Department of Electronic and Telecommunication Engineering
University of Moratuwa, Sri Lanka

VR Kayaking

Supervisors:

Prof.(Mrs.) Dileeka Dias
Dr. Kasun T. Hemachadra

Members:	Group 26
T.I.R. De Zoysa	200115K
G.I. Deshapriya	200118X
R.A.R.L.Ranasinghe	200511V
R.D.P.M.Ranasinghe	200512B

Final Year Project Mid Review Report
submitted in partial fulfillment of the requirements for the course module EN4203

July 31, 2025

Approval

Approval of the project supervisor(s).

Supervisor: Prof.(Mrs.) Dileeka Dias

Signature: . . . . . .

Date: . . . . . .

Supervisor: Dr. Kasun T. Hemachadra

Signature: . . . . . .

Date: . . . . . .

Declaration

We declare that the dissertation titled VR Kayakingand the work presented in it are our own. We confirm that:

- this work was done wholly or mainly in candidature for a B.Sc. Engineering degree at this university,
- where any part of this dissertation has previously been submitted for a degree or any other qualification at this university or any other institute, has been clearly stated,
- where we have consulted the published work of others, is always clearly attributed,
- where we have quoted from the work of others, the source is always given.
- with the exception of such quotations, this dissertation is entirely our own work,
- we have acknowledged all main sources of help,
- parts of this dissertation have been published.

2025/07/01.....
Date

T.I.R. De Zoysa (200115K)

G.I. Deshapriya (200118X)

R.A.R.L.Ranasinghe
(200511V)

R.D.P.M.Ranasinghe
(200512B)

Acknowledgement

We would like to express my sincere gratitude to everyone who supported and guided me throughout the development of this Virtual Reality (VR) Kayaking project.First and foremost, We am deeply thankful to my academic supervisors and lecturers at the Department of Electronic and Telecommunication Engineering, University of Moratuwa, for their invaluable guidance, encouragement, and technical insights. Their expert advice and continuous support were instrumental in shaping the direction of this project.

We also extend my heartfelt appreciation to my project group members and peers for their collaboration, knowledge sharing, and moral support during challenging times. Special thanks go to the laboratory and technical staff for facilitating resources and assistance in hardware testing and system integration.Furthermore, We am grateful to the University of Moratuwa for providing a strong academic environment and access to research facilities, which made this work possible.

Abstract

The VR Kayaking project bridges the gap between real-world physical activity and virtual immersion by developing a virtual simulator using Unity game engine and a physical kayaking platform that overcomes the limitations of traditional outdoor kayaking, such as safety risks and weather constraints. Using Bluetooth Low Energy (BLE) for seamless communication, the system maps real-time paddle and platform movements tracked via IMU sensors and encoders to the virtual environment. For easier user interaction, voice commands can be given via mic, enabling hands-free control over gameplay. This is achieved through the integration of Meta Voice SDK, which converts speech to text. Then, to accurately classify the meaning of the speech input, we use a BERT NLP model trained to categorize the text into predefined classes. The system supports two modes: competitive mode and free mode. In competitive mode, a leaderboard and a score system have developed, and the user competes against AI opponents trained using the ML-Agents Toolkit, utilizing the Proximal Policy Optimization (PPO) algorithm. Free mode plays real 360 kayaking videos as the background, where the water portion has been removed using a instance segmentation model. To enhance the realism of the kayaking experience, we combine both real and virtual worlds in the environment. For water detection and removal, an instance segmentation model is used and further trained to improve accuracy in identifying water regions within the 360° videos. Additionally, a voice chatbot has been developed for real-time interaction within the free mode.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The VR kayaking project is designed to make kayaking more accessible and convenient, allowing users to experience the sport from the comfort of their homes. This innovative approach not only promotes fitness and well-being but also offers a safe training environ ment, facilitates social interaction, and raises awareness about environmental issues. By providing an immersive and realistic kayaking experience, the project aims to bridge the gap between traditional outdoor activities and modern technological advancements.

1.2 Problem statement

In contemporary society, health, fitness, and weight-related issues are increasingly preva lent. Outdoor kayaking, while beneficial, presents several challenges including safety concerns, comfort issues, and a substantial time commitment. Furthermore, there is a lack of infrastructure to support competitive virtual kayaking, which limits opportunities for enthusiasts and professionals alike. Addressing these problems requires a solution that integrates convenience with high-quality user experiences

1.3 Primary objectives

- **Safer and more comfortable environments**: Offers a secure and comfortable way to kayak indoors, eliminating the risks of outdoor kayaking, such as accidents or bad weather.
- **Immersive Experience**: Provides a fully engaging kayaking experience with realistic visuals and water effects, making users feel like they are actually on the water.
- **Competitive Kayaking**: Features single-player mode with challenges against virtual kayaking avatars(bot players), adding an exciting competitive element to the experience.
- **Different Environments**: Allows users to explore a variety of kayaking maps and upload their own real-world environments for playing.
- **Realistic Water Dynamics with Variable Resistance**: Simulates realistic water resistance by the kayaking ergometer and water physics using a virtual river in real time, enhancing the kayaking experience.

1.4 Project scope

The scope of this project includes the development of a 360◦ video-based virtual kayaking simulator equipped with resistive feedback mechanisms. A machine learning model will be utilized to replace the river in the 360◦ video with a dynamically created virtual river, incorporating realistic water physics in real time. The project will also involve real time user gesture tracking via image processing, creating a competitive environment with kayaking races and avatars, and building a system for map selection and uploading new maps.

1.5 Novelty and the uniqueness of the project

- **Immersive Experience**: Theproject provides a deeply engaging kayaking experience by combining real-world videos with virtual elements, making users feel like they're truly on the water.
- **Real World and Virtual Environment**: Unlike commercial platforms that use only artificial environments, our system com bines real-world video with virtual objects, offering a more realistic and varied kayaking experience.
- **User Gesture Tracking**: Instead of using VR controllers, we track user movements by image processing via a camera, making the experience more natural and hands-free.
- **Competitive Mode**: While commercial platforms often focus on multiplayer, our project includes a single-player competitive mode where users can challenge themselves against virtual bot players. So, they don't need any real player.
- **User Friendly**: Our system is designed to be easy to use(for navigate through the User Interface (UI)) without the need for VR controllers, making it simpler for users to navigate and enjoy the kayaking experience.

1.6 Beneficiaries and potential applications

The project benefits a diverse range of stakeholders including kayaking enthusiasts, train ers, the gaming industry, educational institutions, and research teams. It presents sig nificant business opportunities for fitness centers and gaming hubs, particularly in Sri Lanka, and caters to users of all ages. Socially, the project reduces risks associated with outdoor kayaking, offers varied environmental experiences, and combines exercise with entertainment, making it a valuable addition to both recreational and professional settings. In this chapter, we discussed the project's motivation, problem statement, primary objectives, and scope. We also explored the novelty and uniqueness of our project, high lighting its innovative aspects. Additionally, we identified the beneficiaries and potential applications, emphasizing how the project will impact various stakeholders and real-world scenarios.

Chapter 2

Literature Review

In this literature review, we will explore various aspects of the VR kayaking project by ex amining current implementations and contrasting them with our innovative approach. We will delve into the available platform features, comparing commercial and do-it-yourself (DIY) options, and evaluate two primary design strategies for developing the platform. Additionally, we'll discuss mechanisms for enhancing kayaking motions in two stages, and assess alternative strategies for virtual environment creation, preprocessing, development platforms, and controlling boards. Financial and social impacts of the project will also be reviewed to highlight its comprehensive benefits.

2.1 Current Implementation Vs Our approach

Feature	**Kayak VR Mirage**	**Whitewater VR**	**MarineVerse Kayaking**	**Virtual Kayaking 1.0**
Immersive Experience	Available	Available	Available	Available
Kayaking Environment	Artificial	Artificial	Artificial	Real World (Artificial River)
Platform	Not Available	Not Available	Not Available	Available
Competitive Mode	Not Available	Not Available	Not Available	Available

Table 2.1: Comparison of Different VR Kayaking Applications

Virtual Kayaking 1.0 stands out by providing a highly immersive experience with real time water dynamics. Unlike other platforms, it features a single-player competitive mode, allowing users to enjoy kayaking alone. We integrate a real-world 360◦ video with a virtually created river for realistic water physics, and we design both a physical kayaking platform and a VR environment, setting a new standard in the VR kayaking experience.

2.2 Suggested Platform Design Options

We have two options: modifying a commercial ergometer by adding Bluetooth and sensors for lower mechanical complexity, or designing a custom ergometer from scratch, which is cost-effective but requires more effort and expertise.

2.2.1 Option 1- Purchase & Modify

One of the options is to purchase a commercial platform that doesn't have Bluetooth support but offers a lower cost. Then, We will modify this platform by integrating a Bluetooth system for communication, adding encoders to measure ergometer speed, and incorporating mechanisms for free motion. This approach reduces mechanical complexity since the base ergometer mechanism is already available, but incurs a higher cost due to the purchase price of the platform. Modification time is low to moderate, and the required expertise is moderate to high.

Figure 2.1: Low cost Commercial Available Platform

2.2.2 Option 2- Build & Modify

Our second option is to design and build our own kayaking ergometer based on research [1], [2], [3] and web resources [4], [5] like Figure 2.6 ergometer. We will integrate a Bluetooth system, add encoders to measure speed, and develop mechanisms for free motion. While this option requires constructing the entire ergometer mechanism from scratch, which involves low to moderate complexity, it results in a lower cost as we avoid purchasing a commercial platform. The time required for building the system is moderate, and the expertise needed is low to moderate compared to modifying a purchased platform.

Figure 2.2: DIY Ergometer [1]

2.3 Evaluation of Alternative Strategies

VIRTUAL ENVIRONMENT CREATION	Single Video	360°	Multiple Video	360°	Single Video with Virtual River
MODEL FOR VIDEO PRE-PROCESSING		Pre-Trained		Custom-Trained	
VIRTUAL ENVIRONMENT DEVELOPMENT PLATFORM	Unity		Unreal Engine		Godot
CONTROLLING BOARD	Arduino	Raspberry Pi			Jetson Nano

Table 2.2: Evaluation of Alternative Strategies

Note: The options highlighted in green represent the choices selected for our project based on their suitability and effectiveness.

- **Virtual Environment Creation**
  We chose a single 360° video with a virtual river (replace the real river with a virtually created river in real-time) to provide real-time water dynamics feedback and simplify the creation of left-right rotation kayaking motions, enhancing user experience and interactivity. This design concept was inspired by a conference paper.

- **Model for Video Preprocessing**
  To accurately detect river boundaries for virtual river creation, we selected a pre-trained machine learning model due to its high performance, accuracy, and ease of use, ensuring effective video preprocessing.

- **Virtual Environment Development Platform**
  Unity was chosen for its strong community support, ease of use, and suitability for VR applications, facilitating efficient development of immersive virtual kayaking environments.

- **Main Controlling Board**
  We selected Raspberry Pi for its high computational power, ease of use, Bluetooth support, and cost-effectiveness, ideal for handling communication and image processing tasks with the VR system and camera.

2.4 Financial Aspects

The VR kayaking project offers significant financial opportunities by targeting fitness centers and gaming hubs, which can incorporate the technology into their offerings. By promoting the project at tourist sites in Sri Lanka, it can attract both local and international visitors, enhancing its market reach. Additionally, the product appeals to a broad customer base without age limitations, making it suitable for various demographic segments. The focus on single-user experiences capitalizes on the growing market for virtual kayaking, potentially increasing revenue and market penetration.

2.5 Social Aspects

The VR kayaking project addresses social concerns by reducing the risks associated with outdoor kayaking, providing a safer alternative for enthusiasts. It allows users to experience diverse virtual environments, promoting environmental awareness and engagement. Furthermore, it combines exercise with entertainment, encouraging physical activity while ensuring an enjoyable experience. This dual benefit enhances overall well-being and social interaction, making it a valuable addition to both recreational and fitness activities.

This literature review analyzes the VR kayaking project by comparing existing solutions with our approach. We evaluate platform features, contrasting commercial and DIY options, and outline two design strategies for the platform. The review details mechanical design and simulation stages, from basic force models to advanced optimization techniques. It also assesses alternative strategies for creating virtual environments, including different development platforms, to highlight the project's potential impact and suitability.

Chapter 3

Methodology

3.1 Introduction to the Methodology

The methodology chapter explains how to complete the VR kayaking project. It starts by describing the project architecture, identifies the requirements needed, and discusses the system design in terms of integrating major components. Additionally, it mentions the materials and tools needed during the implementation process to provide an understanding of the project's technical details.

3.2 Project Approach

3.2.1 Project Architecture

The VR kayaking system consists of two main components.platform and loacal computer. Platform data, including orientation and wheel speed, transmits to an ESP32 using I2C communication. The ESP32 sends this data to the VR headset over Bluetooth. Communication between the Local Computer and the VR headset operates bidirectionally, allowing for interactive data exchanges, while other data flows are unidirectional to maintain efficiency. Figure 1 illustrates the system's architecture and component interactions.

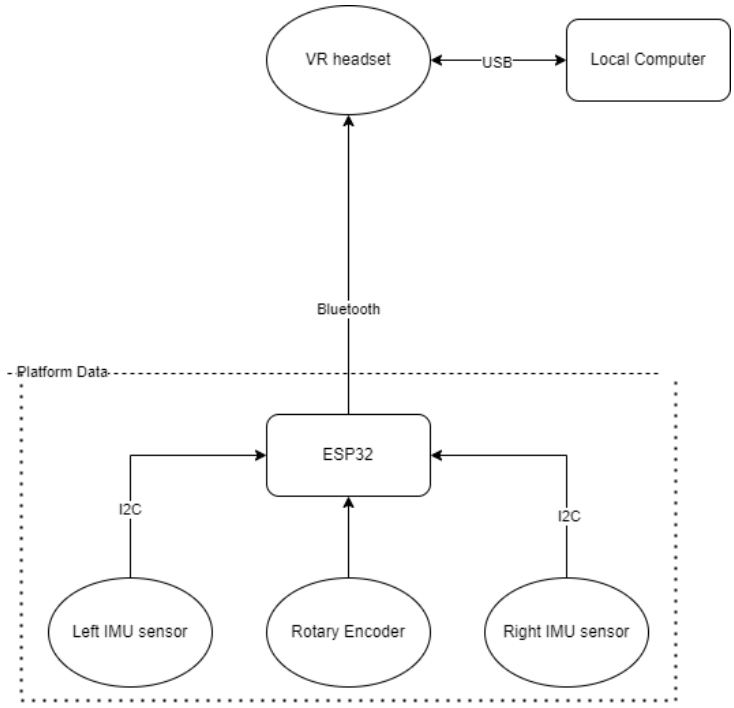


Figure 3.1: System Architecture Diagram

The architecture of the project consists of several key components:

- **IMU:** Provides orientation data of the platform to ESP32 board via I2C.
- **Rotary Encoder:** Measures the speed of the wheel and sends this data to the ESP32.
- **ESP32:** Receives encoder data, and IMU data transfers it to the VR headset via Bluetooth.
- **Local Computer:**Video prepossessing part run in it.
- **VR Headset:** Visualizes the environment to the user based on the processed data from local computer.

