CG4002 Embedded System Design Project
January 2022 semester

"Shoot Shoot"
Design Report

Group **15**	Name	Student #	Specific Contribution
Member #1	Lu Ziyi	A0197282U	Hw Sensors
Member #2	Nigel Tan	A0183490B	Hw AI
Member #3	Chiam Jia-En	A0206297Y	Comms Internal
Member #4	Tan Zheng Chong, Shawn	A0199891A	Comms External
Member #5	Vallamkonda Nagamani Teja	A0202093U	Sw Visualizer

Table of Contents

Section 1 System Functionalities

1.1 Use case diagrams

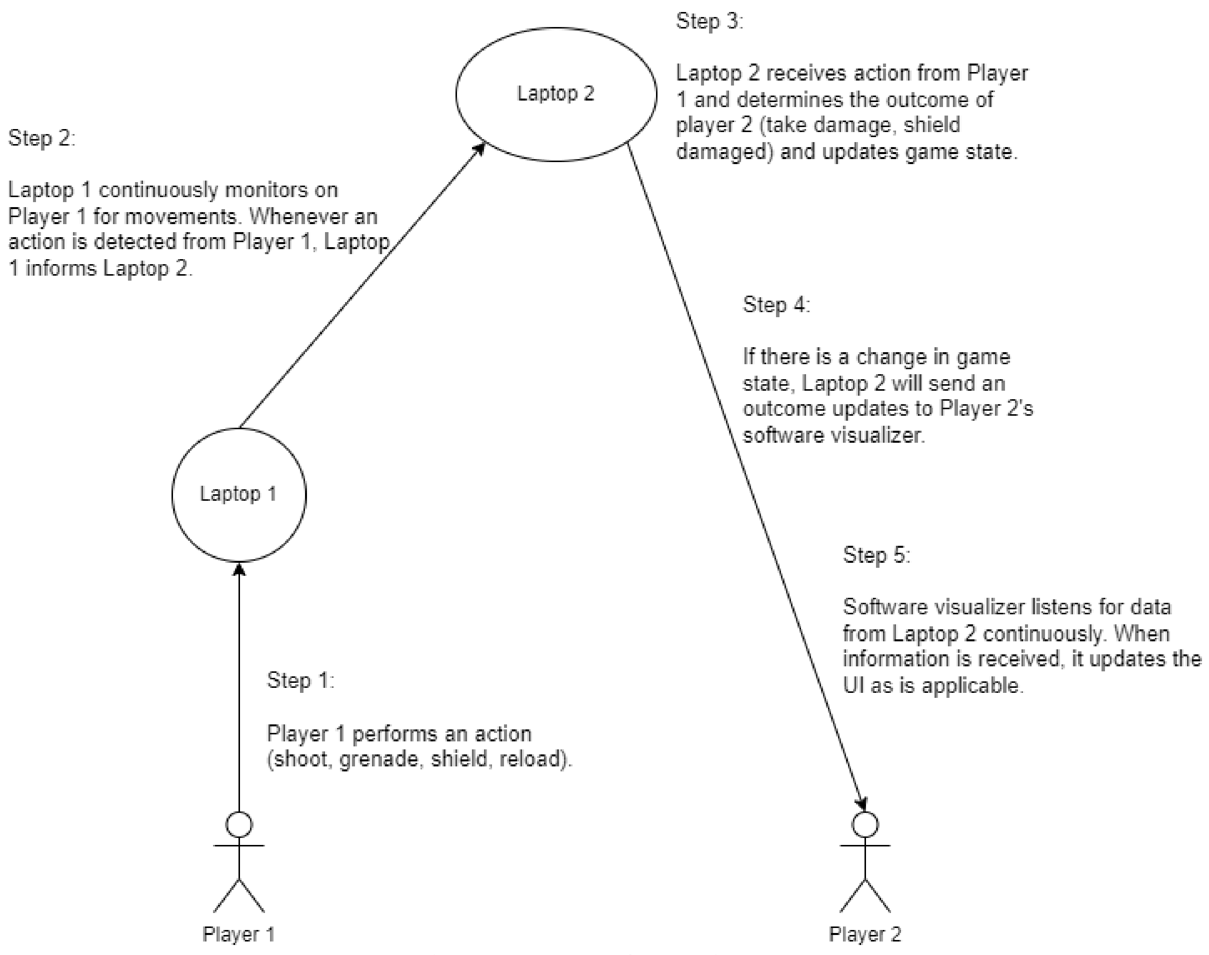

Step 3:
Laptop 2 receives action from Player 1 and determines the outcome of player 2 (take damage, shield damaged) and updates game state.
Laptop 2
Step 2:
Laptop 1 continuously monitors on Player 1 for movements. Whenever an action is detected from Player 1, Laptop 1 informs Laptop 2.
Step 4:
If there is a change in game state, Laptop 2 will send an outcome updates to Player 2's software visualizer.
Laptop 1
Step 5:
Software visualizer listens for data from Laptop 2 continuously. When information is received, it updates the UI as is applicable.
Step 1:
Player 1 performs an action (shoot, grenade, shield, reload).
Player 1
Player 2

Figure 1.1: Use Case Diagram

For the laser tag game, there will be a laptop on-site (Laptop 1) that receives player data on the ground. This data will correspond to the action performed (shoot, grenade, shield, reload).

Laptop 1 is mainly responsible for routing information to a remote laptop (Laptop 2 / Ultra96) that is responsible for managing the overall game state, e.g. which player has how much hp left etc. Laptop 1 may perform auxiliary duties like data cleaning/ noise removal.

Laptop 2 is a remote server that continuously listens for player actions from on-site laptop(s) and is responsible for determining the outcomes/ changes to the game state (if any) for each of these actions.

Suppose Player 1 makes a shot, then Laptop 2 will have to decide whether Player 2 takes damage. This computation can be done as Laptop 2 has access to the overall game state and knows if Player 2 is shielded.

Whenever a game state changes, Laptop 2 will push out these updates directly to the software visualisers on the ground for them to (re-)render the UI.

Note: Player 1 and Player 2 should be interchangeable throughout the game.

1.2 User Stories

As a…	**I want to…**	**So that…**
Player	View my Health	I know how much health I have left and will act accordingly.
Player	View my Shield Status	I can activate it once the cool down is over or even know when all my shields are over.
Player	View my Ammo	I know when I need to reload in order to continue playing the game.
Player	Only attack my enemies	I avoid friendly fire and I won't waste time
Player	Throw the Grenade	I can cause more damage as compared to a normal bullet.
Safety Officer	Make sure that this game is safe to play anywhere	People playing this game won't be in trouble while playing this game.
Competitive Gamer	I want to see my score	I know how many kills I have made.
Competitive Gamer	I want to see how many enemies are remaining	I know how many more I need to kill
Action Movie Fanatic	See cool animations while performing some actions	I can feel that I am a part of a movie myself.

Player with Spectacles	Be able to play this game even with my spectacles on	I can clearly see the objects in front of me
Player that is environmentally conscious	Use a device that does not consume a lot of power	There is less damage to the environment.
Player	Be able to move freely while playing the game	Can truly enjoy the game
Army Regular	Be able to use this platform to practice my response time and hone my shooting skills	I can defend the nation better in Urban Warfare
Player	Be able to view my opponents HP and Ammo	I know when I should activate my shield and when I need to throw the grenade.
Gamer	Have a crosshair	I can aim and shoot properly.

1.3 Feature List

From the User Stories and use case diagrams. We have compiled the following feature list to encompass the various functionalities and plus points we intend to provide to our users.

- Fast and precise hit registration
- Lightweight, sturdy and aesthetically pleasing game hardware
- An integrated and immersive Augmented Reality (AR) game visualizer
- A user-friendly and intuitive Game User Interface (UI)
- Immediate and accurate detection of user actions e.g. grenade/shield
- Secure communication between all components of the game system
- Helpful postgame metrics and logs
- Long lasting and efficient power consumption
- Affordable

Section 2 Overall System Architecture

2.1 Complete High Level System Architecture Diagram

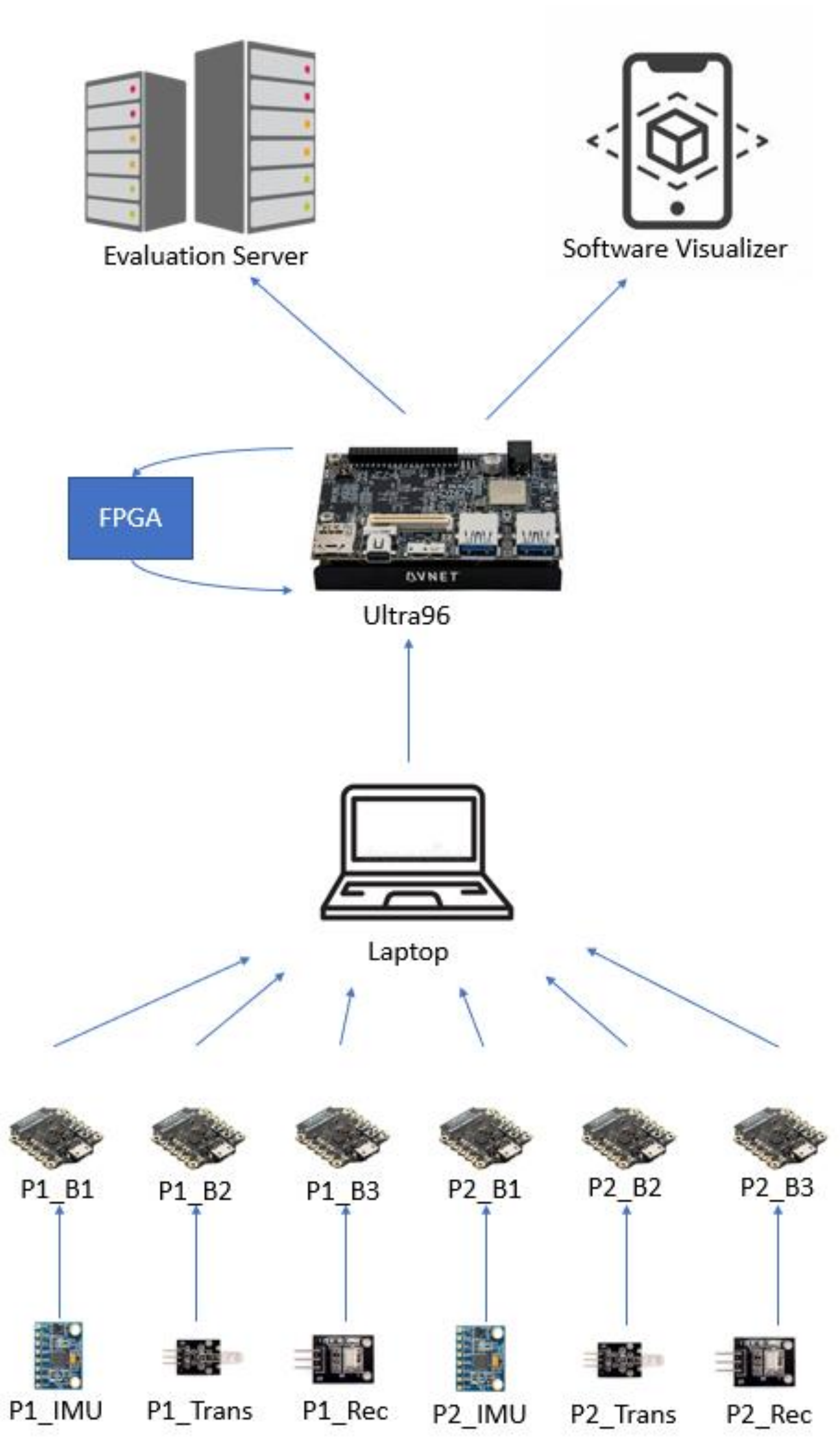


Figure 2.1: Overall network architecture

In our proposed system for the laser tag game, we have 5 separate components that will work together to provide the desired game experience for our users. The components are the Hardware Sensors, the Bluno Beetles, the Laptop, the FPGA and the Software Visualizer.

Here is a brief overview of the network and how its implementation will help to achieve our system.

Starting from the hardware sensors. They collect data from the real world that will be crucial for making predictions and changing the game state. Just like how humans are only able to tell what is going on with the world through their senses, these hardware sensors act as the eyes and ears of our system. More information can be found in: Section 3 Hardware Sensors.

Data collected at the hardware sensors is sent to the Bluno Beetles, which act as the first line of preprocessing. They help to collate the sensor data and relay it to the laptop. Details on how this is done can be seen in: Section 5 Internal Communications.

At the laptop, the data undergoes the next layer of preprocessing, this time simply to coordinate between the data retrieved from all the Bluno Beetles into a more coherent form. The coordinated raw data obtained in that time-frame is then sent to the Ultra96. How this data is being processed and sent can be found in: Section 6 External Communications.

The Ultra96 will be the main brain of the system, it is here that the raw data will finally be processed into a form that can be used to represent a game state. The Ultra96 also handles the entire game state management and judges changes to the game state based on the data it receives. The FPGA assists the Ultra96 in making these game state decisions through Machine Learning. By feeding raw data into the FPGA, it will make a prediction based on the data as to what is going on in the real world and translate them to functions that affect the game state. Details on how the FPGA will be used to make predictions can be found in: Section 4 Hardware AI,

Finally, the fully processed game state data at every point is sent to the Software Visualizer. The Software Visualizer makes use of this game state data to display to the user an augmented reality version of the game, where information from the game state is integrated into a real-life environment. The implementation and methodology we used for the UI can be found in: Section 7 Software Visualizer.

The fully processed game state data is also sent to an evaluation server to provide verification that what is happening in real life is accurately reflected by the system.

As can be seen in the network diagram, there are many links connecting the various components together. These connections will be discussed in the following sections, along with more details about what functions are at each stage.

2.1.1 Hardware Components

The Hardware Sensors selected have a specific role of collecting the raw data that define an "action" in the game. In a laser tag game, shooting the gun, getting hit and doing game specific actions like shielding are all game actions that need to be determined.

Hardware Sensor	Role it Performs
IMU (P1_IMU, P2_IMU)	Senses Movement Actions like grenade, shield, reload and logout.
IR Transmitter (P1_Trans, P2_Trans)	Transmits an IR signal and takes note of time of transmitting to mark a shot taken.
IR Receiver (P1_Rec, P2_Rec	Receives an IR signal and takes note of time to mark a shot hit.

