CS4099 - Nintendo Wii Over IP

Kieran Fowlds - 210018092

Supervisor: Dr. Tom Spink

27th 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[9], 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[12], the Wii no longer supports online multiplayer. This coupled with the fact that most people no longer play multiplayer games locally[7], 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.

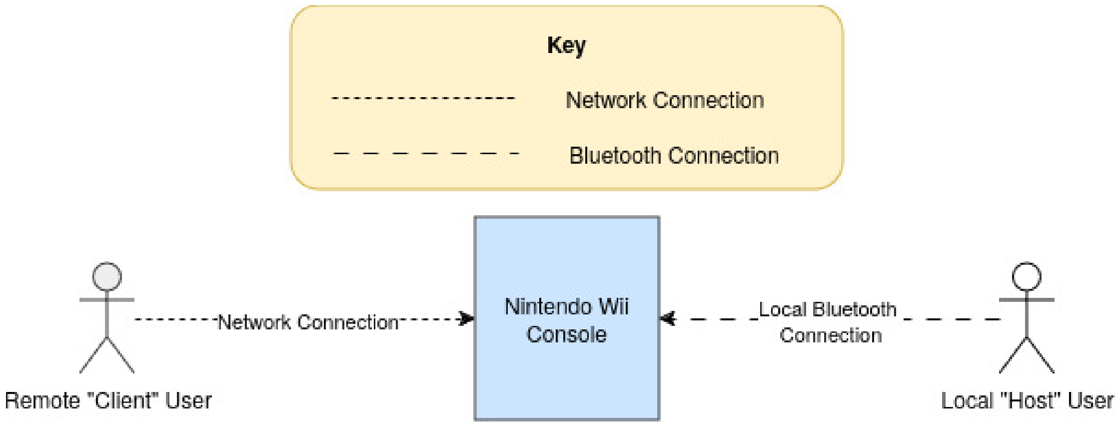

Figure 1.1.: Simple Overview of the System

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.

