CS4099 - Nintendo Wii Over IP

Kieran Fowlds - 210018092
Supervisor: Dr. Tom Spink

25th 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 two key components. First, video and audio streaming techniques capture the Wii's outputs and deliver them to remote devices using low-latency protocols. This ensures fluid gameplay and preserves the authenticity of the original experience. Second, a novel controller input relay system transmits Wiimote signals, including motion and button inputs, over a network. This system addresses challenges such as Bluetooth communication, network variability, and precise synchronisation between audiovisual and control data, ensuring real-time responsiveness.

By bridging the gap between traditional local multiplayer and modern online connectivity, this project extends the life of a beloved console while revitalising classic gaming experiences. 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 largely due to its motion controls, low price, and its local multiplayer gameplay. However, as online gaming has become the norm, the traditional split-screen and communal experiences that defined the Wii era have faced diminishing support and new technical challenges.

The primary aim of this project is to bridge the gap between classic local multiplayer gameplay and the demands of modern distributed gaming. This is achieved by re-engineering both the output and input interfaces of the Wii. On one hand, the project focuses on capturing and streaming audio and video from the console using low-latency protocols to preserve the fluidity and authenticity of the original experience. On the other hand, a novel controller input relay system has been developed to transmit Wii Remote signals – including motion data, IR readings, and button presses – over a network to remote devices. In doing so, the system tackles challenges inherent in Bluetooth communication, network variability, and the need for precise synchronisation between audiovisual streams and control inputs.

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.

Throughout this report, the reader will find detailed 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, innovative 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 vitalising local multiplayer experiences. 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

Released by Nintendo in 2006, the Wii quickly became renowned for its innovative motion-based controls and engaging titles. 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 – cemented its legacy as a console that prioritised communal play.

2.2. Relevant Hardware and Software Technologies

Modern adaptations of the Wii experience leverage a range of hardware and software tools:

`WiimoteEmulator` **[1]:**

This publicly available project on GitHub allows for the emulation of Wii Remote signals, enabling a real Wii console to interface with a computer acting as an external controller. By emulating the communication protocol of the Wiimote, the project provides a basis for further experimentation with input methods. In the context of this dissertation, a fork of the `WiimoteEmulator` has been extended to accept IR and accelerometer data from across a network. This extension is key to bridging remote inputs with local emulation.

`xwiimote` Library[7]:

To capture real Wiimote input, the `xwiimote` library has been employed. Running on a Raspberry Pi, this library facilitates the interfacing of physical Wiimote hardware with software, thereby enabling the capture and processing of motion and button data. This data is then routed through a custom Python script that integrates with the extended emulation system, ensuring that remote control signals are correctly interpreted.

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, the Raspberry Pi is used to capture Wiimote data from a client machine and relay it to the emulation system on the host machine which interfaces with the Nintendo Wii console.

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 largely focused on the emulation of input devices for legacy consoles in order to preserve or extend their operational lifespan. Such projects have typically emphasised local connectivity and hardware replication. The extension to network-based control – wherein sensor data such as IR and accelerometer signals are transmitted remotely – is less common and represents a novel contribution of this work.

Remote Gaming Frameworks:

In recent years, there has been increased interest in remote gaming solutions, driven by advancements in streaming protocols and low-latency communication. While many contemporary projects target high-end gaming platforms, the retro gaming sphere has seen fewer contributions that successfully bridge the gap between traditional, hardware-based control schemes and modern, networked gameplay.

3. Requirements Specification

3.1. Functional Requirements

Video and Audio Capture and Streaming:

The system shall capture the Wii's video and audio outputs and stream them to remote players with minimal latency. This functionality is critical to preserve the fluid, immersive experience typical of classic Wii titles.

Controller Input Relay:

The solution must reliably capture and transmit Wii Remote inputs—including motion data and button presses—over a low-latency network connection. This bi-directional communication is essential for maintaining the real-time responsiveness expected in interactive gameplay.

Synchronization:

To ensure a seamless gaming experience, audiovisual data and controller inputs must be synchronised. The system should adjust for network variability and maintain precise timing to replicate local multiplayer dynamics.

3.2. Non-Functional Requirements

Performance:

The system must operate under strict low-latency conditions to minimise delay and jitter. Efficient processing and optimised data streaming protocols are required.

Reliability and Robustness:

The solution should tolerate variations in network quality, ensuring continuous, stable operation even under less-than-ideal conditions.

Accessability:

An intuitive interface and straightforward setup process should be provided, enabling users to connect and enjoy games with minimal technical intervention.

Evaluation:

Comprehensive testing in real-world environments is necessary. Both quantitative performance metrics and qualitative user feedback will be gathered to assess the overall experience.

