Evaluating and Developing a Universal Legacy Voice Control TV Remote With a Mobile App

Amrit Kooner – 20134667

Course Name – Integrated Master's in Computer Science (MCs)

Department Name – Computer and Data Science

School of Computing and Digital Technology

Faculty of Computing, Engineering, and the Built Environment

Birmingham City University

Submitted May 2023

A thesis submitted to Birmingham City University in partial fulfillment of the requirements of the degree of Bachelor of Science.

Abstract

This report presents the development of a universal voice control tv remote aimed towards legacy TVs without built-in voice control functionality. Its features will consist of compatibility with all TV models, traditional button controls, AI/ML/NLP algorithms to improve upon its voice recognition, and also a mobile app that's connected to the remote via Bluetooth.

Throughout this report regarding the device, its entire development life cycle will be covered, including planning, methodology, designing, implementation, evaluation, testing, and a conclusion.

It highlights the impact of TV remotes on improving the user experience with those with visual or physical impairments. Overall, the report demonstrates the feasibility and success of creating a universal voice control TV remote for legacy TVs, that aims to provide a positive social impact.

Acknowledgements

Firstly, I would like to express my appreciation to my project supervisor, Stish Sarna, for his valuable guidance and insight upon completing my project. As well as my friends and family for always encouraging and supporting me, their faith in me was my main source of motivation and without that, I wouldn't have been able to complete my project.

Table of Contents

Glossary

µs	*microseconds*
CSAT	*Customer Satisfaction Score*

List of Figures

List of Tables

1 Introduction

This section describes the planning phase of the project, outlining the problem definition, scope, rationale, aims, and objectives. This section lays the foundation for the rest of the report by providing a clear understanding of the project's objectives and goals, demonstrating the importance of the project and how it will be implemented.

1.1 Problem Definition

The popularity of voice assistants and the growth of the smart home industry over the years has increased the demand for home automation devices with voice-control capabilities, such as AI assistants and voice-controlled TV remotes, offered by large companies such as Samsung, LG, and Sony.

However, the origin of the issue is that many legacy TVs are not compatible with smart voice-controlled TV remotes, and many non-smart TVs do not include any voice-control features. These larger companies mentioned don't offer alternatives for non-smart legacy TVs.

This issue is significant as non-smart legacy TVs lack voice-control features, and people with disabilities or mobility issues would be restricted from using the TV. Because of this their independence and quality of life may be significantly impacted since they could need help from others to carry out simple tasks like changing the channel or adjusting the volume.

1.2 Scope

The boundaries of the final project, state what will be included, and what will be excluded from it.

1.2.1 Includes

1. Voice control feature with several voice commands that will work on legacy TVs. This is the main feature of the device; it's how I will achieve the goal of the aim and solve the problem definition. Research shows that voice control is becoming an increasingly popular way to control devices.

2. Traditional button controls, as I want to give users multiple ways to use the device to achieve a leger target audience. It's known that traditional button control is still a very reliable way to control devices, which is why they are simply commonly used to this day.

3. AI/ML to improve voice control with NLP in terms of reliability and efficiency. Research shows that integrating AI/ML can greatly improve the accuracy of voice recognition devices.

4. A power button for the microphone so it is not always listening, will also save power consumption and minimize security concerns. Research shows that users are concerned about their privacy on voice-related devices such as mobile phones, as they think it's always listening to them.

5. LED to show if a valid voice command was entered to give visual feedback to the user. Visual feedback is important to indicate that the action was recognized and understood by the device. Research shows that providing visual feedback improves the user experience and reduces confusion.

6. The app can connect to the device via Bluetooth, as I want multiple options for users to use the device to achieve a leger target audience. This feature is unique for TV remotes which can improve accessibility for users. Research has found that smartphones have become an integral part of modern life, enhancing communication, productivity, and convenience.

1.2.2 Excludes

1. Support for multiple languages other than English, as I only speak English so it will be very difficult to integrate other languages. Research has shown that focusing on a single language can produce better results over numerous languages, which can be challenging and resource intensive.

2. A remote case for the device at this moment, as I don't think I can make it compact enough to fit it into one. If I did make a case however it would have been 3D printed using recycled materials.

3. Security features, so everyone should be able to use the device's voice recognition. Research shows that users prioritize ease of use over complicated security features for home entertainment devices, such as AI assistants.

4. Integration with voice assistants like Alexa or Google Assistant, as I don't want the device to be too complicated to use. Research shows that integrating with third-party voice assistants can be complicated and resource intensive.

5. Users to set their custom voice commands, as I don't want the device to be too complicated to use. Research shows that custom voice commands are useful, but not an important feature for most users.

6. An OFF/ON button as the device should be always listening for IR codes so it can be used via traditional buttons. Research shows that removing the button can reduce user cognitive load and create a more seamless and intuitive user experience.

1.3 Rationale

I chose this topic because I was interested in working with voice-related technologies, as they are essentially the future as everything is becoming more automated with voice recognition that can be controlled via voice commands.

To solve the problem definition (1.1), I aim to build a legacy infrared TV remote control with integrated voice recognition, that can also be connected and controlled with a mobile app to give users more options to use the device. It intends to provide easy and accurate voice-controlled features for non-smart legacy TVs without built-in voice-control capabilities, and to improve upon the reliability of present voice-controlled devices using AI/ML modeling, such as noise filtering. I aim it to be a cheaper alternative to present-day voice-controlled devices.

Companies within the field of home automation and AI could benefit from the device, as it could persuade them to offer more voice-controlled products that work on non-smart legacy TVs. Also, for elderly and disability organizations like care homes as the device aims to be a cheaper alternative to current voice-controlled TV remotes.

The device will benefit those who cannot afford smart TVs, so this device offers a cheaper alternative. It will also benefit those who have mobility disabilities or body impairments and those who want a generally better TV viewing experience. Overall, this project aims to have a positive impact on society and make people's lives simpler with its voice control feature.

1.4 Project Aim and Objectives

The main aim of the project is to create an infrared TV remote with integrated voice recognition that utilizes AI/ML with natural language processing to improve the efficiency and reliability of voice recognition, it can also connect to a mobile application to offer users more options and functionality. The device is to work on non-smart legacy TVs that are not capable of voice recognition. What I hope to achieve from this aim is for the device to positively impact society by making people's lives easier with its integrated voice recognition feature.

1.4.1 Objectives

- **Research** the main key topics and themes of the project, being Arduino, Voice Recognition, App development, and Natural Language Processing.
- **Study** existing technologies and platforms related to voice recognition.
- **Design** a prototype of the device using computer circuit designing software.
- **Model** digital wireframes for the app using graphics designing software.
- **Build** the initial device with all required parts including voice recognition.
- **Program** a noise filtering algorithm to model to improve the accuracy of voice recognition using NLP.
- **Develop** a mobile application with a GUI that wirelessly links to the device so can be controlled using a mobile application.
- **Test** all aspects of the project, being the device and the mobile application. As well as to make any required changes for errors and improvements.
- **Evaluate** the performance and functionality of the device, including the accuracy of voice recognition, the ease of use of the app, and the overall user experience.

1.5 Background Information

States the hypotheses and theory, which are to be tested in the course of undertaking the project.

1.5.1 Hypotheses

- The device having voice control, traditional buttons, and a mobile app can improve accessibility and convenience when controlling legacy TVs and increase user experience.

- Voice recognition technology that combines AI/ML and NLP will increase the accuracy and reliability of voice control for legacy TVs.

- Offering a mobile app for legacy TVs can attract younger audiences who prefer to use their smartphone as a remote control. Also, more convenient as people always have their mobile phones on them.

- Using a mobile app to control TVs instead of a physical TV remote can minimize the need for them, thus contributing to an eco-friendlier lifestyle.

1.5.2 Theories

- Voice control tied with AI/ML with NLP can make devices more accessible and inclusive for individuals with disabilities or limitations that may make physical remote control difficult.

- Voice control tied with AI/ML using NLP can cause devices to be more accessible for people with physical disabilities or visual limitations, which can make physical remote controls hard to use.

- Using voice recognition and mobile apps to control devices is part of a larger trend toward connected homes and the Internet of Things. The future may move away from traditional device remotes.

- Developing mobile apps for traditional TV represents an opportunity for innovation and market disruption in the TV industry.

2 Literature Review

2.1 Themes

The main themes of the project that will be explored within the literature review involve Computer Circuits Design and Development With Arduino, Analog-to-Digital Conversion, App Design and Development, Artificial Intelligence with Natural Language Processing, as well as Voice Capturing and Recognition. The other themes that I have previously mentioned in the Literature Search Methodology (2), will link into the main themes acting as sub-themes, being programming in C/C++ and Java. A mind map was created showing the thought process in Appendix A.

2.1.1 Computer Circuits Design and Development with Arduino

The first theme is Computer Circuits Design and Development with Arduino, this needs to be explored as it will be the main computer of the device as well as the foundation for all the other themes, so it is the first and most important theme. Various literature on existing Arduino infrared devices will be explored to see how they are configured. It links with C/C++ programming language which also needs to be explored as it will be the main language the device is programmed in. The analysis of this theme can help me identify how to set up the Arduino environment and determine what components to use and how to configure them.

2.1.2 Voice Capturing and Recognition

The second theme is Voice Capturing Recognition, this needs to be explored as it is the main feature of the device that differentiates it from becoming an ordinary infrared TV remote. It is also required so I can achieve the desired outcome from the creation of the device. The device will need to be able to capture and understand the user's voice, so various literature on existing voice-controlled devices will be explored to see how they work and what techniques were used. The analysis of this theme can help me better understand how voice capturing and recognition work, as well as what other concepts are involved.

2.1.3 Analog-to-Digital Conversion

The third theme is Analog-to-Digital Conversion (ADC), this needs to be explored as it's the process of analog input signals being converted into digital output signals which is the core functionality of the device so it can work. The user's voice inputs will be converted into IR code outputs with this process. I also want to monitor the analog and digital signals so thee the before and after the conversion. The analysis of this theme can help me better understand how the ADC process works, what other concepts are involved, existing libraries for monitoring analog and digital signals, and challenges within the ADC process.

2.1.4 App Design and Development

The fourth theme is App Design and Development, this needs to be explored as it is another feature of the device where it can be connected and controlled with a mobile device through a downloaded app. This is because I want to give users multiple ways to use the device. The theme links with the Java programming language which will be the main language this feature is programmed in, so this will also need to be explored. Various literature on existing devices with mobile application connectivity will be explored. The analysis of this theme can help me understand how to design and develop an interactive mobile application and how to distribute it to various app stores like Play Store and Apple Store.

2.1.5 Natural Language Processing (NLP)

The fifth and final theme is Natural Language Processing (NLP) involving Machine Learning (ML) and Artificial intelligence (AI) that will be used for processing and cleaning analog signals. This needs to be explored as analog signals will be improved by using a filtering technique that will remove unwanted noise. The analysis of this theme can help me better understand how NLP works, and what else can be achieved from utilizing it which I may also implement alongside filtering, what programming languages and libraries are involved, as well as to identify suitable AI/ML filtering algorithms.

2.1.1 Keywords of Themes

The table shows the keywords for each theme that will be utilized to obtain related information from pieces of literature.

Theme	Keywords
Computer Circuits Design and Development with Arduino	• *IDE* • *Components* • *Voice components* • *Infrared* • *IR codes* • *libraries* • *C/C++ Programming* • *Troubleshooting* • *Debugging* • *Arduino software* • *Circuit designing software.* • *Arduino devices*
Voice Capturing and Recognition	• *Techniques* • *Voice commands* • *Voice capture* • *Voice control* • *Voice recognition*

	• *Voice-based devices* • *Speech* • Arduino
Analog-to-Digital Conversion	• *Signal processing* • *Signals* • *Digital signals* • *Analog signals* • *Output signals* • *Input signals* • *ADC* • *DSP (digital signal processing)* • *Sampling* • *Resolution* • *Signal compression* • Arduino
Natural Language Processing	• *Filtering* • *Noise reduction* • *Noise removal* • *Techniques* • *Algorithms* • *Libraries* • *Deep learning* • *Artificial Intelligence* • *Machine Learning* • *Arduino*
App Design and Development	• *Java Programming* • *IDE* • *Libraries* • *Software* • *Mobile SDK* • *Bluetooth* • *Android* • *iOS* • *App designing software* • *App development* • *UX (user experience)* • *UI (user interface)* • *Arduino*

Table 1 – Keywords of Themes Table

2.2 Review of Literature

Eight pieces of literature were researched and reviewed for each theme better understand to better understand them using search terms (1.2.2) and keywords (2.1.6). I research and referenced reports, articles, and books with the majority having a high citation count.

2.2.1 Computer Circuits Design and Development with Arduino

"Simon Monk" (2019), covers the fundamentals of the Arduino microcontroller. It focuses on topics such as how the board works, how to set up the Sketch IDE, and a tour of the board's interface explaining the role of each I/O serial port, GPIO pin, and microprocessor. It was stated that the Arduino Uno is the most commonly used board in the family and the best beginner board for developing small to medium projects. It also dives into Arduino programming language which is a simplified version of C/C++. ***"Brian Evans" (2011)***, explains how to use common circuit components such as capacitors, resistors, diodes, and transistors.

"What is Arduino?" (2015), provides common potential errors and troubleshooting methods revolving around uploading code on the Arduino, being the board not getting recognized by the IDE and not listed under devices, IDE not uploading after the upload button is pressed, and the "avrdude: stk500_recv()" error message when the programmer is not responding. Arduino has lots of community support and documentation, the best bet when running into unexpected problems is to search online for advice. Forum websites like Reddit and Stackoverflow will be used to check if someone has already come across the same error as me, if not I will post a form explaining the issue on multiple threads.

TV remotes emit IR codes consisting of 0's and 1's that are unique for each button, that's then received and computed by the TV to perform the designated action, ***by "Duncan Wilson" (2021)***. "***Chauhan Naman" (2017)***, covers the process of finding the IR codes on a TV remote using an Arduino with the IRremote library.

"Sadat Hasan Shehab" (2020), developed a voice recognition-based home automation system that uses a voice control module to perform speech recognition with the LabVIEW library to monitor sound wave frequency patterns. ***"Rosalyn R Porle" (2022),*** developed a speech-based number recognition system compatible with the Malay language using a voice control module. Malay numbers one through ten are recognized and corresponding LEDs that link to each number light up. ***"Khoa N. Van" (2018),*** developed a text-dependent speaker recognition system using a voice control module, that identifies different users based on their previously stored voice samples.

2.2.2 Voice Capturing and Recognition

“Rafizah Mohd Hanifa” (2020), covers voice biometrics with advances made in the last decade and challenges in this area of research, like the issue with white noise. It states the characteristics of the voice biometric, with accuracy being high, ease of use being high, user acceptance being high, ease of implementation being high, and cost being low. Other concepts used within voice recognition include algorithms such as VQ to improve accuracy and speed, “***Jianliang Meng” (2012)***.

Spoken language is effective for human-human interaction but often has severe limitations when applied to human-computer interaction as they are slower at processing this information, so single-worded to short sentences as commands are used for computers to better understand our speech, “***Ben Shneiderman” (2000)***. “**V Radha” (2012)**, compares different approaches and methods used for voice capturing, being spontaneous recognition that can interpret non-words and sentences such as slang, connected/continuous recognition that can interpret connected words to sentences with minimal pauses between them, and isolated recognition that only interpret single worded commands. The speech recognition system can be classified based on the type of words, vocabulary size, and speaker dependency, “***Pratik K. Kurzekar” (2014)***. It was stated that the vocabulary size of a system affects the complexity, processing requirements, and accuracy of the system.

“Shweta Singhal” (2015), developed an Automatic speech recognition system using the connected/continuous voice recognition approach. HMM, and DTW is used at the back end for feature mapping of unknown words, with MFCC for feature extraction. “***Seok Yeong Jeong” (2006)***, developed a speech recognition for the automobile navigation system using the connected/continuous voice recognition approach. Uses the two-pass search algorithm that can manage a large vocabulary list at a high speed. “***Priyanka P. Patil” (2014)***, developed a connected word speech recognition system using the connected/continuous voice recognition approach. Uses STE and SC algorithms for speech segmentation, and MFCC and HMM for feature extraction.

2.2.3 Analog-to-Digital Conversion

"B. P. Lathi" (2009), explains the process of converting analog input signals such as sounds to digital output signals as binary codes, so systems can understand the human voice. The device will be capturing sound as analog signals, so a microphone will be used. An ADC microprocessor is used for the analogy-to-digital conversion process, Arduino has one built into the board, "***Brian Evans" (2011)***. The analogRead()/digitalRead() function can be used to measure and read analog/digital inputs from the analog input/output pins on an Arduino. "***W.A. Smith" (2022)***, demonstrates how to use this function and display raw analog values.

