CG4002 Computer Engineering Capstone Project

2022/2023 Semester 1

“Laser Tag++ Ultra 5G”

Design Report

Group **B18**	Name	Student #	Specific Contribution
Member #1	Lin Yuheng	A0199409M	Hw Sensors
Member #2	Zhang Haoyu	A0220591M	Hw AI
Member #3	Nishant Rai	A0222650R	Comms Internal
Member #4	Kwek MingShun	A0206038M	Comms External
Member #5	Duan Yuhang	A0201815R	Sw Visualizer

Section 1 System Functionalities (Joint Work)

1.1 Feature List

Core features include:

- Laser tag guns (using IR receivers and transmitters) that can shoot at an opponent and accompanying target suits that can detect the shots from up to 3m away.
- Wearable accelerometers to detect players' hand motions
- Predicts possible motions that match the following actions using machine learning algorithms:
    - Deploying a shield, which protects the player from taking damage for 10 seconds
    - Throwing a grenade
    - Reloading when the player is out of ammo
- Software visualizer that acts as a Heads Up Display (HUD) for the players to see the following details:
    - Current health
    - Remaining ammo
    - Remaining shield uses
    - Remaining grenades
    - Grenade impact when thrown
- Game engine to keep track of game state at all times

Possible features to be added include:

- Immersive sound effect from the phone speaker
- Particle effects in HUD for firing a gun, scoring a hit on the enemy, grenade explosion
- Additional models overlaid on top of the enemy player & enemy weapon in HUD for added immersion

Section 2 Overall System Architecture (Joint Work)

2.1 Overall System Architecture

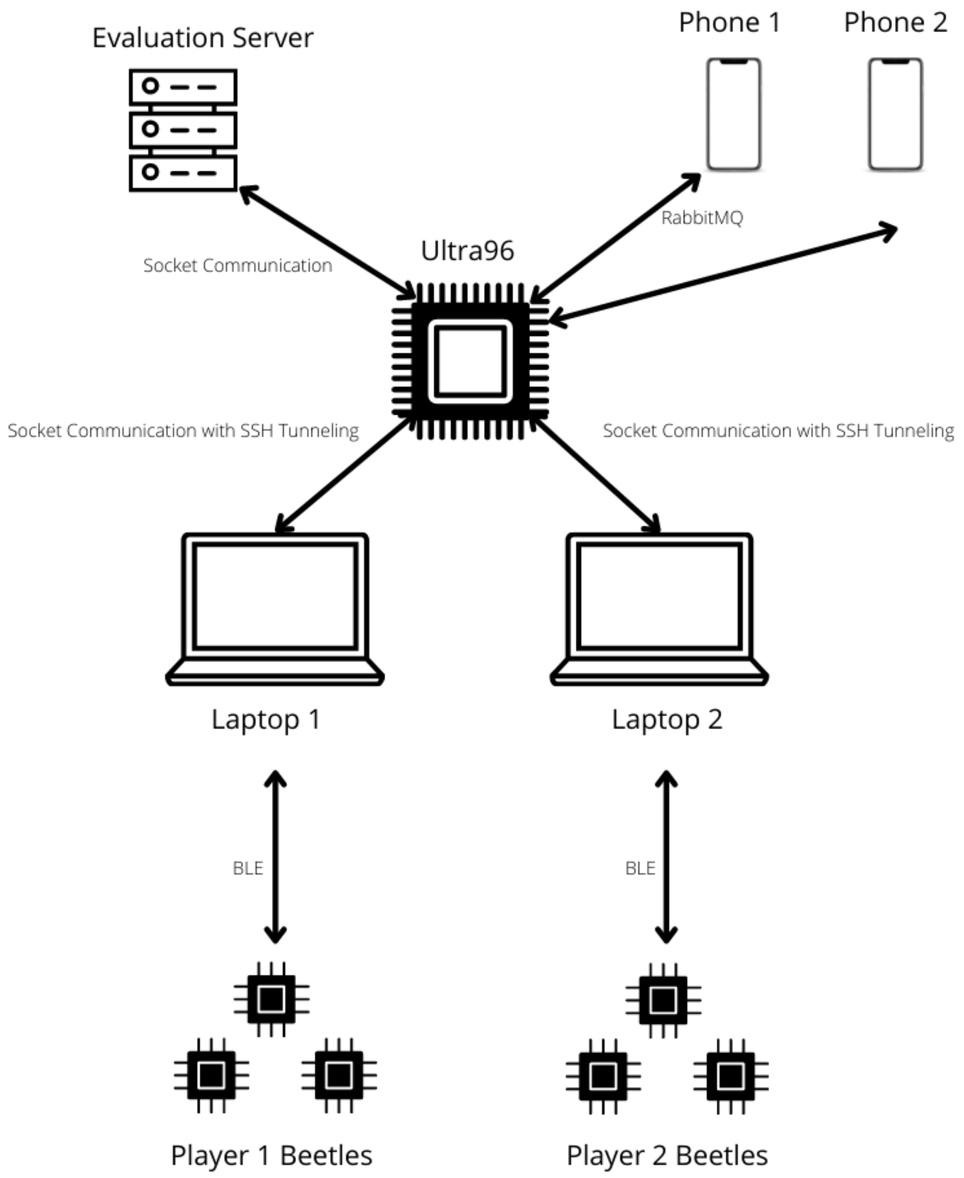


Fig 2.1 Overall System Architecture

Raw data is processed by the Ultra96, which is the main component of the system. An overview of the architecture that would be used in our project can be seen in the image above. Communication between the laptops and Beetles is through BLE outlined in

Section 5. Meanwhile, communication between the Ultra96 and the evaluation server as well as laptops uses socket programming, while communication between Ultra96 and the Visualizer uses MQTT (MQ Telemetry Transport) protocol with RabbitMQ as the broker. Section 6: External Communication explains how data is encrypted and transferred in and out of the Ultra96.

2.2 Hardware Components

2.2.1 Beetle BLE

The Beetle is an Arduino Uno based board with Bluetooth 4.0 (BLE) capabilities. It will be used to process sensor data and send it to the laptops via BLE.

2.2.2 IR Transmitters

Upon a player pulling the trigger of their gun, a signal will be sent to the transmitter module by the beetle to transmit an IR signal out of the gun.

2.2.3 IR Receivers

The IR receivers will always be active. Upon receiving an IR signal, a signal will be sent to the Beetle to indicate that they have been shot.

2.2.4 MPU6050

An MPU will be used on each glove to measure the 3 axis acceleration and 3 axis gyroscope. This data will be forwarded to the Ultra 96. Through feature extraction from these data the action the player is making can be deduced.

2.2.5 Software Visualizers

Two phones will be used as visualizers for each player to be able to see game information such as health, ammo, etc. during the game. These will be mounted on the gun.