Specific hardware information will be available later in section 3 (Hardware Sensors) and section 2.2 (Final Design of laser tag system), but these specific sensors provide the key information for deciding the game state.

2.1.2 Connections and Interfaces

The figure below provides a clear overall data flow between each hardware or software components (in white) as well as their respective interfaces (in blue).

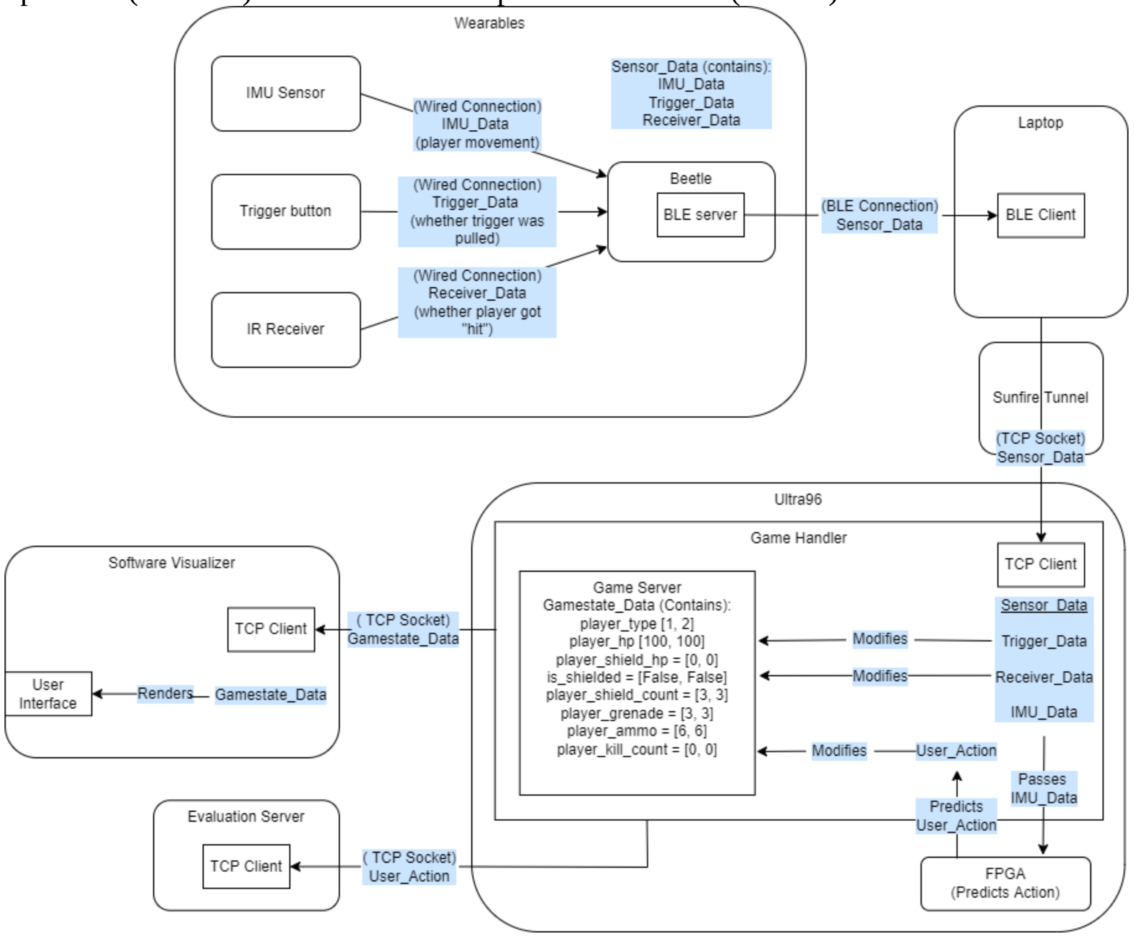


Figure 2.1.2: Overall data flow diagram

2.1.3 Implementation on the Beetle

These are the various tasks that will be handled by the beetle

Tasks	Description
ReadIMUData()	Reads and stores the IMU data from the IMU sensor
ReadTriggerData()	Reads and stores whether the trigger has been fired
ReadReceiver()	Reads and stores whether the receiver has received an IR signal
MakePacket()	Converts that data received from the sensors into a packet to send
GetCommand()	Retrieves the command sent from the laptop
ExecuteCommand()	Executes the command sent by the laptop (this can be to enter data sending state or sleep state)
SendPacket()	Sends Packet Data obtained from MakePacket() to the laptop.
initBeetle()	Initializes the configuration for BLE connection for the beetle.

2.1.4 Implementation on the Ultra96

These are the various tasks that will be performed on the Ultra96

Tasks	Description
update_game_state()	Main caller function in the computing the game state
got_shot()	Computes if hit is registered and updates player hp
make_shot() / reload()	Updates the player ammo
FPGA()	Computes the action done by the user
take_grenade_damage()	Computes if grenade hit is registered and updates player hp
activate_grenade()	Updates the player’s grenade count
send_gamestate()	Sends the game state data to the evaluation server and software visualiser
initialize_game()	Initializes the game state
end_game()	Ends the current session and clears up resources used

2.2 Final Form of laser tag system

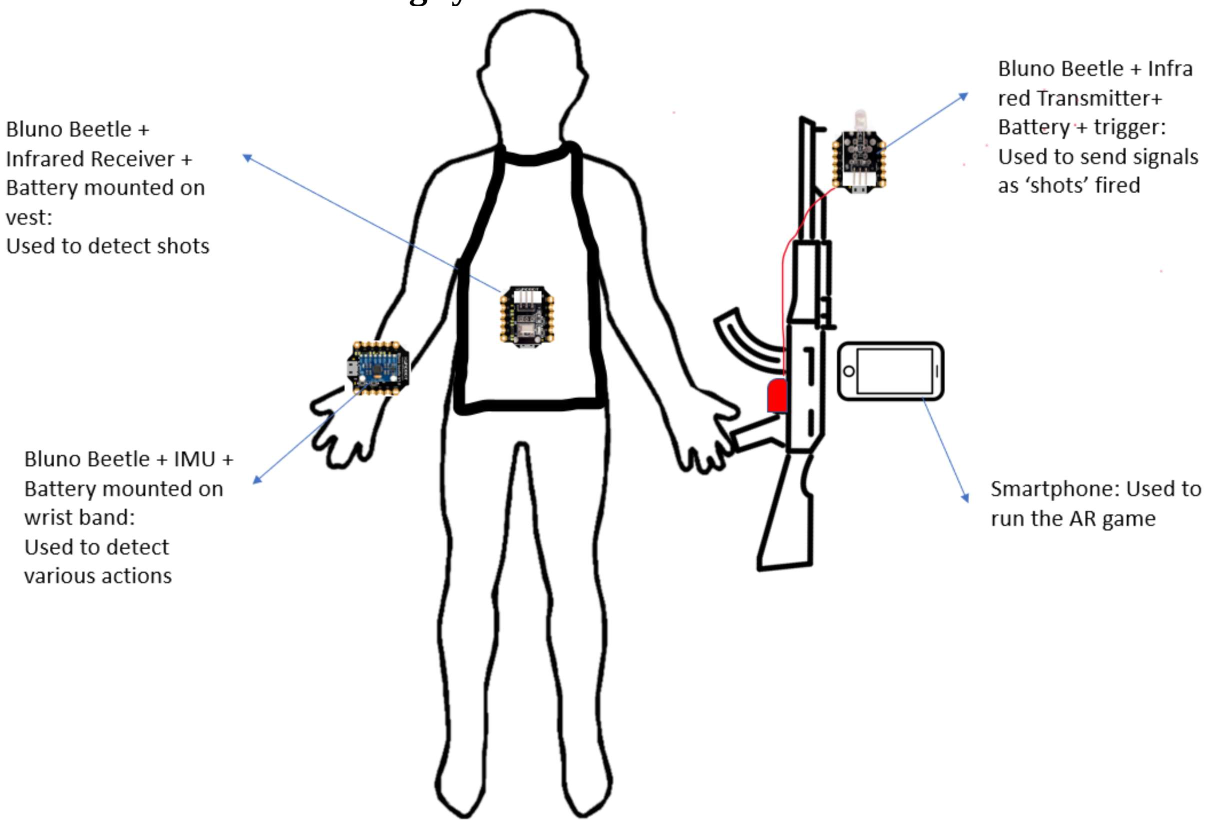

Bluno Beetle + Infrared Receiver + Battery mounted on vest: Used to detect shots
Bluno Beetle + Infra red Transmitter+ Battery + trigger: Used to send signals as 'shots' fired
Bluno Beetle + IMU + Battery mounted on wrist band: Used to detect various actions
Smartphone: Used to run the AR game

Figure 2.2: Final Hardware Design

The above diagram illustrates the final product as a set of wearable equipment on the players. This diagram covers the various hardware components used and their placement locations on the player's body.

2.3 Pseudo Game State Algorithm

To begin the game, we will run a script called `run_game`. This script will first initialize several global variables that, together, will describe the game state in its entirety.

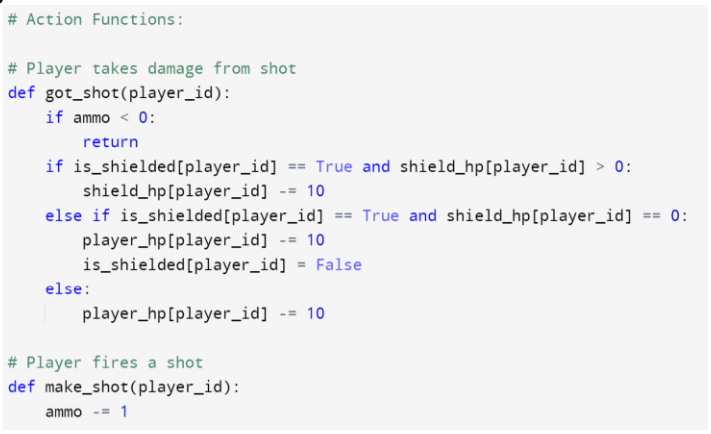
```
# Initial Game State
player_type = [F, E]
player_hp = [100, 100]
player_shield_hp = [0, 0]
is_shielded = [False, False]
player_shield_count = [3, 3]
player_grenade = [3, 3]
player_ammo = [6 , 6]
player_kill_count = [0 , 0]
datapoints[]
```

Figure 2.3a: Initialization of Game States

These global variables will be updated over the course of the game to reflect activities like "a player got shot" and "a player threw a grenade".

In the picture above, the various players have their current hp(s) stored in an array called `player_hp[]`. To reflect that *player 0* got shot, we would perform an update like so `player_hp[0] -= 10`.

After defining said global variables, we will define some functions like `got_shot()` and `take_grenade_damage()` that will access and update those global variables to reflect their corresponding actions.

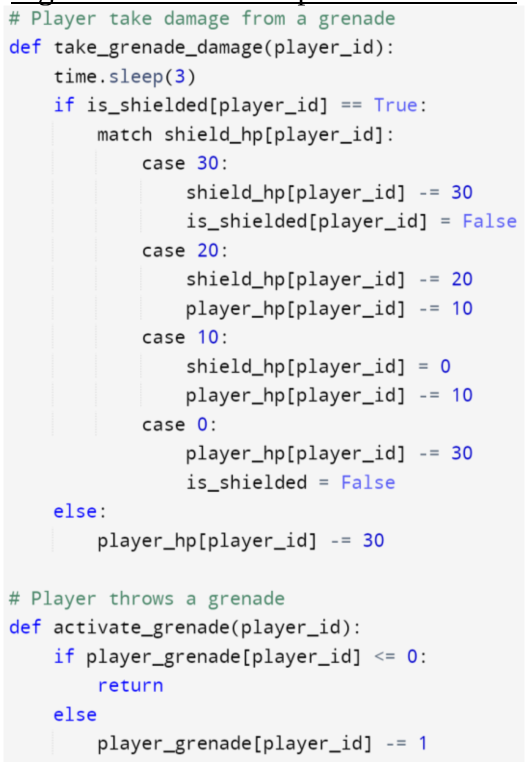


```
# Action Functions:

# Player takes damage from shot
def got_shot(player_id):
    if ammo < 0:
        return
    if is_shielded[player_id] == True and shield_hp[player_id] > 0:
        shield_hp[player_id] -= 10
    else if is_shielded[player_id] == True and shield_hp[player_id] == 0:
        player_hp[player_id] -= 10
        is_shielded[player_id] = False
    else:
        player_hp[player_id] -= 10

# Player fires a shot
def make_shot(player_id):
    ammo -= 1
```

Figure 2.3b: Shot computation function



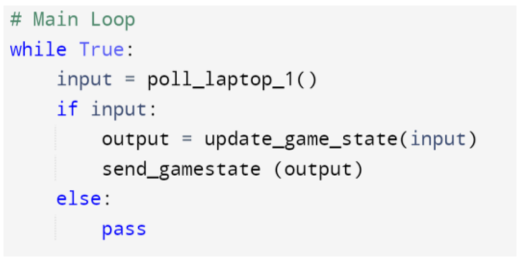
```
# Player take damage from a grenade
def take_grenade_damage(player_id):
    time.sleep(3)
    if is_shielded[player_id] == True:
        match shield_hp[player_id]:
            case 30:
                shield_hp[player_id] -= 30
                is_shielded[player_id] = False
            case 20:
                shield_hp[player_id] -= 20
                player_hp[player_id] -= 10
            case 10:
                shield_hp[player_id] = 0
                player_hp[player_id] -= 10
            case 0:
                player_hp[player_id] -= 30
                is_shielded = False
    else:
        player_hp[player_id] -= 30

# Player throws a grenade
def activate_grenade(player_id):
    if player_grenade[player_id] <= 0:
        return
    else
        player_grenade[player_id] -= 1
```

Figure 2.3c: Grenade computation function

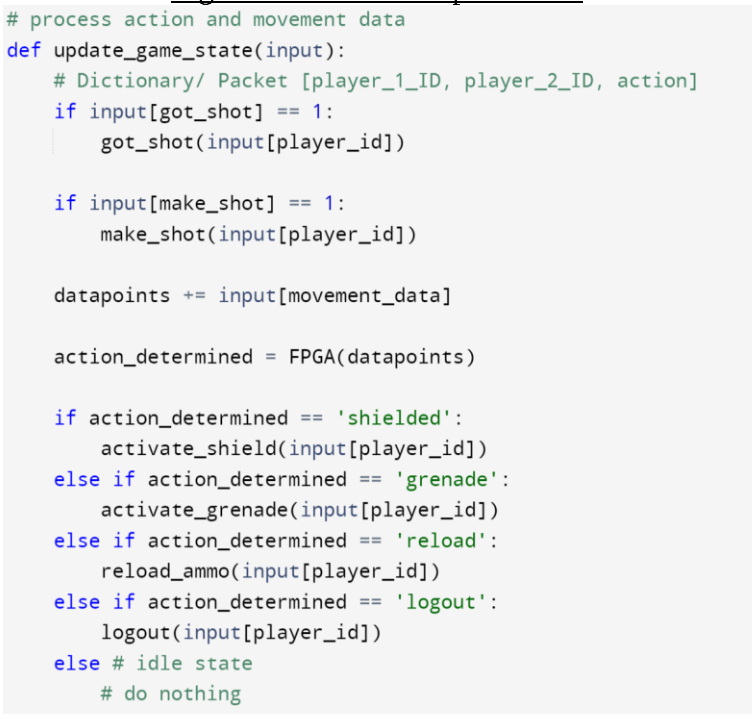
At the end of the script, after defining all global variables and helper functions, we enter an infinite loop that continuously polls data from laptop 1. For each command laptop 1 issues, we will parse it and execute it using the corresponding helper functions.

