Nintendo Wii Over IP

Kieran Fowlds - 210018092
Supervisor: Dr. Tom Spink

30th March 2025

University of
St Andrews

Abstract

The Nintendo Wii is well-known for its innovative, motion-based controls and engaging, family-friendly games such as Mario Kart Wii. Despite its hardware limitations compared to modern consoles, its local multiplayer experiences have cultivated a devoted following. However, with the rapid shift toward online gaming, recreating the Wii's in-person, split-screen experiences has become increasingly challenging. This project proposes a solution that vitalises the Wii's input and output interfaces, enabling remote players to enjoy an experience that mirrors local multiplayer gaming.

The approach centres on a three key components. First, video and audio streaming techniques capture the Wii's outputs and deliver them to remote devices using low-latency protocols. Second, a novel controller input relay system transmits Wii Remote signals, including motion and button inputs, over a network. Third, a Wii Remote emulator interprets the remote player's inputs and forwards them to the Wii console.

By combining these components, the project enables remote players to participate in local multiplayer games on the Wii. Furthermore, it establishes a framework for adapting retro systems to contemporary, distributed gaming environments. The work not only preserves the social and communal essence of local play but also offers broader implications for making nostalgic gaming experiences accessible to players across geographically separated locations.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is [TODO: Add word count] words long

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis as long as proper credit is given to the authors.

I retain the copyright in this work, and ownership of any resulting intellectual property.

Acknowledgements

I would like to thank my supervisor, Dr. Tom Spink, for his encouragement, support, and counsel throughout my Senior Honours project. I would also like to thank my girlfriend, Angelika Blaszczak, for her endless love, patience, and understanding during both the challenging and triumphant moments of this journey. In addition, I am deeply grateful to my family and friends for their constant support and encouragement throughout my time at the University of St Andrews. Finally, I would like to thank the School of Computer Science for providing a stimulating academic environment, excellent resources, and insightful feedback that were essential to the successful completion of this project and my undergraduate studies.

Contents

1. Introduction

This project addresses a pressing challenge in the evolution of gaming experiences: how to adapt and extend the social and immersive qualities of local multiplayer systems – exemplified by the Nintendo Wii – to a modern, online environment. The Nintendo Wii has gained a large and dedicated following due to its motion controls, comparatively low price, and its local multiplayer gameplay. However, as online gaming has become the norm[15], the traditional split-screen and communal experiences that defined the Wii era have faced diminishing support and new technical challenges. The main challenge is that, due to the termination of first party online services[20], the Wii no longer supports online multiplayer. This coupled with the fact that most people no longer play multiplayer games locally[13], means that the Wii's local multiplayer games are no longer accessible to a large portion of its possible player base.

This project aims to address this issue by developing a system that enables remote players to experience the Wii's local multiplayer games in an online setting. At a high level, the system allows a remote "client" user to interface with a Wii console over a network connection while a local "host" user interfaces with the Wii console over a native Bluetooth connection as shown in Figure 1.1.

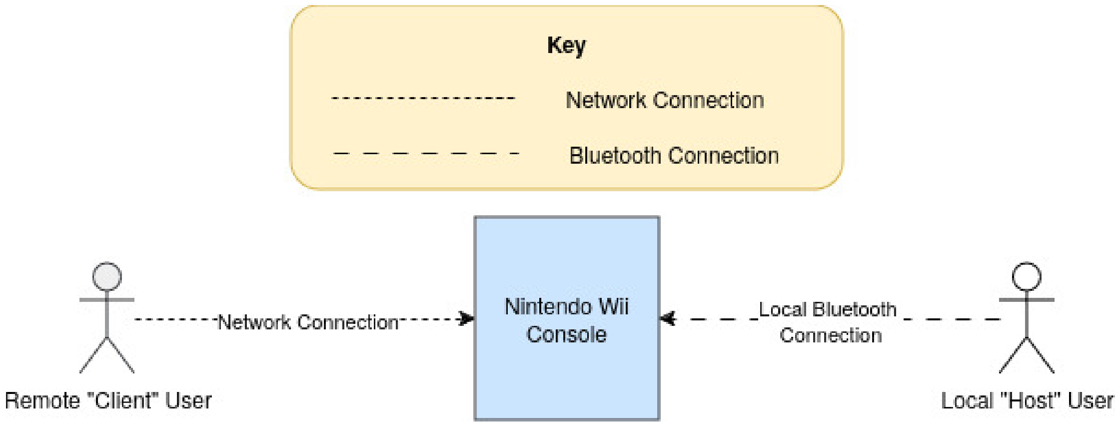

Figure 1.1.: Simple Overview of the System

However, this simple overview does not capture the complexity of the system. The remote user needs to be able to see and hear the game as well as use a physical Wii Remote to control the game. The system – as shown in Figure 1.2 – consists of three main components: a capture and streaming system, a controller input relay system, and a Wii Remote emulator. The capture and streaming system is responsible for capturing the Wii's video and audio output and streaming it to remote players through the host machine. The controller input relay system transmits the Wii Remote's controller data from the client machine to the host machine over a low-latency network connection. The Wii Remote emulator, running on the host machine, receives the controller data and emulates the remote user's Wii Remote input.

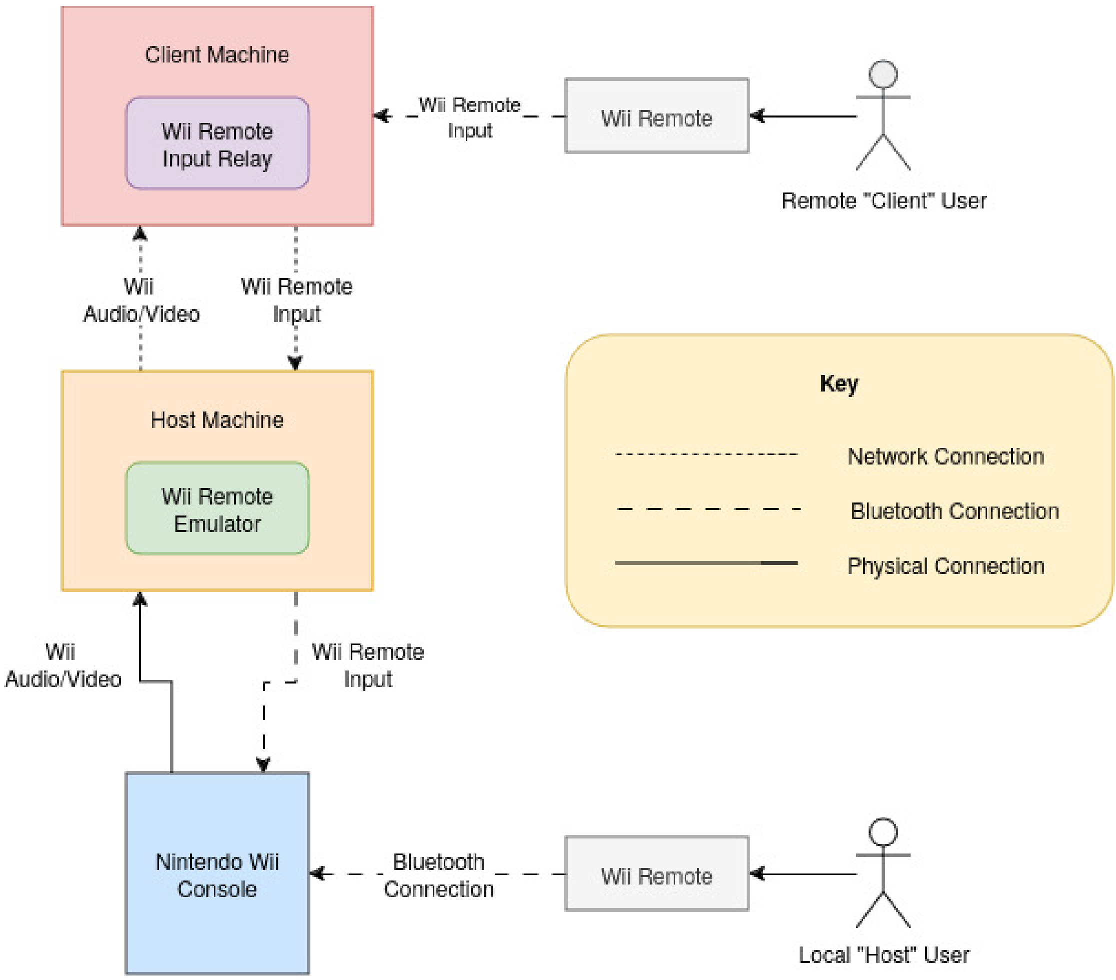

Figure 1.2.: System Architecture and Data Flow

The key objectives of the project are to:

1. Develop a system to capture and stream the Wii's video and audio output to remote players.

2. Develop a system to relay the Wii Remote's controller data over a low-latency network connection.
3. Evaluate the system's performance and user experience in a real-world setting.

