CS4099 - Nintendo Wii Over IP

Kieran Fowlds - 210018092
Supervisor: Dr. Tom Spink

22nd 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 Wiis in-person, split-screen experiences has become increasingly challenging. This project proposes a solution that vitalises the Wiis 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 Wiis 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.

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.

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, family and friends for their support and encouragement throughout my time at the University of St Andrews. Finally, I would like to thank the School of Computer Science's Systems Team for their continuous help.

Contents

Introduction

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 Wiis 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 Wiis local multiplayer formatoften characterised by split-screen or shared-screen experiencescemented 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[5]:**

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 controlwherein sensor data such as IR and accelerometer signals are transmitted remotelyis 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.

Tool and Technology Integration:

The use of open-source libraries such as `xwiimote` alongside custom software modifications to existing projects (e.g., the `WiimoteEmulator` fork) illustrates a growing trend in leveraging community-driven tools to solve complex emulation challenges. Although a comprehensive body of literature specific to this integration is still emerging, the available work provides a solid foundation for exploring how retro systems can be adapted for contemporary, distributed gaming environments.

Requirements Specification

3.1. Functional Requirements

Video and Audio Capture and Streaming:

The system shall capture the Wiis 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 inputsincluding motion data and button pressesover a low-latency network connection. This bidirectional 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 synchronized. 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 minimize delay and jitter. Efficient processing and optimized 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.

Usability:

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.

Design

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.

5.1. 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.

5.2. Selection of Wii Remote Libraries and Addressing Bluetooth Issues

After evaluating multiple libraries and tools for Wii Remote interfacing, the xwiimote[5] library was chosen, particularly for its Python bindings[6], 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[3] 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.

5.3. 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.

5.4. 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[4] 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.

My version[2] 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 remainparticularly in synchronising IR and accelerometer data with the audiovisual stream. Further work is needed to reduce processing overhead in the transformation routines (e.g., matrix inversions in `set_accelerometer`) and to mitigate network jitter under varying conditions.

5.5. Python Script for Input Relay

The systems 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 initializes 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 Remotes 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 `0x03`. The binary format is:

```
[1 byte event type (0x03)] + [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 `0x02`:

```
[1 byte event type (0x02)] + [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 emulators IP address and port are provided via command-line arguments. The script logs key actions and any errors using Pythons 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.

5.6. Automation of Device Setup

To streamline the deployment process, a device setup script 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.

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

5.7. Testing and Validation

Rigorous testing was conducted to evaluate the performance and reliability of the system

Connectivity Tests

Confirmed that the Wii Remote establishes a stable connection with the Raspberry Pis under various conditions.

Latency and Quality Measurements

Evaluated the balance between media quality and streaming latency, with iterative tuning of RTP settings.

End-to-End Gameplay Trials

Real-world testing using games such as Mario Kart provided insights into the systems responsiveness and highlighted areas for future refinement, particularly regarding latency and IR display limitations.

Evaluation

6.1. Challenges and Solutions

6.2. Limitations

6.3. Reflection and Future Work

Conclusion

References

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

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

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

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

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

[6] 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/