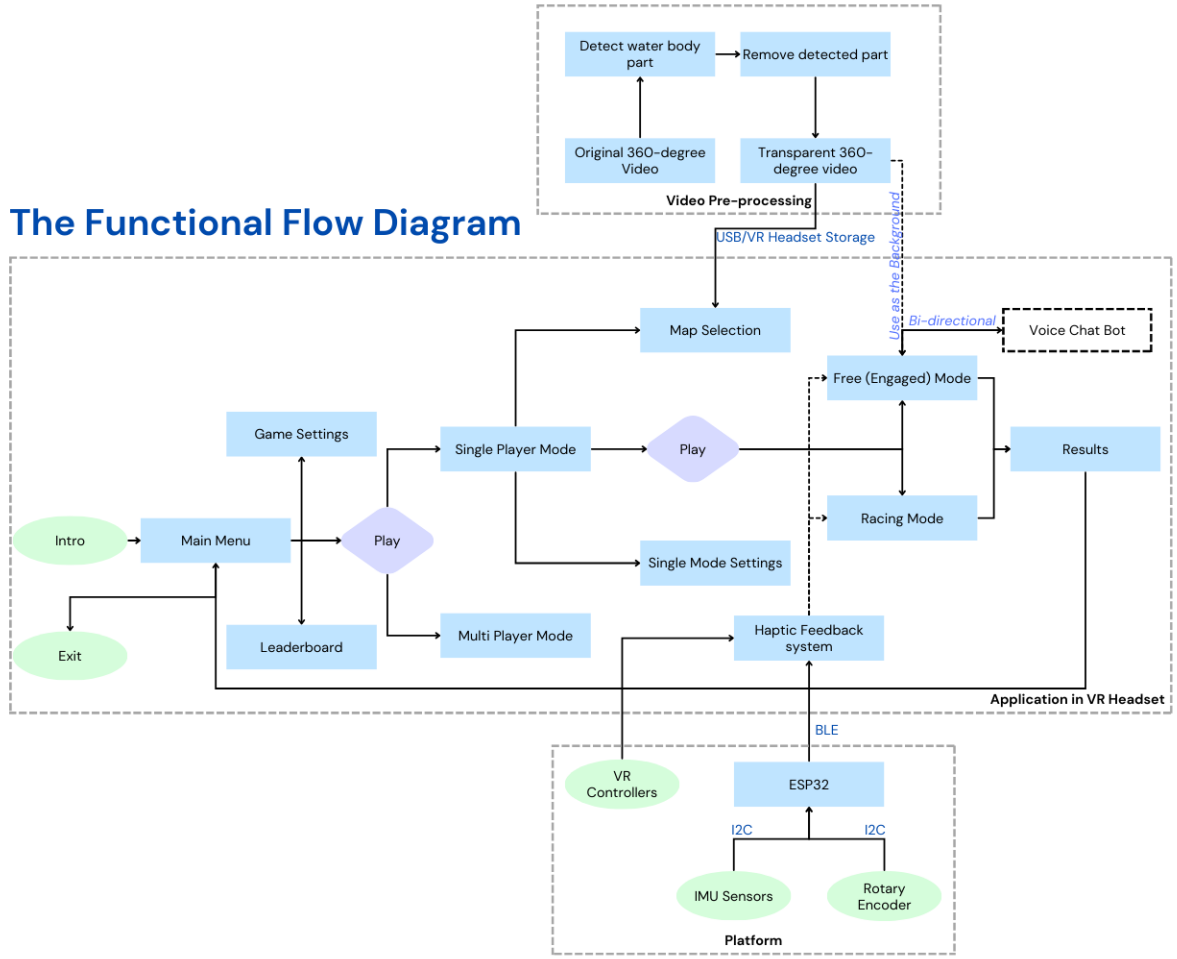


Figure 3.2: The Functional Flow Diagram

3.2.2 Platform and Communication

Previous related projects on VR kayaking [6] failed to incorporate a physical kayaking platform, we use an accessorily designed platform that is intended to provide more realism and immersion to its end-user. The platform shall permit the simulation of natural movement in kayaking, for example, an ergometer mechanism providing resistance as if one is paddling through water-natural movements. With mounted sensors such as IMUs and rotary encoders, it will track the orientation and wheel speed with precision. This information feeds the VR system and ensures that all movements on the platform will be replicated in the virtual environment.

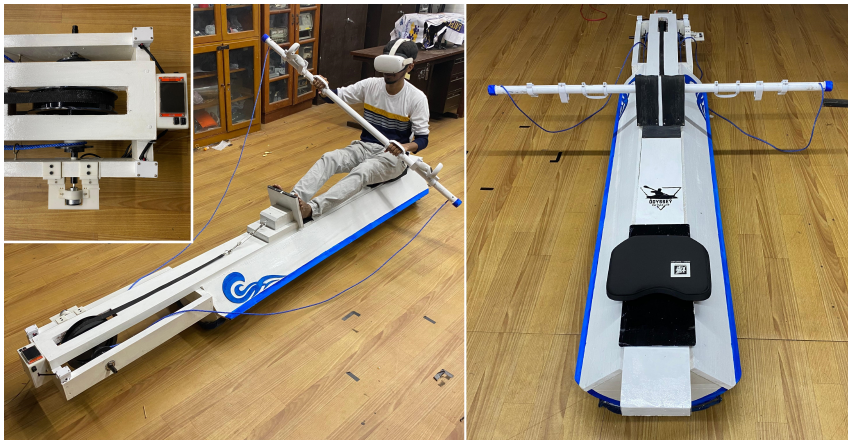


Figure 3.3: Our Kayak platform: It includes an ergometer mechanism, which is made using pulleys and a wooden structure.

In our project, we want to map the physical kayaking platform motion to the unity side. Then we use IMU sensors and rotary encoder data to map. Therefore we want to transfer the sensor data and encoder data to the unity side. Then we use Bluetooth as a communication protocol because Bluetooth has some key advantages for our VR kayaking project. Like Low Power Consumption, Real-Time Data Transmission, Security, Ease Of implementation, Flexibility, and Scalability.

a) Platform data transmitted via Bluetooth

First, we create the BLE server on the ESP32, the device collects data from the sensors and the encoder processes it, and then formats it into packets. These packets, containing information such as orientation and speed, transmit to the VR headset via the established BLE connection.This real-time data transfer makes sure that the VR environment matches the user's movements.

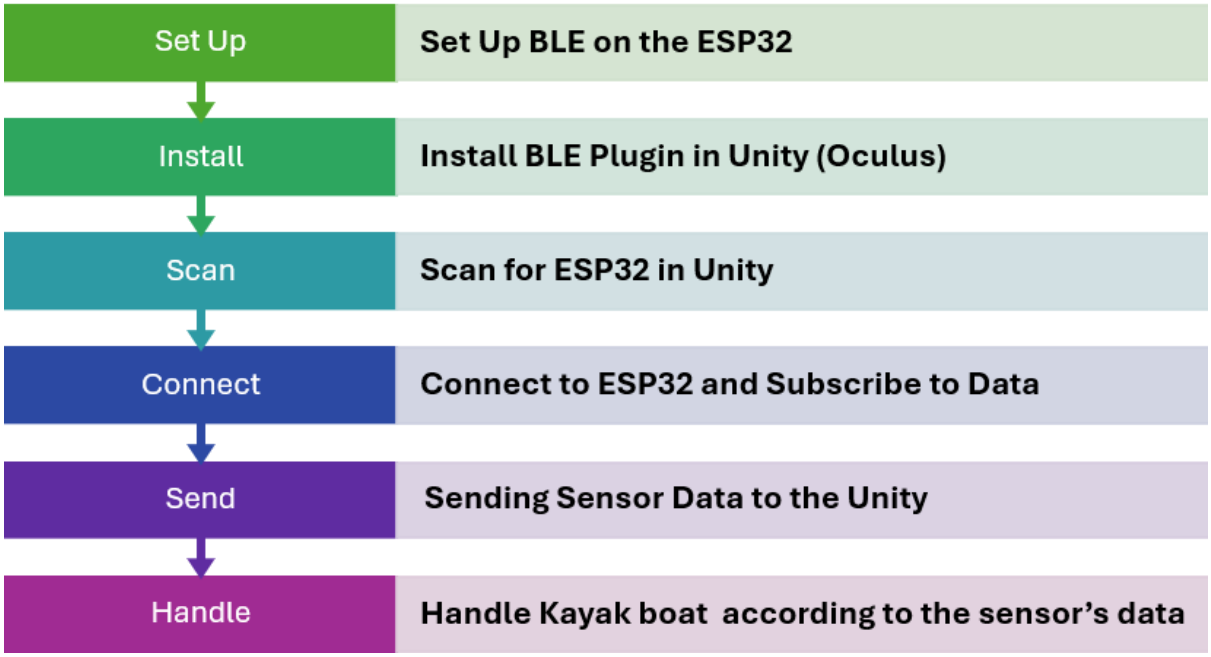


Figure 3.4: Block diagram for mapping platform using IMU sensors and Rotary encoder via Bluetooth

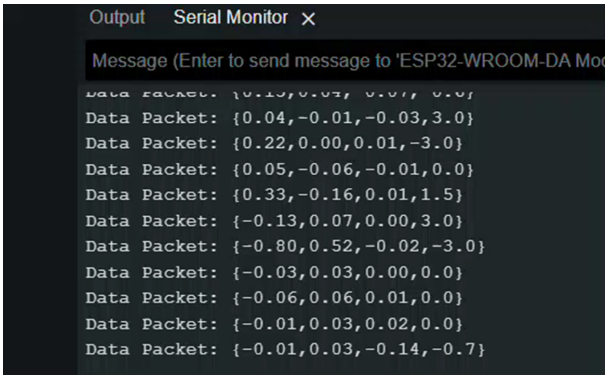


Figure 3.5: Our platform data packet: These packets contain information such as orientation and speed. The first three values represent the x, y, and z orientation data, and the final value represents the speed data.

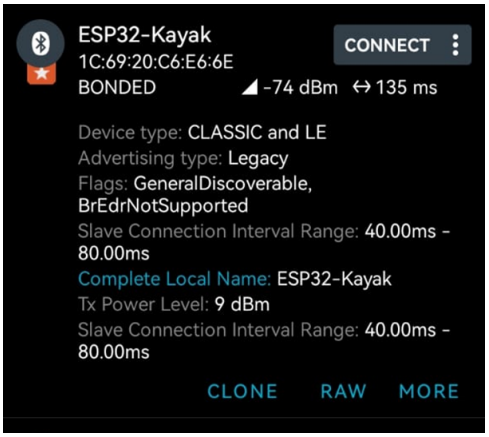


Figure 3.6: Our BLE server information:latency is 135ms and slave connection interval range is 40ms-80ms

In Bluetooth communication, typical latency varies depending on factors such as connection interval settings and environmental conditions. For good Bluetooth communication, latency usually ranges from 4 to 20 milliseconds. Our VR kayaking system has real-time data transfer, then we also want good Bluetooth communication. In more applications, latency might be higher like it can range 20 to 50 milliseconds. That is also manageable for real-time data transmission. If our latency exceeds 50 milliseconds then it potentially impacts our user experience in the VR environment. So we must manage our latency at least 50 to 75 milliseconds.

In Bluetooth communication, the RSSI plays a very crucial role. It holds the signal power level of the received signal at our Bluetooth server. While we cannot visually observe the Bluetooth connection quality between the VR headset and the ESP32 used in our VR kayaking case, we need to track RSSI values to assess the overall connection quality and its reliability [7]. A normal Bluetooth signal, for that matter a strong Bluetooth signal normally measures in the area of 0dBm. Nevertheless, Bluetooth has a good signal strength of -30dBm also a bad signal strength of -100dB RSSI values. To minimize packet loss and ensure low latency, we need to keep the RSSI healthy and above -70 dB. We can detect potential issues such as interference and range limitations by monitoring changes in RSSI, which may affect the stability of real-time data transmission and the quality of the VR. In this case we use the client specifically the nrf connect app to get the RSSI values. After, we change the position of the client from the Server by moving it closer or further away.

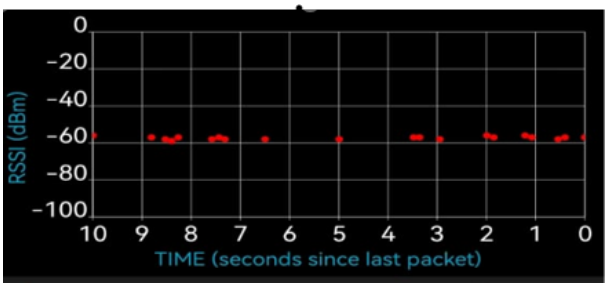


Figure 3.7: RSSI variations for the 0–50 cm. It shows how the RSSI varies regularly between -60 dB to -40 dB, showing good strength with less attenuation .

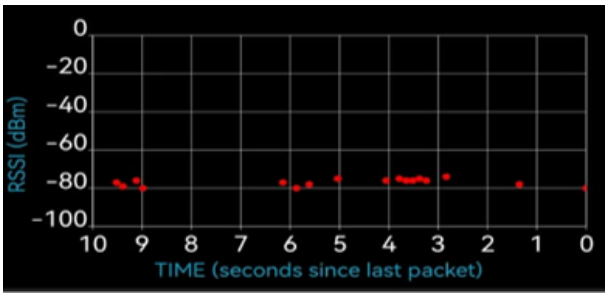


Figure 3.8: RSSI variations for the 100–150 cm range.It shows how the RSSI varies regularly between -75 dB to -85 dB. The increased distance introduces moderate signal degradation.

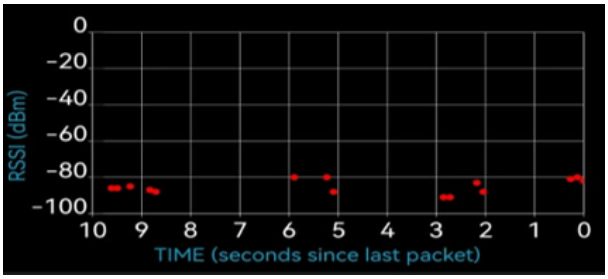


Figure 3.9: RSSI variations for the 150–200 cm .It shows how the RSSI varies regularly between -75 dB to -100 dB. the fact clearly reveals that increased distances have huge impacts on signal strength and stability.

b) Bluetooth Delay Calculation

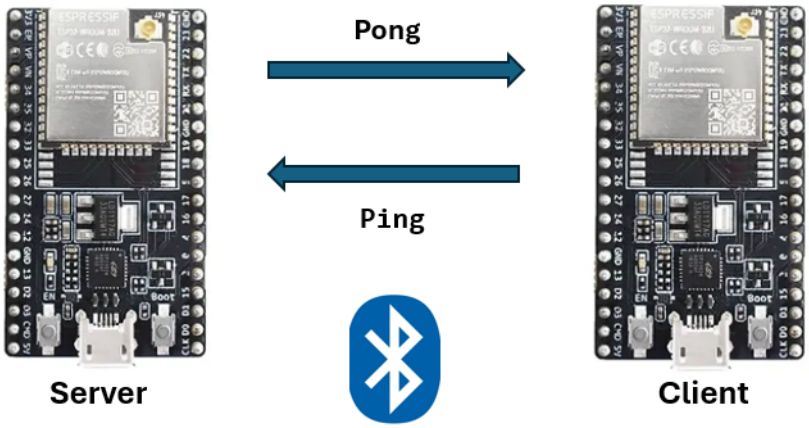


Figure 3.10: Bluetooth delay analysis in the VR kayaking system conducted using two ESP32 modules, configured as a client and a server.

We calculated the delay in two phases: ping-pong communication and data packet transmission. In the ping-pong phase, the client sent a "ping" message to the server every 100 milliseconds, and the server responded with a "pong" message.We considered it as packet loss if the RTT exceeded 40 milliseconds or if there was no response.

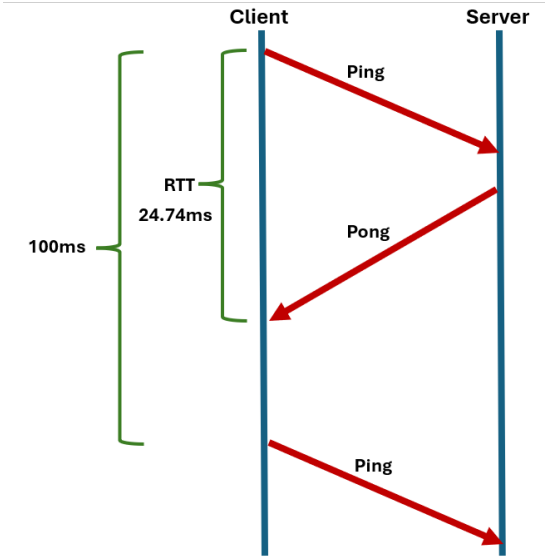


Figure 3.11: Ping and Pong data packet communication:The one-way delay averaged 12.37 ms in this phase, which is lower than the usual latency and causes negligible overhead in this phase.

During the second phase, the client sent packets with values like "2.3, 4.5, 5.0, 30.0" to the server. From the analysis, we found that the average one-way delay for the transmission of data packets was about 23.99 ms.

We compared these findings with the literature on Bluetooth latencies concerning Virtual Reality applications. We found that previous works on latency requirements for interactive VR systems suggest the range of 20 to 50 ms for one-way transmission.; delays greater than this reduce the feeling of immersion and responsiveness. The one-way delay observed in the ping-pong phase was 12.37 ms and 23.99 ms during data transmission, which is within this acceptable range, hence the system is suitable for VR applications. Also, we calculated a time difference between data coming to the unity side and motion start using that sensor data time which value was equal to 0.632ms. Also we calculated a total delay of

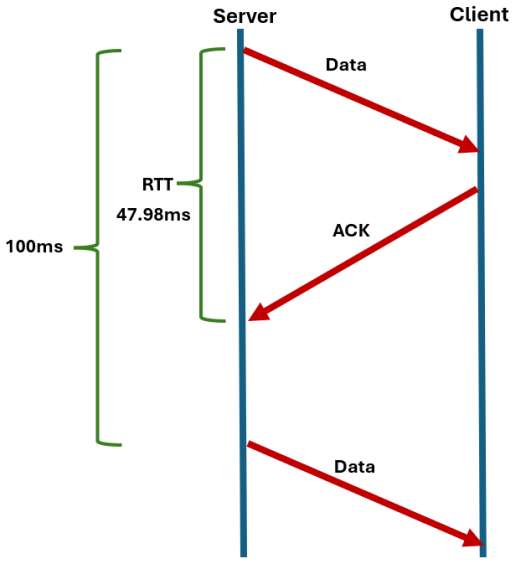


Figure 3.12: Data packet communication:The one-way delay averaged 23.99 ms in this phase.This delay is higher than that obtained in the ping-pong phase due to the higher size of the payload and the extra processing.

24,622ms. That delay was in the acceptable range in VR applications. Furthermore, we confirm reliable communication through negligible packet loss and steady RTTs, which is an essential requirement to keep up the synchronization in VR environments. These findings form a great basis for Bluetooth as one of the viable communication methods in low-latency real-time VR systems.