This report will introduce discussions of the technical design, implementation challenges, and testing procedures that collectively contribute to a solution aimed at revitalising retro gaming experiences. The subsequent chapters present an in-depth analysis of the system architecture, design decisions, and the experimental validation of the proposed solution.

2. Context Survey

This section surveys the broader context of the project by reviewing the historical background, key technologies, and recent initiatives that align with the aim of the project. In particular, it examines the Nintendo Wii's ecosystem, the evolution of its input devices, and the supporting technologies that have enabled both commercial and experimental adaptations.

2.1. The Nintendo Wii and its Ecosystem

Nintendo released the Wii in 2006, quickly earning acclaim for its innovative motion-based controls and engaging game library. Central to its appeal was the Wii Remote (Wiimote), a wireless controller equipped with accelerometers, infrared sensors, and traditional button inputs. These features enabled intuitive, physical interactions, helping to bridge the gap between digital gameplay and physical movement. Over time, the Wii's local multiplayer format – often characterised by split-screen or shared-screen experiences – solidified its reputation as a console focused on communal play. In 2014, Nintendo shut down the Wii's online services[20], officially ending support for online multiplayer. As a result, third-party solutions, like *Nintendo Wii over IP*, remain the only way to play Wii games online.

2.2. Relevant Hardware and Software Technologies

The project draws on a range of hardware and software technologies to achieve its objectives. Figure 2.1 shows a modified version of Figure 1.2 that highlights where the key technologies fit into the system architecture.

Key technologies include:

`WiimoteEmulator`[1]

The open-source `WiimoteEmulator` project on GitHub emulates Wii Remote signals, allowing a real Wii console to interface with a computer acting as an external controller. By replicating the Wiimote's communication protocol, it lays the groundwork for experimenting with alternative input methods. For this dissertation, I extended a fork of `WiimoteEmulator` to accept infrared and accelerometer data over a network. This enhancement plays a crucial role in linking remote inputs with local emulation.

Bluetooth

Bluetooth is a wireless communication protocol that enables devices to exchange data over short distances. In the context of this project, Bluetooth facilitates the direct communication between the client machines and the physical Wiimotes as well as between the Wii Remote emulator running on the host machine and the Wii console.

Raspberry Pi

The Raspberry Pi serves as a versatile, low-cost computing platform that supports the integration of various peripherals and communication protocols. In this project, both the host and client machines are Raspberry Pi devices. The project uses Raspberry Pi devices due to their Bluetooth capabilities, support for Linux-based operating systems, and prevalence in the hobbyist community. Other platforms, such as the Arduino, ultimately proved unsuitable due to limited processing power and lack of support for the required software libraries.

`xwiimote` Library[25]

To capture real Wiimote input, the system uses the `xwiimote` library. Running on a Raspberry Pi, this library interfaces physical Wiimote hardware with software, enabling the system to capture and process motion and button data. A custom Python script then routes this data through the extended emulation system, ensuring correct interpretation of remote control signals.

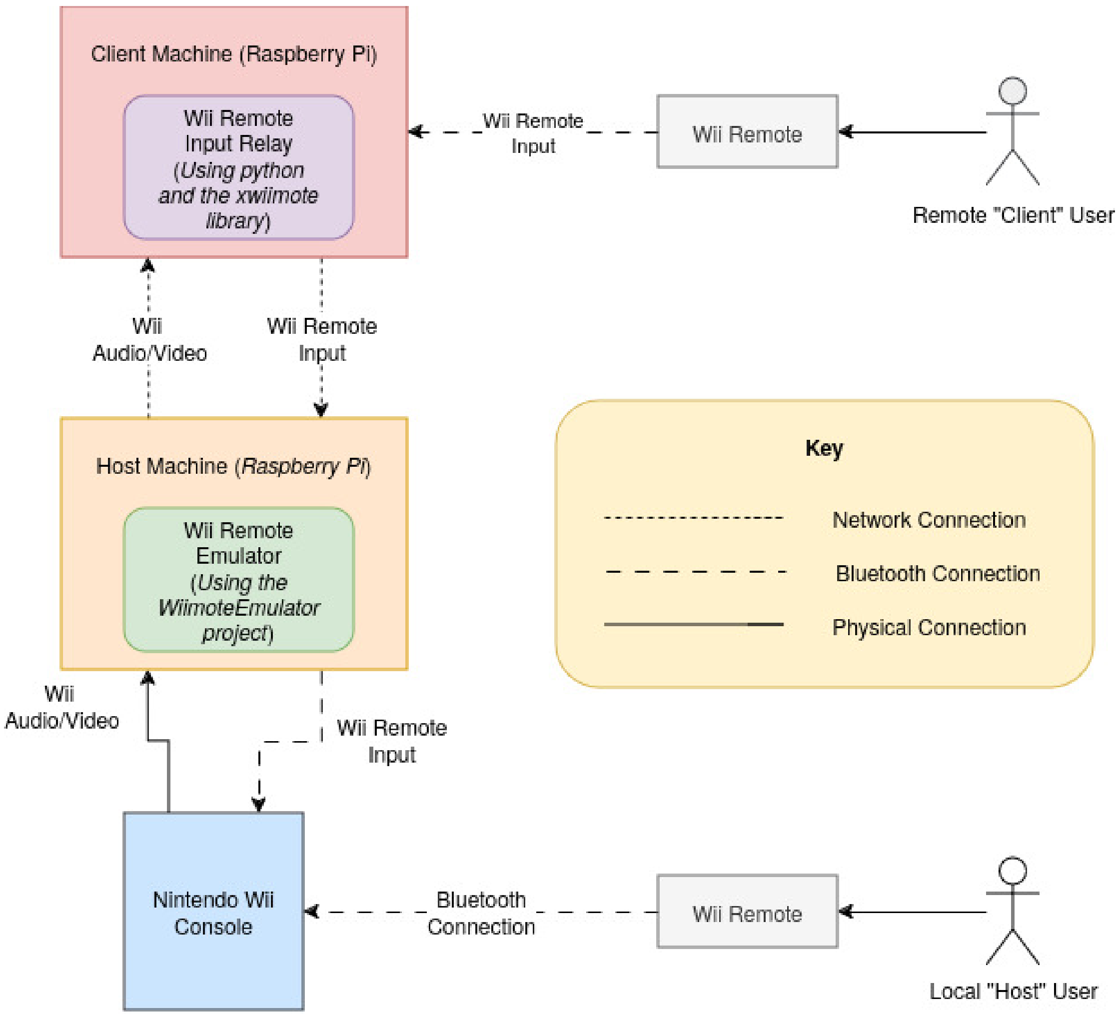

Figure 2.1.: Overview of the System with Key Technologies Linked

2.3. Recent Work and Similar Endeavours

The landscape of remote gaming and controller emulation is relatively niche, with few projects addressing the dual challenge of low-latency audiovisual streaming and precise controller input relay. Beyond the core `WiimoteEmulator` project, the following points are noteworthy:

Controller Emulation for Legacy Consoles

Prior research has focused on the emulation of input devices for legacy consoles in order to preserve or extend their operational lifespan. Such projects have enabled modern controllers to interface with older hardware, allowing users to play classic games without original peripherals. For example, the NES Hub utilises the unused expansion port on the NES console to add Bluetooth connectivity, supporting up to four modern wireless controllers without hardware modifications[14]. Extending to network-based control – transmitting sensor data such as IR and accelerometer signals remotely – is less common and represents a novel contribution of this work.