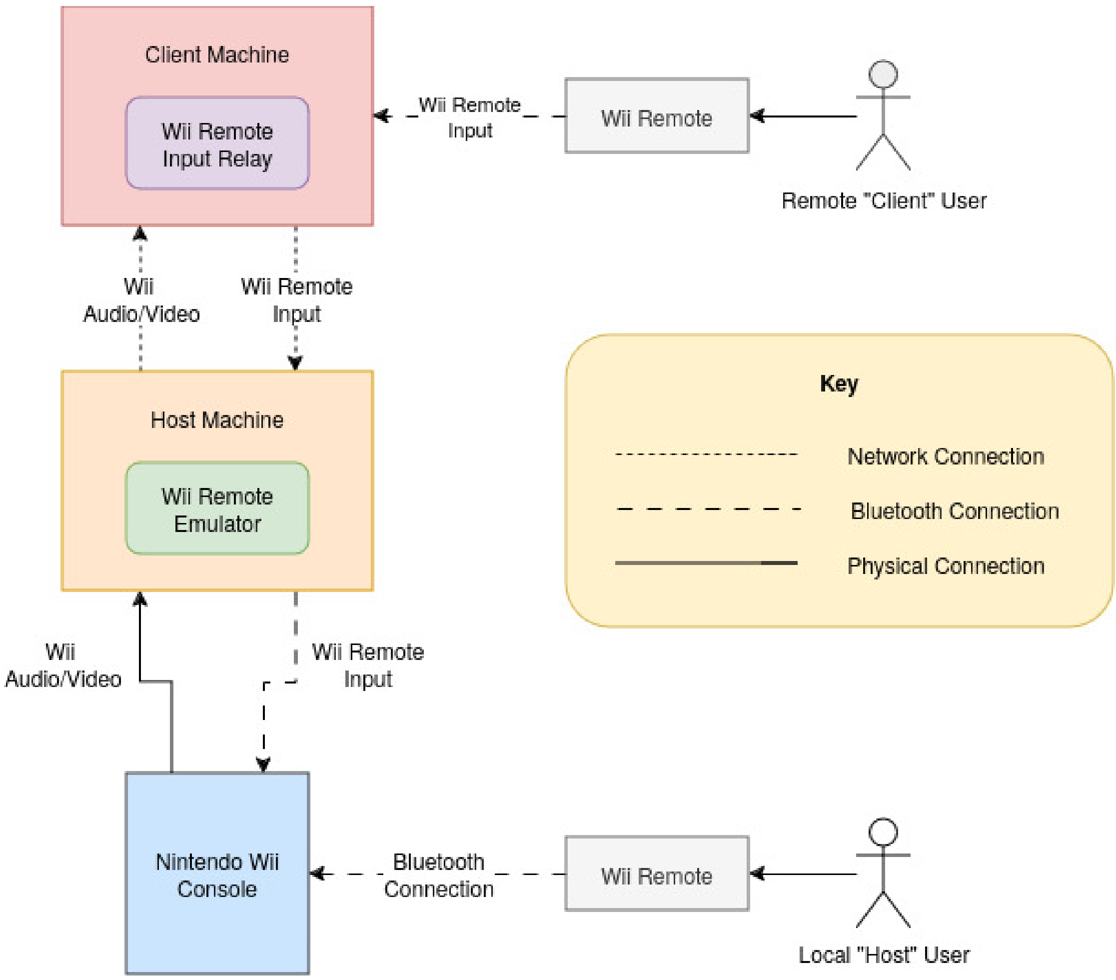

Figure 1.2.: System Architecture and Data Flow

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[12], 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[13]

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.