These helper functions will update the game state accordingly. Finally, we send the latest game state to the software visualizer for it to render or re-render the UI as needed.



```python
# Main Loop
while True:
    input = poll_laptop_1()
    if input:
        output = update_game_state(input)
        send_gamestate (output)
    else:
        pass
```

Figure 2.3d: Main loop function



```python
# process action and movement data
def update_game_state(input):
    # Dictionary/ Packet [player_1_ID, player_2_ID, action]
    if input[got_shot] == 1:
        got_shot(input[player_id])

    if input[make_shot] == 1:
        make_shot(input[player_id])

    datapoints += input[movement_data]

    action_determined = FPGA(datapoints)

    if action_determined == 'shielded':
        activate_shield(input[player_id])
    else if action_determined == 'grenade':
        activate_grenade(input[player_id])
    else if action_determined == 'reload':
        reload_ammo(input[player_id])
    else if action_determined == 'logout':
        logout(input[player_id])
    else # idle state
        # do nothing
```

Figure 2.3e: Update game state function

Section 3 Hardware Sensors

3.1 Hardware Details

The following section will provide in detail the various hardware components as well as supporting components being used in the laser tag system.

3.1.1 Hardware Summary

The following table provides an overview of the list of components being used:

Name of Component	**Quantity**	**Purpose**
Bluno Beetle BLE (DFR0339)	6	Collect data from sensors, process the data and then transmit them to laptop via Bluetooth
MPU6050 (GY-521)	2	Determines the relative position of the user's arm in 3-dimensional space, which will then be used to predict what actions did the user perform
Bluno Link	1	Adds Bluetooth 4.0 to PC
Ultra 96	1	Runs machine learning algorithms that is used to interpret sensor data and predict the actions of the users
IR transmitter and receiver	2	Simulate shots fired by the gun using IR signals

3.1.2 Bluno Beetle BLE (DFR0339)

Figure 3.1.2: Image of the Bluno Beetle Ble

The Bluno Beetle is one of the smallest arduino boards available on the market and is often used in wearable technologies. With an operating voltage of 5V and a clock frequency of 16MHz, the Bluno Beetle is also mounted with a CC2540 bluetooth chip[1]. In this project, the Bluno Beetles will be attached to the various sensors being used by the system and transmit data collected to the game server via bluetooth. The Bluno Beetles will also do some basic data-processing before sending the data.

Each player will receive 3 blunos, attached to the IR receiver, IR transmitter with trigger, and IMU respectively. The consideration for using 3 blunos is that since the gameplay will involve a lot of movements of the players, it is dangerous for the vest, gun and wristband to be connected using wires.

Related datasheet:

- Bluno_Beetle_SKU_DFR0339-DFRobot
- Bluno Beetle SKU:DFR0339

Related library:

- nil

3.1.3 MPU6050 (GY-521)

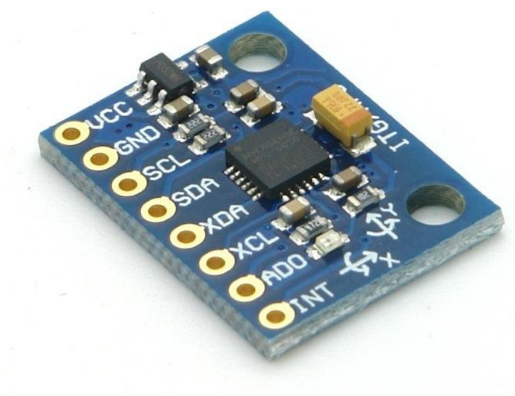


Figure 3.1.3: Image of the MPU6050 (GY-521)

The MPU-6050 IMU(Inertial Measurement Unit) is a micro electro-mechanical system featuring a 3-axis gyroscope, a 3-axis accelerometer, a digital motion processor and a temperature sensor[2]. In this project, we will be making use of its gyroscope and accelerometer to identify the relative positions of the user's arm with respect to 3-dimensional space. These datas will be delivered to the machine learning model, which will then be used to predict what movements have been made by the user.

Related datasheet:

- MPU-6000 and MPU-6050 Product Specification Revision 3.4
- MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2

Related library:

- MPU6050

3.1.4 Bluno Link (TEL0087)

Figure 3.1.4: Image of the Bluno Link (TEL0087)

The Bluno Link is a USB package that adds Bluetooth 4.0 to a PC, and supports bluetooth remote updates of Arduino programs[3].

Related datasheet:

- Bluno Link - A USB Bluetooth 4.0 (BLE) Dongle - DFRobot
- Bluno Link - A USB Bluetooth 4.0 (BLE) Dongle (Support Wireless Sketch Uploading) SKU:TEL0087

Related library:

- nil

3.1.5 Ultra96-V2 Processor

Figure 3.1.5: Image of the Ultra96-V2 Processor

The Ultra96-V2 Processor is the development board that will be used to host the machine learning algorithms for this project. Sensor data to be used for FPGA will be pre-processed by the Bluno Beetles, and then sent to the Ultra96 remotely through WiFi.

Related datasheet:

- Ultra96-V2 Single Board Computer Hardware User's Guide

Related library:

- nil

3.1.6 Infrared transmitter and receiver

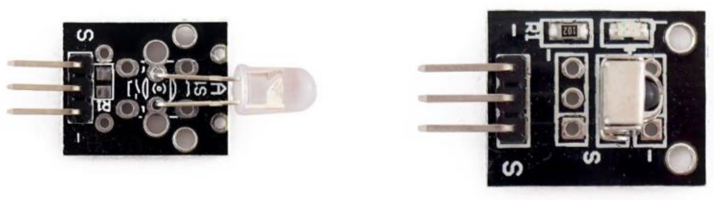


Figure 3.1.6: Images of the Infrared transmitter and receiver modules

The Infrared transmitter and receiver modules will be the sensor used to detect if a valid shot is fired. Once the trigger is pressed, a burst of signal will be fired from the transmitter. The receiver will be turned on throughout the duration of the gameplay, and when it receives a signal from the transmitter, it is considered a hit. This information will be used by the Ultra-96 to decide if the opponent will deduct health points. The bluno attached to the transmitter will also send an information everytime the button is pressed regardless of whether a hit is registered by the receiver. This allows us to account for shots that missed. Since the signal chosen has a frequency of 38kHz, it will have minimal interference from the surrounding.

For the first part of the design, we will try to attach a single receiver to the front of the vest. Since the transmitter has a transmitting angle of 20° and the receiver has a receiving angle of 90°, we will test out the size of the hitbox we have once we set up the equipment, and from there decide if we need to increase the number of receivers. If necessary, we will need to attach an additional receiver to the back of the vest to register shots that hit the back of a player.

Related datasheet:

- IR (Infrared) Transmitter / LED Module – Smart Engineering Solutions
- IR (Infrared) Receiver Module 38kHz – Smart Engineering Solutions

Related library:

- GitHub - Arduino-IRremote/Arduino-IRremote: Infrared remote library for Arduino: send and receive infrared signals with multiple protocols

3.2 Pin Table

Below are the pin mapping tables between the various components and the Bluno Beetles.

Pin Mapping between Beetle and MPU-6050 IMU:

Pins on Bluno Beetle	Pins on MPU-6050 IMU
5V	VCC
GND	GND
SCL	SCL
SDA	SDA
D2	INT

Pin Mapping between Beetle and IR Transmitter:

Pins on Bluno Beetle	Pins on IR Transmitter	Pins on Switch
5V	VCC	VCC
GND	GND	GND
D3	S	-
D2	-	OUT

Pin Mapping between Beetle and IR Receiver:

Pins on Bluno Beetle	Pins on IR Receiver
5V	VCC
GND	GND
D3	S

3.3 Schematic

This section will display the schematics for the various components used in this project

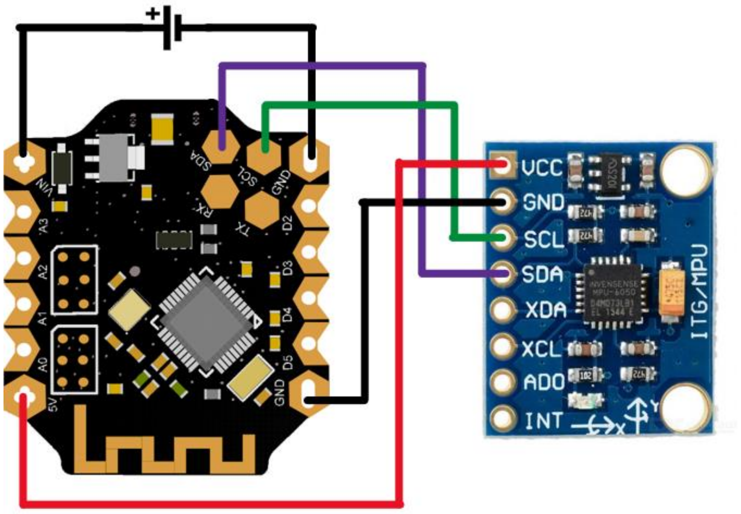


Figure 3.3a: Schematic between Beetle and MPU-6050 IMU

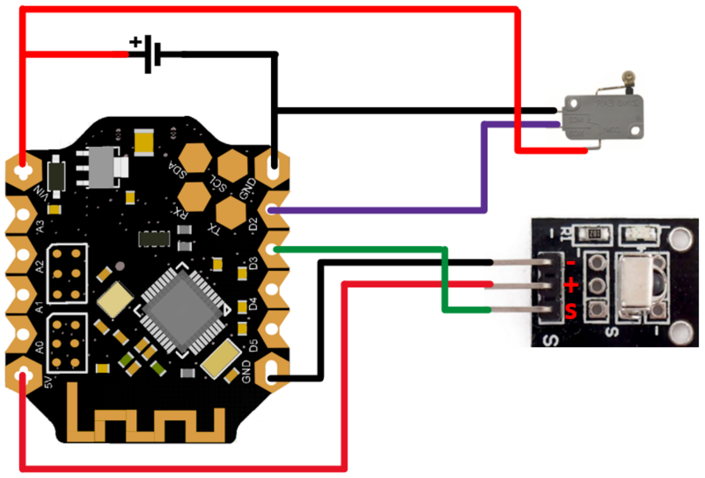


Figure 3.3b: Schematic between Beetle and IR Transmitter with switch

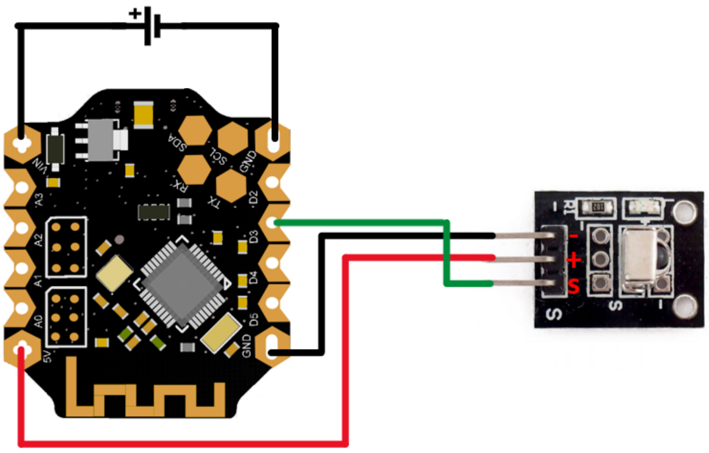


Figure 3.3c: Schematic between Beetle and IR Receiver

3.4 Operating voltage

This section will list out the operating voltage and the current drawn of the various components being used and discuss the relevant power design in this project. Below is a table listing the operational voltage of the various components.

Component	Voltage(V)	Current(mA)	Power(mW)	Remark	References
Bluno Beetle	5	14	70	Active, 16MHz	ATmega328P
MPU-6050 IMU	2.375 ~ 3.46	3.8	13	Gyroscope + Accelerometer (DMP disabled)	MPU-6000 and MPU-6050 Product Specification Revision 3.4
IR Transmitter	1.2 ~ 4.0	100 (continuous forward current)	150		IR (Infrared) Transmitter / LED Module – Smart Engineering Solutions
IR Receiver	2.7 ~ 5.5	unspecified	30	Current drawn assumed to be 10mA based on similar modules	IR (Infrared) Receiver Module 38kHz – Smart Engineering Solutions
Power needed for Beetle + IMU			83		
Power needed for Beetle + Transmitter			220		
Power needed for Beetle + Receiver			100		

Based on the calculations in the table above, we can conclude that the battery capacity needed for individual beetles is not that high, given that each beetle will only oversee one component. Below is a list of considerations we have in terms of the battery we use:

- Cost: Since the project lasts across 3months with regular testing, getting rechargeable batteries/portable batteries will greatly reduce the cost incurred.
- Weight: Components are either worn on the wrist, the waist, or mounted on a gun which will be carried for extended durations. As such, light-weight batteries will be ideal for powering the wearables.

Given the above considerations and the power requirement of the project, our group has decided to use rechargeable Nickel-metal Hydride Batteries, since they are small and have less recurring cost. Typical AAA NiMH batteries have a capacity of around 800mAh, while AA NiMH batteries have capacity of around 2000mAh. Both have a battery voltage value of 1.2V. A single AAA battery is sufficient to support at least 2 hours of testing/game time, thus making NiMH battery a viable option.

