UNIVERSITY OF TARTU

Institute of Computer Science

Computer Science Curriculum

Raiko Valo

Injecting Inputs Over Internet Connections

Bachelor's Thesis (9 ECTS)

Supervisor: Ulrich Norbisrath

Tartu 2024

Injecting Inputs Over Internet Connections

Abstract:

This thesis is about developing a system to control *SIGINT* games with a controller that sends gamepad inputs to the game host machine over a WiFi connection. This involves creating a controller and a receiver script; the client is sending gamepad input data through a local network connection to the server code, which relies on those games by emulating a virtual gamepad.

Keywords:

Gamepad, controller, receiver, server, emulator, M5StickC, ESP32, WiFi

CERCS: P170 Computer science, numerical analysis, systems, Control

Sisendite edastamine internetiühenduse kaudu

Lühikokkuvõte:

See uurimistöö käsitleb süsteemi väljatöötamist *SIGINT* mängu juhtimiseks puldiga, mis saadab sisendid üle WiFi ühenduse mängu jooksutavale masinale. See hõlmab kontrolleri ja vastuvõtja skripti väljatöötamist, klient saadab mängupuldi sisendandmed kohaliku võrguühenduse kaudu serverile, see edastab need mängule kasutades emuleeritud virtuaalset mängupulti.

Võtmesõnad:

Mängupult, kontroller, vastuvõtja, server, emulaator, M5StickC, ESP32, WiFi

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction 4

2. Related work 6

2.1 Latency effect on gaming experience 6

2.2 Bluetooth and WiFi Metrics Comparison 6

3. Technology 8

3.1 M5StickC 8

3.2 JoyC Omnidirectional Controller 8

3.3 Hardware Integration 8

3.4 Flow of the System 9

3.5 Computer Graphics Projects EXPO 2024 10

4. Software 12

4.1 Development Environment 12

4.2 Software Solution 12

4.2.1 Library for Network Input 12

4.2.2 Virtual Gamepad 13

4.2.3 Data Format 13

4.3 Writing a program 14

4.3.1 Receiver 14

4.3.2 Python Controller 15

4.3.3 Arduino Controller 16

5. Testing and Analysis 17

5.1 Test setup 17

5.2 Analysis 17

5.2.1 Latency 17

5.2.2 Package loss 19

5.3 Comparison 19

6. Conclusion and Future Study 21

List of References 23

1. Introduction

The focus of this thesis revolves around creating systems for injecting and receiving inputs over an internet connection, particularly within the context of playing games at large events. The primary objective is developing software and hardware solutions that enable the transmission of controller input through a WiFi connection. The goal of this research is to leverage internet connectivity to transmit controller input signals. By doing so, this thesis aims to broaden the accessibility and flexibility of gaming experiences, allowing players to engage with games using alternative input device setups. The software part of the thesis is visible on https://github.com/raiko-valo/thesis-2024.

The gaming landscape continuously evolving, spurred on by technological advancements such as cloud and virtual reality (VR) gaming, which offer game developers opportunities to create new experiences [1]. The Computer Graphics and Virtual Reality (CGVR) Study Lab at the University of Tartu is one of them for creating ways to play games in the corridor of the Delta building, as presented in Figure 1. This is achieved with a projector that displays content on a smart screen, and games can be played with wireless controllers outside the lab.