2.2.6 Ultra96

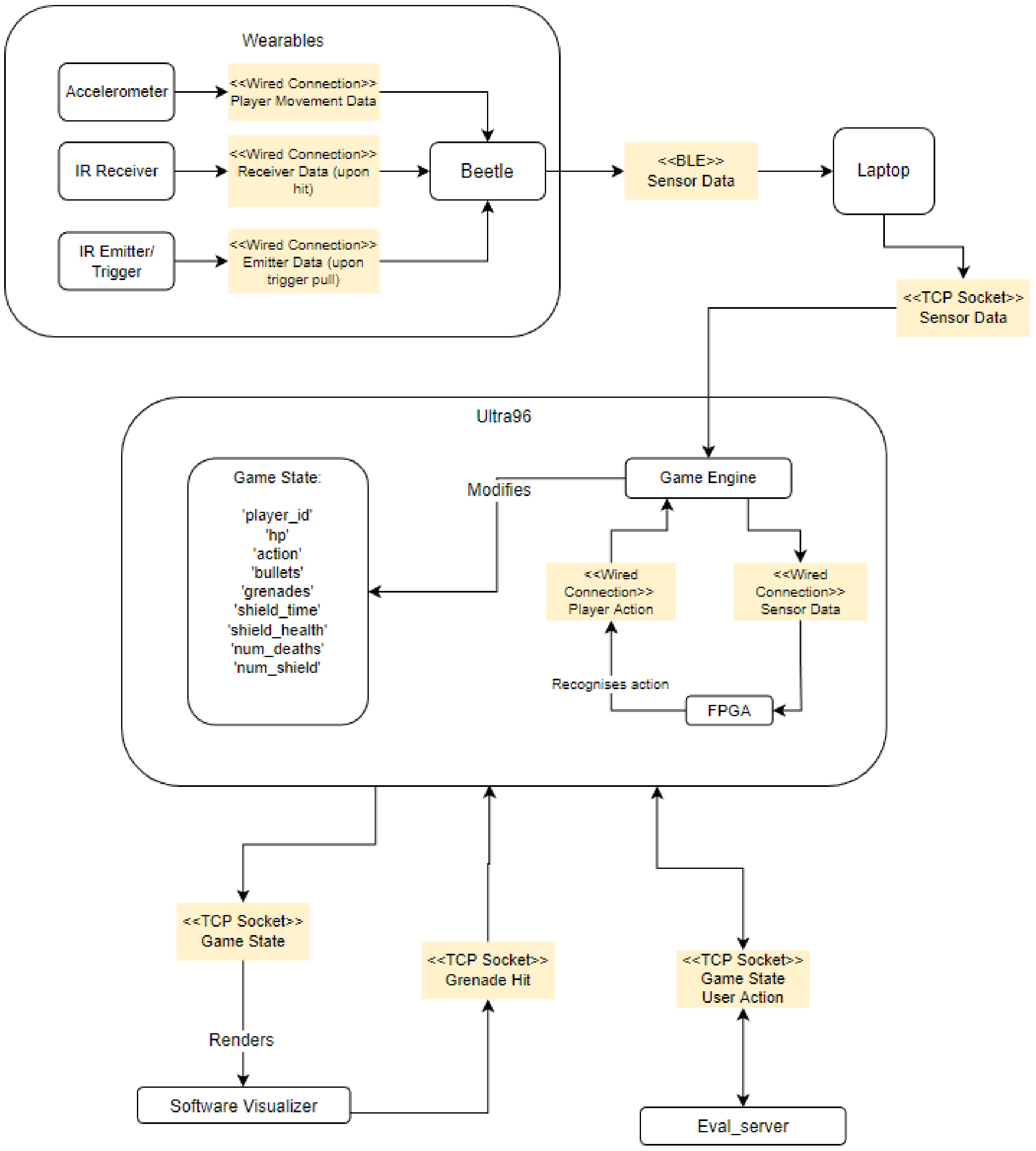
The Ultra96 is a development board which has 64-bit Arm architecture coupled with Xilinx programmable logic (FPGA). The Ultra96 will receive raw sensor data from the Beetles, carry out segmentation and feature extraction, and feed the feature vectors to the neural network; the FPGA on Ultra96 provides hardware acceleration for the neural network model which is implemented on the FPGA. The activity detection output will be sent to the software visualiser and game engine.

2.2.7 Linux Laptops

Two laptops running Linux will be used to receive sensor data from the Beetles, then transmit it to the Ultra96. The communication protocol between the Beetles and laptops is detailed in Section 5, while that between the laptops and Ultra96 is detailed in Section 6.

2.3 System Interfaces

The diagram below shows the connections and interfaces between all components. Components are in white and their connections/interfaces are in yellow.



Wearables
Accelerometer
<<Wired Connection>> Player Movement Data
IR Receiver
<<Wired Connection>> Receiver Data (upon hit)
IR Emitter/ Trigger
<<Wired Connection>> Emitter Data (upon trigger pull)
Beetle
<<BLE>> Sensor Data
Laptop
<<TCP Socket>> Sensor Data
Ultra96
Game State: 'player_id' 'hp' 'action' 'bullets' 'grenades' 'shield_time' 'shield_health' 'num_deaths' 'num_shield'
Modifies
Game Engine
<<Wired Connection>> Player Action
<<Wired Connection>> Sensor Data
Recognises action
FPGA
<<TCP Socket>> Game State
Renders
Software Visualizer
<<TCP Socket>> Grenade Hit
<<TCP Socket>> Game State User Action
Eval_server

Figure 2.3 Connections and Interfaces

2.4 Intended Final Form

The diagram below shows a mockup of the intended final form of the system, including placement of hardware components on the player.

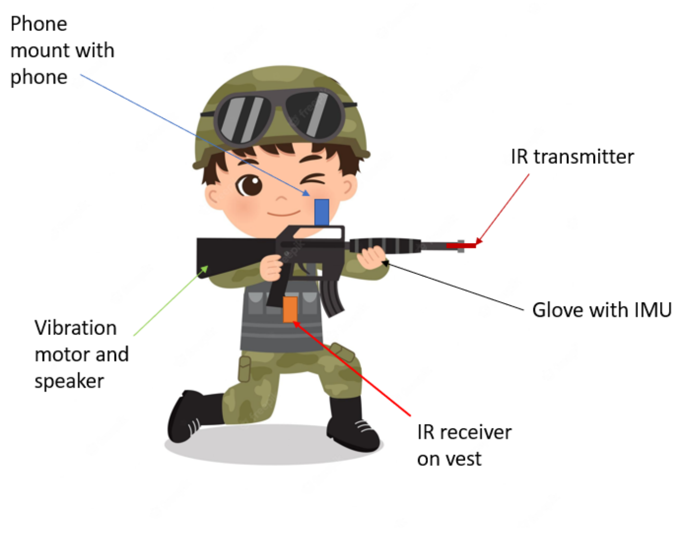


Figure 2.4 Intended Final Form Of System

2.5 Main Algorithm

The overall algorithm of the system is as follows:

1. startGame()
2. while(gameNotEnd)
    a. recvFromBeetles() - Beetles read sensor data and send it to the laptops, which then send it to the Ultra96
    b. runAIAlgo() - Ultra96 conducts feature extraction on received data and detects any player motions, and updates game state if needed
    c. sendToPhones() - Ultra96 sends relevant information (e.g. grenade thrown) to software visualizer
        i. recvFromPhone() (optional) - if software visualizer detects a grenade hit, this information is sent back to the Ultra96 to update the game state
    d. sendRecvEval() - Ultra96 sends the current game state to eval_server for evaluation, and receives the intended game state from the eval_server as well in the case of any discrepancies
    e. checkGameOver()

Section 3 Hardware Sensors (Yuheng)

The following section provides in detail the list of hardware components (with their respective product pages/ documentation) that will be used and the supporting components that are required to work.

3.1 Hardware Summary

Component	Quantity	Description
Bluno Beetle	6	Divided into 2 sets for 2 players. Used for data collection and low level data processing from each of the 3 components (gun, vest and glove) before transmitting the data to the laptop through bluetooth.
Vibration motor NFP-E0724-B	4	1 on vest and 1 on shooting apparatus to simulate recoil and injury (to be used with L298N motor driver and LM3940 voltage regulator to step down the voltage to 3.8V)
Thin Speaker - 0.5W	2	To be attached on shooting apparatus for shooting sound
MPU6050 (GY-521) Library: MPU-6050 6-axis accelerometer/gyroscope Arduino Library	2	To be placed on glove for motion detection hence predict the action It provides 3 axis acceleration
IR transmitter	2	To attach on shooting

		apparatus
IR receiver	2	To attach on vest for detecting shot fired by the gun

3.2 Component Description

3.2.1 Shooting apparatus

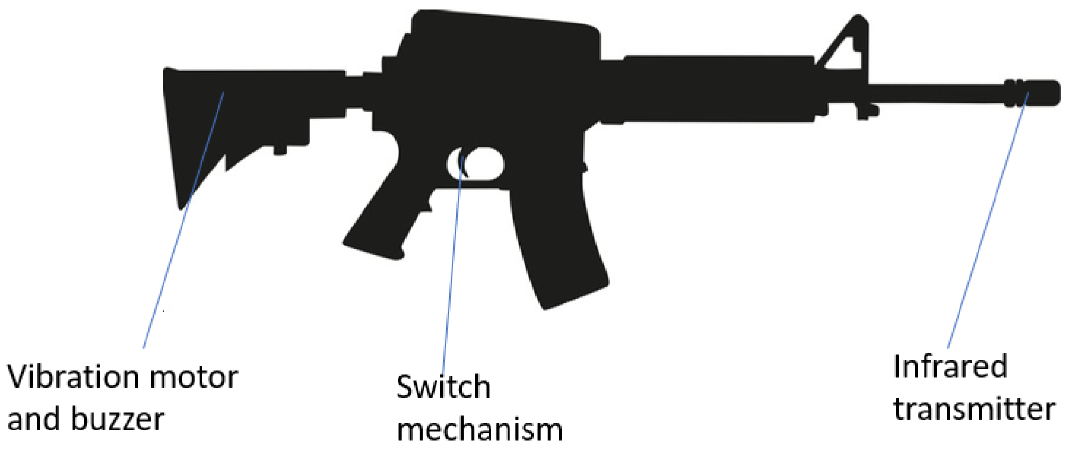


Figure 3.2 Example position of components on shooting apparatus

Shooting apparatus will be 3D printed with batteries and a vibration motor encapsulated within the gun. 4* AAA batteries will be used to provide 5V voltage for the components. Refer to section 3.4 and 3.5 for detailed schematic and power calculation.