Sensor Integration for Platform Motion and Speed Detection

The VR kayaking system utilizes two key sensors to capture user motion: the ICM20948 Inertial Measurement Unit (IMU) and a 600 pulses-per-revolution (PPR) incremental rotary encoder. The ICM20948 is a 9-axis sensor that provides accurate orientation data by combining a 3-axis gyroscope, accelerometer, and magnetometer. This data is processed to determine the tilt angles of the platform, particularly for detecting left and right leaning motions, which are used to simulate kayak steering in the virtual environment.

In addition, a high-resolution 600 PPR incremental rotary encoder is mechanically connected to the flywheel of the kayaking ergometer. As the user paddles, the encoder generates pulses proportional to the rotation speed, which are counted to compute real-time paddling speed. This speed data directly influences the forward motion of the virtual kayak. The combination of orientation and speed sensing enables an immersive and responsive user experience that closely mimics real kayaking dynamics.

Calibrating IMU Angle Accuracy with Complementary Filter

To estimate the rolling motion (tilt around the X-axis) of the kayaking platform, the ICM20948 9-axis IMU is used. The raw data from the gyroscope and accelerometer are fused using a complementary filter to provide stable and responsive orientation measurements. The gyroscope offers high-frequency angular velocity data, while the accelerometer provides a long-term reference based on gravity. Each sensor has limitations — gyroscopes suffer from drift, and accelerometers are sensitive to noise and external acceleration — so combining them enhances accuracy.

Before applying the filter, the IMU undergoes calibration to remove gyroscope biases and ensure correct accelerometer scaling. The resulting roll angle enables the system to accurately detect left and right tilting, which corresponds to the user's steering motion in the virtual kayaking experience.

The **complementary filter** equation used to estimate the roll angle (θ) is:

$$\theta = \alpha(\theta + \omega_{\text{gyro}} \cdot \Delta t) + (1 - \alpha) \cdot \theta_{\text{acc}}$$

Where:

- θ is the estimated roll angle,
- ω_{gyro} is the angular velocity from the gyroscope (around the X-axis),
- Δt is the time step between measurements,
- θ_{acc} is the angle calculated from the accelerometer using:

$$\theta_{\text{acc}} = \arctan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$

- α is the filter constant (typically 0.95–0.98), balancing gyro responsiveness and accelerometer stability.

Determining Optimal Alpha Value for Complementary Filter

o achieve accurate and stable roll angle estimation from the ICM20948 IMU, the alpha value (α) of the complementary filter was experimentally tuned. The IMU sensor was placed on a flat, stationary surface, where the true roll angle is known to be approximately zero degrees. This condition was ideal for analyzing the effect of different alpha values on angle accuracy.

The complementary filter was tested with α values ranging from 0.95 to 0.99, and the estimated roll angles were recorded over time. For each value, the error between the calculated angle and the expected angle (zero) was computed. A graph was plotted showing alpha values on the X-axis and the corresponding mean absolute error on the Y-axis.

The results showed that as α increased, the gyroscope contribution dominated more, leading to reduced responsiveness to accelerometer corrections. At lower alpha values (e.g., 0.95), the filter responded better to noise but introduced instability. The optimal trade-off was observed at $\alpha = 0.98$, which minimized drift while maintaining stability and responsiveness. This value was selected for final implementation in the real-time complementary filter used for roll angle estimation during kayaking movements.

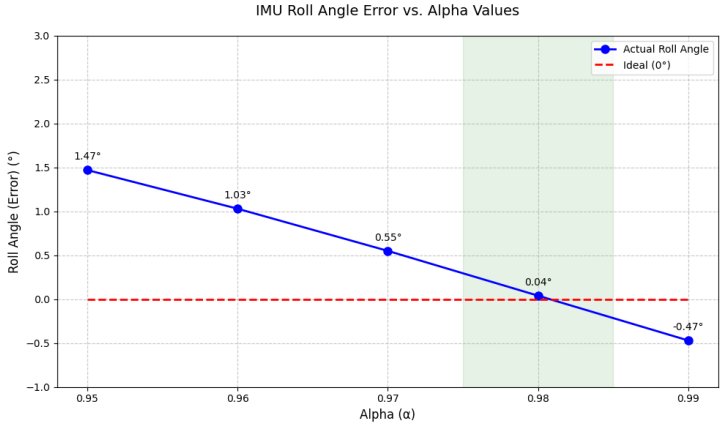


Figure 3.13: α vs Error angle

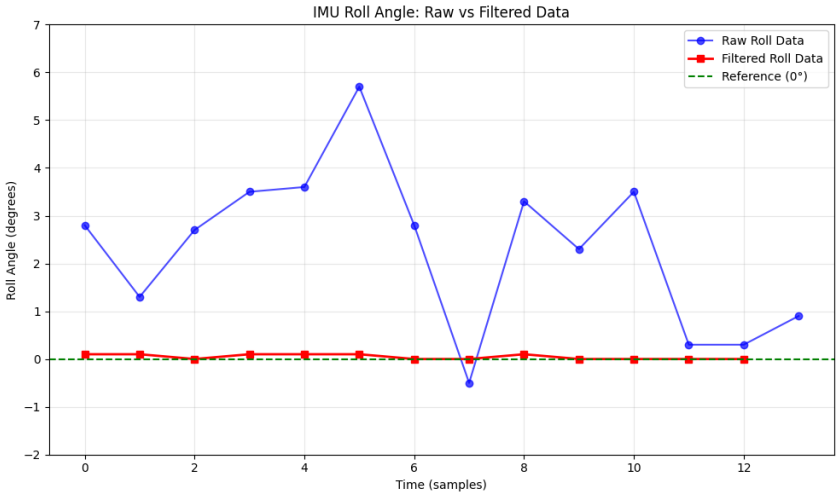


Figure 3.14: IMU Roll Angle: Raw vs Filtered Data

Dual Power Supply System with Automatic Switching Using P-Channel MOSFETs

To ensure uninterrupted operation of the VR kayaking system, a dual power supply mechanism was implemented. The system primarily operates on 230V AC mains power, which is stepped down and regulated using a Hi-Link HLK-PM01 5V power module. In the event of an AC failure, the system automatically switches to a 3.7V Li-ion (18650) battery, boosted to 5V using a DC-DC converter. This automatic switching is achieved using P-channel MOSFETs (IRF9Z34N), which act as high-side switches controlled by the gate-source voltage difference.

When AC power is available, the output of the Hi-Link module provides 5V to the load and also to the gate of the battery-side MOSFET, keeping it OFF ($V_{GS} = 0V$). Simultaneously, the AC-side MOSFET has its gate pulled to 0V, and source at 5V, making $V_{GS} = -5V$, which turns the MOSFET ON and powers the system from AC.

If AC power fails, the Hi-Link output drops to 0V. Now, the gate of the battery-side MOSFET becomes 0V while its source remains at 5V from the battery boost converter, giving $V_{GS} = -5V$, which turns it ON. The AC-side MOSFET now has both gate and source at 0V, making $V_{GS} = 0V$, and thus it turns OFF. This ensures the system seamlessly switches to battery power without any manual intervention.

The IRF9Z34N MOSFET was selected for its low $R_{DS(on)}$, high current handling, and ability to function efficiently in high-side switching applications. This design ensures reliable power delivery, low voltage drop, and efficient energy management for portable and emergency use scenarios in the VR system.

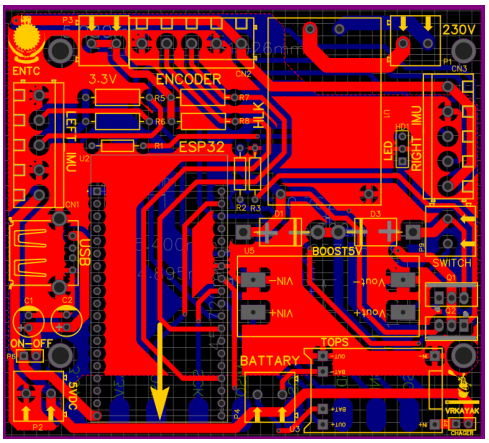


Figure 3.15: Power PCB

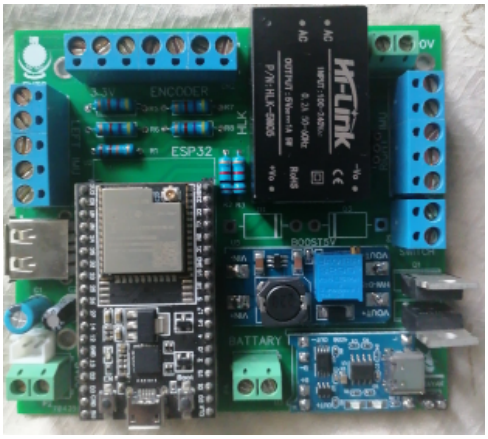


Figure 3.16: Soldered PCB

3.2.3 Unity Development

In the Unity Development process, we have focused to brige the gap between modern user needs and the traditional VR systems in main key areas.

a) ML-Agents Training in VR Kayaking Races

Reinforcement Learning for Autonomous Kayak Racing

In virtual kayaking races, there are Machine Learning (ML) Agents trained by ML-Agents Toolkit in Unity as bot players in races. As demonstrated by [8], the Unity ML-Agents Toolkit is an open-source plugin to the Unity engine, facilitating reinforcement learning (RL) for Artificial Intelligence (AI) behavior training.

The toolkit includes three key components such as agents, environments and actions as in Figure 3.17. ML agents interact with the environment to collect observations, execute actions, and based on their performance, receive rewards. The attached Agent script assigns rewards and observations while Behavior Parameter script behave as the brain of the agent and communicates rewards and observations to the trainer through python API. We designed virtual racing tracks as training environments for our VR kayaking system, similar to a kart racing setup [9]. So, we use a pre-trained model and again trained for a kayaking virtual setup. Actually, we trained 24 karts in a virtual environmnet as in Figure 3.18. Then, we attached the trained kart object at the bottom of the kayak object, so kayak agents move and rotate relative to the kart object. But kayak object also rotates itself according to the water system dynamics.

In reinforcement learning (RL) for kayak racing, agents (a_i) navigate the track over discrete time steps ($t = 1, 2, ..., T$). An agent takes an action ($a_{i,t}$) such as acceleration, braking, or steering at each time step to transition to a new state (s_{t+1}) while providing a reward ($r_{i,t}$). The goal is to learn a policy $\pi_i(a_{i,t} \mid s_t)$ for evaluating state-value ($V_{\pi_i}(s_t)$)

$$V_{\pi_i}(s_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k r_{i,t+k} \mid s_t \right],$$

and action-value ($Q_{\pi_i}(s_t, a_{i,t})$) functions

$$Q_{\pi_i}(s_t, a_{i,t}) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k r_{i,t+k} \mid s_t, a_{i,t} \right].$$

to minimize time to the finish line while avoiding collisions and penalties. A policy is a probability distribution over actions, that maximizes cumulative rewards:

$$J(\pi_i) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t r_{i,t} \right].$$

This model uses the Multi-Agent Proximal Policy Optimization (MA-PPO) algorithm, which is an extension of PPO, to find the policy parameter θ_i that maximizes the cumulative reward over time.

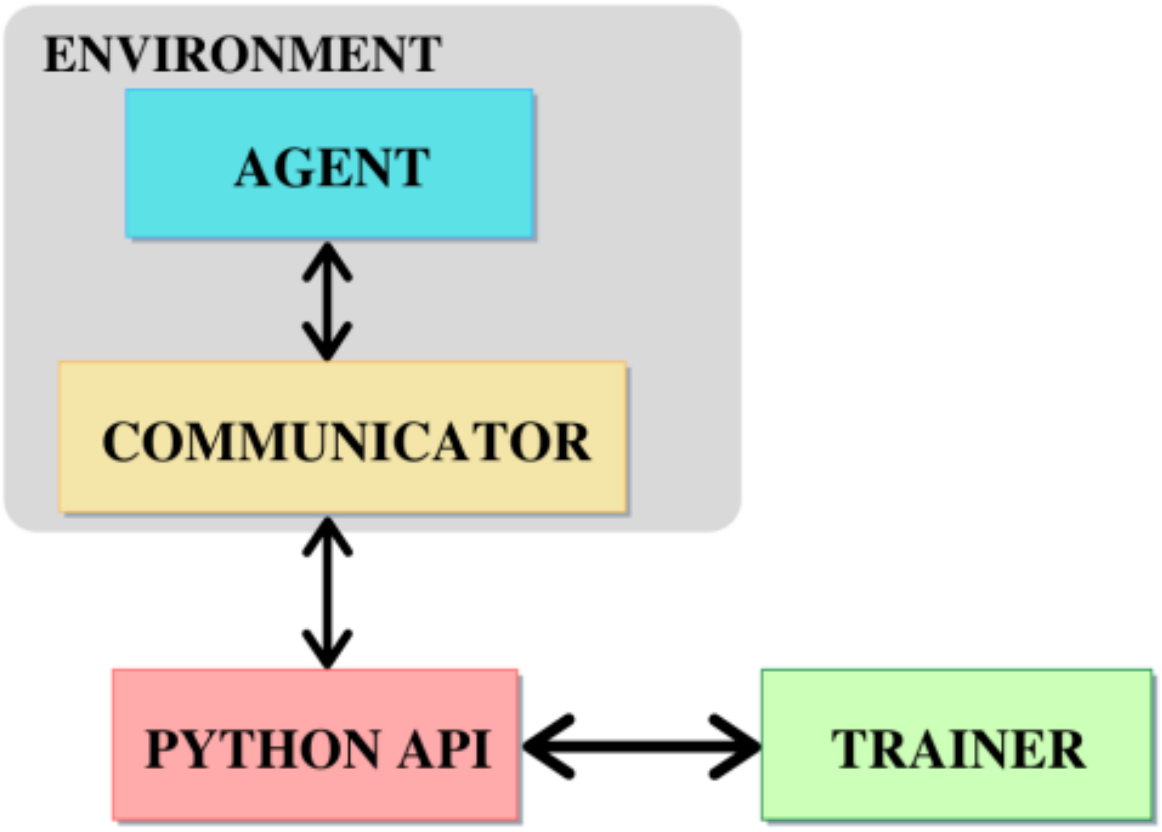


Figure 3.17: There are 3 main components in the ML-agents toolkit: agent, environments and actions. The policy of each agents must access the neural network (NN) for sharing the experiences in the training. They use a communicator for connecting with the trainer through Python API.

MA-PPO: Multi-Agent Proximal Policy Optimization

Multi-Agent Proximal Policy Optimization (MA-PPO) is an extension of the standard Proximal Policy Optimization (PPO) algorithm designed to train multiple agents in a shared environment. PPO is a reinforcement learning algorithm known for its stability and simplicity, utilizing clipped objective functions to maintain a balance between exploration and exploitation while preventing excessively large policy updates.

MA-PPO enhances PPO to handle decentralized policies for multiple agents while considering their interactions within the same environment. This makes it highly suitable for tasks like autonomous vehicle coordination, robotic swarm behavior, and multiplayer game simulations. Each agent maintains its own policy and value network but interacts with other agents through the shared state and reward dynamics of the environment.

In MA-PPO, agents aim to optimize their own policy by maximizing the expected cumulative reward. The algorithm utilizes trajectories collected during agent-environment interactions and applies advantage estimation and surrogate policy gradient optimization using clipped ratios. The process iteratively refines policies to improve cooperative or competitive performance.