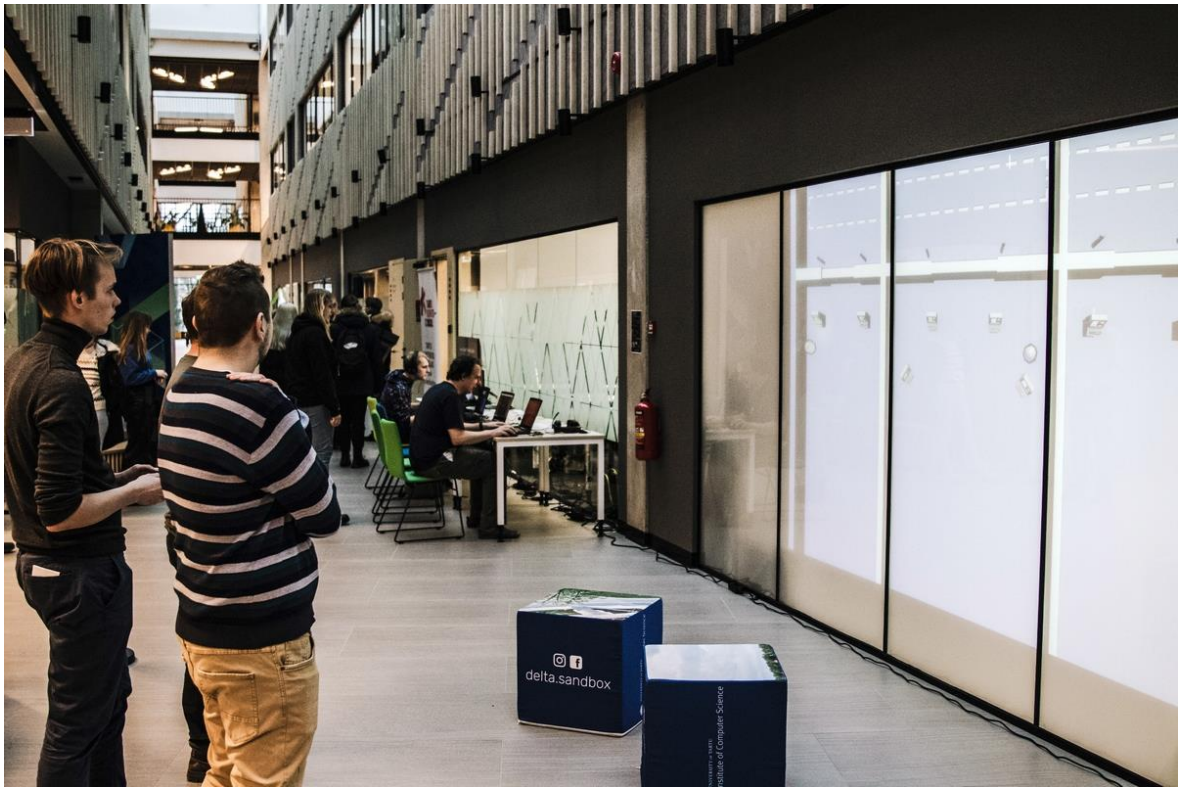

Figure 1. Computer Graphics Projects EXPO 2024.

The following paragraph describes *SIGINT* [2], a game that Mattias Aksli has developed as part of his Bachelor's thesis. It is playable on the vertical smart glass panels of the CGVR Lab in the Delta building of the University of Tartu. The game can be played with two to four players, and the main focus is teamwork and solving environmental puzzles together. One of the goals was to make the glass wall of the room more interactive for those outside. It can be played at https://mattiasa.itch.io/sigint, and the project is available at https://github.com/mattiasaksli/sigint.

The SINGIT game is hosted by a Raspberry Pi, which serves as the central hub for launching and running the game. The projector is displaying the game on the glass wall, making it visible from the corridor side. Players interact with the game using Xbox or PlayStation controllers that have Bluetooth connections, causing problems, since their connection is going through glass and other materials, affecting the signal strength [3].

2. Related work

2.1 Latency effect on gaming experience

In games that are focused more on reaction time, it is essential that the delay between player input and result is minimal because it allows them to react fast to changing situations and environments. The result of the game and the general feel of the game depends on it. Low latency provides control over actions and the situation.

The study [4] by Meixintong Zha and Ying Zhang investigated how input delay affects people's emotions and performance. They created two games specifically for testing purposes: *Sushi Shooter* and *Square Dodger*. The first game aimed to gather feedback on how shooting accuracy changes with different latencies. The second game focused on studying movement, with the objective of avoiding collisions with falling squares. In both games, a delay of 100 milliseconds was added every 30 seconds, and the total duration was 200 seconds. There were a total of 36 participants, and their experiences were measured through three independent variables: performance, emotion, and heart rate. The results of the study showed that in both games, as the input delay increased, the number of inputs decreased, and performance worsened. The hit frequency decreased in *Sushi Shooter*, and participants missed more targets with an increase in delay. In the second game, maneuvering around squares became more complex with longer delays, resulting in a higher average collision rate. Participants' emotions became more negative with increasing latency, with signs of anger and disgust indicating a rise in stress levels.

2.2 Bluetooth and WiFi Metrics Comparison

This paragraph introduces a study [3] conducted to compare the performance of various wireless communication protocols, focusing on the performance disparity between WiFi and Bluetooth. With its good data transmission capabilities and extended range, WiFi is a great contender in scenarios demanding high-speed connectivity over various distances. In contrast, the strength of Bluetooth is its exceptional power efficiency and connectivity within short distances, making it the preferred choice for prioritizing low power consumption and being in close proximity to the host device.

Several factors must be considered when choosing whether to use WiFi or Bluetooth for a setup where the Raspberry Pi is in another room running the game and there is a wall between the

controller and host. Both signals can degrade when passing through obstructions, but they are hardly affected by glass materials [3]. The wall between the host and controller is not entirely glass. Additionally, parts of it are concrete and potentially contain some kind of metal, the type of obstacles that interfere with reception more than Bluetooth and WiFi [3]. Considering the combination of wall and range, WiFi seems to be a better competitor for data transmission.

3. Technology

3.1 M5StickC

M5StickC is a microcontroller used as a computing board for developing the Internet of Things (IoT) [5]. It is built based on an ESP32 microcontroller with low power consumption WiFi capabilities, enabling it to connect to local networks and transmit data over the internet [5]. The decision to incorporate the M5StickC into the project is driven by its compact form, many feature sets, and good computing power. It allows integration with other hardware components, while its computational prowess ensures execution of the software solution. Moreover, its built-in display offers the potential for visual feedback that is good for debugging and interactions, enhancing the user experience within the system.

3.2 JoyC Omnidirectional Controller

A joystick module designed for M5StickC, that supports dual-handed operation [6] is show in Figure 2. It has a socket for a battery that allows M5StickC to work without a wire connection powered by long-lasting batteries, which is crucial for creating controllers. Joysticks have analog input range from 0 to 200. Its input mechanisms make it an ideal choice for controlling and interacting with the software solution, with a smooth and responsive user experience.

