be creating 3 threads with the same priority, each receiving data from one Beetle, and with each relying on the condition

object created earlier together with the underlying RLock object as these 3 threads will be sharing the same RLock with all the other processes in Ultra96. After the handshaking is done, process 1 will call notify and wake up the 3 receiving data threads. The main thread which contains the handshaking process will call wait to release the RLock needed for the 3 receiving data threads to acquire. After all data is collected, the 3 data threads will call notify and wait, similar to what the main thread did previously. The same synchronization mechanism will also be applied to the machine learning threads and evaluation/dashboard server thread, with the exception that now, only the receive data thread will acquire the RLock and not the main thread as handshaking is already done previously.

Section 4.4 Setup and configuration for BLE interfaces

Setting up and configuring the BLE interfaces involves four commands. The first command, "echo BT_POWER_UP > /dev/wilc_bt", powers up the ATWILC300 bluetooth chip, and indicates that the BT requires the chip to be "ON". The second command, "echo BT_DOWNLOAD_FW > /dev/wilc_bt", downloads the BT firmware to the ATWILC device's Intelligent Random Access Memory (IRAM) using Secure Digital Input Output (SDIO). The third command, "echo BT_FW_CHIP_WAKEUP > /dev/wilc_bt", acts to prevent the chip from sleeping. The last command, "hciattach /dev/ttyPS1 -t 10 any 115200 noflow nosleep", is to attach serial UART to bluetooth stack as HCI transport interface. The first argument, "/dev/ttyPS1", specifies the serial device to attach which is PS1. The second, "-t 10", specifies an initialization timeout of 10 seconds. The fourth, "115200", is the baud rate which specifies the UART speed to use. "noflow" means no hardware flow control is forced on the serial link.

Section 4.5 Protocol to coordinate Beetles and Ultra96

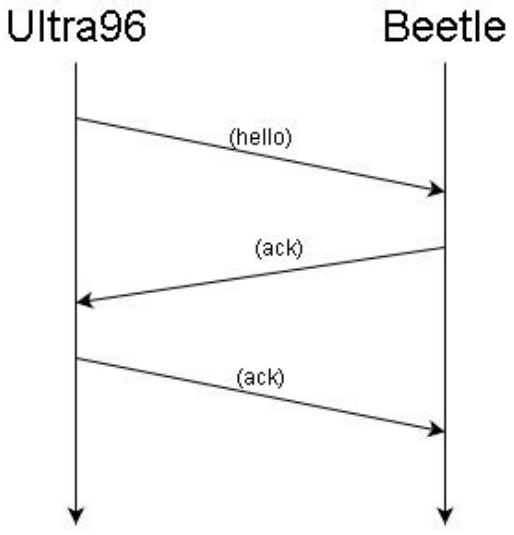

Figure 13: 3-way handshake

The Ultra96 will initialize the 3-way handshake protocol by first sending a “HELLO” packet to the individual Bluno Beetles. Ultra96 will then wait to receive “ACK” packets from the Bluno Beetles as a form of acknowledgement. Whenever Ultra96 receives “ACK” from any Bluno Beetles, it will respond by sending back a “ACK” packet back to the same Bluno Beetle. At this point in time, it is not sufficient for Ultra96 to be able to establish a connection with just one Beetle and not all Beetles utilized in our system. The subsequent processes in Ultra96 downstream will be blocked indefinitely until the condition of establishing stable connections with all Beetles involved is met.

Packet Type	Packet Code
HELLO	‘H’
ACK	‘A’
NAK	‘N’
DATA	‘D’
TIME	‘T’

Defined above are the packet types and their respective codes so that both the Beetle and Ultra96 knows what sort of packets are transmitted between each other. On top of handshaking, “ACK” packets will also be used by Ultra96 to send to the Beetles if it is able to receive complete and valid data packets with a correct checksum value. “NAK” packets will be used by Ultra96 to send to the Beetles if it receives incomplete or invalid data packets or wrong checksum values. ‘DATA’ packets will be used for denoting transmitted sensor data from Beetles to Ultra96. ‘TIME’ packets will be used for sending timestamps between the Ultra96 and the Beetles for clock synchronization.

Packet format (Time calibration packet)					
Packet_type (char)	ID (char)	Timestamp (long)	Timestamp (long)	Timestamp (long)	Timestamp (long)
Total size:18 bytes					

The above defined packet format is for sending timestamp data between Ultra96 and the Beetles to determine the clock offset and transmission delay between the internal clocks of the Beetles and Ultra96 for clock synchronization.

Packet format (Sensor data)					
Packet_type (char)	Timestamp (long)	Start (char)	IMU1[6] (float)	IMU2[6] (float)	Checksum[4] (char)
Total size:58 bytes					

The baud rate specifies how fast data is sent over a serial line. Both the Beetle and Ultra96 needs to have the same baud rate configured so that both send and receive data at the same speed. A baud rate of 115200 bps has been chosen as the standard for our serial communication between the Beetle and Ultra96. It has a fast transmission speed ensuring low latency of communication and yet, not too high that it causes invalid data packets to be received on the receiving end.

Section 4.6 Communication between Ultra96 and evaluation server

Section 4.6.1 Process Overview

- The evaluation server should run first and create a socket and then waits for the client.
- The position and action are inferred from machine learning algorithms on Ultra96.
- Synchronization delay is analyzed and calculated from timestamps obtained from 3 Beetles.
- Data is packed into the format '#position|action|syncdelay|' and encrypted using AES encryption scheme with some secret key for secure communication.
- Encrypted data is sent to the evaluation server by calling client.py to connect client-side TCP to the server TCP.
- The evaluation server receives data and decrypts the message following AES scheme using the same secret key.
- On the client (Ultra96) side, if the action is 'logout', the client closes its TCP connection to the server.

