CG4002 Embedded System Design Project

August 2020 semester

"Dance Dance"
Design Report

Group 5	Name	Student #	Sub-Team	Specific Contribution
Member #1	Tay Tze-wei, Caleb	A0182568R	HW	Hw Sensors
Member #2	Lim Zhi Hui Alden	A0172372H	HW	Hw FPGA
Member #3	Huang Xuan Kun	A0185827N	Comms	Comms Internal
Member #4	Wong Ren-Wei, Ryan	A0183478N	Comms	Comms External
Member #5	Alvarez Nicole Kevin Timbol	A0166667M	SW	Sw Machine Learning

Table of Content

1. System Functionalities

1.1 User Story (Kevin)

User story 1:
As a user who prefers simplicity, I want the wearables to be easy to wear so that I could dance immediately without the hassle of learning how to wear the device.

User story 2:
As a user who uses the device often, I want the wearables to be durable, lightweight and comfortable so that I could dance without strain, pain and the worry that I would break it.

User story 3:
As a user who will use the device for dancing, I want the wearables to last at least for more than the duration of the song.

1.1.1 Feature List (Kevin)

1. To identify dancer's dance moves
2. To identify dancer's relative locations
3. To display dance accuracy and evaluation

1.1.2 Wearable Wristband Feature List (Caleb)

These are some of the functionalities that the wearables will be supporting:

1. Lightweight and portable
2. Comfortable to be worn on the wrists
3. Easy to recharge

2. Overall System Architecture

2.1 High-Level System Architecture (Alden)

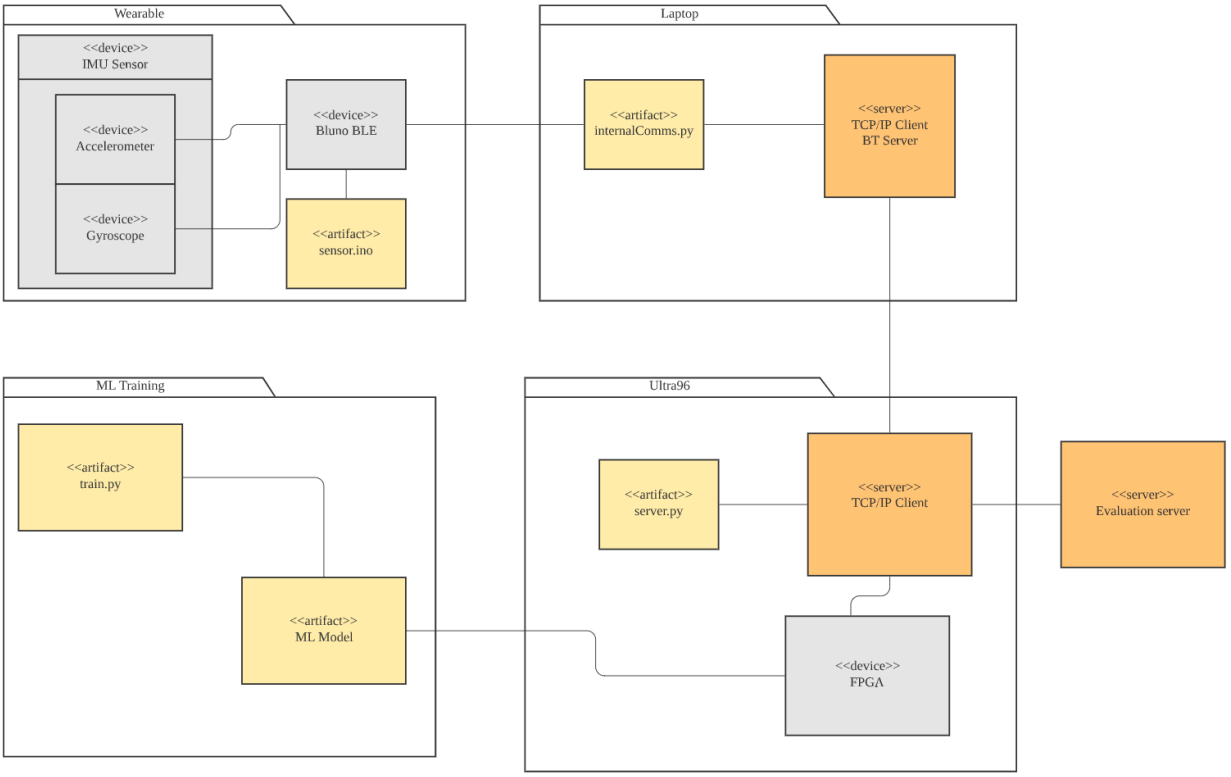


Figure 2.1.1 High Level System Architecture

2.2 High-Level Algorithm for Activity Detection (Alden)

1. Sensors detect movement from dancer
2. Data from sensors will be transmitted to the laptop through BLE.
3. Laptop transfers data to the Ultra96 using TCP/IP protocol.
4. Communication server on Ultra96 receives raw data and preprocesses it.
5. ML model as implemented on the FPGA takes the preprocessed data as an input and returns the corresponding activity as the output.
6. Output is transferred to evaluation server

2.3 Communications Implementation (Ryan & XK)

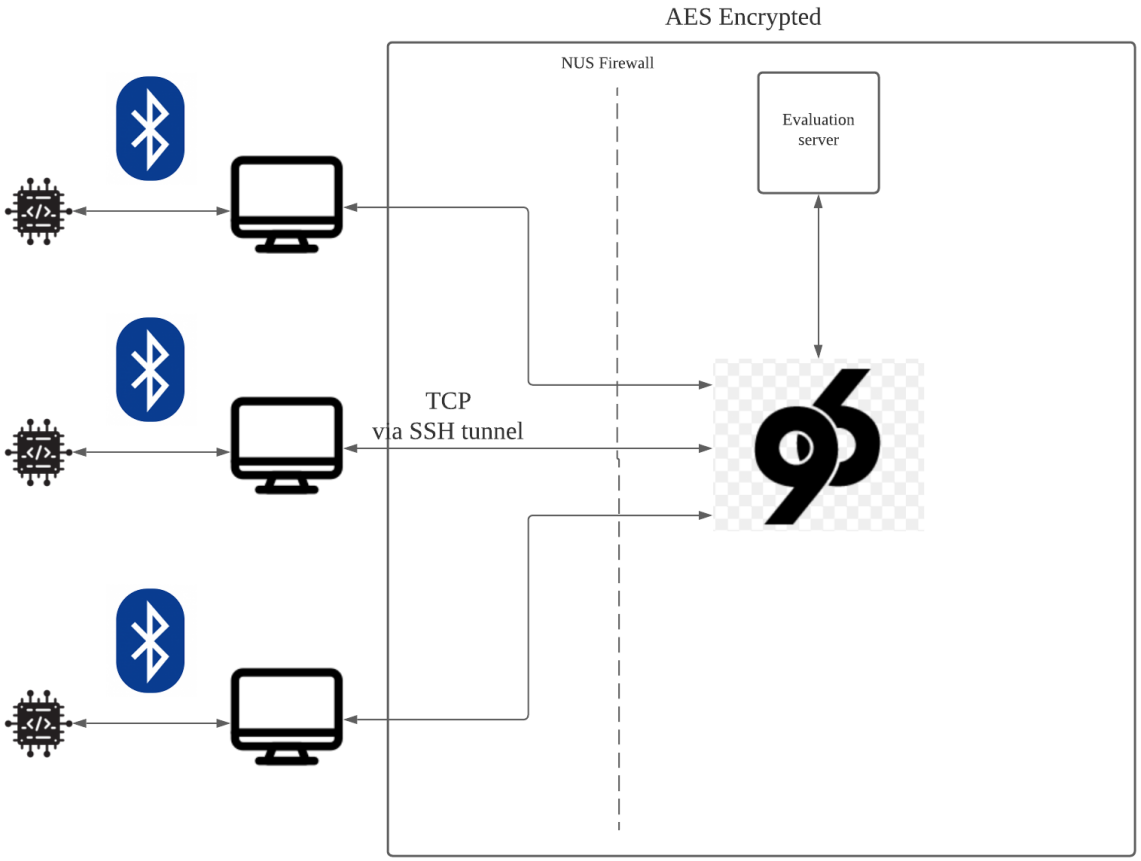


Figure 2.3.1 Communications Architecture

- Internal Communication:
  - Utilized Bluetooth Low Energy (BLE) for communication between Beetle and laptops. IMU data acquired from the accelerometer and gyroscope will be transmitted from Beetles to laptops at fixed frequency.
  - Serial programming is performed in Arduino Uno for implementation of network protocol on Arduino Bluno Beetle. The bluepy library is used for implementing BLE communication in Linux environments.
  - With data received from the beetle, the laptop will process the data with decoding and formatting, and transmit it to the Ultra96 board.

- External Communication:
  - The ultra96 board will host a server that handles communications with the 3 laptops (dancers) and the evaluation server.
  - The Ultra96 board will be accessed from the dancers' laptops through the NUS firewall via SSH tunnel and TCP communication
  - All communication to and from the Ultra96 board will be AES encrypted with a pre-shared key as seen in the diagram above

2.4 Machine Learning Implementation (Kevin)

The objective for the machine learning section is to classify and identify 9 different dance moves (including the resting position) as well as the relative positions of the dancers. We will be using the Multilayer Perceptron (MLPs) for the machine learning model and the Convolutional Neural Network (CNN) for the neural network model. Once the sensors from the wearable have captured the data and sent to Ultra96, the machine learning section will take over to process the data.

Firstly, the domain of the features to be extracted may need to be converted from the time domain to the frequency domain if needed. Preprocessing of data will also be needed if the data has too much noise. This can be implemented either through software or hardware implementations of a low pass filter. The data will then be segmented using an overlapping sliding/rolling window approach. (Before the actual evaluation, the data will be trained and evaluated after the data has been segmented using the K-fold cross validation method.) For any computational heavy processes, it will be performed on the FPGA and the final output will be obtained from the Ultra 96 CPU. It will then be returned back to the external communications server and then to the evaluation server.

2.5 Wearable Wristband Design (Caleb)

For the design of the device, we considered different positions the devices could be placed on and weighed the pros and cons of each possible implementation. Since the purpose of the device is to detect dance moves as well as detect muscle fatigue, the goal would be to choose positions where the device would get the most accurate readings as well as be the least obstructive when the user is dancing.

Position	Pros	Cons
Wrist	Gets the most movement detected	Forearms not used as much by selected dance moves, difficult to get a good reading of muscle fatigue
Forearm	Gets a good amount of movement detected	Not very comfortable to wear
Biceps	Muscle fatigue of biceps can be detected	Does not detect if arm is bent, cannot detect movement of forearm and hands
Shoulder	Can be used to detect muscle fatigue of the shoulders Comfortable to wear	Does not detect arm movement
Ankles	Detects leg movement	Does not detect arm movement

After considering different positions the user could wear the movement & muscle fatigue devices, we decided on implementing the movement detection device on the wrist in order to get the best reading of the user's movement and the muscle fatigue device on the biceps in order to get a good reading of muscle fatigue as well as not obstruct the user's movement.

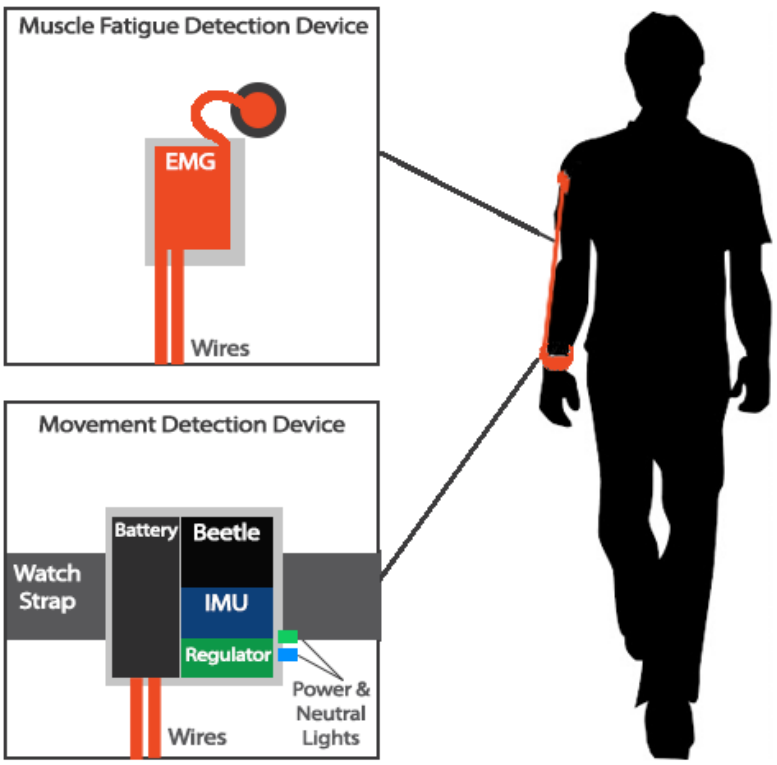
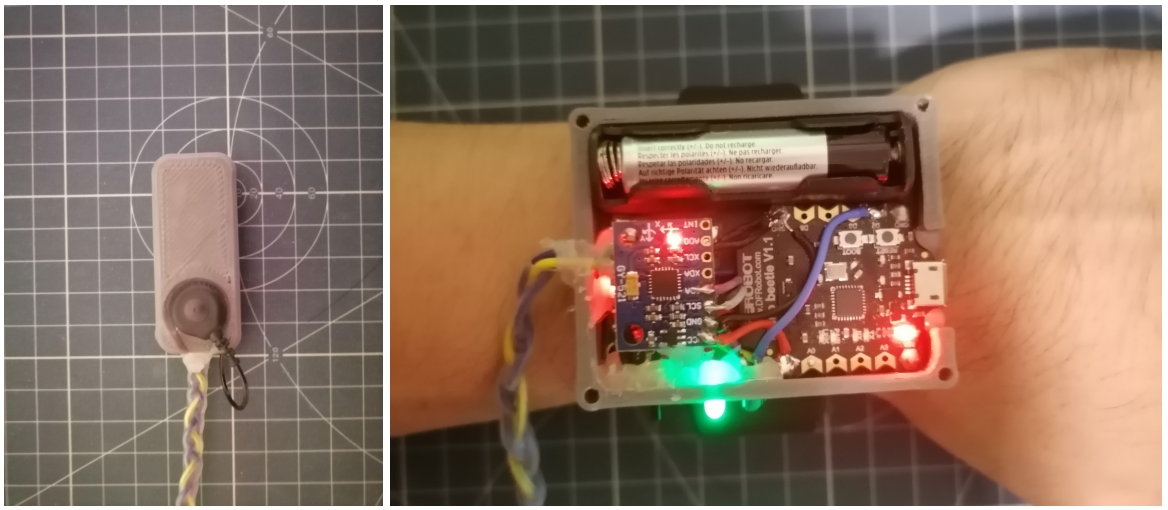