4. Software Engineering Process

The development of this project was guided by the waterfall methodology[6]. Given the fixed deadline and the fact that the project was developed by a single individual, a sequential, plan-driven approach was deemed the most appropriate. Unlike iterative or Agile[2] methods – which offer continuous deployment and rapid iterations – the waterfall model allowed for clear delineation of phases and ensured that each stage of the project was fully completed and documented before progressing to the next.

4.1. Software Development Approach

The waterfall methodology was selected due to two primary reasons:

1. **Fixed Deadline:** With a single, non-negotiable deadline for delivering the complete system, the sequential nature of the waterfall model ensured that all project requirements were addressed in a structured manner. Each phase built on the preceding one, allowing for a well-planned progression from concept to final implementation.
2. **Single Developer Environment:** Since the project was executed by a single developer, the need for complex coordination and iterative refinement – common in multi-developer or Agile environments – was significantly reduced. This environment favoured a more traditional approach where requirements, design, implementation, and testing followed in a linear sequence.

The development process began with a comprehensive requirements specification that defined the system's objectives, such as revitalising the Wii's input and output interfaces for remote multiplayer gaming. Following this, the design phase was initiated, during which the overall architecture was established.

The development of this project was guided by the waterfall methodology. Given the fixed deadline and the fact that the project was developed by a single individual, a sequential, plan-driven approach was deemed the most appropriate. Unlike iterative or Agile methods – which offer continuous deployment and rapid iterations – the waterfall model allowed for clear delineation of phases and ensured that each stage of the project was fully completed and documented before progressing to the next.

4.2. Tools and Technologies

To implement this project, several tools and technologies were chosen based on their suitability and the developer's familiarity. These include:

Programming Languages:

The `WiimoteEmulator` project uses C, while Python was employed for higher-level tasks including input relay and automation. Bash scripts were used to automate system configuration and setup.

Build and Deployment Tools:

Git was used for version control, ensuring that all source code and documentation were managed efficiently. Automated scripts were created to handle environment configuration – loading necessary kernel modules, modifying Bluetooth settings, and setting environment variables such as `LD_LIBRARY_PATH`.

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[7] 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[5] 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 6.1 illustrates the primary components and their interactions.

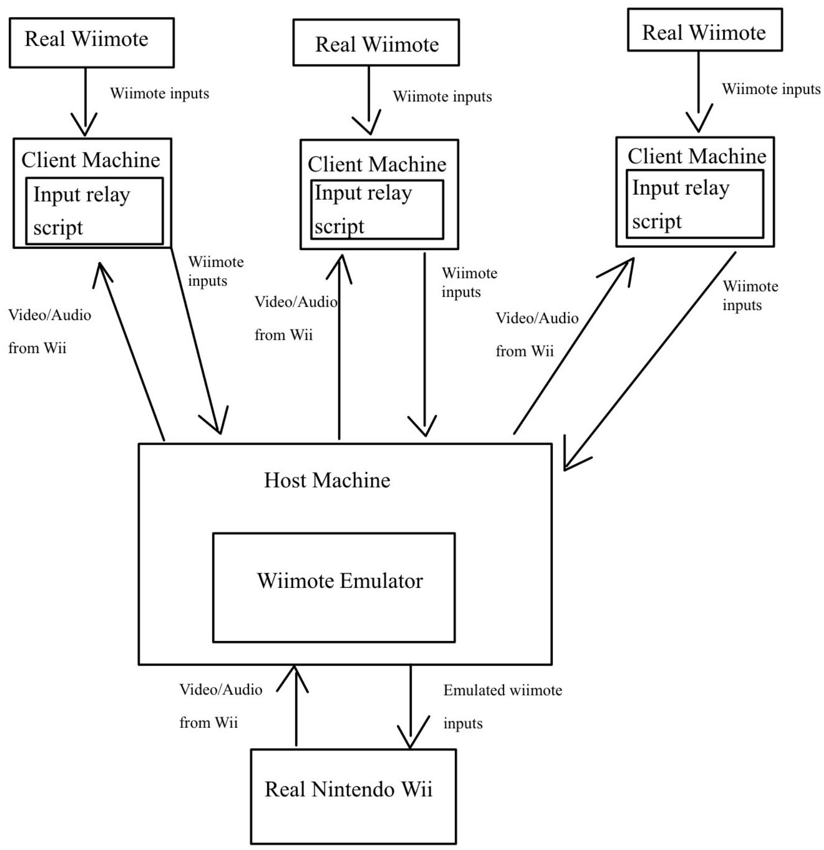

Figure 6.1.: System Architecture and Data Flow

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:

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.

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.

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[7] library was chosen, particularly for its Python bindings[8], 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[4] 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[5] 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[3] further extends this fork by adding support for transmitting IR and accelerometer data over the IP socket interface.

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:

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.

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.

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.

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.

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

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.

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.

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

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

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

[6] 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.

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

[8] 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/