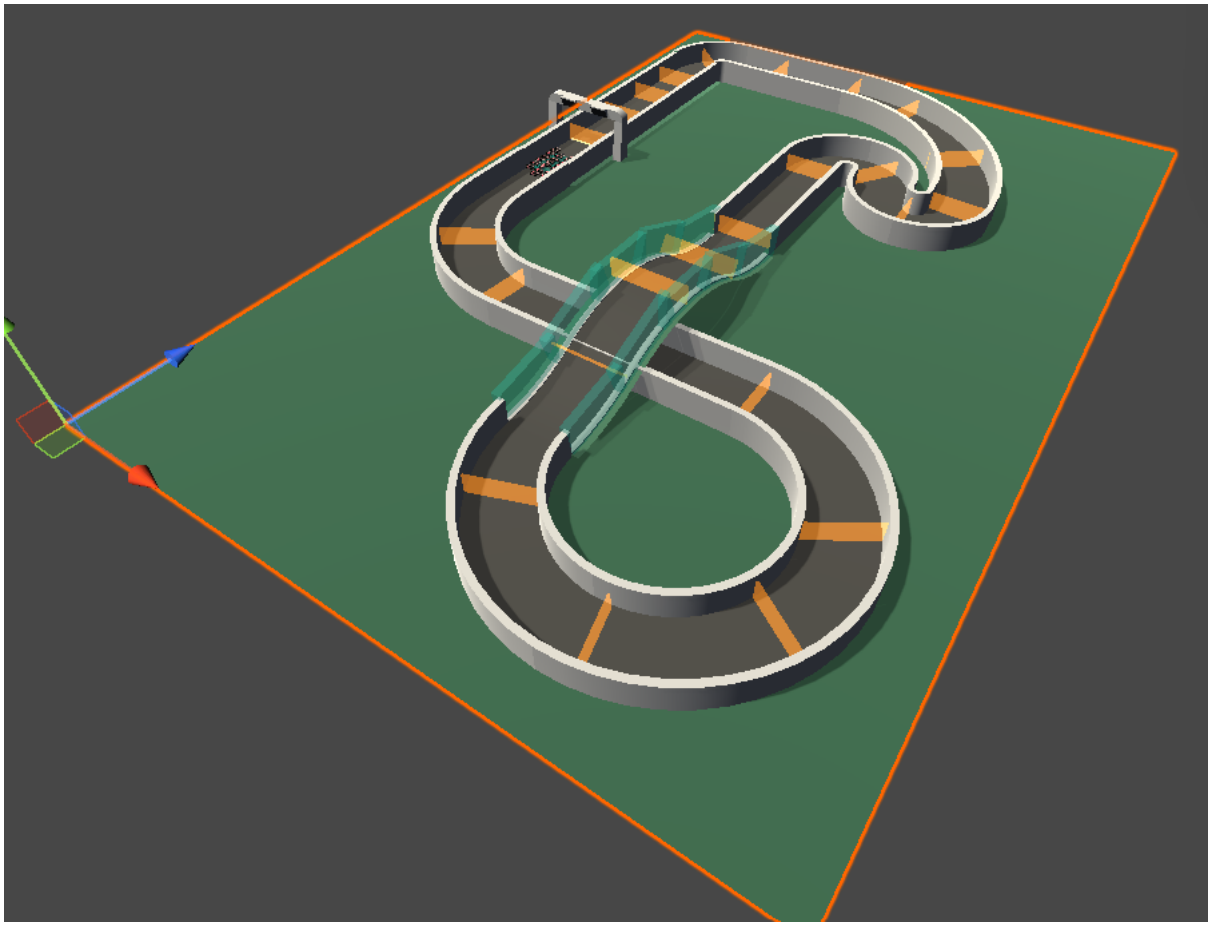


Figure 3.18: The Virtual Training Environment which has various type of junctions, heights and checkpoints

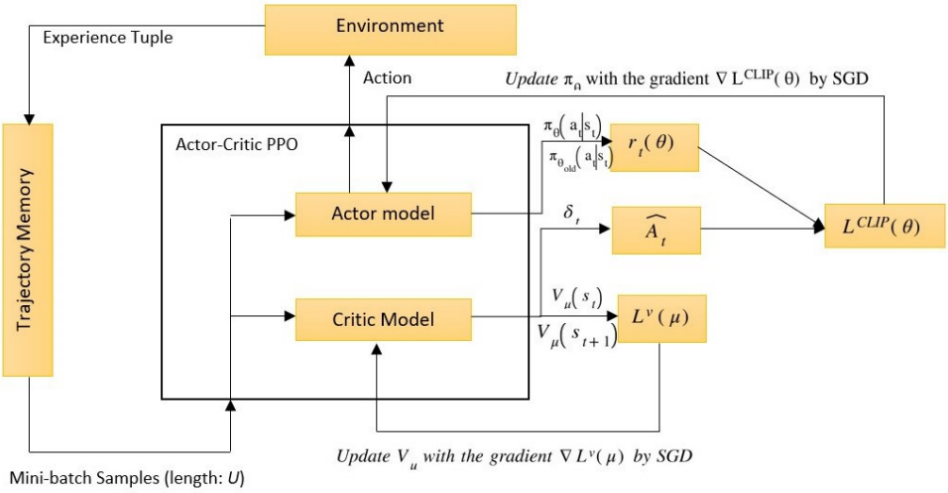


Figure 3.19: MAPPO workflow diagram

Below is a step-by-step breakdown of the MA-PPO algorithm, including its mathematical formulation:

1. **Initialize Networks**
   For each agent $a_i \in A = \{a_1, a_2, \ldots, a_n\}$:
   - Initialize a policy network $\pi_{\theta_i}(a_{i,t}|s_t)$
   - Initialize a value network $V_{\theta_i}(s_t)$

- Set old policy parameters $\theta_i^{\text{old}} \leftarrow \theta_i$

2. **Collect Trajectories**
   For each agent a_i, and for each timestep $t = 0$ to $T - 1$:
   - Sample an action: $a_{i,t} \sim \pi_{\theta_i}(a_{i,t}|s_t)$
   - Execute $a_{i,t}$, observe reward $r_{i,t}$ and next state s_{t+1}
   - Store transition tuple $(s_t, a_{i,t}, r_{i,t}, s_{t+1})$ in τ_i
3. **Compute Returns and Advantages**
   The expected cumulative reward (return) is given by:

$$J(\pi_i) = \mathbb{E}_{\pi_i}\left[\sum_{t=0}^{T-1} \gamma^t r_{i,t}\right]$$

   Use Generalized Advantage Estimation (GAE) or Temporal Difference (TD) method to compute advantages A_t.
4. **Compute Likelihood Ratio**
   Compute the likelihood ratio between the current and old policy:

$$r_t(\theta_i) = \frac{\pi_{\theta_i}(a_{i,t}|s_t)}{\pi_{\theta_i^{\text{old}}}(a_{i,t}|s_t)}$$

5. **Compute PPO Surrogate Objective**
   The clipped surrogate loss is defined as:

$$L^{\text{MA-PPO}}(\theta_i) = \mathbb{E}_{\tau\sim\pi_i}\left[\min\left(r_t(\theta_i)A_t,\ \text{clip}(r_t(\theta_i), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

6. **Update Policy Parameters**
   Perform gradient ascent on the clipped objective:

$$\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} L^{\text{MA-PPO}}(\theta_i)$$

7. **Update Value Function**
   Update the value network using temporal difference learning:

$$V_\theta(s_t) \leftarrow V_\theta(s_t) + \alpha\left(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)\right)$$

8. **Repeat**
   Repeat steps 2–7 for multiple episodes or until convergence.

Reward Mechanism of the RL Process

The following steps [9] explains how end the episodes and when add rewards and punishments.

- **Hit Penalty:** When the agent collides with any obstacle or wall, it receives a punishment of -1 to discourage reckless movement.
- **Checkpoint Reward:** Passing through a checkpoint grants the agent a reward of 1, encouraging progress along the defined route.
- **Towards Checkpoint Reward:** Moving closer to the next checkpoint earns a small positive reward of 0.03 per step, guiding the agent in the correct direction.
- **Speed Reward:** The agent receives a reward of 0.02 proportional to its forward speed, promoting fast traversal.
- **Acceleration Reward:** No direct reward is given for acceleration, i.e., the reward is 0. This ensures that acceleration alone doesn't influence learning unless it contributes to reaching goals more efficiently.

b) Sensor Feedback Mechanism

We have attached the two VR controllers at the end point of the physical paddle as in the paper [10] and assigned Left and Right Controller positional object to KayakPaddleController script. Then it calculates the midpoint between the left and right VR controllers to position the paddle's center, adding an initial offset stored at the start. For taking proper directional orientation of virtual paddle, it takes the vector between the left and right controllers. Then it calculates the rotation values from each axis by Vector product between the normalized difference vector and x, y and z unit vectors and apply this rotation to the paddle mid point. This process update in real-time for precise paddle movement.

We attach a IMU sensor at the midpoint of the physical kayak platform and fix the rotary encoder align with ergometer to touch the rotary wheels each other. Since, attached IMU sensors to the platform detect real kayak platform rotations, as well as the encoder measures the ergometer rotating speed by enabling Haptics Feedback [11], [12], they synchronize with Unity's virtual kayak rotating motions and top speed of the Kayak respectively. Also, our developed communication system utilizes sensor data received from an ESP32 via Bluetooth Low energy (BLE).

c) Mathematical Analysis and Modeling of Paddle Motion in Virtual Kayaking Simulation

Step 01: VR Kayak Paddle Mapping Using Two Controllers

Algorithm 1 VR Kayak Paddle Mapping Using Two Controllers

procedure START
  Compute the **center's position** between both hand controllers:
    `handsMidpoint` ← (`leftHand.position` + `rightHand.position`) / 2
  Compute the initial offset between the virtual paddle center and the center's position of the hand controllers.
    `initialOffset` ← `paddleCenter.position` - `handsMidpoint`
end procedure
procedure UPDATE
  Compute the **center's position** between both hand controllers:
    `handsMidpoint` ← (`leftHand.position` + `rightHand.position`) / 2
  Update the **virtual paddle position**:
    `paddleCenter.position` ← `handsMidpoint` + `initialOffset`
  Compute the **direction vector** of the hand controllers:
    `handDirection` ← (`rightHand.position` - `leftHand.position`)
  Normalize `handDirection`.
  Compute the **rotation** for the virtual paddle using **LookRotation** function and apply:
    `paddleCenter.rotation` ← LookRotation(`normalizedHandDirection`, `Vector3.up`)
end procedure

In virtual reality (VR) simulations, precise mapping of user interactions is essential for immersion. This paper focuses on modeling a VR kayaking paddle controlled by two hand-held controllers, ensuring natural paddle positioning and orientation.

i) Paddle Position Calculation

The paddle's position is determined using the midpoint between the left and right hand controllers. At the **START** (1) and (2) are calculated, then at each frame **UPDATE** (3) to (7).

Midpoint Computation

Let $\mathbf{P}_{LC}$ and $\mathbf{P}_{RC}$ represent the positions of the left and right controllers, respectively. The center's position of the hand controllers is given by:

$$\mathbf{P}_{\text{midC}} = \frac{\mathbf{P}_{LC} + \mathbf{P}_{RC}}{2} \tag{3.1}$$

To maintain a consistent offset from the initial position, an **initial offset** $\mathbf{O}_{\text{init}}$ is computed at the start:

$$\mathbf{O}_{\text{init}} = \mathbf{P}_{\text{midP}} - \mathbf{P}_{\text{midC}} \tag{3.2}$$

where $\mathbf{P}_{\text{midP}}$ represents the center's position of the virtual paddle.

Thus, after (1) and (2) are calculated, the following equations (3) to (7) are executed at each frame. Compute the center's position of the hand controllers same as in **START**:

$$\mathbf{P}_{\text{midC}} = \frac{\mathbf{P}_{LC} + \mathbf{P}_{RC}}{2} \tag{3.3}$$

and the updated virtual paddle position is:

$$\mathbf{P}_{\text{midP}} = \mathbf{P}_{\text{midC}} + \mathbf{O}_{\text{init}} \tag{3.4}$$

ii) Paddle Orientation Calculation

The paddle's orientation must align with the user's hand direction. This is achieved using **quaternion transformations**.

Hand Direction Vector

The primary direction of the hand controllers is defined as the vector between the left and right controllers:

$$\mathbf{D}_{\text{C}} = \mathbf{P}_{RC} - \mathbf{P}_{LC} \tag{3.5}$$

The unit direction vector is:

$$\hat{\mathbf{D}}_{\text{C}} = \frac{\mathbf{D}_{\text{C}}}{|\mathbf{D}_{\text{C}}|} \tag{3.6}$$

Quaternion Rotation for Paddle Alignment

The virtual paddle rotation is computed using **Unity's LookRotation function**, which aligns the virtual paddle's rotation with $\hat{\mathbf{D}}_{\text{C}}$:

$$Q_{\text{P}} = \text{Quaternion.LookRotation}(\hat{\mathbf{D}}_{\text{C}}, \mathbf{y}) \tag{3.7}$$

where:

- Q_{P} is the rotation quaternion of the virtual paddle.
- $\mathbf{y}$ is the global up vector (typically $(0, 1, 0)$).

Mathematical Breakdown of LookRotation Function

LookRotation computes a quaternion that aligns:

- The **Z-axis** with $\hat{\mathbf{D}}_{\text{C}}$.
- The **X-axis** with $\hat{\mathbf{D}}_{\text{C}} \times \mathbf{y}$.
- The **Y-axis** with $\mathbf{Z} \times \mathbf{X}$.

where Z,X and Y axes represent the global axes of the virtual paddle.

Explanation of Quaternion

Eular Angles

Euler angles represent rotation using three angles (α, β, γ) around fixed axes (X, Y, Z). Euler angles follow a **hierarchical rotation order**, meaning each rotation is applied in sequence, affecting subsequent rotations.

Example: Rotation by (0, 90, 0) (Yaw 90° around Y-axis)

$$R = R_y(90^\circ) \cdot R_x(0^\circ) \cdot R_z(0^\circ) \tag{3.8}$$

Euler angles are simple but can cause **Gimbal Lock** when two rotation axes align where it occurs when one axis is lost due to alignment, restricting movement. In as 3D space Degre of Freedom (DoF) reduce to 2 from 3. For more information about Gimbal Lock: Click here.

Fix: Use **quaternions** instead of Euler angles.

Quaternions

A quaternion is defined as:

$$Q = w + xi + yj + zk \tag{3.9}$$

where:

$$w = \cos(\theta/2)$$
$$(x, y, z) = \text{rotation axis} \times \sin(\theta/2)$$

Example: Rotation of (0, 0, 90) around Z-axis

$$Q = \cos(45^\circ) + 0i + 0j + \sin(45^\circ)k \tag{3.10}$$

$$Q = 0.707 + 0i + 0j + 0.707k \tag{3.11}$$

Fixing Gimbal Lock with Quaternions

Instead of Euler angles, use quaternion multiplication to avoid the Gimbal lock.

Example: Applying (90, 90, 0) Rotations

$$Q_1 = 0.707 + 0.707i + 0j + 0k$$
$$Q_2 = 0.707 + 0i + 0.707j + 0k$$
$$Q_{\text{result}} = Q_1 \times Q_2 = 0.5 + 0.5i + 0.5j + 0.5k$$

For more information about Quaternions: Click here.

This research presents a mathematical model for mapping a VR kayak paddle using two controllers. By computing the midpoint for positioning and using quaternion-based alignment for orientation, we ensure smooth and realistic paddle interactions in VR.

Step 02: Virtual Kayaking Simulation based on the Virtual Paddling

Algorithm 2 Virtual Kayaking Simulation based on the Virtual Paddling

procedure START
  Compute previous recorded virtual paddle blade positions relative to virtual kayak center:
    $P_{LC_K}(t-1) \leftarrow P_L(t) - C_K(t)$
    $P_{RC_K}(t-1) \leftarrow P_R(t) - C_K(t)$
  Set global variables for water detection status:
    $WP_L(t) \leftarrow \text{IsInWaterPlane}(P_L(t))$
    $WP_R(t) \leftarrow \text{IsInWaterPlane}(P_R(t))$
end procedure
procedure UPDATE **after each fixed time duration**
  if $WP_L(t) = 1$ or $WP_R(t) = 1$ **then**
    DIRECTIONCOMPUTATION
    TORQUECOMPUTATION
    FORCECOMPUTATION
  end if
  WATERDETECTIONANDSTATEUPDATE
end procedure
procedure DIRECTIONCOMPUTATION
  Compute displacement vector of the virtual paddle blade:
    $D(t) \leftarrow (P(t) - C_K(t)) - P_{C_K}(t-1)$
  Compute cardinal direction:
    $O(t) \leftarrow G(D(t)) = (D_x, D_z)$
end procedure
procedure TORQUECOMPUTATION $T(t) \leftarrow \tau(P_{C_K}(t), P_{C_K}(t-1), T_P = Vector3(0,0,0))$
  Compute direction vector:
    $V \leftarrow P_{C_K}(t) - P_{C_K}(t-1)$
    A = $P_{C_K}(t-1)$, B = $P_{C_K}(t)$, S = T_P
  Compute projection of $S - A$ onto V:
    $Proj_V(S-A) \leftarrow \frac{(S-A)\cdot V}{V\cdot V} V$
  Compute perpendicular force vector:
    $V_\perp \leftarrow (S-A) - Proj_V(S-A)$
  Compute cross-product for torque:
    $U \leftarrow V \times V_\perp$
  Compute turning power:
    $T_f(t) \leftarrow ||P(t) - P(t-1)|| + O_t$
  Compute applied torque:
    $T(t) \leftarrow \frac{U_y(t) P_t}{T_f(t)^2}$
end procedure
procedure FORCECOMPUTATION
  Compute force application point:
    $F_p(t) \leftarrow C_K(t)$
  Compute force direction:
    $F_d(t) \leftarrow (-O_x(t), 0, -O_y(t))$
  Apply force:
    $F(t) \leftarrow F_d(t) \cdot P_e$
end procedure
procedure WATERDETECTIONANDSTATEUPDATE
  Update water plane status:
    $WP_L(t) \leftarrow \text{IsInWaterPlane}(P_L(t))$
    $WP_R(t) \leftarrow \text{IsInWaterPlane}(P_R(t))$
  Update previous paddle positions:
    $P_{LC_K}(t-1) \leftarrow P_L(t) - C_K(t)$
    $P_{RC_K}(t-1) \leftarrow P_R(t) - C_K(t)$
end procedure

Efficiently computing the movement direction of kayak paddle blades is crucial for achieving realistic virtual kayaking simulations. The function `CheckPaddleDirection()` determines the influence of paddle movements on the kayak's dynamics by analyzing the positional changes of the left and right paddle blades, computing forces and torques accordingly.

