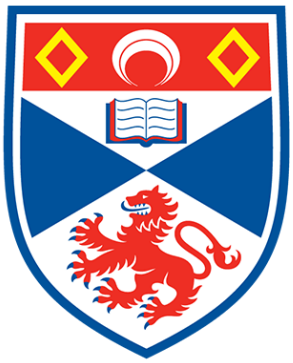
Nintendo Wii Over IP

Kieran Fowlds - 210018092
Supervisor: Dr. Tom Spink

1st April 2025



University of
St Andrews

Abstract

This dissertation presents a novel system designed to enable remote multiplayer gameplay on the Nintendo Wii console while preserving the original local multiplayer experience. Local multiplayer gaming has historically provided unique communal experiences, but as online gaming has become dominant, these traditional modes have diminished, leaving a significant portion of the Wii's legacy inaccessible. This work introduces a solution that enables remote players to participate in Wii games originally designed for local play.

The system integrates three core components. A capture and streaming subsystem is responsible for real-time extraction and transmission of the Wii's video and audio outputs. A controller input relay converts Wii Remote signals into fixed-length binary packets and transmits them over a low-latency network connection. Finally, a Wii Remote emulator running on the host device interprets the relayed inputs and communicates with the console via Bluetooth. This configuration strives to preserve the original responsiveness and interactive quality of the Wii experience, while addressing challenges such as transmission delay and input accuracy.

The project addresses key technical obstacles including latency minimisation, accurate input replication, and audiovisual quality maintenance. Rigorous testing and iterative design confirm that remote gameplay can closely approximate native local multiplayer interactions, despite residual challenges.

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 8499 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[17], 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[24], the Wii no longer supports online multiplayer. This coupled with the fact that most people no longer play multiplayer games locally[15], 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.

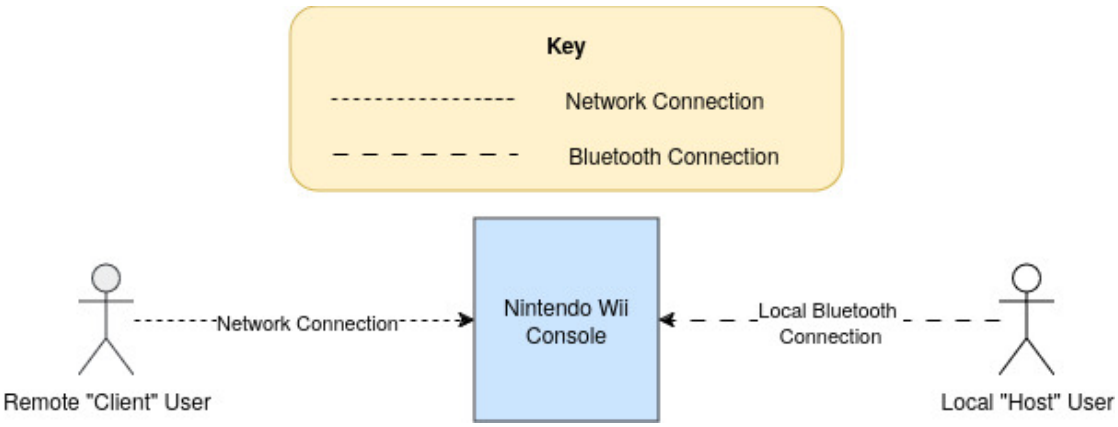


Figure 1.1.: Simple Overview of the System

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 (Wiimote) 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.