With reference to Figure 3.2, infrared transmitter will be used on the gun to simulate firing bullets. A switching mechanism will be placed at the trigger for detecting when the trigger is pressed. The vibration motor and buzzer will be placed at the back to give feedback to the player that a shot has been fired. Triggering of IR transmitter, motor and buzzer will be processed by Bluno Beetle on board. Signal will be transmitted to the laptop through bluetooth when the trigger is pressed for ammo calculation. IR transmitter will be coned up with aluminium foil to make it more directional. Shape of the cone may be modified and may consider encoding the IR signal to identify the source of the 'bullet' depending on the test result.

3.2.2 Vest

Vest will be attached with an IR receiver and vibration motor at the front for receiving the IR signal sent from the opponent gun and notify the player that he has been shot. IR receiver should be wrapped with an outward going cone shaped aluminium foil to increase the reception from the front direction. A 4*AAA battery pack will be used to provide 5v Voltage for the components. Data will be transmitted to the laptop when IR signal is received for health/shield calculation. Additional IR receiver may be used depending on the testing results.

3.2.3 Glove

A single AAA battery with a step up converter will be used to provide power for the component on the glove to ensure that it is lightweight. Sensors do not consume as much power as other components as well. MPU-6050 Inertial Measurement Unit will be used to measure the 3 axis acceleration and orientation. These data will be transmitted to the computer and then to the Ultra 96. Ultra 96 will process the data with a machine learning algorithm to deduce with a hand gesture the player is making.

3.3 Pin tables

<u>Shooting Apparatus</u>

Pins on Bluno Beetle	Sensors/Output
Vin	Battery pack positive
5V	IR transmitter VCC, step down voltage regulator-> motor driver vcc, button switch vcc
GND	IR transmitter gnd, motor driver gnd, battery pack negative, button switch gnd, voltage regulator gnd.
D2	IR transmitter sig
D3	Motor driver AIN1

Vest

Pins on Bluno Beetle	Sensors/Output
Vin	Battery pack positive
5V	IR receiver VCC, step down voltage regulator-> motor driver vcc
GND	IR transmitter gnd, motor driver gnd, battery pack negative, button switch gnd, voltage regulator gnd
D2	IR receiver out
D3	Motor driver AIN1

Glove

Pins on Bluno Beetle	Sensors/Output
Vin	Step up voltage regulator out
5V	IMU VCC
GND	Battery negative, IMU GND
SCL	IMU SCL
SDA	IMU SDA

3.4 Schematic

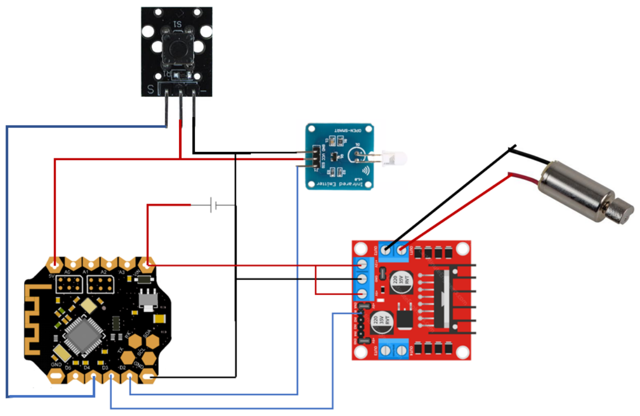


Figure 3.4a: Schematic for shooting apparatus

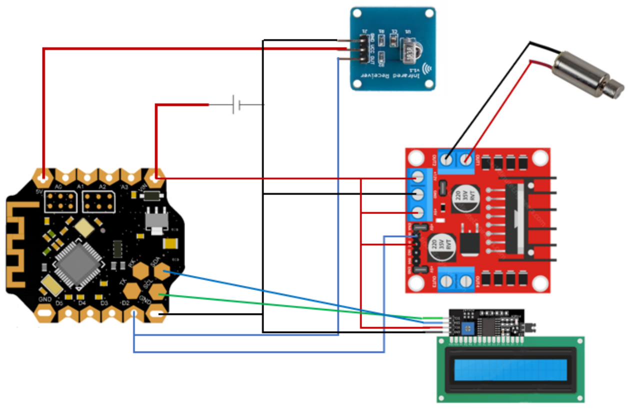


Figure 3.4b: Schematic for vest

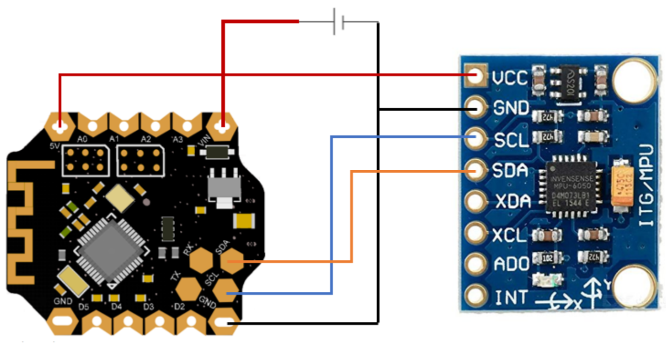


Figure 3.4c: Schematic for MPU

3.5 Operating Voltage

This section lists out the operating voltage and current drawn by each component and discusses the relevant design to take care of these requirements.

Component	Operating voltage(V)	Operating current(mA)	Power (mW)	Reference
Bluno Beetle	5.0	10	50	https://www.dfrobot.com/product-1259.html
MPU-6050-IMU	2.375-3.46	4.1	13	https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf
Mini 38KHz IR Infrared Transmitter Module	5.0	1.3	6.5	https://www.arduino-tech.com/mini-38khz-ir-infrared-transmitter-module-ir-infrared-receiver-sensor-module-for-arduino-rpi-stm32/
IR Infrared Receiver Sensor Module	5.0	1.2	6.0	
Vibration Motor	3.0	100	300	https://www.nfpmotor.com/products-micro-dc-motors-encapsulated-vibration-motor-NFP-E0724.html
Motor Driver Logic consumption	5.0	36	180	https://components101.com/modules/l293n-motor-driver-module
LCD display	5.0	1.5	7.5	https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf

Wearable power consumption = 50 + 13 = 63mW

Vest power consumption = 50 + 6.0 + 100 + 180 + 7.5 = 343.5mW

Shooting apparatus power consumption = 50 + 6.5 + 100 + 180 = 336.5mW

While in theory for a battery of 1.2V, 2000mAh, it takes 1.2 * 2000 / 336.5 ≈ 7h to fully discharge the battery, with reference to figure 3.5 knee of discharge tend to be before the 100% discharge capacity based on the discharge rate. With a step up transformer the discharge rate would be 4 times higher than normal and hence it would not last as long as the calculation in theory.