Figure 2.5.1 Intended Form of Wearable



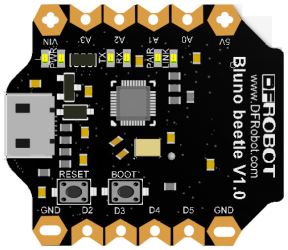
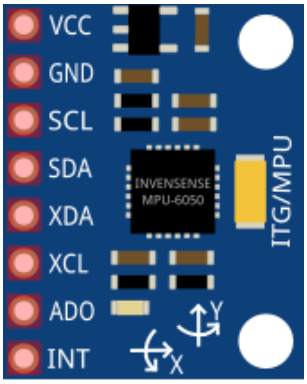
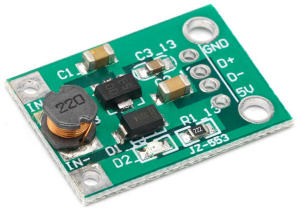
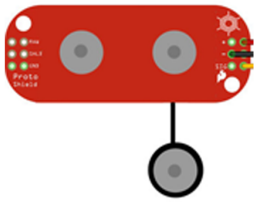
Figures 2.5.2 & 2.5.3 Final Form of Wearable

The devices both consist of a smart watch styled layout in order to make the device simpler for users to get used to. The movement detection device is intended to use a smart watch strap and the muscle fatigue unit is intended to carry heart rate sensors for runners. This is to ensure that users find the device comfortable and do not feel obstructed when they are dancing. We intend to use a single AAA battery to power the device as it is easy to change and has less risk of fire as compared to Lithium Polymer batteries. The muscle fatigue detection device will have a separate housing from the other components and be connected via wires. The emg sensor will stick out of the muscle fatigue detection device and will be stuck to the user's shoulder via the EMG pads. Lastly, a 3D printed case will be used in order to protect the electronics from shock and moisture.

3. Hardware Details (Caleb)

3.1. Hardware sensors

3.1.1 Hardware Components

	Device	Requirements	Datasheet
	Bluno beetle BLE, DFR0339	- Provided	Blueno Beetle Datasheet
	IMU board, GY-521	- Provided	MPU Datasheet
	AAA battery 1.5v 200mAh		Energizer AAA Datasheet
	JZ-553 DC-DC 1V-5V TO 5V 500mA Boost Converter Step Up Power Module		JZ-553 Specs
	EMG sensor Myowear Muscle Sensor	- Provided	Myowear Datasheet

	Versa-Trode Solid Gel Tape	- Fits EMG sensor - Able to stick electrode onto shoulder	
	Apple watch strap	- Comfortable	

3.1.2 Pin Table

Connection Type	Wired from	Wired to
Beetle's Power	Beetle VIN Pin	JZ-553 Output +ve Pin
Beetle's Ground	Beetle GND Pin	JZ-553 Output -ve Pin
JZ-553 Power Input	JZ-553 Input +ve Pin	AAA Battery +ve Pin
JZ-553 Ground Input	JZ-553 Input -ve Pin	AAA Battery -ve Pin
MPU Power	MPU VCC Pin	Beetle 5V
MPU Ground	MPU GND Pin	Beetle Gnd
MPU SDA	MPU SDA Pin	Beetle SDA Pin
MPU SCL	MPU SCL Pin	Beetle SCL Pin
EMG Power	EMG Vin Pin	Beetle 5V Pin
EMG Ground	EMG Gnd Pin	Beetle Gnd Pin
EMG Signal	EMG Sig Pin	Beetle A0 Pin

3.1.3 Hardware Schematics

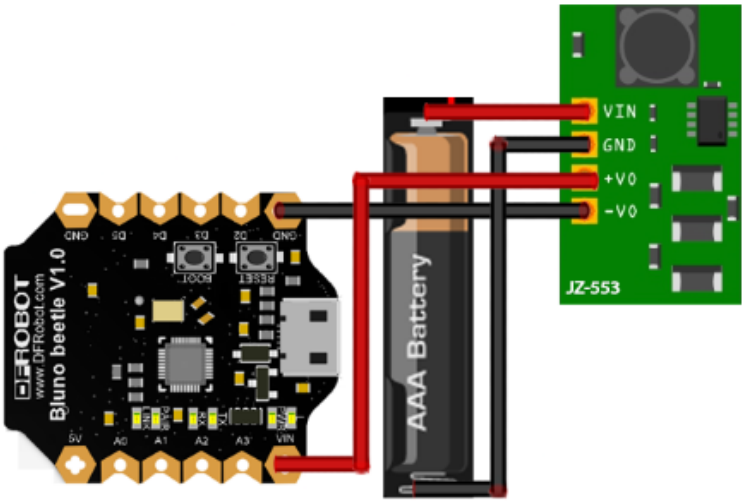


Figure 3.1.1 Schematics for Connection of battery system.

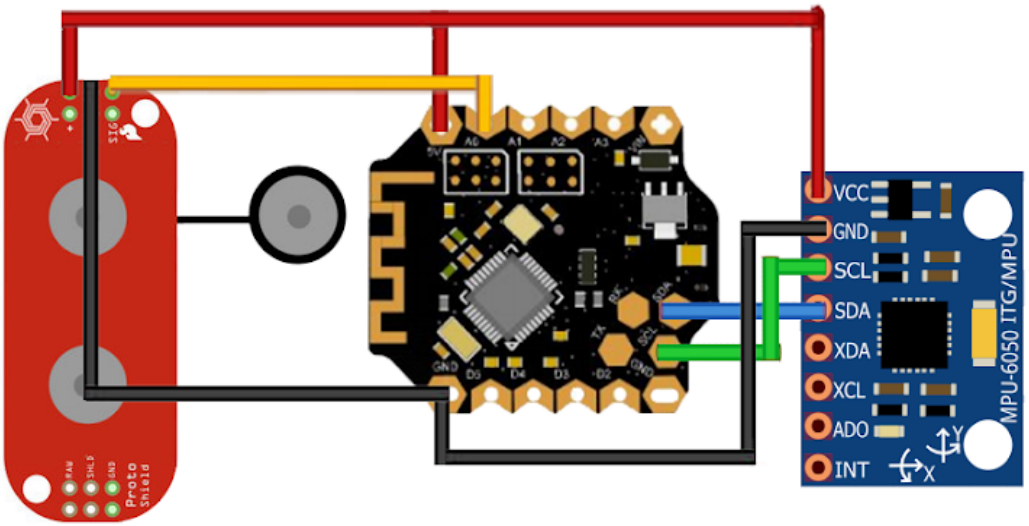


Fig. 3.1.2 Schematics for Connection of IMU & EMG sensors.

3.1.4 Power Considerations

Power system, logic level considerations

Component Power Specifications

Device	Operating Conditions	Current (mA)	Voltage (V)	Reference
Beetle	Normal	10 - 35	5	Datasheet Battery Consumption
IMU Sensor (GY-521)	Gyroscope + Accelerometer (DMP disabled)	operating: 3.6 - 10mah. Standby 5µA.	0 - 3.3	Datasheet
Myowear Muscle Sensor		9 - 14	5	Datasheet
JZ-553 Step-up regulator		NIL	5	JZ-553 Specs
AAA Battery	850mAh	NIL	1.5	Energizer AAA Datasheet

Estimated System operating Current:

- Beetle: 10 - 35mAh
- IMU: 3.6 - 10mAh
- EMG: 9 -14mAh
- Total System(No EMG): 35mAh + 10mAh = 45mAh
- Total System(EMG): 35mAh + 10mAh + 14mAh = 59mAh

Power System Calculations:

Since only one user has to utilise the EMG to detect muscle fatigue, both scenarios were considered during our calculations.

Expected Operation Time of device (No EMG):
850mAh / 45mAh ≈ 18.9h

Expected Operation Time of device (With EMG):
850mah / 59mAh ≈ 14.4h

Our Intended Operating Range	5V
Expected Operation Time	14.4 - 18.9h

3.1.5 Algorithms

Raw Data from IMU Sensor

To obtain the raw values from the IMU sensor, the following references and libraries were used.

Libraries:

1. Jrowberg i2cdevlib

The variables to read from sensors:

1. Gyroscope readings:
    a. eulerX: Euler rotation with respect to X axis
    b. eulerY: Euler rotation with respect to Y axis
    c. eulerZ: Euler rotation with respect to Z axis
2. Accelerometer readings:
    a. accX: Acceleration with respect to the X axis
    b. accY: Acceleration with respect to the Y axis
    c. accZ: Acceleration with respect to the Z axis
3. EMG muscle detection signal readings:
    a. EMG: Analog reading of muscle activation

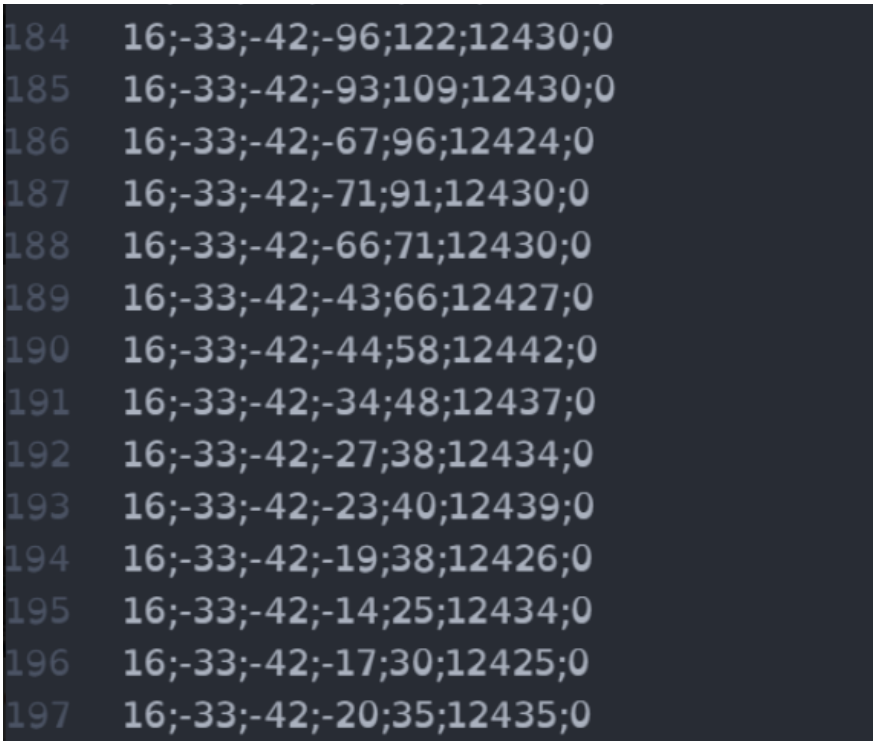


Figure 3.1.3 Raw Data Sample

Data from Fig 3.1.3 is represented in the form EulerX,EulerY,EulerZ,accX,accY,accZ,EMG

Adding MPU offsets for devices

When we began testing the raw data output from multiple devices, we noticed significant differences in the euler and acceleration data between each device despite the devices being arranged in similar rotations and left untouched. After some research, we discovered that all MPU devices require a different offset in order to properly zero the data. Using a calibration script made by Luis Ródenas, we were able to obtain the offsets for each of our MPU devices and set them when our devices' MPUs were initialised.

```
753  void calibrateDevice(int device_num)
754  {
755    switch (device_num)
756    {
757    case 1:
758      mpu.setXGyroOffset(210);
759      mpu.setYGyroOffset(-11);
760      mpu.setZGyroOffset(35);
761      mpu.setXAccelOffset(451);
762      mpu.setYAccelOffset(-1927);
763      mpu.setZAccelOffset(80);
764      break;
765    case 2:
766      mpu.setXGyroOffset(158);
767      mpu.setYGyroOffset(47);
768      mpu.setZGyroOffset(-17);
769      mpu.setXAccelOffset(-207);
770      mpu.setYAccelOffset(-2384);
771      mpu.setZAccelOffset(671);
772      break;
```

Figure 3.1.4 calibrateDevice function

The calibrateDevice function takes in the device index which is defined at the beginning of the script and uses a switch case to set the correct offset for the device.