Figure 2. M5StickC and JoyC Omnidirectional Controller.

3.3 Hardware Integration

Communication between the ESP32 microcontroller and a Raspberry Pi unit within the same WiFi connection is a crucial aspect of the software solution. To transport data from one machine to another, those two must speak through transport layer protocol, known as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a connection-oriented

protocol, meaning the server and client must establish a connection before any data exchange can occur [7]. This protocol creates reliability since the client gets feedback if data reaches the server and automatically retransmits if it does not [7]. Used in applications and systems where it is essential to receive data without losses and in the correct order, for example, text communication, transfer for files, and in Simple Mail Transfer Protocol [8]. On the other hand, UDP is a connectionless protocol, meaning a connection between server and client is not required [7]. Best for use cases where continuous data transmitting is needed, suits well for online games, video conferencing, and Voice Over IP [8]. This protocol sends datagrams to the IP, and from there, the server picks up those datagrams and the client's IP [7].

While UDP lacks reliability with potential packet loss from one-sided communication, compared to TCP, it still has faster transmission speeds, making it more suitable for low latency experience. In this case, I started using UDP because the game controller does not require a two-way connection, and thanks to that, the receiver does not need to make a connection with all controllers that are sending information to it. Packet loss and incorrect ordering are not issues because controllers continuously send out information about the current state and, in the long run, fix all lost or inaccurate ordering of data.

3.4 Flow of the System

The communication system between two devices in same network, illustrated with Figure 3. Data flow is mostly one-sided, meaning that controller sends out data package, but does not get feedback from the receiver. An outlier is while conducting tests, the host machine is sending back the package for controller to timestamp for calculating delays. The hosting machine serves two roles, functioning both as the host for a game and executing the receiver code responsible for managing incoming data. The client device acts as information source, capturing user inputs and transmitting them over the network. The receiver code initializes a socket server within the local network environment. This server is link between game hosting machine and client devices. It listens for incoming data packages and processes them accordingly. Additionally, this code emulates a virtual gamepad. Data is interpreted to corresponding joystick movements and button inputs states. These are then relayed to the game running on same machine. Client devices, such as computers, smartphones, or dedicated input devices, may vary in form and type. Client devices must format data correctly and transmit it over the local network to the designated IP address and port.

The flow of data begins with the M5StickC, which is receiving input values from the JoyC Omnidirectional Controller as well as from its own buttons. Inputs are then formatted and sent through the local network to IP address on the Raspberry Pi. The server is then capturing these packets, processing and utilizing within the game host system.

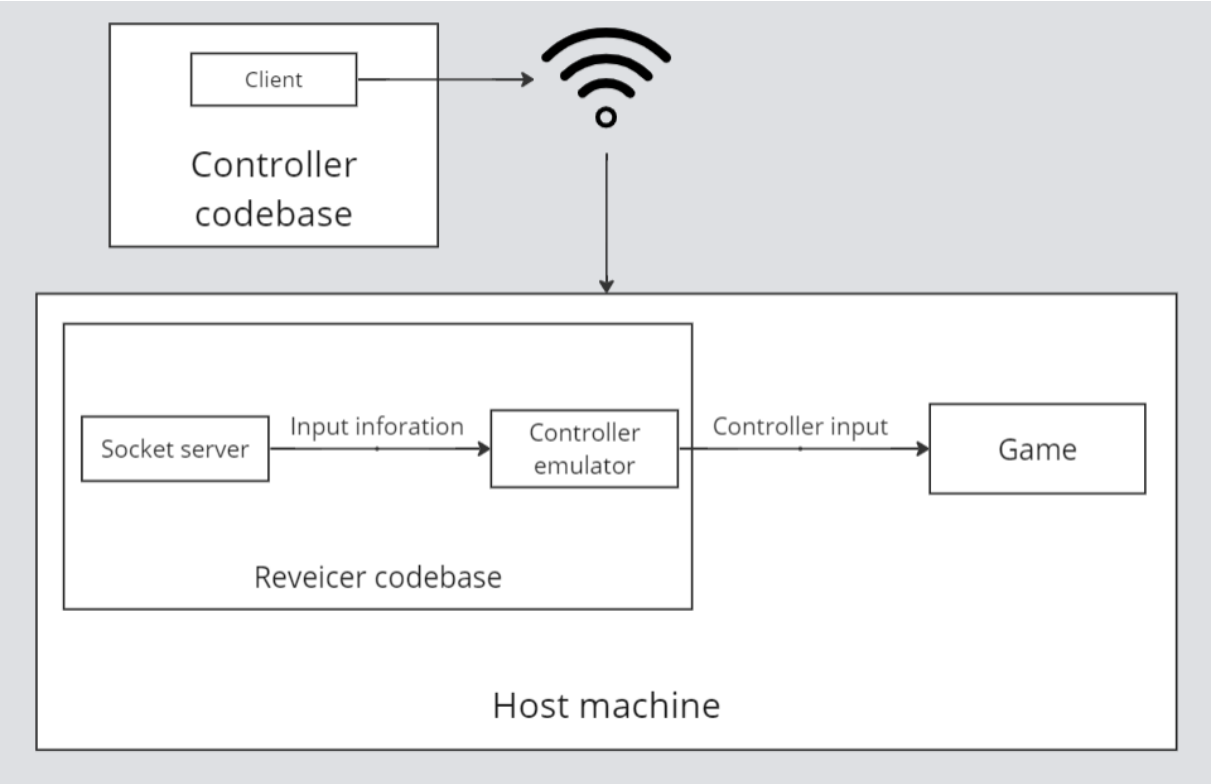

Figure 3. System flowchart.