"Li Lee" (1996), discusses the process of the normalization technique with ADC, where analog inputs are scaled to fit within the input range, so analog signals can be accurately sampled and converted to digital signals, as outside the input range they would not be accurate. Some types of normalization algorithms that are suited for scaling signals are Min-Max, Energy, and Z-score normalization.

"Ze-bin Wu" (2019), discusses how the vector quantization algorithm can be used to represent the digital output of the ADC in a more efficient way using lossy data compression by distribution vector signals. It produces an output of higher resolution with many bits per sample by reducing the amount of data required to represent the signal. It works by partitioning a high-dimensional vector set so they are stored more efficiently, but some information is lost. It sacrifices the quality of information (accuracy) for efficiency with compression (speed). Many VQ algorithms were explored, such as classified VQ, feedback VQ, and fuzzy VQ. I may implement this algorithm within the device if the ADC process is too slow.

"Kunjabihari Swain" (2021), utilizes the LabVIEW library which possesses interactive real-time visualization tools so analog and digital signals can be readable and understandable for monitorization for comparison and analysis. MATLAB library is similar to LabVIEW as it also has visualization tools that can be used to view analog and digital signals. It also offers pre-processing tools for cleaning and preparing data like for noise filtering. ***"Onur Toer" (2019)***, utilized the MATLAB library for visualizing raw signals.

Unwanted noise and other distortions in the analog input signal can affect the accuracy of the digital output of ADC. So techniques should be considered to improve the performance of the ADC process by minimizing the effect of noise on signals, with filtering, decimation, and noise shaping algorithms, "***Zilong Jiao" (2018)***.

2.2.4 App Design and Development

"Ghita K" (2018), covers the best practices in the development and management of mobile applications with the essential concepts of the field, such as interface design with user experience, developing, testing, security, and distribution. Various technologies and tools used in mobile app development are analysed, including frameworks like Android and iOS, as well as programming languages such as Java and Kotlin which was one of the sub-themes. Programming languages and tools for developing mobile apps are platform-specific, for example, Android apps are created in Java via the Eclipse-based Android SDK, whereas Apple iOS apps are developed using either Objective-C or Swift via the XCode tool, "***Ivano Malavolta" (2016)***.

The Arduino Uno does not have integrated Bluetooth technology which is required for connectivity with mobile devices. A Bluetooth module should be connected to the Arduino to enable Bluetooth functionality with other devices; however, they have connectivity short ranges, ***"Ján Hurtuk" (2017)***. A Bluetooth module was added to the system requirements.

"Anshuman Kamboj" (2019), developed a Portable Water Quality Testing Device as an android mobile application, that was created using the MIT App Inventor web application. ***"Jignesh Patoliya" (2015)***, developed a war field spy robot with night vision capabilities and the wireless camera that can be viewed from an android mobile phone with an application, that was created using the MIT App Inventor web application. "***Seree Khunchai" (2019)***, developed a smart home system that can be controlled via an android mobile application, that was created using the MIT App Inventor web application. It works by dragging and dropping components into a design view and using a visual block language to program the behaviour of the application, "***Siu-Cheung Kong" (2019)***. It is popular in circuit development with microcontrollers like Arduino because it's easy to program the application to interact with physical devices and sensors.

"Hwansoo Kang" (2015), covers how to use MIT App Inventor to develop android mobile applications. The development process is composed of five stages, which are discovering an idea, designing with wire-framing, prototyping, implementation/testing, and deployment.

2.2.5 Natural Language Processing (NLP)

"Terena Bell" (2021), explains that NLP is a branch of AI/ML used to train computers using algorithms to understand, process, and generate human language, as well as to improve existing computers with data cleaning and pre-processing. NLP is heavily used within various voice and text-based technology, such as search engines, machine translation services, and voice assistants. NLP has many challenges such as ambiguity where sentences and words have two or more interpretations, and synonyms can lead to issues because many different words are used to express the same idea, "***Inés Roldós" (2020)***.

One of the most challenging problems in digital signal processing is to receive the information signal without any loss. It is important to reduce random noise and improve the performance of the signal, "***Hina Magsi" (2018)***. Cleaning and pre-processing techniques within NLP will be used for the device for filtering unwanted noise from analog signals. The common method for the removal of noise is the optimal linear filtering method, and some algorithms in this method are Wiener, Kalman, and spectral subtraction technique filtering, "**Kshipra Prasadh" (2017)**. "***Jacob Benesty" (2005)***, states that it is the Weiner Filtering algorithm is the most fundamental and popular approach as it works well with other cleaning and pre-processing algorithms.

It is not possible to apply NLP techniques directly on the Arduino platform, as NLP typically requires a lot of processing power and memory, which are limited on Arduino boards. However, external computers can be used to perform NLP tasks and communicate back and forth with Arduino. Python offers useful and powerful libraries for NLP, such as NLTK and spaCy with NLTK being the most popular NLP Python library, "***Claire D. Costa" (2020)***. C++ also offers useful and powerful libraries for NLP, such as text2vec and MeTA, "***Steve Emms" (2019)***. Both Python and C++ NLP libraries were researched as I still was not sure what programming language to use for it. For now, I will stick to NLTK with Python as there are more resources online that could assist me.

"Nitin Hardeniya" (2016), covers NLP essentials using NLTK with Python involving various cleaning and pre-processing concepts and algorithms, such as filtering algorithms for removing unwanted noise, feature extraction that can also be used for filtering, and normalization for scaling to eliminate redundancy and inconsistent dependency in signals. Each technique is explained and demonstrated. Some noise filtering algorithms analysed are Lemmatization, Stemming, stop word, and transforming chunks and trees. It was stated that it is a good resource for those wanting to quickly master NLP with the NLTK library.

2.2 Software Development Methodology Review

A review of methodologies to better understand them to then identify the most suitable one to use through the development of the device. I review the Agile, DevOps, Lean, and Waterfall methodologies as these are some of the top methodologies used for software development.

2.2.1 DevOps

The DevOps methodology emphasizes collaboration and communication between development and operations teams. Its goal is to accelerate development, increase efficiency, and produce quality software that meets user standards. DevOps consists of seven phases including planning, development, continuous integration, implementation, monitoring, feedback, and improvement. It can be used by independent developers or teams but implementing it can be challenging and requires changes to an organization's working environment.

For this current project, DevOps is not suitable as it is more suitable within a team environment, although it can still be used independently, even though it's still useable independently, implementing and following it will be challenging alone. Using a methodology that benefits me to work independently is better for the project he worked on independently.

However, successful implementation depends on effective collaboration and communication between teams. Despite its challenges the benefits of DevOps, including faster development, higher quality software, and increased efficiency, make it a promising approach for this project, "Damonlang Lamare" (2023).

Strengths –

- Provides the flexibility to regularly update and add features to the TV remote and app.
- Encourages continuous feedback, which can help the development team quickly identify and resolve issues with the TV remote and its app.
- Accelerate development and deployment, potentially bringing the TV remote and its app to market faster.

Weaknesses –

- Compatibility issues can appear with different TV and mobile models, and resolving these issues with DevOps can be challenging.
- Focus heavily on automation, which is not always appropriate for certain aspects of app development, such as GUI/UX/UI design and testing.
- May not be compatible with older systems, which is what the device is aimed towards, legacy TVs.

2.2.2 Lean

The Lean methodology is developed to maximize productivity and minimize waste. It consists of five stages, being value, value stream, flow, pull, and perfection. The Lean approach is well-suited for projects that require flexibility and adaptability throughout the development process. The goal is to accelerate the software development process, increase efficiency and quality, and reduce costs.

For the current project, a Lean approach may be a good choice, as requirements are subject to change during development. Lean focuses on flexibility and adaptability, which can be particularly useful when working on a project with changing requirements such as this.

However, implementing the lean methodology can be challenging and may require changes in the organization. Despite this, the Lean methodology's ability to optimize the development process and increase efficiency while maintaining a focus on customer value makes it a powerful approach for this project, “Jon Terry” (2018).

Strengths –

- A customer-based approach helps ensure that the final product meets user needs. This is especially important for electronics such as TV remotes.
- Help develop and refine the remote control's voice recognition algorithms, which is a main feature of the project.
- Helps minimize waste to reduce development costs for the remote and app.

Weaknesses –

- Difficult to ensure that all aspects of the TV remote development process are completed, and the project is on schedule.
- Difficult for a solo developer to track progress and ensure that all aspects of device development are completed on time, especially if developing multiple features such as devices, apps, and AI/ ML/NLP.
- The customer feedback approach may not be effective in improving the developed algorithm, as the customer may not be able to provide the technical feedback needed to improve the algorithm.

2.2.3 Waterfall

The Waterfall Method is a linear process consisting of six stages, being requirements, design, implementation, integration, deployment, and maintenance. While simple to understand and manage, it is also inflexible, making it difficult to adapt to changing requirements once a phase is completed.

For the current project, the waterfall approach is not suitable. This is because the requirements of the project may change during development and the Waterfall methodology cannot adapt effectively to these changes.

Overall, the Waterfall method is a well-defined approach that works best for projects with stable requirements that will not change throughout development, "Leeron Hoory" (2022). However, it is not suitable for projects like mine, which require flexible and adaptive development.

Strengths –

- A comprehensive testing phase could help ensure that the device's technical requirements, including AI algorithms for voice recognition, are thoroughly tested and functioning as intended.
- Easier to manage the device's technical components, such as app integration and traditional button controls, etc.
- Ensures that all technical requirements and testing, including app, voice commands, button controls, and AI, are well planned, documented, and thoroughly tested.

Weaknesses –

- Not suitable for devices that require constant change, such as a voice-controlled TV remote using AI algorithms, as it can be difficult to anticipate all potential issues and requirements in the future.
- May not be well suited for complex devices like this project.
- Voice recognition can be a complex and iterative process that may not work well with the linear development approach of the waterfall.

2.2.4 Agile

Agile methodology is a very flexible approach to software development that can adapt to changing requirements during the development process. It's about breaking the project into smaller, more manageable pieces called sprints to constantly refine and improve those pieces until the final product is complete. Agile consists of six phases being, requirements, design, development, testing, deployment, and verification.

For this current project, Agile may be very suitable due to the need for adaptability and flexibility, as the requirements of the project may change during development. It will allow the need for changing quickly to new requirements or customer needs.

Overall, Agile is a flexible and efficient approach that allows developers to deliver high-quality software flexibly. But it may not be appropriate for all types of projects, “wrike” (2021). It may not be appropriate for all projects, such as those that are too large or complex.

Strengths –

- Suitable for devices with AI voice recognition algorithms to constantly change and adapt to changing needs.
- Customer satisfaction is very important in Agile, which can help ensure the device will meet user standards, which is good as the device is aimed toward the public.
- Resolve issues earlier in development, reducing overall costs and improving the device's reliability.

Weaknesses –

- Sometimes result in device security or scalability being overlooked, which can be a problem for AI devices and mobile apps.
- Lead to insufficient testing, which can result in bugs in the app and AI algorithms.
- A flexible development schedule can be challenging for a hardware device like the voice control TV remote with the app and AI, which requires specific timelines for testing.

2.3 Discussion and Recommendation

2.4.1 Literature Review Summary

After conducting the literature review (2.2), various useful guides, techniques, algorithms, libraries, and circuit components have been identified. These findings will lead to future choices being made within design and development, as it's understood what techniques, libraries, algorithms, and circuit components to use for the development of the device, as well as the initial order of development that will be conducted. Also achieved a better understanding of each theme and how they are essential, and the various concepts involved.

- Techniques

Techniques found were the voice recognition type of the device, which is connected/continuous recognition that can interpret connected words to sentences with minimal pauses between them. The device will have two worded commands like “volume-up” and “volume-down”. Two different techniques were found for removing unwanted noise from signals, being feature extraction and filtering. Also, the analogRead() and digitalRead() functions can be used for reading analog and digital signals.

- Libraries

Libraries found were NLTK for Python which offers NLP tools, as well as MATLAB and LabVIEW which can be used for visualizing analog and digital signals, as I want to visualize the ADC process. The IRremote library for finding the IR codes of a TV remote will be important when each button needs to be mapped to a voice command, so the correct IR code is outputted. Also, the Scipy.signal library that provides the wiener filter algorithm that I will use for removing unwanted noise in signals.

- Components

Circuit components found were a voice recognition module as a common pattern found amongst pieces of literature developing voice-based devices with an Arduino, as it can capture voice analog signals. Also, a Bluetooth module, so I can connect a mobile phone to the device, is needed, as the Arduino does not have integrated Bluetooth technology. These components were added to the device requirements.

- Algorithms

Algorithms found were for filtering unwanted noise, such as Weiner, Kalman, and Spectral subtraction filtering. Some algorithms were found for data cleaning and pre-processing with vector quantization, such as K-means and fuzzy VQ for data compression to increase ADC pre-processing and data cleaning. Also, normalization algorithms, such as Min-Max, Energy, and Z-score normalization for scaling eliminate redundancy and inconsistent dependency in signals. And feature extraction, such as MFCC and HMM can also be used to remove unwanted noise from signals.

- Guides

Several guides were found that will be helpful for the development of the

device. One guide focuses on NLP NLTK, covering essential concepts and algorithms for cleaning and pre-processing text data, another guide covers the fundamentals of Arduino and C/C++, along with troubleshooting common errors. I also found one that uses MIT App Inventor to develop mobile applications, which will be useful for connecting the mobile app with Arduino. The development process involves discovering an idea, wire-framing, prototyping, implementation/testing, and deployment, which I will follow.

2.3.1 Limitations and Options of Themes

From the review of the literature (2.2), various limitations were found within the themes. In the Computer Circuits Design and Development with Arduino theme, it was discovered that the Arduino had no Bluetooth technology which is required so the device can link to a mobile phone, so it can be controlled through a mobile application. I already found a solution for this by using a Bluetooth control module which I added to the system requirements, but now I would have to implement this, so it makes the device more complicated to develop and it increases development costs.

In the Natural Language Processing theme, the limitations were that few recourses involved Arduino development using NLP and with the NLTK library that offers NLP algorithms. So, I will continue to do more research on this theme to find resources that can help me. Another limitation of the NLTK library is that it does not offer any filtering or feature extraction algorithms such as the Weiner or MFCC algorithm which I want to use to remove unwanted noise from signals. So, I will use the Scipy.signal library instead as it offers the Weiner filtering algorithm and the MFCC feature extraction algorithm. But NLP with the NLTK may still be used as it offers data pre-processing and cleaning techniques for normalization and vector quantization. It was stated by ***"Inés Roldós" (2020)*** that NLP has many challenges such as ambiguity where sentences and words have two or more possible interpretations, and synonyms can lead to issues because many different words are used to express the same idea.

In the voice recognition theme, ***"Rafizah Mohd Hanifa" (2020)*** stated many limitations of voice recognition in voice biometrics. Variability issues such as emotions, rate of utterances, mode of speech, disease, and the speaker's mood. Also, voice signals may change due to different transmission channels, such as the separate types of microphones and headphones used. Background noise is a significant factor that impacts the accuracy of voice-based devices, such as white noise or music worsens the performance of voice capturing and recognition, accuracy is high for clean samples and low for noisy samples with no impact. So, I have to make sure a sufficient filtering algorithm must be implemented, and I may develop and test multiple of them to identify and use the best one.

I discovered two different options that can be used to remove unwanted noise from signals, being filtering and feature extraction. I will use the filtering option as it was stated in "***Jacob Benesty" (2005)***, that the Weiner Filtering algorithm is the most fundamental and popular approach as it works well with other cleaning and pre-processing algorithms, so it will work well with Vector Quantization and Normalization which I may also implement. However, I will use feature extraction as a backup option if filtering fails.

2.3.2 Final Methodology Conclusion

After the review of methodologies (2.3), I determined to use the agile methodology throughout the development of the device. I chose this methodology as it's the most adaptive to change as the device may or may not use certain algorithms, such as normalization or vector quantization since they are situational based. It was out of Agile and Lean as they both provide flexibility, and I ultimately chose Agile as its better suited for complex and fast-changing projects while Lean is better suited for improving processes and reducing waste in a stable and predictable environment.

I will start with the filtering technique, but I may switch to the feature selection technique instead if I fail to implement filtering. The algorithms used for feature selection and filtering may change if they fail and so backup algorithms will be used. Agile is suitable for projects of any duration, but it focuses more on the short-term such as the project which is only that is set to be completed from April to May. It encourages continuous testing and feedback which can make the device better before the deadline with sprints. The weakness of the Agile methodology doesn't apply to the project, as it's only suitable for small to medium-sized projects such as mine.

3 Method and Implementation

3.1 Design

The following section of the report discusses the system and user requirements of the device, including its features and usage with step-by-step processes. The concept solution and development process are described using the Agile methodology, and an Architecture Overview is provided to illustrate the appearance of the device and the interaction between its components. The system design is outlined, including the decision-making process for completing each step. All aspects of the device, such as components, libraries, software, and algorithms, are defined.