Detecting the beginning & end of a dance move

In order to detect the start and end of a dance move, we decided to create a state machine with two states, move and neutral to define when the user is dancing. The internal comms script would detect the first packet with the move state as the start of the dance and the first packet with the neutral state as the end of the move.

To determine what state the device was in, we implemented functions that used thresholds to check if the device was at the side of the user and was not in motion.

```
bool inNeutralPos()
{
  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.dmpGetEuler(euler, &q);

  float eulerX = euler[0] * 180 / M_PI;
  float eulerY = euler[1] * 180 / M_PI;
  float eulerZ = euler[2] * 180 / M_PI;

  if (eulerX < -50 && eulerX > -100 && eulerZ > -120 && eulerZ < -60)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

Figure 3.1.5 inNeutralPos function

In the inNeutralPos function, we check that the euler angles of the device are within the defined threshold and return true if so.

```
bool notMoving()
{
  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.dmpGetAccel(&aa, fifoBuffer);
  mpu.dmpGetGravity(&gravity, &q);
  mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
  mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);

  float accX = aaWorld.x;
  float accY = aaWorld.y;
  float accZ = aaWorld.z;

  if (abs(accX) < 2500 && abs(accY) < 2500 && abs(accZ) < 2500)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

Figure 3.1.6 notMoving function

In the notMoving function, we check if the absolute acceleration in each direction is less than the threshold value of 2500 units and return true if so.

3.2 Hardware FPGA [Lim Zhi Hui Alden]

3.2.1 Synthesis and Simulation Setup.

We tested 2 methods of utilizing the FPGA hardware. The first was to use the HLS libraries in C++ and code out each individual layer of the convolutional neural network (CNN) using the model given to us by the software teammate. Each layer is coded out with the correct weights in C++ with the passing of data between each layer being handled by the HLS library code. The validity of the coded network can be tested through a C++ test code which takes a series of inputs and expected outputs generated by the network run on python and compares them. This ensures that the conversion of the network into C++ HLS compatible code did not change it.

Once the C++ code has been tested, Vivado HLS is then used to synthesize the code into an Intellectual Property (IP), that can be used in a Vivado block design to be properly used by FPGA hardware. This block design can then be put through a simulation on Vivado by writing a verilog file to ensure that it is functioning as intended. This block design is then used to generate a bitstream and hardware description file for an Ultra96.

Finally a driver file is written for deployment on the ultra96. This is a python file that reads the bitstream and hardware description file and utilizes the hardware to run the neural network inference in accordance to the block design on Vivado. The functionality of this is tested in a full integration test with the hardware and communications with the rest of the team.

The second method that we used was by first taking the trained neural network provided by the software teammate and then binarizing the weights and activations through the use of the FINN framework. The BNN-PYNQ framework also provides C++ libraries for the usage of synthesizing Neural Networks. Each layer of the neural network already exists in the library and does not need to be manually coded, only put together. Combining these libraries with the binarized weights and networks creates a functional neural network in C++. From this point, the same steps as previously detailed are followed

3.2.2 Neural Network Design

We will be testing classification using a Support Vector Machine (SVM) and a Convolutional Neural Network (CNN). The specifics of these designs will be discussed in section 5.1. As a SVM is not considered a neural network, the focus of this section will be the reason behind our choice of using a CNN.

CNNs have reached remarkable levels of accuracy in the field of classification, most well known for its usage in image classification and High Level Synthesis Tools have demonstrated considerable progress and reliability in generating FPGA-based hardware designs from high level abstractions. We believe that a CNN is an appropriate choice in the context of deployment on an FPGA because there are various research papers on the effectiveness of CNN acceleration techniques on FPGAs (Kala, S., & Nalesh, S. , 2020).

We plan on training a Quantized Neural Network(QNN) based on the CNN that we generate through the Machine Learning section of our project. QNNs are neural networks with extremely low precision weights and activations. This reduction in precision allows for a

reduction in hardware utilization which should lead to a decrease in power consumption (Qiao, Y., et al, 2016).

3.2.3 Mapping onto FPGA

As previously mentioned there were 2 methods being used to map the neural network onto the FPGA. The first was to manually code out each layer of the neural network in C++ with the help of HLS libraries. A small example of what that looks like is depicted below

```
void conv_layer1(hls::stream<float24_t> &out, hls::stream<float24_t> &in,
        float24_t weight[CONV1_KERNEL_SIZE][CONV1_KERNEL_SIZE][CONV1_CHANNELS][CONV1_FILTERS],
        float24_t bias[CONV1_BIAS_SIZE]) {
    int i, j, k, filter;
    float24_t sum, placeholder;
    int row_offset, col_offset, channel_offset;
    xf::cv::LineBuffer<CONV1_BUFFER_SIZE, 1, float24_t> conv_buff;

    for (i = 0; i < CONV1_BUFFER_SIZE; i++) {
        if (in.empty() == 0) {
            in >> placeholder;
            conv_buff.shift_up(0);
            conv_buff.insert_top(placeholder, 0);
        }
    }
    for (i = 0; i < (IMAGE_SIZE - CONV1_KERNEL_SIZE + 1); i += CONV1_STRIDE)
        conv_layer1_label9: for (j = 0;
                j < (IMAGE_SIZE - CONV1_KERNEL_SIZE + 1); j += CONV1_STRIDE)
                {
            conv_layer1_label2: for (filter = 0; filter < CONV1_FILTERS;
                    filter++) {
                sum = 0;
                conv_layer1_label6: for (row_offset = 0;
                        row_offset < CONV1_KERNEL_SIZE; row_offset++)
                    conv_layer1_label7: for (col_offset = 0;
                            col_offset < CONV1_KERNEL_SIZE; col_offset++)
                        conv_layer1_label8: for (channel_offset = 0;
                                channel_offset < CONV1_CHANNELS;
                                channel_offset++) {
                            int t1, t2;
                            static float24_t val1, val2;
                            t1 = row_offset * IMAGE_SIZE * IMAGE_CHANNELS;
                            t2 = col_offset * IMAGE_CHANNELS;
                            val1 = conv_buff.getval(t1 + t2 + channel_offset,
                                    0);
                            val2 =
                                    weight[row_offset][col_offset][channel_offset][filter];
                            sum += val1 * val2;
                        }

                out << relu(sum + bias[filter]);
            }
```

Fig 3.2.1 Manually coded convolutional layer

A second method of mapping to the FPGA involves the BNN-PYNQ framework. This involved first binarizing the neural network then coding out the neural network in C++ to be synthesized into an IP

This IP is then used as part of a block design that allows proper usage of the FPGA hardware in deploying the model

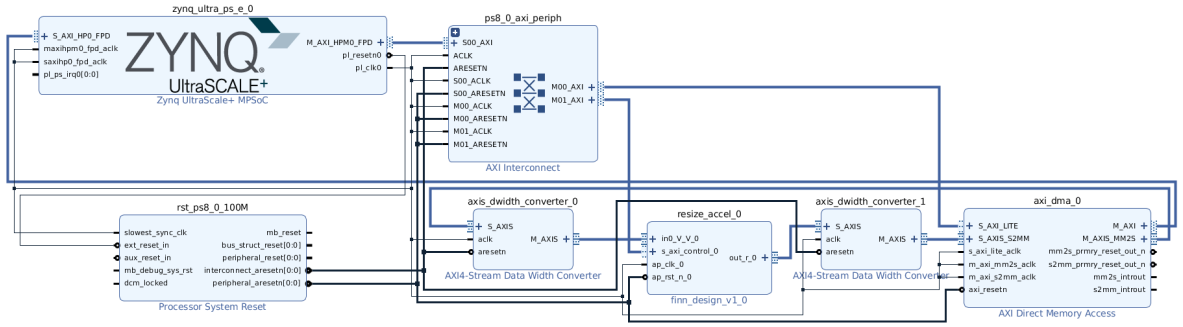


Fig 3.2.2 Vivado Block Design

Finally a bitstream and hardware description file is generated, which is then used by a driver script written in python, which utilizes the PYNQ libraries, to do the inferencing during usage

3.2.3.1 Functional Evaluation

We are able to use simulations to help us to evaluate the functionality of our design. The main functionality testing happens at 3 stages. First is after the neural network has been written in C++ code, prior to synthesis. A check is done here to ensure all inputs into the C++ code generates the same outputs as the same inputs being given to the Neural Network being run on python

The second stage in which simulations are carried out is post IP generation. This allows us to ensure that the conversion process from C++ code into and IP does not change the functionality.

Finally, simulations are done after the block design is generated. This allows us to ensure that the full integration of the IP for inferencing works.

We then proceed with doing a full integration test with the rest of the systems.

3.2.4 Optimizations

We will be using 3 common metrics in order to evaluate our performance and we will be optimizing for these 3 metrics

Resource Utilization: We could want to minimize the amount of resources being utilized as much as possible. Resource Utilization can be measured through things like the number of channels used and the number of inputs which can be seen through the Report Cell Usage function. It can also be measured through Look Up Table Utilization % through the Report Utilization function and the CPU Usage % through PowerTOP

Latency and Throughput: We would want to ensure that we are able to provide outputs at a reasonable speed with relation to our use cases. More importantly, we would want to ensure that the hardware acceleration can at least help to outperform running the model on our laptops. Latency and throughput can be measured by timing the speed at which the quantized model is able to provide outputs.

Accuracy: We would want to ensure that the accuracy of our outputs are maintained to a reasonable degree, where they are still sufficiently accurate for our use case.

With these 3 metrics in mind, there are a few potential optimizations possible with tradeoffs between all 3 metrics being made for each one.

Quantization of floating point values: Reducing the bit widths of our floating point values will allow us to reduce the resource utilization of our model. However, if we quantize it to a bit width that is too small, we may be making a severe trade off in accuracy.

Level of Folding: Reducing the level of folding allows more data parallelism, which allows for throughput to increase. However, increasing parallelism also increases resource utilization.

Model reduction: We can run pruning algorithms to remove weights and layers in the model so that we can reduce resource utilization and increase throughput. Doing this too aggressively may result in a huge drop in accuracy

3.2.4.1 Manual Coded Network

In the manual coding of the neural network. We attempted to reduce the data width to a 8 bit floating point value with 4 bits reserved for integer values and 4 bits for decimal places. We also manually incorporated folding and unrolling of loops through directives. Depicted below is an example of that

```
set_directive_dataflow -disable_start_propagation "nnet"
set_directive_unroll "conv_layer1/conv_layer1_label6"
set_directive_unroll "conv_layer1/conv_layer1_label7"
set_directive_unroll "conv_layer1/conv_layer1_label8"
set_directive_unroll "conv_layer1/conv_layer1_label2"
set_directive_pipeline -II 8 "conv_layer1/conv_layer1_label9"
set_directive_unroll "conv_layer2/conv_layer2_label10"
set_directive_unroll "conv_layer2/conv_layer2_label11"
set_directive_pipeline "conv_layer2/conv_layer2_label12"
set_directive_unroll "conv_layer2/conv_layer2_label13"
set_directive_unroll "conv_layer2/conv_layer2_label3"
set_directive_unroll "conv_layer1/conv_layer1_label0"
set_directive_unroll "conv_layer1/conv_layer1_label1"
```

Fig 3.2.2 Loop unrolling and data pipelining

The effect it had on the hardware utilization and accuracy were as follows:

Accuracy: Maintains 100% of the same outputs as the model when run on python. Any further reduction of the data width causes a drop in accuracy which was deemed to not be a worthwhile trade off

Hardware utilization:

1. CLB Logic

Site Type	Used	Fixed	Available	Util%
CLB LUTs	16945	0	70560	24.02
LUT as Logic	16417	0	70560	23.27
LUT as Memory	528	0	28800	1.83
LUT as Distributed RAM	304	0		
LUT as Shift Register	224	0		
CLB Registers	18559	0	141120	13.15
Register as Flip Flop	18559	0	141120	13.15
Register as Latch	0	0	141120	0.00
CARRY8	711	0	8820	8.06
F7 Muxes	10	0	35280	0.03
F8 Muxes	0	0	17640	0.00
F9 Muxes	0	0	8820	0.00

Fig 3.2.3 Hardware Utilization Table (Manual Coded Network)

3. BLOCKRAM

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	190	0	216	87.96
RAMB36/FIFO*	9	0	216	4.17
RAMB36E2 only	9			
RAMB18	362	0	432	83.80
RAMB18E2 only	362			

Fig 3.2.4 Block RAM Utilization Table (Manual Coded Network)

Unfortunately, the Block RAM usage seems to be bottlenecking the utilization of the hardware. Block RAM was 87.96% utilized while the rest of the hardware maxed out at around 24% utilization. The large data widths that needed to be maintained might be causing this, which in turns limits the amount of parallelization being done.

Latency: The hardware acceleration increased the inferencing speed from 26 times per second to 8000 times per second.