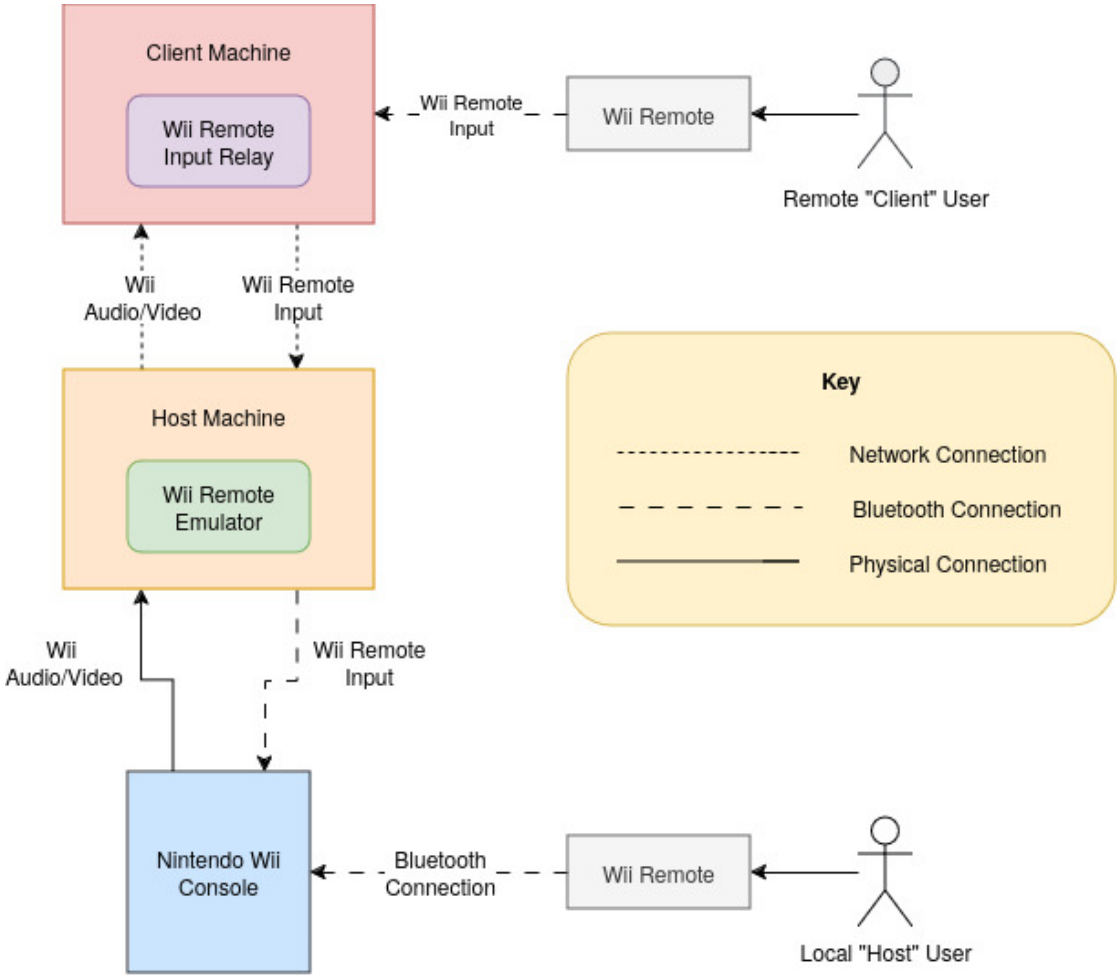


Figure 1.2.: System Architecture and Data Flow

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[24], 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[29]

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.

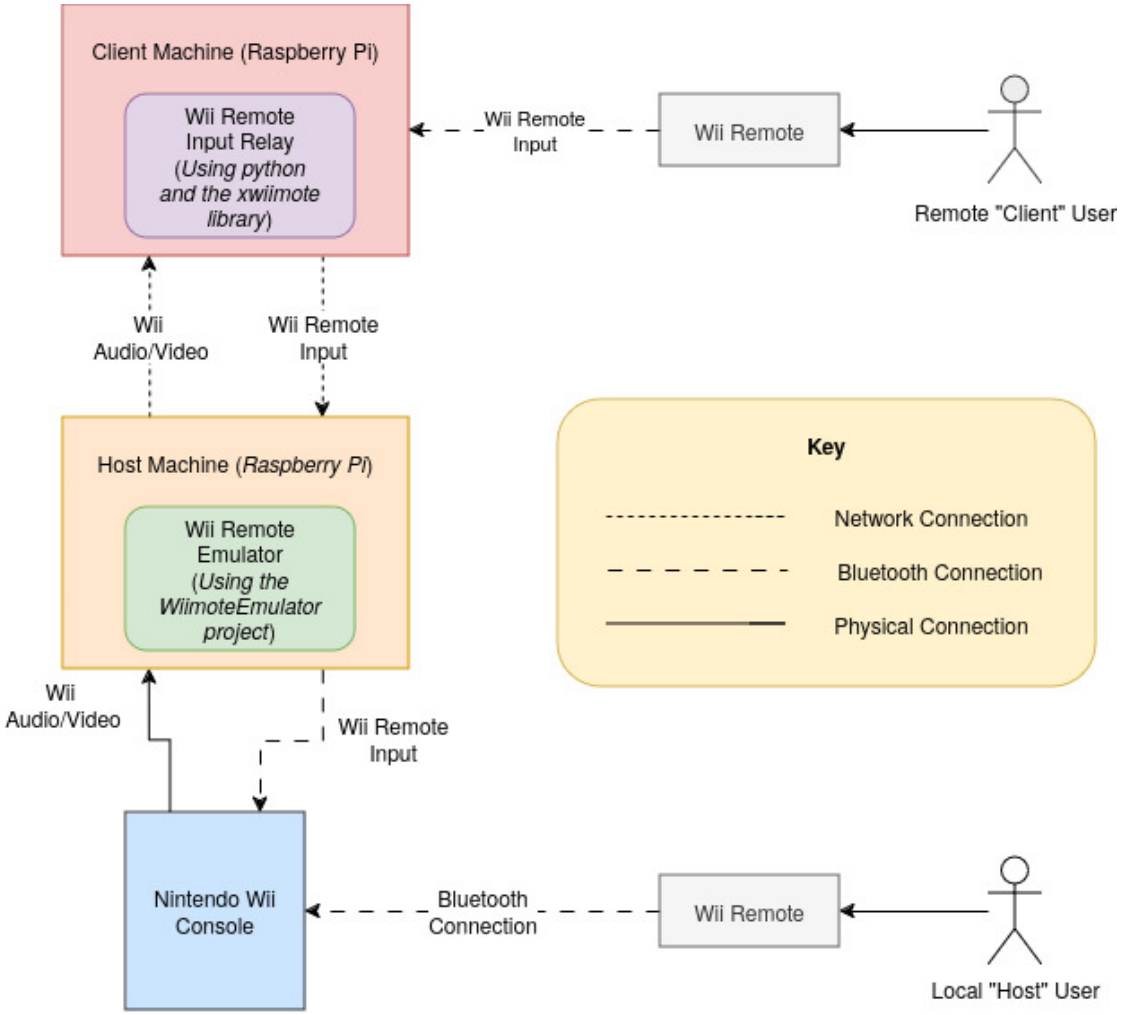


Figure 2.1.: Overview of the System with Key Technologies Linked

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[16]. 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[23]. 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[20]. This structured, sequential model fit the constraints of a fixed deadline and a single-developer workflow. Unlike Agile[2] or iterative models, which encourage continuous delivery and rapid prototyping, the waterfall approach provided distinct stages, enabling full completion and documentation of each phase before moving forward.

4.1. Software Development Approach

The waterfall model supported the project in two key ways:

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

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

4.2. Tools and Technologies

4.2.1. Programming Languages

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

C

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

Python

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

Bash

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

4.2.2. Build and Deployment Tools

Version control through Git supported code integrity, traceability, and future collaboration, 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.

6.1. System Architecture Overview

At a high level, the system comprises several loosely coupled components that work together to enable remote gameplay on the Nintendo Wii console. Figure 1.2 illustrates the primary components and their interactions.