Depending on the hardware components the team acquires in the future, we will either consider using a 1.2V to 5V voltage regulator, or to use a series connection of 4 batteries to power the Bluno Beetles.

Section 4: Hardware AI

This section describes the implementation details of the hardware AI.

4.1 Ultra96 Synthesis and Setup

The ML training and C++ source code development will be done locally on my laptop. Once both are complete, I will use Vivado to generate a bitstream from the C++ source code using high-level synthesis (HLS).

Lastly, the FPGA hardware will consume this bitstream and be programmed.

4.2 Neural Network

I will be using a multi-layer perceptron (MLP) consisting of four layers:

1. Input layer (~100 neurons)
2. Hidden layer 1 (~100 neurons)
3. Hidden layer 2 (~100 neurons)
4. Output layer (5 neurons)

4.3 Model Training and Testing

I will perform feature extraction on the gesture data and use them as the input layer when training the ML model. The actual training will be conducted by a third-party library (TensorFlow, PyTorch, etc) using generic techniques like backpropagation and gradient descent.

On the output side, there will be 5 neurons corresponding to these verdicts:

(i) Grenade

(ii) Shield

(iii) Reload

(iv) Game End

(v) Neutral (a.k.a "none of the above")

Testing will be done by splitting the gesture data into training and validation data in a 8:2 ratio. Specifically, 80% of the gesture data will be used to train the ML model. Once that is done, the ML model will attempt to predict the remaining 20%.

Performance/ accuracy of the ML model is equal to the percentage of validation data correctly predicted.

4.4 Realizing the Neural Network

Once the MLP has been trained, I will download the final weights and biases in a text file. Using metaprogramming, I will write a script that will write a C++ file containing these weights and biases as global variables.

Then, the C++ file will use these hardcoded values to perform the computations a MLP model would have done to derive the values of the output neurons. That is, I will manually compute the values of the N^{th} layer based on the $N-1^{st}$ layer using the hardcoded weights and biases until I reach the output layer.

4.5 Evaluating design's timing, power and area

The Vivado framework comes with an in-built “Report Power” feature which can be used to evaluate the design's timing, power and area with the given dataset.

Section 5 Internal Communications

For this section, we will be going over all the details of communication between the Beetles and the laptop. It will be split into 4 main subsections that aim to impart the following:

- A high-level overview of the task management and data flow between the Beetles and the laptop
- A rundown on how to set up and configure a basic BLE connection between the Beetle and laptop
- The protocol specifics on how the Beetle establishes and coordinates the connection with the laptop
- Potential reliability and security issues and how we intend to resolve them

5.1 Task Management on the Beetles and laptop

During the planning process, we focused upon 3 main design considerations that play a significant role in affecting our decisions on which protocols and methods to use. These considerations are Connection Speed, Reliability and Power Consumption.

5.1.1 Tasks on the beetle

The following table depicts the tasks on the Beetles.

Task Name	Specific Beetle	Description
Acknowledge()	General	Sends an acknowledgement that message has been received
SendIMUData()	Wrist Beetle	Collects and sends data from the IMU
SendIRData()	Vest Beetle	Collects and sends data from the IR receiver
SendTriggerData()	Gun Beetle	Sends whether the trigger has been activated
Sleep()	General	Puts the beetle into a sleep state

5.1.2 Processes on the laptop

The following table shows the different processes on the laptop

Process Name	Description	Frequency
CollectRequest()	Sends a request to the beetle to start sending over data	150Hz (See section 5.3.4 Baud Rate for details)
ProcessData()	Processes the data sent over	Variable
ReadData()	Reads the data from the beetle	Variable
Acknowledge()	Sends an acknowledgement to the laptop	Variable

5.1.3 State Machine Diagram

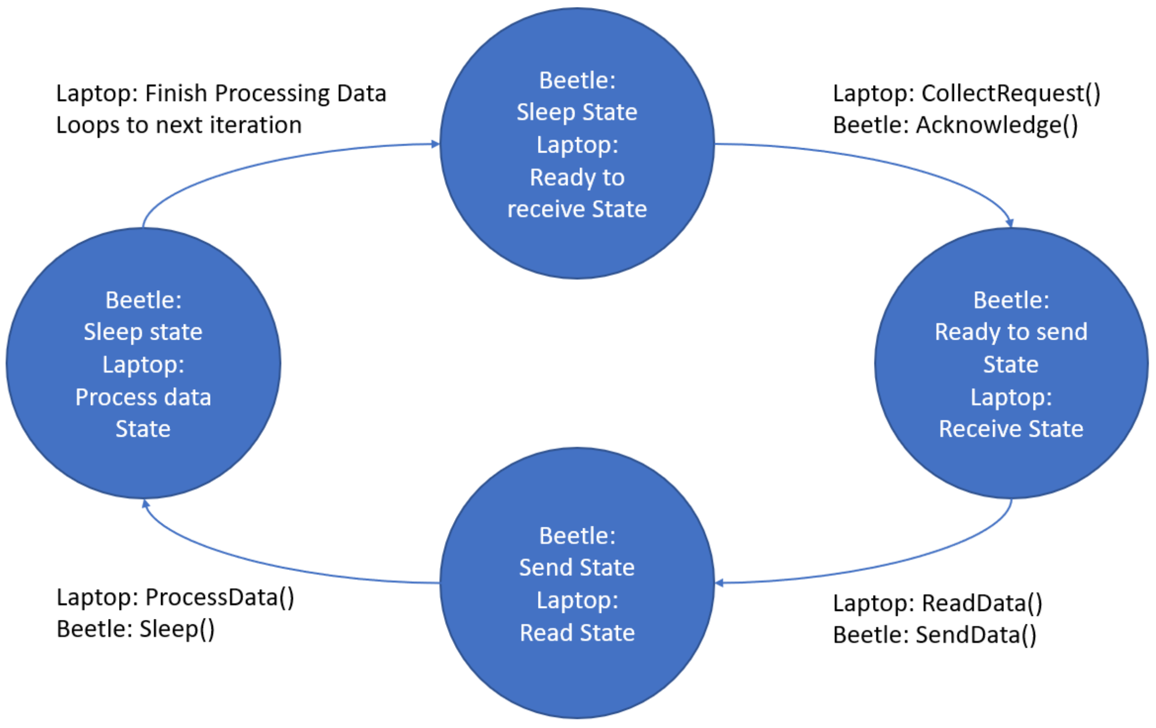


Figure 5.1.3: Connection State Machine Diagram

Our data acquisition is designed around a **Periodic Polling approach** by the laptop. The communication will be over the **Bluetooth Low Energy (BLE)**. Within the BLE specification, the beetle will be configured as the peripheral device and act as a data server and the laptop will be set up as the central device with the role of a client to access data. The laptop will first connect to the beetle via the bluepy API using the MAC address of the beetle that will be previously discovered (more on this in section 5.2). After the connection, the beetle will initially be in a sleeping state. When the laptop is ready to receive data, it attempts to initiate a connection to the beetles by sending them a handshake packet. This can be done through using the write request ATT operation on Characteristic Properties[4] for the Characteristic that the beetle will be accessing. This initiates the 3-way handshake protocol between an individual beetle and the laptop (see section 5.3.1 for details). This process will wake the beetles from their sleeping state and confirm that both parties are ready for communication. The beetles will then begin to send data periodically to the laptop by writing to a GATT Characteristic that the laptop will then access via a known unique UUID. At this stage, the laptop is coordinating between data input threads from 3 beetles. After obtaining enough data for an interval, the laptop will send a packet to alert the beetle to stop broadcasting data. Upon receiving this, the beetle stops updating the GATT characteristic and enters a sleep state again, waiting for the handshake packet from the laptop.

5.1.4 Having a sleep state

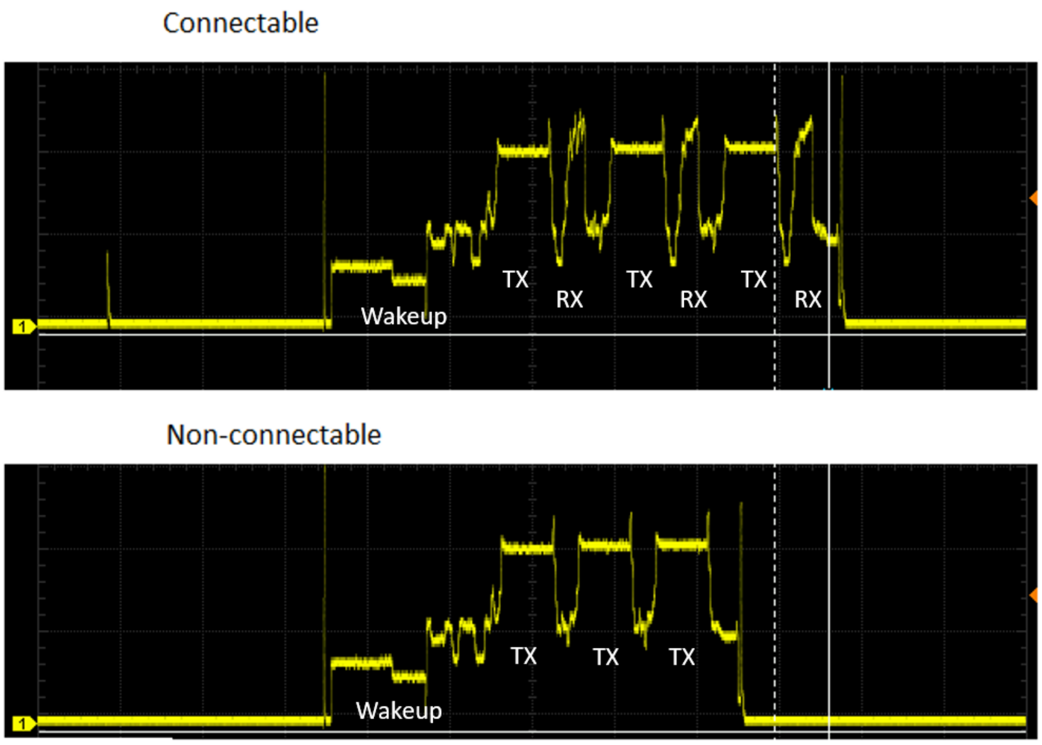


Figure 5.1.4: Power draw against time diagram[5.2]

From the graph above, we can see that a BLE connection that is constantly polling its data is very energy intensive, especially when transmitting data[5], the current has to be maintained at a high level to ensure that the transmission is successful. Hence, implementing a sleep state will enable the Beetle to only be polling its data when the laptop is ready to receive it, hence saving a lot of power as the beetle is not polling any data that will not be directly used by the laptop.

5.1.5 Handling Concurrency on the Beetle

To handle concurrency on the beetle we are intending to just use a loop with conditionals to manage the different beetle states. Examples of conditionals that decide the beetle's state would be whether a Handshake packet was received or what its previous state was. Based on the different conditions, the beetle will decide on the proper instructions to execute.

While running an RTOS may seem like an attractive option to ensure that code runs deterministically and provide future expandability for the Beetle's code. We are choosing not to use an RTOS as the current running requirements of the beetle simply do not require the need for it. We are basing this on two considerations. First, the beetle does not have many concurrent tasks running. RTOS benefits the beetle by ensuring that the beetle can schedule and execute all its tasks reliably. However, the few tasks running on the beetle should not have a high scheduling demand. Hence, the main benefit of an RTOS would be wasted. Second, the running of an RTOS requires a high commitment of resources[6]. The limited storage and processing resources on the beetle could be better utilized in storing and processing the sensor data.

5.1.6 Multi-Threading vs Multi-Processing on the Laptop

For the laptop, it will have to maintain a stable pairing and connection with at least 3 beetles for a single player game. This would mean that it would have 3 different BLE data streams communicating simultaneously. We plan to achieve this using the multithreading python library. Which can be used to run beetle FSM threads simultaneously. Because modern day laptops have processing speeds of several Ghz, it is more than capable of receiving and processing the data received from multiple beetles. The code itself is more network bound through the speed of the BLE connection, as compared to being CPU bound by any extremely complicated calculations that the CPU might need to perform on the data. Hence, we feel like we don't need the added computational power associated with using multi-processing. Using threading, the same memory is shared among the threads. With the use of "locks" in the multithreading library, we can maintain threadsafe access to common variables and thereby build a resulting data packet that combines the data received from the beetles.

5.2 Setup and configuration of the BLE interfaces

We plan to follow the following steps to set up the BLE interface between the laptop and the Beetles.

1. First connect the Beetle to the laptop and open the Arduino IDE application. Make sure to select the right board (Arduino Uno) and the right port option.
2. Open up the serial monitor and ensure that the settings in the two drop-down boxes in the bottom right are "No line ending" and "115200 Baud". After which type the following command into the text box at the top and send

   ```
   +++
   ```

   If successful, the serial monitor will return the phrase "Enter AT Mode". If unsuccessful, simply retry from step 1 and the beginning of step 2.

3. Change the setting in the first drop-down box from "No line ending" to "Both NL & CR". Then key in the following command and send

   ```
   AT+SETTING=DEFPERIPHERAL
   ```

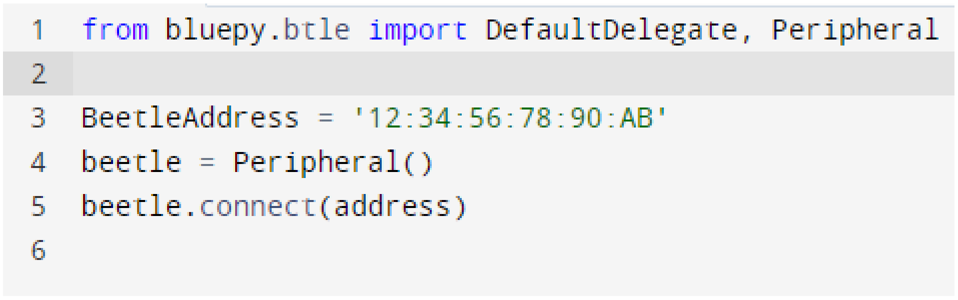
   A successful response will be "OK" while an unsuccessful response will be "ERROR CMD". If unsuccessful, simply retry the command and check that the settings are correct. This step makes the Beetle discoverable as a BLE device and sets it up as a peripheral that can be connected to.