3.2.4.1 FINN/BNN-PYNQ assisted Network

With the help of the FINN and BNN-PYNQ frameworks, a binarized network was retrained and these new weights and activations were put together in C++ to synthesize an IP.

Accuracy: Maintains 100% of the same outputs as the model when run on python up to a bit width of 2 for the weights and 1 for the activations. The reduction can be more aggressive while maintaining accuracy compared to naively reducing bit widths because the network is specifically retrained to handle smaller bit widths. The effect of this more aggressive reduction can be seen in the hardware utilization and latency

Hardware Utilization:

1. CLB Logic

Site Type	Used	Fixed	Available	Util%
CLB LUTs	35663	0	70560	50.54
LUT as Logic	33328	0	70560	47.23
LUT as Memory	2335	0	28800	8.11
LUT as Distributed RAM	2160	0		
LUT as Shift Register	175	0		
CLB Registers	39541	0	141120	28.02
Register as Flip Flop	39541	0	141120	28.02
Register as Latch	0	0	141120	0.00
CARRY8	1945	0	8820	22.05
F7 Muxes	146	0	35280	0.41
F8 Muxes	0	0	17640	0.00
F9 Muxes	0	0	8820	0.00

Fig 3.2.4 Hardware Utilization Table (FINN BNN Network)

3. BLOCKRAM

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	113.5	0	216	52.55
RAMB36/FIFO*	55	0	216	25.46
RAMB36E2 only	55			
RAMB18	117	0	432	27.08
RAMB18E2 only	117			

Fig 3.2.4 Block RAM Utilization Table (FINN BNN Network)

As seen from the report, there is a much more balanced hardware utilization with Block RAM and LUTs being utilized at a 50% rate.

Latency: The hardware acceleration improved the inferencing from 26 times per second to 40000 times per second

3.2.5 Power Management

The Ultra96 will be handling all computations on both its CPU and FPGA while also handling communications both from our laptops and to the evaluation server. As such, we would want to ensure that we optimize our power usage as much as possible. Part of these optimizations come from the reductions to the resource utilization from our neural network design optimizations.

We will want to keep track of our power usage using Vivado Power Utilization features, which should tell us the Total On-Chip Power and the breakdown of how this power usage is being split among Clocks, Signals, Logic, Bram and PS. We would want to use this as part of our considerations for our neural network design optimizations while also exploring whether there are other ways we can reduce overall power consumption.

The Zynq UltraScale+ MPSoC comprises of four major power domains:

1. Full Power Domain (FPD)
2. Low Power Domain (LPD)
3. Programmable Logic (PL) Power Domain
4. Battery-power Domain

The Xilinx power management framework was used to help reduce power usage, via functions such as XPM_SelfSuspend and XPM_RequestWakeup which suspends CPU usage when it is not needed. The final consumptions were as such

1. Summary

Total On-Chip Power (W)	2.947
Design Power Budget (W)	Unspecified*
Power Budget Margin (W)	NA
Dynamic (W)	2.624
Device Static (W)	0.323
Effective TJA (C/W)	2.7
Max Ambient (C)	91.9
Junction Temperature (C)	33.1
Confidence Level	Medium
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Fig 3.2.4 Power Utilization Table (Manual Coded Network)

1. Summary

Total On-Chip Power (W)	3.204
Design Power Budget (W)	Unspecified*
Power Budget Margin (W)	NA
Dynamic (W)	2.879
Device Static (W)	0.326
Effective TJA (C/W)	2.7
Max Ambient (C)	91.2
Junction Temperature (C)	33.8
Confidence Level	Medium
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Fig 3.2.4 Power Utilization Table (FINN/BNN Coded Network)

As expected, the manually coded network which had lower utilization of the hardware due to BlockRAM bottlenecking consumed less power

4. Firmware & Communications Details

4.1 Internal Communications [Xuan Kun]

4.1.1 Beetle Tasks Management

No.	Tasks	Description
1	Initialization & Calibration	Initialize Arduino program on Beetle and perform calibration.
2	Reading IMU data	Read IMU data with interrupt every loop.
3	Processing IMU data	Process the obtained IMU data with transformation and scaling.
4	Reading BLE Packet from laptops	Read any incoming BLE packet every 50ms
5	Transmitting BLE packet to laptops	Transmits the IMU data and flags in predefined packet format to the laptop via BLE.
6	Handshake	Handshake is triggered at the initialization of the program, and after the receiving reconnection flag from the laptop.

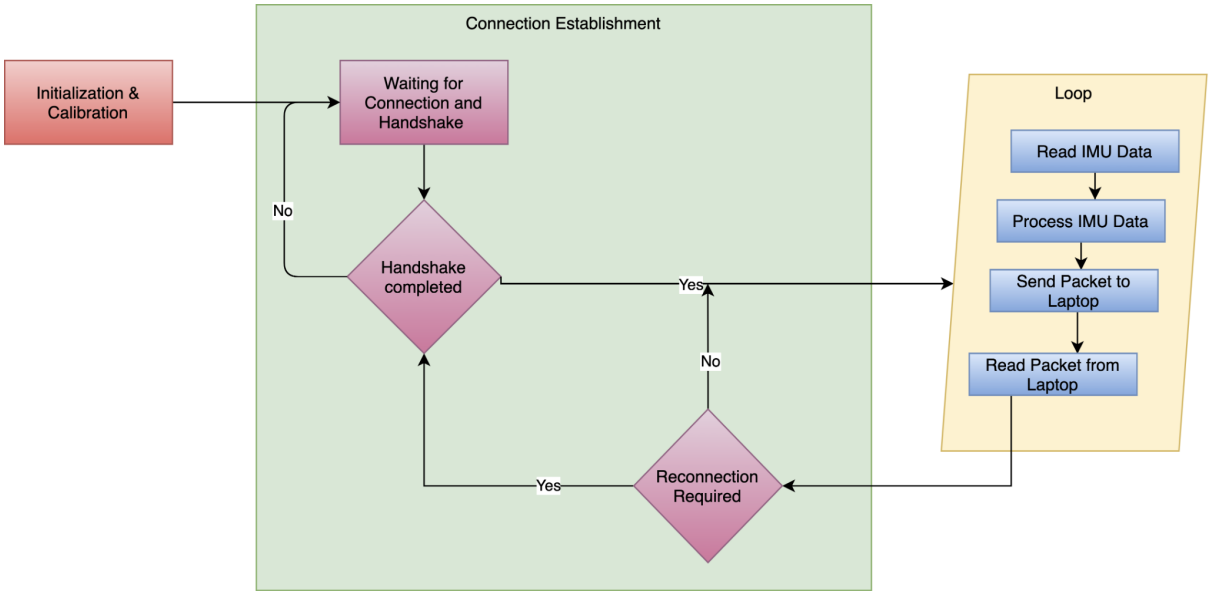


Fig. 4.1.1.1 BLE Task Overview

Design Considerations on BLE Tasks

Interrupt vs RTOS

Considerations were made to implement either Interrupt or RTOS functionality. The FreeRTOS library is consuming large memory space, and the beetle provides only 32KB of Flash memory and 2.5KB of SRAM. At the same time, our program only requires IMU data at a fixed frequency which RTOS is not actually necessary as there is no need for real-time task scheduling. With the above considerations, we decided to implement the interrupt between each loop with fixed frequency to obtain the IMU data, and to read and send packets within the loop.

Handshake frequency

The handshake task takes place at the initialization of the program, and after each reconnection. Considerations were made on whether to perform handshake at a fixed frequency, or only after each establishment of connection. After an experiment on the stability of BLE connection, we discovered that there is no need to perform handshake other than the initialization of connection and reconnection as:

1. There is no transmitting of timestamp between laptop and beetle
2. Pairwise BLE connection is extremely stable with low packet corruption

Loop Delay Setup

Before Week 7

The loop delay was set to **30 ms** before integration. It works well on Raspberry Pi 3 B+ model with dummy hardware and external communications. However, with the integration of other components, and using a AAA battery power supply, 30 ms was resulting in a significantly high packet corruption rate.

Final Implementation after week 7

We have found that battery level and distance from laptop are the two major factors causing the increases in data corruption. Thus, experiment is performed to find the optimal delay for each loop in the Ardino program.

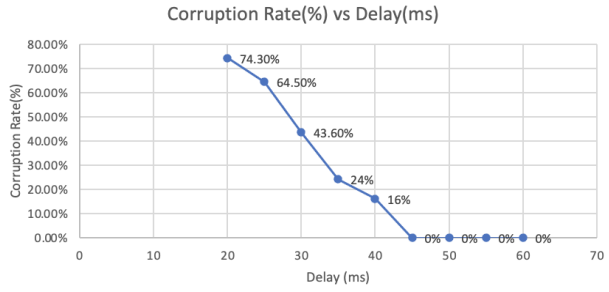


Figure 4.1.1.2 Battery used for 20 mins

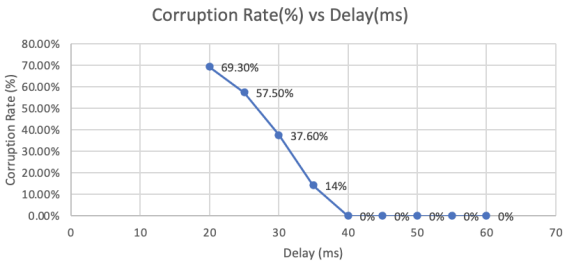


Figure 4.1.1.3 Full Battery

The above experiments are conducted with full battery and used battery for 20 mins with a Razer laptop. The beetle is 4 meters away from the laptop, and the experiment lasts for 5 minutes.

With the experiment results, we decided to use a delay of 50 ms in our final evaluations since it provides a consistent performance even with lower battery level.

4.1.2 BLE Setup and Configuration

Beetle Configuration Commands with Arduino IDE

Command	AT Command
Enter AT Mode	+++
Exit AT Mode	AT+EXIT
Find Mac Address	AT+MAC=?
Set Binding Mac Address	AT+BIND=
Set Minimum Connection Interval	AT+MIN_INTERVAL=
Set Maximum Connection Interval	AT+MAX_INTERVAL=

Laptop CLI Configuration Commands

Description	Command
Configure Bluetooth device	hciconfig
Activate bluetooth interface	sudo hciconfig hcix on
Access hcix interface configuration directory	cd /sys/debug/bluetooth/hcix
Set max interval to xx	echo xx > conn_min_interval
Set max interval to xx	echo xx > conn_max_interval”

Design Consideration on BLE connection interval

The connection intervals of a BLE device is inversely proportional to the power consumption and the data throughput. At the same time, if the interval is set beyond the BLE bandwidth of our laptop, there will be an increase in packet corruption rate.

Thus, it is important to find the balance between data throughput and power consumption. Even though setting the minimum & maximum interval to 10ms will result in a higher throughput and obtaining the ML result with lower delay, the interval settings on Beetle is set back to default of [20ms, 40ms] after each reboot. This will lead to higher packet corruption rate if either the beetle or laptop is set with a higher connection interval.

Thus, compromisation is done by keeping the connection interval at default value, and to modify our ML model such that it requires a smaller packet frame to provide an accurate output.

4.1.3 Communication Protocol

Pre-handshake Processes

Beetles act as the peripherals and laptop acts as central devices. Before establish connection between The connection processes between beetles and laptops are listed as follows:

1. Obtain *MAC address* of each Beetle using *Arduino Serial Monitor*
2. Laptops scan for Beetles based on MAC address. Each detected Beetle acts as a *peripheral*, and will be assigned with a UUID.
3. A *delegate* object is set for each connected *peripheral* to handle notification or indication is received.
4. The handshake process is then carried out for each connected beetle.

Three-way Handshake

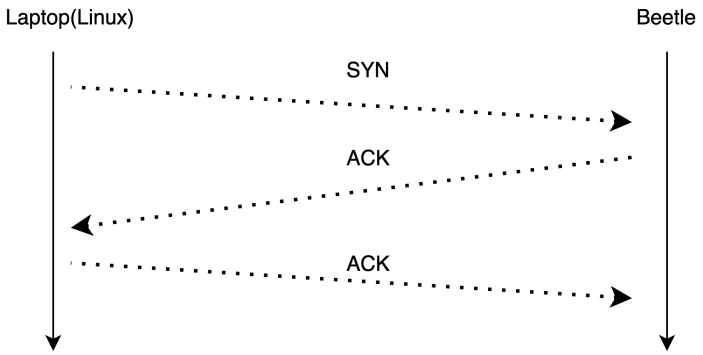


Figure 4.1.3.1 Three-way handshake

Similar to classic TCP-IP 3-way handshake, the laptop will first initiate the handshaking process by sending a 'SYN' packet to each detected Beetle. The laptop will then wait for the Beetle to respond with an "ACK" packet as an indication of acknowledgement. Finally, the laptop will acknowledge the Beetle response by sending back an "ACK" packet and communication will be established between them. In case of missing ACK response from either laptop or beetle after pre-defined timeout, the handshaking process will be repeated.

Packet Format Overview

There are two types of packets

- Response packet: packet used for handshake and indicating the connection
- data packet: packet used for transmitting IMU data and flags

Differentiation between response packet and data packet

Beetle: Beetle will only receive response packets from the laptop, which consists of a one byte character. Thus there is no need for differentiating packet type.

Laptop: The differentiation between two types of packets on a laptop is straightforward, since responses packets contain only 1 byte character, after splitting packet with packet delimiter, we compare the decoded value directly.