3.1.1 Theory

The theory section provides information on the AI/ML/NLP algorithms I will use, and how they will benefit the device. Being feature extraction, normalization, vectorization, and noise filtering. However, all these algorithms may not be used depending on the device's end performance without them, as they may make it perform worse.

3.1.1.1 Vectorization

Vectorization is a signal processing technique used in voice-controlled devices that transforms audio signals into numerical representations that can be analysed and interpreted. It divides the audio signals into short segments and represents each segment as a vector of numbers. It then uses machine learning algorithms to identify patterns and features in voice recognition and command interpretation. Vectorization reduces the complexity of audio data, makes it easier to process, and even leads to better voice recognition. It's powerful for improving the accuracy and performance of devices with voice control, resulting in a better user experience.

3.1.1.2 Filtering

Filtering is a technique in voice-controlled devices that removes unwanted noise from the audio signal to improve the accuracy of voice recognition. It uses spectral subtraction and Wiener filtering to remove noise to allow the device to better interpret voice commands. Spectral subtraction subtracts noise from a signal, and Wiener filtering estimates the signal and noise components. Both techniques improve voice recognition accuracy and improve the user experience. Effective noise filtering allows users to control their devices with voice commands without being misinformed by background noise or white noise.

3.1.1.3 Feature Extraction

Feature extraction is a signal processing technique used in voice-control devices to identify important features from an audio signal for voice recognition. Algorithms such as MFCC are used to analyse the power spectrum of the audio signal and extract only the important features, which are then compared to a database of pre-recorded voice commands to identify the most likely match. By improving the accuracy and reliability of voice recognition, feature extraction can improve the responsiveness and usability of voice control devices.

3.1.1.4 Normalization

Normalization is a signal processing technique used in voice-control devices to adjust and scale the data within a specific range, typically from 0 to 1. This ensures that microphone input levels are consistent for different users and environments by reducing variation in voice commands. It involves dividing each input value by the maximum value in the input range, this makes it easier to compare and process data. It can be combined with noise filtering and compression to further improve the accuracy and reliability of voice recognition. Normalization ensures the accurate recognition of voice commands from different users, regardless of tone or volume. This improves the user experience, making the voice control devices more responsive and easier to use.

3.2 Specification / Requirements

3.2.1 User Requirements

Outlines the general requirements an average user will expect and need on the device, as well as the reasoning behind the requirement with its priority, and whether it's functional or non-functional.

No.	User Requirement	Reasoning	Priority	Functionality
1.	Effective voice control feature with 17 voice commands	Users should be able to use their voice to control the TV with 17 commands. They can change the channel, volume, pause/resume, turn the TV OFF/ON, and numbers 0 – 9.	**High**	Functional
2.	Traditional button controls	Should be able to be used as a regular TV remote. As I want to give users many options to use the remote, targeted so multiple people can use it.	**High**	Functional
3.	App Graphical User Interface (GUI)	GUI is required on the mobile application so users can control the device using their phones. The GUI must have good UI and UX designs choices to satisfy users.	**High**	Functional
4.	Universal	Users to be able to use the device on every TV model as it will be universal.	**High**	Functional
5.	User feedback	So users know if their action has been successful or not. Will provide a better user experience.	**High**	Functional
6.	Reset IR codes	Current stored IR codes are to be cleared, so users can enter new ones for a different remote they want to use.	**High**	Functional
6.	Wireless connectivity	Users to able to use the device wirelessly like normal TV remotes, as it should be portable.	**Medium**	Functional
7.	Easy connectivity	The app should easily and quickly connect with the device through Bluetooth, for better user experience.	**High**	Non-functional
8.	Secure connection with Bluetooth connectivity	To satisfy users, it should have a secure and consistent Bluetooth connection between it and the device,	**High**	Non-functional
9.	Quick response time	To satisfy users, it should have a quick and consistent response time between the device and TV, and between the device and mobile phone.	**High**	Non-functional

10.	Compatible with Android and iOS devices	To be compatible for users with both Android and iOS devices which they can download from the Play/App store.	**Medium**	Non-functional

Table 2 – User Requirements Table

A use case diagram was created to demonstrate the different ways users can interact and use the device. This was created with the User Requirements (3.2.1) in mind.

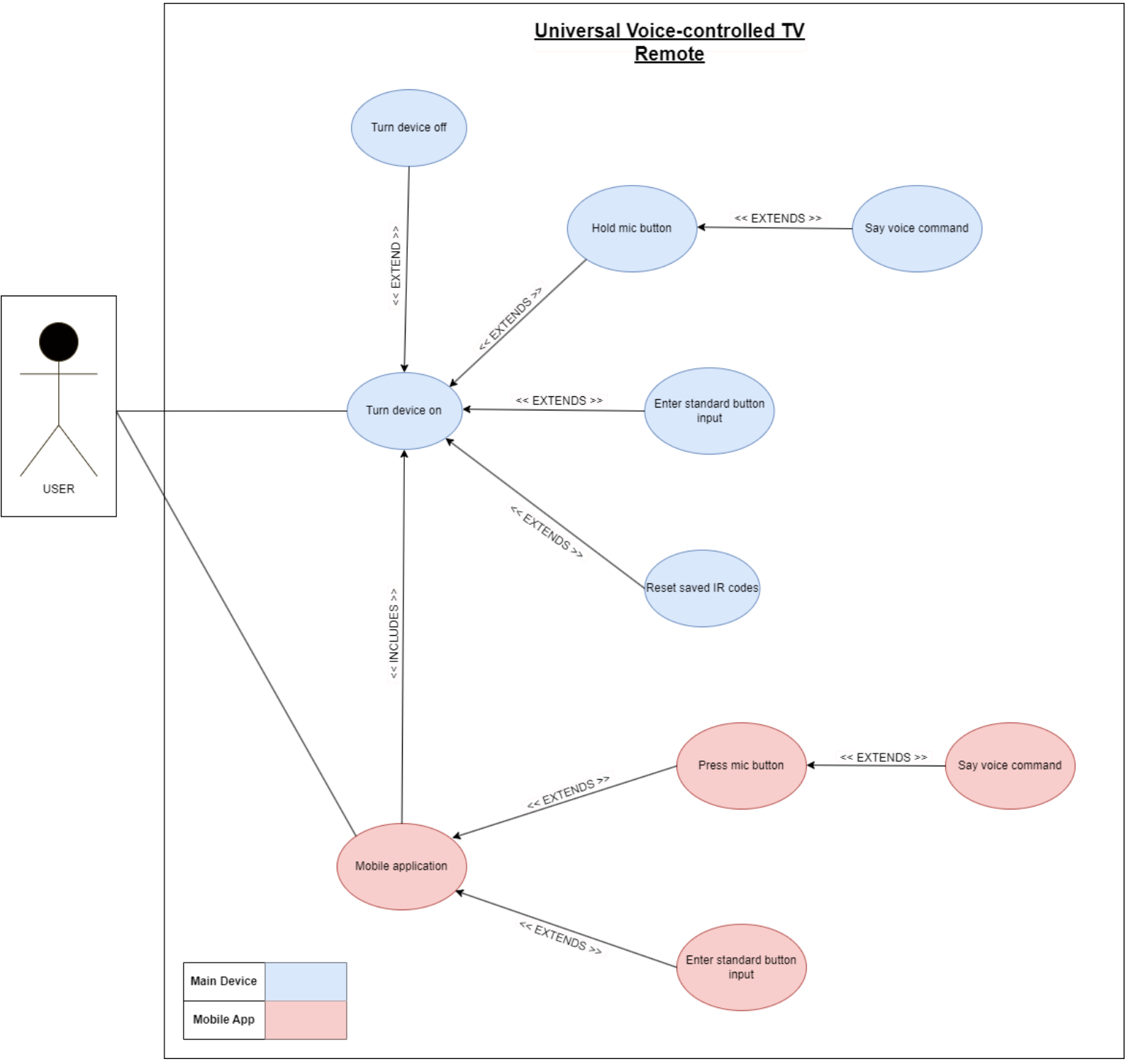


Figure 1 – Use Case Diagram

3.2.2 Design Specification

Outlines the design specification of the system requirement of the device. Many of the system requirements are similar to user requirements as I will provide their needs. I state each system requirement, reasoning with priority, and whether it is functional or non-functional.

No.	System Requirement	Reasoning	Priority	Functionality
1.	Mobile GUI	The app will have a GUI that will be designed similarly to the layout of a TV remote, so the device can be controlled with it.	**High**	Functional
2.	Voice control feature	The device can be controlled with the user's voice via many voice commands. It will have 17 voice commands, channel next/previous, volume up/down, pause/resume, power, and numbers 0 – 9.	**High**	Functional
3	Microphone	So the user's commands can be captured and examined.	**High**	Functional
4.	Capture IR codes	Should be able to capture IR codes by a TV remote so they can be stored and later outputted once a voice command is said.	**High**	Functional
5.	Both traditional buttons	The device will have many buttons for traditional use. A total of 17 buttons as they are 17 voice commands.	**High**	Functional
6.	OFF/ON button for the microphone	The microphone should not always be recording, so it should have an OFF/ON button.	**High**	Functional
7.	Reset button	Reset the button to clear currently stored IR codes. This will make the device universal as then new IR codes can be entered.	**High**	Functional
8.	Noise removal feature	Algorithm for the removal of unwanted noise in signals, such as background noise, white noise, music, etc. This is to improve the accuracy of the voice recognition feature.	**Medium**	Non-Functional
9.	Bluetooth technology	The device will have Bluetooth technology so mobile devices can be wirelessly connected to it for the app.	**High**	Non-Functional

10.	Wireless usability	The device will be able to be used wirelessly with an external power source that can be recharged.	**Medium**	Non-Functional
11.	Android and iOS mobile app compatibility	The app will be compatible with both Android and iOS mobile devices.	**High**	Non-Functional
12.	Data pre-processing and data cleaning	Devices may have data pre-processing and data cleaning algorithms to improve performance and speed.	**Medium**	Non-Functional

Table 3 - System Requirements Table

HTA diagrams were created to demonstrate the features of the device with each of the steps required for its execution. This was created with the User Requirements (3.2.1) in mind.

Process of using the TV remote with traditional buttons

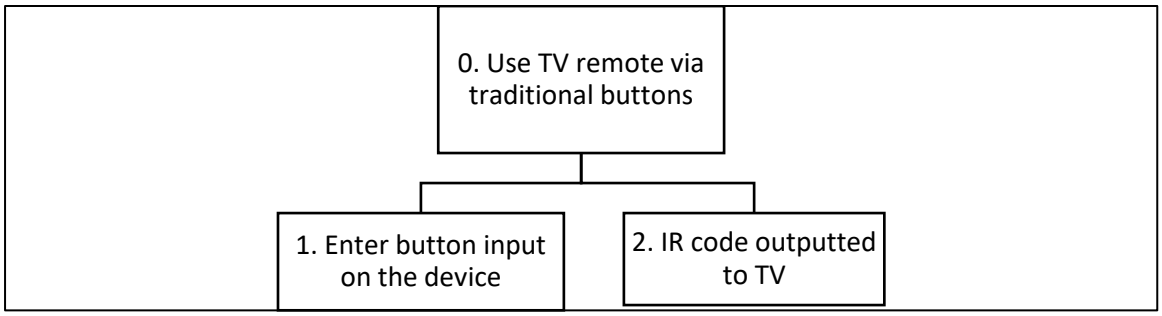


Figure 2 – TV remote with traditional buttons HTA

Process of using the TV remote by voice control with voice commands.

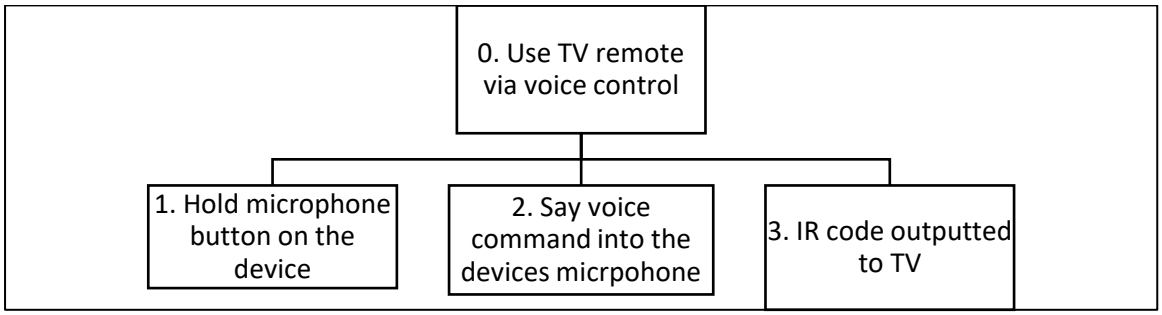


Figure 3 – TV remote with voice control HTA

Process of using a mobile app connected to the device and controlling it by traditional buttons.

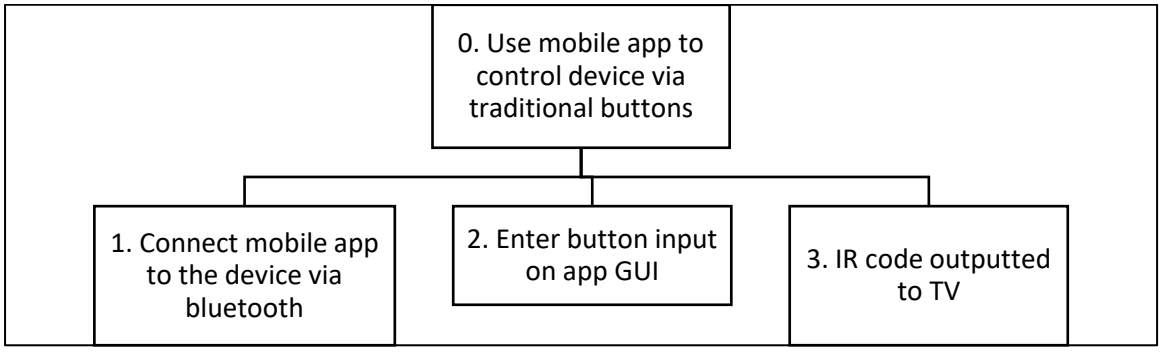


Figure 4 – mobile app with traditional buttons HTA

Process of using a mobile app connected to the device and controlling it by voice control with voice commands.

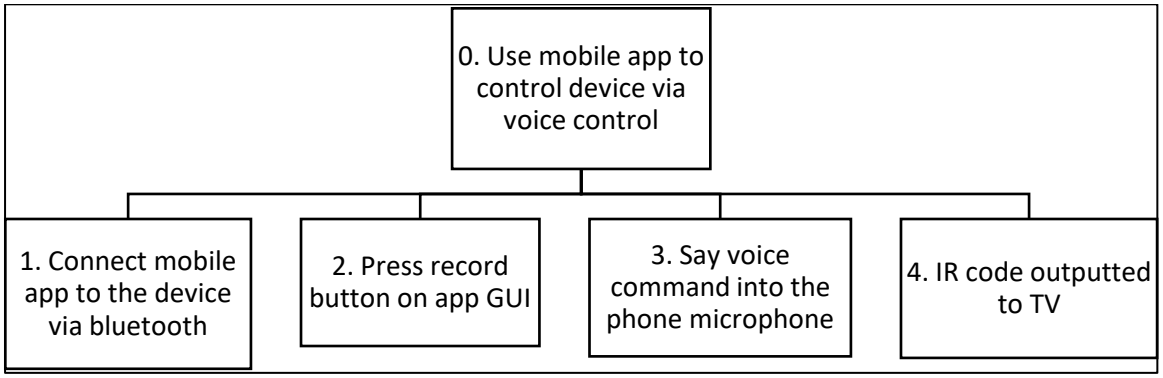


Figure 5- the mobile app with voice control HTA

3.3 Concept Solution and Architecture Overview

A block diagram was created to demonstrate the device's architecture and how important components interact with each other. This was created with the User Requirements (3.2.2) in mind.