3.5 Computer Graphics Projects EXPO 2024

The Computer Graphics Projects EXPO 2024, held on 26th January 2024, served as a platform for students and the local entertainment software industry to exhibit and evaluate various projects spanning video games, virtual reality, and computer graphics [12]. Among the showcased projects was SINGIT, which featured a custom controller solution. Participants had the opportunity to engage with the game using both traditional Xbox controllers and a custom controller created with JoyC and M5stickC technologies. Behind the scenes, data transmission occurred via a WiFi connection from the M5stickC device to the Raspberry Pi.

In conclusion, the Computer Graphics Projects Expo 2024 was deemed a success despite the time constraints faced during the implementation phase of the JoyC and M5stickC technologies, which I acquired mere days before the event. Despite this limitation, the system underwent brief testing before the expo, during which minor bugs were identified and promptly addressed.

However, user feedback highlighted latency as the predominant issue with the custom controller. Participants reported challenges in maneuvering characters precisely within the game, exacerbated by the time constraints imposed by each level. Nevertheless, amidst these challenges, it was noted that the custom controller exhibited slightly better performance than the standard Xbox controller. The additional issue faced in the event was that the Xbox controller constantly disconnected from Raspberry and repeatedly needed to be reconnected. In large-scale events with many exhibitions, this issue causes inconvenience because the controller must be reconnected many times. This experience underscores the potential of further refinement and optimization of the custom controller solution for future iterations, demonstrating the value of iterative development and user feedback in enhancing the overall user experience.

4. Software

4.1 Development Environment

Thanks to its extensive libraries and straightforward syntax, I chose Python for its ease of realizing and developing small-scale programs, projects, and scripts. Additionally, Python offers excellent accessibility and readability for feature development, making it ideal. The codebase is regularly updated, ensuring compatibility with the latest features and improvements, and I can conveniently track it on GitHub for version control and collaboration. Regarding interoperability, I designed the Python implementation to integrate with other programming languages, particularly C++, which is widely used in robotics.

Visual Studio Code (VSCode) was chosen as the primary integrated development environment (IDE) because it supports extensions, facilitating the integration of tools and plugins to meet specific project needs. For the use of VSCode the decision was driven by my daily use of is, which offers essential extensions such as PlatformIO for Arduino development and extensions for C++, enabling the convenience of coding receiver and controller code within a unified workspace.

4.2 Software Solution

4.2.1 Library for Network Input

Python library libni [9] was developed by Ulrich Norbisrath, the supervisor of this thesis. It is an open-source codebase designed to facilitate the communication between networked input devices and applications. With libni, developers can easily implement support for various input devices such as game controllers, motion sensors, or specialized peripherals like VR or AR.

The utilization of the libni library, last updated eight years ago, posed significant challenges in integrating networked input functionality into our codebase. Despite efforts to configure the environment with the appropriate Python version and install requisite libraries, compatibility issues with newer Python versions and deprecated dependencies persisted. Addressing initial matters led to a cascade of new problems, rendering the library codebase inoperable. Consequently, I abandoned the idea of using this library for custom controller implementation since creating a new codebase seemed to give better results.

4.2.2 Virtual Gamepad

Game *SIGINT* is for two to four people and is designed to be played with a controller [2]. Upon exploring options to emulate a gamepad with Python, we found a library that did just that. Developed by Yann Bouteiller, vgamepad is a Python library capable of emulating Xbox 360 and DualShock 4 gamepads, facilitating the transmission of analog inputs to games [10]. Utilizing this library, a program was able to interpret keyboard inputs into analog signals. This solution enabled the mapping of all requisite gamepad commands to keyboard keys. Thereby, the game could be played without a controller.

4.2.3 Data Format

In this project, the data transmission format follows a structured approach to ensure efficient communication between devices. The format is based on JavaScript Object Notation (JSON), which is to be read by humans and easy for machines to parse [11]. This gives data flexibility by allowing the addition of new key-value pairs based on specific needs.

The data transmission format consists of key-value pairs encapsulated within a JSON object. Not every value is required, giving flexibility and control over the receiver code without changing it. Only “player” and “data” key-value pairs are required. Each key corresponds to a specific piece of information related to the controller's joystick values and other relevant parameters. The structure of the JSON object is as follows:

- “player”: This key designates the intended recipient of the data transmission. It includes the local IP address of the sender, allowing the receiving device to identify the source of the data, and since IP is unique for every device in the same network, this value indicates for which controller emulator is data indented.
- “data”: Contains the actual joystick values captured by the controller. These values include the x and y coordinates of the joystick, as well as any additional input data relevant to the game or application.
- “Controller”: The vgamepad library allows using Xbox 360 and DualShock 4 gamepads. The values are for this “Xbox” and “ds.” By default, the Xbox controller is chosen if the value is within the package, making this not a required value to add to the package.