Remote Gaming Frameworks

Advancements in streaming protocols and low-latency communication have driven increased interest in remote gaming solutions. Cloud gaming services have transformed gaming by making high-quality games accessible without expensive hardware[3]. Projects like Tailscale transform the ideas of earlier solutions like Hamachi, providing secure, low-latency connections between devices[19]. However, these solutions are typically designed for modern games and platforms, rather than retro gaming experiences.

3. Requirements Specification

3.1. Functional Requirements

Req. ID	Title	Description	Rationale/Justification
FR-01	Video and Audio Capture and Streaming	Capture the Wii's video and audio outputs and stream them to remote players with minimal latency.	Enable remote players to experience the game in real-time by providing audiovisual data. If the remote user could not see or hear the game, the project would be non-functional.
FR-02	Controller Input Relay	Capture Wii Remote inputs – including motion data and button presses – and transmit them over a low-latency network connection.	Allow remote players to interact with the game in real-time by relaying controller inputs, without which the project would be non-functional.
FR-03	Wii Remote Emulation	Emulate the remote user's Wii Remote inputs through a local Bluetooth connection to the Wii console.	Enable remote players to control the game as if they were physically present by emulating controller inputs. Without this feature, the project would be non-functional.
FR-04	Performance	Operate under strict low-latency conditions by minimising delay and jitter using efficient processing and optimised data streaming protocols.	Ensure a responsive and immersive gaming experience by reducing latency and maintaining consistent performance, which is crucial for real-time gameplay. If latency is too high, the game will be unplayable.

Table 3.1.: Functional Requirements

3.2. Non-Functional Requirements

Req. ID	Title	Description	Rationale/Justification
NFR-01	Reliability and Robustness	Tolerate variations in network quality while maintaining continuous and stable operation under sub-optimal conditions.	Enhance system dependability by accommodating fluctuations in network performance, which is essential for uninterrupted gaming experiences.
NFR-02	Accessibility	Provide an intuitive interface and a straightforward setup process that enables easy connectivity and configuration without extensive technical effort.	Increase usability and lower technical barriers, allowing a broader range of users to enjoy the gaming experience without specialised assistance.
NFR-03	Evaluation	Conduct comprehensive testing in real-world environments and gather both quantitative performance metrics.	Validate overall system performance and quality through systematic assessment, facilitating informed decisions for iterative improvements and refinement.

Table 3.2.: Non-Functional Requirements

4. Software Engineering Process

Development followed the waterfall methodology[17]. This structured, sequential model fit the constraints of a fixed deadline and a single-developer workflow. Unlike Agile[2] or iterative models, which encourage continuous delivery and rapid prototyping, the waterfall approach provided distinct stages, enabling full completion and documentation of each phase before moving forward.

4.1. Software Development Approach

The waterfall model supported the project in two key ways:

1. **Fixed Deadline:** A single, non-negotiable deadline for delivering the complete system demanded a structured approach, and the sequential nature of the waterfall model addressed all project requirements methodically. Each phase built upon the previous one, enabling a well-planned progression from concept to final implementation.
2. **Single Developer Environment:** The lack of team coordination requirements made the overhead of Agile methods, such as complex coordination and iterative refinement, unnecessary. Instead, linear development – where requirements, design, implementation, and testing followed in sequence – offered clarity and focus, streamlining the process.

The process began with a detailed requirements specification, outlining goals and objectives for the system. The design phase followed, defining system architecture and planning integration points across subsystems.

4.2. Tools and Technologies

4.2.1. Programming Languages

The project uses a combination of C, Python, and Bash to target system-level performance, ease of integration, and automation.

C

Chosen for low-level hardware interaction and performance-critical components. The existing `WiimoteEmulator`, originally implemented in C, required extended functionality – best achieved by maintaining the same language to ensure compatibility, reduce overhead, and optimise latency. Consideration went to alternatives such as C++ or Rust; however, C++ introduced unnecessary abstraction for a project of this scope, and Rust's stricter compile-time checks and learning curve made it less suitable within the project's time constraints.

Python

Used for the input relay system. Python enabled rapid development of code to handle Bluetooth connectivity, input detection, and communication with the host machine. Its extensive libraries and readability made it ideal for writing higher-level control scripts. Languages like Java or Node.js fell out of consideration due to limited library support and less straightforward integration with low-level system components.

Bash

Utilised for system configuration and environment setup. A Bash script automates the configuration of the development environment, installation of dependencies, and compilation of the C codebase. This allowed reproducible builds and simplified development environment setup. While Python could also perform these tasks, Bash offered a more concise and native approach for shell-level automation.

4.2.2. Build and Deployment Tools

Version control through Git supported code integrity, traceability, and collaboration-readiness, even in a single-developer context. Tracking each change allowed for reliable rollback, documentation of evolution, and structured branching for experimental features. Git also ensured compatibility with deployment tools and safeguarded against data loss across environments. The ability to manage different stages of development through commits and tags proved essential for both incremental testing and final delivery.

5. Ethics

There are no ethical considerations for this project. The preliminary self-assessment form for ethics has been completed and it has determined that this project does not require an ethics application. See Appendix A: Ethics Approval Form for the signed ethics approval form.

6. Design

This chapter presents an in-depth discussion of the system's design, examining the overall architecture, the rationale behind key design decisions, and the unique aspects that distinguish this project. The design of the system is inherently modular, partitioning functionality into defined subsystems that interact through well-specified interfaces. This approach not only promotes ease of development and testing but also facilitates future expansion and maintenance.

6.1. System Architecture Overview

At a high level, the system comprises several loosely coupled components that work together to recreate a local multiplayer experience in a remote gaming context. Figure 1.2 illustrates the primary components and their interactions.

The primary subsystems include:

- **Controller Input Relay** A custom Python script acts as an intermediary, capturing input events (such as accelerometer data, IR signals, and button presses) using the `xwiimote` library[25] and translating them into a binary format. The script transmits these updates over UDP[22] to a Wiimote emulator running on the host machine.

- **Wiimote Emulator:** The emulator, derived from a fork[12] of the `WiimoteEmulator` project[1], interprets the incoming UDP packets and simulates the corresponding Wii Remote inputs, which are then relayed to the Wii console via a Bluetooth connection. This project extends the emulator's functionality to include streaming IR and accelerometer data over UDP, which is essential for emulating the Wii Remote's motion controls based on the remote player's inputs.

- **Audio and Video Streaming:** An audiovisual streaming component enables remote gameplay by capturing video and audio outputs from the Wii console. The system transmits these streams to a client machine via the host machine using the Real-time Transport Protocol (RTP)[16]. A significant design challenge was the trade-off between stream quality and latency, leading to a careful tuning of encoding parameters and RTP settings.

- **Automation and Deployment:** An automation script applies system configurations – such as loading kernel modules and installation of dependencies – consistently across devices. This not only simplifies the initial setup but also mitigates issues that might arise from manual configuration errors.

6.2. Controller Input Relay

6.2.1. Design Rationale

The relay is a custom Python script, leveraging Python's rapid development capabilities and its extensive library support. The integration with the `xwiimote` library was particularly attractive because it allowed for the efficient capture of diverse input events (e.g., accelerometer readings, IR signals, and button presses). This design decision provided a robust platform that could quickly adapts when new input modalities or requirements emerge.