The primary subsystems include:

- **Controller Input Relay** A custom Python script acts as an intermediary, capturing input events (such as accelerometer data, IR signals, and button presses) using the `xwiimote` library[29] and translating them into a binary format. The script transmits these updates over UDP[26] to a Wiimote emulator running on the host machine.
- **Wii Remote Emulator:** The emulator, derived from a fork[14] of the `WiimoteEmulator` project[1], interprets the incoming UDP packets and simulates the corresponding Wii Remote inputs, which are then relayed to the Wii console via a Bluetooth connection. This project extends the emulator's functionality to include streaming IR and accelerometer data over UDP, which is essential for emulating the Wii Remote's motion controls based on the remote player's inputs.
- **Audio and Video Streaming:** An audiovisual streaming component enables remote gameplay by capturing video and audio outputs from the Wii console. The system transmits these streams to a client machine via the host machine using the Real-time Transport Protocol (RTP)[19].
- **Automation and Deployment:** An automation script applies system configurations – such as loading kernel modules and installation of dependencies – consistently across devices. The setup is tedious, error-prone, and not user-friendly without automation. This problem compounds when considering the need to deploy the system across multiple devices for users of varying technical expertise.

6.2. Controller Input Relay

6.2.1. Design Rationale

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

6.2.2. Communication Strategy

The relay utilises fixed-length binary packets that transmit over UDP to minimise communication latency and relay each sensor event in as close to real time as possible. The choice of UDP, despite its lack of error-checking compared to TCP[25], was deliberate: it reduces overhead and is better suited for applications where timely data delivery is more critical than guaranteed packet delivery. All of the I/O operations in the relay are non-blocking, further enhancing responsiveness and ensuring that the system remains highly interactive.

6.2.3. Modularity and Extensibility

Modularity was a guiding principle in the relay's design. By mapping all input types to a unified binary format, the system remains adaptable. New sensors, like Wii MotionPlus[27] or alternate controllers, like nunchucks, can integrate without significant changes to the core architecture. This design supports future enhancements while maintaining a clear separation between data acquisition, processing, and transmission.

6.3. Wiimote Emulator

6.3.1. Leveraging Established Codebase

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

6.3.2. Enhanced Input Simulation

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

6.4. Audio and Video Streaming

6.4.1. Protocol Selection

Adopting the Real-time Transport Protocol (RTP) leverages its framework for live media streaming. RTP's design supports low-latency transmission, which is essential for achieving the lowest latency possible. The use of other protocols, like UDP, was considered but ultimately rejected due to the lack of built-in support for audio and video streaming. It is not the use of RTP alone that reduces latency but the specific configurations that the protocol allows for which UDP does not.

6.4.2. Use of FFmpeg and FFplay

The system relies on FFmpeg[7] and FFplay[9] for capturing and streaming audiovisual data. FFmpeg's extensive codec support and FFplay's real-time playback capabilities make them well-suited for this application. The design leverages these tools to ensure seamless audiovisual streaming, with FFplay providing a low-latency display of the Wii's output on the client machine. Notably, this requires remote users to use FFplay, or similar tools that integrate with FFmpeg, to view the game stream. This restricts users to a possibly unfamiliar interface, but ensures a consistent experience across all clients.

6.5. Automation and Deployment

6.5.1. Centralised Configuration Management

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

6.5.2. Accessibility and Usability

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

6.5.3. Scalability and Future-Proofing

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

6.6. Novel Design Features

6.6.1. Responsive Motion Control Emulation

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

6.6.2. Local Play Game Features

Other third party projects, like `wiimmfi`[28], have focused on replicating the functionality of the Wii's original online services. However, many Nintendo Wii games have specific gameplay features that require local multiplayer or never supported online play – such as Wii Sports. This project's focus on enabling local multiplayer experiences over IP is a unique design feature that caters to a broader range of games and plugs a gap in the existing ecosystem.

6.6.3. Automated Environment Configuration

Automates device setup by configuring kernel modules, adjusting Bluetooth configuration files, and installing necessary dependencies through a comprehensive script. This minimises configuration errors and drastically reduces setup time, ensuring that all devices operate from a consistent baseline. This design feature simplifies the deployment process, making the system accessible to users with varying technical expertise. The focus on accessibility and usability is novel in the context of remote gaming systems, where complex configurations can be a significant barrier to entry[6].

7. Implementation

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

7.1. Establishing Wii Remote Connectivity

7.1.1. Linux Driver Configuration

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

```
modprobe hid-wiimote
```

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

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

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

7.1.2. Bluetooth Pairing and Connection

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

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

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

   ```
   trust <Wii Remote MAC address>
   ```

   to trust the Wii Remote.
6. Put the Wii Remote into pairing mode and enter

   ```
   connect <Wii Remote MAC address>
   ```

   to establish a connection.

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

7.2. Selection of Wii Remote Libraries and Addressing Bluetooth Issues

Evaluation of multiple libraries and tools for Wii Remote interfacing led to the selection of the xwiimote[29] library,, particularly for its Python bindings[30], 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[13] – modifying the Bluetooth configuration file at `/etc/bluetooth/input.conf` and adding the following line resolves this issue:

```
1 ClassicBondedOnly=false
```

This setting, when set to `true`, restricts HID connections to bonded devices only. A bonded device is one that pairs with the host Bluetooth device using a link key, which is a shared secret between the two devices. This setting is useful for ensuring that only trusted devices can connect to the host Bluetooth machine. However, the Wii Remote does not support link key based encryption, so it is necessary to set this value to `false` to allow the Wii Remote to connect to the client machine. The setup script automates this configuration step to ensure consistent performance across devices.

7.3. Audio and Video Streaming Optimisation

Streaming audio and video from the host machine to the client machine 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 supported by FFmpeg[7]. Although further optimisations remain possible, this configuration currently offers a balanced compromise between low latency and acceptable media quality.

7.3.1. `broadcast-rtp.sh` Script

The broadcast-rtp.sh script leverages FFmpeg to capture video from the connected camera device (`/dev/video0`), encode it in real time, and broadcast it over the Real-time Transport Protocol (RTP).