Handshake Packet

Packet Type	Code	Description
SYN from Laptop	'H'	Initializing handshaking
ACK from Beetle	'A'	Indicating the acknowledgment from beetle
ACK from Laptop	'R'	Indicating the acknowledgment from laptop

Figure 4.1.3.1 Handshake packet format

The handshake packet contains a single character, which requires **1 bytes** of payload.

IMU Data Packet [Before week 7 evaluation]

Component	Value	Type	Size (in Byte)
Accelerator Data[3]	−32,768 to 32,767	Int (2 Bytes)	2 x 3 = 6
Gyro Data[3]	-0 to 1023	Int (2 Bytes)	2 x 3 = 6
Neutral Flag	0, 1	Int	1
Move Flag	0, 1	Int	1
Position Change	'L', 'N','R'	Char	1
Checksum	0-255	Int	1
End Delimiter	'\n'	Char	1
Variable Delimiter	';'	Char	1 x 9 = 9

Figure 4.1.3.2 Beetle data packet format

1. Neutral Flag: Indicates if the dancer is in neutral position
2. Move Flag: Indicates if the dancer is in motion
3. Delimiter: Split the data token

Data Packet format: 122; 124; -12414; 435; 12; 523; 0; 1; 64 \n

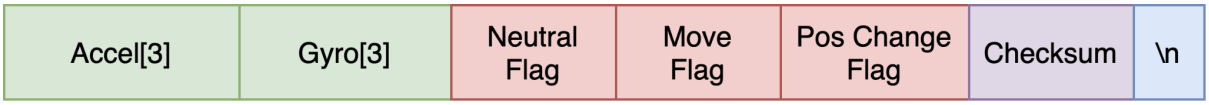


Figure 4.1.3.3 Data packet format

The sum of payload size is **26 bytes**.

IMU Data Packet [Final Implementation after week 7]

Design consideration is made by implementing the position change detection on the internal comms module using python. With this implementation, we save 2 bytes of payload from *Position Change* and *Delimiter*.

Figure 4.1.3.4 New data packet format

The new sum of payload size is **24 bytes**.

(The EMG data is not included in our final evaluation due to not having a dashboard. However, during the previous evaluation, the inclusion of emg data will lead to an additional 2 bytes payload.)

Response Data Packet [Final Implementation after week 7]

A new response data packet is implemented after week 7.

Packet Type	Code	Description
Connection Indicator	'C'	Send from laptop to beetle to indicate the connection status.

Figure 4.1.3.5 Beetle data packet format

This packet is used to request the beetle to continue reading IMU data and sending the packet to the laptop. It is implemented for **power saving**. The beetle will stop performing tasks once it stops receiving this C packet from the laptop. It helps in reducing power consumption during calibration and testing, and during the set-up stage in the actual evaluation.

Design Consideration on Packet Format

Remove timestamp from the BLE packet

We removed the timestamp component from BLE transmission due to the following reasons:

1. The time component obtained from Beetle is returning the number of milliseconds passed since the initialization of Beetle, which is affected by the time drifting overtime.
2. At low battery level, the milis time component often reset to zero due to restarting or internal resetting.
3. Including the timestamp in packets will significantly increase packet payload since the timestamp is recorded in milliseconds which is much larger than a 2 bytes integer.
4. The delay caused by polling and laptop data processing is much larger than the delay caused by BLE transmission, thus the delay of transmission is negligible.

In conclusion, the timestamp from the Beetle Millis is inaccurate due to drifting and power issues, which itself does not reflect a proper timestamp that is useful to the external comms. At the sametime, it introduces higher packet payload and additional cost of processing. Thus, it is not included in our final implementation.

Remove Position Change Flag from the BLE packet

Initially, the position change detection is done on Beetle in Arduino, and the result is transmitted in the data packet to the laptop.

However, since the position change flag is only sent once per dance move, including it in the packet will introduce higher complexity and payload. Since the position change detection only relies on acceleration data which can be buffered at the internal comms module too, I implemented the position change detection on laptop to achieve:

1. Lower Beetle Ram consumption
2. Lower packet payload which saved 2 bytes
3. Reduce complexity and delay on Beetle data processing

4.1.5 Laptop Processing

Laptop Tasks [Before week 7 evaluation]

No.	Tasks	Description
1	Establish Connection	Establish BLE Connection between laptop and Beetle
2	Handshake	Perform handshake by sending H and ACK packet.
3	Reading Packet from Beetle	Read incoming BLE packet from beetles at a constant interval
4	Sending packet to Beetle	Sending C response packet to indicate the request of data packet in the next 5 seconds.
5	Packet Processing	Process incoming packets from beetle by decoding and tokenization. Drop corrupted data packet.
6	Pushing packet to share memory queue for external communication.	Include receiving timestamp and flags after post-processing, and passing the formatted packet into a shared memory queue for external communication.

Laptop Tasks [Final Implementation after week 7]

An additional task is introduced to perform position change detection

No.	Tasks	Description
7	Position Change Detection	Detect position change using buffered acceleration data.

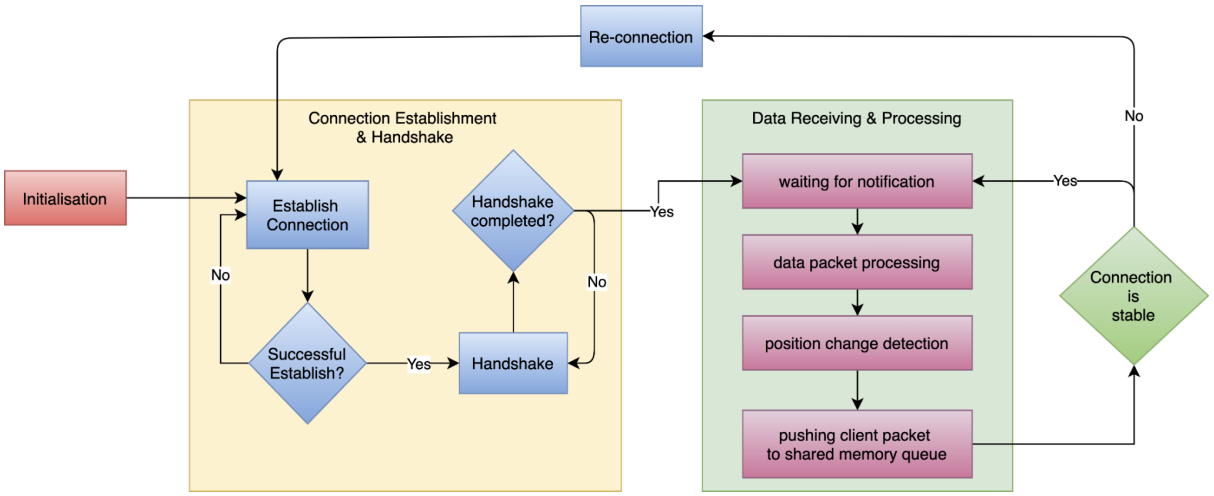


Figure 4.1.5.1 Laptop processing overview

Deliver Data Packet to External Communications

As illustrated in Figure 4.1.5.1, the decoded data packet will be reformatted and pushed to a shared memory queue for external communications.

Before Week 7

Before the week 7 evaluation, only deviceId, IMU data and motion flag are passed to external communications.

```
packet = {
    "Id": index,
    "GyroX": gyro_data_arr[0], "GyroY": gyro_data_arr[1], "GyroZ": gyro_data_arr[2],
    "AccelX": accel_data_arr[0], "AccelY": accel_data_arr[1], "AccelZ": accel_data_arr[2],
    "MoveFlag": motion_flag,
}
self.buffer_tuple.put(packet)
```

Figure 4.1.5.2 Packet format overview for ex-comms

Improvement in Final Implementation after Week 7

After week 7, the position change flag and time stamp upon receiving the packet are included in the client packet too.

```
packet = {
    "Id": index,
    "GyroX": gyro_data_arr[0], "GyroY": gyro_data_arr[1], "GyroZ": gyro_data_arr[2],
    "AccelX": accel_data_arr[0], "AccelY": accel_data_arr[1], "AccelZ": accel_data_arr[2],
    "MoveFlag": motion_flag, "PosChangeFlag": pos_change_flag,
    "Time": timeRecv
}
self.buffer_tuple.put(packet)
```

Figure 4.1.5.3 New Packet format for ex-comms

Why include a timestamp in the internal communication module instead of external communication?

The processing of data packets takes a significant amount of runtime, which is approximately 10ms on average. Thus, by including the timestamp upon receiving the packet allows ex-comms to consider the delay in processing data for clock sync.

Why include an Id in the packet?

The id indicates the Beetle Id. Since our ML module has a different model for each of the Beetles, the Id allows the ML module to keep track of which model to be used for prediction.

Why include MoveFlag and PosChangeFlag in the packet?

It allows external communications to keep track of the dancer movement, and only transmit meaningful packets which the dancer is dancing to the ML modules.

Position Change Detection [Final Implementation after week 7]

As mentioned in Section 4.1.3, the position flag is removed from the BLE packet. It means that the position change detection has to be done on the laptop.

Only data when the dancer is in a neutral position is considered to perform the position change detection. It means the dancer can only move left or right with the left hand put down at the side, which is illustrated in the diagram at the right.

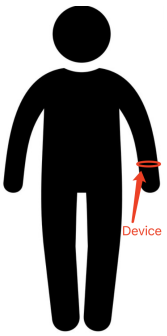


Figure 4.1.5.4 Dancer needs to move with hand down

The detection system makes use of a queue to buffer the previous y-acceleration data. The maximum queue size is set to 30 such that only packets received within the recent few seconds are taken into consideration.

With the number of packets in the queue exceeding 15, the detection is performed by checking the maximum and minimum buffer y-acceleration.

```
# moving right
if (len(yAccelDeque) >= 15 and minY < -70 and maxY > 90 and minYIndex < maxYIndex and maxYIndex - minYIndex >= 15):
    yAccelDeque.clear()
    return 1

# moving right
elif (len(yAccelDeque) >= 15 and minY < -90 and maxY > 70 and minYIndex > maxYIndex and minYIndex - maxYIndex >= 15):
    yAccelDeque.clear()
    return -1
```

Figure 4.1.5.5 Position change detection code

Why set the maximum queue to 30?

It takes about 2.5 - 3 seconds to fill up the queue, thus only packets received in the last 3 seconds are considered. It also reduces the runtime cost of iterating the queue in each loop.

Why only detect with queue size exceeding 15?

The detection makes use of the max and min acceleration. However, when a dancer stops walking, the acceleration will always exceed the threshold due to the slow down. This low acceleration should be ignored until 15 packets (1.5 seconds) later since it does not belong to the next slow down movement.

4.1.6 Reliability

Convert from Multi-processing to Multi-threading

Before week 7

Multithreading features are provided to ensure connection from multiple beetles. The *threading* library from native python module will be utilised. One thread will be used for data transmission for each beetle.

Final Implementation after week 7

Since each beetle is paired to a laptop, the multithreading implementation is removed, and replaced by a multi-processing architecture.

Why use Multi-Processing over Multi-Threading?

With integration of external communications modules, the tasks carried out are no longer limited to BLE connection itself. Multi-processing allows external communications to push packets to the Server for ML prediction, while reading data from the shared memory queue. At the same time, multi-processing allows internal communication to have its own memory space which reduces the complexity of integration.

```
72      inputQueue = Queue()
73      blunoProcess = Process(target=internal_comms.connect_to_pi, args=("p1", inputQueue, int(dancerID)))
```

Figure 4.1.6.1 Integration is done within 2 lines of code

Data Corruption Handling

Before week 7

Checksum is implemented to detect packet corruption. Upon an in-accurate checksum, the whole serial input string will be dropped. Such that a few packets will be dropped even though other packets might not be corrupted.

Final Implementation after week 7

Thus, after tokenizing the incoming BLE packets, only packets that failed the checksum test will be dropped.

Why not drop the whole input buffer that contains a corrupted packet?

With improvement on hardware implementation and power supply, the chances of packet corruption is nearly zero. It only occurs when the battery level is low.

Thus, it is no longer necessary to drop the whole buffer string. In fact, when there is a corrupted packet, it indicates that there is a need to replace the beetle with a new battery.

Timeout Mechanism

Before week 7

In case of missing ACK response during the handshaking, either from laptop or Beetle, a timeout mechanism will be implemented to restart the handshaking process.

The timeout threshold of the waitForNotification is set to 5 second.

Final Implementation after week 7

The timeout threshold of the waitForNotification is changed from 5 seconds to 0.5 second.

Timeout mechanism is also implemented for data packet transmission. When reponse from Beetle is beyond the timeout limit, a reconnection module will be called for rebuilding the connection between laptop and Beetle. The reconnection mechanism will be further explained in the next section.

Why set the timeout from 5 to 0.5 second?

The beetle is sending a data packet in a loop with delay set to 40ms. Considering the hardware processing time and delay for polling, each packet will take a maximum 100ms second to reach the laptop. Five seconds timeout could result in losing 500 packets, which will introduce a very high delay in obtaining the ML prediction. By setting the timeout to 0.5 second reduces the packet lost from 500 to 50.

Reconnection Mechanism

Before week 7

In case of disconnection, a reconnection mechanism will be carried out. The *btle.BTLEException* provided by the bluepy will be the indicator of calling the reconnection module.