6.2.2. Communication Strategy

The relay utilises fixed-length binary packets that transmit over UDP to minimise communication latency and relay each sensor event in as close to real time as possible. The choice of UDP, despite its lack of error-checking compared to TCP[21], was deliberate: it reduces overhead and is better suited for applications where timely data delivery is more critical than guaranteed packet delivery. All of the I/O operations in the relay are non-blocking, further enhancing responsiveness and ensuring that the system remains highly interactive.

6.2.3. Modularity and Extensibility

Modularity was a guiding principle in the relay's design. By mapping all input types to a unified binary format, the system remains adaptable. New sensors, like Wii MotionPlus[23] or alternate controllers, like nunchucks, can integrate without significant changes to the core architecture. This design supports future enhancements while maintaining a clear separation between data acquisition, processing, and transmission.

6.3. Wiimote Emulator

6.3.1. Leveraging Established Codebase

Instead of developing a Wii Remote emulator from scratch, the project builds upon the `WiimoteEmulator` project. Building a new Wii Remote emulator would have been too time-consuming and risked introducing bugs. One benefit of building an emulator from scratch would be the ability to minimise the latency between the emulator and the Wii console – something that the `WiimoteEmulator` project didn't focus on. However, the latency introduced by the emulator doesn't make the games unplayable, as shown in the Playlability Analysis section. Adapting an existing project allowed for a focus on design improvements rather than foundational re-implementation.

6.3.2. Enhanced Input Simulation

Extensions to the emulator support streaming IR and accelerometer data over UDP. This feature enables the remote player's inputs to be accurately reflected in the Wii console's gameplay. The original `WiimoteEmulator` project is able to emulate IR and accelerometer data but only through a mouse and keyboard interface. In addition, the original project calculates all the IR and accelerometer data in small movements, i.e. there is no way to quickly snap from one end of the screen to the other. Thus, the design improvements of the `WiimoteEmulator` project centre around fixing these shortcomings.

6.4. Audio and Video Streaming

6.4.1. Protocol Selection and Trade-offs

Adopting the Real-time Transport Protocol (RTP) leverages its robust framework for live media streaming. RTP's design supports low-latency transmission, which is essential for synchronising audiovisual output with user inputs. The design process involved carefully weighing the trade-offs between achieving high media quality and minimising transmission delay, with a preference for a configuration that favours timely delivery over maximal resolution.

6.4.2. Use of FFmpeg and FFplay

The system relies on FFmpeg[6] and FFplay[8] for capturing and streaming audiovisual data. FFmpeg's extensive codec support and FFplay's real-time playback capabilities make them well-suited for this application. The design leverages these tools to ensure seamless audiovisual streaming, with FFplay providing a low-latency display of the Wii's output on the client machine. Notably, this requires remote users to use FFplay, or similar tools that integrate with FFmpeg, to view the game stream. This restricts users to a possibly unfamiliar interface, but ensures a consistent experience across all clients.

6.5. Automation and Deployment

6.5.1. Centralised Configuration Management

A comprehensive setup script manages critical configuration task – such as loading essential kernel modules, updating Bluetooth settings, and installing necessary libraries. Centralising these tasks into an automated process reduces the risk of inconsistencies and ensures that every deployment starts from a known baseline.

6.5.2. Accessibility and Usability

The automation script priorities user-friendliness and accessibility. By abstracting complex setup procedures into a single command, the script lowers the barrier to entry for users who may lack technical expertise. This design choice aligns with the project's goal of providing an accessible and intuitive remote gaming experience meant for a broad audience.

6.5.3. Scalability and Future-Proofing

A key design decision was to ensure that the deployment process could scale with the system. The automation script is easy to extend and can incorporate new configuration steps when new features or requirements arise. This foresight in design allows the system to evolve without necessitating a complete overhaul of the deployment process, thereby supporting long-term scalability and maintainability.

6.6. Unusual and Innovative Design Features

6.6.1. Responsive Motion Control Emulation

The system's motion control emulation is a novel feature that enables remote players to interact with the game in real-time. By streaming IR and accelerometer data over UDP, the system accurately reflects the remote player's inputs on the Wii console. Allowing a remote user to control the game as if they were physically present is a unique design feature that enhances the system's immersive qualities.

6.6.2. Local Play Game Features

Other third party projects, like `wiimmfi`[24], have focused on replicating the functionality of the Wii's original online services. However, many Nintendo Wii games have specific gameplay features that require local multiplayer or never supported online play – such as Wii Sports. This project's focus on enabling local multiplayer experiences over IP is a unique design feature that caters to a broader range of games and plugs a gap in the existing ecosystem.

6.6.3. Automated Environment Configuration

Automates device setup by configuring kernel modules, adjusting Bluetooth configuration files, and installing necessary dependencies through a comprehensive script. This minimises configuration errors and drastically reduces setup time, ensuring that all devices operate from a consistent baseline. This design feature simplifies the deployment process, making the system accessible to users with varying technical expertise. The focus on accessibility and usability is novel in the context of remote gaming systems, where complex configurations can be a significant barrier to entry[5].

7. Implementation

This chapter details the practical development and testing of the system. It focuses on the integration of various hardware and software components, the novel modifications made to existing projects, and the challenges encountered along the way. The discussion covers the connection setup between the Wii Remote and Raspberry Pi, the streaming of audiovisual data, the extension of Wii Remote emulation, and the creation of a Python-based input relay.

7.1. Establishing Wii Remote Connectivity

7.1.1. Linux Driver Configuration

One of the initial challenges was to reliably connect the Wii Remote to the Raspberry Pis. The first step to allow a Wiimote to connect to a Linux machine is enabling the Linux driver for Wiimotes using the following command:

```
modprobe hid-wiimote
```

Running the following command ensures that this driver loads automatically at boot:

```
echo hid-wiimote | sudo tee /etc/modules-load.d/wiimote.conf
```

The automated device setup script incorporates this step to ensure that the driver is always loaded correctly.

7.1.2. Bluetooth Pairing and Connection

The next step is to pair the Wii Remote with the Raspberry Pi via Bluetooth. As Bluetooth requires authorisation to connect two wireless peers, simply loading the kernel driver is not enough. The Wiimote needs to associate with the Wiimote with the client machine by trusting it and then establish a connection.

Using the `bluetoothctl`[4] command-line tool, to pair the Wii Remote with the Raspberry Pi, follow these steps:

1. Run `bluetoothctl` to enter the Bluetooth control interface.
2. Enter `scan on` to start scanning for Bluetooth devices.
3. Immediately press the **1** and **2** buttons simultaneously on the Wii Remote to put it in pairing mode.
4. In the list of discovered bluetooth devices, find the Wii Remote and note its MAC address.
5. Put the Wii Remote into pairing mode and enter `trust <Wii Remote MAC address>` to trust the Wii Remote.
6. Put the Wii Remote into pairing mode and enter `connect <Wii Remote MAC address>` to establish a connection.

If the connection is successful, one of the indicator LEDs on the Wii Remote will remain lit. The Wii Remote is now connected to the Raspberry Pi and ready for use.

7.2. Selection of Wii Remote Libraries and Addressing Bluetooth Issues

Evaluation of multiple libraries and tools for Wii Remote interfacing led to the selection of the xwiimote[25] library,, particularly for its Python bindings[26], which allowed for seamless integration into a Python script. During testing, an issue arose where the Wii Remote connected via Bluetooth but exhibited continuously flashing lights, with xwiimote failing to register inputs. Luckily this is a known issue[11] – modifying the Bluetooth configuration file at `/etc/bluetooth/input.conf` and adding the following line resolves this issue:

```
ClassicBondedOnly=false
```

This adjustment enabled proper pairing and stable operation of the Wii Remote. The setup script automates this configuration step to ensure consistent performance across devices.