Let $P_L(t)$ and $P_R(t)$ represent the positions of the left and right virtual paddle blades at time t, respectively. The previous recorded paddle blade's positions relative to the center of mass of the kayak, are given by $P_{LC_K}(t-1)$ and $P_{RC_K}(t-1)$. The kayak's center of mass is denoted as $C_K(t)$, and the force application height offset is given by h_f.

i) Initialize the Function

At the **START**, the following are calculated for initialize the function.

$$P_{LC_K}(t-1) = P_L(t) - C_K(t)$$

$$P_{RC_K}(t-1) = P_R(t) - C_K(t)$$

Then, the global variables $WP_L(t)$ and $WP_R(t)$ which are represent the left paddle blade in water and right paddle blade in water respectively, are set using the function `IsInWaterPlane()`, where it returns the status of the blade is in the water or not is detected.

$$WP_L(t) = \text{IsInWaterPlane}(P_L(t)), \quad WP_R(t) = \text{IsInWaterPlane}(P_R(t))$$

Then, only after the **fixed time duration** has passed in **UPDATE**:

- `Direction Computation`, `Torque Computation` and `Force Computation` are executed, if left or right virtual paddle blade is inside the water (based on the status given by $WP_L(t)$ and $WP_R(t)$).
- Finally, `Water Detection and State Update` is updated.

ii) Direction Computation

If either paddle blade is submerged, its displacement vector relative to the kayak center is computed as:

$$D_L(t) = (P_L(t) - C_K(t)) - P_{LC_K}(t-1) \quad \text{if left paddle blade in water}$$

$$D_R(t) = (P_R(t) - C_K(t)) - P_{RC_K}(t-1) \quad \text{if right paddle blade in water}$$

The directional output is determined using the function `GetCardinalDirection()`, which maps the displacement vectors $D_L(t)$ or $D_R(t)$ to a predefined set of movement directions. The function is mathematically defined as:

$$G(D(t)) = (D_x, D_z)$$

where $G(D(t))$ is the computed cardinal direction for displacement $D(t)$. Here, the function extracts only the horizontal components x and z of the displacement vector, ignoring the vertical y component. Also, the returned `GetCardinalDirection()` direction is stored as the `outputDirection` globally.

$$O(t) = (O_x(t), O_y(t)) = G(D(t))$$

iii) Torque Computation

If the left paddle blade is submerged, the torque applied to the kayak is computed as:

$$T_L(t) = \tau(P_{LC_K}(t), P_{LC_K}(t-1), T_P)$$

where $\tau(a, b, c)$ represents a torque computation `AddTorqueKayak()` function based on the position difference around the specific torque position T_P (In this case T_P = Vector3(0,0,0)). Similarly, for the right paddle blade:

$$T_R(t) = \tau(P_{RC_K}(t), P_{RC_K}(t-1), T_P)$$

The function `AddTorqueKayak()` computes torque for turning the kayak. Given the current and previous paddle blade's positions relative to the kayak's center of mass, projected onto the horizontal plane:

$$P(t) = (P_x(t), 0, P_z(t)), \quad P(t-1) = (P_x(t-1), 0, P_z(t-1)), \quad T_p = (T_x, 0, T_z)$$

Cross Product Calculation for Torque Computation

The function `CalculateCrossProduct()` is used to determine the perpendicular force vector contributing to the kayak's turning motion. Given three points:

- A = $P_{C_K}(t-1)$
- B = $P_{C_K}(t)$
- S = T_P

$$A = (A_x, A_y, A_z), \quad B = (B_x, B_y, B_z), \quad S = (S_x, S_y, S_z)$$

A direction vector is defined as:

$$V = B - A$$

A projection of $S - A$ onto V is computed as:

$$Proj_V(S - A) = \frac{(S - A) \cdot V}{V \cdot V} V$$

The perpendicular vector is:

$$V_\perp = (S - A) - Proj_V(S - A)$$

The cross-product is then computed as:

$$C = V \times V_\perp$$

which represents the torque-inducing vector applied to the kayak.

The following graph visualizes the cross-product calculation for torque computation. It illustrates the vectors $V = B - A$, $S - A$, the projected component $\text{Proj}_V(S - A)$, the perpendicular vector $V_\perp$, and the final torque-inducing cross product $C = V \times V_\perp$.

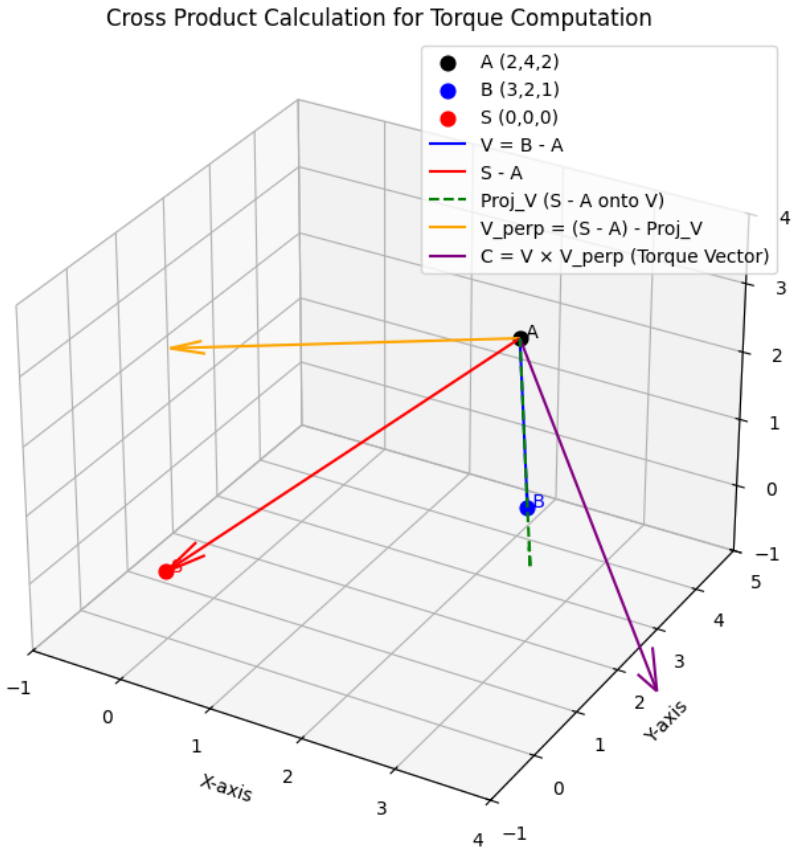


Figure 3.20: Visualization of Cross Product Calculation for Torque Computation

The function `CalculateCrossProduct()` calculates the cross product between the force vector ($P_c - P_p$) and perpendicular vector from T_p to the force vector. So, the cross-product computation for torque is given by:

$$U(t) = \lambda(P(t), P(t-1), T_p)$$

where U_y is extracted as the up-vector component. The turning power factor is calculated as:

$$T_f(t) = ||P(t) - P(t-1)|| + O_t$$

where O_t is a tuning offset. The applied torque to the kayak using **Unity's AddTorque function** (as an `Acceleration Force Mode`) is:

$$T(t) = kayakRigidbody.AddTorque(\frac{U_y(t)P_t}{T_f(t)^2}, ForceMode.Acceleration)$$

where P_t is the turning power coefficient.

iv) Force Computation

The force application point on the kayak is computed as:

$$F_p(t) = C_K(t)$$

The function `AddForceKayak()` applies force to the kayak at a specific force position $F_p(t)$. The force direction is computed as:

$$F_d(t) = (-O_x(t), 0, -O_y(t))$$

where $O(t) = (O_x(t), O_y(t))$ represents the `outputDirection` vector. The applied force to the kayak using **Unity's AddForceAtPosition function** (as an `Acceleration Force Mode`) is given by:

$$F(t) = kayakRigidbody.AddForceAtPosition(F_d(t) \cdot P_e, ForceMode.Acceleration)$$

where P_e is the engine power coefficient. The force is applied at $F_p(t)$ using positional force application.

v) Water Detection and State Update

The function checks if each paddle blade is submerged in the water using:

$$WP_L(t) = \text{IsInWaterPlane}(P_L(t)), \quad WP_R(t) = \text{IsInWaterPlane}(P_R(t))$$

where `IsInWaterPlane()` is defined as:

$$WP(P(t)) = \begin{cases} 1, & P_y(t) < W_L \\ 0, & P_y(t) \geq W_L \end{cases}$$

where $WP(P(t))$ determines if the paddle blade $P(t)$ is below the water level W_L.

Then, the previous paddle blade's positions are updated for subsequent calculations as:

$$P_{LC_K}(t-1) = P_L(t) - C_K(t), \quad P_{RC_K}(t-1) = P_R(t) - C_K(t)$$

This function systematically calculates paddle motion, determines directionality, and applies forces and torques to simulate realistic kayaking dynamics. By utilizing displacement vectors, torque computations, and force applications, it ensures accurate movement representation in the virtual environment.

d) Voice Commands Recognition

The voice command system was fully integrated into a Unity-based VR kayaking application using a combination of the Meta XR All-in-One SDK and a BERT-based NLP model. The process involves both real-time voice input from the user and model inference for command classification.

Step 01: Speech to Text with Meta XR SDK

The Meta XR All-in-One SDK was installed in Unity to enable voice capture on compatible VR headsets (e.g., Meta Quest 2). This SDK provides APIs to access the microphone and convert voice to plain text using built-in Automatic Speech Recognition (ASR).

- The Unity script listens for user input using the Meta Voice SDK.
- When a user speaks, the SDK captures the audio and internally processes it through ASR.
- The output of this process is a plain text string of the user's spoken command.
- This text is passed to the NLP model for classification.

Step 02: Text Classification with BERT-base-uncased Model

The transcribed text is processed using a fine-tuned BERT-base-uncased NLP model on a custom dataset containing voice commands for classification to 4 type of predefined classes as shown in Table 3.1. This step determines the specific command and triggers the corresponding function in the application.

- **BERT-base-uncased Model:** Used for robust text classification.
- **WordPiece Tokenizer:** Tokenizes the input text efficiently.
- **Optimizer:** AdamW (Adam with Weight Decay) ensures stable convergence.
- **Loss Function:** Cross Entropy Loss is used for multi-class classification tasks.

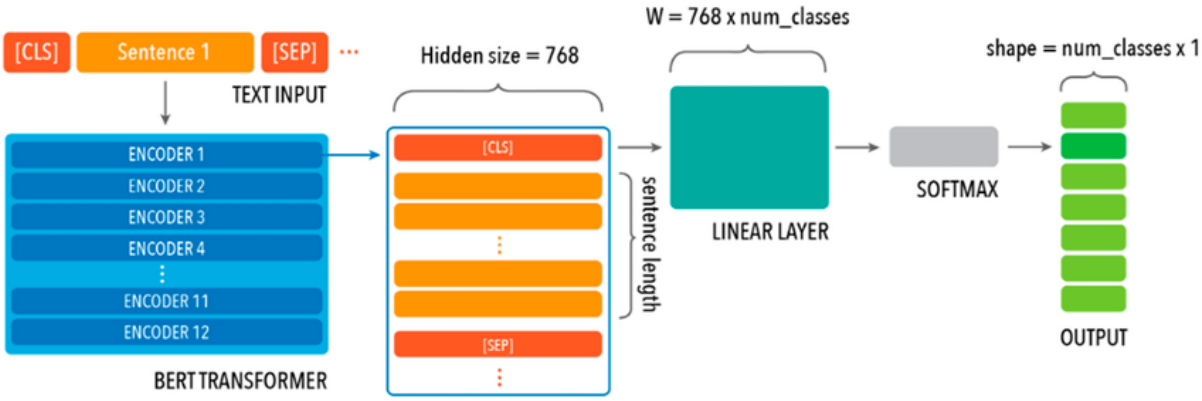


Figure 3.21: BERT base uncased model architecture, which comprises 12 transformer block layers, each with a hidden size of 768, and an added linear layer and softmax, which yields the output

Command Classes and Functions

The classified text corresponds to one of the predefined command classes as shown in Table 3.1. Each class is linked to a specific function within the system.

Table 3.1: Classes & Functions

Class	Function
`pause_menu`	Trigger the Pause menu
`main_menu`	Exit to Main menu
`align_arrow`	Align the boat along the arrow
`update`	Change the virtual environment

e) Score System

The LeaderBoard script manages the leaderboard functionality for a Unity game for displaying and updating high scores. The SetHighscoreEntry function updates the leaderboard by getting the high score table from PlayerPrefs, using the player's first name. Because the entire system identifies each users considering users First name field as the key identity (ID). There are common saving points for both leaderboard and reward system for all the players to add scores when achieving in the game. So, the function adds the current player's marks to their existing score of the player or creates a new entry if the player is not on the table. It uses a binary search algorithm to determine the correct position for new entries and sort the leaderboard. After updating, it stores the updated table back to PlayerPrefs. For allowing users to filter leaderboard entries by name, it includes search functionality. A special Congratulation object appears to celebrate players who achieve the first rank while highlighting the top three positions with visual elements like trophies and colors.

The RewardManager script tracks and unlocks predefined achievements for gaining additional scores. When a user completes an achievement reward manager stores its index in PlayerPrefs common current store point. Then the manager combines current and saved reward indexes and stored them to corresponding player. Meanwhile, it updates UI elements like buttons and text to unlock the corresponding rewards for collection. Once collected, the reward's score is added to the user's total leaderboard score. The reward item is then reset and locked again in future achievements.

Passing each checkpoint, winning given number of races, finish the race before a target time etc., are some of examples for gaining scores and the system shows them as Hints.

3.2.4 Kayaking Environment Background

The reason why we use both real and virtual worlds: always to keep the control of the player's movements in the game environment. We cannot do this using only real-world environments, since they rely on pre-recorded videos which cannot change their course of action dynamically with respect to the players' actions. For example, while thinking that the player paddles to the left, the environment should reflect this action. Using only real-world environments, requires recording various of videos in various directions, captures frames for every possible direction a player could face, and making transitions between them. This requires many computer resources and is an extremely complicated job. We can integrate the virtual river, where, with every action of the player, environmental aspects change dynamically for a smooth and realistic experience.

To realize this concept effectively, we followed a structured workflow that combined both real-world video data and dynamic virtual elements. The following workflow outlines the key stages involved in creating our VR Kayaking experience.

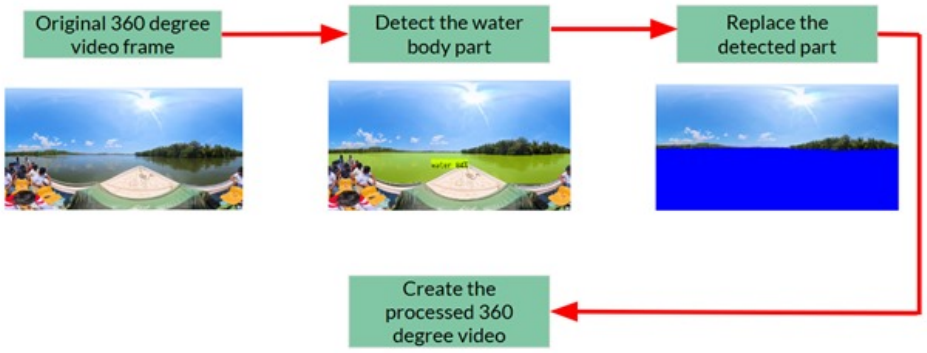


Figure 3.22: Video Preprocessing Work Flow

Original 360 degree video frame(Extracting the Equirectangular Projection)

The 360-degree video I worked with was recorded using an equirectangular projection method. This approach takes in the entire spherical view all around the camera—from left to right (360 degrees across) and from top to bottom (180 degrees up and down)—which is then "flattened" into a rectangular image.

This flattening allows us to represent and process the entire environment as a2D video frame whose width coincides with the entire horizontal rotation and whose height corresponds to the vertical angle.

Utilizing equirectangular projection has the added benefits of being able to more easily use computer vision and machine learning tools (where a frame becomes a standard image) and still maintain the 360-degree view forVR.

Figure 3.23: Equirectangular Projection

Splitting the Video into Frames

Rather than trying to process the entire video at once, I'veused Python's OpenCV library to loop over a video and output individual frames. By working on one frame at a time (sampling every few frames to save processing time), I could analyze and modify each image separately, making the entire process faster and more efficient.

Detecting the Water Region Using AI with Roboflow

For detecting water areas in each frame, I used a pre-trained computer vision model from Roboflow, which is a popular platform to deploy computer vision models via API. The model was trained to detect the "water" class in images.

Methodology for Accessing the Roboflow Model

- First, create a free Roboflow account at roboflow.com.
- Get your API key on the Roboflow dashboard.
- Utilize the given model id, in this instance "vr-kayaking/5," to pass image data to the Roboflow Application Programming Interface (API).
- The model outputs predictions that contain the coordinates of the water regions identified.

For each frame:-

- Resized the image to the model's expected input size (512x512 pixels).
- Encoded the image in base64 format to send securely over the web.
- Sent the encoded frame to Roboflow's inference API.
- Received back the water region coordinates in the form of polygons.