4. After making the Beetle a discoverable BLE device, we can then use an application like "BLE Scanner" on the android or "LightBlue" on IOS to discover and record the MAC address of the beetle (eg. 12:34:56:78:90:AB). This MAC address will help us to establish a Point-to-Point connection to the Beetle.
5. We are running the BLE client on Linux OS and hence we have access to the bluepy API to interface with BLE devices. We will use the following commands to include this dependency. (Ensure that you have python installed onto your computer)

   ```
   sudo apt-get install python-pip libglib2.0-dev
   ```

   ```
   sudo pip install bluepy
   ```

6. Here is a code sample that makes use of the bluepy API and the acquired MAC address to connect to the Beetle.



```python
from bluepy.btle import DefaultDelegate, Peripheral

BeetleAddress = '12:34:56:78:90:AB'
beetle = Peripheral()
beetle.connect(address)

```

Figure 5.2: Sample connection code

With the bluepy API, only the static MAC address is needed to establish a Point-to-point connection with the Beetle. Hence, there is no need to continually scan for the Beetles in our connection protocol as we will have a lookup table already hard-coded into the beetles for the connection. After the connection, other features in the API can be used to communicate between the Beetle and the laptop.

5.3 Connection protocol between the Beetles and laptop

In this section, we will be going into the details of how our beetle will initiate communication with the laptop and the different packet types and formats. Finally, we will touch upon our estimations for the protocol's data rate.

5.3.1 Handshake Protocol

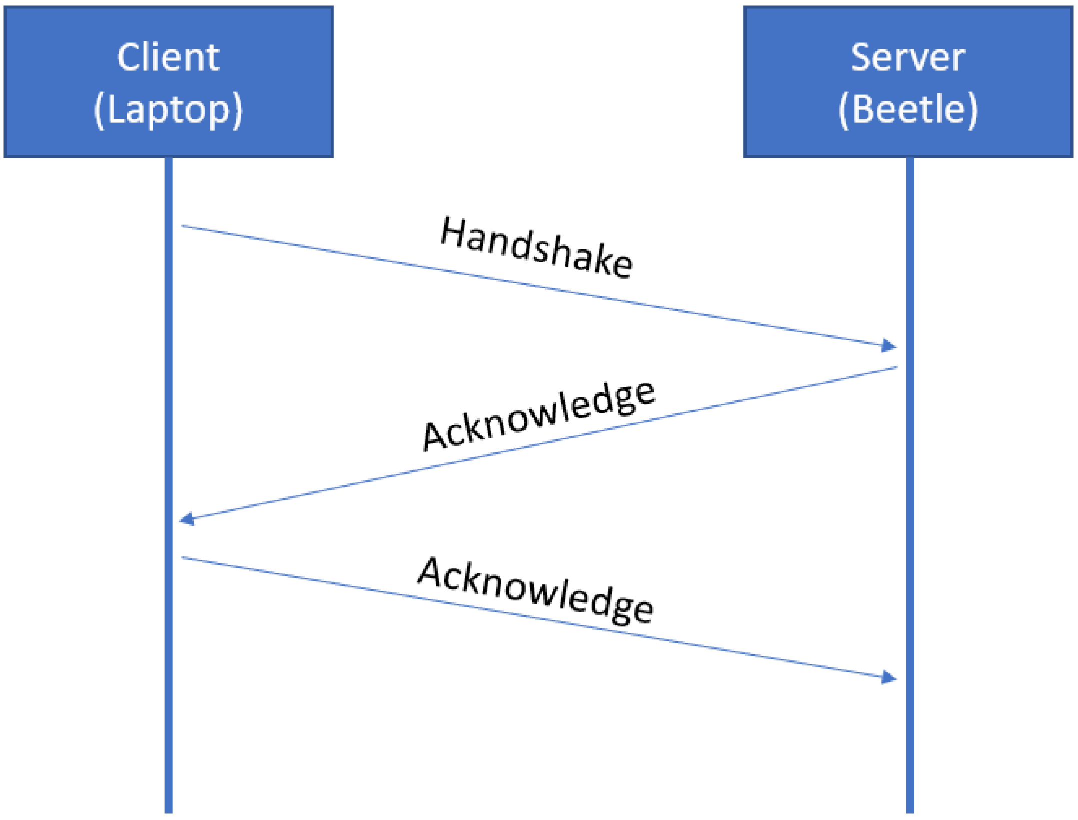


Figure 5.3.1: Handshake Protocol Diagram

To wake the beetle from its sleep state, we implement a 3-way handshake protocol. In the first step, the laptop initiates the communication by sending a handshake data packet to the beetle, signaling to the beetle that it is ready to accept more data inputs and for the beetle to start sending data over. Upon reading this handshake packet, the beetle sends an acknowledgement packet to the laptop. This lets the laptop know that the beetle is ready to send over new data. Finally, the laptop sends its own acknowledgement packet to the beetle to once again confirm with the beetle that it is ready to receive the new data packets. Thereby ensuring that both the beetle and the laptop are aware of each other's readiness to connect.

5.3.2 Packet Types

Packet Type	Packet Identifier	Description
Handshake	‘H’	Sent from the laptop to the beetle to wake the beetle
ACK	‘A’	Sent by both the beetle and the laptop to acknowledge packets sent
IMUData	‘I’	Sent from the beetle to the laptop containing IMU Data
TriggerData	‘T’	Sent from the beetle to the laptop containing Trigger Data
IRData	‘R’	Sent from the beetle to the laptop containing IR Receiver Data
Stop	‘S’	Sent from the laptop to the beetle to tell the beetle to go back to sleep

Packet types will be in utf-8 format, which has a maximum of 4 bytes.

5.3.3 Packet Format

In the packets that we are sending over, we are only interfacing with the ATT Payload portion of the data packet[7], which happens to be the section that corresponds to application accessible data. We plan to implement a constant size for all the packets sent to adhere to, using padding to fill any extra bits. This is to ensure that the packet size from the application layer is consistent. We will also be sending the identity of the beetle sending the message, this information would include the player the beetle is assigned to as well as the component the data is being sent from.

Component Type	Component ID	Expected data corresponding to player 1 (Hex 0x00: 1byte)
IMU	1	11
Trigger	2	12
IR Receiver	3	13

A sample of the IMU data packet is shown below. The IMU provides 3 accelerometer readings and 3 gyroscope readings which are 2 bytes each. Resulting in the data being 12 bytes in total.

Category	Packet Type	Data	Identity	Padding
Data Type	char	int	byte	byte
Data Size (bytes)	4	6 * 2 = 12	1	3

Total size: 20 bytes

A sample of the Handshake data packet is shown for comparison as well. Since the handshake packet is from the laptop, the timestamp is not critical for the game state and will not be included.

Category	Packet Type	Padding
Data Type	char	byte
Data Size (bytes)	4	16

Total size: 20 bytes

This maximum total packet size is verified to be possible from the BLE 4.0 specification which has a maximum ATT payload size of 20 bytes[7].

5.3.4 Baud Rate

The Baud Rate we choose is 115200Hz. A higher Baud Rate corresponds to a higher rate of bit transfer[8]. Hence more data can be transferred between the beetle and the laptop at any given time, making our system faster as a larger amount of critical data can be accessed. Additionally, with a higher rate of data transfer, the time the radio is on for the beetle can be less as we can transfer all the data needed in less time. Hence, because the radio is on in smaller intervals, more power can be saved that way.

With a Baud Rate of 115200Hz, we can approximate that the expected speed of the transfer would be 115200 bits/s = 14400 bytes/s. This is verified to be possible from the CC2540 chip on the Bluno, which has the specification of LE 1M PHY[7], which corresponds to a theoretical speed of 1Mbps. Based on the estimation of 14400 bytes/s and our standard packet size of 20 bytes, we can assume to send a theoretical maximum of 720 packets every second. However, this is unrealistic as there will be other factors such as inter-frame spacing and unaccounted packet overhead. Based on this article on supposed throughput via a CC2540 chip[9], we have a conservative estimate of about 150 - 200 data packets/s being transferred from the beetle to the laptop.

5.4. Handling reliability issues

In this subsection, we will be introducing some possible reliability issues in communication and explaining how they will be resolved either through existing methods or the application's communication protocol.

5.4.1 Corrupted data packets

The BLE specification comes with Cyclic Redundancy Check (CRC)[7] that helps in error detection. The CRC block is appended to the frame during communication at the sender and then used by the receiver to check if there are any bit errors in data received. Hence both the laptop and the beetle would be able to detect if there are any corrupted bits in the messages during communication. Hence, we believe that the CRC implemented in the BLE specification will be enough to handle the burst and single bit errors in data transfer. Detecting this error will trigger a resending of the corrupted data packet.

5.4.2 Possible disconnection

At any point, there may be beetles that disconnect for reasons such as distance or low battery power. The first solution that we have to this is a connection timeout mechanism in the client code that will trigger a flag in the application after a certain number of Handshakes have not received responses. A forced disconnect will then be called in the code to first ensure that the connection has been disconnected and clear up the resources associated. Finally, the connection method will be called to attempt a connection again. Our implementation of choosing the use of the pre-discovered values of the beetle MAC addresses is helpful here because we can skip the step of having to rediscover the beetle. Allowing for even faster reconnection.

Our second solution to deal with disconnections will be to station laptops strategically around the playing field to maximize the signal coverage. These laptops will act like radio towers for beetles to connect to. When a disconnect happens, the laptop may coordinate a poll to establish a connection with the disconnected beetle. With more laptops available at various locations, the chances that a beetle may be able to reconnect to a laptop successfully is higher.

5.4.3 Security

The BLE 4.0 specification comes with AES-128 encryption[10]. Which by modern-day standards is considered safe[11]. In addition, the nature of the BLE connection is also a Point-to-point connection. This does not allow packet interception by other sources which may attempt packet sniffing via an intermediary device like a router. To enable the security offered by the BLE specification, an additional physical step will be added. This is to pair the beetles to the laptop via the laptop's bluetooth pairing before initiating any handshake protocol.

While the BLE communication protocol is safe, it is not the only factor that comes into play when we are running the system. A vulnerability presents itself from a feature of the beetle, which allows for wireless programming. An outside party can discover the MAC address of our beetles and wirelessly program the beetle, replacing our code with potentially malicious code without us knowing.

Our solution to this would be a physical policy when handling beetles. First, to keep the beetles labeled and identifiable as belonging to the team. Second, the beetles will all be uploaded with the most updated code before every connection trial to ensure that there is no code tampering before usage.

Section 6 External Communications

The external communication system encompasses the individual players' laptops, an Ultra96 FPGA for data predictions, an evaluation server and phone visualizers for graphical effects. This section details how secure wireless communication between the various components is achieved. Following, clock concurrency and synchronization between these components will be further elaborated.

6.1 High level overview of external communications

The following figure provides an overview of the network communication

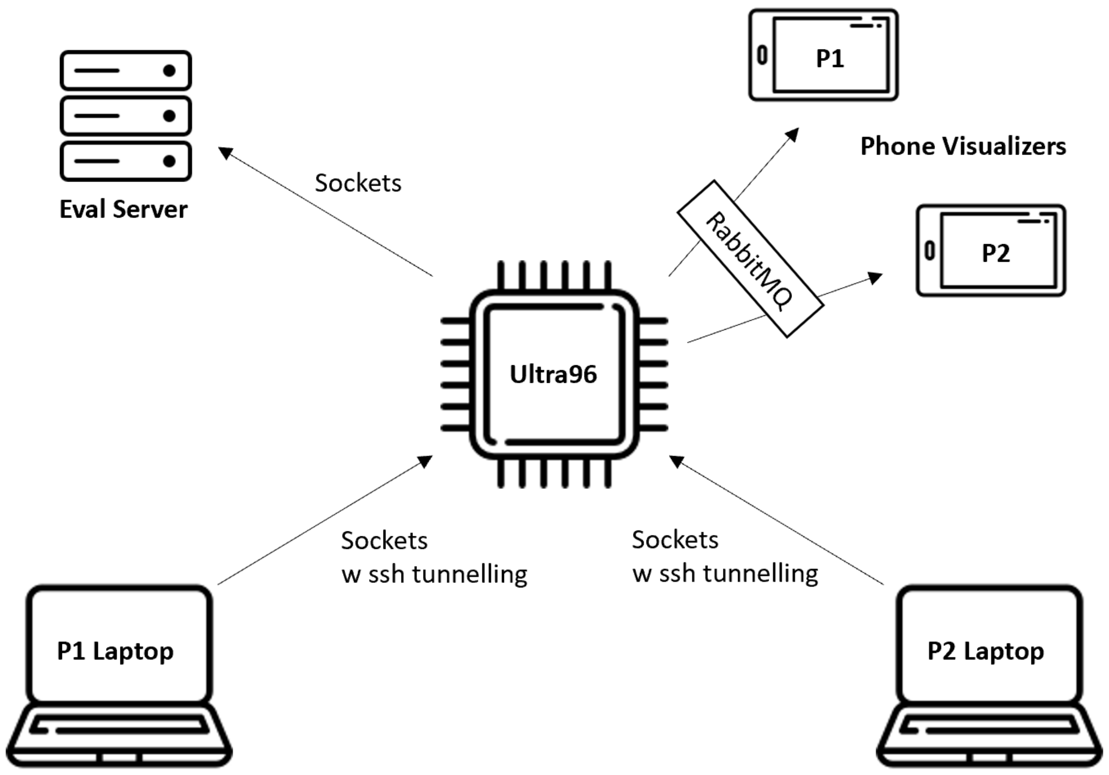


Figure 6.1: Overall external comms design

6.2 Communication between Ultra96 and the evaluation server

Secure communication between the Ultra96 FPGA and the evaluation server is established via Transmission Control Protocol/ Internet Protocol (TCP/IP) and can be programmed using the already available Python Socket Programming library. The sockets and socket API allow messages to be sent across a network locally or across other networks. TCP/IP is our preferred protocol over User Datagram Protocol (UDP) as it is more reliable and can deliver data in-order[12]. This is important as we want to ensure every action data is read rightly by the evaluation server.

The eval_server.py script first creates a listening socket on the evaluation server. Following, the script attempts to bind the inputted IP address and port number with the socket. Thereafter, the script will listen and wait to accept a single incoming connection. Meanwhile, a client.py script creates a socket on the Ultra96 and attempts to connect with the same local IP address and port number as the evaluation server. When the evaluation server accepts the TCP/IP connection from the client script, a two-way communication is established between the server and the Ultra96.