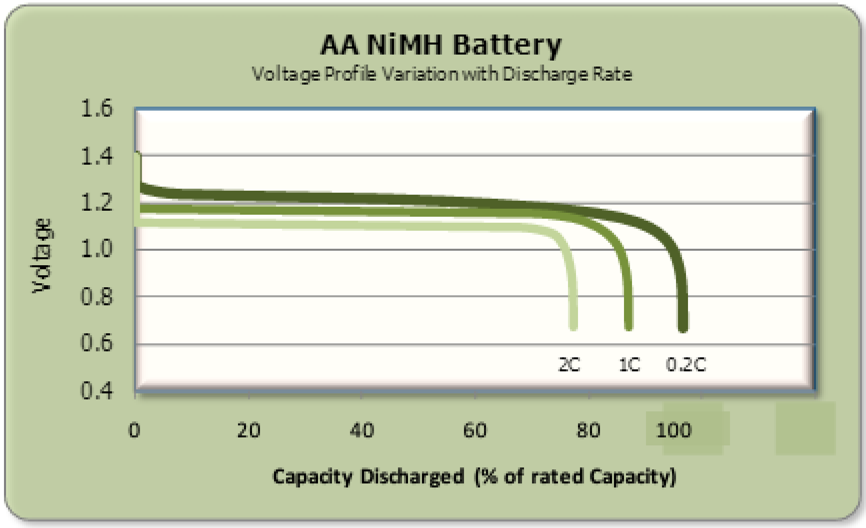


Figure 3.5: Discharge graph for Energizer NiMH Battery [0]

As seen from the power calculation, the glove will consume much less power compared to the vest and gun. Gloves need to be worn on hand as well hence it should be lightweight. Therefore we decided to use a single AA battery with a step up voltage regulator. While for vest and shooting apparatus 4 * AA battery is used to provide 5V directly to the beetle and the motors. This is because of the motor's higher power consumption. It is acceptable to have a heavier weight for the gun and vest to simulate the actual weight of a gun and load bearing vest.

With consideration of cost and reusability, we will use rechargeable NiMH batteries as it is easy to set up with a battery case and also can be reused over the project testing period. A typical AA battery has a capacity of 2000mAh while AAA battery has a capacity of 800mAh. We decided to choose the AA battery as the small increase in size and weight increase the durability by 2.5 times. This allows a longer game time without recharging with a small compromise. Power bank was not chosen as it typically requires a minimum current to be drawn to continuously operate which will not be achieved by the beetle sometimes.

Section 4: Hardware AI (Haoyu)

4.1 Data segmentation

In order to detect the activities of the user, we need to firstly segment the sensed data correctly. To do so, we need to identify the processing window and activity window. The processing window is the range of sensed data the program will process, and the activity window is the interval in time between the start and the end of a specific activity.

Processing window: The length of the processing window should be long enough to encompass all three actions. There are three actions: reload, grenade and shield. Since all three actions take around 1.5 seconds, and in case some users may be slower when executing these actions, 2 seconds may be a safe duration for the processing window. This means that the program will keep sensor data in the last 2 seconds to analyse any hidden pattern from them.

Activity window: All of the three actions have a common movement: the left hand will suddenly move up. Hence, we plan to look for this gesture. Whenever the sensors on the left hand detect an increase in the acceleration in any direction, the program will interpret that a potential action has been initiated, and it will start to look for patterns in the currently sensed data to recognise the action. For simpler implementation, we will not detect the end of the action but assume that each meaningful action will take 2 seconds.

4.2 Feature extraction

After segmenting the data, we can then proceed to explore features hidden in the data collected in the activity window (2 seconds). We are planning to explore six features: three features in the time domain (mean, variance and interquartile range) and three features in the frequency domain (mean, energy and interquartile range). These six features will be calculated in Ultra96 for each x, y, z acceleration and angular velocity, hence there are in total 36 feature values.

The raw data collected will be a 20 * 6 * 1 tensor, consisting of 20 samples of (x_acc, y_acc, z_acc, x_angular, y_angular, z_angular) data. The sampling frequency is set as 10 Hz tentatively, hence there are 20 samples in 2 seconds. The features will be calculated in the following ways, using the example of x_acc.

Time-domain features:

Mean: $\overline{a_x}$ is the average of a_{x_i}. $\overline{a_x} = \frac{1}{20}\sum_{i=1}^{20} a_{x_i}$.

Variance: $Var(a_x) = \frac{1}{20}\sum_{i=1}^{20}(a_{x_i} - \overline{a_x})^2$

Interquartile range: let $a_{x_{nth\ largest}}$ be the nth largest value. The interquatile range is equal to $\frac{a_{x_{5th\ largest}} + a_{x_{6th\ largest}}}{2} - \frac{a_{x_{15th\ largest}} + a_{x_{16th\ largest}}}{2}$

Frequency-domain features (need to convert to frequency domain first using FFT to obtain a list of magnitude: A_{x_1} … $A_{x_{20}}$):

Mean: $\overline{A_x} = \frac{1}{20}\sum_{i=1}^{20} A_{x_i}$

Energy: $E = \sum_{i=1}^{20} \left|A_{x_i}\right|^2$

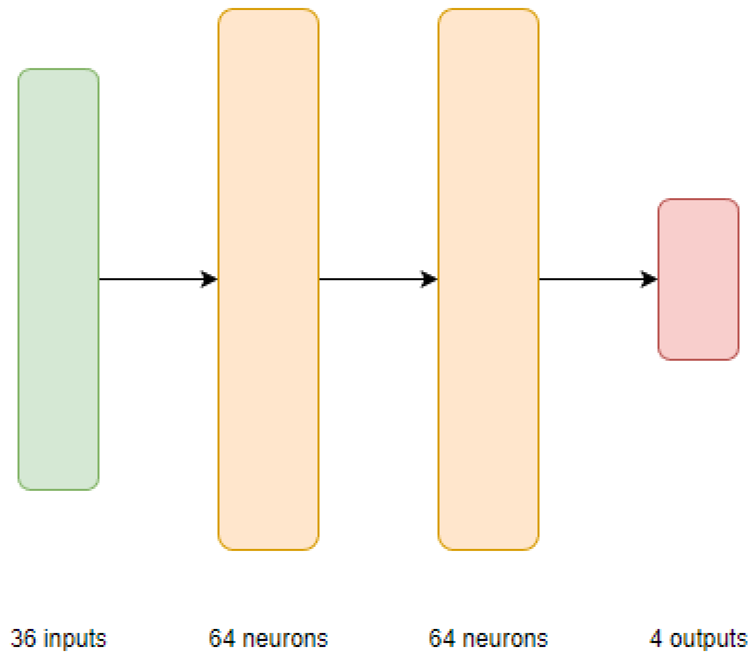
Interquartile range: let $A_{x_{nth\ largest}}$ be the nth largest frequency. The interquatile range is equal to $\frac{A_{x_{5th\ largest}} + A_{6th\ largest}}{2} - \frac{A_{x_{15th\ largest}} + A_{x_{16th\ largest}}}{2}$

They will then be packed into a 36*1 feature vector to be processed by the machine learning model.

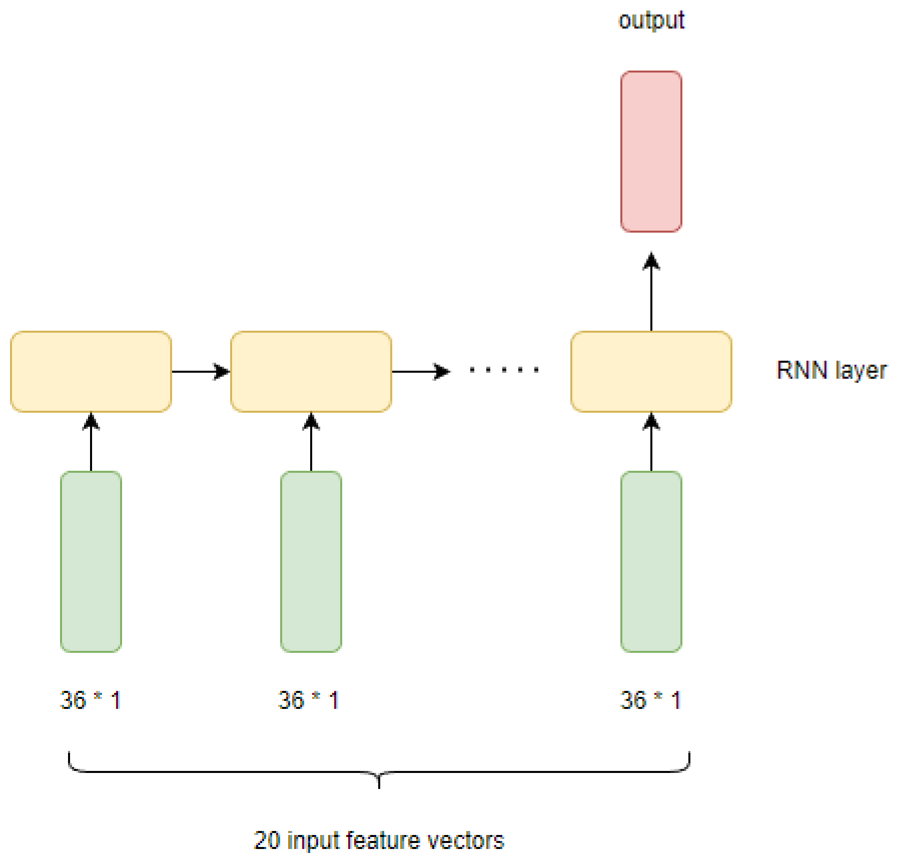
4.3 Machine learning models

Two types of machine learning models will be explored:

Multilayer Perceptron (MLP) neural network: We will firstly try MLP with ReLU activation. The input data will be a 36*1 feature vector. Then it will be passed to two fully connected layers (with ReLU activation) and a softmax layer to generate a 4*1 label vector as the output. The structure is shown below.