Model Output:-

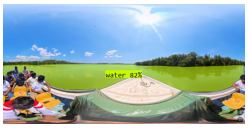


Figure 3.24: Model Output image


```
{
  "predictions": [
    {
      "x": 738,
      "y": 455,
      "width": 1474,
      "height": 172,
      "confidence": 0.819,
      "class": "water",
      "points": [
        {
          "x": 326.723,
          "y": 370.5
        },
        {
          "x": 324.406,
          "y": 371.658
        },
        {
          "x": 291.966,
          "y": 371.
```


Figure 3.25: Model Output Values

Performance Analysis of the model:-

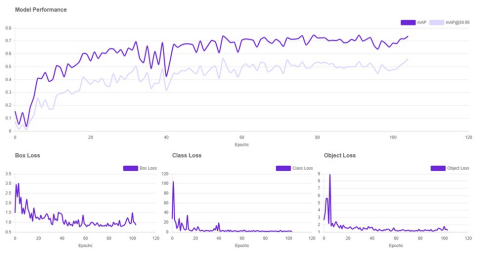


Figure 3.26: Model Performance analysis

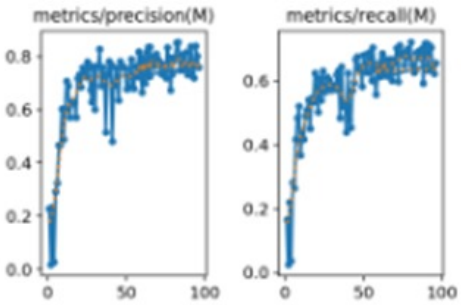


Figure 3.27: Model precision and recall

Applying Water Mask and Custom Coloring

Once I obtained the water coordinates, I developed a mask emphasizing the specified regions. A semi-transparent blue overlay (RGBA: 0, 0, 255, 255) was applied to predominantly highlight the water pixels. This made the water stand out visually and helped separate it from the rest of the scene.

Replacing the Boat and Water Regions

My custom algorithm took this further by coloring the water and lower parts of the frame where water was detected. This step prepared the frames for integration with the virtual river and kayak in Unity. Although the frame still looked natural, this colored overlay acted as a placeholder for the virtual elements we added later, allowing smooth combination of real and virtual content.

This algorithm is designed to detect and remove the water region (and kayak, if applicable) from a video frame, starting from the top-left corner and working down each column.

Step-by-Step Explanation:

1. **Start from the Top-Left Corner:**
   - Begin scanning the video frame column by column.
   - For each column, start from the top-most pixel and move downward.
2. **Detect the First Water Pixel in Each Column:**
   - While scanning down a column, check each pixel to see if it belongs to the water class (i.e., classified as part of the water body using a segmentation or classification model).
   - Once a pixel belonging to the water class is found, perform an additional check.
3. **Verify with Additional Pixels:**
   - After detecting the first water pixel in the column, check the next two pixels below it.
   - If all three consecutive pixels belong to the water class, it is considered a valid detection.
4. **Remove the Water Region in the Column:**
   - Once confirmed, remove (or mark for removal) all pixels starting from that first detected water pixel down to the bottom of the column.
   - This effectively removes the water region for that vertical slice of the frame.
5. **Repeat for All Columns:**
   - Continue the same process for every column across the width of the frame.

Saving the Processed Frames

Each processed frame—with the highlighted water regions—was saved as a high-quality PNG file with transparency. These files were stored in a folder to be used for video reconstruction.

Reconstructing the 360-Degree Video Using FFmpeg

After removing the water region and kayak from each frame of the 360-degree video, a sequence of processed image frames remains. To recreate the video from these individual frames, FFmpeg—a powerful command-line tool for handling multimedia files—was used.

The FFmpeg Command Explained

Here's what each part of the command means:

```
ffmpeg -framerate 30 -i frame_%05d.png -c:v libx264 -pix_fmt yuv420p output360.mp4
```

Figure 3.28: Video reconstruction function

`-framerate 30`

- This sets the frame rate to 30 frames per second (fps).
- It tells FFmpeg that each second of video should be made up of 30 image frames.
- This ensures the video playback speed matches the original timing and provides smooth motion.
- For example, if you have 300 processed frames, the final video will be 10 seconds long (300 ÷ 30).

`-i frame_%05d.png`

- This specifies the input file pattern:
  - `frame_00001.png`, `frame_00002.png`, ..., `frame_99999.png`
  - `%05d` means the numbers are zero-padded to 5 digits.
- FFmpeg reads these image files in numerical order and treats each one as a video frame.

`-c:v libx264`

- This sets the video codec to `libx264`, which is a widely used H.264 encoder.
- H.264 is efficient, giving good quality at a relatively small file size.
- It ensures compatibility with most modern video players and platforms.

`-pix_fmt yuv420p`

- This sets the pixel format to `yuv420p`, which is the most compatible format for video playback.
- Many video players and web platforms require this format.
- Without this, some players (especially browsers or mobile devices) may not render the video properly.

`output360.mp4`

- This is the name of the output video file that will be created from your image sequence.
- It will be a smoothly playing, 30 fps 360-degree video, built from the water-removed and processed frames.

Sending the Video to Unity

The final video was imported into Unity, where it was mapped onto the inside of a 3D sphere. The user stands at the center of this sphere, allowing them to look around and experience the real-world environment in 360 degrees.

Along with this video sphere, a virtual river was created inside Unity based on the water masks from the video frames. This mix of real video and virtual objects created an immersive VR kayaking experience, combining reality and virtual design seamlessly.

Technologies and Tools Used

- **Python & OpenCV**: For extracting video frames and image processing.
- **Roboflow API**: For water region detection using a pre-trained machine learning model.
- **FFmpeg**: For video reconstruction from processed frames.
- **Unity 3D**: For creating the VR environment and combining real video with virtual objects.
- **Equirectangular Projection**: To handle and process the 360-degree video format.

This process was key to merging real-world 360-degree videos with interactive virtual environments. By detecting and highlighting water regions, we could integrate a virtual river and kayak, enhancing the realism and engagement of the VR kayaking simulation. Using Roboflow's AI model allowed for fast and accurate water detection without building a model from scratch. This integration of real and virtual elements created a unique and immersive VR experience.

3.2.5 Voice Chatbot System

Introduction

In this final year project, I developed a voice-based chatbot system integrated into a Virtual Reality (VR) application. The primary objective of this system was to enable users to interact naturally with a virtual agent using voice. Unlike traditional text-based chatbots, the voice chatbot allows for immersive, hands-free communication within a 3D VR environment.

The system combines real-time audio recording, WebSocket communication, speech-to-text transcription, natural language understanding, and text-to-speech (TTS) synthesis. The chatbot backend leverages a large language model (LLM) and Google Cloud Speech-to-Text, while the frontend is implemented using Unity for VR. This chapter outlines the development process, system components, and key technologies used in the implementation of this voice chatbot.

System Overview

The voice chatbot system consists of two major components:

- A Unity-based VR application that captures user voice input, sends it to a backend server, and plays back the synthesized voice response.
- A Python-based backend server that performs speech-to-text, processes input using an LLM, generates a response, and returns the synthesized speech.

System Architecture

The architecture follows a client-server model, with real-time bidirectional communication between Unity and the backend using WebSockets. The following components are part of the architecture:

- **Unity VR Client:** Captures voice input using Unity's Microphone API, encodes the audio in Base64, and sends it to the backend via WebSocket.
- **Backend Server:** Built with FastAPI, it decodes the audio, transcribes it using Google Cloud Speech-to-Text, generates a response using a large language model (e.g., Gemini Pro), converts the response to speech using Google Text to Speech API, and returns the result as Base64-encoded WAV.
- **Audio Playback:** Unity decodes the Base64 audio, converts it into an AudioClip, and plays it in the VR environment.

Audio Capture and Encoding in Unity

The Unity side of the application uses a button of a VR controller to trigger audio recording. When pressed, Unity begins recording the user's voice and stops recording on release. The recorded audio is then converted into a WAV byte array using a custom `WavUtility` class and encoded to Base64.

WebSocket Communication

Unity connects to the backend via a WebSocket endpoint `/ws/chat`. It sends a JSON payload with user ID and Base64-encoded audio. The backend decodes, processes, and sends back a JSON payload containing Base64-encoded audio of the chatbot's response.

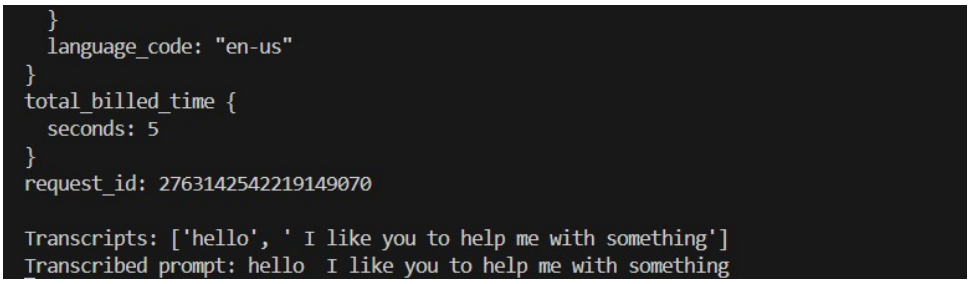


Figure 3.29: Debug trail of unity

Speech Recognition and Text Processing

The backend uses Google Cloud Speech-to-Text for accurate transcription. Before transcription, the audio is converted to mono format and the sample rate is extracted dynamically to ensure compatibility.

After transcription, the text is sent to the LLM for preprocessing. Because of environmental noise, speech-to-text transcription may not be very accurate. Therefore, our algorithm will feed that prompt to the LLM to obtain a more accurate prompt. Finally, the prompt will be given to the LLM. The chatbot maintains user-specific sessions to handle conversation context effectively.

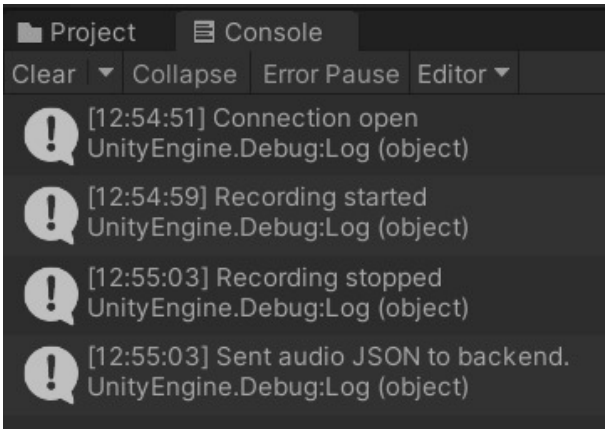


Figure 3.30: Speech recognition

Text-to-Speech Conversion

The chatbot's response is cleaned (removing special characters or repeated punctuation) and converted to audio using the `Google` TTS engine. The result is saved as a WAV file using `pydub` for compatibility with Unity.

Audio Decoding and Playback in Unity

Unity receives the Base64-encoded WAV data and decodes it into an AudioClip using a modified `WavUtility` script. This script extracts metadata like sample rate and channel count from the WAV header. The decoded audio is then played back using Unity's `AudioSource` component.

Error Handling and Debugging

Several issues were encountered and addressed during development:

- WAV files with zero sample rate or channel count caused division errors. These were mitigated by validating and setting defaults.
- Unity failed to parse unsupported WAV formats (e.g., compressed audio). This was resolved by ensuring the backend generates PCM-encoded WAV files.
- Low volume from TTS was fixed by normalizing audio using `pydub`'s gain controls.

Technologies Used

- **Unity3D** – VR application and voice interface
- **FastAPI** – Backend server for WebSocket handling
- **Google Cloud Speech-to-Text** – Audio transcription
- **Gemini Pro / ChatGPT** – Natural language understanding and response generation
- **pyttsx3 and pydub** – Offline TTS and audio format conversion
- **WebSockets** – Real-time communication between client and server

3.3 Techniques and Tools

Section	**Techniques/Tools/ Software**	**Reason**	**Alignment**
Platform Design	Curve-Based Platform Design	The curved base replicates the rolling motion experienced when paddling on water.	Works alongside the motion tracking system to translate physical motions into virtual kayak behavior & improves physical interaction and reduces motion sickness.
Sensor Communication	Bluetooth Low Energy (BLE)	Provides low-latency and low-power wireless communication, essential for real-time data transfer.	Ensures seamless and efficient transmission of sensor data in the VR kayaking system, enabling real-time feedback and control.
	ESP32 Microcontroller	Offers integrated BLE capabilities and sufficient processing power for handling multiple sensor inputs.	Simplifies hardware design by combining sensor interfacing and wireless communication, critical for accurate and timely data processing.
Unity Development	Unity Software	The best latform for building VR application & Built-in support for VR headsets ensures smooth development and deployment.	Easy to create the entire VR application with UIs, Environmnets, ML-Agents.
	Unity ML-Agents Toolkit	Allows to create ML-Agents by using Reinforcement Learning	Bot players are trained to simulate realistic competitors, making single-player mode more engaging.
Kayaking Environment Background	Use Transparent 360-degree video superimpose with the virtual river	Able to change the virtual environment according to the player actions. Instead if we use only single real world video we unable to change the virtual environment according to the player actions.	Lies the transparent 360-degree video on to the unity sphere and inside the unity sphere there is a virtual river.
Interact Voice Bot	LLM	It is free and have good cognition ability. Have a method to initialize chat. So the response will be influence by previous conversations.	The LLM enables natural and context-aware conversations by remembering past interactions.
	Python Backend	Python is well-suited for backend development due to its simplicity, flexibility, and strong support for AI/ML libraries.	The Python backend serves as the core processing unit, handling chatbot requests, managing conversation history, and interfacing with the LLM.

Table 3.2: The specific techniques, tools, or software used for research, design, development, or analysis

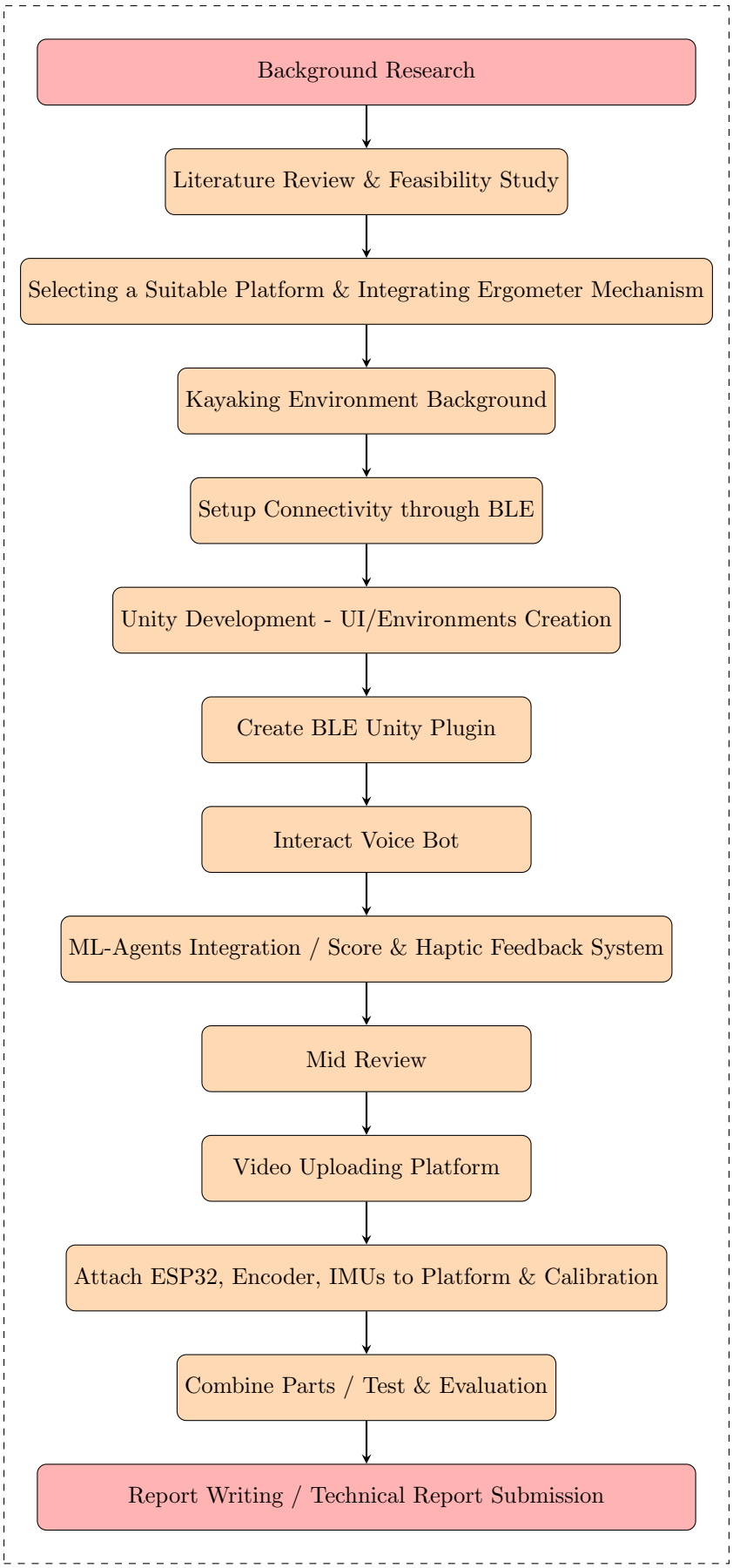
3.4 Resource Requirements and the Budget

Item	No. of Units	Unit Price (Rs)	Cost for 1 Set (Rs)
Meta Quest 3	1	180000	180000
Platform with Ergometer Mechanism	1	138750	138750
IMU Sensor	2	3000	6000
Encoder	1	5000	5000
ESP32	1	1500	1500
Unity Assets	-	70000	70000
Subtotal			**401250**
Contingencies (10%)			40125
Subtotal with contingencies			**441375**
5% UNIC financial management fee			22068.75
Total			**463443.75**

Table 3.3: Budget Breakdown for Kayaking Project

* The specific techniques, tools, or software used for research, design, development, or analysis

3.5 Steps Undertaken



3.6 Task Delegation

Tasks	Rasanjana	Deshapriya	Ranuka	Prabhashana
Feasibility study				
Literature Review				
Platform design				
Environment Background				
Unity Development				
Sensor Communication				
Interact Voice Bot				

Table 3.4: Task Delegation

3.7 Timeline

Figure 3.31: Project Timeline

3.8 Project Deliverables

- **Graphical User Interface (GUI)** Design and implement the user interface, including the introduction screen, main menu, in-game HUD, pause menu, and result menu.
- **Kayaking Races and Bot Players** Develop a virtual kayaking racing mode with AI-controlled bot players using ML-Agents.