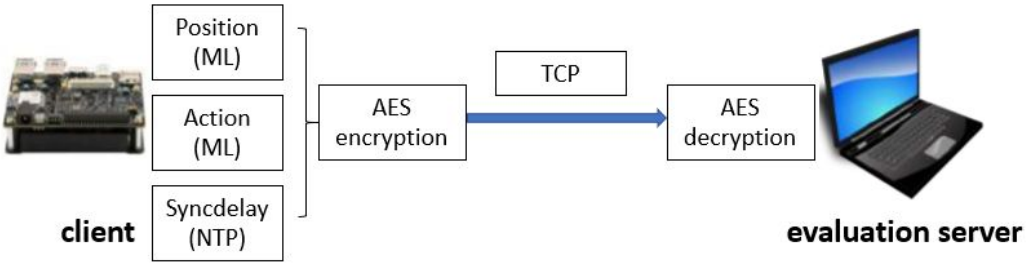

Figure 14: Communication between Ultra96 and evaluation server

Section 4.6.2 Message Format

#position|action|syncdelay|
Positions can be ['1 2 3', '3 2 1', '2 3 1', '3 1 2', '1 3 2', '2 1 3']
Actions can be ['muscle', 'weightlifting', 'shoutout', 'dumbbells', 'tornado', 'facewipe', 'pacman', 'shootingstar', 'logout']

Section 4.7 Communication between Ultra96 and dashboard server

Section 4.7.1 Process Overview

- The client (Ultra96) sends encrypted sensor data using TCP to the dashboard server by specifying host address and port number .
- The user of the dashboard needs to decrypt data and then process them for storage or analyses.

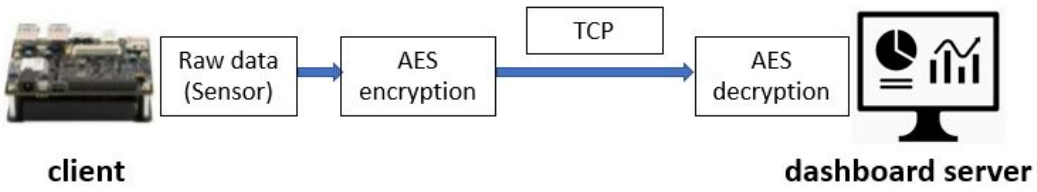

Figure 15: Communication between Ultra96 and dashboard server

Section 4.7.2 Message Format

All raw data from sensors
#data1|data2|data3|...|
(what data to use depends on further experiments)

Section 4.8 Communication Implementation

A socket is one endpoint of a two-way communication link. A socket is a combination of a port number so that the TCP layer can identify the other application. An endpoint is a combination of an IP address and a port number (Docs.oracle.com, 2020).

For communication between Ultra96 and evaluation/dashboard servers, we use python socket library.
Libraries used: socket.socket, socket.connect, socket.sendall, socket.bind, socket.listen, socket.accept…

Section 4.9 Secure Communication

As for secure communication between Ultra96 and the evaluation/dashboard server, we use Advanced Encryption Standard (AES) scheme for secure communication to protect the users' privacy. AES is quite secure and is fast which is suitable for our product (Quora.com, 2020).

AES has 3 types AES-128, AES-192 and AES-256 depends on how many bits the key is. The longer the key is, the more secure the encryption is and the slower the encryption process can be. For our product, we adopt AES-128 because it provides sufficient security and it is faster (sluiter, 2019). So, when running our client.py, a secret key of length 16 is provided (16 chars = 16 bytes = 128 bits).

Section 4.9.1 AES Encryption and Decryption Scheme

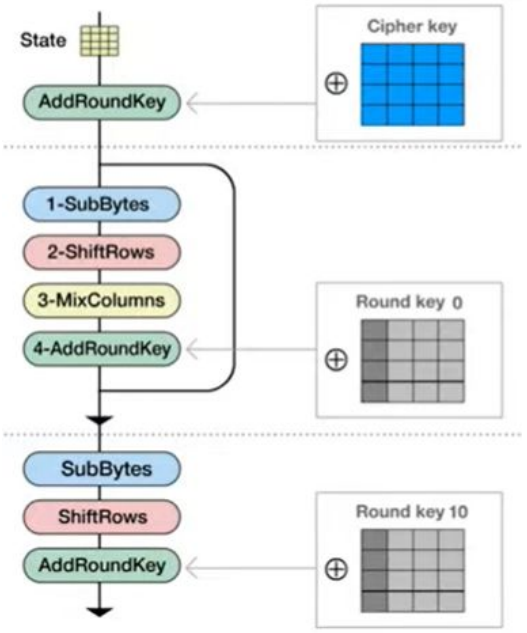

Figure 16: AES Encryption and Decryption Scheme

To encrypt the text, in the first step, the original text needs to perform XOR operation with computed round key from the secret key. Then in the SubBytes process, Rijndael S-box lookup table is used to replace the blocks in the state. Next, the rows in the state are shifted and the columns are mixed by multiplying them with some matrices. Finally the round key is added to the state and the result is the ciphertext.

Decryption of ciphertext is done through inverse processes of the above encryption processes.

Section 4.9.2 AES Encryption Implementation

Library APIs used: Crypto.Cipher.AES, Crypto.Random

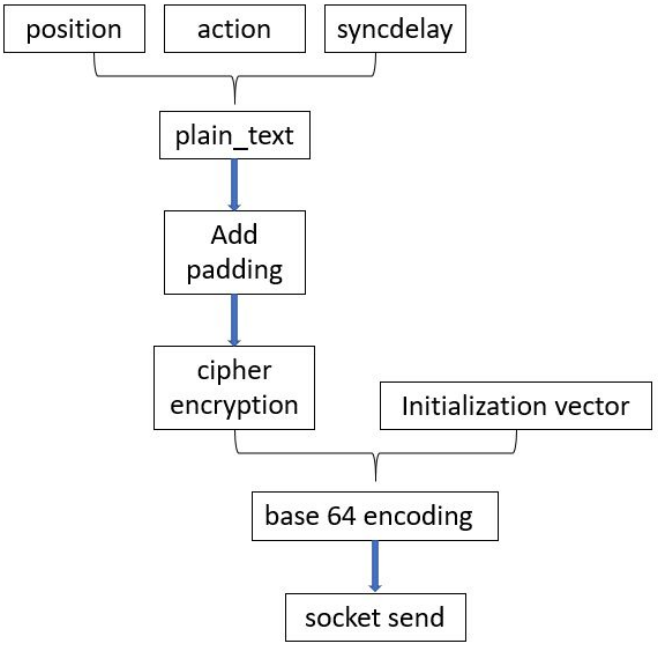

Figure 17: AES Encryption Implementation

Section 4.10 Clock Synchronization Protocols

Section 4.10.1 Time Calibration

Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems (En.wikipedia.org, 2020). In our design, we use a simplified version of NTP and take Ultra96 time as the reference clock because there is only 1 Ultra96.