Recurrent neural network (RNN): Since the data is a time series, it is possible to use RNN to process the series of raw data collected and do the classification task. In this way, feature extraction is not necessary. We will try to a simple many-to-one RNN model [1]. The structure may be further modified to improve performance.



4.4 Model training and testing

After the model is constructed, we will use data to train the model. Firstly, a dataset will be built by collecting accelerometer and gyroscope feature vectors that correspond to different activities. This should be done after the hardware sensors are integrated into the wearable devices. Different people will wear the sensors and conduct three actions with their motions measured. The raw data will be pre-processed by normalisation and feature extraction, and the dataset consists of a range of feature vectors (each contain 6 features mentioned previously in section 2) and their corresponding activity labels. We will try to collect and label 1000 samples. After the dataset is built, it will be split into training dataset (90%) and validation dataset (10%). The training dataset will be used to train the model. The model will be trained for 100 epochs with batch size of 32 using adam optimizer and check the validation loss and accuracy along the way to check if the model has been properly trained. If the accuracy is not good enough, we will change the hyperparameters and train again.

After the model is trained, its performance will be tested. Different from the training dataset and validation dataset that come from the same pool of data, we plan to build the test dataset by collecting data from another new group of people. If the trained model can perform well on this brand new dataset, it should be able to generalise well on unseen data. [2] We will use the test accuracy to compare different models that we have implemented and pick the model that has the best performance.

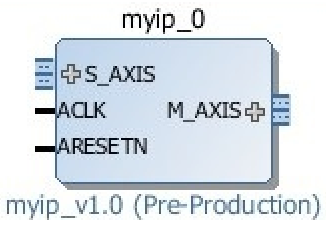
4.5 Implementation of neural network model on FPGA

After we have implemented the neural network on the software level and trained it to get the weights that can be used for inference, we will realise the neural network model on the FPGA for hardware acceleration.

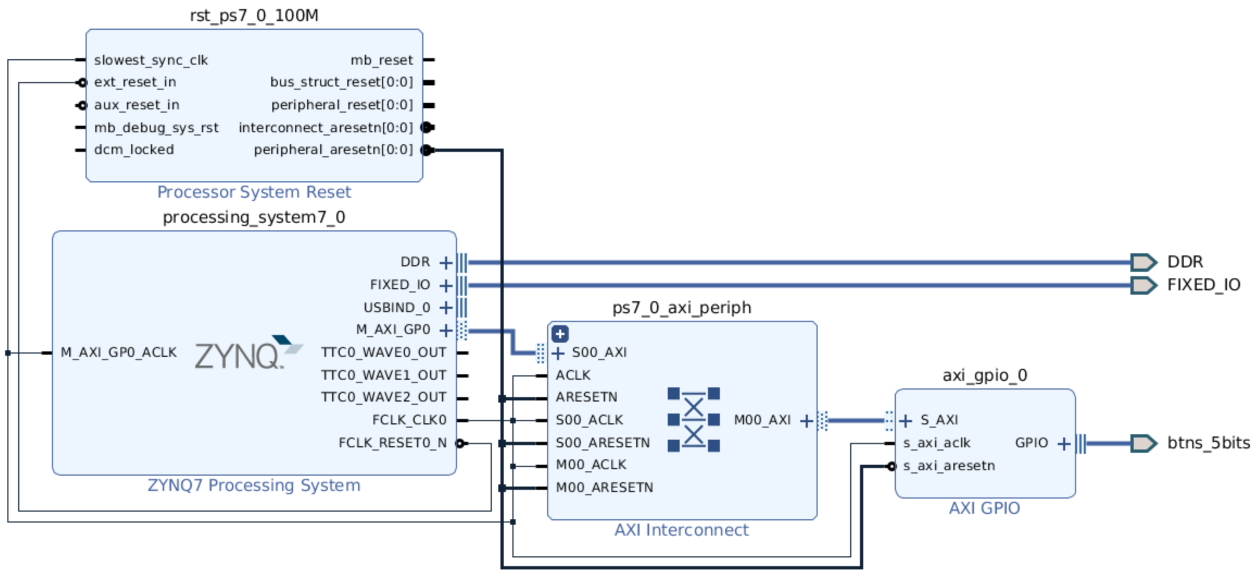
Two approaches are available for synthesising the model on FPGA. [3] The first one is High Level Synthesis (HLS), which is to use a higher level language (such as Python or C++) to describe the neural network structure. The second one is to synthesise in Register-Transfer Level (RTL), which is to use Hardware Description Language (HDL) to implement the neural network. The strength of the HLS approach is that it is more high-level and easier to implement, whereas the strength of the RTL approach is that it enables the developer to make design decisions at a lower level to increase the performance and efficiency of the embedded system. We plan to choose the HLS approach to implement the model quickly first, and if the efficiency or performance is not ideal, we will rewrite the model using HDL and make adjustments in lower level for better efficiency. In order to use HLS, we have installed Vitis HLS. (we also plan to explore other HLS tools such as Hls4ml and FINN.)

After the synthesis of the model, simulation will be required to test if the model is correctly constructed. The simulation will be done in either Vivado HLS or Vitis HLS. Specific test inputs will be supplied to the model and observe the simulated output and debug accordingly.

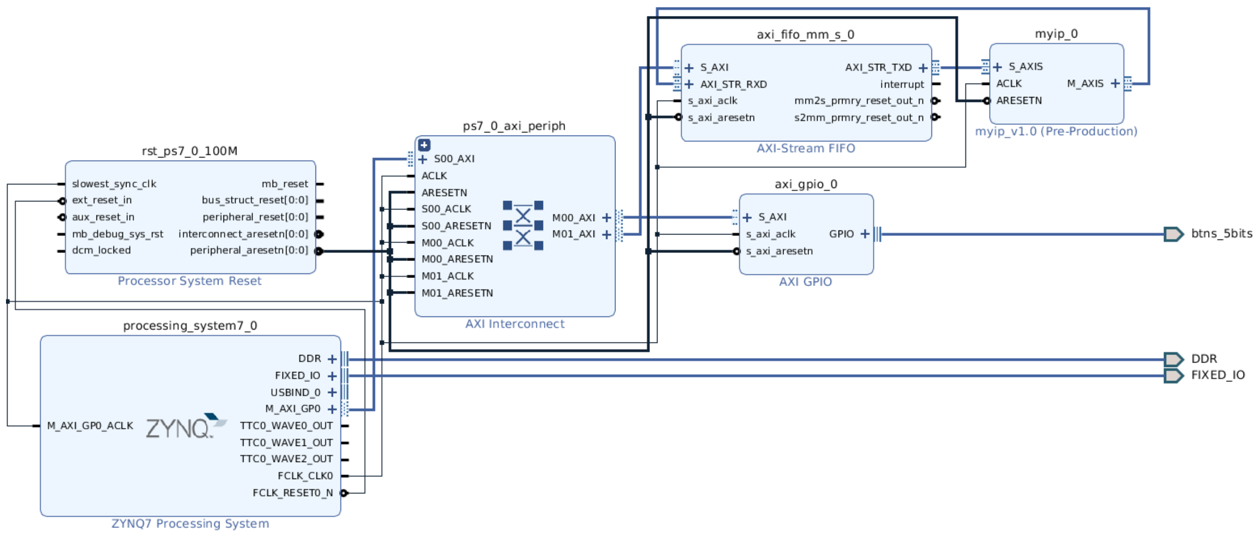
With the necessary tools and setups specified, we can then implement the neural network model on FPGA. Once the neural network is trained, the parameters (weights and biases) of each layer in the neural network will be extracted. With these parameters, the layers of the neural network can be written using HLS code. Then, each layer will be exported as IPs which are then stitched together to build the neural network. Once the IPs are built, it will look like the diagram below.