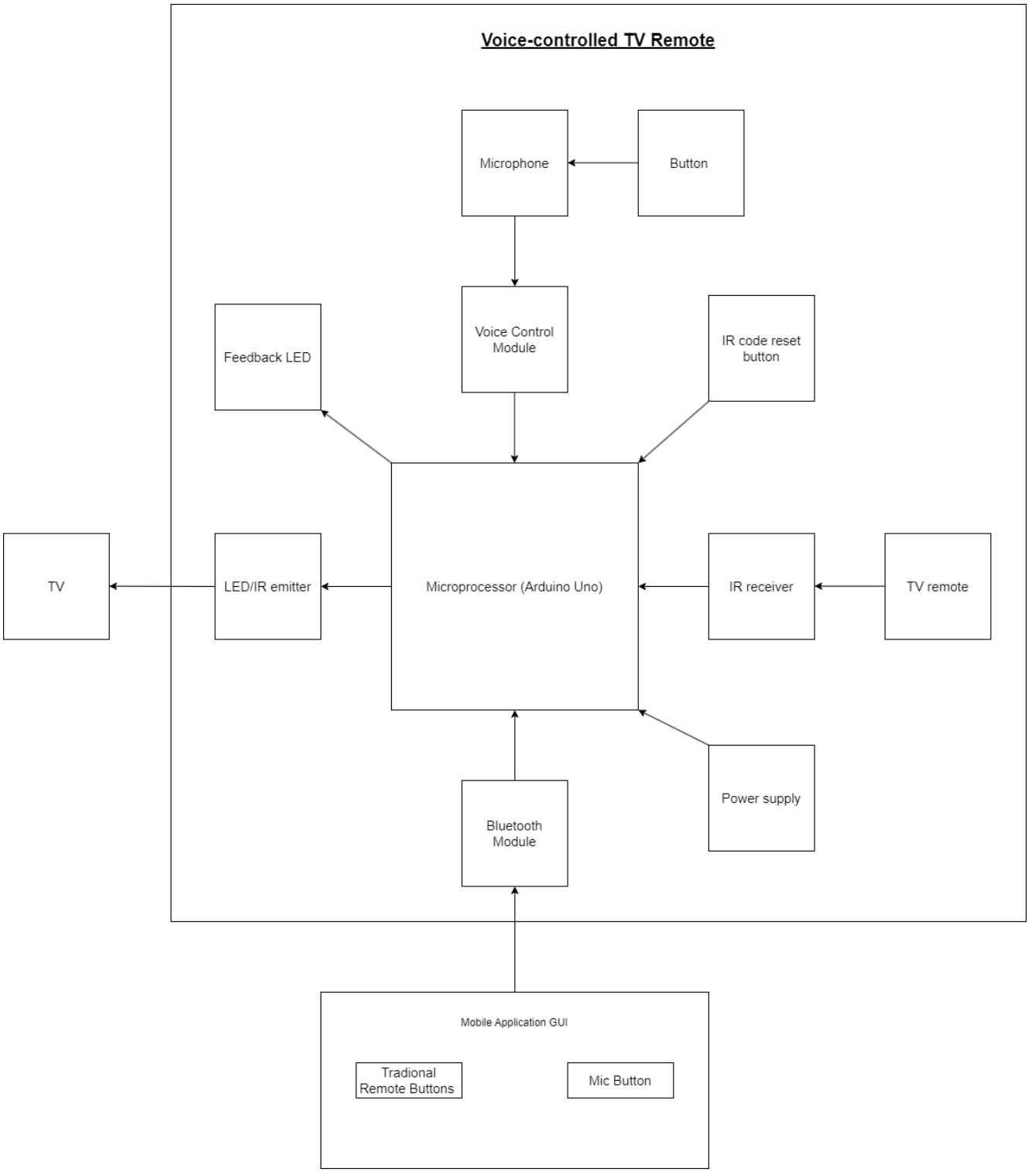


Figure 6 – Box Diagram Showing Architecture Overview

A block diagram was created to demonstrate the concept solution and development process of the device using the Agile methodology with the objectives, involving requirements design development testing, deployment, and review phases.

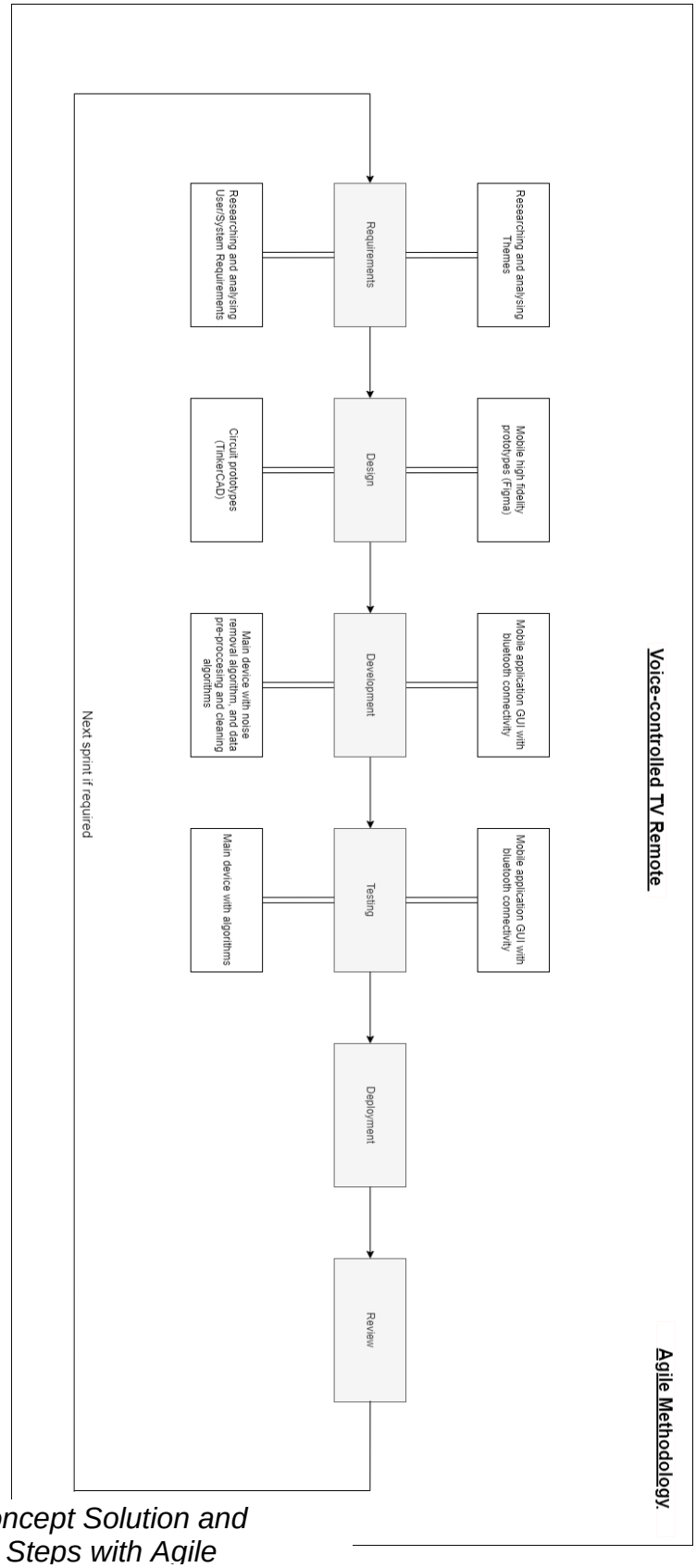


Figure 7 – Concept Solution and Development Steps with Agile

3.4 System Design

The following table outlines the many system requirements of the device, involving algorithms, libraries, software, and circuit components. I state each system requirement and its reasoning.

No.	Requirement	Reasoning
	ALGORITHMS	
1.	Filtering (*Weiner*)	The first method to remove unwanted noise from signals
2.	Feature Extraction (*MFCC*)	The second method is to remove unwanted noise from signals. (BACK UP)
3.	Vector Quantization (K-means)	Used to compress signals and represent them more efficiently, to improve the speed
4.	Normalization (Min-Max)	Used to adjust the amplitude, power, or energy of a signal, to improve accuracy
	LIBRARIES	
5.	IRremote	Displays inputted IR code of buttons pressed on an IR device
6.	NLTK	Offers signal processing algorithms for normalization and vector quantization
7.	Scipy.signal	Offers filtering and feature extraction algorithms
8.	EEPROM	Used so I can save the IR codes in the Arduino ROM, so variables can still be accessed even if the program is reset.
	SOFTWARES	
10.	TinkerCAD	Used to create circuit prototype designs
11.	Figma	Used to create mobile app GUI low/ mid/high fidelity prototypes
12.	MIT App Inventor	Used to create the mobile application and link it with the device
	COMPONENTS	
13.	Arduino Uno	Main microprocessor
14.	Push buttons	So the microphone can be powered only when the button is held down, and a button that resets the IR codes.
15.	LED	To provide the user with visual feedback for successful actions, like when the app is connected.
16.	Bluetooth module	To give Arduino Bluetooth capabilities
17.	Voice control module	Device to capture voice commands
18	Microphone	Used with the Voice control module
19.	External USB power bank	So the device can be powered and used wirelessly
20.	Breadboard	Base of device
21.	TV remote	For traditional remote buttons
22.	IR LED	To output IR codes to the TV
23.	IR receiver	IR code receiver for the traditional TV remote. And so IR codes can be read and saved into the ROM.
24.	Wires	To connect components. Used with the breadboard.

25.	Resistors	Used for components like LED to regulate electrical current
26.	TV	Will receive the IR codes from the device and perform the output

Table 4 – System Parts Table

Figure 2 from User Requirements (3.2.1) demonstrates how users would interact with the device, and figure 3, 4, 5, and 6 demonstrates the many steps of each feature within the device. Figure 7 from Concept Solution and Architecture Overview (3.3), demonstrates the device's architecture and how important components interact with each other. Table 3 and 4 from Specification / Requirements (3.2), contains all the functional and non-functional requirements for the device for user and system requirements.

Created a simple TinkerCAD script that emulates how the microphone button will work, as the microphone will only turn on while the button is held down. This is to minimize security concerns and to save power. However, TinkerCAD did not have a microphone component so a LED was used instead, but it will work the same as the LED will only get power when the button is held down. It works by checking the button's state in an if statement, if it's high so when the button is pressed, it will power the LED, else it won't power the LED.

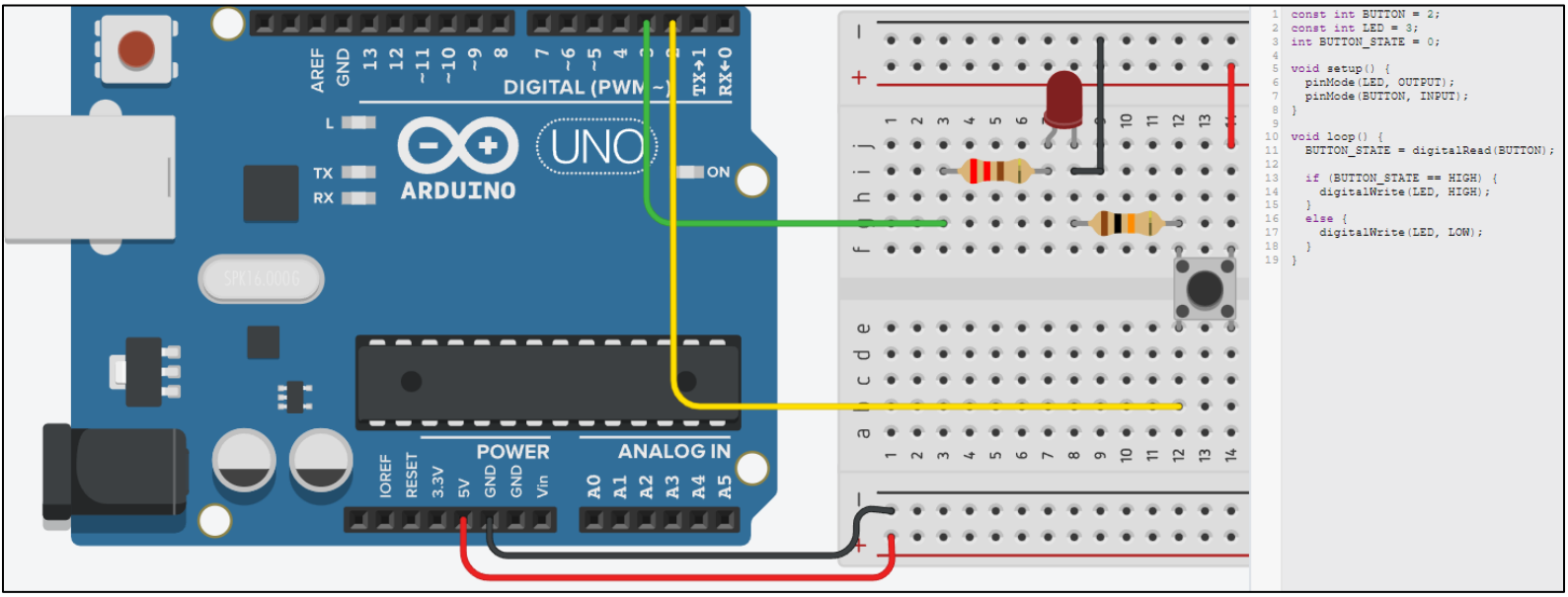


Figure 8 – TinkerCAD design of OFF/ON simulation for microphone using LED

Created a simple TinkerCAD script that emulates how hex codes will be captured with the IR receiver so they can be printed and compared using the IRremote library. This is needed so IR codes can be trained, and so traditional button controls can work. It works by contest looping to check for any incoming IR signals, and if received its stored in the results variable. A switch statement is used to check if the received value matches the hex code, If it does then the LED is turned on for 150 milliseconds. As with the real device, the LED with turn on and output the corresponding IR signal of the input.

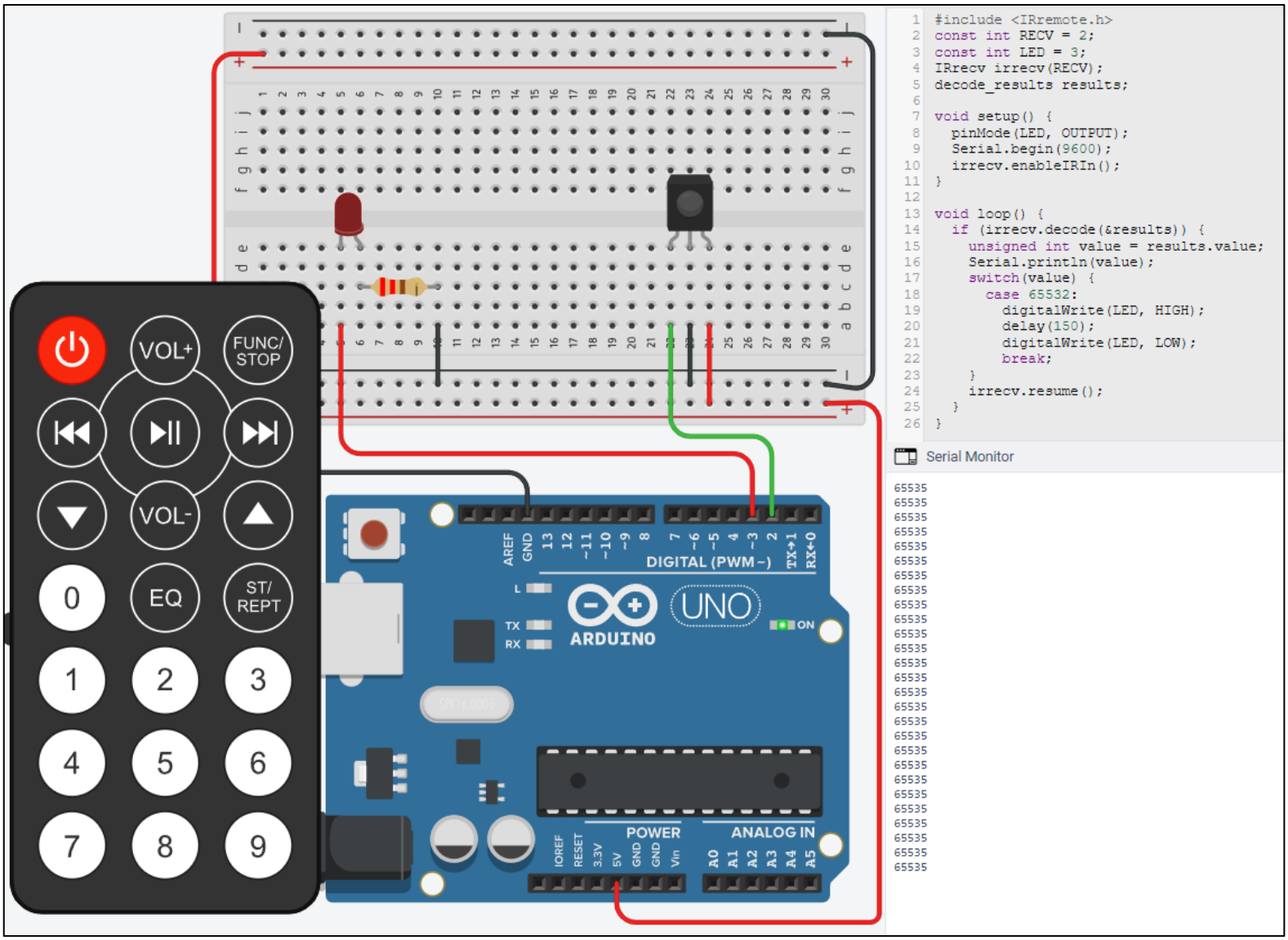


Figure 9 – TinkerCAD design using IRemote library to print out the remotes IR codes

A flow chart was created to demonstrate the flow of a process for the device and the decisions needed to complete each process. One flow chart was created for the main device and another for connectivity with the mobile application.