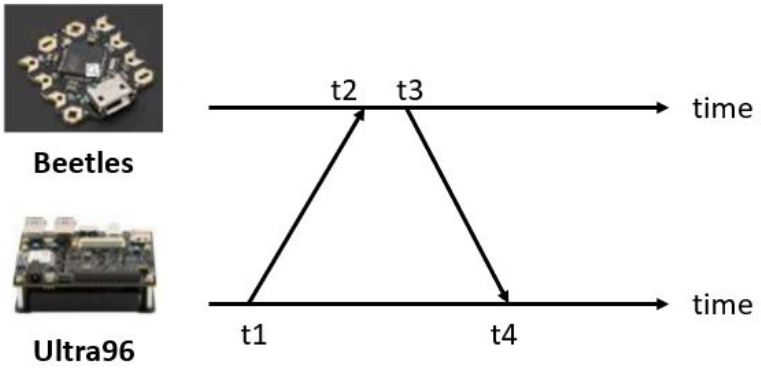

Figure 18: Time calibration

The time calibration starts when the Ultra96 sends a packet containing its timestamp (t1) to Ultra96. When the Beetle receives the packet, it adds its own timestamp (t2) and a transmit timestamp (t3) to the packet, and sends it back to the Ultra96. When the Ultra96 receives the packet it will record its receipt time (t4) in order to estimate the clock offset (AG, 2020). The formula for calculating clock offset is the following:

$$RTT = (t4 - t1) - (t3 - t2)$$

One-way communication delay = RTT/2

Clock offset = t2 - t1 - RTT/2

For time synchronization, we have 2 plans. Plan A is that we only do this calibration at the beginning and then stick to the same clock offset and communication delay in the following evaluations. Plan B is that we calibrate every polling interval and we will need to do some experiments to determine the most suitable polling interval to trade off between speed and accuracy. Short polling intervals update the parameters frequently but are sensitive to jitter and random errors. Long intervals may require larger corrections with significant errors between the updates (Ntp.org, 2020). Normal operating system uses 1024s as the polling interval.

Section 4.10.2 Asynchrony between Dancers

Once the dancer starts to move, the Beetles set flags and send flag packet with local timestamp of Beetles (local clock time) to Ultra96. Base on the clock offset and communication delay, the asynchrony between dancers is calculated on Ultra96:

Flag packet time of Beetle 1, 2, 3 are t1, t2, t3

Clock offset of Beetle 1, 2, 3 are o1, o2, o3

Start time of actions on Beetle 1, 2, 3 (Ultra96 time): t1+o1, t2+o2, t3+o3

Asynchrony between the dancers: max(t1+o1, t2+o2, t3+o3) - min(t1+o1, t2+o2, t3+o3)

Section 4.10.3 Packet Format

We design the following packet format.

Packet format (Time calibration packet)				
ID (char)	Timestamp (long)	Timestamp (long)	Timestamp (long)	Timestamp (long)
Total size:33 bytes				

The ID can be ‘A’, ‘B’ or ‘C’ which distinguishes the 3 Beetles. During message transmission between Ultra96 and Beetles, timestamps are appended to the packet. Based on the timestamps, Ultra96 is able to calculate the asynchrony between the dancers.

Section 5 Software Details

Section 5.1 Segmenting the sensed data

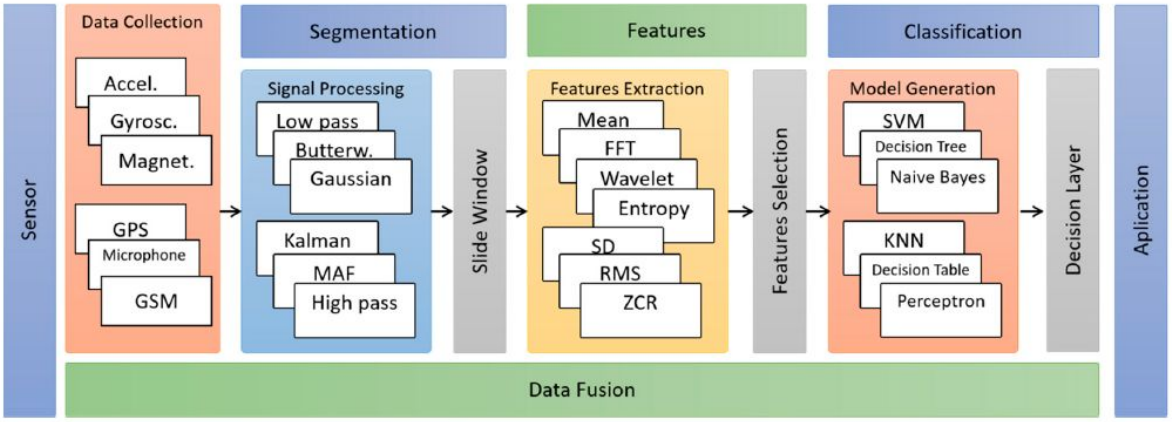

Figure 19: Overall machine learning process

Segmentation is the process of separating data into meaningful subgroups that share the same traits. Subgroups are represented by signal segments in a given time interval. The objective of segmentation is for each segment to contain enough characteristics to allow recognition of a dance move at a specific moment in time.

The sensor data will be collected at 20Hz. Studies have shown that frequencies of 20Hz contain enough information about physical activities, and thus it is believed that this would be sufficient to capture enough information about dance moves.