Additionally, the schematic for Ultra96 processor is shown below.



Then, to integrate the IPs into Ultra96, AXI Streaming FIFO IP will be used to bridge AXI Stream (point-to-point link) and AXI interface (interface with the main processor). The integrated hardware system schematic is displayed below.



The synthesised model will then be tested using simulation. Finally, the bitstream of the model will be generated and loaded onto the FPGA of Ultra96. With the PYNQ framework pre-installed in Ultra96, specifically the PYNQ Overlays (hardware libraries) [4], we can use Python scripts to utilise the neural network model on FGPA for inference [5].

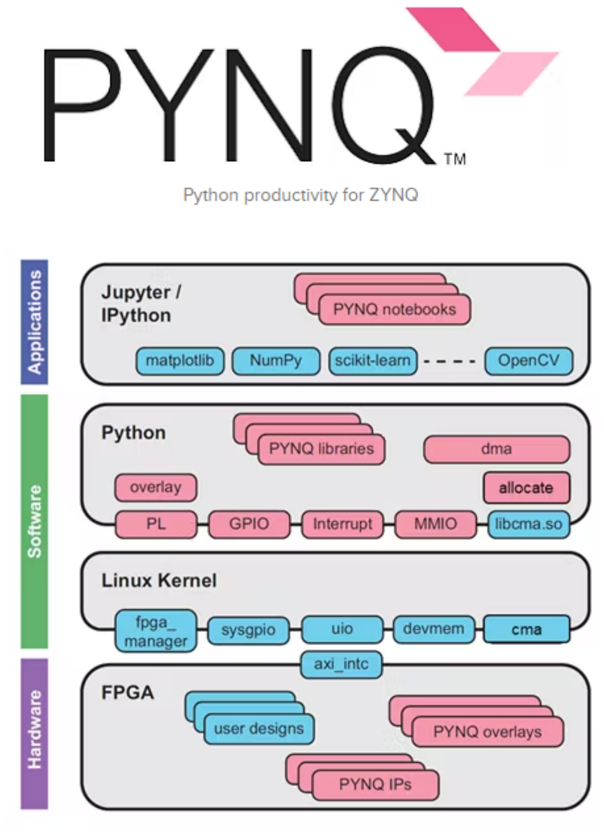


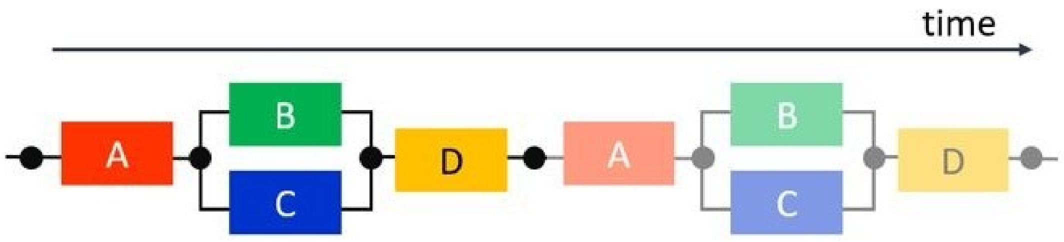
Figure 4.5 Overview of PYNQ framework

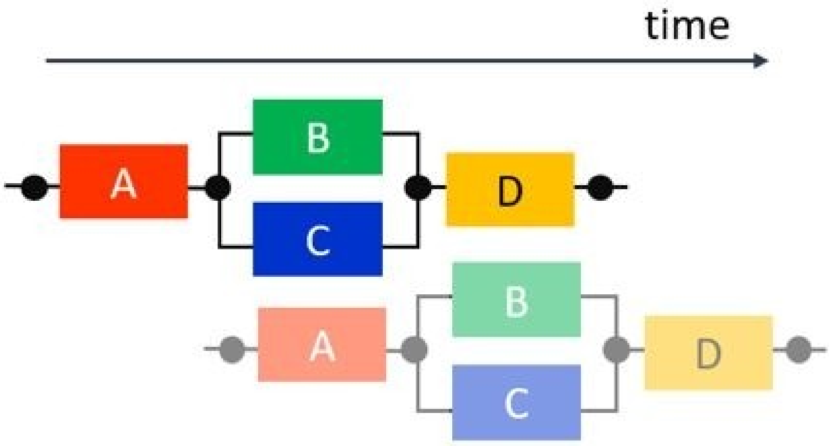
4.6 Performance evaluation of the neural network on FPGA

After implementing the neural network on FPGA, we will evaluate its performance by its latency, power and area. The design's latency, power and area will be shown after synthesisation. Ideally, the latency and power should be as low as possible to ensure fast inference and high efficiency. There is a specific requirement for area, since we are accessing the Ultra96 remotely. However, the larger the area, the higher the power used as more transistors are used. Hence, when implementing the neural network model, the area should be minimised if possible. The design will be further optimised for lower latency and lower power consumption.

After the bitstream of the design is uploaded to the Ultra96 and the model starts to work, the actual latency can be measured by recording the time the feature vectors are fed into the model and the time the results come out. Additionally, the PMBus tool [6] can be used to measure the actual power consumption of the Ultra96. Further modifications on the model and Ultra96 settings can be made to reduce the actual latency and power consumption.

We will attempt to optimise the digital design at HLS level. Two possible angles of optimisation are parallelism and pipelining (shown below). They can be implemented using HLS dataflow pragma.





With parallelism and piplining, latency can be reduced, and possibly reduce the resources used because pipelining facilitates the reuse of computational resources. However, the resources used may also increase after optimisation because of additional resources needed for parallelism. We will check the Memory section of the synthesis report to monitor the use of resources and determine the optimal setting that balance the latency, power and use of resources.

Section 5 Internal Communications (Nishant)

5.1 Task management on Beetles

The Beetles will always carry out their tasks sequentially. First they will read sensor data, then they will pre-process it, and finally send it to the laptop. This can be implemented using a simple interrupt that triggers at a set frequency, which will start the aforementioned sequence.

5.2 Implementation on Laptops

On the laptops, the task of receiving packets from the Beetles is IO bound (i.e. bound by the time it takes for packets to arrive over BLE) rather than CPU bound, thus multithreading will be used. As such, a different thread will be used to receive packets from each Beetle to ensure that whenever a packet is sent to a laptop by a Beetle, the laptop will switch to the thread assigned to that Beetle and process the packet received as soon as possible. This can be implemented using the threading module in Python.

5.3 Setup/ Configuration of BLE

The setup and configuration of BLE interfaces will be done using the following commands on Ubuntu Linux (retrieved from ATWILC3000 User Guide [7]):

1. echo BT_POWER_UP > /dev/wilc_bt

   This command powers up the ATWILC3000 chip and indicates that the BT requires the chip to be on.

2. echo BT_DOWNLOAD_FW > /dev/wilc_bt

   This will download the BT firmware using SDIO.

3. echo BT_FW_CHIP_WAKEUP > /dev/wilc_bt

   This will prevent the chip from sleeping.

4. hciattach /dev/ttyPS1 -t 10 any 115200 noflow nosleep

   This attaches serial UART to the bluetooth stack and sets the baud rate of the bluetooth firmware to 115200 which is required according to the documentation.

5.4 Internal Communication Protocol

Upon bootup, a 3-way handshake will be executed between each Beetle and the laptop as shown below.

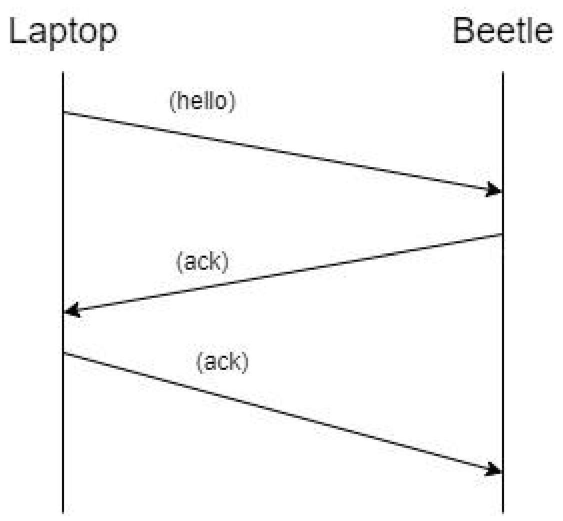


Figure 5.4: 3-Way Handshake

The laptop initiates the handshake by sending a HELLO packet to each Beetle. Upon receiving this packet, each Beetle sends an ACK packet back to the laptop to acknowledge reception of the HELLO packet. The laptop then also sends an ACK packet back to each Beetle that it received the ACK packet from.