- “send_time”: Records the timestamp at which the data is dispatched from the sender. This timestamp is crucial for calculating latency.
- “type”: Specifies the data type of the transmitted information. Controls what type of action is to be done. By default, it is processing information to update the gamepad joystick and button states.

4.3 Writing a program

4.3.1 Receiver

The receiving part of the code went through many iterations. To understand how devices communicate with each other through sockets, I created a simple server that accepts TCP communication standards. Through research and the fact that many controllers are connected simultaneously, with a continuous stream of transmitting data, UDP was a better choice because of its better speed.

The next iteration consisted of adding vgamepad virtual controllers to the codebase. This version of the server received data containing player and input values separated with spaces in the following format: “P1 150 100 60 0”. The first part is the unique code to identify for which controller was the new state sent. The next two numbers represent values of the left joystick's x and y axis. The following number is for directional button values; if the number was greater or lesser than a certain threshold, it was interpreted as up or down button clicks. Moving the right joystick of the JoyC Omnidirectional Controller up or down was read as clicking directional up and down buttons. The last value in this format showed whether the button on M5StickC was clicked, and it signaled the virtual Xbox controller to click the “a” button. This form included many issues. It threw errors every time there was a slight inconsistency in data. Sometimes, due to UDP communication, some messages are received simultaneously, merging them and causing errors. To counter this, the controller added a semicolon to the end of every message so the receiver could separate messages.

The last iteration of the code first sets up a communication system between the server and multiple clients using UDP sockets. The server continuously listens to incoming client messages, parsing them into JSON data to be utilized. The data is expected to be in the format mentioned in the 3.3 paragraph. The only requirements for controller devices are that they could be connected to a network via UDP connection and transmit data in the same format. After confirming the data correctness, the new Xbox 360 or DualShock 4 virtual emulator is created by vgamepad, depending on which type of controller is required by the key value

“controller.” Then, inbound data is processed and adjusted to refine joystick positions and button states, creating a smoother gameplay experience. Finally, the updated state of the gamepad is sent to the host machine. The script accepts and handles new client connections until they are stopped.

4.3.2 Python Controller

The first version of the controller is in Python, the reason behind it was a faster development cycle due to prior knowledge and not including any extra device into the environment since the controller was running on the same machine as the receiver. The controller's first iteration includes the server's client side, which can send messages.

The next improvement was interpreting keyboard buttons to corresponding gamepad actions to be able to control player movements in the game. For example, keys “w,” “a,” “s,” and “d” represented the maximum and minimum values of the left joystick. This version had vgamepad virtual controller included without realizing that playing games this way only works if controller code was executed on the game host machine. After understanding the mistake, I moved parts of the code containing the gamepad emulator to the receiver codebase.

The following iteration monitors keyboard inputs for two players, which is the minimum requirement to complete *SIGINT* successfully. The recorded data is formatted accordingly and dispatched to the server. This marked the completion of a minimum viable product, and to ensure that development was going in the right direction, I measured the speed of communication. For that, this script should be able to receive messages; for this reason, I added the server to the controller code. In this way, added timestamps of send and receive times could be compared. Recorded values showed low transmitting times, confirming the development direction.

The last iteration was improving the server side of this Python script. By this time, M5StickC was already programmed, and *SIGINT* was playable. Conducting tests on package loss and latency, the information needed to be stored, and since the last version of controller code already had server setup, it was convenient to add data recording to this server. This functionality was to receive data, calculate the time between the first and second timestamps, and store the information in a text file, creating a database for analysis.

4.3.3 Arduino Controller

With the help of PlatformIO extension, I programmed ESP 32 to inject data into a local network server. The main functionality of the code involves capturing inputs from joystick controllers and buttons, processing the input, and transmitting it over a local network connection using UDP. Upon initialization, the code establishes a WiFi connection and configures communication parameters. In the main loop, the code is monitoring the state of joysticks and buttons, parsing values to JSON objects and sending values to specific hosts. Display is utilized for human interaction, dynamically updated to reflect the values of both joystick axes, as well as the state of the M5StickC button. This visual effect offers users immediate feedback regarding the positions and movement of the joysticks and the state of the button. The codebase is accessible via the following link: https://github.com/raiko-valo/thesis-2024.

5. Testing and Analysis

5.1 Test setup

The configuration of the testing setup included a various already in use controllers and a custom controller equipped with developed software. To get as accurate testing results, the I conducted tests in environment that mimicked real-world use case scenarios as closely as possible, it took place in corridor next to CGVR Study Lab. Raspberry Pi served as the central hub for testing by running *SIGINT* and receiving and echoing back input data. Latency and package loss testing were analysed based on captured input packages. Personal computer was employed to collect and store the completed packages, meaning it had sending and receiving timestamps added. To make comparisons I recording controllers with the screen at the same time to analyse footage afterwards.