The data will then be pre-processed to reduce the noise caused by environment, movements and changes in user behaviour while performing the dance moves. A survey showed that smartphone based human activity recognition used filters such as the low pass filter, Butterworth filter, Kalman filter and the Moving Average filter. Software packages will be used to implement these filters. An example for a software implementation of the Butterworth filter is a Python package which can be found at:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html

Sensor data will be divided into consecutive segments so that each one of them can be analysed separately and sequentially. These segments are known as time windows. We will be using sliding window-based segmentation. Sliding window-based segmentation divides sensor data into subsequences over time, as shown in Fig 20 below. The start and end of the time windows could be the changes in signal mean over time. This is called the Page Hinkley technique, and solves the problem related to mixed sensor data from multiple activities in the

same time window. Initially, it is decided that 50% of the frames will overlap, but this will change after further testing of our model.

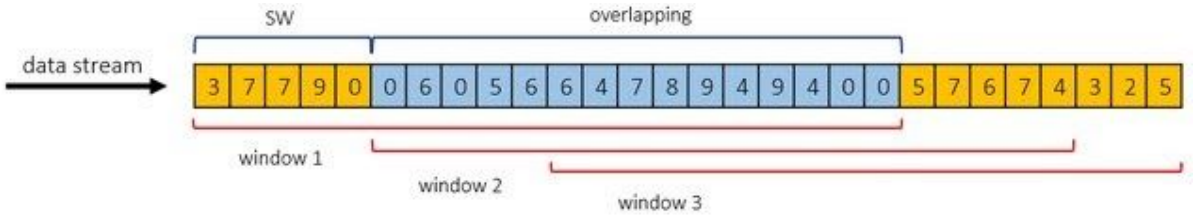

Figure 20: Sliding window-based segmentation

Section 5.2 Features to be explored

Features will be explored on the time domain and the frequency domain.

In the time domain, we can utilise statistical and non statistical functions. Statistical functions include mean, variance, median while non statistical functions include areas and bins distribution. The magnitude of x, y and z values can be used to assess level of movement intensity, and it is independent of the orientation of sensors on the user's body.

In the frequency domain, we can explore features such as the spectral energy, entropy, peak frequency and power of the signal. This is obtained via Fourier transform, which can be useful for capturing repetitive patterns of signals in terms of frequency. These features however, are dependent on the orientation of the sensors on the user's body. The Fourier transform will be implemented via software, such as this Python library: https://docs.scipy.org/doc/numpy/reference/routines.fft.html

Feature extraction is necessary because raw data from the sensors is not appropriate for use by standard machine learning algorithms. Time domain features have a lower computational cost when compared to frequency domain features, but features in the frequency domain can better represent contextual information in terms of signal patterns. Thus, an appropriate number of features need to be picked from each domain so that it is not too computationally heavy while also maintaining a reliable level of accuracy.

Section 5.3 Machine learning models under consideration, and optimizing chosen machine learning algorithm

Support Vector Machines (SVMs) are widely used in classification problems. It finds a hyperplane in an N-dimensional space that has the maximum distance between the data points of 2 or more classes, N being the number of features. The classes will represent the

different types of dance moves. SVM is a huge consideration as it requires less computation power. The scikit-learn documentation for SVM can be found at: https://scikit-learn.org/stable/modules/svm.html

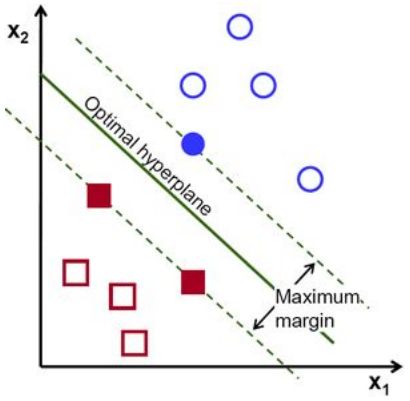

Figure 21: Visual representation of the SVM algorithm

Fig 21 above shows a case where the data can be classified linearly. However, for a dataset as shown in Fig 22 below, a linear classifier would not be suitable:

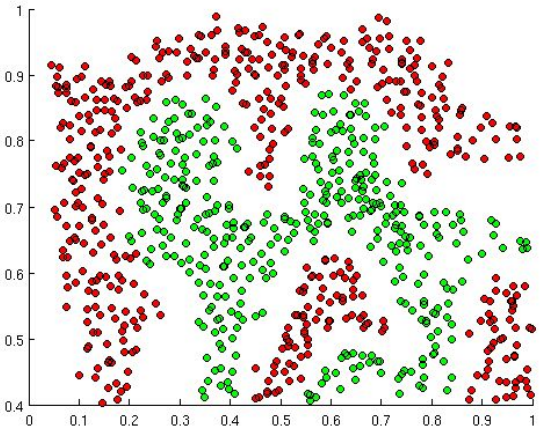

Figure 22: Dataset that cannot be classified linearly

In this case, a kernel for the SVM would have to be used. An appropriate kernel to use for this problem would be the Radial Basis Function (RBF) kernel:

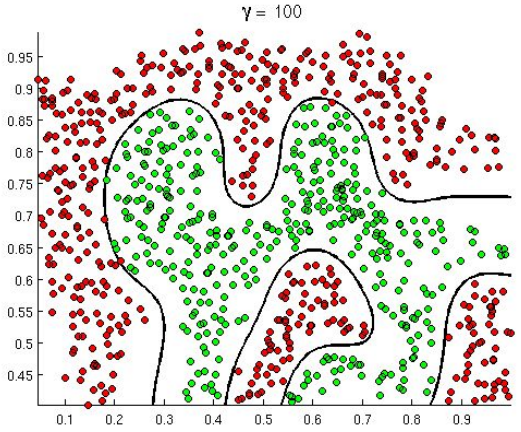

Figure 23: Dataset classified using the RBF kernel

The kernel can be specified in the scikit-learn library as documented in the link above.

The K Nearest Neighbors (KNN) algorithm assumes that in a dataset, the closer the data points are, they refer to the same thing. It commonly uses straight-line (Euclidean) distance to calculate the distance between the data points. It acquires the K nearest neighbors by finding the distances between a query and all the examples in the data, adds them to an ordered collection, sorts them in ascending order and picks the first K values from the sorted collection. The algorithm must be tried with different values of K, and the value of K which gives the lowest number of errors and hence stable predictions should be selected for the algorithm to be reliable.