Packet Type	**Packet Code**
ACK	A
Hello	H
Data	D

Data Packet Format			
Packet Seq. No. (0 or 1)	Packet Code (char)	Accelerometer Data (6 floats)	Checksum (4 chars)
30 bytes			

Above are the packet details for our communication protocol. Similar to rdt (reliable data transfer) 3.0, ACK packets are used with sequence numbers in place of NAK packets, where consecutive ACK packets with the same sequence number means the second ACK packet is equivalent to a NAK. The baud rate has been chosen to be 115200 bps.

5.4.1 Reliability

To ensure reliability in communication over BLE, the protocol uses many features found in rdt 3.0.

If a packet received by the laptop is corrupted, it will send a duplicate ACK packet (with the same sequence number as the previous ACK packet) to the Beetle which will indicate that retransmission of the packet is required.

There will also be a timeout implemented on the Beetle to handle packet losses or connection drops. If a certain period of time has passed since data transmission from the Beetle and there is no ACK package from the laptop, the Beetle will automatically retransmit the packet. As such, if the packet was indeed lost then the retransmission will recover it, otherwise if the data packet had already been received by the laptop and there was either a delay in the laptop sending the ACK packet or it was lost/ corrupted, then the laptop will discard it since its sequence number will be repeated.

Packet fragmentation can be handled by making the packet sizes known to the laptops/Beetles, so that they can ensure any packet received is complete by ensuring it is of the given size.

Section 6 External Communications (MingShun)

6.1 Communication Between Ultra96 and Evaluation Server

Communication between Ultra96 and the evaluation server would be established using the Transmission Control Protocol/ Internet Protocol (TCP/IP) communication. This could be achieved through the application of Socket programming.

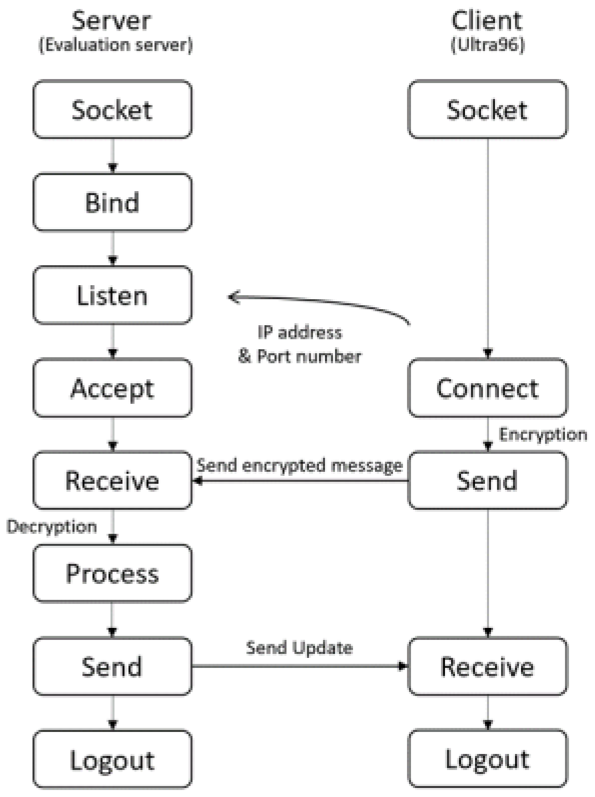


Figure 6.1 Communication between Server and Client

The above image details the process between the client (Ultra96) and the server (Evaluation Server). According to the eval_server.py, the server creates a listening socket when initialised. As such, the Ultra 96 will attempt to connect to the port/ IP address of the server's listening socket and only after the connection is accepted by the server, a communication channel would have then been established between the server and the client.

After the communication channel has been established, the Ultra96 will send an AES encrypted message to the evaluation server to be decrypted. As specified in the lecture slides, the message format sent to the server would be in the form of JSON, AES encryption expected from the client side will be of base 64 with padding.The message format would be in the form of tuples from both the players. The tuples of “P1_action, P2_action” should be the format as received by the server.

In terms of security and encryption, AES is implemented. AES, which stands for Advanced Encryption Standard, is a block cipher that has a key size that can be up to 256 bits long and encrypts data in blocks of 128 bits each. Simply put, it takes in an input of 128 bits and outputs 128 bits of encrypted cipher text. AES relies on substitution-permutation network principle which refers to a series of replacing and shuffling linked operations of the input data. AES can be implemented in the python library using the Cryptodome Cipher Library.

With regards to the encryption-decryption process, during encryption, a 16 bit variable will be generated that would act as the key or initialisation vector (IV). This randomly generated key ensures that each iteration of encryption is unique and independent of the inputs. Following that, the inputs will be converted into a byte string using an encode function and padded with a block size of 16 bytes. The key will encrypt the padded byte input before the resulting encrypted message would be sent to the evaluation channel through the connected socket channel.

6.2 Communication Between Ultra96 and Laptops

Communication between the Ultra96 and laptops makes use of the mentioned TCP/IP socket programming. The Ultra 96 would act as a server to the laptop clients to allow data to be transmitted after the initial creation and listening of socket as well as connection from the client to the Ultra96. The problem of tunnelling is understood as the Ultra96 being isolated within the SOC’s sunfire network and is only accessible through SSH tunnelling with port forwarding. The use of “sshtunnel” python library will mediate that problem and help make the connection between the Ultra96 and the laptops.

This could be done through setting up the local port forwarding from the laptop’s localhost to host 1, sunfire. Then, we can then tunnel from the latter to host 2, Ultra96 using the same port number. This process essentially sets up a tunnel from the laptop to sunfire, then from sunfire to Ultra96 through the first tunnel. Through this, communication between the Ultra96 and laptops would be possible.

6.3 Concurrency On Laptops/Ultra96

Concurrency is necessary since there would be multiple connections connecting to the same port of the Ultra96 server. If not managed, IO issues will appear between clients, which in this case refers to the laptops, resulting in communication being disrupted. Concurrency helps reduce these IO issues while helping to improve the performance of the system. The use of multithreading and asynchronous locks like mutex/semaphores will help with the control flow through a single port. Python libraries like “threading” and “asyncio” would be the target for concurrency.

In addition to concurrency in network channels, the concept of concurrency is required when running the game mechanism in the Ultra96. As there would be multiple processes occurring simultaneously be it updating game state or players’ status, concurrency to reduce IO problems is an essential implementation. Priority threads are introduced such that the game states will update sequentially and the game can run smoothly.

6.4 Communication between Ultra96 and Visualizer

With regards to the communication between Ultra96 and Visualizer, we are exploring the option of using MQTT (MQ Telemetry Transport) protocol. It is a lightweight open messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information in low-bandwidth environments. Instead of the traditional client/server relationship, MQTT protocol can be better described as a client/broker relationship. The broker handles authentication, managing connections as well as data flows on the network. Its main job is to receive all published messages and then send them to subscribed clients. With respect to the specification of the communications between the Ultra96 and the Visualizer which do not require us to store data, this protocol seems like an appropriate choice, which leads us to suggest the use of RabbitMQ broker for data communication between the Ultra96 and Visualizer.