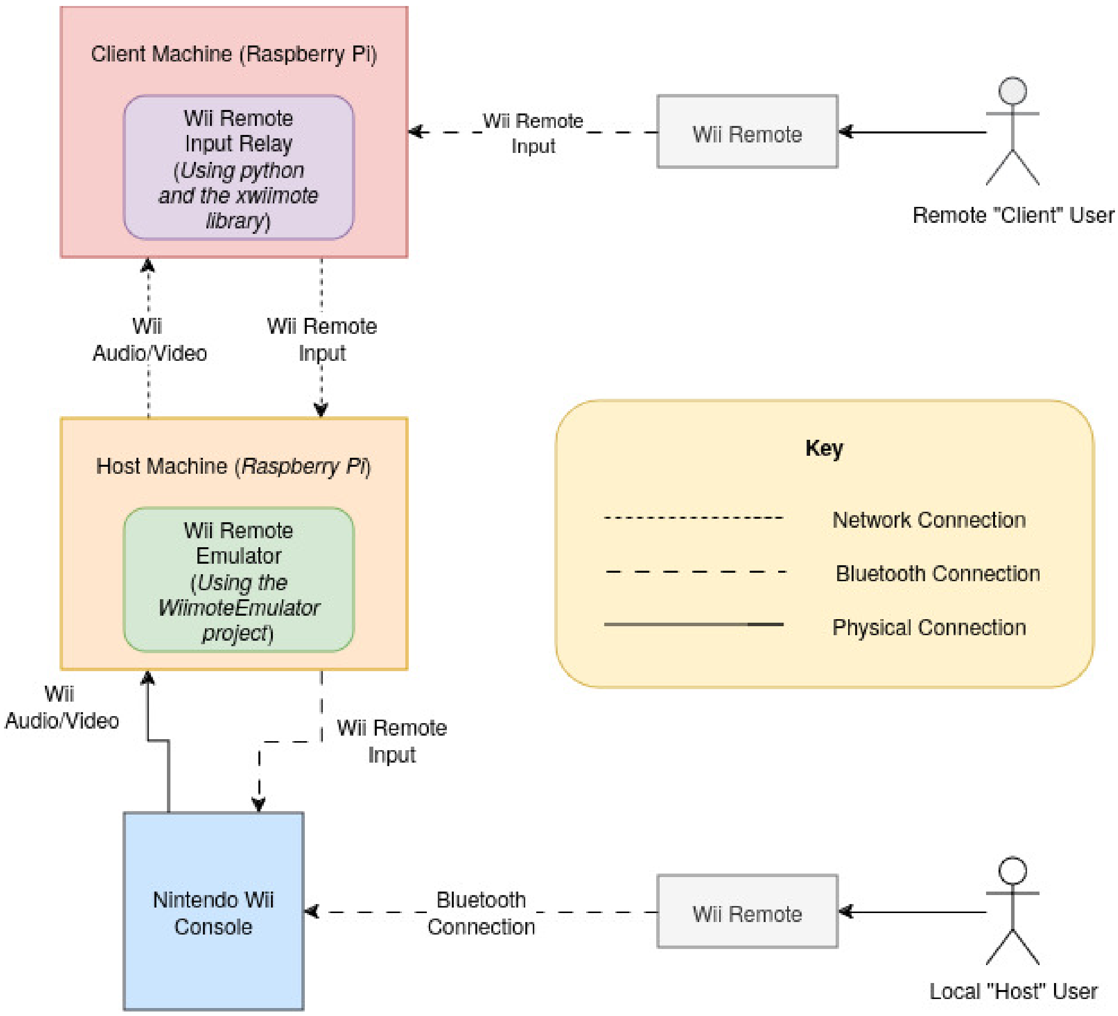

Figure 2.1.: Overview of the System with Key Technologies Linked

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[8]. 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[11]. 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[10]. This structured, sequential model fit the constraints of a fixed deadline and a single-developer workflow. Unlike Agile or iterative models[2], 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 clearly 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. 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[13] and translating them into a binary format. These updates are transmitted over UDP to a wiimote emulator that runs on the host Raspberry Pi.
- **Wiimote Emulator:** The emulator, derived from a fork[6] of the `WiimoteEmulator` project[1], has been extended to handle IR and accelerometer data, bridging the gap between physical inputs and the emulated control signals expected by the Wii. The emulator processes incoming UDP packets, updates its internal state, and generates the corresponding output signals.
- **Audio and Video Streaming:** To recreate the authentic gaming experience, the system includes an audiovisual streaming component. Using the Real-time Transport Protocol (RTP), the video and audio outputs from the host are captured and transmitted to a client device. 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 ensures that all system configurations—such as loading kernel modules and setting up environment variables—are applied consistently across devices. This not only

simplifies the initial setup but also mitigates issues that might arise from manual configuration errors.

6.2. Component-Level Design and Data Flow

The system's architecture emphasises clear data flow and modularity. Figure 1.2 illustrates the primary components and their interactions.

At the core of the design is the input relay mechanism, which operates as follows:

1. **Input Capture:** The Wii Remote's events are captured using the `xwiimote` library. Both analog (accelerometer, IR) and digital (button press) events are monitored continuously.
2. **Event Processing:** In the Python script, events are handled in a non-blocking manner using `select.poll()`. The script processes each event by normalising sensor data and mapping it into the expected range. For example, IR data is normalised to a [0,1] scale and then converted to a resolution that matches the Wii's requirements, while accelerometer data is similarly scaled.
3. **Packet Formation and Transmission:** Processed events are packaged into binary data packets. The design utilises fixed-length packets with a dedicated header byte to distinguish between different types of events (e.g., `0x01` for IR and `0x02` for accelerometer data). These packets are transmitted over UDP to the emulator, which interprets them to simulate the corresponding inputs.
4. **Emulation and Output:** The emulator on the host Raspberry Pi receives the UDP packets and integrates the data into its internal state. The emulation layer uses transformation routines to convert the incoming data into the simulated state of the Wii Remote, including generating IR positions and accelerometer readings.