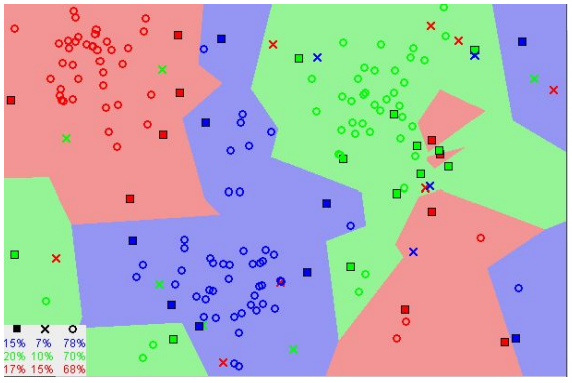

Figure 24: Visual representation of the KNN algorithm

The KNN algorithm is versatile as it delivers reliable results across different classification problems. However, it gets slower as the number of data points increases. Details for implementing KNN in scikit-learn can be found at:
https://scikit-learn.org/stable/modules/neighbors.html

A neural network is a machine learning algorithm that is able to capture complex patterns in datasets by using many hidden layers and non-linear functions associated with those layers. As shown in Fig 25 below, it takes in x1, x2, … , xN as inputs and outputs f(X). For our project, the inputs will be our datasets segmented according to features and the outputs will be the labels of dance moves according to the number (1, 2, … , 8). The scikit-learn implementation for feed-forward neural networks can be found at: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

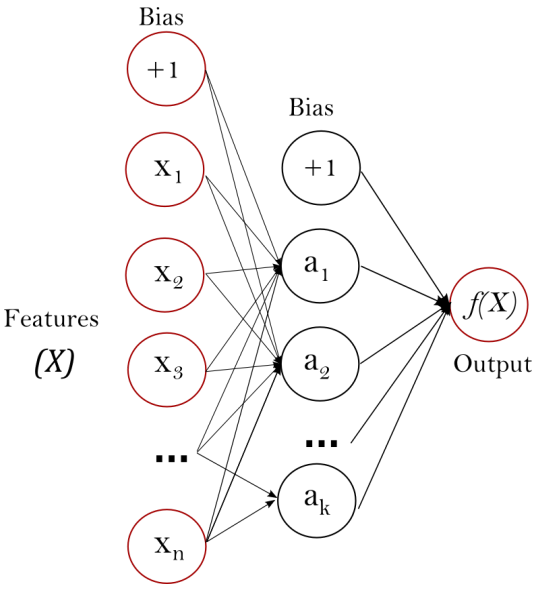

Figure 25: Feed-forward neural network

The number of middle hidden layers can be tweaked to obtain a model that is able to predict accurately and efficiently. It is also important to note that the labels (dance moves) must be given in number form for the scikit-learn implementation.

After trying the various machine learning models, the model that is able to predict reliably with minimal computing power will be chosen for our wearable device.

Section 5.4 Training and validating the model

There are various ways to train the model, and it is important to pick the right method for our project. There are many ways to split the model into testing and training sets: holdout method, leave one out cross validation (LOOCV) method and the K-fold cross validation method.

The holdout method involves splitting the data set into 2 sets: a training set and a test set. The proportion can be chosen as desired. For example, the dataset can be split into 70% for training and 30% for testing, as shown below:

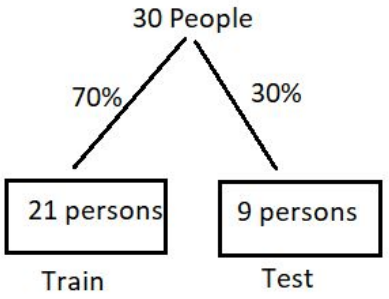

Figure 26: Holdout method

The holdout method is fairly simple to implement. However, it could result in the case where the the model makes better predictions on the training data but poorer predictions on the test data. This is caused by sample variability between training and test sets.

The cross validation method eliminates this problem. The data is divided into subsets, the model is trained on a few subsets and the other subsets are used to evaluate the model's performance. To reduce variability, many rounds of cross validation are performed with different subsets from the same data, and the results are combined to come up with an estimate of the model's prediction ability.

One type of cross validation is the Leave One Out Cross Validation (LOOCV) method. One observation is used for testing and the rest of the dataset is used for training the model. In essence, if the dataset contains n observations, then the training set has n-1 observations and the testing set has 1 observation. This is repeated until every observation has been used as the test set.

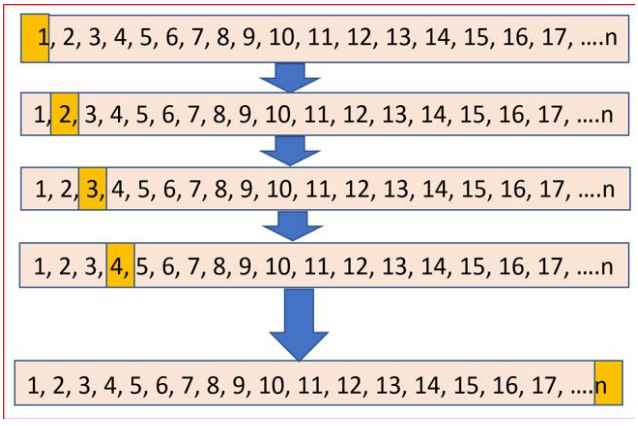

Figure 27: LOOCV method

The strongest benefit of the LOOCV method is that it has very little bias. However, it requires a high computational cost as the model has to be trained n times, n being the number of observations in the dataset. Another point to consider is that if the test observation is an outlier, then the variability of test results is high, which will affect evaluation metrics.

Another type of cross validation is the K-fold cross validation method. The dataset is randomly separated into k groups of about equal size. The first group is used for testing, the model is trained on

k-1 groups. This is repeated k times until every group has been used as the test set. This method is similar to LOOCV, except that k = n in the case of LOOCV.