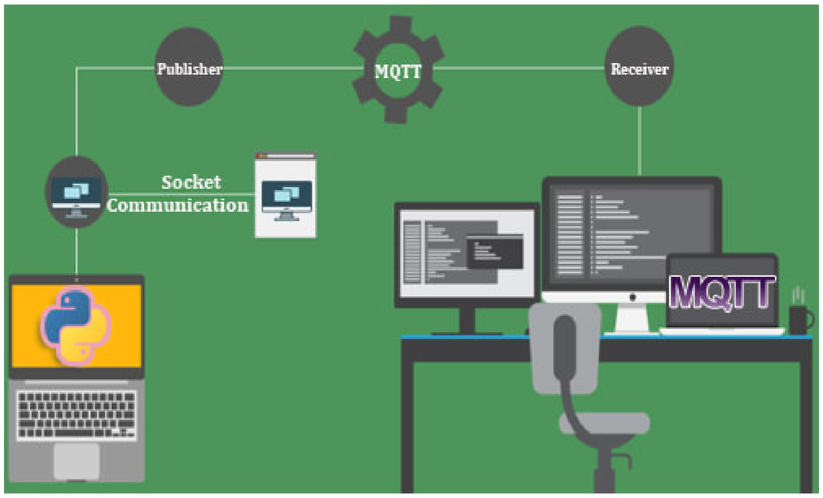


Figure 6.4 Visualisation of MQTT protocol [8]

The Ultra96 will act as a publisher and send filled data packets to the RabbitMQ broker which would be stored in a dictionary with a range of key value pairs. RabbitMQ will then accept the data packets and route it to the appropriate message queues for the respective software visualizer, which will act as the receiver in the above image and will retrieve these messages from the queue to display them on the screen.

Section 7 Software Visualizer (Yuhang)

7.1 User Survey

In the user survey, we surveyed potential players of the laser tag game on the information which they prefer to be shown on the screen, given the design of the game. Overall it was commonly agreed that information about the player, such as number of lives, ammo, health, grenade count and shield count should always be displayed for players to conveniently track the progress of the game. The surveyees also indicated that it would be helpful to have visual and audio feedback for actions such as shield activation, grenade explosion, firing of the weapon and being hit.

7.2 Visualizer Design

As such, the software visualiser will display a HUD for player information – current health, ammo remaining in current magazine, number of grenades/shields remaining, and number of lives remaining. Additionally, during a shield activation, there will be a status indicator for the health and duration of shield remaining, as well as a visual effect (e.g. a circular orange tint) and audio effect to indicate the activation of a shield. After the shield expires, there will be a shield cooldown counter displayed. If the player attempts to activate the shield during this cooldown period, there will be an audio effect to indicate that the shield is not available yet.

Not only so, there will be a short section of gun barrel rendered on the screen (assuming the phone camera doesn't include the player's gun in the viewfinder), with corresponding particle effects when firing the gun and when the player scores a hit on the enemy. Similarly, there will also be a grenade model created on grenade throw, with corresponding particle effects for its explosion. Lastly, we plan to overlay some models on the enemy player for increased realism. For instance, rendering a helmet or even body armour on the enemy player, as well as rendering a more futuristic/sci-fi gun on the enemy's weapon.

The phone will be mounted on the gun. Naturally, given the small size of the phone screen, ensuring that all information is clearly visible from a reasonable distance without cluttering the screen is a design constraint that will have to undergo user testing. We are also considering using tablets for a larger display and more immersive gameplay. We plan on using Unity and the AR Foundation framework for cross-platform AR design.

7.3 Communication With Ultra96

The phone will communicate with the Ultra96, using the RabbitMQ broker over MQTT. Relevant data includes player actions (shield, grenade, reload, shoot), hitting an enemy, and being hit by enemy gunfire or grenade.

7.4 Phone Sensors

We intend to use the phone’s vibration motor to simulate recoil when the player shoots the gun, as well as the phone’s speakers to playback audio for the various actions – for instance playing a sound effect to indicate the player being hit, a shield sound effect on shield activation/deactivation etc. That said, the latency incurred from the bluetooth/wifi connection may impact the gameplay experience, and is something that has to be user tested.

7.5 AR Effects

AR effects on the enemy will rely on player detection via a marker on the enemy as well as object detection using CNN (e.g. identify a person’s head in order to render a helmet over it, or to display particle effects when the enemy fires their weapon or gets hit by a grenade).

Section 8 Project Management Plan (Joint Work)

Project Week	Overall Progress	Hardware Sensors	Hardware AI	Internal Communications	External Communications	Software Visualizer
3	Ready to begin prototyping	Research on suitable components	Research on: data segmentation and features to be explored; machine learning models to be used; model training; FPGA-related set-up and analysis	Able to send dummy data from Beetle to laptop over BLE	Research on hardware components as well as libraries for network communication protocol	Explore AR frameworks and relevant toolkits. Design a basic AR application (without HUD for now)
4	Basic prototype working	Purchase component needed	Train machine learning models using sample datasets; explore and implement feature extraction	Achieve stable connection between one Beetle and laptop for at least 60 seconds	Basic implementation of network commands and testing it	Design software HUD for player information
5	Full subsystems (close to) working	Test out various sensor and build basic prototype, IMU working	Implement the model on FPGA using HLS; test using simulation; upload bitstream to ultra96 and run the model	Achieve stable connection between three Beetle and laptop for at least 60 seconds using multithreading (for eval)	Have a functioning communication channel between beetles and laptops	Add assets for shield/grenade, particle effects for firing weapon and being hit. Conduct user testing.
6	Fully working subsystems	Complete gun and glove for player 1	Bug fixing for FPGA; start data collection (using hardware sensors)	Bug fixing and improving code for evaluation and to prepare for integration	Settling bugs and research and implementation of multithreaded environment	Polishing the AR visualiser, fixing bugs
Recess week	Subsystem integration	Test and debugging	Train the model using real data; implement the model using HLS and upload the generated model to ultra96	Begin integration with external communications: ensure packets received from Beetle are suitable for Ultra96 to receive	Implemented multithreaded environment and debugging	Integrations with external communications. Gather feedback from team members and improve UI
7		Pair with internal communication	Test the performance and improve the model	Fully integrated with external communication	Working on improving performance	Fully integrated with external communication. Explore additional

						AR effects/models for immersive gameplay
8		Demo and debug	Demo and debug	Demo with actual sensor placement on players and debugging	First Demo and debugging	Demo and debug for 1-single player
9	Full system integrated and working for one player	Set up for player 2	Optimise the neural network training and implementation on FPGA	Fully integrated system, debugging prepare for 2 players	Integration of system with others and debugging	Debugging. Polish UI and add extra AR effects/models
10	Extend functionalities to 2 players	Test and debugging	Bug fixing	Bug fixing and performance improvement	Testing of integrated system and working on improvements of code, performance and calibration	
11	System working fully for 2 players					Debugging
12	System ready for final evaluation	Final report				Final report
13	Final Evaluation					

References

[0] Energizer Brands, LLC. (2018). *Nickel Metal Hydride (nimh) - energizer*. Nickel Metal Hydride (NiMH) Handbook and Application Manual. Retrieved August 31, 2022, from https://data.energizer.com/pdfs/nickelmetalhydride_appman.pdf

[1] S. Solanki, "PyTorch RNNs For Text Classification Tasks by Sunny Solanki," *Developed for Developers by Developer for the betterment of Development*, 05 Apr. 2022, https://www.coderzcolumn.com/tutorials/artificial-intelligence/pytorch-rnn-for-text-classification-tasks.

[2] Brownlee, Jason. "What Is the Difference Between Test and Validation Datasets?" Machine Learning Mastery, 14 Aug. 2020, https://www.machinelearningmastery.com/difference-test-validation-datasets/.

[3] Gurel, Muhsin. "A Comparative Study between RTL and HLS for Image Processing Applications with FPGAs." EScholarship, University of California, 17 Dec. 2016, https://www.escholarship.org/uc/item/9vx1s37b.

[4] "PYNQ Overlays." PYNQ Overlays - Python Productivity for Zynq (Pynq) v1.0, https://www.pynq.readthedocs.io/en/v2.3/pynq_overlays.html.

[5] Wadulisi. "Easy AI with Python and PYNQ." Hackster.io, 6 May 2021, https://www.hackster.io/wadulisi/easy-ai-with-python-and-pynq-dd4822#toc-step-3---install-the-python-dpu-package-3.

[6] "What is PMBus?," Total Phase Blog, 07 May 2021. [Online]. Available: https://www.totalphase.com/blog/2021/02/what-is-pmbus/.

[7] *Wi-Fi Link Controller Linux Guide.* [Online]. pp28, 29. Available: https://www.microchip.com/en-us/product/ATWILC3000 .

[8] *"How to Connect Socket Communication Using MQTT in Python?" NEX Softsys,* https://www.nexsoftsys.com/articles/socket-communication-using-mqtt-in-Python.html.