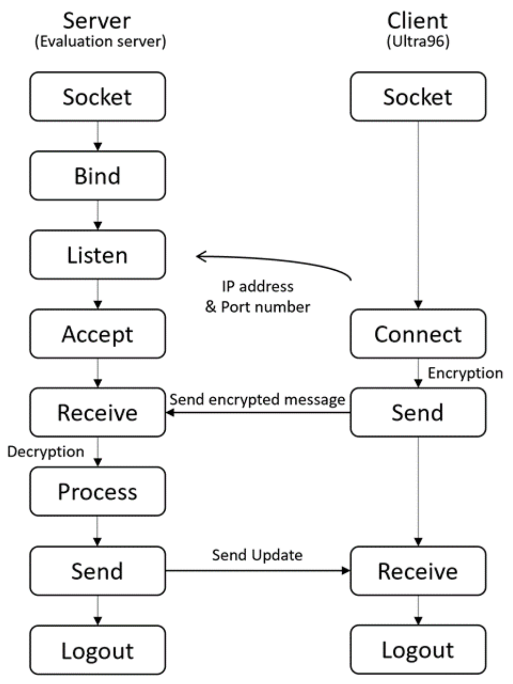


Figure 6.2a: Socket communication flow

Once a connection is established, the Ultra96 will send an encrypted message string to the evaluation server. Upon AES decryption on the evaluation server side, the expected message string should consist of 2 segments in the following format.

‘#action_1|action_2|’

Figure 6.2b: Expected message string sample

“#action_1” represents the expected action from Player 1 while “#action_2” represents the expected action from Player 2. There are a total of four in-game actions that each player can do; ‘shoot’, ‘shield’, ‘grenade’, ‘reload’. There will also be a logout action that allows each player to exit the game. One such example is as shown below.

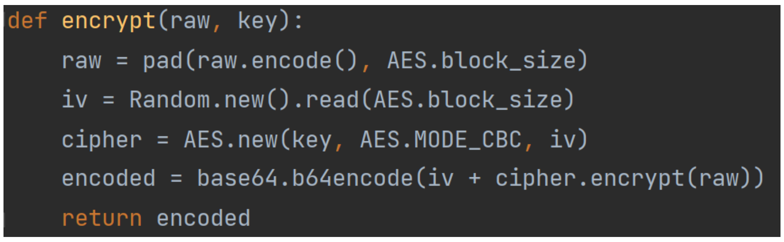
‘#reload|shield|’

Figure 6.2c: Example message string sample

To ensure that communication between the client and server remains secure, an Advanced Encryption Standard (AES) scheme is implemented to protect the above message string during delivery. Similarly, AES can be easily implemented in python using the Cryptodome Cipher Library. AES is a symmetric block cipher for which it is capable of handling 128-bit blocks using differing key sizes of 128, 192 and 256 bits[13]. In terms of security, AES is a better option compared to Data Encryption Standard (DES) as it has longer keys, hence a hacker will be less likely to gain access to any information. In addition, AES is the most widely used encryption algorithm in the current context. Other special mentions include the

RSA algorithm however, RSA is more computationally intensive and hence it is slower[14]. In the context of our project, AES will be the more viable option as it warrants security and is more incline with our real-time requirements. We opted to use AES-128 with Cipher Block Chaining (CBC) encryption mode as it is more than sufficient to ensure security for our use cases and requires the least processing power.

Before the encryption process, the script will randomly generate a 16-bit variable that acts as the secret key or initialization vector (IV). The randomly generated variable ensures that the encrypted string will be unique every time regardless of the contents of the message or keys. Next, the raw string input parameter will be converted into a byte string using an encode function and then padded with a block size of 16 bytes. Lastly, the cipher, together with the IV, will encrypt the padded byte input using base64 encoding before the encrypted message can be sent to the evaluation server.



```python
def encrypt(raw, key):
    raw = pad(raw.encode(), AES.block_size)
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    encoded = base64.b64encode(iv + cipher.encrypt(raw))
    return encoded
```

Figure 6.2d: Encryption function snippet

The evaluation server has its own AES decryption function that will decrypt the encrypted message using the same key from the encryption process. Once decrypted, the original message will display the respective actions performed by the two players in a string. Lastly, these data will be evaluated and compared with the expected output to determine its correctness.

6.3 Communication between Ultra96 and the players' laptops

Like communication between Ultra96 and evaluation server, the communication between Ultra96 and the laptops will also make use of the TCP/IP sockets. The Ultra96 will acts a remote server to the two laptop clients, allowing data to be transmitted once connection is established. As the Ultra96 is isolated within the SOC's Sunfire network, it can only be accessible remotely through SSH tunneling with port forwarding via the Sunfire network. "sshtunnel" python library is expected to be used to make the connection between the Ultra96 and the players' laptop[15].

6.4 Concurrency handling

Due to the sheer amount of data to be transferred from the Beetle to the Ultra96 and to the phone visualizer, using a purely synchronous and sequential application for the game may not be sufficient in terms of performance. Since communication over the network plays a pivotal role in this project, the various programs are likely to be I/O-bound without any alterations. Hence, the most plausible way to improve performance will be to reduce the waiting time between the sent request and response. [16].

Concurrency can be the solution to achieving satisfactory run time while reducing the waiting time between responses. Since most of our code will be written in Python, we will be looking to use "threading" or "asyncio" python libraries for concurrency.

Meanwhile, concurrency can be used within the Ultra96 when running the game mechanism. This includes updating game state and players current status (HP, Shield HP, number of ammo left). We must prioritize the various game actions throughout the game. For example, if player 2 raises up their shield, the shield hp will have to take precedence over the other actions and temporarily boost player 2 shield hp before taking any damage. Multiple priority threads will be required such that the game state will be updated sequentially and the game can run smoothly.

6.5 Communication between Ultra96 and the phone visualizers

To allow smooth data transmission from the Ultra96 to the phone visualizers, we are exploring the possibilities of using an online database or application, such as PostgreSQL, MongoDB and RabbitMQ.

6.5.1 Analysis of data transmission application

PostgreSQL is an open object-relational database system that allows the use of SQL language and provides a gateway for our phone application to access the current game state for updating the UI within the application[17].

Meanwhile, MongoDB is a NoSQL database that allows high volume data storage. It is a document-oriented database system for which it stores data in documents, therefore it will be easier to use[18].

As for RabbitMQ, it is more of a message broker, which supports several message protocols. It acts as a “middleman” that helps reduce load and delivery times by delegating tasks efficiently while allowing producers to publish data and consumers to subscribe to them. It is also considerably lightweight and handles data quickly, allowing fast transmission of data[19].

6.5.2 RabbitMQ for data transmission

Considering the three plausible data transmission applications and our high time priority requirements, we are more inclined towards RabbitMQ as it is the most lightweight and can transfer data quickly. Additionally, since we do not need to technically store data, PostgreSQL and MongoDB key advantages become less of a priority.

RabbitMQ offers both MQTT and AMQP as messaging protocols. AMQP provides a more secure messaging feature over MQTT, therefore, we will be looking to work with it as our primary protocol and if it fails, we will proceed with MQTT as an alternative. [20]

Using the AMQP message model, the Ultra96 will act as a producer and continually publish filled data packets to the RabbitMQ broker. The data packets are sent as a dictionary with a range of key-values pairs. Within RabbitMQ, an exchange will accept the data packets and directly route it to the correct message queue. Finally, the software visualizer as the consumer will subscribe and retrieve these messages from the queue and display them on the screen as UI.

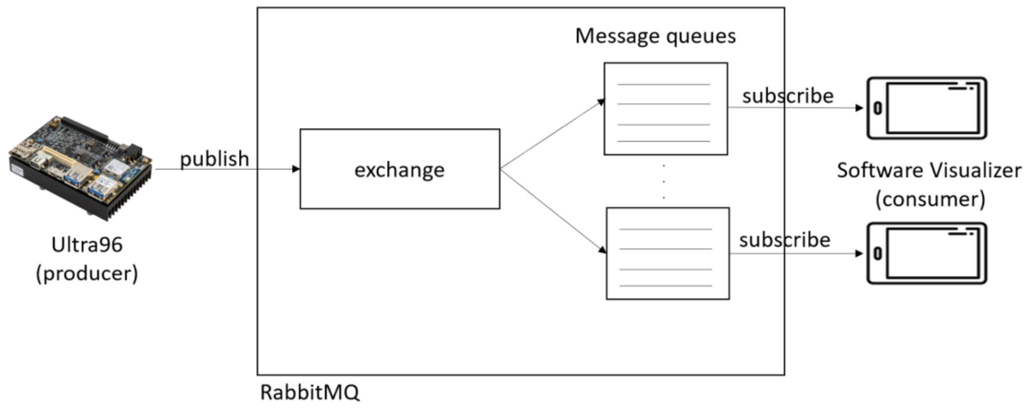


Figure 6.5.2: Data transmission from Ultra96 to Visualizer

Section 7 Software Visualizer

7.1 User Survey and Evaluation

User Survey was created to gather information not only from our capstone group but also from other students as well to better understand the common consensus on the placement of components and the aesthetic design choices as well.

User Survey 1:

The User Survey can be divided into Two Parts

First Part:

This part consists of questions regarding the choice of attributes and components present within the game. From the control of the game to details of what needs to be shown on the player's screen.

Second Part:

This part of the survey consists of questions to understand everyone's preferred placement options for the components needed for the user. This helped greatly in designing the UI/UX for the AR application. The questions present in this section will not be explained separately as all the questions are very similar in nature and they serve the same purpose.

Questions:

1. Let's start with something simple. According to you what components should a player be able to see while playing a Laser Tag game? (Multiple Response Question)
2. Do you want the phone to be mounted on the gun or should be placed on a headset?
3. The design of the Health Bar (Player)
4. The design of the Health Bar (Opponent)
5. The Score Card
6. Ammo Display
7. Health Bar: Player
8. Timer Position
9. Position of Enemy's Status
10. Grenade and Shield Position
11. Should there be a Health Percentage next to the health bar?
12. Should there be a numerical representation of all the variables?

User Survey 2:

This survey was conducted after taking in the inputs from the first user survey. This survey was designed to present the mockup to the target audience to check if their needs were met.

Questions:

1. Does this design match what you were looking for?
2. Is the design of the UI appealing enough to you?
3. Would you recommend this game to your friends just based on aesthetic reasons?
4. Do you have any other improvements? Any suggestions?
5. Overall, how much would you rate it out 10.

Results of the Survey

User Survey 1:

1.

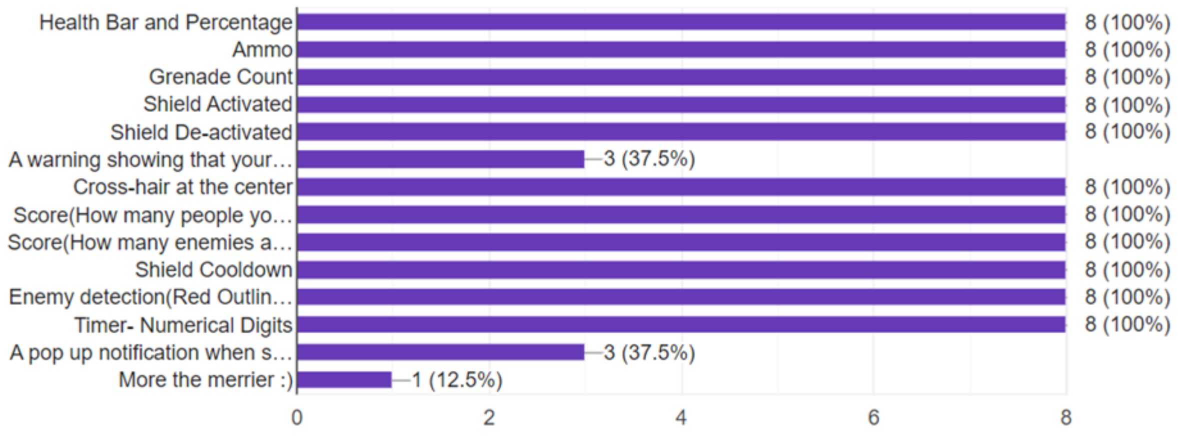


Fig 7.1a: Results for Question 1

2. Mount on the Gun

3.

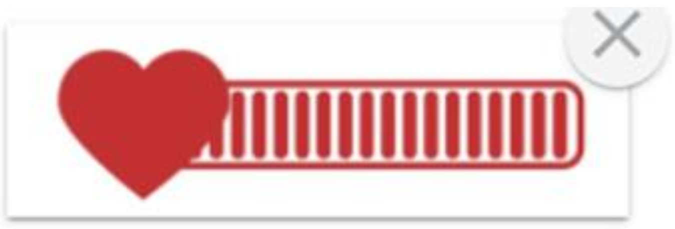


Fig 7.1b: Results for Question 3

4.

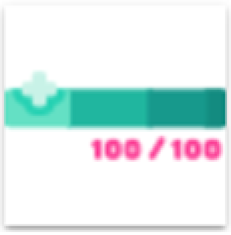


Fig 7.1c: Results for Question 4

5. Top Left

6. Top Right

7. Bottom Left

8. Bottom Middle

9. Top Right

10. On top

11. Bottom Right

12. Yes

User Survey 2:

1. Yes
2. Yes
3. Yes
4. Change the colour of the shield to blue, Increase the size of the scoreboard.
5. 7/8

Since the mock-up was based on the user's needs and wants, they liked the design of the dashboard. However, there were some stylistic issues that had to be resolved.

7.2 Visualizer Design

The Initial Design of our AR Application:

1. **Player's POV:**

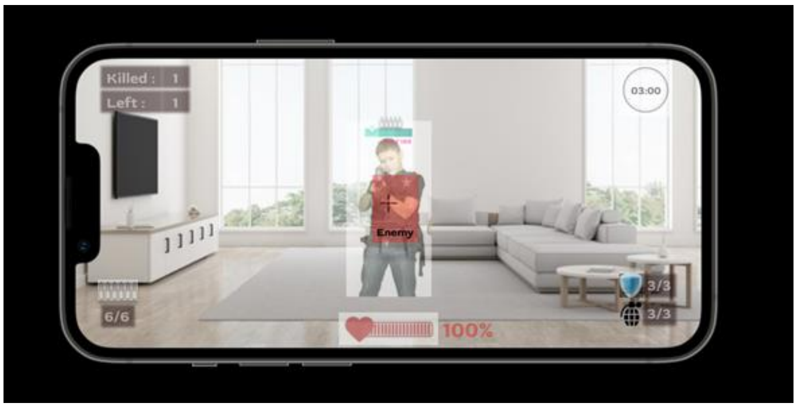


Fig 7.2a: The player's initial screen.