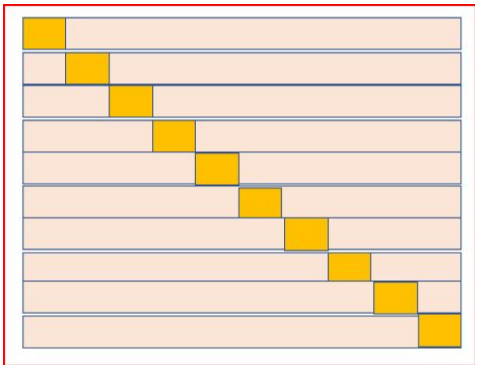

Figure 28: K-fold cross validation method

K-fold cross validation has less computational cost compared to LOOCV as the model has to be only trained k times, where k < n. There is also reduced bias as com
pared to the holdout method, and the variance of test set is reduced compared to LOOCV due to multiple observations in the test set. However, if the value of k is large, then there would be a high computational cost incurred, similar to the case of LOOCV. Hence, it is desirable to pick a value of k which has a good trade off between accuracy and computational cost.

Our team will be using the K-fold cross validation method to split our testing and training datasets as it gives a good balance of reliability and efficiency as compared to the other methods.

Section 5.5 Testing the trained model

After training the model, it is vital that we test it to see that it is making accurate predictions. To evaluate the performance of a model, we can use metrics that inform, in mathematical terms, how reliable the model is in detecting an activity. Examples of evaluation metrics include: accuracy, precision, recall and F1 score.

Accuracy is the fraction of correct results among the total number of cases.

Accuracy = number of correctly classified activities / total number of instances

Accuracy is a good choice of evaluation metric for classification problems which are well balanced. However, it treats every activity as equally important, and is inefficient in the case where there are imbalanced datasets.

Precision is the fraction of predicted positives among the true positives.

Precision = True Positives / (True Positives + False Positives)

Precision is looked at when we want to be very sure that the model's prediction is correct. In the case for our project, if we want to be very sure that the model predicts every dance move correctly, we should look at the precision metric.

Recall is the proportion of actual positives that is correctly classified.

Recall = True Positives / (True Positives + False Negatives)

Recall is useful when we want to capture as many positives as possible. For example, we should look at recall if we want our model to recognise a dance move even if it is not too sure.

F1 score is a tradeoff between precision and recall, and this makes it a very useful metric.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Formula for F1 score

In many instances, we want to have a model with both good precision and recall, so F1 is a useful metric in that regard. However, it gives equal weight to both precision and recall and there would be instances in which we would want to factor in more recall or more precision when looking at our metric. To do this, we can create a weighted F1 metric by adjusting β as shown below:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

Formula for weighted F1 metric

A confusion matrix will also be generated to assess the model's performance.

Section 5.6 Dashboard design and displayed analytics support

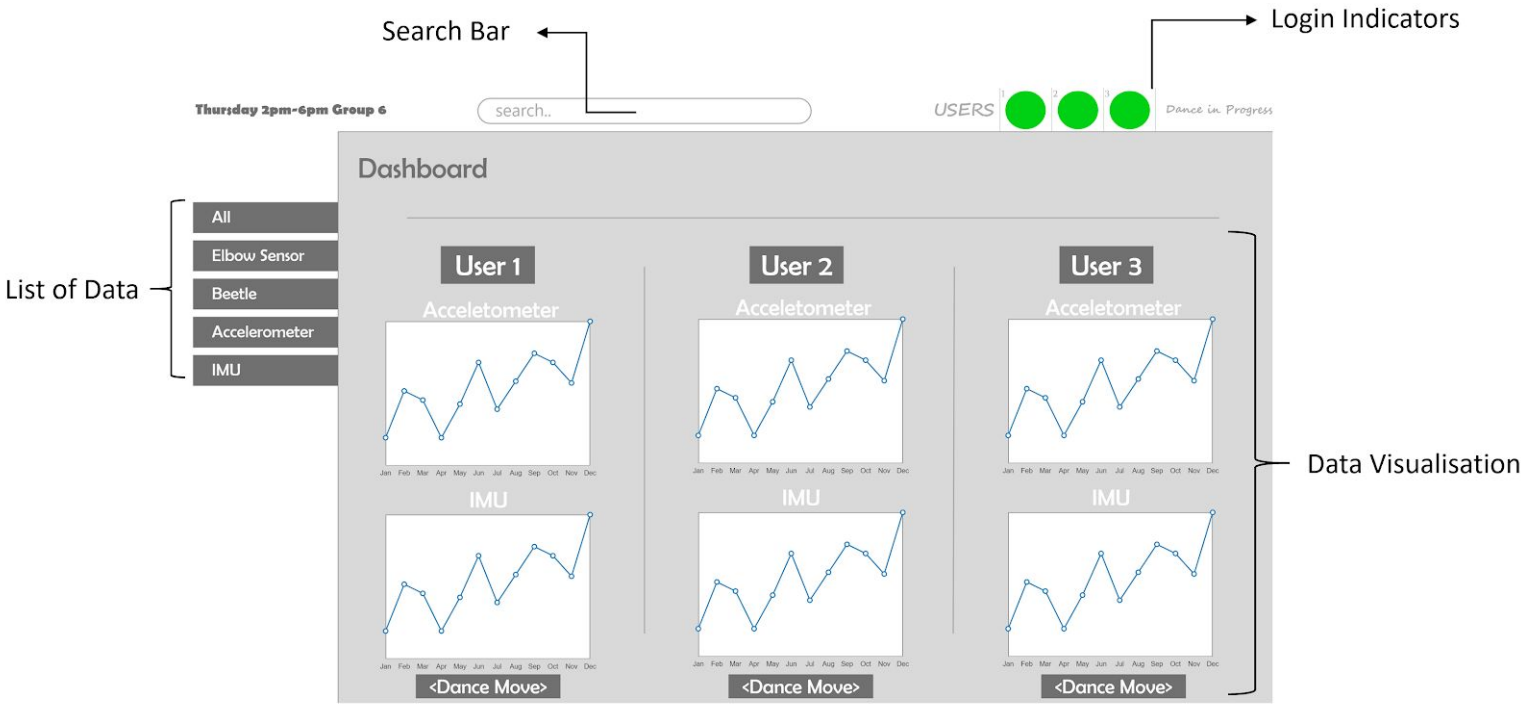

Figure 29: Mockup of Dashboard Design

Above is a mockup of the design dashboard. This was done in Adobe XD. The main features include:

No	Name	Features
1	Login Indicators	- Each green light represents a user. There are 3 green lights that represent 3 dancers. Before the Ultra96 and Beetles are connected, the lights will be red. Upon connecting, the lights will turn green. - On the right side of the lights, there is a text box to show display any system information.
2	List of Data	- Shows different sensor graphs that are available - Upon clicking on the relevant button, it will show the specific graph in the Data Visualisation area
3	Search Bar	The search bar is used to filter which sensor graph is to be shown. In the event, there is too much sensor data, the search bar would be a feature to have quick access to specific graphs

4	Data Visualisation	- The area that will show the graph with its corresponding name and the type of dance move that has been executed. - Further, there will be data analytics that will be present in this portion when deem fit. E.g. Dance moves executed with machine learning data explanation.