5.2 Analysis

5.2.1 Latency

The methodology involves adding a timestamp to the outgoing data at the sender's end. Once it has reached the receiver, it is echoed back by broadcasting the same timestamp back. This process involves measuring the time when the data is successfully received by the sender, ensuring timestamp consistency within the same code runtime and device. This approach is crucial as it prevents any discrepancies with external time measuring devices, particularly significant given the precision of measurements in milliseconds, where even the slightest mismatch could have a big effect.

I conducted controller latency testing in two phases: the first phase involved testing with a single controller, while the second phase was done with four controllers simultaneously. Every calculation with given data is rounded to the ceiling to avoid digits and not to betray better results than were in reality. It is assumed that the time taken for sending and receiving data is approximately equal, resulting in estimations for the obtained results. Figures 4 and 5 present data without dividing time by half, ensuring accuracy in results. Those figures include over 90% of the measurements from both phases, providing a representation of the collected data. The reason for not including all records in Figures 4 and 5 was that there would have been too many unique values, making the figures unreadable due to overcrowding.

In the dataset for the single-controller testing, a total of 2013 time measurements were recorded, ranging from 5 to 291 milliseconds. Arithmetic mean time for sending packet to game host was

12 milliseconds, with a median of 10 milliseconds and a mode of 10 milliseconds. The longest time spent on receiving data from the controller was 146 and the shortest was 3 milliseconds. Smallest time for sending and receiving data across all tests was 5 milliseconds, possibly indicating a software limitation preventing faster transmission.

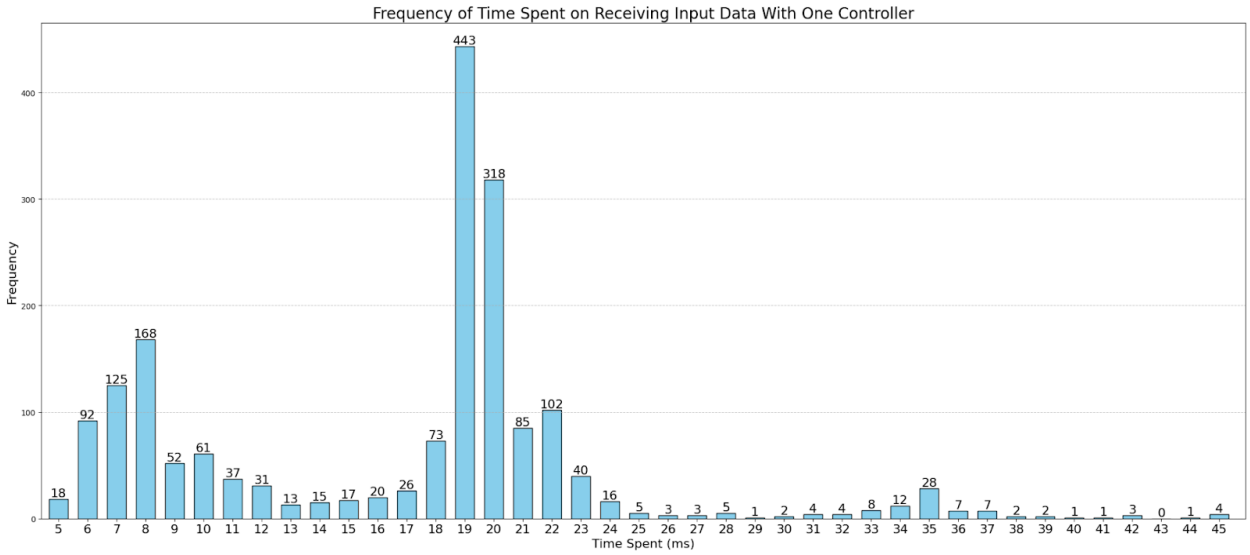

Figure 4. Frequency of Time Spent on Receiving Input Data with One Controller.

During testing with four controllers, a total of 2573 measurements were captured, spanning from 5 to 72 seconds. Inputs were inserted simultaneously with each controller and data was echoed back to the exact same machine, guaranteeing accuracy of timestamps. Contrary to expectations, the arithmetic mean and median were smaller this time, approximately 9 milliseconds, with a mode of 10 milliseconds.