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

```
ffmpeg -max_delay 0 -max_probe_packets 1 -f v412 -framerate 25 -
  video_size 720x576 -threads 1 -i /dev/video0 -vcodec libx264 -
  pix_fmt:v yuv420p -g:v 1 -preset ultrafast -tune zerolatency -
  crf 17 -max_delay 0 -fflags +nobuffer -flags low_delay -f rtp -
  muxdelay 0 rtp://192.168.20.20:42423
```

Table 7.1 explains the arguments used in the script in detail. See the official `ffmpeg` documentation for more information[8]. Notably, the script uses the `-preset ultrafast` option to minimise processing time and the `-tune zerolatency` option to optimise the encoder for real-time transmission which reduces latency. It also uses the `-crf 17` option which sets the constant

rate factor to 17. Constant rate factor (CRF) is a quality-based variable bitrate control method that adjusts the quality of the video stream while maintaining a constant bitrate. In the H.264 codec, the range of the CRF scale is 0–51, where 0 is lossless, 23 is the default, and 51 is worst quality possible. A lower value generally leads to higher quality. A CRF value of 17 or 18 is essentially visually lossless or nearly so[12]. Keeping the default CRF value of 23, along with the other latency reducing settings, results in video quality that is not acceptable as text and detail become illegible so to increase the quality, the system uses a CRF value of 17.

7.3.2. `play-rtp.sh` Script

The `play-rtp.sh` script is responsible for receiving and playing the RTP stream on the client machine. The script runs the following command:

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

```

Table 7.2 explains the arguments used in the script in detail. See the official `ffplay` documentation for more information[10]. Notably, the script uses the `-sync ext` option to synchronise playback using an external clock, ensuring accurate timing with the incoming RTP stream. The combination of the `-framedrop`, `-flags low_delay`, `-max_delay 0`, and `-fflags nobuffer` options helps maintain real-time playback by dropping frames if necessary, minimising buffering delays, and enforcing low-delay processing.

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

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

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

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

7.4. Python Script for Input Relay

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

7.4.1. Wiimote Connection and Monitoring:

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

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

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

7.4.3. Event Processing and Binary Packet Formation:

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

Accelerometer Events

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

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

IR Events:

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

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

Button (Key) Events:

The script also processes key events (e.g., pressing the `+`, `HOME`, `A`, etc buttons). For context, Figure 7.1 shows the layout of the Wii Remote buttons. The input relay handles these by sending text-based command packets (e.g., `"button 1 WIIMOTE_PLUS"`) over UDP to indicate button press and release actions.

7.4.4. UDP Communication:

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

7.4.5. Robust Error Handling:

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

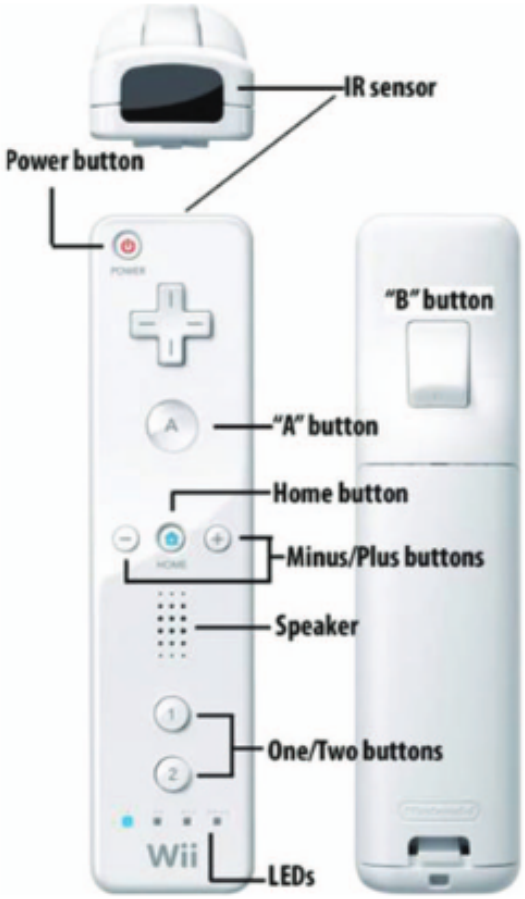


Figure 7.1.: Wii Remote Button Layout. Image source[18]

7.5. Wii Remote Emulation Enhancements

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

More specifically, this project builds on a fork of the project by JRogaishio[14] as it fixes two critical bugs. The first bug is that the IP command in the original project was not working due to an index error. The second bug is that the original project was not compiling due to a call to `graceful_disconnect()` which was not defined.

The version created in this project[11] builds upon this fork by adding support for transmitting IR and accelerometer data through the IP socket interface.

7.5.1. Enhancements

IR Emulation

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

Next, `set_motion_state` uses this transformation to compute two sensor points (`sensor_pt0` and `sensor_pt1`). In this process, a sensor point is defined as

$$p = (p_x, p_y, p_z, p_w)$$

A custom perspective projection matrix (`make_cam_projection_mat`) projects these points into a normalised coordinate system. Putting a sensor point through this perspective divide and normalisation yields:

$$p' = \left(\frac{p_x}{p_w}, \frac{p_y}{p_w}, \frac{p_z}{p_w} \right).$$

The normalised coordinates are then mapped to screen space and used to compute the IR sensor output as follows:

$$x_{\text{ir}} = \text{round}\Big(1023 \cdot \frac{p'_x + 1}{2}\Big),$$

$$y_{\text{ir}} = \text{round}\Big(767 \cdot \frac{p'_y + 1}{2}\Big),$$

$$\text{size} = \text{round}\Big(\text{min\_pt\_size} + \Big(1 - p'_z\Big)^2 \cdot \big(\text{max\_pt\_size} - \text{min\_pt\_size}\big)\Big),$$

with $\text{min\_pt\_size} = 1.0$ and $\text{max\_pt\_size} = 15.0$.

This process ensures that the emulated IR data closely mimics the signals produced by a real Wii Remote when pointing at a sensor bar.

Accelerometer Emulation

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

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

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

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

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

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

Latency Reduction

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

- **Non-blocking I/O:** In the `input_socket.c` file, UDP sockets use the `SOCK_NONBLOCK` flag to ensure that the system can continuously poll for new input events without stalling on network reads. Without this flag, the emulator would block on network reads meaning that the emulator would not be able to process any other events until the network read is complete. This is especially important since the input relay is streaming the IR and accelerometer data to the emulator in close to real-time so the emulator could miss multiple events if it was blocking on reads.
- **Optimised Data Pipelines:** The system uses lightweight binary protocols for both IR and accelerometer updates. Sending fixed-length packets (e.g., 13-byte packets for IR and accelerometer data) reduces the overhead associated with parsing, improving overall throughput.

A shown in the Playlability Analysis, the latency introduced by the emulator is minimal, with the majority of latency stemming from the video streaming component.

7.6. Automation of Device Setup

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

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

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

8. Evaluation

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

8.1. Experimental Setup

The core hardware comprised a Nintendo Wii system, two complete sets of Raspberry Pi units (each paired with a monitor and keyboard), and multiple Wii Remote controllers. A network switch was used to interconnect the devices on a closed, direct Ethernet network. To support gameplay evaluation, a disk copy of a split-screen multiplayer Wii game – Mario Kart in this case – was used, ensuring that the system was tested under conditions closely resembling a typical gaming scenario.

In the playability analysis setup, the configuration was modified slightly to accommodate specialised measurement requirements. The Wii's video output was split using a composite splitter, which enabled the signal to be simultaneously routed to the USB composite capture device attached to one Raspberry Pi and to an extra monitor. This additional monitor served to display the original, real-time composite output from the Wii, providing a baseline for latency comparisons. This additional display allowed for a direct, real-time view of the Wii's output which doesn't go through the capture device. By comparing the real-time composite feed with the processed stream, the analysis could isolate and quantify any additional delays introduced during capture, encoding, network transmission, and decoding.

8.2. Playlability Analysis

The playability analysis takes into account 5 distinct parts of the system that contribute to the overall interaction lag[22].

8.2.1. Latency of Audiovisual Transmission

The first test focused on quantifying the delay in the complete video path—from the moment the signal left the Wii, through capture by the composite device, encoding on the host machine, network transmission, decoding by the client machine, and finally its display on the client monitor. To determine this delay, both the client monitor and the additional monitor (showing the direct Wii output) were recorded simultaneously. Analysis of the recordings revealed a consistent lag of 21 frames between the two displays. Given that the video was captured at 25 frames per second and played back at one-eighth speed, this delay corresponds to approximately 105 milliseconds of lag. This measured latency encompasses the cumulative delays inherent in video capture, encoding, and decoding, offering a clear benchmark for the audiovisual subsystem's performance.

8.2.2. Wii Remote Emulator Latency

The second test evaluated the performance of the Wii Remote emulator itself. By incorporating a `gettimeofday` call in the C code immediately upon receiving an input from the network and again right after dispatching the corresponding emulated input, the latency intrinsic to the emulation process was determined and output to the terminal. The results, shown in Listing 8.1, indicate that the average latency for IR data was 1511 microseconds (1.511 milliseconds), while the accelerometer and button data exhibited average delays of 1498 and 5388 microseconds (1.498 and 5.388 milliseconds), respectively. These measurements illustrate that the emulation system introduces only minimal delays.