Section 5.7 Storage of incoming sensor data

Two database languages that work well with Dash library with python are MySQL and PostgreSQL. The Ultra96 will be sending packets to my laptop and the data will be stored in PostgreSQL database. The process of sending packets from the Ultra96 will be done by the external communications personale. On my laptop, there will be a server code written in python to receive data by sockets and write into a local PostgreSQL DB. All incoming sensor data will be posted to a remote laptop to store and used for data visualisation. Hence, the format briefly includes "#MachineLearningData, TestInput, AccelerometerValues, IMUValues…." This is to store all the relevant data that can be used for data analytics.

MySQL vs PostgreSQL

After discussions with team members we decided that PostgreSQL was better for the team as some members had used PostgreSQL for prior modules. Hence, MySQL will be a backup in the event that PostgreSQL poses issues.

Section 5.8 Approaching the issue of real-time streaming

Using Dash by Python enables real-time streaming, there are different tutorials online to build a dashboard. After testing it out for 2 hours, Dash seems to be an extremely fast method to produce real-time graphs. Dash is built on top of a Flask server, using React.JS, React and Plotly.JS to render the front-end. In pure python, a front-end dashboard can be created quickly.

An alternative would be to use a MERN stack. MERN stack being MongoDB, ExpressJS, React and NodeJS as a full stack application. Our team has some experience with a MERN stack, however, it is not beginner friendly and takes a lot of time to learn. Therefore, a more cost effective strategy was to use Dash.

Section 5.9 Logging

The system will also store a log of the current test inputs in the PostgreSQL table.

Section 5.10 Suggested Data Analytics

An insight from data analytics is to show the differences between same dance moves being executed but sensor readings may differ. For example, if all the dancers waved their hand for

a dance move, the ML process may output the same move, but there may be differences in sensor data for all the x-coordinates.

Section 6 Project Management Plan

Section 6.1 Team Structure

For this team project, we adopt egoless team structure. For each week, each of us takes turns to be the team leader. Team members will post our questions and discussion topics and the team leader is in charge of collecting the questions, managing progress for the week and hosting the lab meeting efficiently.

We will use Trello and Github issues to track the progress of each team member.

Section 6.2 Timeline

Week	Team	HW1	HW2	Comms1	Comms2	SW1	SW2
Week 3	Research	Research					
Week 4	Design Report	* Complete design report and research * Purchase the required hardware components	* Complete Report and research * Try running existing trained models on the Ultra96	*Complete Report and research *Setting up device scanning, and connecting a single Beetle to Ultra96	*Complete Report * Get socket code running on my laptop * Research on NTP for timing protocol and synchronizatio n	*Complete Report and research *Try running sample ML code in python on any dataset	*Complete Report and research on technologies to use for dashboard
Week 5	Prototypin g of individual component s	* Wire up all the hardware components into a wearable (1	* Discuss with SW1 regarding his research on ML	*Get a Beetle to be able to maintain stable connection	*Run socket code on Ultra96 *Design protocol on	*Obtain a suitable dataset modeling human	*Test code for database storage on MySQL *Find out

		set) * Get readings from sensors using Arduino Beetle	Algorithms to finalize their implementations * Try creating a simple neural network with Brevitas and FINN Compiler	and send data to Ultra96 for at least 1 minute	my laptop	activities *Apply machine learning libraries *Determine appropriate segmenting methods and features	what data points will be needed for the dashboard
Week 6	Progress Checkpoint First individual prototype test	* Get sensor readings for 2 - 4 dance moves * Wire up the other hardware components (the other 2 sets)	* Explore the mapping of a neural network onto the FPGA as demonstrated in https://www.youtube.com/watch?v=DoA8hKBltV4 * Try to implement a hardware accelerated variant of the first ML algorithm.	*Get 3 Beetles to be able to maintain stable concurrent connections and send data to Ultra96 for at least 1 minute	* Get timing protocol running between BLE and Ultra96	*Apply k-fold model validation *Output metrics of models - classification accuracy and confusion matrix *Determine relative location	*Integrate Dash and MySQL to run data that is sent from another computer
Recess week		Buffer time					
Week 7	Individual subcomponent test	* Ensure all hardware components are working as expected * Research to make power consumption more efficient	* Try to implement a hardware accelerated variant of the second ML algorithm. * Compare the hardware utilization,	*Ensure robust concurrent connections between 3 Beetles and Ultra96	*Ensure socket connection is reliable between Ultra96 and evaluation server *Ensure synchronization is correctly	*Ensure ML works efficiently on dataset and metrics are readily available *Review algorithm for relative location	*Integrate dashboard and DB with the Ultra96 to send information *Figure out what data is relevant to be displayed

			speed and accuracy of both ML algorithms, one of which being the neural network.		obtained		
Week 8	Integration of subcompo nents into main system	* Debugging of issues that arise from integration * Improve on power consumption	* Finetune the implementati on of both ML algorithms, making comparisons to determine which is better.	*Integrating sensor packet data received from the Beetles to the machine learning model *Improve speed and reliability of data transmission whenever possible	*Test communicatio n with actual data obtained from algorithms *Fix bugs and make modifications	*Work with FPGA team to amend ML algorithms for FPGA acceleration	*Makes to received feedback *Decide useful trends that display data
Week 9	Testing of Integrated prototype	* Gather data for machine learning models	* Train both ML models on newly gathered data from HW1	*Perform limit testing on the entire internal communicatio ns protocol in real time *Fix any bugs arising and ensure communicatio n is robust	*Make communicatio n faster and more reliable	*Assess efficiency of ML algorithm on the data collected from actual sensors *Collect data from users	*Work on the UI/UX portion to make more aesthetic *Fine tune data details in the graphs
Week 10	First evaluation test (3 dancer, 3 moves)	Test our product on the 3 moves					
Week 11	Fine-tunin g of integrated system	* Experiment and finetune the integrated system	* Fine-tune both ML models and finalize on the	*Fine-tuning the integrated system	*Experiment and tune	*Fine tune based on the results from the first	*Work on documentati on and fine tuning of