7.3. Audio and Video Streaming Optimisation

Streaming audio and video from the host Raspberry Pi to the client Pi posed a significant challenge, with a trade-off observed between media quality and latency. Higher quality streams resulted in high latency, while lower quality streams compromised user experience. The solution was to adopt the Real-time Transport Protocol (RTP) with carefully tuned broadcast and playback settings. Although further optimisations remain possible, this configuration currently offers a balanced compromise between low latency and acceptable media quality.

7.3.1. `broadcast-rtp.sh` Script

The `broadcast-rtp.sh` script runs the following command:

```
ffmpeg -max_delay 0 -max_probe_packets 1 -f v4l2 -framerate 25 -video_size 720x576 -threads 1 -i /dev/video0 -vcodec libx264 -pix_fmt:v yuv420p -g:v 1 -preset ultrafast -tune zerolatency -crf 17 -max_delay 0 -fflags +nobuffer -flags low_delay -f rtp -muxdelay 0 rtp://192.168.20.20:42423
```

Table 7.1 explains the arguments used in the script in detail. See the official `ffmpeg` documentation for more information[7].

7.3.2. `play-rtp.sh` Script

The `broadcast-rtp.sh` script leverages `ffmpeg` to capture video from the connected camera device (`/dev/video0`), encode it in real time, and broadcast it over the Real-time Transport Protocol (RTP). This configuration is carefully tuned to minimise latency while still delivering acceptable video quality for interactive applications. The script runs the following command:

```
ffplay -vcodec h264 -max_delay 0 -analyzeduration 1 -protocol_whitelist file,udp,rtp -fflags nobuffer -strict experimental -framedrop -flags low_delay -probesize 32 -vf setpts=0 -sync ext rtp-h264.sdp

```

Table 7.2 explains the arguments used in the script in detail. See the official `ffplay` documentation for more information[9].

Argument	Explanation
‘-max_delay 0‘	Processes incoming packets immediately without waiting for additional data, reducing buffering delay.
‘-max_probe_packets 1‘	Limits probing to one packet to accelerate stream setup.
‘-f v412‘	Specifies the input format provided by the video capture device; ensures compatibility with the incoming video data.
‘-framerate 25‘	Sets the frame rate to 25 fps to balance smooth playback and processing load.
‘-video_size 720x576‘	Defines the video resolution at 720x576 pixels.
‘-threads 1‘	Restricts encoding to a single thread for consistent low-latency performance.
‘-i /dev/video0‘	Specifies the video capture device as /dev/video0.
‘-vcodec libx264‘	Uses the H.264 encoder (libx264) for video encoding.
‘-pix_fmt:v yuv420p‘	Enforces the yuv420p pixel format for wide compatibility with RTP receivers.
‘-g:v 1‘	Forces every frame to be a keyframe, which improves stream recovery and reduces latency.
‘-preset ultrafast‘	Minimises processing time by using the fastest encoding preset.
‘-tune zerolatency‘	Optimises the encoder for real-time transmission by reducing latency.
‘-crf 17‘	Sets the constant rate factor to balance quality and compression efficiency.
‘-fflags +nobuffer‘	Disables internal buffering in ffmpeg to minimise latency.
‘-flags low_delay‘	Enables low-delay processing modes for reduced end-to-end latency.
‘-f rtp‘	Specifies the output format as RTP.
‘-muxdelay 0‘	Sets the multiplexing delay to zero, ensuring immediate packetisation of the stream.
‘Output Destination: rtp://192.168.20.20:42423‘	Designates the target IP address and port for the RTP stream transmission.

Table 7.1.: `broadcast-rtp.sh` Argument Explanations

Argument	Explanation
‘-vcodec h264‘	Specifies that the incoming stream is encoded in H.264, ensuring the correct decoder is used.
‘-max_delay 0‘	Eliminates buffering delays on the playback side, promoting real-time viewing.
‘-analyzeduration 1‘	Limits the analysis duration to speed up the startup process.
‘-protocol_whitelist file,udp,rtp‘	Whitelists required protocols (file, UDP, RTP) for reading the SDP file and handling the RTP stream.
‘-fflags nobuffer‘	Disables input buffering to reduce latency during playback.
‘-strict experimental‘	Allows usage of experimental features if necessary for stream compatibility.
‘-framedrop‘	Drops frames if decoding lags, keeping playback as close to real-time as possible.
‘-flags low_delay‘	Enforces low-delay processing to maintain alignment with the low-latency broadcast.
‘-probesize 32‘	Sets a minimal data probe size (32 bytes) to reduce initialisation time.
‘-vf setpts=0‘	Resets presentation timestamps to help maintain sync in a low-latency context.
‘-sync ext‘	Synchronises playback using an external clock for accurate timing with the RTP stream.
‘Input File: rtp-h264.sdp‘	Provides the SDP file that describes the RTP stream parameters, essential for interpreting the incoming data correctly.

Table 7.2.: `play-rtp.sh` Argument Explanations

7.4. Python Script for Input Relay

A core component of the project is the custom Python script (`wiimote_to_emulator.py`) that serves as a bridge between the physical Wii Remote and the emulation backend running on the host Raspberry Pi. This script leverages the `xwiimote` Python bindings to interface directly with the Wii Remote hardware, continuously monitoring for various input events and relaying them to the Wii Remote Emulator via UDP.

7.4.1. Wiimote Connection and Monitoring:

The script initialises a `xwiimote` monitor to detect when a Wii Remote connects to the client machine. When the input relay detects a device, it creates an interface with the device and opens it for both reading and writing. This method of setting up the Wii Remote connection differs from the traditional approach other libraries use, which typically use the device files located in the `dev` directory automatically without the need for manual setup.

7.4.2. Non-Blocking I/O and Event Polling:

Using the `select.poll()` mechanism, the script sets up non-blocking I/O on the Wii Remote's file descriptor. This allows the script to efficiently wait for input events without stalling the main event loop. When the script detects events, the script calls `dev.dispatch(evt)` to process them.

7.4.3. Event Processing and Binary Packet Formation:

Depending on the event type, the script processes the data accordingly:

Accelerometer Events

When the machine receives an accelerometer event (identified by `xwiimote.EVENT_ACCEL`), the script retrieves the raw accelerometer values from channel 0. It then normalises these values (using a custom scaling and offset transformation) and packs them into a binary packet with the header `0x02`. The binary format is:

`[1 byte event type (0x02)] + [4 bytes float ax] +`
`[4 bytes float ay] + [4 bytes float az]`

IR Events:

For IR events (identified by `xwiimote.EVENT_IR`), the script retrieves the IR coordinates and normalises them to a [0,1] range. It then packs the data into a binary packet with header `0x01`:

`[1 byte event type (0x01)] + [4 bytes float x] +`
`[4 bytes float y] + [4 bytes float z]`

Button (Key) Events:

The script also processes key events (e.g., pressing the `+`, `-`, `HOME`, `A`, and `B` buttons). The input relay handles these by sending text-based command packets (e.g., `"button 1 WIIMOTE_PLUS"`) over UDP to indicate button press and release actions.

7.4.4. UDP Communication:

The script creates a UDP socket to transmit the binary (and text-based) update packets to the Wii Remote Emulator. The user provides the IP address and port of the host machine running the Wii Remote emulator via command-line arguments. The script logs key actions and any errors using Python's built-in logging facilities, ensuring that debugging information is available during operation.

7.4.5. Robust Error Handling:

The script catches and logs exceptions (such as I/O errors during event dispatching). This approach ensures that transient errors do not break the event loop, thereby maintaining reliable real-time transmission of control data while providing feedback to the user. The script also handles keyboard interrupts gracefully, ensuring that the Wii Remote connection is properly closed before exiting.

7.5. Wii Remote Emulation Enhancements

The system's final major component is the emulation of the Wii Remote on the host Raspberry Pi. The implementation utilises a modified version of the `WiimoteEmulator` originally developed by Ryan Conrad[1] (known as rnconrad on GitHub). `WiimoteEmulator` is able to emulate a bluetooth wii controller in software, enabling control of the wii through various different input devices such as a keyboard, mouse, or text commands over a network.