2. **When you activate a shield:**

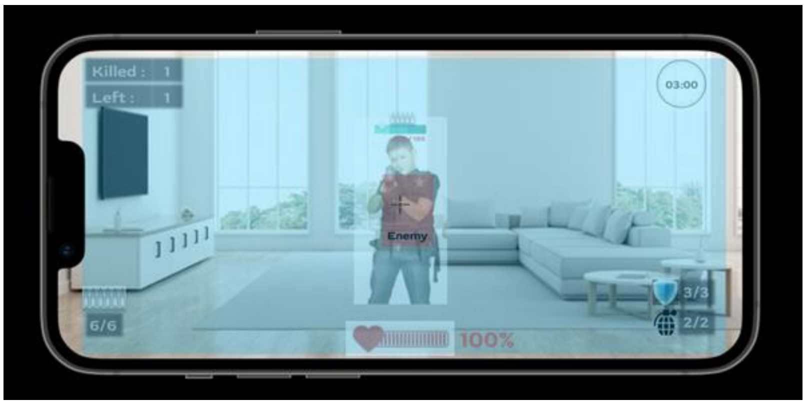


Fig 7.2b: The player's screen when they activate a shield.

3. **When you are hit:**

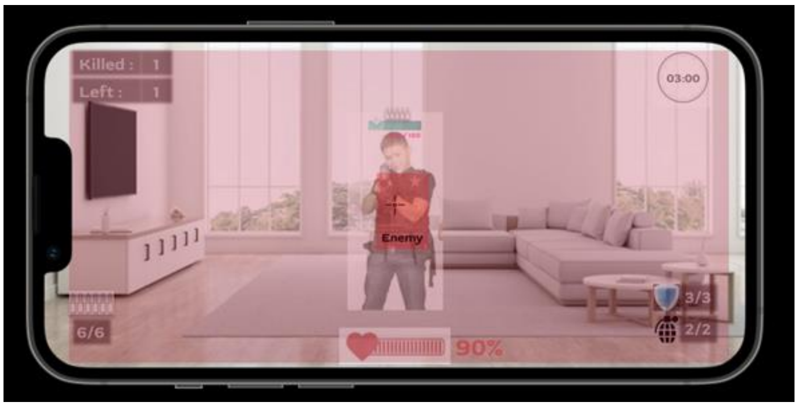


Fig 7.2c: The player's screen when they are in low health.

7.2.1 Data displayed

Health Bar & %	Ammo & No.	Grenade & No.	Shield & No.
Shield Activated/Deactivated	A warning on the screen indicates low Ammo.	Crosshair at the centre	Score: Killed and left
Shield Cooldown	Enemy Detection: Red Box	Timer in Numerical Digits	Enemy's Stats: Health and Ammo

7.2.2 Placement of the Phone Visualiser

Head-mounted:

Pros:	Cons:
Since its head-mounted, the experience is more immersive.	Hardware Limitations. Some smartphones do not function effectively in Google Cardboard.
	Heating up issues. Since the phone will have to render high-resolution graphics the processing power used will also be quite great which in turn increases how the phone is going to be and this is extremely dangerous when the phone is kept at a very close distance from the eyes.
	Compatibility Issues. People wearing glasses might find it extremely uncomfortable to wear a google cardboard set-up. Moreover, wearing it for prolonged periods of time may result in a neck sprain.

	Google cardboard itself has been discounted by google and there are not that many resources in order to refer from when it comes to developing a google VR application

Mounted on the Gun:

Pros:	Cons:
Simulates a real-life shooting scenario where you will investigate the scope while shooting. Mounting the phone on the gun will also provide us with a similar interface.	The experience may not be as immersive as head-mounted.
Phone Compatibility. It is shown to be compatible with all the latest phones.	
Less strenuous on the eyes. Since we will only be looking into it when we perform an action, it won't be detrimental to the eyes.	
Won the User-Survey. This means that the users prefer this more than head-mounted.	

In accordance with the survey results and after carefully considering the pros and cons, we decided that <u>the phone will be mounted on top of the gun to simulate a real-life shooting scenario</u>.

7.2.3 Design Constraints

1. Phone Screen

This plays a major role as every mobile phone has a different display, especially the iPhone as they have curved edges which will lead to some of the components being blocked. Hence, all the components weren’t completely on the side.

2. Ensuring the “keep it simple” rule

Having too many components on the screen at a time can make the application needlessly complicated and will clutter the screen for the user. By adhering to this principle, only the necessary components are shown at a single point in time and different components will be shown based on the actions a user might perform. For example, the timer and the cooldown for the shield will only be shown when the user activates the shield.

3. Onboarding the User

If it’s the player’s first time using an AR application, all the components must not overlap each other, and they should be very intuitive for the user to understand how the game will function. This means that some inspiration will have to be taken from the most popular video games in order to make the user feel more comfortable with the application.

4. User Environment

This is an AR application, which means that it can be played in real life and hence we need to pick a colour scheme that will not be blocked or affected by its surroundings. This affects our vision and sometimes we might have to compromise on some aesthetic options for functionality. Another major concern is safety. The decision to make the components translucent is so that the user can still see the entire environment hence making it less dangerous for them to play in.

5. The AR software

Google AR core is a software development kit developed by Google that allows for augmented reality applications to be built. It is mainly developed for Android applications however it also supports iOS as well but since it has been built keeping android phones in mind, some of the UI elements are not rendered smoothly on the IOS platform. Hence this made us adopt a simple colour scheme and design so that there won’t be any issues on both platforms and will truly be a multi-platform application.

7.3 Visualizer software architecture

7.3.1 Software Frameworks and Libraries

This section deals with the software architecture of the software visualizer, what kind of frameworks are being used and will also deal with how the incoming data from the Ultra 96 will be stored and accessed.

Initial Thought Process

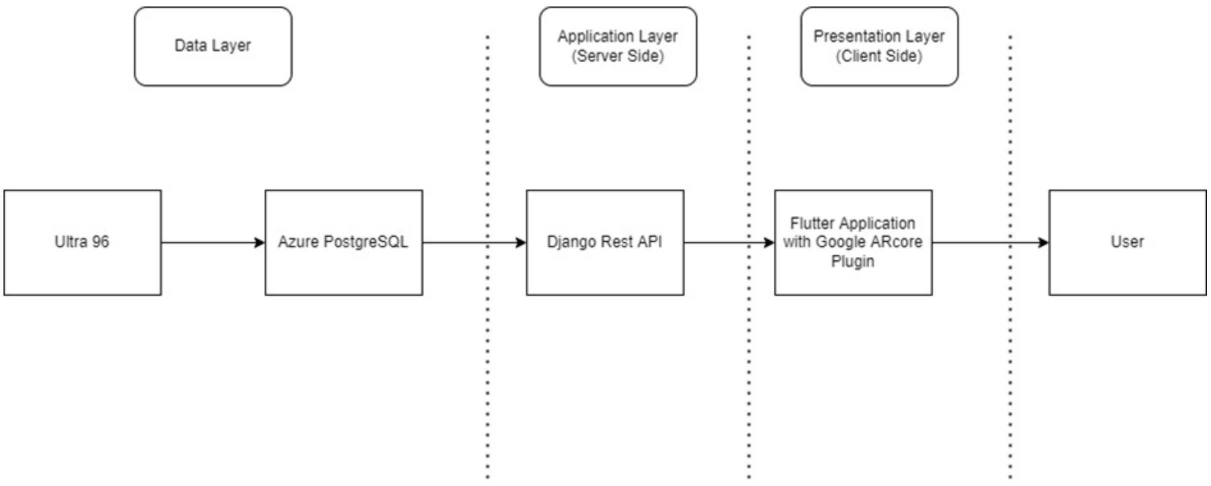

Data Layer
Application Layer (Server Side)
Presentation Layer (Client Side)
Ultra 96
Azure PostgreSQL
Django Rest API
Flutter Application with Google ARcore Plugin
User

Fig 7.3.1a: Software Architecture for Initial Thought Process

As shown in the figure above, the software architecture can be abstracted into three main layers. Application layer, Data layer and Presentation layer. The first layer is the data layer and it here where we get the data from the Ultra 96. To first store the data, we will be using an online database on cloud, Azure Database for Postgre SQL. Using Django, which is a high-level Python web framework, we will be using the Rest Framework to for us to use the RESTful APIs as the flutter application cannot directly access or connect to the Azure PostgreSQL database. Hence, we need to create a RESTAPI to expose data from the database to the mobile application. The last layer would be the presentation layer where the flutter application will take the data from the online database and will present it to the user using the GoogleARCore framework. We are planning to use the polling methodology where, we will, at every frame that is being rendered, to check the values of the data values present in the online database to accurately show the user the updates in real time.

Final Consensus:

We realized that due to the firewall restrictions in place for the ultra96, it may not be feasible for the ultra 96 to connect with the azure database over cloud. There have been some troubles connecting with AWS as well and hence we shifted to using a middleman known as RabbitMQ. We are going to use the AMQP messaging protocol to communicate between the application and the middleman. There will be no changes to the kind of data that will be obtained by the flitter application. It will still get a dictionary with arrays and we will still employ a long polling mechanism to get the data from the queue.

Figma is also being used to design the UI/UX for the Software Visualizer. Figma was used to develop the mock-up resent in this report. We use Figma to quickly build out a prototype to improve and test as well.

We planned to use Flutter with the google ARCore plugin instead of Unity 3D ARCore.

Flutter

Flutter is a free and open-source mobile UI framework created by Google for building beautiful, natively compiled, multi-platform applications from a single codebase. [21]

ARCore plugin

ARCore is Google's platform for building augmented reality experiences. Using different API's, ARCore enables your phone to sense its environment, understand the world and interact with information. [22]

Figma

Figma is a web-based graphics editing and user interface design app. [23]

Django REST API

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. [24]

Django Rest Framework will let us use RESTful APIs to transfer information between an interface and a database in a simple way.[25]

7.3.2 Data Inputs

The following will be the data inputs received from the online Azure Database:

player_type = [F, E]	player_hp = [100, 100]	player_shield_hp = [0, 0]
is_shielded = [False, False]	player_shield_count = [3, 3]	player_grenade = [2, 2]
player_ammo = [6, 6]	player_kill_count = [0, 0]	

This application will be built on Flutter. Flutter is an open-source UI software development kit created by Google. Alongside this, we will be using the Unity 3D ARCore plugin for AR development. The reason why we chose flutter over unity is that while unity is incredibly powerful for games, flutter on the other hand is objectively better for applications because of its libraries and interfaces. Moreover, unity does not have a package manager and its files are comparatively much bigger.

7.3.3 Modules of the Phone Visualiser

Sceneform (1.15.0) library:

Sceneform makes it straightforward to render realistic 3D scenes in AR and non-AR apps, without having to learn OpenGL.

API calls that we would use(non-exhaustive):

1. ARCoreView
2. arCoreController
3. onNodeTap
4. onPlaneTap
5. ArCoreMaterial
6. ArCorShape
7. ArCoreNode

7.4 Phone sensors

ARCore mainly looks at 3 major capabilities to integrate virtual content with the real world as seen through your phones' camera.

Motion Tracking

Using the phone's camera, GoogleARcore will use SLAM, also known as simultaneous localization and mapping to identify distinct points in the given space and will then label them as feature points. These features points are then used to identify the location of the phone and a change in the location. Coupled with the phone's IMU sensor, the framework can estimate the position of the camera relative to the world. The AR objects are then overlayed on these feature points and make it seem as if they are a part of the real world.

Environmental Understanding

ARCore searches for clusters of feature points near common flat surfaces, such as a table or a chair and this, in turn, is shown as geometric planes on our phones.

Light Estimation

Using the camera of the phone, ARCore can detect information about the lighting of its environment based on a camera image. This is paramount to the sense of realism AR provides to the user.

Another major capability of GOOGLE ARCore would be **Depth Understanding**.

To create depth maps, which is basically calculating the distance between surfaces and a given point, this API uses the hardware depth sensor present in a phone such as the TOF sensor. (Time-of-flight) sensor.[26][27]

7.5 To overlay the scoreboard over the camera feed

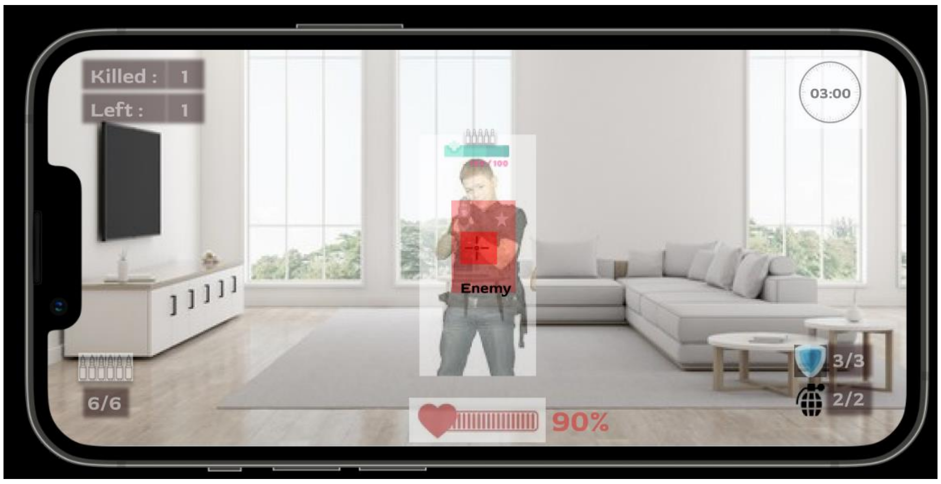
1. We must first enable the camera widget using the flutter application. We can do this by first adding the camera package as a dependency to the flutter application. After which we will have to initilaise the CameraController Widget which will first establish a connection to the device's camera. After which we need allow the phone to display a preview of the phone's camera's feed. We can do this by using the CameraPreview Widget to show the feed.
2. The camera feed is now being shown using the application. The next step would be to overlay the scoreboard over the camera feed. We can do this by using the stack widget and then decreasing the opacity of our scoreboard widget which will make our scoreboard appear in front of the camera application.

7.6 To display the AR of game actions shown on opponents

There are different types of game actions and all of them have different methodologies. The following list is a collection of different types of actions and their respective explanations.