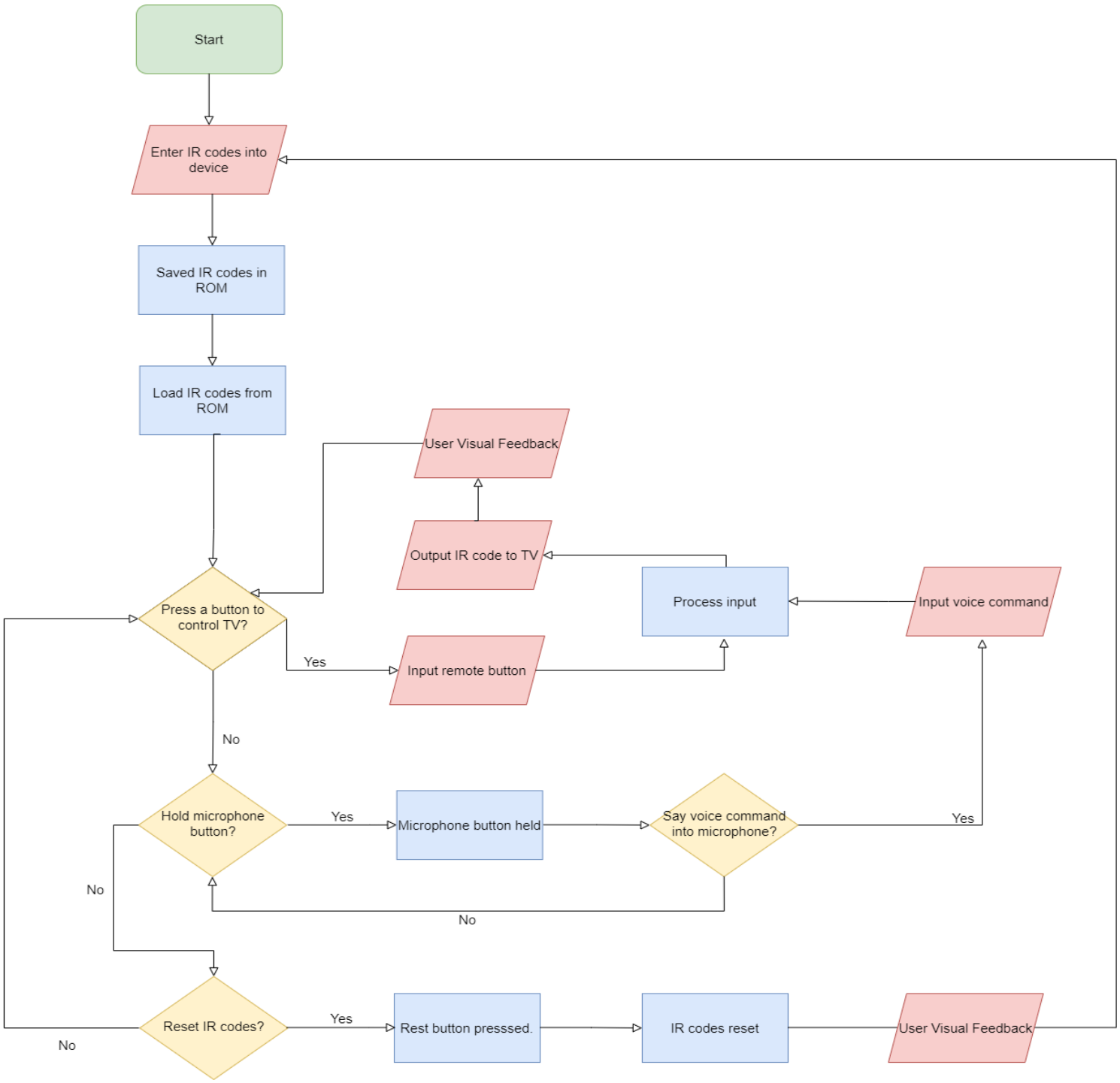


Figure 10 – Flowchart of Main Device

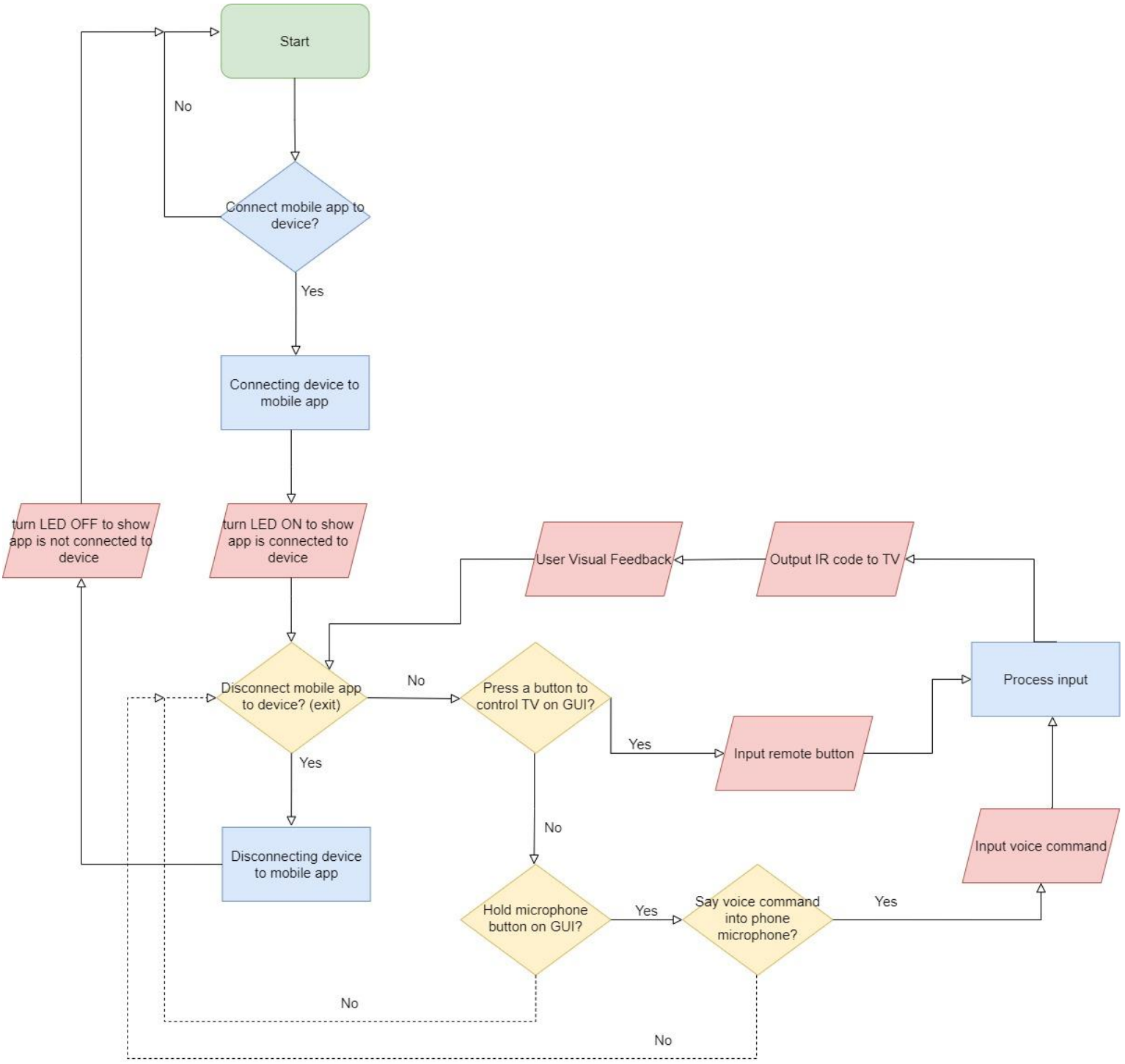


Figure 11 – Flowchart of Mobile Application Connectivity with Main Device

A low-fidelity design was created for the mobile app GUI using Figma. It consists of a mic and exit buttons, as well as other traditional TV remote buttons.

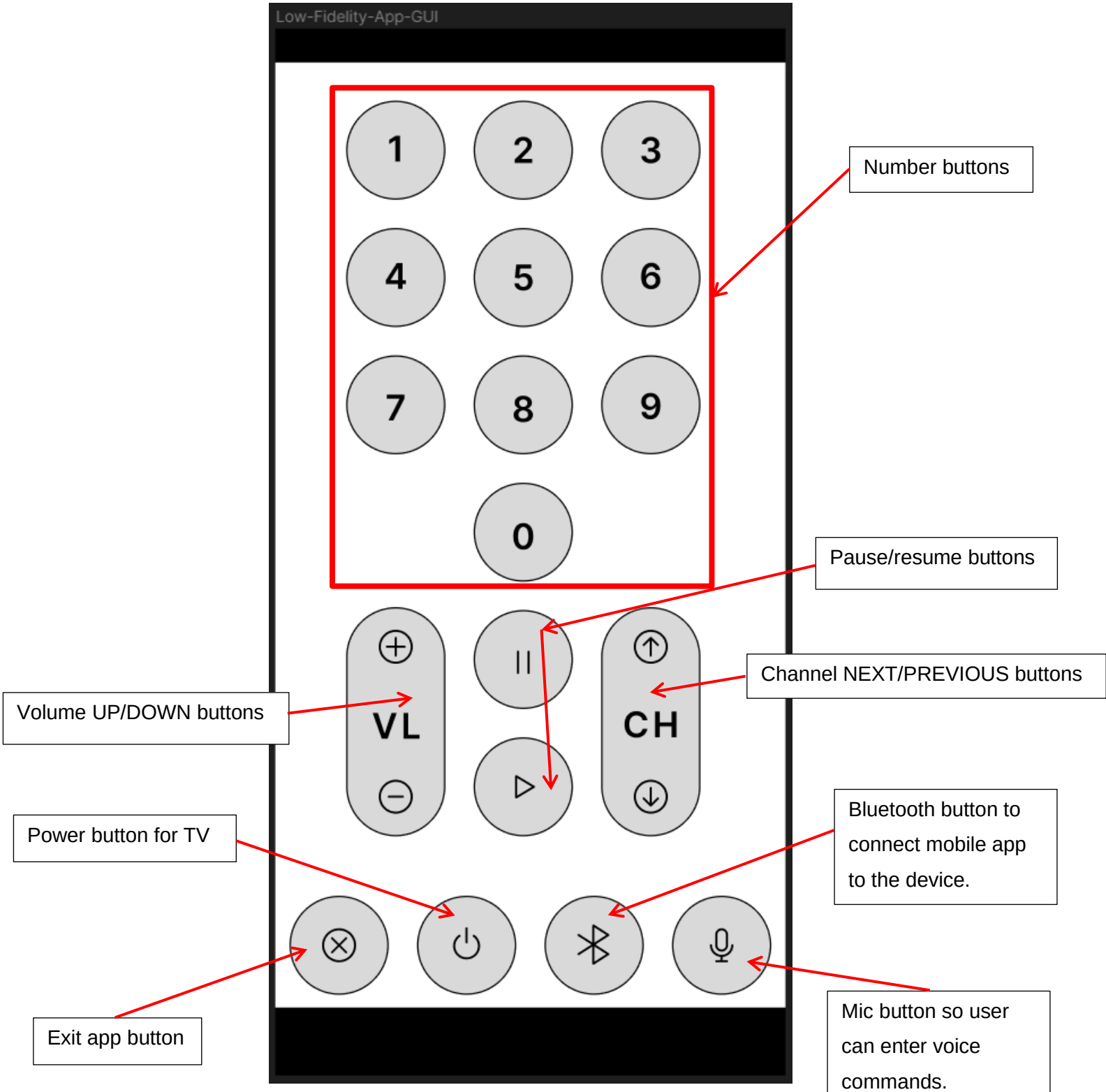


Figure 12 – Low Fidelity Design of Mobile Application GUI

Then a high-fidelity design was created with added color that hope to increase user experience. Decided on dark colours theme as it makes the bottom nav bar stand out.

Figure 13 – High Fidelity Design of Mobile Application GUI

3.5 Implementation

This section of the report discusses the implementation of the system, including the various components, techniques, and libraries utilized in the development process. It provides an overview of the technical aspects of the system, along with insights into the design decisions and the thought and step-by-step process throughout the development of the device.

3.5.1 Traditional Button Controls

Began with the development of traditional button controls, which involved a regular TV remote, breadboard, IR LED, and an IR receiver. The regular TV remote will offer the traditional button controls as it essentially will be integrated with the main device. So, the IR receiver will take in its IR signals from the remote and saves it as a HEX code, to be then converted and outputted as the same IR signal received.

Set the IR receiver to pin 7 and the IR receiver to pin 13 in conjunction with a 470-ohm resistor to limit the current flowing through it and prevent it from being damaged. The IR LED was set to an OUTPUT pinMode and the IR receiver to an INPUT pinMode, so signals can be sent and received to and from the components. For debugging and testing purposes a regular LED was used to visually see the IR signals as they are invisible with the IR LED.

```
const byte IR_EMITTER = 7;
const byte IR_LED = 13;
```

Figure 15 – Setting Pins

```
pinMode(IR_LED, OUTPUT);
pinMode(IR_EMIITTER, INPUT);
```

Figure 14 – Setting pinMode

The method used to convert the received IR signals to HEX codes and back to IR signals, were tools offered by the IRremote library that helps in building custom TV remote control applications. First set the *IRsend* to the IR LED pin so that allows IR signals can be outputted, and the *IRrevc* to the IR receiver pin that allows IR signals can be checked if they were inputted to be then decoded to HEX codes. Wanted to convert the IR signal to a HEX code so for debugging and testing if unique IR signals were being captured, and also to print them in the serial monitor.

```
#include <IRremote.h>
```

Figure 16 – Declaring IRremote Library

```
IRsend irsend(IR_LED);
IRrecv IR(IR_EMITTER);
```

Figure 17 – Setting IR output and input to components

The code first checks if an IR signal has been detected by the IR receiver. If so, the IR signal is decoded as a set HEX code and saved as a string called *inputCode*. And then the stringed HEX code was parsed and saved as an unsigned long called hexCode using the *strtoul* function, as the HEX code needs to be a number format to be outputted with the IR LED.

It's saved as an unsigned long as IR codes are usually represented as 32-bit or 64-bit hexadecimal numbers. The maximum 32-bit IR code that can fit in a 32-bit unsigned integer on an Arduino is 4,294,967,295. In comparison, the maximum value for a 32-bit signed integer on the Arduino is only 2,147,483,647, which may not be suitable for some larger IR codes. Could not retrieve the HEX code as an unsigned long, it had to be saved as a string first.

```
if (IR.decode()) {
  String inputCode = String(IR.decodedIRData.decodedRawData, HEX);
  unsigned long hexCode = strtoul(inputCode.c_str(), NULL, 16);

  printIRcode(hexCode);
  Serial.println(hexCode, HEX);

  IR.resume();
}
```

Figure 18 – Print IR Signals Code

Then the *hexCode* variable is given as an argument for the *printIRcode* method, which converts and prints out the HEX code as an IR signal to the previously specified IR LED in *IRsend*. It uses a *NEC* protocol to encode the HEX code into a 32-bit signal as it provides a sufficient amount of unique codes with 4,294,967,296 possible combinations.

```
void printIRcode(int HEX_CODE) {
  irsend.sendNEC(HEX_CODE, 32);
}
```

Figure 17 – printIRcode Function

The *hexCode* variable is also printed into the serial monitor as I wanted to visualise each unique HEX code. Lastly, the *IR.resume* function is called to reset the IR receiver so it can wait to receive the next IR signal.

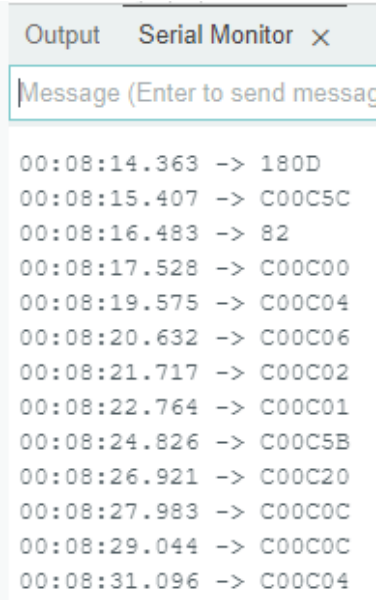


Figure 19 – Printed IR codes

3.5.2 Training Device with IR codes

For the device to be universal and work on all TV models, it needs to be first trained with the same IR codes that are used by the TV model. It is trained by first entering the IR codes into the device's IR receiver using a remote that is already connected to the TV, so the corresponding IR codes can be saved and reused.

These IR codes need to be hard saved, so they are not lost when the device is reset, or power is lost. To do this the *EEPROM* library was used which allows variables to be hard saved into the Arduino ROM, and from where they can be later accessed.

```
#include <EEPROM.h>
```

Figure 20 – Delarcing EEPROM Library

```
int numCodesSaved = 0;
unsigned long irCodes[16];
```

Figure 21 – Training IR Code Variables

The variables that will be saved in the ROM is an unsigned long array called *irCodes* that will hold 17 HEX codes, as the device will have 17 commands. And an int called *numCodesSaved* that will make sure all 17 HEX codes are saved.