6.3. Unusual and Innovative Design Features

Several aspects of the system's design stand out due to their innovative nature:

6.3.1. End-to-End Multiplayer Revival

The system is designed to recreate the local multiplayer experience of the Nintendo Wii in an online setting. By combining audiovisual streaming with real-time input relay, the project aims to provide a seamless and authentic gaming experience that captures the essence of the original console. The project is easy to setup and use due to the automation scripts removing the need for advanced technical knowledge. This holistic approach to reviving the multiplayer capabilities of the Nintendo Wii is unique and distinguishes the project from other remote gaming solutions.

6.3.2. Modular and Extensible Input Processing

The design of the input processing pipeline is highly modular. Different types of inputs (e.g., IR, accelerometer, button events) are handled in discrete sections of the code. This modularity allows for independent testing and future enhancements; for instance, additional sensor types or new control schemes can be incorporated with minimal changes to the overall architecture.

6.3.3. Automated Environment Configuration

Another unusual aspect of the design is the automated device setup. Recognising the complexity involved in configuring kernel modules, Bluetooth settings, and environment variables across multiple devices, an automation script was developed. This script ensures that all prerequisites for running the system are met without manual intervention, significantly reducing setup time and potential human errors.

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

One of the initial challenges was to reliably connect the Wii Remote to the Raspberry Pis. This was achieved by enabling the Linux driver for the Wii Remote using:

```
modprobe hid-wiimote
```

To ensure that this driver is loaded automatically at boot, the following command was run to add the wiimote drivers to the modules-load configuration:

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

This step was crucial for providing a persistent connection between the Wii Remote and the Raspberry Pi environment.

7.2. Selection of Wii Remote Libraries and Addressing Bluetooth Issues

After evaluating multiple libraries and tools for Wii Remote interfacing, the xwiimote[13] library was chosen, particularly for its Python bindings[14], 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[5] and could be resolved by modifying the Bluetooth configuration file at `/etc/bluetooth/input.conf` and adding the following line:

```
ClassicBondedOnly=false
```

This adjustment enabled proper pairing and stable operation of the Wii Remote.

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.

7.4. Wii Remote Emulation Enhancements

A core component of the project is the emulation of the Wii Remote on the host Raspberry Pi. This was implemented by adapting 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, allowing the wii to be controlled by many different input devices such as a keyboard, mouse, or text commands over a network.

A fork of the project by JRogaishio[6] was selected 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 researcher's version[4] further extends this fork by adding support for transmitting IR and accelerometer data over the IP socket interface.

7.4.1. 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 normalized pointer coordinates (`pointer_x` and `pointer_y`). This matrix defines the orientation of the emulated Wii Remote relative to a virtual screen, where physical dimensions (e.g., screen width, sensor bar width) and viewing distance are factored in.

Next, `set_motion_state` uses this transformation to compute two sensor points (`sensor_pt0` and `sensor_pt1`). These points are projected into a normalized coordinate system via a custom perspective projection matrix (generated by `make_cam_projection_mat`). Once the homogeneous coordinates are normalized, the resulting positions are mapped 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

Accelerometer emulation is handled primarily in the function `set_accelerometer` in `motion.c`. The goal is to simulate the Wii Remote's accelerometer readings based on its orientation. A fixed gravity vector (set as `{0, -1.0, 0}`) represents the effect of gravity on the remote. This vector is then transformed by the inverse-transposed 3x3 submatrix extracted from the Wii Remote's orientation matrix (computed in `look_at_pointer`).

The transformed acceleration values are clamped to a plausible range (between -3.4 and 3.4) to prevent unrealistic sensor readings. Finally, these values are scaled and shifted using the constants `accelerometer_zero` and `accelerometer_unit` to match the raw data format that the Wii Remote firmware expects. Although the accelerometer emulation code in `set_accelerometer` is currently commented out in some testing scenarios (with real input values handled in `input.c`), it provides a framework for generating synthetic accelerometer data based on the current pointer orientation. Fine-tuning of these calculations is ongoing, especially to ensure compatibility with specific game dynamics (e.g., the sensitivity required by Mario Kart).