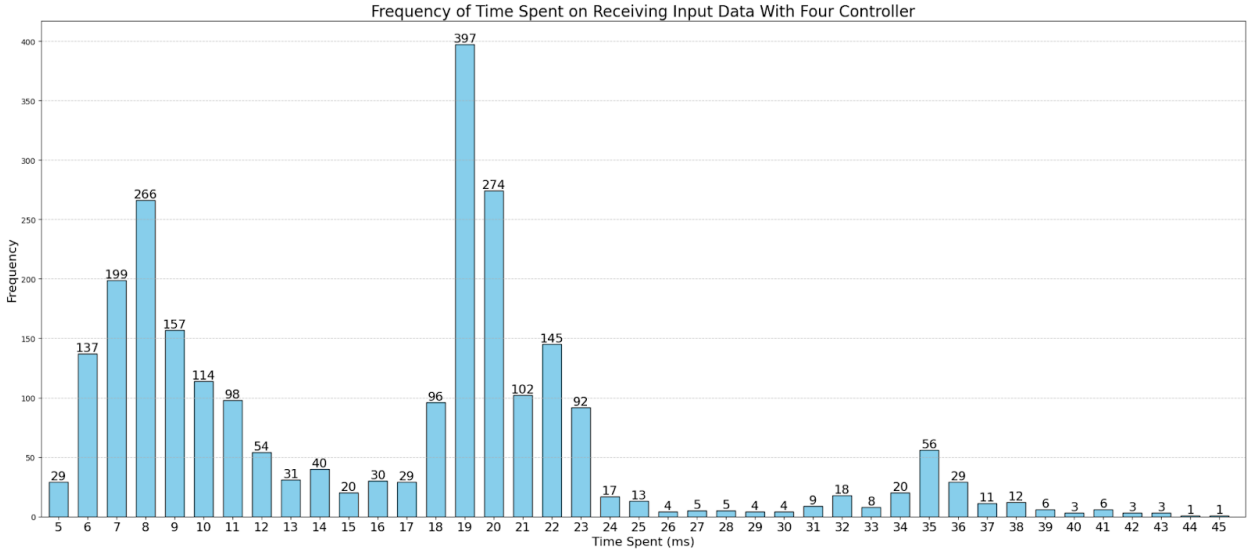

Figure 5. Frequency of Time Spent on Receiving Input Data with Four Controller.

5.2.2 Package loss

Using UDP for communication, some data may go missing and would not be received by the game host machine. To verify that the game is receiving data from the controller in a consistent manner, I conducted a package loss test. This test involved assigning a numeric value to each data packet, denoted by the key “id”, which incremented by one with each injection. By examining the received data, package loss can be calculated by finding the biggest id and then counting how many unique ids are missing. Total of 8346 recordings of input injecting were examined. Among these recordings, only 58 packages of data were lost on receiving, making the loss rate under one percent.

The low rate underscores the reliability of the communication channel. Despite the loss of a few packets, the majority of data was successfully transmitted, ensuring precise movement when using the custom controller. This validates the efficacy of the communication protocol and confirms the controller's ability to deliver consistent input to the game host machine

5.3 Comparison

The study examines the latency performance of three distinct controllers: an Xbox controller connected to a Raspberry Pi via a wireless adapter, a PlayStation (PS) 5 controller connected via Bluetooth, and an M5StickC with a JoyC Omnidirectional Controller connected through Wi-Fi transmission. To capture latency measurements, a smartphone capable of recording approximately 120 frames per second (FPS) was filming the controller movements and the glass panel at the same time in slow motions. Afterwards recordings were analysed frame by frame, finding latency between input initialization and observed movement.

The Xbox controller, has relatively high latency when used with Firefox, with an average of 94.125 frames or approximately 784.375 milliseconds. In contrast, the PS controller demonstrated low across both browsers, with an average delay of 12.875 frames or approximately 107.292 milliseconds on Chrome and 88.857 frames or approximately 740.476 milliseconds on Firefox. The M5StickC with a JoyC Omnidirectional Controller, represented by the ESP32, exhibits distinct latency profiles depending on the browser used. While Chrome displays a latency of 18.125 frames or approximately 151.042 milliseconds, Firefox shows a significantly higher latency of 57.25 frames or approximately 477.083 milliseconds.

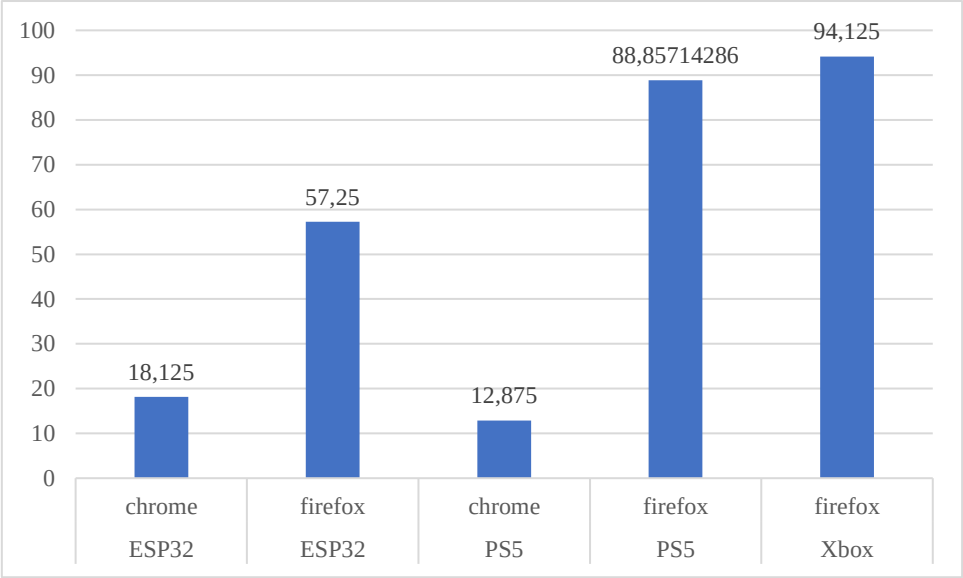

Figure 6. Delay comparison between browsers and controllers.