	(remember peer review)		best implementation for Hardware Acceleration.			evaluation test, collect more data if needed	code
Week 12	Mock evaluation test of integrated system (final test before final evaluation)	* Final test on full system prior to final evaluation	* Resolve issues that surfaces from the Mock Evaluation if any.	*Final testing on pre-evaluation integrated system		*Ensure ML algorithm is working reliably	*Heavy user testing of product
Week 13	Final evaluation (10 moves) + Final Design Report	Final evaluation					

References

[1] Arvind Sanjeev. *How to Interface Arduino and the MPU 6050 Sensor.* [online] Available at: https://maker.pro/arduino/tutorial/how-to-interface-arduino-and-the-mpu-6050-sensor

[2] Dfrobot. *Beetle BLE - The smallest Arduino bluetooth BLE (4.0).* [online] Available at: https://www.dfrobot.com/product-1259.html

[3] Theorycircuit. *Myoware Muscle Sensor Interfacing with Arduino.* [online] Available at: http://www.theorycircuit.com/myoware-muscle-sensor-interfacing-arduino/

[4] Maxim Krasnyansky. *hciattach(8) - Linux man page*. [online] Available at: https://linux.die.net/man/8/hciattach [Accessed 2 Feb. 2020]

[5] Jim Blom. *Serial Communication*. [online] Available at: https://learn.sparkfun.com/tutorials/serial-communication/all [Accessed 2 Feb. 2020]

[6] threading - *Thread-based parallelism*. [online] Available at: https://docs.python.org/3/library/threading.html [Accessed 3 Feb. 2020]

[7] Medium.com. (2018). *Using Multiprocessing to Make Python Code Faster*. [online] Available at: https://medium.com/@urban_institute/using-multiprocessing-to-make-python-code-faster-23ea5ef996ba [Accessed 6 Feb. 2020]

[8] sluiter, s. (2019). *How does AES encryption work? Advanced Encryption Standard*. [online] YouTube. Available at: https://www.youtube.com/watch?v=lnKPoWZnNNM [Accessed 2 Feb. 2020].

[9] Sridhar, J. (2020). *Using AES for Encryption and Decryption in Python Pycrypto | Novixys Software Dev Blog*. [online] Novixys Software Dev Blog. Available at: https://www.novixys.com/blog/using-aes-encryption-decryption-python-pycrypto/#3_Initialization_Vector [Accessed 2 Feb. 2020].

[10] En.wikipedia.org. (2020). *Network Time Protocol*. [online] Available at: https://en.wikipedia.org/wiki/Network_Time_Protocol [Accessed 2 Feb. 2020].

[11] AG, A. (2020). *Time Synchronization with NTP*. [online] Akadia.com. Available at: https://www.akadia.com/services/ntp_synchronize.html [Accessed 2 Feb. 2020].

[12] Ntp.org. (2020). *How does it work?*. [online] Available at: http://www.ntp.org/ntpfaq/NTP-s-algo.htm#Q-ALGO-POLL-BEST [Accessed 2 Feb. 2020].

[13] Docs.oracle.com. (2020). *What Is a Socket? (The Java™ Tutorials > Custom Networking > All About Sockets)*. [online] Available at: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html [Accessed 2 Feb. 2020].

[14] Quora.com. (2020). *Cryptography: What are the advantages and disadvantages of AES over Triple-DES? - Quora*. [online] Available at: https://www.quora.com/Cryptography-What-are-the-advantages-and-disadvantages-of-AES-over-Triple-DES [Accessed 3 Feb. 2020].

[15] Postgresql.org. (2020). *PostgreSQL: Documentation: 9.3: Connections and Authentication*. [online] Available at: https://www.postgresql.org/docs/9.3/runtime-config-connection.html [Accessed 3 Feb. 2020].

[16] Sahin, Suhap, et al. “Neural Network Implementation in Hardware Using FPGAs.” *Neural Information Processing Lecture Notes in Computer Science*, 2006, pp. 1105–1112., [Accessed 5 Feb. 2020]

[17] Shawahna, A., Sait, S. M., & El-Maleh, A. (2019). FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access*, *7*, 7823–7859. [Accessed 8 Feb. 2020]

[18] Afifi, S., Gholamhosseini, H., & Sinha, R. (2016). Hardware Acceleration of SVM-Based Classifier for Melanoma Images. *Image and Video Technology – PSIVT 2015 Workshops Lecture Notes in Computer Science*, 235–245., [Accessed 9 Feb. 2020]

[19] Christian, A. (2017, December). Analyzing User Emotions via Physiology Signals. Retrieved February 9, 2020, from https://www.researchgate.net/publication/323935725_Analyzing_User_Emotions_via_Physiology_Signals

[20] Ng, A. (n.d.). Non-linear SVM classification with kernels. Retrieved February 9, 2020, from http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html

[21] Gandhi, R. (2018, July 5). Support Vector Machine - Introduction to Machine Learning Algorithms. Retrieved from https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[22] Harrison, O. (2019, July 14). Machine Learning Basics with the K-Nearest Neighbors Algorithm. Retrieved from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[23] Khandelwal, R. (2019, January 25). K fold and other cross-validation techniques. Retrieved from https://medium.com/datadriveninvestor/k-fold-and-other-cross-validation-techniques-6c03a2563f1e

[24] Agarwal, R. (2019, November 26). The 5 Classification Evaluation metrics every Data Scientist must know. Retrieved from https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226

[25] Lima, W. S., Souto, E., El-Khatib, K., Jalali, R., & Gama, J. (2019). Human Activity Recognition Using Inertial Sensors in a Smartphone: An Overview. *Sensors*, *19*(14), 3213. doi: 10.3390/s19143213