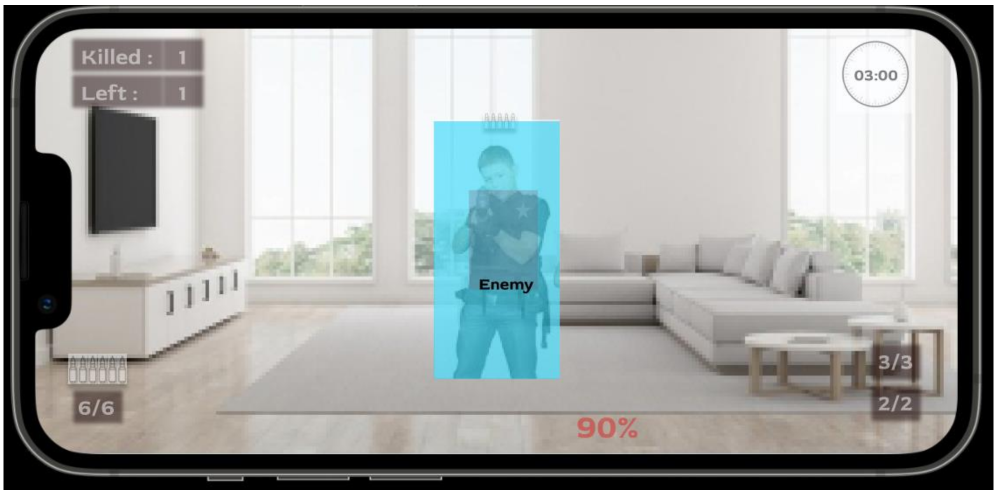
1. The opponent gets hit.

Since we are using a marker-based tracking system, the player will always have access to the opponent's HP and ammo status. Once we have detected that the player has successfully dealt damage then we can see a reduction in health. Moreover, for illustration and for a better experience purpose we will mark the detection with a red crosshair that will be placed in the opponent showing that the player has been hit.



2. The opponent activates a shield.

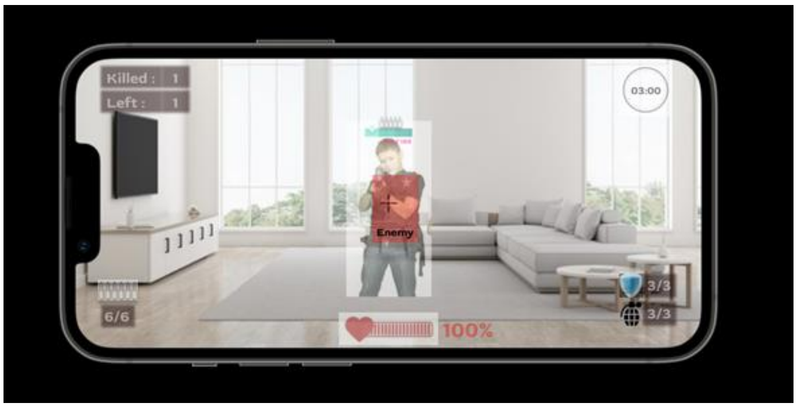
As aforementioned. We will be using a marker-based system and hence the camera will try and detect the QR code, or an image placed in the opponent. Once the camera detects that, the respective AR objects connected.



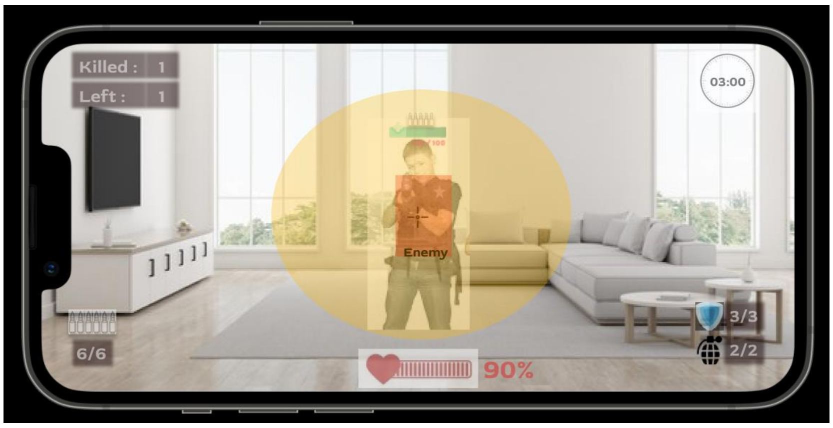
3. The stats of the opponents are shown.

We are using a marker-based tracking module to make this work. We will be placing an image or a QR code on the person’s right shoulder so that the camera can detect it and show the opponents stats on the image as an AR object. We will be placing these QR codes or images all around the player so that regardless of which way the opponent is facing we will still be able to detect that.

The health bar and the ammo will be two different ar objects that will display once the camera scans and find the image.



4. The grenade animation, when it lands on the opponent.



As mentioned in the FAQ, our grenade will be launched with a simple projectile motion which will be released from where the cross hair is located. Using a projectile motion motion, it will land and immediately create a yellow circle which dictates the radius of the explosion.

We have two methods of implementation:

1. The simplest procedure would be that if the player's phone faces the opponent, the grenade will hit the opponent regardless of the splash radius of the grenade.
2. This is a bit more complex as compared to the previous method. We will implement a splash radius and will cross check the splash radius and the location of the oppnonent to check if the opponent has been hit or not. We will need to use the depth API accurately and efficenly to make this work.

Section 8 Project Management Plan

Project Week	Overall Progress	Hardware Sensors	Hardware AI	External Comms	Internal Comms	Software Visualizer
3	Research on project requirements	Research on project requirements	Research on project requirements	Research on project requirements	Research on BLE stack and read project requirements	Research on project requirements
4	Work on initial design report	Continue with research and purchase components needed	Research on different types on neural networks and see which is suitable	Test out eval server and study its algorithm	Test out basic commands and connecting to the beetle	Research on different frameworks out there in order to find the best framework for application development
5	Complete report and prototype individual components	Test out functionality of various sensors used and prepare test codes	Try to generate a bitstream and load it into the FPGA for running	Work on initial design report for submission	Test out sending packets between the Beetles and the Laptop	Work on the initial design report and also try to create a simple application
6	Individual progress checkpoint	Obtain additional equipment if necessary and build one set of equipment	Try to train a dummy model and load it into the FPGA	Test out the various communication between interfaces (ssh tunneling, socket)	Craft out FSM into code for the beetle. Attempt multi-threaded connections	Try to create UI of the application using Flutter onto an android mobile
Recess Week	Continue with individual component	Build 2 additional sets of equipment and improve accuracy of devices	Try training the model with dummy data and load it into the FPGA	Work on L2 game algorithm and finish up communication foundations	Complete the FSM for the laptop and start calibrating values for better results	Try to link the application to the cloud using Azure Database
7	Continue with individual component	Continue to calibrate sensors and fine tune codes. Improve data-processing within Bluno	Continue training the model and address any issues that come up	Test with simple data streams to see if communication is properly established	Limit testing the system and working on improvements, speed/reliability	Try to calibrate the software visualizer and resolve any design and calibration issues
8	Individual subcomponent tests	Prepare for individual test and make sure hardware are functional	Continue training the model and address any issues that come up	Demo and bug fixing	Demo and bug fixing	Demo and Bug testing
9	Integration and testing	Assist with integration and perform system maintenance when necessary	Try training the model with real data and load it into the FPGA	Integration with other members	Integration with other members	Integration with other members and check the latency
10	Complete integration and have complete	Continue to test on integrated system and perform	Collect more data if needed. Continue to	Finish up integration and bug fixing	Testing integrated system and bug fixing	Testing integrated system and bug fixing

	gameplay for one player	maintenance when necessary	improve the inference model.			
11	Single-player test	Prepare for tests by performing complete equipment checks of all parts of the system, and perform maintenance when necessary	Debug accordingly	Prepare for tests and ensure the latest program is running smoothly	Testing and performing calibration tests on the system. Ensuring all beetle software is properly updated.	Prepare for the test and check everything is running smoothly with minimal lag.
12	Double-player test					
13	Final evaluation and shootout					

References

[1] “Bluno_beetle_sku_dfr0339,” DFRobot. [Online]. Available: https://wiki.dfrobot.com/Bluno_Beetle_SKU_DFR0339. [Accessed: 09-Feb-2022].

[2] “MPU-6000 and MPU-6050 product Specification Revision 3 - TDK.” [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. [Accessed: 09-Feb-2022].

[3] “Bluno link - a USB bluetooth 4.0 (BLE) Dongle,” DFRobot. [Online]. Available: https://www.dfrobot.com/product-1220.html. [Accessed: 09-Feb-2022].

[4] K. Townsend, Cufí Carles, Akiba, and R. Davidson, Getting started with Bluetooth Low Energy. Mumbai: SPD, 2015.

[5] “Optimizing current consumption in bluetooth low energy devices,” Optimizing Current Consumption in Bluetooth Low Energy Devices - v2.13 - Bluetooth API Documentation Silicon Labs. [Online]. Available: https://docs.silabs.com/bluetooth/2.13/general/system-and-performance/optimizing-current-consumption-in-bluetooth-low-energy-devices. [Accessed: 09-Feb-2022].

[6] R. Barry, “Using the FreeRTOS Real Time Kernel: Pic32 edition,” Amazon, 2011. [Online]. Available: https://aws.amazon.com/freertos/faqs/#:~:text=and%20session%20semantics.-,Q.,processing%20speed%20and%20%3E64KB%20RAM. [Accessed: 09-Feb-2022].

[7] M. Afaneh, “Bluetooth 5 speed: How to achieve maximum throughput for your ble application,” Novel Bits, 16-Jun-2020. [Online]. Available: https://www.novelbits.io/bluetooth-5-speed-maximum-throughput/. [Accessed: 09-Feb-2022].

[8] P. Momtahan, “Coherent baud rates: Is higher always better?,” www.infinera.com, 15-Sep-2021. [Online]. Available: https://www.infinera.com/blog/coherent-baud-rates-is-higher-always better/tag/optical/#:~:text=Baud%20Rates%3A%20The%20Higher%2C%20The%20Better&text=Baud%20rate%2C%20the%20number%20of,levers%20for%20increasing%20wavelength%20capacity. [Accessed: 09-Feb-2022].

[9] P. R. Narendra, “What is the speed of Bluetooth 4.0? - quora.” [Online]. Available: https://www.quora.com/What-is-the-speed-of-Bluetooth-4-0. [Accessed: 09-Feb-2022].

[10] A. Duque, “Deep dive into Bluetooth LE security,” Medium, 03-Mar-2018. [Online]. Available: https://medium.com/rtone-iot-security/deep-dive-into-bluetooth-le-security-d2301d640bfc. [Accessed: 09-Feb-2022].

[11] B. Daniel, “What is AES encryption? [the definitive Q&A guide],” Trusted Computing Innovator. [Online]. Available: https://www.trentonsystems.com/blog/aes-encryption-your-faqs-answered#:~:text=Out%20of%20128%2Dbit%2C%20192,is%20technically%20the%20least%20secure. [Accessed: 09-Feb-2022].

[12] Real Python, “Socket programming in python (guide),” Real Python, 05-Jun-2021. [Online]. Available: https://realpython.com/python-sockets/. [Accessed: 09-Feb-2022].

[13] C. Bernstein and M. Cobb, “What is the Advanced Encryption Standard (AES)? definition from searchsecurity,” SearchSecurity, 24-Sep-2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard. [Accessed: 09-Feb-2022].

[14] R. Franklin and P. Editor, “AES vs. RSA Encryption: What are the differences?,” Precisely, 13-May-2021. [Online]. Available: https://www.precisely.com/blog/data-security/aes-vs-rsa-encryption-differences. [Accessed: 09-Feb-2022].

[15] “Sshtunnel,” PyPI. [Online]. Available: https://pypi.org/project/sshtunnel/. [Accessed: 09-Feb-2022].

[16] A. Astori, “Concurrency and parallelism in Python,” Medium, 23-Apr-2021. [Online]. Available: https://towardsdatascience.com/concurrency-and-parallelism-in-python-bbd7af8c6625. [Accessed: 09-Feb-2022].

[17] “About,” PostgreSQL. [Online]. Available: https://www.postgresql.org/about/. [Accessed: 09-Feb-2022].

[18] D. Taylor, “What is mongodb? introduction, architecture, features & example,” Guru99, 01-Jan-2022. [Online]. Available: https://www.guru99.com/what-is-mongodb.html#5. [Accessed: 09-Feb-2022].

[19] “What can rabbitmq do for you?,” RabbitMQ. [Online]. Available: https://www.rabbitmq.com/features.html. [Accessed: 09-Feb-2022].

[20] “AMQP vs MQTT: Comparing instant messaging protocols,” RSS, 13-Aug-2021. [Online]. Available: https://www.cometchat.com/blog/amqp-vs-mqtt-comparing-instant-messaging-protocols. [Accessed: 09-Feb-2022].

[21] Django. [Online]. Available: https://www.djangoproject.com/. [Accessed: 09-Feb-2022].

[22] “Fundamental concepts | arcore | google developers,” Google. [Online]. Available: https://developers.google.com/ar/develop/fundamentals#motion_tracking. [Accessed: 09-Feb-2022].

[23] “Overview of arcore and supported development environments | google developers,” Google. [Online]. Available: https://developers.google.com/ar/develop. [Accessed: 09-Feb-2022].

[24] R. Chen, “Your 3-Minute Guide to Augmented Reality (AR): How does it work?,” Digital Marketing Agency in Singapore. [Online]. Available: https://www.constructdigital.com/insight/how-does-augmented-reality-ar-work. [Accessed: 09-Feb-2022].

[25] Sunilagarwal, “Azure database for postgresql documentation,” Microsoft Docs. [Online]. Available: https://docs.microsoft.com/en-us/azure/postgresql/#:~:text=Azure%20Database%20for%20PostgreSQL%20is,high%20availability%2C%20and%20dynamic%20scalability. [Accessed: 09-Feb-2022].

[26] “What is Figma? (and how to use Figma for beginners),” Theme Junkie, 14-Sep-2020. [Online]. Available: https://www.theme-junkie.com/what-is-figma/. [Accessed: 09-Feb-2022].

[27] “Build apps for any screen,” Flutter. [Online]. Available: https://flutter.dev/. [Accessed: 09-Feb-2022].