To then save these variables into the ROM, the saveEEPROM function was created that opens the ROM with *begin* and stores the given variables with *put* with a unique memory address being their size so these variables can be later accessed, then closes the ROM with the *end*.

```
void saveEEPROM() {
  // 2 variables are save being, irCodes and numCodesSaved.
  EEPROM.begin();
  EEPROM.put(sizeof(bool), numCodesSaved);
  EEPROM.put(sizeof(bool) + sizeof(int), irCodes);
  EEPROM.end();
}
```

Figure 22 – saveEEPROM Function

The code to train the device was added with the functionality of using traditional remote controls, as they both use the IR receiver to retrieve the IR code to convert them into HEX codes. It first checks if *numCodesSaved* is under 17 and if hexCode is not 0, if true then it saves the hexCode in the irCodes array in the index of *numCodesSaved* which is initially 0 but goes up by 1 after a HEX code is saved inside the array, so the next HEX code can be saved at the next index within the array, this continues until *numCodesSaved* equals 17 which is the maximum number of commands the device has.

numCodesSaves get checked if it's 17, meaning that the irCodes array is full of HEX codes. If true, then it calls the *saveEEPROM* function so save the irCodes array and *numCodesSaves into the ROM.*

The hexCode is checked not to be 0 as sometimes it saves as 0, which is not wanted. Also, the device needs to be trained in a certain order so the correct IR code can be printed, the order being power, pause, resume, numbers 0-9, volume up/down, channel next/previous.

```
if (IR.decode()) {
  String inputCode = String(IR.decodedIRData.decodedRawData, HEX);
  unsigned long hexCode = strtoul(inputCode.c_str(), NULL, 16);

  if (numCodesSaved < 17 && hexCode != 0) {
    irCodes[numCodesSaved] = hexCode;
    numCodesSaved++;
    Serial.print("\nCode captured: ");
    Serial.print(numCodesSaved);
    Serial.print(" - ");
    Serial.print(hexCode);

    if (numCodesSaved == 17) {
        codesSaved = true;
        saveEEPROM();
    }
  } else {
    if (hexCode != 0) {
      printIRcode(hexCode);
      Serial.println(hexCode, HEX);
    }
  }
  IR.resume();
}
```

Figure 23 – Training IR Signals Code

Now that the HEX codes have been saved in the ROM, they need to be accessed from it. The *loadEEPORM* function was created for this, which retrieves the previously saved variables using *get*, being *numCodesSaved* and *irCodes*.

```
void loadEEPROM() {
  // 3 variables are loaded being, irCodes and numCodesSaved.
  EEPROM.begin();
  EEPROM.get(sizeof(bool), numCodesSaved);
  EEPROM.get(sizeof(bool) + sizeof(int), irCodes);
}
```

Figure 24 – loadEEPROM Function

```
void setup() {
  EEPROM.begin();
  loadEEPROM();
```

Figure 25 – Calling loadEEPROM

loadEEPROM was called in the setup function and it was the first entity declared. So, now the variables with intact values can be accessed and used, the HEX codes can be converted into IR codes whenever need which will be later done.

Shows the process of 17 HEX codes being recognised and saved via the serial monitor.

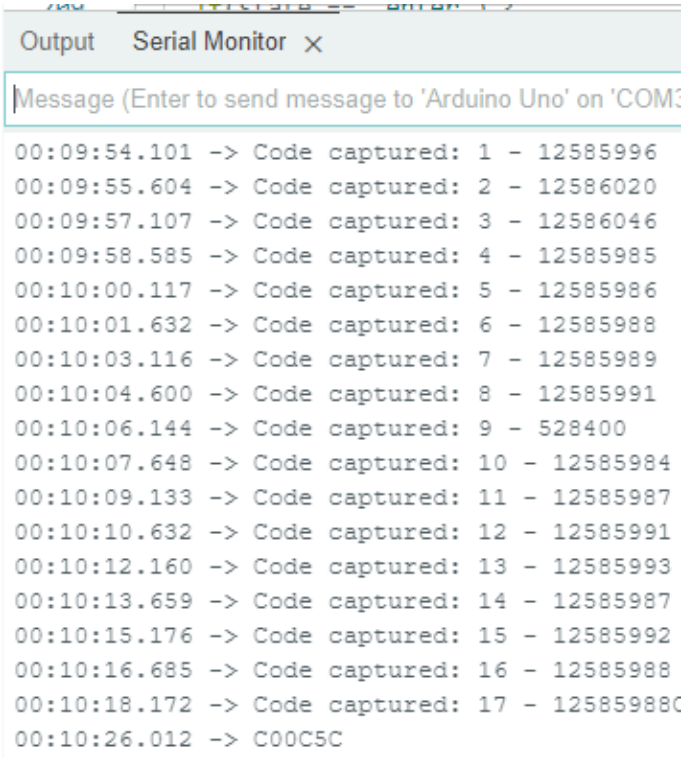


Figure 26 – Training IR codes Process

3.5.3 Reset IR Codes

The resting of IR codes is essential so the device can be compatible with other TVs, as it would not make sense for the device to only work on the TV it was first trained with. For this, the EEPROM library was used again to access the Arduino ROM so the hard-saved variables can be deleted from it.

The user resets the IR codes by pressing a button on the device, so this button is set to pin 10 with a pinMode of *INPUT_PULLUP* as it will send the device a signal when the button is pressed. It's given *INPUT_PULLUP* as in Arduino it simplifies the circuit by using the internal pull-up resistor of the board to pull the button input pin to a logical HIGH state when the button is not pressed, eliminating the need for an external resistor. Pressing the button connects the input pin to the ground, creating a logical LOW state.

```
const byte BUTTON_CODES_RESET = 10;
```

Figure 27 – Set Reset Pin

```
pinMode(BUTTON_CODES_RESET, INPUT_PULLUP);
```

Figure 28 – Set Reset pinMode

For this, the *resetEEPROM* function was created that loops through the ROM's memory locations and sets each of their values to 0.

```
void resetEEPROM() {
  // loops the ROM values and sets them all to 0.
  for (int i = 0; i < EEPROM.length(); i++) {
    EEPROM.write(i, 0);
  }
}
```

Figure 29 – resetEEPROM Function

Another function called checkResetButton was created that checks if the reset button was pressed, if true then it calls the resetEEPROM function and prints that the IR codes have been reset in the serial monitor.

```
void checkResetButton() {
  if(digitalRead(BUTTON_CODES_RESET) == LOW) {
      Serial.print("\nIR codes Reset");
      resetEEPROM();
      resetFunc();
  }
}
```

Figure 30 – Calling resetEEPROM

Now the *checkResetButton* function gets called to check if the reset button is pressed in the *waitForButton* function, as the IR codes can only be reset when the microphone button is not pressed.

```
while (digitalRead(BUTTON_PIN_MIC) == HIGH) {
  appLogic();
  checkResetButton();

  if (IR.decode()) {
    String inputCode = String(IR.decodedIRData.decodedRawData, HEX);
    unsigned long hexCode = strtoul(inputCode.c_str(), NULL, 16);

    if (numCodesSaved < 17 && hexCode != 0) {
      irCodes[numCodesSaved] = hexCode;
```

Figure 31 – Calling CheckerResetButton Function

Shows the IR codes have been reset in the serial monitor.

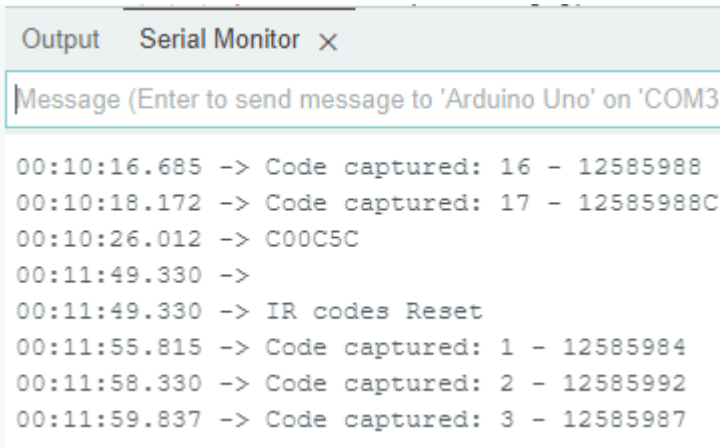


Figure 32 – Reseting Saved IR Codes

3.5.4 Voice Control Feature

The device will be equipped with a voice control module, enabling users to operate it through 17 pre-set voice commands that first need to be created and trained. The voice control module can be trained using the VoiceRecognitionV3 library, which provides a programming interface for this. Trained commands will be hard saves, do they will remain intact if the device is reset, or power is lost. Once trained the module can register new incoming voice commands and compare them to trained valid commands, if a said command exists it performs the designated action of printing the IR code.

```
#include "VoiceRecognitionV3.h"
```

Figure 33 – Delclare VR3 Library

A *VR* instance called *myVR* was initialized *with pins* connected to the voice control module, this prepares the voice control module so it can be utilized later then the instance is called.

The uint8_t records[7] array stores voice commands from the user to compare them with trained voice commands for acting, which will be later used for comparing this to the trained voice command.

The *uint8_t buf[64]* array temporarily stores raw audio data from the user's voice input before processing it with the voice recognition algorithm. This buffer acts like a container for audio data while it is being prepared for analysis. Once the audio data is collected, it is used to identify the user's voice command and determine the appropriate action to take.

```
VR myVR(3, 4);
uint8_t records[7];
uint8_t buf[64];
```

Figure 34 – set VR Pin And buf/records

A button was placed that cuts off the power to the voice control module, this is so the microphone is only listening while the button is held down, as it shouldn't always be listening for security reasons. The button is set to pin 2 with a pin Mode of *INPUT_PULLUP,* as it will send the device a signal when the button is held down which in this case is the power to the component.

```
const byte BUTTON_PIN_MIC = 2;
```

Figure 36 – Set Button Mic Pin

```
pinMode(BUTTON_PIN_MIC, INPUT_PULLUP);
```

Figure 35 - Set Button Mic pinMode

Then a *waitForButton* function was created that waits until the microphone button is pressed, while it's not pressed users can still train the device with IR codes, reset IR codes, use traditional remote button controls, and also use the app when connected.

```
void waitForButton() {
  myVR.clear();
  Serial.begin(9600);

  while (digitalRead(BUTTON_PIN_MIC) == HIGH) {
    appLogic();
    checkResetButton();

    if (IR.decode()) {
      String inputCode = String(IR.decodedIRData.decodedRawData, HEX);
      unsigned long hexCode = strtoul(inputCode.c_str(), NULL, 16);

      if (numCodesSaved < 17 && hexCode != 0) {
        irCodes[numCodesSaved] = hexCode;
        numCodesSaved++;
        Serial.print("\nCode captured: ");
        Serial.print(numCodesSaved);
        Serial.print(" - ");
        Serial.print(hexCode);
        if (numCodesSaved == 17) {
          saveEEPROM();
        }
      } else {
        if (hexCode != 0) {
          printIRcode(hexCode);
          Serial.println(hexCode, HEX);
        }
      }
      IR.resume();
    }
  }
}
```

Figure 37 – waitForButton Function

In setup this if statement checks if the voice control module is powered so it's ready to receive commands, if true it continues with the rest of the code, else it loads the waitForButton function so other features that don't require the microphone can be used.

```
IR.enableIRIn();
myVR.begin(9600);
Serial.begin(115200);

if(myVR.clear() == 0){
  Serial.println("Recognizer cleared.");
  commandLoad();
} else {
  waitForButton();
  resetFunc();
}
```

Figure 38 – Check If VR Is Powered

Now these voice commands need to be initialized and trained using the programming interface the VoiceRecognitionV3 library provides. It works by entering the *Train* command in the serial monitor with an ID so the command can be later accessed, then the wanted voice command is said into the microphone twice until they successfully match and then it has saved.

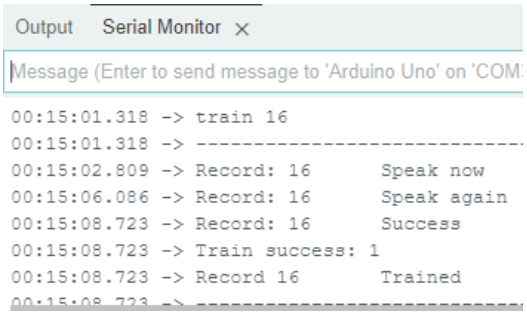


Figure 39 – Training Voice Coammands

A total of 20 voice commands will be created with 20 unique IDs, with 3 being parent commands, 14 being sub commands and 3 being normal commands. Because all 17 main commands should not get loaded at once, but instead be loaded when a parent command is specified, this is called the *dynamic loading* technique which will make the device more efficient and load faster.

Since all 20 voice commands have been created, they need to be defined with their ID and a variable so they can be later loaded when needed.

```
#define POWER (1)
#define PAUSE (3)
#define RESUME (4)
#define VOLUME (5)
#define UP (6)
#define DOWN (7)
#define CHANNEL (8)
#define NEXT (9)
#define PREVIOUS (10)
#define NUMBER (11)
#define ZERO (12)
#define ONE (13)
#define TWO (14)
#define THREE (15)
#define FOUR (16)
#define FIVE (17)
#define SIX (18)
#define SEVEN (19)
#define EIGHT (20)
#define NINE (21)
```

Figure 40 – Load Command Variables

Then the *commandLoad* function was created that loads the initial voice commands being the normal and parent commands. The sub-commands are later loaded if a parent command is received.

```
void commandLoad() {
  myVR.clear();
  myVR.load((uint8_t)POWER); // normal command
  myVR.load((uint8_t)VOLUME); // parent command
  myVR.load((uint8_t)CHANNEL); // parent command
  myVR.load((uint8_t)NUMBER); // parent command
  myVR.load((uint8_t)RESUME); // normal command
  myVR.load((uint8_t)PAUSE); // normal command
}
```

Figure 41 – Load Initial Commands

```
IR.enableIRIn();
myVR.begin(9600);
Serial.begin(115200);

if(myVR.clear() == 0){
  Serial.println("Recognizer cleared.");
  commandLoad();
} else {
  waitForButton();
  resetFunc();
}
```

Figure 42 – Call commandLoad Fucntion

The *commandLoad* function is called in *setup* when the voice control module is powered so it's ready to receive commands.

In the void loop the recognize function from the *VR* module checks if a voice command has been received. If *ret* is greater than 0, it means a voice command has been recognized and proceeds to the *switch* statement. The *switch* statement then checks the second byte of the *buf* buffer, which is the current said voice command that's to be checked to the valid voice command cases to perform the specific action being to print out the IR code corresponding to the command. If *ret* is not greater than zero, it means that no voice command has been recognized, and the code does not execute any of the cases inside the *switch* statement.

```
int ret;
ret = myVR.recognize(buf, 50);
if(ret>0){
  switch(buf[1]){
    case POWER:
      printIRcode(irCodes[0]);
      break;
    case PAUSE:
      printIRcode(irCodes[1]);
      break;
    case RESUME:
      printIRcode(irCodes[2]);
      break;
```

Whatever voice command is matched, it will call the printIRcode function with the HEX code in the irCodes by its corresponding index.

Again, this is why the IR codes were trained in a specific order so they can be accessed from the array.

The power, pause, and resume are the only commands that don’t require a parent command for it to be activated.

Figure 43 – Voice Control Power Pause Resume Code

```
case NUMBER:
  digitalWrite(LED_PIN, HIGH);
  myVR.clear();
  myVR.load((uint8_t)SEVEN); // sub command
  myVR.load((uint8_t)EIGHT); // sub command
  myVR.load((uint8_t)NINE); // sub command
  myVR.load((uint8_t)ZERO); // sub command
  myVR.load((uint8_t)ONE); // sub command
  myVR.load((uint8_t)TWO); // sub command
  myVR.load((uint8_t)THREE); // sub command
  myVR.load((uint8_t)FOUR); // sub command
  myVR.load((uint8_t)FIVE); // sub command
  myVR.load((uint8_t)SIX); // sub command
  break;
case ZERO:
  printIRcode(irCodes[3]);
  break;
case ONE:
  printIRcode(irCodes[4]);
  break;
case TWO:
  printIRcode(irCodes[5]);
  break;
case THREE:
  printIRcode(irCodes[6]);
  break;
case FOUR:
  printIRcode(irCodes[7]);
  break;
case FIVE:
  printIRcode(irCodes[8]);
  break;
case SIX:
  printIRcode(irCodes[9]);
  break;
case SEVEN:
  printIRcode(irCodes[10]);
  break;
case EIGHT:
  printIRcode(irCodes[11]);
  break;
case NINE:
  printIRcode(irCodes[12]);
  break;
```

Figure 44 – Voice Control Numbers Code

Can see that when the number command is matched, it loads all the number sub commands.