```
1  IR:          average 1511 µs (1326 samples)
2  Accelerometer: average 1498 µs (1457 samples)
3  Button:      average 5388 µs (25 samples)
```

Listing 8.1: Wii Remote Emulator Latency Results

8.2.3. Input Relay System Latency

The third test assessed the efficiency of the Python-based input relay system. Similar in concept to the emulator test, this measurement was conducted by recording timestamps using Python's `time.monotonic()` function immediately before an input was processed and again right after it was sent over the UDP socket then printing the results to the terminal. The results, shown in Listing 8.2, show

that the latency of the input relay system is negligible, with average delays of 18, 13, and 18 microseconds for the accelerometer, IR, and button inputs, respectively. These results confirm that the input relay mechanism imposes an extremely minor delay.

```
INFO:wiimote_streamer:  Accelerometer: average 18 µs (3389 samples)
INFO:wiimote_streamer:  IR:            average 13 µs (3058 samples)
INFO:wiimote_streamer:  Button:        average 18 µs (26 samples)
```

Listing 8.2: Input Relay System Latency Results

8.2.4. Network Latency

The fourth test concentrated on the network latency between the host and client machines. In this evaluation, the Raspberry Pis were connected directly via Ethernet within a closed network, a configuration that minimises external interference. The command `ping -c 100 -i 0.2 192.168.20.10` was executed to collect latency statistics. The analysis, shown in Listing 8.3, revealed an average round-trip time (RTT) of 0.203 milliseconds as well as a minimum and maximum RTT of 0.190 and 0.289 milliseconds, respectively. The results demonstrate that the network latency of the system is very low. It is important to note that while these values demonstrate extremely low latency on a dedicated network, typical home networks experience latencies in the 30–100ms range on average[31].