- **Kayaking Physical Platform with Feedback System** Integrate a physical kayaking platform with real-time feedback mechanisms, including kayak rotation, ergometer speed tracking, and VR controller-based motion synchronization.
- **Score System** Implement a leaderboard and reward system to track player performance and encourage competitive gameplay.
- **Interact-Bot for Voice Chat** Develop an AI-powered bot to facilitate real-time voice interaction between users.
- **Multiple Map Selection** Allow users to choose between virtual kayaking racing maps and 360-degree video-based environments.
- **360-Degree Video Environment with Virtual River** Enhance realism by integrating a virtual river into a 360-degree video-based kayaking environment.

3.9 Validation Plan

To ensure the accuracy, efficiency, and robustness of our VR kayaking system, we employ a structured validation approach across multiple components.

3.9.1 BLE Communication Validation

We evaluate the BLE communication performance by measuring the delay between two ESP32 devices. The primary metrics used for validation include:

- **Ping-Pong Delay:** Measured using histograms, with an average delay of 12.37ms.
- **Data Packet Delay:** Verified through repeated trials, averaging at 23.99ms.

The obtained results confirm that the BLE communication remains within the acceptable latency range, making it suitable for real-time VR applications.

3.9.2 Unity Development Validation

The validation of the Unity-based VR kayaking application focuses on functionality, performance, and user interaction.

- **Scene Navigation:** Each scene transition (e.g., Intro, Main Menu, Game, Results) is tested for smooth loading and responsiveness.
- **HUD and UI Elements:** The interface elements, such as race ranking, speed meter, and maps, are verified for correctness and real-time updates.
- **VR Motion Tracking:** Validation of the motion tracking system ensures accurate mapping of the VR controllers to paddle ends, allowing natural kayaking movement.
- **Water Simulation:** The fidelity of water effects, including FFT Waves, ripples, reflections, and buoyancy physics, is evaluated under various scenarios.

3.9.3 Score System Validation

To validate the scoring mechanism, we measure:

- Correct assignment of scores based on race completion and difficulty level.
- Functionality of the Leaderboard system, ensuring accurate ranking updates.
- Verification of the "Hints" and "Rewards" system to confirm correct score increments upon unlocking achievements.

3.9.4 PPO Model Training Validation

The PPO model used for AI-based opponent navigation is validated using:

- **Mean Cumulative Reward:** Evaluated after re-training, achieving 0.7702.
- **Value Loss:** Ensuring minimal deviation, with a value of $2.0040e^{-5}$.

These results confirm the reliability of the trained PPO model for ML-based racing opponents.

3.9.5 Kayaking Environment Background Validation

We validate the accuracy of water region detection under various conditions:

- Daytime and nighttime scenarios.
- Turbulent and normal water conditions.

3.9.6 Overall System Benchmarking

The system is benchmarked for latency, frame rates, and real-time responsiveness. Key evaluation criteria include:

- **Latency:** Ensuring real-time responsiveness with low communication delays.
- **Frame Rate:** Maintaining a stable frame rate of at least 60 FPS in VR environments.
- **User Testing:** Conducting user trials to assess immersion, usability, and engagement.

This structured validation approach ensures the VR kayaking system meets performance, usability, and reliability standards.

3.10 Addressing Limitations

Section	Limitation	How Managing
Platform Design	All players have the same feeling while on kyakaing.	Use a spring connect with the ropes and we can adjust th eresistacne of that rops connect to the paddle then e can chage theb resistace value.
Unity Development	High-quality water physics and real-time interactions can strain computational resources, affecting frame rates.	Using LOD (Level of Detail) and GPU-based shaders to balance performance and realism.
Sensor Communication	Sensor readings often suffer from noise and drift, leading to errors in data interpretation.	Apply advanced filtering techniques (e.g., Kalman Filter, Complementary Filter) to mitigate noise and stabilize sensor outputs.
Unity Development	High-quality water physics and real-time interactions can strain computational resources, affecting frame rates.	Using LOD (Level of Detail) and GPU-based shaders to balance performance and realism.
	Merging real-world footage with virtual elements while maintaining immersion is challenging.	Combining 360° video and virtual elements with AI-based adjustments to improve realism.
Kayaking Environment Background	If the 360° camera is placed low to the kayaking person head, it's hard to remove the kayaking person from the transparent video.	We give instructions to the user The camera should be mounted above the kayaker's head for a clear and unobstructed view of the river
	If the person recording the video is kayaking, the camera will experience excessive motion, making water detection accuracy low.	The video should be recorded from a stable platform, such as a racing boat, without paddling to minimize movement and vibrations.
Interact Voice Bot	In the voice description part audio will not describe the kayaking environment exactly.	When uploading the video get few details about the kayaking place. We used it to create the voice description.

Table 3.5: Limitations in the methods used and how these are being managed

3.11 Future Work in Methodology

The next phase of the project involves two key methodological steps:

3.11.1 Platform Design Enhancement

Currently, only the wooden base structure of the physical platform has been built. The next phase involves:

- Constructing the **ergometer mechanism** and integrating it with the base structure.
- Attaching key hardware components, including:
  - **IMU sensor** for motion tracking.

  - **Rotary encoder** for precise movement detection.
  - **ESP32 module** for wireless communication.
  - **Power supply unit** for stable operation.
- **Calibrating the sensors** to ensure accurate motion replication in the VR environment.

3.11.2 Video Uploading System

Since our VR application supports a **360-degree video background** in Free Mode, we need to develop a system allowing users to upload their own videos. The planned steps include:

- Developing a **full-stack web application** to handle video uploads.
- Implementing the **backend, frontend, and database connectivity**.
- Deploying the application to allow user access.
- Integrating a Kayaking Environment Background step to:
  - **Extract and remove the river section** from uploaded videos.
  - Use the processed video as the background in the VR environment.

This approach ensures that when a user uploads a video, developers can extract the river region, enabling seamless integration into the VR system.

Chapter 4

Results

4.1 Introduction to Results

: The results of our VR kayaking project demonstrate the effectiveness of our selected approach in enhancing user experience and system performance. Key findings include the successful implementation of a real-time virtual river environment, accurate motion tracking using IMU sensors, and stable BLE communication between the ESP32 and Oculus Quest. These results validate the feasibility of our design choices and highlight the potential of VR-based rehabilitation and fitness applications.

4.2 BLE Communication

We measured the communication delay between two ESP32 devices using BLE. The below histograms show Ping and Pong data delay, and Data packet delay. The results on Ping and Pong delay are shown as Averaging 12.37ms of delay, while the delay of the data packet, shown averaged at 23.99ms. These results prove that the BLE communication delay stays within the acceptable range and is suitable for real-time VR applications.

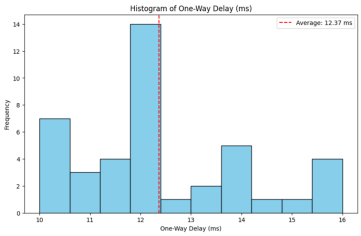


Figure 4.1: Histogram for Ping and Pong data delay

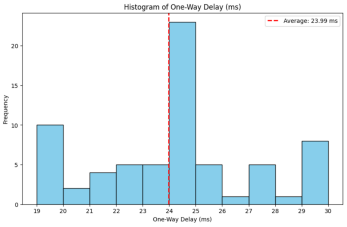


Figure 4.2: Histogram for data packet communication

4.3 Unity Development

We have used Unity software for the development of VR kayaking system application. The final output system follows the functional flow as the following given architecture block diagram. We have developed

a user interface (UI) for navigating between scenes and each component. The Unity system incorporates four scenes as shown in the architecture diagram. The "Intro" scene shows the starting loading window for the kayaking system, followed by the "Main Menu" scene, which afford main entry points of the functionalities.

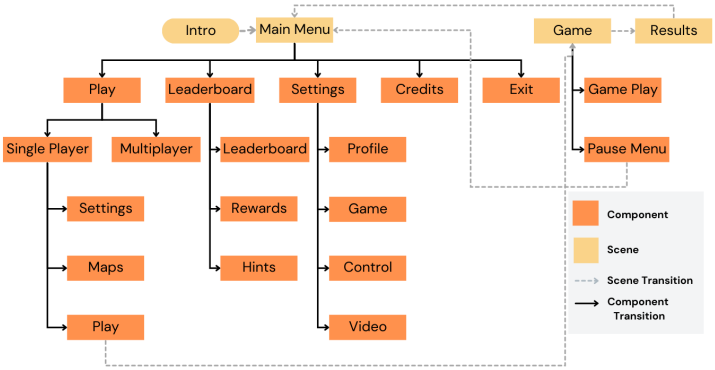


Figure 4.3: This architecture shows the final result of Unity Application and mainly focuses on the UI design, because we can explain properly the output functionality by navigation though the UI

4.3.1 Single Player Mode

In this menu, users can navigate "Play" to choose between Single Player or Multiplayer modes. In the "Single Player" panel, the "Settings" tab allows adjustments to game-related features, including game difficulty (Easy, Normal, Hard), music volume, Racing Mode (On, Off), Haptics Feedback (On, Off), the number of bot players, and kayak color customization. If users enable the Racing Mode, users can participate in virtual kayaking races which have Machine Learning (ML) as the opponent. If users disable the Racing Mode, users can play in Free Mode. So, this mode renders the 360 videos as the background on a Unity sphere. For making only the environment beyond the water visible and others invisible, we already preprocess these videos with image processing techniques, then remove water and kayak visuals. The "Maps" tab shows the virtual kayaking tracks or 360-degree video environments according to the chosen Racing Mode.

By clicking "Play" users navigate to the "Game" scene, where the kayaking gameplay begins.

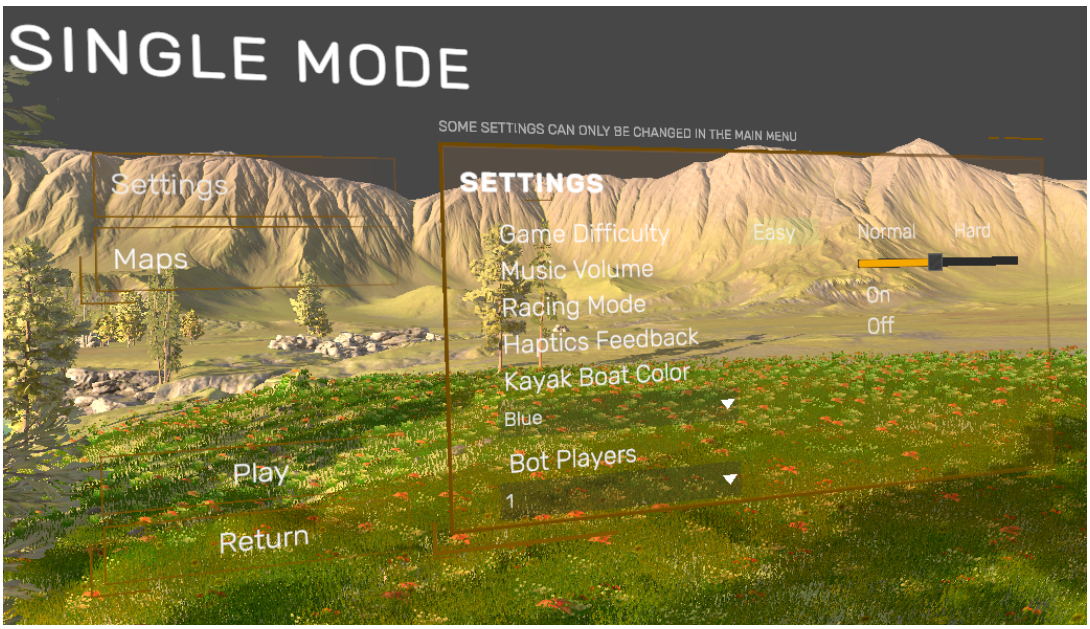


Figure 4.4: Single Mode User Interface (UI) with tabs:Settings (change game configuration settings) and Maps(choose between different Maps)

4.3.2 Game Play

While in the "Game" scene, players can improve kayaking experience with real-time HUD UI feedback, including a map, speed meter, and race ranking updates. By navigating to the "Pause Menu" the players can configure game options such as enabling or disabling the directional arrow which dynamically indicates the direction to next checkpoints, displaying key bindings, adjusting frame rate visibility and take screenshots. Users can also check the BLE connection with the ESP32 module and return to the Main Menu at any point. As a result of real-time mapping of two VR controller positions to paddle ends, users can move and rotate the kayak based on the physical paddling movements. High-fidelity water system enhances the water visuals integrating features such as FFT Waves, ripples, reflections, attenuation, particles, buoyancy physics etc. Upon completing the race, the system transitions to the "Results" scene providing results of the race and after users can navigate to the Main Menu.

Figure 4.5: A screenshot of the Game play. The display shows various HUD UI components: the Ranking list, speed meter, map and time remaining etc., Also, there is the arrow showing the direction of next checkpoint.

4.3.3 Score System

In the Main Menu, "Leaderboard" panel for promoting the boost engagement and learning outcomes [13] of the users, represents three tabs. The "Leaderboard" tab provides a comprehensive view of player rankings based on their gained scores. The "Hints" tab exhibits various strategies how to increment their scores and rise the leaderboard. The players have the opportunity to complete predefined achievements which the "Rewards" tab shows and unlock them to earn additional scores. By the "Settings" panel in the Main Menu, players are able to change various system configurations. The "Profile" tab enables users to view, edit, or create player profile details. The "Game" tab allows adjustments to gameplay settings, such as difficulty level, music volume, HUD and tooltips visibility while the "Control" tab configures VR controller input keys and sensitivity. In the "Video" tab, users can customize graphics settings. Additionally, the "Credits" panel highlights the development history of the VR kayaking system and show details of people who created.

4.3.4 The training result of PPO Model

After the re-training, the PPO model outputs best results with a mean cumulative reward and value loss of 0.7702 and 2.0040e−-05, respectively. Therefore, these results represent that we are able to use the output model for navigating the ML Agents properly. Figure 4.6 & Figure 4.8 show the corresponding plots.