Final Implementation after week 7

Despite reconnecting with *btle.BTLEException,* reconnection will also be triggered when

- No receiving of BLE packets for the past 3 seconds
- Timeout is triggered when waiting for a Notification.

Why implement additional conditions to trigger reconnection?

The BTLEException is only thrown when there is an issue in the connection. However, in case of slow packet transfer which will eventually result in BLE buffer overflow, performing reconnection immediately will reduce the number of corrupted packets. At the same time, the reconnection process can be completed within half an second, thus there will not be many packets lost.

4.2 External Communications

4.2.1 Client-Server architecture

The communications between the laptops and ultra96 board are implemented in a client-server structure with the 3 laptops acting as the clients and the ultra96 board acting as a server.

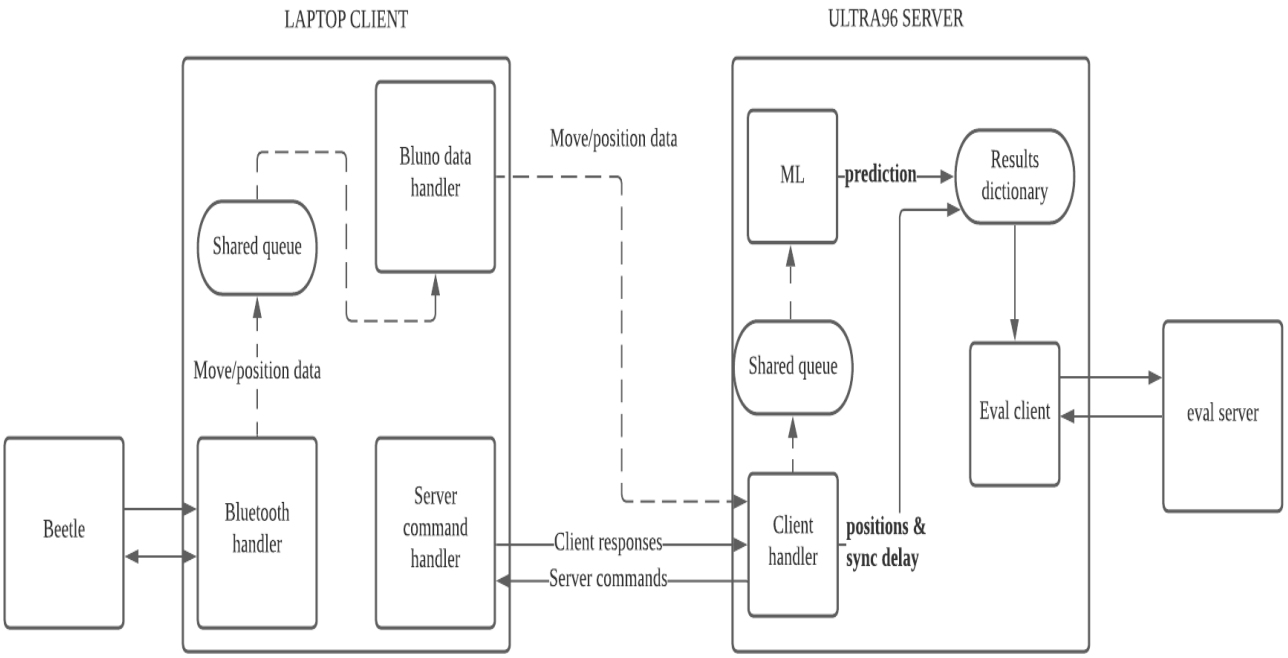


Figure 4.2.1 shows the architecture and data flow between one dancer and the server

For the purposes of this report, I will be referring to each of the components(threads or processes) as seen in the above diagram by name.

Things to note regarding Figure 4.2.1:

- Dotted line flows indicate the flow of data used by the ML to identify a move being danced
- Other line flows indicate movement of commands, responses, position data, sync delay information as well as events and locks used to signal and synchronize separate processes/threads.
- Rounded boxes indicate a shared data structure used to pass information between processes/threads
- Edged boxes represent a device, process or thread depending on the context.

Referring to Figure 4.2.1, the laptop client consists of 3 main components implemented as 3 concurrent processes:

1. Bluetooth handler: handles incoming bluetooth communications from the beetles and forwards data to process 2, explained in-depth in section 4.1
2. Bluno data handler: processes the incoming data, parses and sends relevant information to the ultra96 server
3. Server command handler: processes incoming server commands from the ultra96 and reacts/responds accordingly

Referring to Figure 4.2.1, the server client involves multiple multithreaded components including:

1. Client handler: One thread for each dancer/laptop to handle incoming communications from the laptop clients
2. Clock sync handler: One thread for each dancer/laptop to handle the clock synchronization protocol used to determine clock offset
3. Eval Client: One thread to handle communications with the evaluation server
4. ML: One thread to handle processing the data and feeding it to the artificial intelligence

4.2.2 SSH tunnel and encryption protocols

Communication between laptops and Ultra96 board will be achieved by the following steps for each dancer's laptop:
1. SSH tunnel will be established between laptop and sunfire
2. SSH tunnel will be established between sunfire and ultra96 board
3. Laptop will communicate with ultra96 via these 2 tunnels using socket binding

Secure communication between laptops <-> ultra96, ultra96 <-> eval_server is ensured via the use of AES encryption. A pre-shared key will be first distributed amongst the laptop users and loaded into the ultra96 board, evaluation server and dashboard. After this, AES-CBC encryption with padding is used to secure all comms between the laptops and the board. These messages will all be base64 encoded.

4.2.3 Library APIs

Library APIs:
- Python socket:
https://docs.python.org/3/library/socket.html
- Python base64
https://docs.python.org/3/library/base64.html
- Pycryptodome
https://www.pycryptodome.org/en/latest/
- sshtunnel
https://sshtunnel.readthedocs.io/en/latest/

4.2.4 Message format and message preservation

ultra96 to laptop

Messages sent from the ultra96 server to the laptop client will be commands which require a response. These will be in simple text format as follows:

1. "quit" : Tells the laptop client to close the TCP socket connection and exit all processes
2. "start": Tells the laptop client that the evaluation has started; Before this command is received, the laptop will not send any move data to the ultra96 server
3. "moveComplete" Tells the laptop that the current move has been evaluated and it is ok to stop sending move data; The laptop client will then wait for the next move before resuming the datastream to the ultra96 server

4. "sync": Initiates the clock synchronization protocol with the client, discussed later in 4.2.5

All of the above commands are received, parsed and handled by *the handleServerCommands* process in src/laptop/laptopClient.py

laptop to ultra96

All messages sent from the laptop client to the ultra96 server are sent in json format with a "command" key which indicates how the server should respond and what is contained within the "message" key, containing data or relevant information. The following is a table mapping each command to a feature and the message details. All the commands are sent

Command	**Purpose**	**Message contents**	**Server reaction**
shutdown	Tells the server that the laptop has received the "quit" command and that the connection has been closed	NIL	Closes TCP socket with the dancer/laptop client
clocksync	First response from client after server sends "sync" command	timestamp indicating client system time when the message was propagated (t1)	responds to client with the following json packet: {"command": "clocksync", "message": t2 | t3}
offset	Second response from client after server sends "sync" command	offset indicating the calculated clock offset relative to the ultra96 system clock	update clock offset for respective dancer locally
timestamp	Used to indicate that a dancer has started his/her dance move	Recorded timestamp when the dancer started dancing (relative to client system clock)	Adjusts the timestamp value using the clock offset calculated during synchronization, then updates internally
data	Sent during a dance move with data for the AI to use in its predictions.	Dictionary containing various values used by the AI to determine the current dance move	Parses, and stores the data in the shared data queue so that the ML process can access it
poschange	Sent before a dance move with data	Message contains one integer value in	Updates position data internally to be

	indicating a change in dancer positions	[-1,0,1] to indicate [left, no change, right] movement respectively	sent to evaluation server **discussed further in section 4.2.6*

Fig 4.2.4 Message formats from laptop to ultra96
**Note: For more information on t1-t4 regarding clock synchronization, refer to section 4.2.5*

Preserving message boundaries in TCP/IP

During the early stages of testing we noticed that the boundaries of messages sent via TCP/IP started to become unclear when the send rate increased beyond a certain value. This tended to happen specifically while the datastream containing move data recorded by the beetle hardware was being transmitted to the ultra96 server.

To combat this, all messages sent from the laptop client to the ultra96 server are delimited by a comma "," ASCII 44. Due to base64 encoding in which the comma is an invalid character, we would not need an escape character for the delimiter.

A recvall() function was implemented at the ultra96 server to ensure that:

1. Everytime data is received, all the received data is in the form of complete packets
2. The server is able to separate packets in the case where 2 or more packets are received at once

Here is how it was done:

```
def recvall(self,conn: socket.socket):
    fullMessageReceived = False
    data = b''
    while not fullMessageReceived:
        data += conn.recv(1024)
        if data[-1] == 44: # 44 corresponds to ',' which is delimiter for end of b64 encoded msg
            fullMessageReceived = True
    return data
```

4.2.5 Clock synchronization

Process (1 iteration)

During clock synchronization, the server first broadcasts the "sync" message which indicates to the laptop clients that clock synchronization is to be performed. Then, a message sent from a laptop to the ultra96 board will contain a timestamp denoting the time in which the laptop propagated the message through the network. This time will be known as t1.

On the Ultra96 board, the time in which the message was received will be denoted as t2. The Ultra96 board will then send a response packet back to the laptop from which the initial message was received, containing t2 as well as the time of propagation for this response packet, t3.

On the laptop again, the time which the network packet is received will be denoted as t4. RTT and clock offset are then calculated as follows:
- RTT = (t4-t1) - (t3-t2)

- clock offset = (t3-t4-RTT/2) OR (t2-t1-RTT/2) **Client takes average of the 2 values for higher accuracy*

The clock offset is then sent back to the server, where calculation of synchronization delay will be done.

Ensuring accuracy and combating clock drift

On the ultra96 server, a sanity check is performed for every clock synchronization to ensure that the clock offset calculated is reasonable. The ultra96 server, upon receiving a new clock offset from the client, will compare the values of the last 10 clock offsets.

If the variance of those 10 values are above a certain threshold (based on experimental data, it was decided to be threshold=1e-05), the server will request a repeat of the clock synchronization protocol by sending the "sync" message to the culprit client once again.

In order to combat clock drift, the clock offsets are updated before the evaluation and in between each dance move. The frequent updates are our system's defence against clock drift.

When is clock sync performed?

For every dance move, there is a lull period where less data is being sent from the laptop client to the ultra96 server. This is the period right after the artificial intelligence has received enough data to accurately determine a dance move.

It was decided that clock synchronization would take place during this period so as to reduce the effects on system latency and avoid interfering with the rest of the system since very little is being done at this point.

Even if the clock synchronization process stretches into the next move due to the variance exceeding the threshold, each iteration only takes roughly 40ms which means it will be completed by the time the dancers have finished changing their positions. Thus, it will not affect system latency as very little data is being sent/processed during this period (only 1 position change flag signal per change in position).

4.2.6 Position change mechanism

How?

Every data frame sent from the beetle contains a Position Change value (refer to table figure 4.1.3) which contains information on the change in a dancer's position. 1 indicates that the dancer moves to the right, 0 indicates he stays still, and -1 indicates that he moves to the left.

When a position change is detected by the laptop client, a "poschange" command is sent to the ultra96 server with the corresponding integer value (see fig 4.2.4). These values are stored on the server for each dancer and reset to 0 after every dance move in a list int[3]. For example, during the first dance move, if dancer 1 moves to the right, dancer 2 stands still and dancer 3 moves to the left, the list will be [1,0,-1], etc.

When it is time for the server to send data to the evaluation server, the Eval Client on the ultra96 server will process the changes in position to come to a conclusion on the final dancer

positions, and place those values into a list with the same indexing as the position change list. Using the same example, the dancer positions will be changed from [1,2,3] to [3,2,1].

The server handles these changes logically via a logic tree. It first identifies the movement of the middle dancer, and then logically checks the conditions for the other 2 dancers. An example code segment where the middle dancer does not move is as follows…

```
# middle dancer doesn't move
if positionChange[middleDancerIndex] == 0:
    # all dancers don't move
    if positionChange[rightDancerIndex] == 0 and positionChange[leftDancerIndex] == 0:
        print("[UPDATE DANCER POSITIONS][NO CHANGE]", dancerPositions)
        positionChange = [0,0,0]
        return

    elif positionChange[rightDancerIndex] < 0 and positionChange[leftDancerIndex] > 0:
        temp = dancerPositions[0]
        dancerPositions[0] = dancerPositions[2]
        dancerPositions[2] = temp
        print("[UPDATE DANCER POSITIONS][RIGHT AND LEFT SWAP POSITIONS]", dancerPositions)

    else:
        print("[UPDATE DANCER POSITIONS][INVALID CHANGE]", dancerPositions)
```

The same is done for the cases where the middle dancer moves to the right or left and can be found at line 36 in *src/ultra96/evalClient.py*.

Why?

- **Flexibility**: Because the dancer positions are only evaluated when the ML has successfully identified the dance move, dancers have ample time to correct wrong position changes during the dance move itself.
- **Room for error**: Because the logic only takes into account the direction of movement and does not care about how many times the laptop client and hardware detects a right or left movement, the dancer does not have to worry about moving a specific distance and only needs to worry about crossing the threshold for movement where a position change is detected.