```
--- 192.168.20.10 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 20195 ms
rtt min/avg/max/mdev = 0.190/0.203/0.289/0.010 ms
```

Listing 8.3: Network Latency Results

8.2.5. Bluetooth Latency:

In this project, the Bluetooth latency of the Wii Remote was not directly measured, however, measurements of Wii Remote latency have been conducted by others. The MiSTer FPGA Input Latency Tester[32] is a tool that can measure the latency of various controllers, including the Wii Remote. A collection of measurements from this tool called the “MiSTer Input Lag Database”[21] shows that the Wii Remote has an average latency of 34.115 to 36.071 milliseconds.

From this data, it can be inferred that the Bluetooth latency of the Wii Remote is approximately 35 milliseconds which is a significant contribution to the overall system latency.

8.2.6. Overall Interaction Lag

The interaction lag, defined as the delay between a user's input and the corresponding on-screen response[22], can be calculated by summing the latencies of each part of the system. In this setup, the total interaction lag is:

$$\begin{aligned}\text{Total Interaction Lag} &= \text{Audiovisual Latency} + \text{Wii Remote Emulator Latency} \\ &\quad + \text{Input Relay System Latency} + \text{Network Latency} \\ &\quad + (\text{Bluetooth Latency} * 2) \\ &= 105 + 5.388 + 0.018 + 0.203 + (35 * 2) \\ &= 180.609 \text{ milliseconds}\end{aligned}$$

This total is assuming the worst case (button press) for the Wii Remote emulator and input relay system latencies. The Bluetooth latency is multiplied by two to account for the fact that there is a Bluetooth connection between the Wii Remote and the client machine as well as between the Wii Remote emulator running on the host machine and the Wii Console. The network lag is only counted once as the Audiovisual latency includes the network latency on the trip back to the client machine. Note that using a typical home network with latencies in the 30-100ms range would increase the total interaction lag to 240-380ms.

8.3. Challenges and Solutions

IR Sensor Emulation

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

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

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

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

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

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

Audiovisual Streaming

The initial implementation of the streaming pipeline suffered from high latency and frequent buffering issues, resulting in a sub-optimal user experience. To address these challenges, the system underwent several rounds of optimisation, focusing on reducing delay while maintaining media quality.

Manual tests refined the audiovisual streaming component by evaluating each parameter adjustment for its impact on latency and video quality. After trying each option, the system's response was observed in real time to assess whether the perceptible latency met the requirements for responsive gameplay and whether any degradation in video quality was within acceptable limits.

Initially, the adoption of options such as `-preset ultrafast` and `-tune zerolatency` resulted in the most significant reduction in noticeable latency. However, these settings also led to a decline in video quality, introducing compression artefacts and reduced image clarity that hampered the overall viewing experience. In contrast, incorporating the `-crf 17` parameter improved the visual quality substantially. Although it introduced a slight increase in the latency, this trade-off proved acceptable as the added delay was barely perceptible, while the quality enhancement ensured that details remained crisp and legible during fast-paced gameplay.

This iterative testing and fine-tuning process ultimately converged on the final set of streaming parameters, shown in section 7.3, that best balanced low latency with high video quality. The final configuration represents a carefully considered compromise, ensuring that the audiovisual streaming pipeline delivers a responsive and visually satisfactory experience for remote gameplay.

Latency

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

On the streaming side, the low latency results from a combination of high-speed encoding and aggressive buffering control. The use of `ffmpeg` with options like `-g:v 1` (forcing every frame as a keyframe) helps in rapid recovery and minimises delays during packet loss or re-synchronisation events. The broadcasting pipeline's non-buffered approach, as enforced by the `-max_delay 0` and `-fflags +nobuffer` flags, keeps the end-to-end delay to a minimum, although this makes the system more sensitive to network jitter.

The input relay component, responsible for transmitting controller data from the Wii Remote to the emulator, further mitigates latency by employing non-blocking I/O and a lightweight binary protocol. By packaging sensor data (IR and accelerometer values) into fixed-length packets and transmitting them over UDP, the system avoids the overhead associated with more complex data formats.

Despite these efforts, some latency remains when compared to native Wii gameplay. As shown in the Playlability Analysis, the system has a total interaction lag of approximately 200milliseconds with the audiovisual streaming contributing 105 milliseconds of this. This latency is noticeable in fast-paced games but is still within a playable range.

8.4. Limitations

Although the system meets its primary objectives, several limitations remain that constrain its overall performance and usability. In this section, each limitation is discussed in detail, accompanied by potential solutions to address them in future iterations.

8.4.1. Peripheral Support

One significant limitation of the current implementation is its restricted support for peripheral devices. At present, the system only handles input from

the primary Wii Remote, excluding peripherals such as the nunchuck – a crucial peripheral for many Wii titles. This limitation not only narrows the scope of emulated experiences but also diminishes the authenticity of gameplay, as many games rely on the complementary inputs of both controllers. The absence of nunchuck support restricts the system's appeal and usability, especially for titles that require coordinated two-handed control. Addressing this limitation would involve extending the existing input relay architecture to include additional routines for capturing and processing nunchuck signals.

8.4.2. Scalability

Another key limitation is the system's scalability. During testing, the setup was evaluated with only a single remote player, leaving its performance under multi-user scenarios unexamined. This single-user focus raises concerns about how the system will manage increased network load, concurrent input streams, and potential synchronisation issues when multiple players connect simultaneously. Scalability issues may manifest as network congestion, increased latency, or even data loss in real-world multi-player environments. To resolve these challenges, future research should incorporate scalability testing under simulated conditions that mimic the demands of multiple concurrent connections. Moreover, potential improvements might include optimising the network communication protocols, adopting load balancing techniques, or restructuring the input relay to efficiently handle the increased volume of simultaneous inputs without compromising performance.

8.4.3. Latency

Latency remains a critical limitation, as it directly impacts both the responsiveness of the audiovisual streaming and the precision of input relays. Despite several optimisation strategies – such as the use of non-blocking I/O, tuned streaming settings, and emulation improvements – a noticeable delay persists when compared to native Wii gameplay. This residual latency arises from the cumulative delay introduced at various stages, including video capture, encoding, network transmission, decoding, input processing, and emulation. However, the largest contributor to latency is the audiovisual streaming pipeline, which accounts for approximately 105 milliseconds of the total interaction lag as shown in section 8.2. To further mitigate latency, future work could explore more advanced compression algorithms, refine the UDP transmission pipeline,

or experiment with alternative low-latency streaming protocols. Additionally, implementing predictive buffering or adaptive latency compensation algorithms might help offset some of the unavoidable delays, thereby enhancing the overall responsiveness of the system.

8.4.4. Accelerometer Calibration

The final limitation concerns the calibration of the accelerometer emulation. Currently, the system relies on hand-tuned parameters to interpret accelerometer data, which may not be universally optimal across all games or user preferences. This manual calibration approach can lead to inconsistencies, where the sensitivity and accuracy of motion controls vary between different games. In some cases, this might necessitate game-specific adjustments to achieve satisfactory performance, thereby undermining the system's general applicability. A promising solution to this problem is the use of dynamic calibration techniques that automatically adjust the accelerometer settings in real time. Using these techniques, the system could analyse gameplay data on the fly and fine-tune the calibration parameters to match the current game's requirements and individual user behaviour. Additionally, incorporating a user-friendly calibration interface would empower users to make personalised adjustments, further bridging the gap between the emulated inputs and the original Wii experience. Furthermore, if the Wii Remote emulator was able to perfectly replicate the accelerometer data then there would be no need for calibration which would be the ideal solution.

8.5. Reflection and Future Work

8.5.1. Evaluation of Objectives

As stated in the Introduction, 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 Playlability Analysis demonstrates that the system is capable of providing a playable experience for users, despite some limitations.

8.5.2. Comparison with Related Work

WiimoteEmulator

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

Audiovisual Streaming in Remote Gaming

Cloud gaming platforms traditionally deploy extensive infrastructure and proprietary solutions to manage audiovisual streams. This allows for high-quality, low-latency streaming, but it comes at huge monetary and technical cost[5]. In contrast, this project's solution uses open-source tools and protocols to achieve similar functionality. This system has higher latency and lower quality than commercial cloud gaming services, but everything can run locally for free on low-end hardware.