More specifically, this project builds on a fork of the project by JRogaishio[12] as it fixes two critical bugs. The first bug is that the ip command in the original project was not working due to an index error. The second bug is that the original project was not compiling due to a call to `graceful_disconnect()` which was not defined.

The version built in this project[10] builds upon this fork by adding support for transmitting IR and accelerometer data through the IP socket interface.

7.5.1. Enhancements and Challenges

IR Emulation

IR emulation in the system is responsible for generating the infrared (IR) sensor data that the Wii Remote expects when pointing at a sensor bar. The implementation leverages the functions defined in `motion.c`. First, the function `look_at_pointer` computes a transformation matrix based on normalised pointer coordinates (`pointer_x` and `pointer_y`). This matrix defines the orientation of the emulated Wii Remote relative to a virtual screen, factoring in physical dimensions (e.g., screen width, sensor bar width) and viewing distance.

Next, `set_motion_state` uses this transformation to compute two sensor points (`sensor_pt0` and `sensor_pt1`). A custom perspective projection matrix (`make_cam_projection_mat`) projects these points into a normalised coordinate system. After normalising the homogeneous coordinate, the resulting positions map to the resolution expected by the Wii Remote (typically a range of 0-1023 in x and 0-767 in y). The size of each IR object is also computed based on the depth component (`z`) of the projected points, simulating the apparent size changes of IR sources with distance. This process ensures that the emulated IR data closely mimics the signals generated by a physical sensor bar.

Accelerometer Emulation

Accurately emulating the Wii Remote accelerometer involves reading accelerometer data transmitted from the client machine over the network, processing this data, and effectively injecting it into the emulated Wii Remote state on the host machine. This component is critical for games that heavily rely on motion controls, such as racing and sports games, which interpret device orientation and motion through accelerometer inputs.

The Wii Remote Emulator captures the binary accelerometer packets using a non-blocking UDP socket interface implemented in `input_socket.c`. Upon receiving a packet, the emulator extracts the floating-point acceleration values using a network-to-host float conversion function `ntohf()`. Specifically, the following procedure occurs within the emulator's UDP socket polling loop:

1. Check if the packet starts with the correct identifier byte (`0x02`) and has the correct packet size (13 bytes).
2. Extract the network-order binary floats representing accelerations along the three axes.
3. Convert these network-order floats to host-order floats using `ntohf()`.

These extracted values are then assigned directly into the Wii Remote emulator's state structure:

```
state->usr.accel_x = event.analog_motion_event.x;
state->usr.accel_y = event.analog_motion_event.y;
state->usr.accel_z = event.analog_motion_event.z;
```

The input relay normalises the raw accelerometer data before transmitting it to the emulator. Thus, the Wii Remote emulator can directly use these values to update the emulated Wii Remote's accelerometer state.

Latency

Latency is a critical performance metric when determining the playability of video games. In general, the lower the latency, the more responsive the game feels to the player. As such, minimising latency across the system was a key focus during development. The system employs several strategies to reduce latency and improve responsiveness, including:

- **Non-blocking I/O:** In the `input_socket.c` file, UDP sockets use the `SOCK_NONBLOCK` flag to ensure that the system can continuously poll for new input events without stalling on network reads. This approach is essential for maintaining responsiveness.
- **Optimised Data Pipelines:** The system uses lightweight binary protocols for both IR and accelerometer updates. Sending fixed-length packets (e.g., 13-byte packets for IR and accelerometer data) reduces the overhead associated with parsing and error checking, improving overall throughput.
- **Increase Polling Frequency:** The original `WiimoteEmulator` project uses a 20ms polling interval for reading input events. By decreasing this value, thus increasing the polling frequency, the system can respond more rapidly to incoming data, reducing the overall latency of the system.

Despite these efforts, some latency issues remain due to the inherent latency present in the `WiimoteEmulator`. Further work on the `WiimoteEmulator` project should aim to address these challenges and improve the overall responsiveness of the system.

7.6. Automation of Device Setup

To streamline the deployment process, the project includes a device setup script (`setup.sh`). This script requires administrative privileges (sudo) and automates several critical configuration tasks, including:

- Loading necessary kernel modules.
- Editing system files (such as /etc/bluetooth/input.conf) to adjust Bluetooth settings.
- Configuring environment variables and export paths for library dependencies.
- Installing `xwiimote` and its Python bindings.
- Downloading and compiling the custom Wii Remote Emulator.

By automating these tasks, the setup script minimises manual configuration errors and ensures a consistent environment across multiple devices.

8. Evaluation

This chapter evaluates the system with respect to the original objectives, and it critically compares the projects approach to related work in the field. Specifically, it assesses the system's playability, identifies challenges encountered during development, and outlines solutions to these issues. The chapter also discusses the limitations of the system and suggests areas for future work.

8.1. Playability Analysis

8.2. Challenges and Solutions

IR Sensor Emulation

At first, the IR emulation only mapped to the bottom half of the screen due to a scaling value error.

Originally, the vector for the three IR coordinates in the Wiimote emulators 3D space was as follows:

```
vec3 pointer_world = {(pointer_x − 0.5) * screen_width, (pointer_y −
    0.5) * screen_width / screen_aspect, −screen_distance};
```

Changing removing the constant -0.5 and the screen aspect ratio from the y-coordinate calculation fixed the issue:

```
vec3 pointer_world = {(pointer_x − 0.5) * screen_width, (pointer_y)*
    screen_width, −screen_distance};
```

By correcting this error, the IR sensor data was correctly positioned on the screen, allowing for accurate pointing and cursor control. This fix was crucial for maintaining the playability of IR-dependent games.

Audiovisual Streaming

The audiovisual streaming component is a critical element in delivering an engaging remote gameplay experience. The system utilises the Real-time Transport Protocol (RTP) as its backbone for transmitting both video and audio streams. RTP is well suited to handling real-time data over IP networks, where timely delivery is as important as data integrity.

To minimise delay while preserving media quality, the streaming pipeline relies on the open-source `ffmpeg` framework. The broadcasting script (`broadcast-rtp.sh`) is finely tuned with parameters that target a reduction in buffering and processing overhead. For example, flags such as `-max_delay 0` and `-fflags +nobuffer` ensure immediate processing of incoming packets without accumulating extra data. Similarly, the `-preset ultrafast` and `-tune zerolatency` options accelerate encoding, even if this occasionally comes at the expense of compression efficiency.

On the client side, the playback script (`play-rtp.sh`) mirrors these optimisations by disabling internal buffering (`-fflags nobuffer`) and enabling low-delay decoding strategies (`-flags low_delay` and `-framedrop`). The inclusion of a dedicated SDP file further aids in accurately interpreting the stream parameters, reducing the initial synchronisation overhead.

The combined effect of these settings is a streaming setup that strikes a balance between media fidelity and transmission speed. Although higher quality settings tend to increase processing delays, the current configuration offers a compromise that preserves the authenticity of the Wii experience in remote environments.

Latency

Latency is a central performance metric in this project, as it directly impacts both the responsiveness of audiovisual feedback and the accuracy of input relays during gameplay. The overall system latency encompasses several stages: video capture, encoding, network transmission, decoding, and input processing. Each of these stages has been the focus of targeted optimisation efforts.

On the streaming side, the low latency results from a combination of high-speed encoding and aggressive buffering control. The use of `ffmpeg` with options like `-g:v 1` (forcing every frame as a keyframe) helps in rapid recovery and minimises delays during packet loss or re-synchronisation events. The broadcasting pipeline's non-buffered approach, as enforced by the `-max_delay 0` and

`-fflags +nobuffer` flags, keeps the end-to-end delay to a minimum, although this makes the system more sensitive to network jitter.