4.2.7 Internal Signals and Sequencing

Laptop Client

The laptop client makes use of the following events/locks for synchronization and sequencing:

1. **evalStarted** *Event*:
   This event is used as a flag to signal that the evaluation has started. While it is not set, it means that the server has not sent the "start" command to signal the beginning of the evaluation and that all data coming from the bluno should be ignored. The event acts as a gate to the rest of the process. Before receiving the start command, nothing on the laptop client (other than clock synchronization) can proceed.

2. **moveStarted** *Event*:
   This event is used as a signal to indicate to the laptop client that the dancer has started dancing. It is set when a flag is sent from the bluno is received indicating dance

movement from the dancer. (Move Flag == 1, refer to fig 4.1.3.2).

If this happens while moveStarted is not set, it indicates that the packet is the first packet of the dance move, signalling that the laptop client should send a timestamp indicating the start of the move. In the same vein, If moveStarted is set, it means that the move is already in progress and no timestamp should be sent.

When the "moveComplete" command is received from the server, this flag is cleared so as to indicate the completion of the previous move.

Ultra96 Server
The ultra96 server has many flags and locks which are used to do signalling both internally within each component(thread), or between them.

Internal:

- Server signalling (internal within *src/ultra96/server.py*):
    - clockSyncResponseLock *Event*: Since clock synchronization with clients is done 10 times successively when triggered, this lock ensures that the received responses from the laptop client are mapped to the correct clock synchronization request, preventing further requests until the previous set of responses has been received and handled.

Cross process:

- doClockSync *Event*: Signals to the clock synchronization handler that clock synchronization should be carried out, set by the ML handler every time enough data has been received to send a result to the evaluation server.
- globalShutDown *Event:* Signals to all processes to shutdown
- dataQueueLock *Event:* Signals the end of the current move. Set by the ML process every time enough data has been received to process and predict a move. Locks the data queue in between the client handler and the ML process to prevent any more data from being unnecessarily added to the queue. Cleared by the client handler when a new move has started, signalled from the Client Handler.
- rdyForEval *Event:* Signals that the ML has successfully predicted the current move and that the Eval Client can send the results to the evaluation server. Set by the ML process and cleared by the Eval Client once data has been successfully sent to the evaluation server.

4.2.8 Additional network optimizations

By coding the laptop client in such a way that it only selectively sends data when required, and idles when not, network usage and system latency is reduced, as well as power used by the ultra96 in receiving messages from the open TCP connection.

We achieve this by selectively choosing when to send data received from the beetle:
At many points throughout the evaluation, data recorded by the hardware sensors and sent to the laptops can be ignored:

- Before the evaluation begins, no data needs to be sent
- While the dancer is performing a position change, no data needs to be sent to the server apart from the singular position change flag
- From the moment the ML process has enough data to accurately determine the dance move, no more data needs to be sent from the laptop until the beginning of the next

move. This is achieved by the use of the "moveComplete" flag as well as the internal signalling performed by the laptop client as described in sections 4.2.4 and 4.2.7

5. Software Details

5.1 Software Machine Learning

5.1.1 Segmentation of Data.

Data segmentation is used to separate and meaningfully group data into similar traits. Since it is a real-time stream of data that the system is receiving, we would have to specify the amount of data we will like to segment so as to capture enough data to extract features from for a particular move.

To help with the segmentation of the data, each movement will be made distinct using a start or stop move from the dancer. This will help better indicate a move made by the dancer. This will be shown from the data provided by the accelerometer where the signal will reveal a pattern or a threshold that is being exceeded. This threshold is measured by the sensor side. Hence by the time segmentation is performed, we will know that the data is only collected when the move is performed. This pattern recognition and threshold detection will allow us to segment the data to detect the dance moves of the dancer by looking at the features we intend to explore.

The raw training data received from the hardware includes the flag for indicating a move has started, the ID for each dancer, timestamps, acceleration (AccelX, AccelY, AccelZ) and gyroscope (GyroX, GyroY, GyroZ) data. The first 3 columns are data used by hardware and external comms to determine which dancer is dancing and if they have started a move. Manually, I removed the first 3 columns for both training and evaluation of the model so as to prepare it for segmentation and feature extraction.

We used the rolling window segmentation from the pandas.DataFrame.rolling function. When choosing the window size, we experimented with various window sizes on the basis of how much data is generated for a single move. We initially had a window size of 30 however we realised we were not capturing enough data to extract and this resulted in having a model that was not accurate enough. We figured that perhaps a complete dance move required more than 30 data points. Hence we measured the average amount of data it actually took to capture any of the moves required. Based on our findings, we then changed the window size to 50.

When deciding the overlapping window, we decided to have an overlap of 50% as we wanted to capture as many points from the previous window while ensuring we do not capture too much data from the previous window as well. We do not want to capture too much from the previous window as we do not want to both train and predict the same features as the previous window. However we also want to capture as much data from the previous window so as to As shown in Fig. 5.1, the difference between having a 50% overlap, and others is that one can overlap too much of the data while one does not capture as much data from the previous round. Overlapping windows allow us to ensure that we try to capture as many features for the particular move that we are trying to predict.

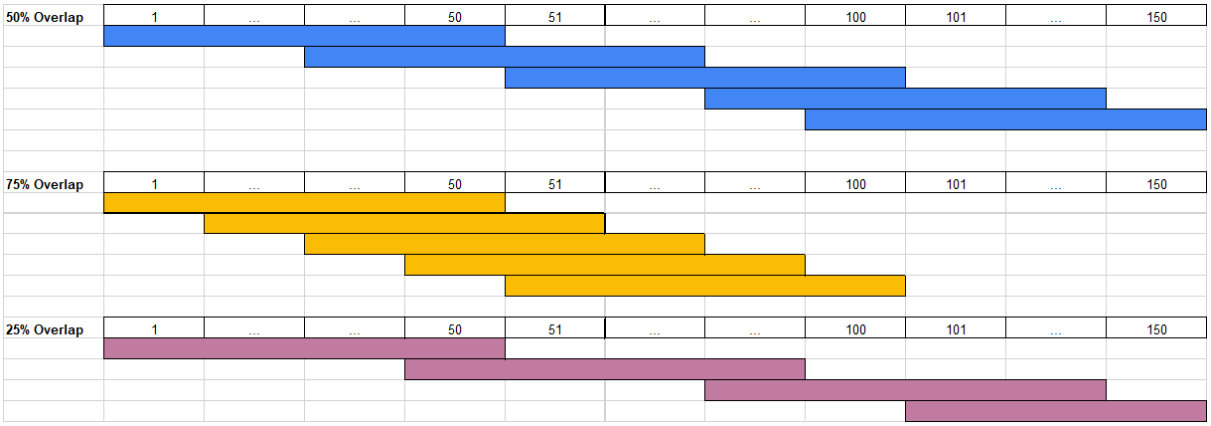


Fig. 5.1. Overlapping Windows

5.1.2 Feature selection

Selection of features is the process of selecting which traits or characteristics will be used to predict the moves.The features we will be exploring will be chosen by examining which characteristics or features (time-domain or frequency domain) can be used to predict the dance moves. By doing so, we can reduce the amount of irrelevant data/features to classify. This reduction may also make the classification more accurate. The computational workload to run machine learning models will also reduce. For this specific project, we will be looking into time domain features such as mean, variance and standard deviation.

Extracting features like the minimum and maximum points of the data allows the model to train for moves that show a spike in the acceleration and gyroscope data. For example, a side pump will show a huge spike in the AccelY data column. To emphasize this feature we get its minimum and maximum of the collected data segment. Other features to extract like skewness, kurtosis and correlation provide data features that bring out the relationship between acceleration and gyroscope data.

5.1.3 Machine Learning Models

This project is faced with a Multi-Classification problem as it requires us to identify various dance moves. For the Machine-Learning Algorithms, we explored traditional machine learning algorithms such as a Support Vector Machine (SVM). For non-traditional algorithms, we used neural networks such as a Multilayer Perceptron (MLP) and a Convolutional Neural Networks (CNN).

SVMs are good for binary classification but it can also be tweaked to solve multi-classification problems. This can be done by breaking down multiclass problems into multiple binary classification cases. We may need to transform the data non-linearly as SVM requires the use of a linear classifier.

MLPs are a generic implementation of a feedforward neural network. The features that we have selected/extracted act as the input node while the categorisation of the dance moves act as the output nodes. There are hidden layers in between the input and output nodes. These

hidden layers and outer layers have nodes whereby their values are calculated from the previous nodes and multiplied by the trainable weights with a bias added to the sum as shown in Fig 5.2. (Shukla, 2019)

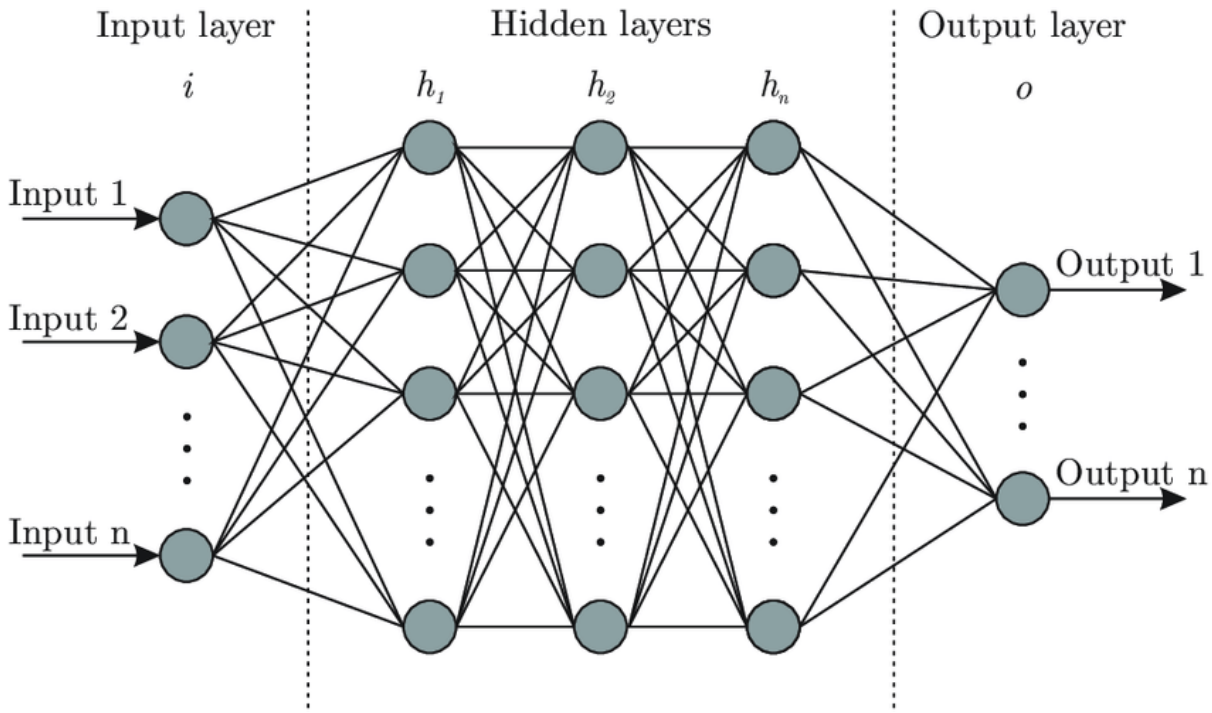


Fig 5.2 Neural Network Structure

This weighted sum is passed into an activation function before it becomes the value of the current node and the cycle repeats to the outer node as shown in Fig 5.3. (Ganesh, 2020) The activation functions we use are softmax and rectified linear units (ReLU). Softmax is a mathematical function that takes an input vector of numbers and normalizes it into a vector of probabilities. (Brownlee, 2020) ReLU on the other hand is a piecewise linear function that will return the input if it is positive and 0 for negative values. (Brownlee, 2019)

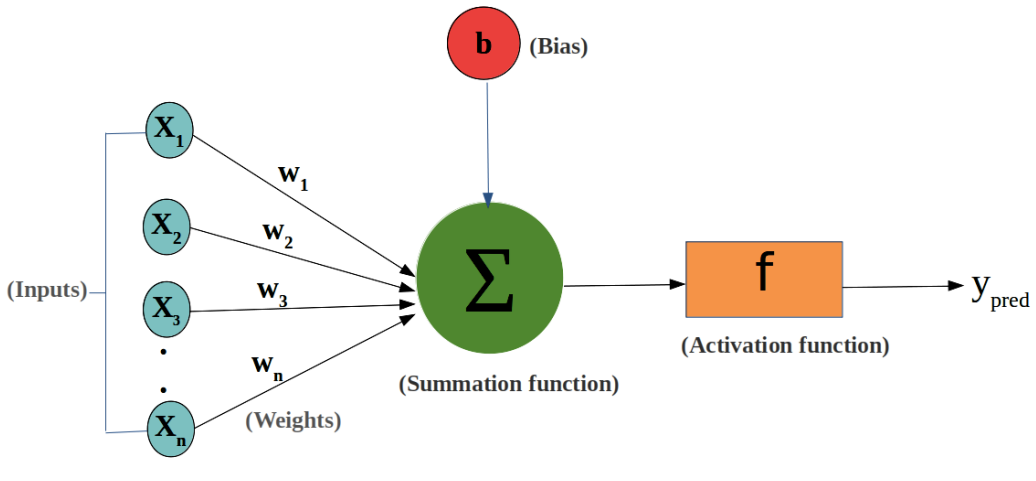


Fig 5.3 Weights & Bias in a Neural Network

When training the model, the values are passed through for each iteration and the hyperparameters are trained such as the number of hidden layers, the number of nodes in each layer, the activation function, the loss function, number of epochs to train the model, and other aspects of the model structure.