Wii Online Services

The original Wii Online services offered a better user experience than this project, as its design was tailor made for the Wii's hardware and software. However, Nintendo discontinued these services in 2014[24]. Some third-party projects like Wiimmfi[28] have focused on replicating the functionality of the Wii's original online services. However, many Nintendo Wii games have specific gameplay features that require local multiplayer or never supported online play – such as Wii

Sports. This project aims to address this limitation by enabling remote multiplayer experiences for games that were not designed for online play.

8.5.3. Future Work

Enhanced Peripheral Integration

An avenue for future work involves expanding the system to support a broader range of Wii peripherals. For example, incorporating the nunchuck would not only enhance the gaming experience but also extend the system's compatibility with a wider variety of games that rely on additional input modalities. This expansion would require revisiting the current input relay architecture to integrate new input types and handling the unique sensor characteristics of each peripheral. Furthermore, the integration of these devices may necessitate revisions to the calibration routines and error-handling mechanisms, ensuring that the system processes the additional inputs with the same level of precision as the primary Wii Remote.

Scalability Testing

Future work should focus on scalability testing to evaluate how the data relay mechanisms perform when subjected to higher network loads. This entails setting up experiments that mimic real-world scenarios with multiple simultaneous connections, stressing the network to identify potential bottlenecks or points of failure in the project. By adopting robust simulation and testing methodologies, researchers could optimise the system's architecture to better manage concurrent data streams and ensure that performance degrades gracefully under load. Ultimately, such testing will provide valuable insights into the system's resilience and inform subsequent iterations of both software and hardware optimisations.

Latency Optimisation

Despite the progress made in reducing system latency, there is still room for improvement, especially for fast-paced gaming scenarios. Future work should explore advanced techniques for latency optimisation across all components of the system. This could involve a deeper investigation into the `WiimoteEmulator` fork and the input relay program to identify bottlenecks and streamline existing hot paths. Additionally, examining alternative RTP streaming parameters and

exploring new network protocols might reveal opportunities to shave off critical milliseconds of delay.

Dynamic Calibration Techniques

Another important direction for future research is the development of dynamic calibration techniques for the accelerometer data. The current system relies on hand-tuned parameters, which, while effective in a controlled setting, may not perform optimally across different games or individual user preferences. Future work could focus on implementing adaptive algorithms that automatically adjust calibration settings in real time, using data gathered during gameplay. This might involve leveraging machine learning models to predict optimal calibration profiles based on user interactions or game-specific demands. By continuously refining the calibration process, the system can provide more accurate input replication, reducing discrepancies and further narrowing the performance gap between the emulated and original Wii experiences.

9. Conclusion

This dissertation set out to explore the feasibility of remote multiplayer gameplay on the Nintendo Wii console while retaining local multiplayer dynamics. Through the development of a comprehensive system that integrates audiovisual streaming, low-latency input relay, and Wii Remote emulation, the project has successfully demonstrated the feasibility of remote gameplay that closely mimics the native local multiplayer experience.

The project's most significant achievement lies in its successful integration of multiple complex components into a cohesive system. The implementation of a custom input relay, in tandem with a modified Wii Remote emulator, allowed for real-time transmission of controller inputs. This solution ensured that remote players could experience accurate and responsive control, even though some latency remained compared to native gameplay.

Simultaneously, the audiovisual streaming subsystem managed to balance the trade-offs between media quality and latency, providing an immersive, real-time gaming experience. Overall, the system has demonstrated that with careful design and tuning, it is possible to deliver remote gameplay that respects the original Wii experience.

Despite these successes, the project also encountered significant challenges and limitations. One key drawback was the persistent latency observed throughout the system. Although non-blocking I/O and aggressive buffering strategies reduced delays, the cumulative effect of video capture, encoding, network transmission, and emulation meant that some degree of lag remained – particularly noticeable in fast-paced gaming scenarios.

Additionally, limited peripheral support constrains the current implementation; the focus on the Wii Remote excludes important accessories like the nunchuck, thereby narrowing the scope of supported games. Manual accelerometer calibration also posed a challenge, as the fixed parameters did not adapt dynamically to different game requirements or user preferences.

In conclusion, this dissertation has successfully explored the transformation of a nostalgic gaming console into a modern, remote multiplayer platform. While certain technical challenges and limitations remain, the achievements outlined

herein demonstrate the potential for further innovation in this space. The insights gained from this work not only provide a foundation for future innovations in remote gaming technology but also reaffirm the enduring appeal of the classic Nintendo Wii experience.

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. (Visited on 26/03/2025).

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

[5] Wei Cai et al. 'A Survey on Cloud Gaming: Future of Computer Games'. In: *IEEE Access* 4 (Jan. 2016). DOI: 10.1109/ACCESS.2016.2590500. (Visited on 30/03/2025).

[6] Dirk Colbry. *Reducing the barrier to entry using portable apps*. https://www.researchgate.net/publication/254003063_Reducing_the_barrier_to_entry_using_portable_apps. (Visited on 28/03/2025).

[7] *FFmpeg - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/FFmpeg. (Visited on 28/03/2025).

[8] *ffmpeg Documentation — ffmpeg.org*. https://ffmpeg.org/ffmpeg.html#Options. (Visited on 29/03/2025).

[9] *ffplay Documentation*. https://ffmpeg.org/ffplay.htm. (Visited on 28/03/2025).

[10] *ffplay Documentation — ffmpeg.org*. https://ffmpeg.org/ffplay.html#Options. (Visited on 29/03/2025).

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

[12] *H.264 Video Encoding Guide — trac.ffmpeg.org*. https://trac.ffmpeg.org/wiki/Encode/H.264#a1.ChooseaCRFvalue. (Visited on 01/04/2025).

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

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

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

[16] 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. (Visited on 26/03/2025).

[17] *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. (Visited on 26/03/2025).

[18] Y. Peng et al. *A new respiratory monitoring and processing system based on Wii remote: Proof of principle*. https://www.researchgate.net/publication/245026366_A_new_respiratory_monitoring_and_processing_system_based_on_Wii_remote_Proof_of_principle. 2013. (Visited on 01/04/2025).

[19] *Real-time Transport Protocol - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Real-time_Transport_Protocol. (Visited on 28/03/2025).

[20] 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.

[21] *RPubs - MiSTer FPGA Input Latency Tester - Input Lag Database — rpubs.com*. https://rpubs.com/misteraddons/inputlatency. (Visited on 01/04/2025).

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

[23] *Tailscale vs. Hamachi: A Modern VPN Replacement for Gaming and Collaboration — tailscale.com*. https://tailscale.com/blog/hamachi. (Visited on 26/03/2025).

[24] *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. (Visited on 26/03/2025).

[25] *Transmission Control Protocol - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Transmission_Control_Protocol. (Visited on 28/03/2025).

[26] *User Datagram Protocol - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/User_Datagram_Protocol. (Visited on 28/03/2025).

[27] *Wii MotionPlus - Wikipedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Wii_MotionPlus. (Visited on 28/03/2025).

[28] *Wiimmfi*. https://wiimmfi.de/. (Visited on 28/03/2025).

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

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

[31] Mehdi Zeinali and John Thompson. *Comprehensive practical evaluation of wired and wireless internet base smart grid communication*. https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/stg2.12023. 2021. (Visited on 01/04/2025).

[32] Louis Zezeran. *Lag Test Your Controller - MiSTer FPGA Input Latency Tester - Cathode Ray Blog — cathoderayblog.com*. https://www.cathoderayblog.com/lag-test-your-controller-mister-fpga-input-latency-tester/. (Visited on 01/04/2025).

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

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/

B. User Manual

Before proceeding with any operations, ensure that you have run the `setup.sh` script. This script installs and compiles all necessary dependencies, so there is no need to build or compile components manually.

B.1. Determining Bluetooth Addresses

B.1.1. Determining the Wii Remote Address

1. Use the command:

   ```
   hcitool scan
   ```

   to search for discoverable devices.
2. Press the red sync button (located under the battery cover) on the Wii remote.
3. The Wii remote should appear (usually listed as `Nintendo RVL-CNT-01`) with its corresponding MAC address.

B.1.2. Determining the Wii Console Address

If the Wii remote emulator is operational, simply press the console's sync button. The emulator will automatically connect and display the Wii console's address.

B.2. Wii Remote Connection to Client Machine

To connect a Wii remote to the client machine, follow these steps:

1. **Identify the Wii Remote's MAC Address:** Follow the steps outlined in Section B.1.1 to determine the Wii remote's MAC address.
2. **Establish a Connection:**

a) Open a terminal and launch `bluetoothctl`.

b) Enable scanning by typing