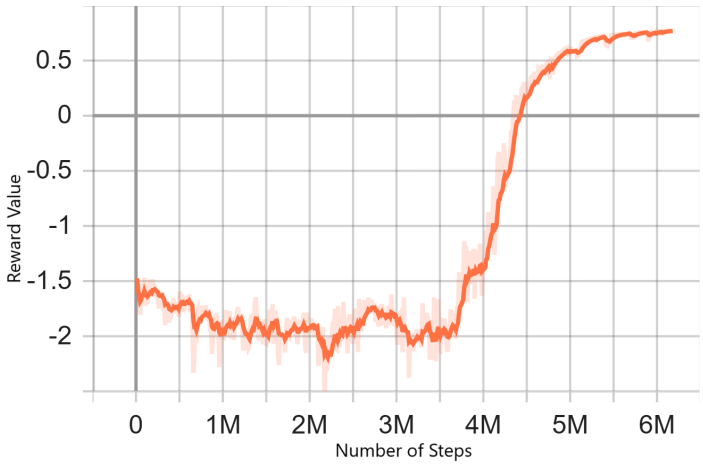


Figure 4.6: Default PPO result plots: Reward value vs Number of steps taken to train the NN model

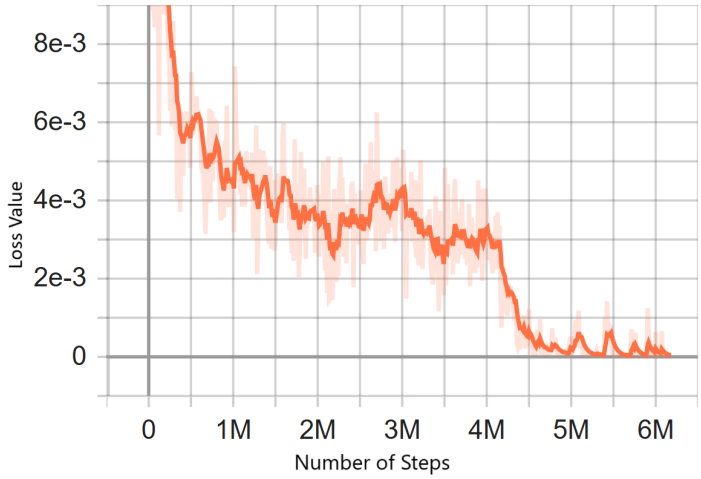


Figure 4.7: Default PPO result plots: Loss value vs Number of steps taken to train the NN model

4.3.5 Training Results of BERT-based Text Classification Model

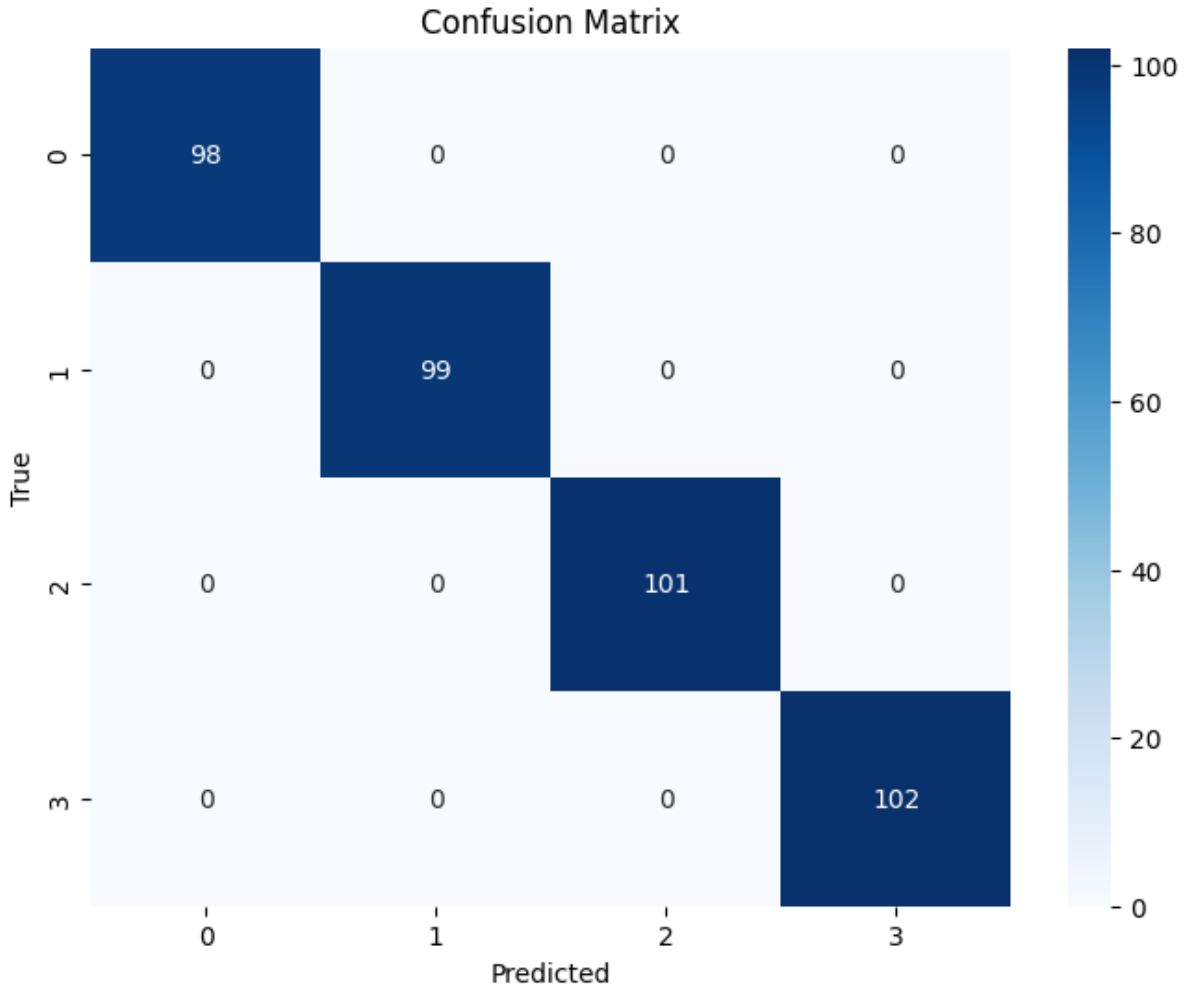


Figure 4.8: Confusion Matrix

4.4 Kayaking Environment Background Results

We measure the model accuracy in detecting the water region over different kinds of frames in different conditions, i.e., for daytime and nighttime conditions for both turbulent and normal water. Our model shows a very good detection of water regions, maintaining high accuracy and finding the best performances across the considered scenarios.

4.5 Presentation of Results

:

4.5.1 BLE Communication Delay Results Details

We measured the communication delay between two ESP32 devices using BLE. The results for Ping and Pong delay and data packet delay are summarized below. These results show that the BLE communication delay remains within an acceptable range for real-time VR applications.

- **Ping and Pong Delay:** Averaged 12.37 ms.
- **Data Packet Delay:** Averaged 23.99 ms.

These results demonstrate that the BLE communication delay is suitable for VR applications, ensuring minimal latency between the ESP32 devices.Also, we calculated a time difference between data coming to the unity side and motion start using that sensor data time which value was equal to 0.632ms. Also we calculated a total delay of 24,622ms. That delay was in the acceptable range in VR applications. Furthermore, we confirm reliable communication through negligible packet loss and steady RTTs, which is an essential requirement to keep up the synchronization in VR environments. These findings form a great basis for Bluetooth as one of the viable communication methods in low-latency real-time VR systems.

4.5.2 Water Detection Model Accuracy

The accuracy of the water detection model was evaluated under different scenarios: daytime and nighttime conditions with both turbulent and normal water. The model performed consistently well across various scenarios, demonstrating high robustness and adaptability.

Table 4.1: Results of Water Detection Accuracy Across Scenarios

Scenario	**Accuracy (%)**
Daytime - Turbulent Water	85
Daytime - Normal Water	92
Nighttime - Turbulent Water	78
Nighttime - Normal Water	88

Table 4.2: Results of Water Detection Accuracy Across Scenarios

The model achieved an overall accuracy range between **78% to 92%** depending on the conditions. The highest accuracy (92%) was achieved in daytime conditions with normal water, while the lowest (78%) was observed in nighttime turbulent water.

4.5.3 Unity Development Results Details

The Unity-based VR kayaking application was developed to provide users with an immersive kayaking experience. The system architecture includes several scenes and UI features, with the following components being of particular importance:

- **Intro Scene:** The loading screen for the system.
- **Main Menu:** Contains options such as selecting between Single Player and Multiplayer modes, settings, and map selection.
- **Single Player Mode:** Allows users to configure settings like game difficulty, racing mode, and kayak color customization.
- **Game Play:** Provides real-time feedback such as speed, map, and race ranking.
- **Results:** Displays race results after completion.

4.5.4 Score System Results Details

The Score System is designed to enhance user engagement and learning outcomes. It includes:

- **Leaderboard:** Displays player rankings based on scores.
- **Hints:** Offers strategies to improve performance.
- **Rewards:** Unlocks achievements and boosts scores.

4.5.5 PPO Model Training Results Details

We utilized a Proximal Policy Optimization (PPO) model to train Machine Learning (ML) agents for navigation within the VR kayaking environment. After re-training, the PPO model achieved the following:

- **Mean Cumulative Reward:** 0.7702
- **Value Loss:** 2.0040e−05

These results indicate that the model is performing well and can be effectively used for guiding ML agents in the VR kayaking system.

Result Type	**Value**
Ping and Pong Delay (ms)	12.37
Data Packet Delay (ms)	23.99
Daytime - Turbulent Water Accuracy (%)	85
Daytime - Normal Water Accuracy (%)	92
Nighttime - Turbulent Water Accuracy (%)	78
Nighttime - Normal Water Accuracy (%)	88
PPO Mean Cumulative Reward	0.7702
PPO Value Loss	2.0040e−05

Table 4.3: Summary of Results

4.6 Key Outcomes

The project has demonstrated promising outcomes in various critical areas. The BLE communication delay between two ESP32 devices was measured, with Ping and Pong delay averaging 12.37 ms and data packet delay averaging 23.99 ms. These results indicate that the BLE communication is fast and reliable, making it suitable for real-time VR applications. Additionally, the water detection model proved to be highly accurate, with detection rates of up to 92 under optimal conditions (daytime, normal water) and remaining robust across various challenging scenarios, including nighttime and turbulent water.

The PPO model, used for machine learning-based navigation, showed encouraging results with a mean cumulative reward of 0.7702 and a low value loss, indicating that the model is successfully learning and can be effectively used for further training. Overall, the project's outcomes suggest that the system's performance is aligned with the objectives, and preliminary validation of the results, including the water detection model and BLE communication, has been successfully achieved. These results validate the feasibility and robustness of the VR kayaking system for practical use.

Chapter 5

Discussion

The purpose of this discussion chapter is to interpret and analyze the results obtained in the previous sections in the context of the project objectives. The discussion will explore the significance of the findings, compare them with initial expectations, and address the challenges and limitations encountered throughout the project. Additionally, the implications of these results for the future steps of the project will be considered, along with suggestions for improvements or adjustments to enhance the project outcomes.

5.1 Interpretation of Results

The results obtained demonstrate that the VR kayaking system has performed well within the expected range for most of the key parameters. The BLE communication delay between two ESP32 devices was measured to average 12.37 ms for Ping and Pong delay, and 23.99 ms for data packet delay, indicating that the communication speed is well-suited for real-time applications. These low latency values validate the effectiveness of BLE for transmitting data between the ESP32 and the Oculus Quest 2, ensuring a smooth user experience for the VR kayaking system.

The water detection model demonstrated high accuracy across various conditions, with the best performance seen in daytime and normal water scenarios (92 accuracy), and acceptable results in challenging conditions such as nighttime and turbulent water (78 accuracy). This shows the robustness and adaptability of the model, which is critical for the VR system's immersive experience.

Additionally, the PPO model used for machine learning-based navigation produced promising results with a mean cumulative reward of 0.7702 and a low value loss, suggesting the model's potential for further refining the virtual kayaking experience through improved AI behavior.When comparing these findings with existing research and benchmarks, the results are in line with expectations for BLE communication delays and water detection accuracy. The low latency for BLE communication and the robust water detection model are key achievements that contribute to the overall effectiveness of the VR kayaking system.

5.2 Address Challenges and Limitations

While the results are promising, several challenges and limitations were encountered during the project. One of the challenges was ensuring the robustness of BLE communication, particularly with varying environmental factors such as interference from other devices. Although the latency was within acceptable limits, further testing in real-world conditions could provide additional insights into how well the system performs under more varied and unpredictable circumstances.

The water detection model, while effective, faced some limitations under extreme conditions, particularly at night and in turbulent water, where the accuracy dropped to 78. This suggests that there is room for improvement in the model's ability to adapt to different environmental variables, and further optimization may be required for consistent high performance across all conditions.Moreover, the PPO model's performance is promising, but further training is required to fully optimize the model for real-time applications in the VR system. The current model may not fully capture the complexity of user interactions with the kayak, which may impact its effectiveness in highly dynamic environments.

5.3 Implications

The results of this project have significant implications for the development of real-time VR systems, particularly those that involve physical interaction such as VR kayaking. The successful implementation of BLE communication with low latency suggests that this technology is a viable option for real-time data transmission in immersive VR applications. Additionally, the water detection model's high accuracy across most conditions indicates that the system can provide realistic visual feedback, enhancing user immersion.

The PPO model's promising performance opens the door for further exploration of machine learning techniques to improve the virtual kayaking experience. With further optimization, the model could potentially provide more sophisticated AI behavior, offering users a more challenging and engaging experience.Overall, the results indicate that the project is on track to meet its objectives and has the potential for broader applications in VR rehabilitation or other fields requiring immersive virtual environments.

5.4 Next Steps

To improve the system, several adjustments and optimizations are necessary. First, further testing of the BLE communication system in real-world environments should be conducted to assess its performance under varied conditions and ensure consistent low latency. In addition, the water detection model should be refined to improve accuracy in nighttime and turbulent water conditions. This could involve incorporating additional data sources, such as depth or motion sensors, to enhance the model's robustness.

For the PPO model, more training and fine-tuning are required to improve its performance in dynamic environments and ensure it can adapt to user interactions with the kayak. This could include adjusting the reward system and exploring alternative training algorithms to better capture the complexity of real-time user behavior.Finally, the user interface (UI) could be enhanced to improve usability and streamline the navigation between different scenes and settings. Feedback from users will be crucial in identifying areas for improvement.

5.5 The Overall FYP Outcome So Far

So far, the project has made significant progress in achieving its primary objectives. The VR kayaking system's key components, including BLE communication, water detection, and machine learning-based navigation, have been successfully implemented and validated. The results indicate that the system is capable of providing a realistic and immersive experience for users, with low-latency communication, robust water detection, and promising AI behavior.

While there are challenges and areas for improvement, particularly in optimizing the water detection model and refining the PPO navigation model, the project is on track to meet its goals. The next steps outlined above will help address these challenges and enhance the overall system's performance. The project has demonstrated the feasibility of using VR technology for applications such as rehabilitation and immersive gaming, and its successful implementation lays the foundation for future research and development in this field.Currently, approximately 65 of the project has been completed, with significant achievements in both hardware and software development. The integration of BLE communication, water detection, and AI navigation models represents major milestones. With further work needed to refine the models and enhance system performance, the project is well-positioned for successful completion.

Chapter 6

Conclusion

So far, the VR kayaking system has made significant progress in achieving its primary objectives, including the successful implementation of BLE communication, water detection, and machine learning-based navigation. Major milestones include validating low-latency communication, robust water detection, and AI-powered navigation, which have enhanced the user experience and engagement. Despite challenges such as optimizing the water detection model and refining the PPO navigation model, these issues have been mitigated through iterative testing and adjustments. The next phase will focus on improving these models, as well as optimizing the overall system for stability and user interface enhancements. This project not only demonstrates the feasibility of VR technology in addressing real-world issues like kayaking infrastructure and safety, but it also holds potential for use in gaming cafes, fitness centers, and tourism promotion, allowing users to experience kayaking locations virtually. The project is on track to meet its goals, and with the remaining tasks in sight, its successful completion is confidently anticipated.

Bibliography

[1] D.I.Y. Paddle Ergometers, "D.i.y. paddle ergometers," 2019, instructions, Available: https://diypaddleergometers.wixsite.com/instructions, Accessed: Aug. 10, 2024.

[2] A. Barmpoutis, R. Faris, S. Garcia, J. Li, J. Philoctete, J. Puthusseril, L. Wood, and M. Zhang, "Virtual kayaking: A study on the effect of low-cost passive haptics on the user experience while exercising," in *Advances in Human Factors and Ergonomics*. Springer, 2020, pp. 147–155.

[3] M. Begon, F. Colloud, and M. Estivalet, "Optimised preliminary design of a kayakergometer using a sliding footrest-seat complex (p174)," in *Proceedings of the International Conference on Sports Engineering*. Springer, 2009, pp. 171–177.

[4] J. Park and J. Yim, "A new approach to improve cognition, muscle strength, and postural balance in community-dwelling elderly with a 3-d virtual reality kayak program," *The Tohoku Journal of Experimental Medicine*, vol. 238, pp. 1–8, 2016. [Online]. Available: https://www.researchgate.net/publication/286511096_A_New_Approach_to_Improve_Cognition_Muscle_Strength_and_Postural_Balance_in_Community-Dwelling_Elderly_with_a_3-D_Virtual_Reality_Kayak_Program

[5] D. The. (2020, February 04) Homemade diy kayak ergometer from nordic track ski machine. Accessed: Aug. 10, 2024. [Online]. Available: https://davethekayaker.com/2020/02/03/home-made-kayak-ergometer-from-nordic-track-ski-machine/

[6] K.-H. Liu, T. Sasaki, H. Kajihara, A. Hiyama, M. Inami, and C.-H. Chen, "Virtual kayaking: A local culture-based virtual reality paddling experience," in *Lecture Notes in Computer Science*. Springer, 2020, pp. 630–642.

[7] Q. Dong and W. Dargie, "Evaluation of the reliability of rssi for indoor localization," *2012 International Conference on Wireless Communications in Underground and Confined Areas, ICWCUCA 2012*, pp. 1–6, 2012.

[8] J. Hanski and K. B. Biçak, "An evaluation of the unity machine learning agents toolkit in dense and sparse reward video game environments," Ph.D. dissertation, Dissertation, 2021.

[9] Y. Savid, R. Mahmoudi, R. Maskeliunas, and R. Damaševičius, "Simulated autonomous driving using reinforcement learning: A comparative study on unity's ml-agents framework," *Information*, vol. 14, p. 290, 2023.

[10] A. Barmpoutis, "Virtual kayaking: A study on the effect of low-cost passive haptics on the user experience while exercising," in *Communications in Computer and Information Science*, 2020, pp. 147–155.

[11] K.-H. Liu, T. Sasaki, H. Kajihara, A. Hiyama, M. Inami, and C.-H. Chen, "Virtual kayaking: A local culture-based virtual reality paddling experience," in *Lecture Notes in Computer Science*. Springer, 2020, pp. 630–642.

[12] A. Barmpoutis, R. Faris, S. Garcia, J. Li, J. Philoctete, J. Puthusseril, L. Wood, and M. Zhang, "Virtual kayaking: A study on the effect of low-cost passive haptics on the user experience while exercising," in *Advances in Human Factors and Ergonomics*. Springer, 2020, pp. 147–155.

[13] S. Park and S. Kim, "Development of leaderboard design principles to improve motivation in a gamified learning environment," *JMIR Serious Games*, vol. 9, 2019.