CNNs are another type of feedforward neural network and are similar to MLP, the data is passed through convolution and pooling layers. Convolutional layers mimic how the human visual brain works. (Wood, -) Pooling layers provide a way to down sample feature maps by summing up the presence of the features in batches within the map. (Brownlee, 2019) CNNs are ideal for processing image data due to local translation invariance and it could be effective for categorising dance moves at any point in the segment. When training CNN, it is similar to that of MLP though there are additional hyperparameters in the number of filers & pooling used when convoluting the filter with the data.

When comparing between MLP and CNN, there are more advantages that MLP has. MLP takes a much shorter time to compile and train. MLP has fewer hyperparameters, this meant that it was easier to optimise. One advantage that CNN has is that its local translation invariance is better verified compared to MLP which helps with identifying moves from any point in the window. However since there is already an overlapping sliding window in place, the MLP will be trained to classify it correctly as the overlaps try to cover as much data. Hence the translation invariance is still ensured. Therefore, MLP is the model of choice for classifying our dance moves.

5.1.4 Training of Model

Before training with the actual data, and while trying out various models, we used the data from a study that recorded 30 participant's performing daily activities, namely Walking Upstairs, Downstairs, Sitting, Standing and Laying. (UCI Machine Learning, 2020) The data was very similar to what we wanted to do hence we decided to train with this dataset first.

Once the wearables were ready, we collected 1000 data points for each person. This meant there were at least 5000 data points collected for each move. We also had to ensure that the data collected for each dance move is standardized so as to ensure that the model was trained well to the particular move that was required. While collecting these data, it was also imperative that we collected them with varying speeds and tempo so as to have a more generalised sample for the model to train and not trained towards a specific tempo.

In terms of hyperparameter tuning, we wanted to find the best combination of hyperparameters that produced the best neural network performance. We determined the hyperparameters to tune which are the number of hidden layers, the number of nodes in each layer, the activation function, the loss function and number of epochs to train the model. We also have an optimiser, namely the Stochastic Gradient Descent (SGD) as we realised it would result in a better model.

5.1.5 Model Validation and Testing

For model validation, we used 2 methods namely, a train test split and the K-fold cross validation. For the train test split, we used a library from sklearn, sklearn.model_selection.train_test_split. This helps split the data set randomly into training and testing data. Although the data we are receiving is time series based, we have already preprocessed the data into segments hence it would not matter when using this method.

For K-fold cross validation, we will split the data into K groups and use K-1 groups to train the model while using the remaining group to test the model where we define K to be any number. One variant of this validation is leave-one out cross validation. However this is too computationally intensive hence, for the dataset we received, we used a 5-fold cross validation shown in Fig. 5.4. (sckit-learn, -)

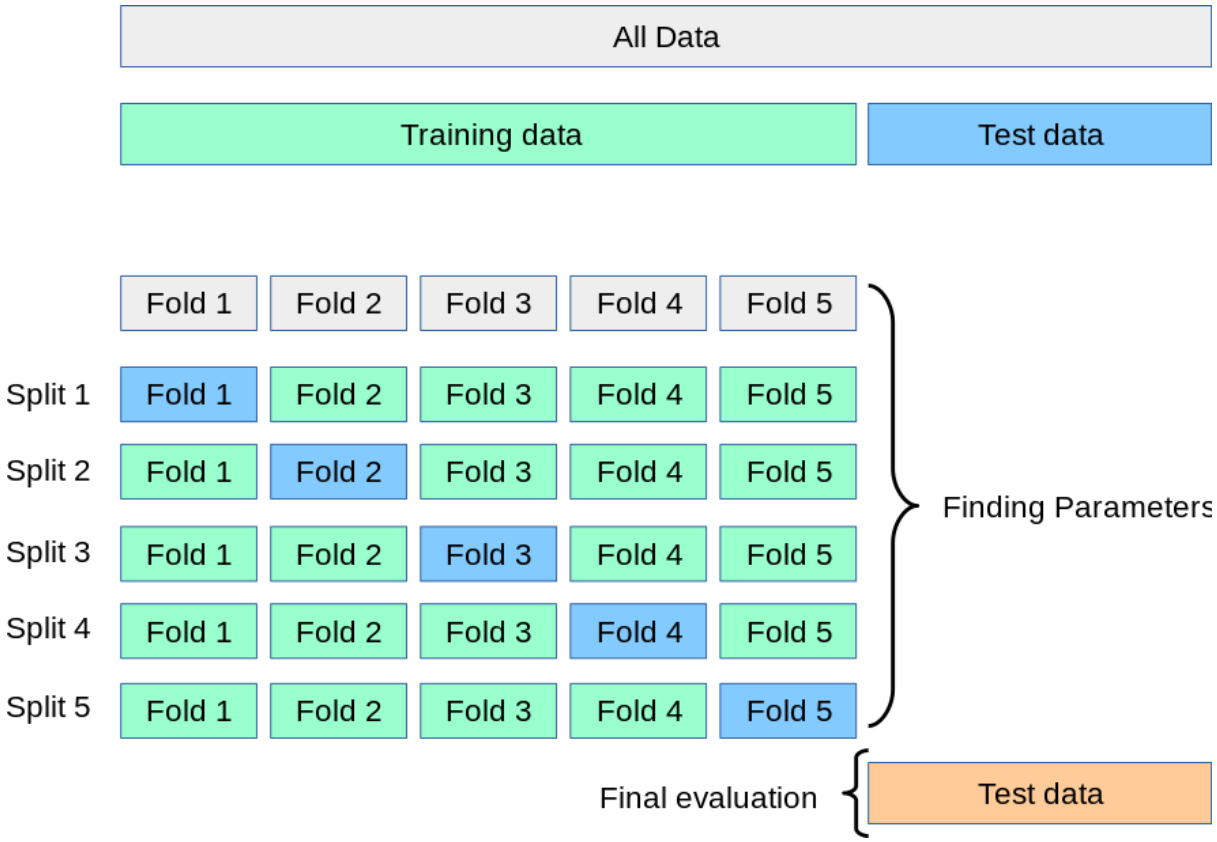


Fig 5.4 Cross Validation

Once the model was trained and we had a high enough accuracy, we tested the model with a smaller dataset of about 100 data points to mimic the evaluations. We also used a confusion matrix to determine the accuracy of our model and check back. We then collected batches of around 100 data points and tested to check the outputs and calculate the accuracy. Finally, we then tested it in real time to ensure the preprocessing was done right and to determine the real-time accuracy of the model.

6. Project Management Plan

Week	HW1 (Sensors)	HW 2 (FPGA)	Comms 1 (Internal)	Comms 2 (External)	SW 1 (ML)
3	Individual Role Research				
4	Design Report (5% Design Report Section 1 & 2, 10% Individual)				
	- Complete report and solder one set of IMU Sensor + Bluno Beetle - Buy batteries and charging units	Research into potential hardware acceleration for Neural Networks and provide feedback to SW1 on selection of NN to use Complete design report	Complete design report. Finish initial setup and configuration for Beetles and BLE scan.	Complete individual subcomponents for aes encryption, TCP communication and clock sync	Complete design report and research ML Models
5	Feedback on design Report				
	- Write arduino code to detect the start of dance and end of dance - Test MegunoLink's exponential filter on the Acceleration data	Prepare Development environment Begin implementing Brevitas for generation of ONNX neural networks from given model	Complete transmission of dummy data. Explore the suitable latency for packet transmission.	Modify previous week's work based on feedback	Find Dummy Data and Work on SVM and CNN.
6	Individual Progress Checkpoint (5%)				
	- Research how to detect displacement using the IMU sensor	Work on Synthesis of HLS layers and IP stitching	Implement multiple threading features for connection with multiple beetles.	Integrate all subcomponents, using dummy processes to spoof communication from the	Continue work on SVM and CNN.

				Bluno and machine learning output	
RW	- Design the 3D Model for the device housing in Fusion360 - Integrate system with internal comms	Begin doing performance testing on hardware	Complete the integration with sensors and external comms. Test the IMU data transmission with real data from the sensors.	Complete integration, do final checks to ensure requirements for week7 are met	Use Dummy Data for SVM and CNN models.
7	Individual Component Test (20%)				
8	- Resolve issues from 1st evaluation that are related to hardware and sensors	Begin doing implementation of ML model structure (Weights and layers might not be finalized)	Improve reliability based on feedback upon integration.	Integrate components with a single laptop and Machine Learning output	Gather data from wearables and start training the model and tweak any implementation if necessary.
9	Single Dance System Progress Checkpoint				
	- Additional tweaks for 1st evaluation	Optimization explorations and bug fixes	Tweaks for 1st evaluation.	Integrate components with 3 laptops	Continue training the model with the actual data.
10	- Resolve issues from 1st evaluation that are related to hardware and sensors	Optimization explorations and bug fixes	Modify and improve programs based on feedback from 1st evaluation.	Do testing on 3 laptop system	Continue training the model. Tweak if necessary.
11	3 Dancers System Demo. Peer Review.				

12	- Resolve any issues from 2nd evaluation test	Finalize implementation, ensure it is bug free and reasonably performant	Final testing with all improvements and bug fixing.		Finalise the model.
13	Final Demo/Final Design Report				

7. Societal and Ethical Impact

7.1 Security Concerns

Our communication between the laptop and ultra96 server involves the handling of some sensitive data including, potentially data regarding the training regimen of a dance group or the data that could compromise the dancers' intellectual property.

For example, a bad actor could find personal data about the dancers' training schedule and how often they practice, leaking it to paparazzi or other malicious parties. Their intellectual property could also potentially be stolen if their unreleased dance choreography is released or sold to malicious parties.

Even though the data is encrypted with AES CBC encryption, it is always possible that someone is able to crack the encryption or learn the secret key, especially since it currently has to be typed manually into the evaluation server and the key is static and unchanging.

7.2 Concerns regarding dance as an art form

Dancing is, generally speaking, a lot about the visual impact and artistic vision of the group that is performing. In the case of the system we have created, it is possible that their original intention or vision regarding the performance is hindered by the restrictions placed on them with regard to how they perform a move (move prediction via AI) and timing restrictions (synchronization delay).

For example, the group may have an idea where the center dancer is more "flashy" and thus performs the move differently than his/her group mates. In this case, it is possible they are evaluated as dancing wrongly and forced to move identically. In addition, if they choose to delay or time their moves so as to create an intentional visual impact, they would be evaluated as "out of sync". These restrictions would greatly hinder an art form and must be taken into account.

In addition, the overemphasis on statistical perfection could make the dancers focus too much on certain parts of their performance while ignoring those that cannot be tracked by the system that would otherwise add value to their performance. E.g facial expressions, more subtle movements, etc.

7.3 Ethical Concerns

As with all technologies involving machine learning, we must always be mindful of potential ethical issues.

With the rise in popularity of wearable smart devices such as smart watches, we have to ensure that move detection technology does not become used for things such as detecting what a wearer is doing in order for the wearable company to collect data on them. One example would be training a wearable to detect if a person plays computer games through their wrist movements in order to send them targeted ads.

With data privacy becoming a more and more controversial topic, any form of technology that can generate more data insights about a user has to be carefully considered.

8. References

Kala, S., & Nalesh, S. (2020). *Efficient CNN Accelerator on FPGA.* IETE Journal of Research, 66(6), https://doi.org/10.1080/03772063.2020.1821797

Qiao, Y., Shen, J., Xiao, T., Yang, Q., Wen, M., & Zhang, C. (2016). *FPGA-accelerated deep convolutional neural networks for high throughput and energy efficiency.* Concurrency and Computation: Practice and Experience, 29(20), e3850. https://doi.org/10.1002/cpe.3850

Shukla, L. (2019, - -). *Weights and Biases*. Designing Your Neural Networks. https://www.kdnuggets.com/2019/11/designing-neural-networks.html

Ganesh, S. (2020, Jul 25). *What's The Role Of Weights And Bias In a Neural Network?* What's The Role Of Weights And Bias In a Neural Network? https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f

Brownlee, J. (2020, Oct 19). *Softmax Activation Function with Python*. Softmax Activation Function with Python. https://machinelearningmastery.com/softmax-activation-function-with-python/

Brownlee, J. (2019, Jan 9). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. A Gentle Introduction to the Rectified Linear Unit (ReLU). https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=The%20rectified%20linear%20activation%20function,otherwise%2C%20it%20will%20output%20zero.

Wood, T. (-, - -). *Convolutional Neural Network*. Convolutional Neural Network. https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network

Brownlee, J. (2019, April 22). *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

UCI Machine Learning. (2020, - -). *Human Activity Recognition with Smartphones*. Human Activity Recognition with Smartphones. https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones

sckit-learn. (-, - -). *Cross-validation: evaluating estimator performance*. Cross-validation: evaluating estimator performance. https://scikit-learn.org/stable/modules/cross_validation.html