The results of tests and video analysis suggest that in comparison of different browsers and controllers, the delay between pressing the key and movement happening on screen is mostly affected by performance of the browser. Overall the PlayStation controller is performing best for everyday use, it has lowest latency out of those three controllers when using Chromium browser.

6. Conclusion and Future Study

As a result, I developed a system as a software and hardware solution for input transmission over a local internet connection aimed at gaming environments. The study aimed to create a controller and system with minimal latency, ensuring users a smooth and reliable way to play games on CGVR Lab smart glass panels. For controllers, I selected the M5StickC microcontroller to work as a brain because it uses ESP32, which has great computing power, low energy consumption, and WiFi capabilities. Pair it with the JoyC Omnidirectional Controller for joysticks and the battery socket.

The test results about the connection between the controller and host machine provide a seamless connection with low delay. It is as low as three milliseconds between sending out the data packet and the receiver picking it up from the connection. The mean time was around ten milliseconds, implying almost no delay in transmitting inputs through an internet connection. Packet loss is one of the most significant downsides of using UDP, but in this case, it was no problem since it was minimal, under one percent, making it almost impossible to recognize while it occurs. Through analyzing the video recordings, it became evident that the browser choice had a more significant impact on latency than the type of connection used. Chromium browser gives better results than Firefox.

I created the WiFi-based system because controller signals are passing through glass and other materials to reach the game-hosting Raspberry Pi. This may create potential issues in large-scale events, considering that many devices are connected using Bluetooth or other signals. The custom solution makes the controller more fitting for events, granting a higher range of connections and ensuring uninterrupted connection between the controller and the game host machine.

For future study, the custom controller could be finished by replacing the JoyC Omnidirectional Controller with an individual joystick and 3D printed handle to improve holding comfortability. This allows adding extra buttons to get a standardized layout with four directional, four action, four shoulder buttons, and double joysticks. It is making it more flexible for playing games with controller support.

Furthermore, the system should be tested in larger events, like the Computer Graphics Projects EXPO in upcoming years, since my testing was in a period when only a few devices were used nearby. By conducting the same type of tests, the outcome could underline if the controller

connected with WiFi outperforms widely used gamepads in this particular setup and use case. Latency and package loss would show how well the connection between M5StickC and Raspberry Pi is maintained and how large-scale events with many nearby devices hinder package loss and the speed of receiving data.

Developing the system with custom software and hardware solutions was a success. M5StickC, paired with JoyC Omnidirectional Controller, could compete and even excel with existing well-designed, refined, continuously improved controllers. The custom controller had no connection issues throughout the Computer Graphics Projects EXPO 2024 event.

List of References

[1] Team EMP. Leveling Up: The Latest Gaming Trends Dominating 2024. EMB Blogs 2024, 3. https://blog.emb.global/latest-gaming-trends/ (25.04.2024).

[2] Aksli M. SIGINT – A Cooperative Puzzle Game on Vertical Wall Panels. University of Tartu Institute of Computer Science. 2021. https://hdl.handle.net/10062/92039 (08.01.2024).

[3] Eridani D., Rochim A. F., Cesara F. N. Comparative Performance Study of ESP-NOW, Wi-Fi, Bluetooth Protocols based on Range, Transmission Speed, Latency, Energy Usage and Barrier Resistance. *2021 International Seminar on Application for Technology of Information and Communication (iSemantic)*. Semarangin: IEEE, 2021, 322-328. https://doi.org/10.1109/iSemantic52711.2021.9573246 (13.05.2024).

[4] Zha M. X., Zhang Y. The Effects of Network Latency on Player Gaming Experience. 2019. https://core.ac.uk/download/pdf/213002493.pdf (08.01.2024).

[5] M5StickC. https://docs.m5stack.com/en/core/m5stickc (28.04.2024).

[6] JoyC (W/O M5StickC) Omni-directional Controller. https://shop.m5stack.com/products/joyc-w-o-m5stickc (28.04.2024).

[7] Kalita L. Socket programming. *International Journal of Computer Science and Information Technologies*, 2014, Vol. 5(3), pages 4802-4807.

[8] GeeksforGeeks. Examples of TCP and UDP in Real Life. https://www.geeksforgeeks.org/examples-of-tcp-and-udp-in-real-life/ (13.05.2024).

[9] Norbisrath U. ulno/libni: Open library and examples for building and receiving commands networked input devices. https://github.com/ulno/libni (08.01.2024).

[10] Bouteiller Y. yannbouteiller/vgamepad: Virtual XBox360 and DualShock4 gamepads in python. https://github.com/yannbouteiller/vgamepad (08.01.2024).

[11] JSON. https://www.json.org/json-en.html (25.04.2024).

[12] The CGVR Study Lab. Computer Graphics Projects EXPO 2024. 2024, 1. https://cgvr.cs.ut.ee/computer-graphics-projects-expo-2024/ (01.05.2024).

I. License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Raiko Valo

1. grant the University of Tartu a free permit (non-exclusive licence) to:

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

Injecting Inputs Over Internet Connections

supervised by Ulrich Norbisrath,

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in points 1 and 2.
4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Raiko Valo

15/05/2024