The input relay component, responsible for transmitting controller data from the Wii Remote to the emulator, further mitigates latency by employing non-blocking I/O and a lightweight binary protocol. By packaging sensor data (IR and accelerometer values) into fixed-length packets and transmitting them over UDP, the system avoids the overhead associated with more complex data formats. Additionally, reducing the polling interval in the `WiimoteEmulator` from the original 20 ms to a shorter duration allows the system to react more swiftly to user inputs.

Despite these efforts, some latency remains when compared to native Wii gameplay. The inherent delay introduced by network transmission and the processing limitations of the emulation environment contribute to a noticeable, albeit reduced, lag. For fast-paced or highly reactive gaming scenarios, even these minimised delays could affect user performance.

8.3. Limitations

Despite meeting the primary project objectives, several limitations remain:

1. **Peripheral Support:** The current implementation does not support nunchuck input, thereby limiting the scope of the emulated Wii experience.
2. **Scalability:** The system underwent testing with only a single remote player. The projects needs additional testing to verify its performance in multi-user scenarios.
3. **Latency:** Although the system successfully transmits inputs and streams audiovisual data, the emulator exhibits a noticeable latency compared to native Wii play. This latency could impact the experience in highly responsive, fast-paced games.
4. **Accelerometer Calibration:** The accelerometer emulation relies on hand-tuned parameters, which may not be optimally calibrated for all games. This could require game-specific adjustments to achieve the best user experience.

8.4. Reflection and Future Work

8.4.1. Evaluation of Objectives

Reflecting on the project's primary goals, it is evident that the system has successfully addressed the key challenges outlined in the Introduction.

Objective 1: Capturing and Streaming the Wii's Video and Audio Output

This objective focuses on the development and integration of automated scripts specifically designed to capture and stream the Wii's audiovisual data. The broadcasting script (`broadcast-rtp.sh`) plays a crucial role by leveraging the Real-time Transport Protocol (RTP) in conjunction with `ffmpeg` to encode and transmit video and audio data with minimal buffering. The script's carefully tuned parameters—including non-buffered input, forced keyframes, and low-latency encoding settings—ensure that the streaming pipeline efficiently manages the trade-off between media quality and transmission delay. On the client side, the playback script (`play-rtp.sh`) mirrors these optimisations to maintain synchronisation and reduce decoding overhead. Together, these scripts automate the processes required for capturing the Wii's output, ensuring that remote players receive a real-time and immersive multimedia experience.

Objective 2: Relaying the Wii Remote's Controller Data over a Low-Latency Network Connection

The second objective centres on the relay of controller data from the Wii Remote to the Wiimote emulator. First, a custom Python-based input relay system, built using the xwiimote Python bindings, captures real-time input events—including button presses, accelerometer data, and IR signals—and translates them into fixed-length binary packets. By adopting a lightweight binary protocol over UDP, the input relay minimises overhead, thereby ensuring low-latency transmission. In addition, enhancements made to the Wiimote Emulator enable it to accurately interpret these packets, thereby seamlessly integrating the physical controller inputs into the virtual environment. The combined approach of precise input relay and robust Wii Remote emulation confirms that the system meets the objective of providing responsive and accurate control for remote gameplay.

Objective 3: Evaluating System Performance and User Experience

The final objective involves a comprehensive playability analysis measuring system performance under realistic conditions. This analysis quantifies latency across different system stages, with detailed measurements assessing network latency during data transmission, the overhead introduced by the emulation process, and the interaction lag[18] – that is, the delay between a user's input and the corresponding on-screen response. The findings indicate that while the latency is within a playable range, a measurable interaction lag remains. This evaluation not only validates the system's capability to deliver a playable gaming experience but also highlights specific areas where further optimisation can bring the performance closer to that of native Wii gameplay.

8.4.2. Comparison with Related Work

WiimoteEmulator and Its Derivatives

The original `WiimoteEmulator` project by rnconrad and subsequent forks (e.g., JRogaishio's version) primarily focused on emulating the Wii Remote for local control using Bluetooth. In contrast, this work extends these foundations by implementing IR and accelerometer emulation over IP sockets. This system adapts the concept of Wii Remote emulation to enable remote gameplay – a feature not present in the original projects.

Input Relay Techniques

While several research efforts and projects have addressed low-latency input relay for gaming peripherals, many rely on text-based communication protocols or lack the integration of real-time sensor data. This approach, which utilises a binary protocol to transmit fixed-length packets, reduces overhead and improves performance, thereby offering a competitive edge in scenarios requiring rapid response times.

Audiovisual Streaming in Remote Gaming

In the broader context of remote gaming, solutions such as cloud gaming platforms have tackled the challenge of low-latency audiovisual streaming. However, these platforms often require substantial infrastructure and proprietary solutions. This system, by leveraging RTP for streaming and integrating it with the custom input relay, creates a unified framework that bridges both input and output channels in a manner that is both accessible and reproducible using open-source tools.

Overall System Integration

Compared to other projects that may focus solely on either streaming or input emulation, this work represents a holistic solution that aims to preserve the full multiplayer gaming experience. The integration of automated configuration, error handling, and modular software components differentiates this system, offering both flexibility and robustness.

8.4.3. Future Work

However, there are clear avenues for future improvement. Future iterations could explore the following areas:

Enhanced Peripheral Integration

Future iterations could include support for additional Wii peripherals, such as the nunchuck, to provide a more comprehensive emulation of the original gaming experience. This would require extending the existing input relay system to accommodate the unique features of each peripheral.

Scalability Testing

More extensive testing with multiple remote players is necessary to assess the system's performance under higher network loads and to refine the data relay mechanisms accordingly.

Latency Optimisation

Further research into reducing latency through improvements in the `WiimoteEmulator` fork, custom input relay program, and RTP streaming parameters could enhance the system's responsiveness and bring it closer to native Wii play.

Dynamic Calibration Techniques

Developing adaptive calibration algorithms for the accelerometer data could improve accuracy and tailor the emulation more effectively to different game genres and user preferences. This could involve machine learning techniques or game-specific calibration profiles.

9. Conclusion

This dissertation has presented a comprehensive approach to adapting a classic local multiplayer experience for the modern era by bridging the gap between the Nintendo Wii's original design and contemporary online gaming environments. The project's core achievement lies in the development of a system that revitalises the Wii's input and output interfaces – capturing audio and video with low latency, and relaying controller inputs over a network in real time.

Key achievements of the project include:

- The successful enhancement of the `WiimoteEmulator` project to support IR and accelerometer data, enabling the accurate emulation of Wii Remote inputs in a networked environment.
- The implementation of a novel controller input relay system that processes and transmits IR, accelerometer, and button data using a low latency binary protocol.
- The deployment of RTP-based audiovisual streaming techniques that balance media quality with the essential requirement of low latency, thereby preserving the authenticity of the Wii gaming experience.
- The development of automation scripts that streamline the setup process, reducing the potential for manual errors and ensuring a reproducible environment across multiple devices.

Despite these successes, the project also encountered significant challenges and limitations. Notably, some latency issues remain, the project has not been thoroughly tested with more than 1 remote player, and other traditional Wii input devices such as nunchucks are not supported . Additionally, tuning the accelerometer to cater to different game-specific requirements, such as those observed in titles like Mario Kart, continues to present challenges. These drawbacks highlight areas where further research and development are necessary.

Looking to the future, several directions could further enhance the system:

- **Optimisation of Latency:** Future work could focus on further reducing latency through further enhancements to the `WiimoteEmulator`, improved network protocols, or more efficient data processing.

- **Enhanced Accelerometer Calibration:** Refining the mathematical models and calibration procedures for accelerometer data may improve the accuracy and responsiveness of motion controls thus resulting in a more pleasant gaming experience.
- **Broader Platform Support:** Expanding the framework to support additional retro consoles or other legacy input devices could broaden the system's applicability and impact.
- **User Interface Improvements:** Enhancing the interface for setup and control, possibly through graphical tools or integrated diagnostics, would further improve usability and adoption.