Parent commands are only active while the microphone button is held, so sub commands can be said continuously.

```
case VOLUME:
  /** turn on LED */
  digitalWrite(LED_PIN, HIGH);
  myVR.clear();
  myVR.load((uint8_t)UP); // sub command
  myVR.load((uint8_t)DOWN); // sub command
  break;
case UP:
  for (int i=0; i<5; i++) {
    printIRcode(irCodes[13]);
  }
  break;
case DOWN:
  for (int i=0; i<5; i++) {
    printIRcode(irCodes[14]);
  }
  break;
```

Figure 45 – Voice Control Volume Code

The volume parent command loads the up and down sub commands.

Each of these sub commands are in a for loop at iterated 5 times, so the TVs volume would go +/-5, as it would not be efficient if it went +/-1.

```
  case CHANNEL:
    digitalWrite(LED_PIN, HIGH);
    myVR.clear();
    myVR.load((uint8_t)NEXT); // sub command
    myVR.load((uint8_t)PREVIOUS); // sub command
    break;
  case NEXT:
    printIRcode(irCodes[15]);
    break;
  case PREVIOUS:
    printIRcode(irCodes[16]);
    break;
  default:
    Serial.println("Voice command undefined");
```

Figure 46 – Voice Control Channel Code

The channel parent command loads the next and previous sub commands.

When no voice command is matched then its stated so in the serial monitor.

3.5.5 Mobile App

After the creation of the mine device, a Mobile app was developed using MIT App Inventor which would be able to connect to the main device via Bluetooth, enabling it to be controlled from the app to control the TV. MIT app inventor was used as it offers voice recognition with Google Voice To Text that's used to add voice control to the app.

It will have the same 17 commands as the main device but no parent commands, and additional commands for Bluetooth connectivity, exiting the app and loading voice control. Data as bytes from the app is sent to the main device using the SoftwareSeiral Library over a serial connection with the Arduino.

```
#include <SoftwareSerial.h>
```

Figure 47 – Declaring SoftwareSerial Library

This is the design of the app on MIT App inventor, it's not how it will look as the software makes the design look stretched. Followed the low and high-fidelity designs when creating the GUI.

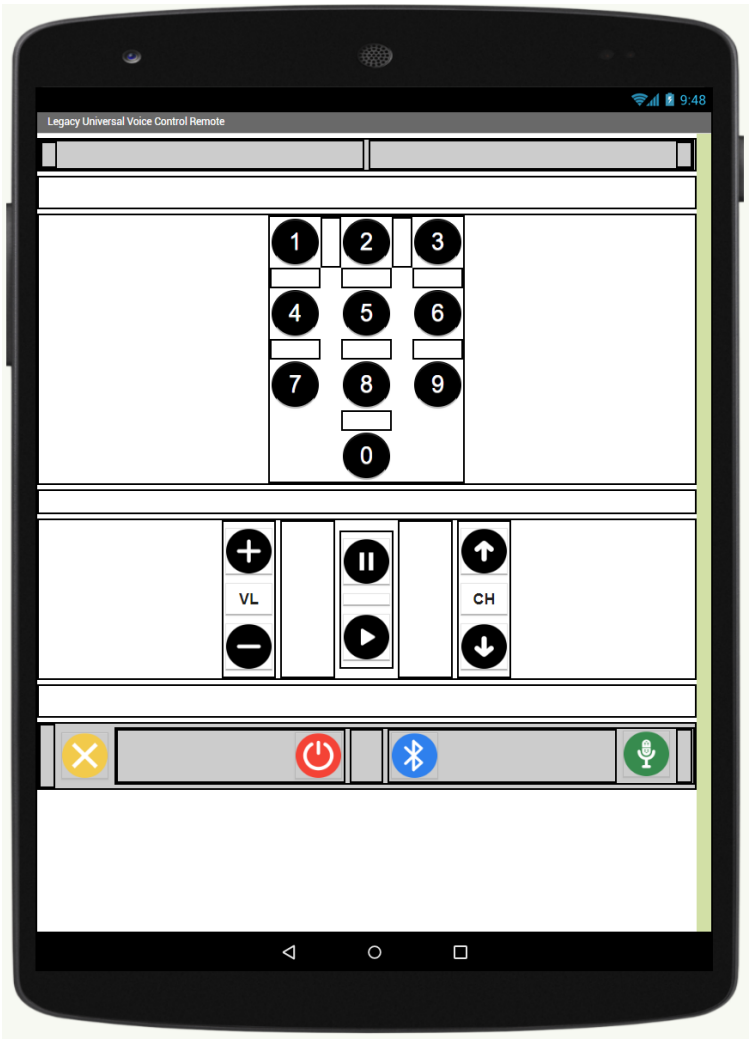


Figure 48 – App Deisgn in MIP App Inventor

This was added to give the app Bluetooth functionality so it can be connected to the main device.

ListPicker1 is the Bluetooth button and when pressed it displays all Bluetooth devices for the user to select, this is why it's under *BeforePicking* as the Bluetooth client has not been chosen. After a client has been chosen it will do *AfterPicking,* when the app attempts to connect to the selected client.

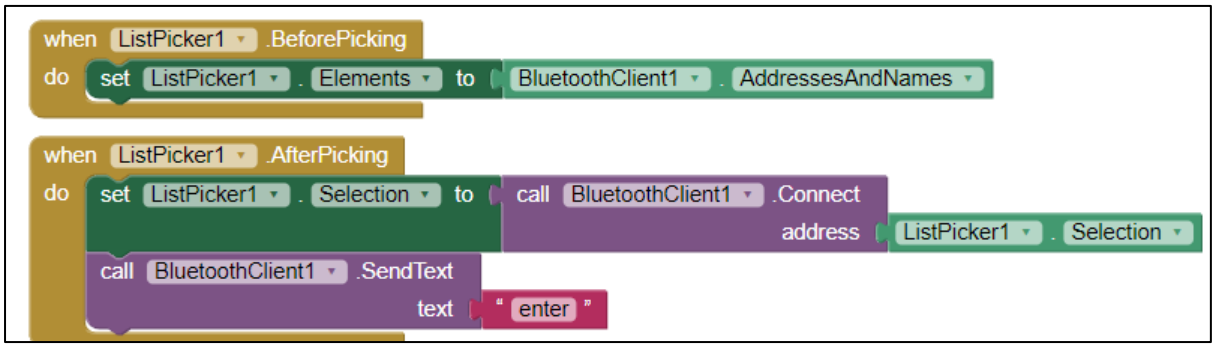


Figure 49 – App Bluetooth connectivity

This checks every second if a client is connected to the app, if true then it prints *CONNECTED*, else it prints *NOT CONNECTED*. So the user knows if the device has successfully connected to the client or not.

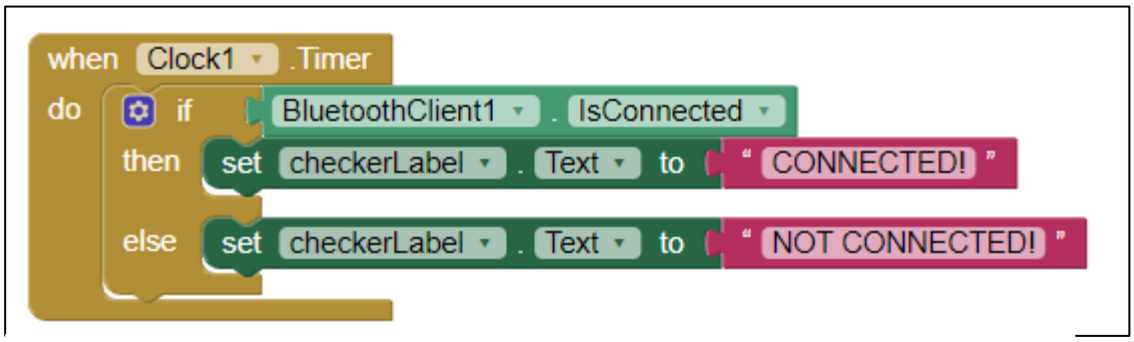


Figure 50 – App Bluetooth connect Checker

This part handles the app's voice control feature. When the microphone button is pressed it will load the voice recognizer being Google Voice To Text. Once a command is said it is sent to the client in text format. Also, made it so the GUI displays the stated voice commands with the *commandLabel* so the user can know what action was last taken.

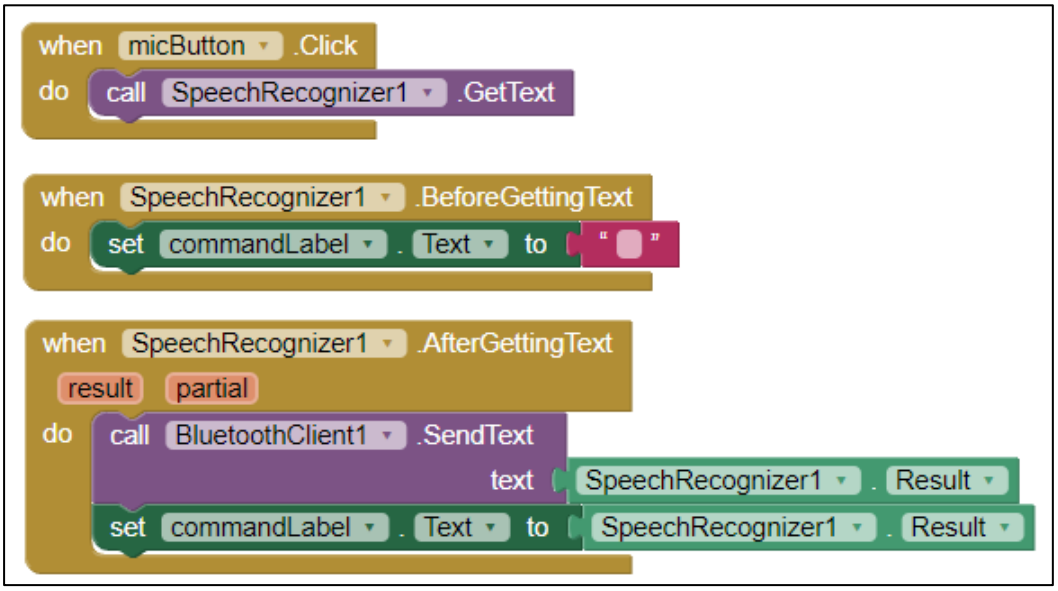


Figure 51 – App Voice Recognition Block

Each button will send a unique message to the device when pressed. So depending on what message was received, the corresponding action can be taken by the device.

Figure 52 – App Main Button Control Blocks

The Arduino does not have integrated Bluetooth technology, so a Bluetooth module was used. Pins for the Bluetooth module were set with the SoftwareSerial library so data as bytes can be received from the app, as it enables serial communication between the main device and Bluetooth module. Also, a string called *state* was created that will store the unique message sent from the app when a button is pressed or the stated voice command, it will store the exact words the user says.

```
SoftwareSerial BT(0, 1);
```

Figure 54 – Declaring BT object

```
String state;
```

Figure 53 – State Message String

The *appLogic* function handles the functionality of the data received from the app, it decides what IR codes are outputted based on what message is received.

These messages are sent as bytes, so they need to be collected and built as a string for them to be compared. So the for-loop loops and collected each sent byte which is each character of the message, to be then added on the state sting to form the message. The loop will end when the sent message is complete.

```
void appLogic() {
  myVR.clear();
  BT.begin(9600);

  while (BT.available()) {
    char c = BT.read();
    state += c;
  }
```

Figure 55 – appLogic Function

The state is the message is then checked to see if it matches a valid command, so the corresponding IR code can be outputted. It works similarly to the voice control functionality of the main device, as whatever command is matched, it will call the printIRcode function with the HEX code in the irCodes by its corresponding index.

```
if (state.length() > 0) {
  //Serial.println(state);
  if(state == "enter") {
    digitalWrite(LED_APP_CONNECT, HIGH);
  } else if(state == "exit") {
    digitalWrite(LED_APP_CONNECT, LOW);
  } else if(state == "power") {
    printIRcode(irCodes[0]);
  } else if(state == "pause") {
    printIRcode(irCodes[1]);
  } else if(state == "resume") {
    printIRcode(irCodes[2]);
  } else if(state == "number zero" || state == "number 0") {
    printIRcode(irCodes[3]);
  } else if(state == "number one" || state == "number 1") {
    printIRcode(irCodes[4]);
  } else if(state == "number two" || state == "number 2") {
    printIRcode(irCodes[5]);
  } else if(state == "number three" || state == "number 3") {
    printIRcode(irCodes[6]);
  } else if(state == "number four" || state == "number 4") {
    printIRcode(irCodes[7]);
  } else if(state == "number five" || state == "number 5") {
    printIRcode(irCodes[8]);
  } else if(state == "number six" || state == "number 6") {
    printIRcode(irCodes[9]);
  } else if(state == "number seven" || state == "number 7") {
    printIRcode(irCodes[10]);
  } else if(state == "number eight" || state == "number 8") {
    printIRcode(irCodes[11]);
  } else if(state == "number nine" || state == "number 9") {
    printIRcode(irCodes[12]);
  } else if(state == "volume up") {
      for (int i=0; i<5; i++) {
        printIRcode(irCodes[13]);
      }
  } else if(state == "volume down") {
      for (int i=0; i<5; i++) {
        printIRcode(irCodes[14]);
      }
  } else if(state == "channel next") {
    printIRcode(irCodes[15]);
  } else if(state == "channel previous") {
    printIRcode(irCodes[16]);
  }
  state ="";
}
```

Figure 56 – appLogic Main Body Code

At the end the state is then set to empty to restart the cycle, so a new received message can be later built from bytes.

```
void waitForButton() {
  myVR.clear();
  Serial.begin(9600);

  while (digitalRead(BUTTON_PIN_MIC) == HIGH) {
    appLogic();
    checkResetButton();
```

Figure 57 – Calling appLogic Function

The *appLogic* function was called in the waitForButton function, as it works without need of the microphone being on. It will check if data from the app has been received.

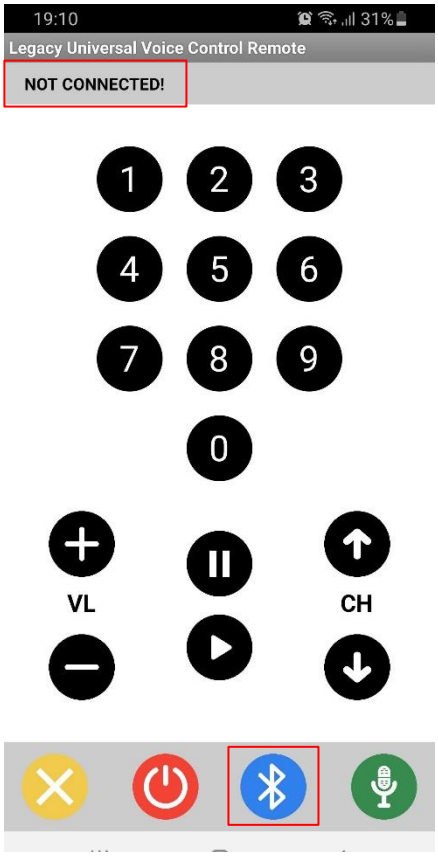


Figure 58 – App Not Connected

The app is called *Legacy Universal Voice Control TV remote*, and this is how it looks like on the phone. Can see it says NOT CONNECTED at the top, as it hasn't been connected to the device yet. To do so, the Bluetooth button is pressed at the bottom.

Figure 59 – App Bluetooth Connecting

When the Bluetooth button is pressed, it will display the user will all available Bluetooth technologies in their area. The device is called *HC-05,* as it's what the Bluetooth module is called.

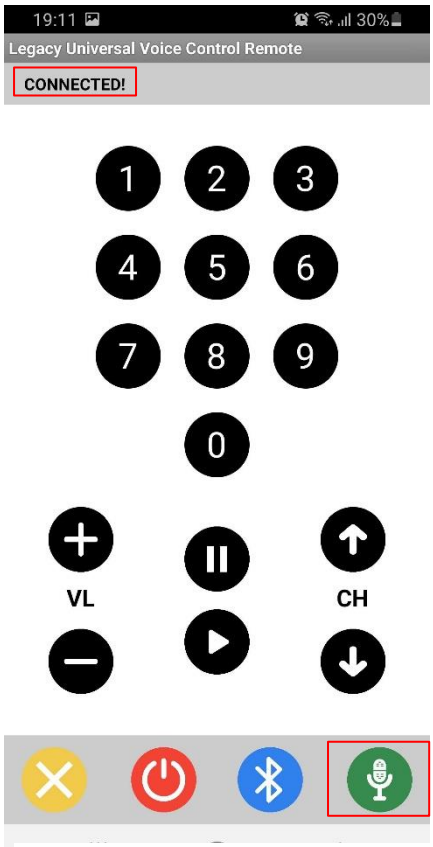


Figure 60 – App Connected

Once connected, it will now say CONNECTED at the top. To disconnect the app from the device, the user can press the yellow exit button at the bottom or close the app.