Latency

Latency is a critical performance metric for both the audiovisual streaming and the emulation of controller inputs. Several design decisions were made to minimise latency across the system:

- **Non-blocking I/O:** In the `input_socket.c` file, UDP sockets are configured with 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. By sending fixed-length packets (e.g., 13-byte packets for IR and accelerometer data), the overhead associated with parsing and error checking is reduced. These binary packets are handled in `input_socket.c`, where functions such as `ntohf` convert network-order floats to host-order values with minimal delay.

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

7.5. Python Script for Input Relay

The system's final major component is a 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.

Key features and design details include:

7.5.1. Wiimote Connection and Monitoring:

The script initialises a `xwiimote` monitor to detect when a Wii Remote is connected. Once a device is found, it creates an interface with the device and opens it for both reading and writing. This setup is essential to capture both analog events (e.g., accelerometer and IR data) and digital button presses.

7.5.2. 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 events are detected, the script calls `dev.dispatch(evt)` to process them.

7.5.3. Event Processing and Binary Packet Formation:

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

Accelerometer Events

When an accelerometer event is received (identified by `xwiimote.EVENT_ACCEL`), the script retrieves the raw accelerometer values from channel 0. It then normalizes 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 normalizes 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). These are handled by sending text-based command packets (e.g., `"button 1 WIIMOTE_PLUS"`) over UDP to indicate button press and release actions.

7.5.4. UDP Communication:

A UDP socket is created to transmit the binary (and text-based) update packets to the Wii Remote Emulator. The target emulator's IP address and port are provided 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.

7.5.5. Robust Error Handling:

Throughout the script, exceptions (such as I/O errors during event dispatching) are caught and logged. This approach ensures that transient errors do not break the event loop, thereby maintaining reliable real-time transmission of control data.

7.6. Automation of Device Setup

To streamline the deployment process, a device setup script (`setup.sh`) was developed. 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. The chapter also discusses the limitations of the system and suggests areas for future work.

8.1. Playability Analysis

8.2. Challenges and Solutions

During development, several significant challenges emerged, each addressed with innovative solutions.

Input Relay and Data Synchronisation

Integrating the `xwiimote` library with the modified `WiimoteEmulator` fork presented challenges in synchronising accelerometer and IR data. Custom matrix transformations in `motion.c` and hand-tuned calibration routines ensured that the emulated signals closely replicated the physical Wii Remote behaviour. Additionally, the adoption of a binary protocol for transmitting sensor data reduced overhead and improved overall system responsiveness.

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

Balancing high-quality streaming with low latency was addressed by careful tuning of RTP parameters. Iterative testing of the `broadcast-rtp.sh` and `play-rtp.sh` scripts resulted in a workable compromise between video quality and responsiveness.

Latency Reduction

While the system successfully relayed input data and streamed audiovisual content, latency remained a persistent challenge. The use of RTP for streaming and a custom binary protocol for input relay helped minimise delays, but further optimisation is needed to bring the system closer to native play responsiveness.

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 has been tested with only a single remote player. Additional testing is needed 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

As stated in the Introduction chapter, the key objectives of the project were:

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.

Reflecting on the project in respect to these objectives, it is clear that the project successfully fulfils all three objectives. The system developed is capable of capturing and streaming the Wii's video and audio output to remote players, relaying the Wii Remote's controller data over a low-latency network connection, and has been evaluated in a real-world setting. The system has been tested in a controlled environment, and the evaluation has shown that the system is capable of providing a playable experience, despite some limitations.

8.4.2. Comparison with Related Work

When comparing this project with related work in the public domain, several points emerge:

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 network-based control. By integrating a custom binary protocol for IR and accelerometer data 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. While certain aspects (e.g., latency and peripheral support) still need refinement, the combined approach sets a new benchmark for retro gaming adaptation in distributed environments.

8.4.3. Future Work

However, there are clear avenues for future improvement. The following areas could be explored in future iterations:

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

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

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

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

[8] 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].

[9] *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].

[10] 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.

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

[12] *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].

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

[14] 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/