In summary, this project demonstrates a viable method for adapting a legacy gaming system to modern, distributed gaming environments while preserving the original charm and social dynamics of local multiplayer play. The work not only provides a framework for further experimentation and improvement but also contributes to the ongoing dialogue about preserving and revitalising classic gaming experiences in the digital age.

References

[1] Ryan Conrad (rnconrad). *WiimoteEmulator*. https://github.com/rnconrad/WiimoteEmulator. (Visited on 19/03/2025).

[2] Atlassian. *What is Agile? — Atlassian*. https://www.atlassian.com/agile. (Visited on 25/03/2025).

[3] Ghazal Bangash, Pierre-Adrien Forestier and Loutfouz Zaman. *Cloud Gaming: Revolutionizing the Gaming World for Players and Developers Alike*. https://interactions.acm.org/archive/view/july-august-2024/cloud-gaming-revolutionizing-the-gaming-world-for-players-and-developers-alike. [Accessed 26-03-2025].

[4] *bluetoothctl(1) - Arch manual pages — man.archlinux.org*. https://man.archlinux.org/man/extra/bluez-utils/bluetoothctl.1.en. [Accessed 29-03-2025].

[5] Dirk Colbry. *Reducing the barrier to entry using portable apps*. https://www.researchgate.net/publication/254003063_Reducing_the_barrier_to_entry_using_portable_apps. [Accessed 28-03-2025].

[6] *FFmpeg - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/FFmpeg. [Accessed 28-03-2025].

[7] *ffmpeg Documentation — ffmpeg.org*. https://ffmpeg.org/ffmpeg.html#Options. [Accessed 29-03-2025].

[8] *ffplay Documentation*. https://ffmpeg.org/ffplay.htm. [Accessed 28-03-2025].

[9] *ffplay Documentation — ffmpeg.org*. https://ffmpeg.org/ffplay.html#Options. [Accessed 29-03-2025].

[10] Kieran Fowlds. *WiimoteEmulator - Kieran Fowlds' fork*. https://github.com/Kieranoski702/WiimoteEmulator. (Visited on 19/03/2025).

[11] jessienab. *xwiimote issue 109*. https://github.com/xwiimote/xwiimote/issues/109. (Visited on 19/03/2025).

[12] JRogaishio. *WiimoteEmulator - JRogaishio's fork*. https://github.com/JRogaishio/WiimoteEmulator. (Visited on 19/03/2025).

[13] Jasmine Katatikarn. *Online Gaming Statistics and Facts: The Definitive Guide (2024) — academyofanimatedart.com*. https://academyofanimatedart.com/gaming-statistics. [Accessed 26-03-2025].

[14] Andrew Liszewski. *You can add wireless controller support to the NES through its unused expansion port — theverge.com*. https://www.theverge.com/2024/10/9/24266019/nintendo-nes-expansion-port-retrotime-bluetooth-wireless-controller. [Accessed 26-03-2025].

[15] *Online Gaming Industry Trends Report 2025: Global Market to Reach $388.1 Billion by 2033, Driven by eSports, Cross-platform Gaming, In-game Purchases, Social Integration, AR/VR - ResearchAndMarkets.com — businesswire.com*. https://www.businesswire.com/news/home/20250325163198/en/Online-Gaming-Industry-Trends-Report-2025-Global-Market-to-Reach-%24388.1-Billion-by-2033-Driven-by-eSports-Cross-platform-Gaming-In-game-Purchases-Social-Integration-ARVR---ResearchAndMarkets.com. [Accessed 26-03-2025].

[16] *Real-time Transport Protocol - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Real-time_Transport_Protocol. [Accessed 28-03-2025].

[17] W. W. Royce. 'Managing the development of large software systems: concepts and techniques'. In: *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)*. Monterey, California, USA: IEEE Computer Society Press, 1987, pp. 328–338. ISBN: 0897912160.

[18] Volker Seeker. *Measuring QoE of Interactive Workloads and Characterising Frequency Governors on Mobile Devices — volkerseeker.com*. https://www.volkerseeker.com/publication/2014_IISWC_lagmarking. [Accessed 30-03-2025]. 2014.

[19] *Tailscale vs. Hamachi: A Modern VPN Replacement for Gaming and Collaboration — tailscale.com*. https://tailscale.com/blog/hamachi. [Accessed 26-03-2025].

[20] *Termination of Nintendo Wi-Fi Connection — nintendo.com*. https://www.nintendo.com/en-gb/Support/Wii/Setup-amp-Connect/Termination-of-Nintendo-Wi-Fi-Connection/Termination-of-Nintendo-Wi-Fi-Connection-859609.html. [Accessed 26-03-2025].

[21] *Transmission Control Protocol - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Transmission_Control_Protocol. [Accessed 28-03-2025].

[22] *User Datagram Protocol - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/User_Datagram_Protocol. [Accessed 28-03-2025].

[23] *Wii MotionPlus - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Wii_MotionPlus. [Accessed 28-03-2025].

[24] *Wiimmfi*. https://wiimmfi.de/. [Accessed 28-03-2025].

[25] xwiimote. *xwiimote*. https://github.com/xwiimote/xwiimote. (Visited on 19/03/2025).

[26] xwiimote. *xwiimote-bindings*. https://github.com/xwiimote/xwiimote-bindings. (Visited on 19/03/2025).

A. Ethics Approval Form

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self-Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your ethical considerations.
Tick one box

☐ **Staff Project**
☐ **Postgraduate Project**
☒ **Undergraduate Project**

Title of project

Nintendo Wii over IP

Name of researcher(s)

Kieran Fowlds

Name of supervisor (for student research)

Dr Tom Spink

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self audit has been conducted **YES** ☒ **NO** ☐

There are no ethical issues raised by this project

Signature Student or Researcher

Kieran Fowlds

Print Name

Kieran Fowlds

Date

26/09/2024

Signature Lead Researcher or Supervisor

Print Name

Dr Tom Spink

Date

30/09/24

This form must be date stamped and held in the files of the Lead Researcher or Supervisor. If fieldwork is required, a copy must also be lodged with appropriate Risk Assessment forms. The School Ethics Committee will be responsible for monitoring assessments.

Computer Science Preliminary Ethics Self-Assessment Form

Research with secondary datasets

Please check UTREC guidance on secondary datasets (https://www.st-andrews.ac.uk/research/integrity-ethics/humans/ethical-guidance/secondary-data/ and https://www.st-andrews.ac.uk/research/integrity-ethics/humans/ethical-guidance/confidentiality-data-protection/). Based on the guidance, does your project need ethics approval?

YES ☐ **NO** ☒

** If your research involves secondary datasets, please list them with links in DOER.*

Research with human subjects

Does your research involve collecting personal data on human subjects?

YES ☐ **NO** ☒

If YES, full ethics review required

Does your research involve human subjects or have potential adverse consequences for human welfare and wellbeing?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Will you be surveying, observing or interviewing human subjects?
Does your research have the potential to have a significant negative effect on people in the study area?

Potential physical or psychological harm, discomfort or stress

Are there any foreseeable risks to the researcher, or to any participants in this research?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Is there any potential that there could be physical harm for anyone involved in the research?
Is there any potential for psychological harm, discomfort or stress for anyone involved in the research?

Conflicts of interest

Do any conflicts of interest arise?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Might research objectivity be compromised by sponsorship?
Might any issues of intellectual property or roles in research be raised?

Funding

Is your research funded externally?

YES ☐ **NO** ☒

If YES, does the funder appear on the 'currently automatically approved' list on the UTREC website?

YES ☐ **NO** ☒

If NO, you will need to submit a Funding Approval Application as per instructions on the UTREC website.

Research with animals

Does your research involve the use of living animals?

YES ☐ **NO** ☒

If YES, your proposal must be referred to the University's Animal Welfare and Ethics Committee (AWEC)

University Teaching and Research Ethics Committee (UTREC) pages
http://www.st-andrews.ac.uk/utrec/