```
scan on
```

c) Type the command

```
trust [address]
```

while pressing the 1 and 2 buttons on the Wii remote which will put the Wii Remote into sync mode.

d) Next, type

```
connect [address]
```

ensuring that the Wii Remote is still in sync mode by pressing the 1 and 2 buttons.

B.3. Running the Wii Remote Emulator and Input Relay

To emulate Wii remote functionality, perform the following steps from within the `WiimoteEmulator` directory:

1. **Disable and Stop the Bluetooth Service:**

```
sudo systemctl disable bluetooth
sudo systemctl stop bluetooth
sudo systemctl status bluetooth
```

2. **Start the Custom Bluetooth Daemon Manually:**

```
sudo ./bluez-4.101/dist/sbin/bluetoothd
```

3. **Launch the Emulator:** Run the emulator with the Wii Console's MAC address (as determined in Section B.1.2) and specify the IP address and port:

```
sudo ./wmemulator [MAC Address] [IP Address] [Port]
```

4. **Start the Python Input Relay on client machine:** On another machine, run the input relay script:

   ```
   ./input_relay.py --host [IP Address] --port [Port]
   ```

5. **Restoring Bluetooth Services:** Once finished, execute the following on the host machine:

   ```
   sudo killall bluetoothd
   sudo systemctl enable bluetooth
   sudo systemctl start bluetooth
   sudo systemctl status bluetooth
   ```

B.4. Streaming Wii Video to Another Machine

To stream video from the Wii, set up the host and client machines as follows:

1. **On the Host Machine:** Run the broadcasting script:

   ```
   ./broadcast-rtp.sh
   ```

2. **On the Client Machine:** Run the playback script:

   ```
   ./play-rtp.sh
   ```

3. **Terminate the Streaming Session:** When you wish to stop streaming, press `q` on the client machine and then terminate the broadcast process by pressing `Ctrl+C` on the host machine.