Traditional button controls will work as expected. To use the voice control feature, the green mic button is pressed at the bottom.

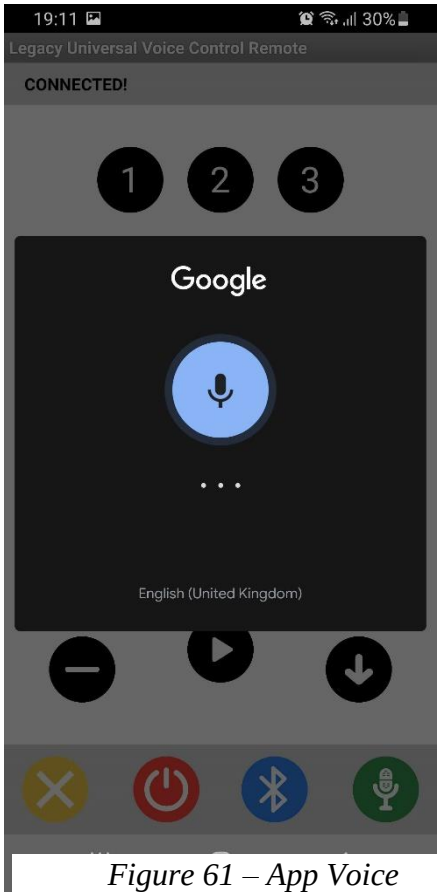


Figure 61 – App Voice Control 1

When the Bluetooth button is pressed, it will load Google Text To Speech.

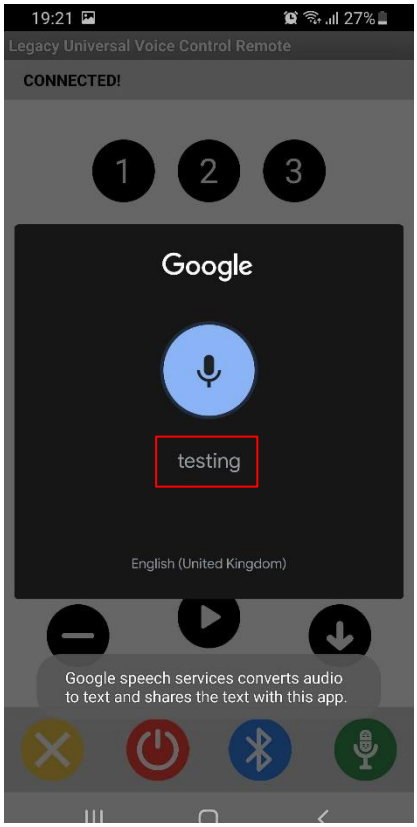


Figure 62- App Voice Control 2

From there voice commands can be entered.

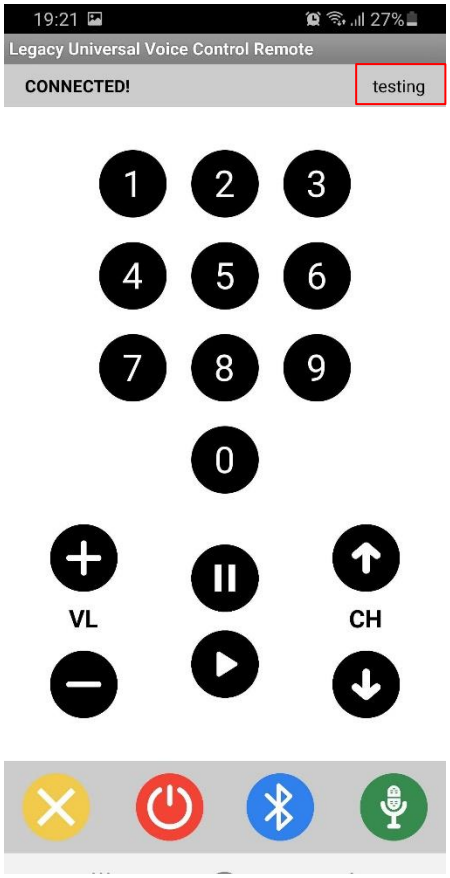


Figure 63 – App Display Input

Once entered it will send the message to the

3.5.6 User Visual Feedback

Visual user feedback was developed using multiple LEDs. To show that if the app was connected to the device, a parent command is active, a valid command was entered, the device is powered, and the resetting of IR codes. So, 4 LEDs were used but the resetting of IR codes used the same LED for valid command. Set the pins for every 4 LEDs in conjunction with a 470-ohm resistor to limit the current flowing through it and prevent it from being damaged. Each LED pin was given a corresponding output *pinMode in the setup so signals can be sent to it.* Visual feedback was also developed for debugging and testing purposes to see if each feature of the device worked as intended.

```
const byte LED_PARENT_COMMAND = 5;
const byte LED_PIN_POWER = 6;
const byte LED_VALID_COMMAND = 8;
const byte LED_APP_CONNECT = 9;
```

Figure 65 – User Feedback Pins

```
pinMode(LED_PARENT_COMMAND, OUTPUT);
pinMode(LED_PIN_POWER, OUTPUT);
pinMode(LED_VALID_COMMAND, OUTPUT);
pinMode(LED_APP_CONNECT, OUTPUT);
```

Figure 64 - User Feedback Pinmodes

For the visual feedback of valid commands and IR code reset, a void function was created called *userFeedback* was created that blinks the given LED once between a 500-millisecond delay. This function is called when a valid voice command is received from the main device or mobile app, including the reset button but not the mic button as the voice control module has a LED for that. Also, when training the device with IR codes the function will be called, to show the IR codes are valid and have been saved.

```
void userFeedback() {
  digitalWrite(LED_VALID_COMMAND, HIGH);
  delay(500);
  digitalWrite(LED_VALID_COMMAND, LOW);
}
```

Figure 66 - UserFeedBack Function

For valid voice control inputs, if a stated command is not matched in the system the validCommand boolean gets set to False which was initially set to True. It then checks if the boolean is True in an if statement, if so it calls the *userFeedback* function.

```
    default:
      Serial.println("Voice command undefined");
      validCommand = false;
      break;
  }
  /** voice recognized */
  printVR(buf);

  if(validCommand) {
    userFeedback();
  }
}
}
```

Figure 67 - Calling UserFeedBack Function

For valid traditional control input, the *userFeedback* function only gets called when training the device with IR codes and not when traditional button controls are used, as the user feedback will be displayed on the TV anyway.

```
if (IR.decode()) {
  String inputCode = String(IR.decodedIRData.decodedRawData, HEX);
  unsigned long hexCode = strtoul(inputCode.c_str(), NULL, 16);

  if (numCodesSaved < 17 && hexCode != 0) {
    irCodes[numCodesSaved] = hexCode;
    numCodesSaved++;
    Serial.print("\nCode captured: ");
    Serial.print(numCodesSaved);
    Serial.print(" - ");
    Serial.print(hexCode);
    userFeedback();
    if (numCodesSaved == 17) {
        codesSaved = true;
        saveEEPROM();
    }
  } else {
    if (hexCode != 0) {
      printIRcode(hexCode);
      Serial.println(hexCode, HEX);
    }
  }
  IR.resume();
```

Figure 68 - Calling UserFeedBack Function

For valid app inputs, it is similar to valid commands, being if a stated command from the app is not matched in the system the *validCommand* boolean gets set to False which was initially set to True. It then checks if the boolean is True in an if statement, if so it calls the *userFeedback* function.

```
    } else if(state == "channel previous") {
      printIRcode(irCodes[16]);
    } else {
      validCommand = false;
    }
    state ="";

    if(validCommand) {
      userFeedback();
    }
  }
}
```

Figure 69 - Calling UserFeedBack Function

For resetting IR commands, the *userFeedback function is called inside the checkResetButton function* when the reset button is pressed.

```
void checkResetButton() {
  if(digitalRead(BUTTON_CODES_RESET) == LOW) {
      Serial.print("\nIR codes Reset");
      resetEEPROM();
      userFeedback();
      resetFunc();
  }
}
```

Figure 70 – Calling UserFeedBack Function

For visual feedback of when the device is on, the LED was simply set to HIGH in the setup before the main part of the code is run.

```
void setup() {
  EEPROM.begin();
  loadEEPROM();

  pinMode(BUTTON_PIN_MIC, INPUT_PULLUP);
  pinMode(BUTTON_CODES_RESET, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  pinMode(LED_PIN_POWER, OUTPUT);
  pinMode(LED_VALID_COMMAND, OUTPUT);
  pinMode(LED_APP_CONNECT, OUTPUT);
  pinMode(IR_LED, OUTPUT);

  digitalWrite(LED_PIN_POWER, HIGH);
```

Figure 71 – Device Powered User Feedback

For visual feedback of when a parent command is active, if a command is deemed to be a parent command then the LED is set to HIGH until it is inactive. This is the case for every parent command, being volume, channel, and number.

```
case VOLUME:
  /** turn on LED */
  digitalWrite(LED_PIN, HIGH);
  myVR.clear();
  myVR.load((uint8_t)UP);
  myVR.load((uint8_t)DOWN);
  break;
case UP:
```

Figure 72 – Volume User Feedback

```
case CHANNEL:
  digitalWrite(LED_PIN, HIGH);
  myVR.clear();
  myVR.load((uint8_t)NEXT);
  myVR.load((uint8_t)PREVIOUS);
```

Figure 73 – Channel User Feedback

```
case NUMBER:
  digitalWrite(LED_PIN, HIGH);
  myVR.clear();
  myVR.load((uint8_t)SEVEN);
```

Figure 74 – Number User Feedback

So, the LED will set back to LOW when the parent command is no longer active, and this happens when the microphone button is no longer held down, as the program is reset, and the initial commands are loaded.

```
void loop() {
  static bool lastButtonState = LOW;
  bool currentButtonState = digitalRead(BUTTON_PIN_MIC);

  if (currentButtonState == HIGH && lastButtonState == LOW) {
    digitalWrite(LED_PIN, LOW);
    resetFunc(); // reset the program
  }
  lastButtonState = currentButtonState; // update last button state
```

Figure 75 – Parent Command User Feedback

For visual feedback on the connectivity of the app, the app checks if it is connected to the device via Bluetooth, if so it sends a message to the device being *enter*. Same when the exit button is pressed, it sends the device a message being *exit*.

Sends the client being the device the enter message if it's connected to it.

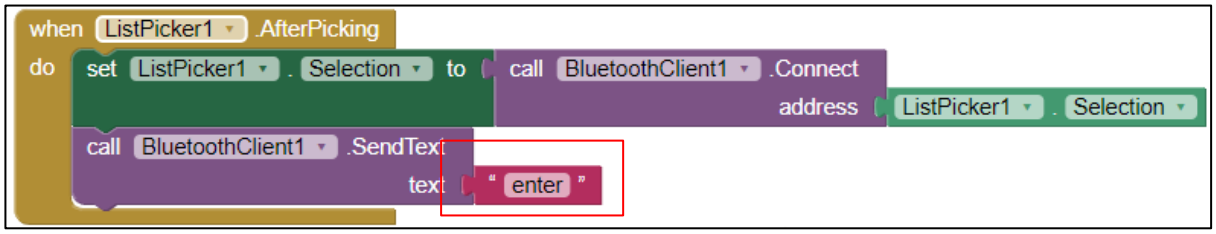


Figure 76 - Enter App User Feedback Block

Sends the client being the device the exit message when the exit button is pressed.

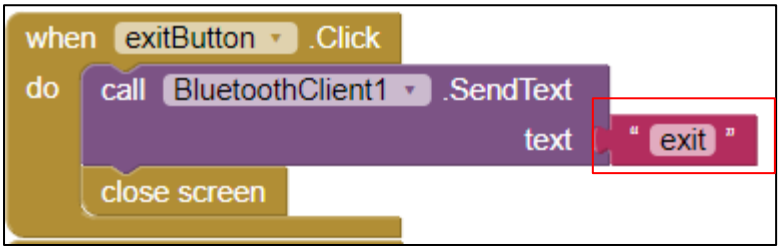


Figure 77 – Exit App User Feedback Block

```
if (state.length() > 0) {
  //Serial.println(state);
  if(state == "enter") {
    digitalWrite(LED_APP_CONNECT, HIGH);
  } else if(state == "exit") {
    digitalWrite(LED_APP_CONNECT, LOW);
  } else if(state == "power") {
```

Figure 78 – Enter/Exit App User Feedback Code

The *appLogic* function additionally checks if the app has sent the *exit* or *enter* message. If *enter* was received it will turn the LED to HIGH, and if *exit* was received it will turn the LED to LOW.

3.5.7 Improving Voice Recognition with NLP

Unfortunately, due to time constraints, the implementation of AI/ML/NLP algorithms into the project was not achieved. It was planned to implement normalization, vectorization, filtering, and feature extraction to improve the voice recognition of the device. However, it is intended that this feature will be implemented in future iterations of the project.

4 Evaluation

This section provides the evaluation and testing of the overall project. Many methodologies will be utilized to gather qualitative and quantitative research data for evaluation. This data will help provide better insight into the project so potential improvements can be made in terms of performance and user experience. Also, previously stated theories and hypotheses will be investigated and evaluated using these methodologies (1.5).

4.1 Evaluation Methodology

For quantitative research, performance testing will be undertaken that focuses on evaluating the performance of the project. This was chosen as it will help me determine if the device is fully functional as intended without any errors or bugs. The project will be tested in terms of response time and latency for each button and voice command, its max IR signal, voice control, and Bluetooth range. Voice recognition consistency will also be tested using different microphones. Both the main device and mobile app with be thoroughly tested. Data will be collected within a table.

For qualitative research, surveys will be created and sent to participants who have used the device. It focuses on evaluating the user experience and satisfaction of the project. This was chosen as it will help me gather user feedback and determine if the project has achieved its intended goal and if the problem statement has been solved. Depending on user feedback, if it's constructive it may be utilized to improve the device's functionality and user experience. The survey will have 11 questions that aim to obtain positive and negative constructive feedback so potential improvements are clear. Also, a CSAT at the end measures the user's overall satisfaction. Surveys will be created and sent using *Google Forms*.

4.1.1 Evaluation Metrics

The table shows the various metrics that will be used for each evaluation factor when quantitative and qualitative research is undertaken.

No.	Factor	Metric	Research	Comment
1.	Response time	Microseconds (µs)	Quantitative	Both button and voice commands will be tested for both the main remote and the app. Want to test if voice or button controls have a faster response time.
2.	Latency	Microseconds (µs)	Quantitative	Both button and voice commands will be tested for both the main remote and the app. Want to text if voice or button controls with have lower latency.
3.	Max Bluetooth connectivity range	Meters	Quantitative	For mobile app only.
4.	Max IR signal range	Meters	Quantitative	For remote only.
5.	Voice control range	Meters	Quantitative	For remote and mobile apps. Will use 2 different microphones, AKG earphones, and an Elechouse microphone.
6.	Voice control accuracy	Percentage	Quantitative	For remote and mobile apps. Will use 2 different microphones, AKG earphones, and an Elechouse microphone.
7.	Error rate	Units	Quantitative	For remote and mobile apps.
8.	User satisfaction	Customer Satisfaction Score (CSAT)	Qualitative	The last question asked in the survey.

Table 5 – Evaluation Metrics Table

4.2 Results

Results of performance testing from quantitative research. Response time was calculated by starting a timer after the input that ends after the output (response time = (end time – start time)). But with latency, the timer ended before the output (latency = (end time – start time)).

Factor	Metric Result	Conclusion / Evidence
Response time (Button Controls)	**Power: 55916 µs**	02:05:43.169 -> RT: 55916 µs
	Pause: 55912 µs	02:08:23.850 -> RT: 55912 µs
	Resume: 59292 µs	02:10:06.988 -> RT: 59292 µs
	Volume Up: 54720 µs	02:10:55.259 -> RT: 54720 µs
	Volume Down: 54668 µs	02:11:35.467 -> RT: 54668 µs
	Channel Next: 54808 µs	02:12:31.713 -> RT: 54808 µs
	Channel Previous: 54696 µs	02:13:13.933 -> RT: 54696 µs
	0: 53684 µs	02:14:22.305 -> RT: 53684 µs
	1: 54828 µs	02:15:06.523 -> RT: 54828 µs
	2: 54820 µs	02:16:00.803 -> RT: 54820 µs
	3: 55928 µs	02:16:53.072 -> RT: 55928 µs
	4: 54824 µs	02:17:50.322 -> RT: 54824 µs
	5: 55924 µs	02:18:29.551 -> RT: 55924 µs
	6: 55940 µs	02:20:45.096 -> RT: 55940 µs
	7: 57076 µs	02:21:26.311 -> RT: 57076 µs
	8: 54780 µs	02:22:05.532 -> RT: 54780 µs
	9: 55948 µs	02:24:07.024 -> RT: 55948 µs
Latency (Button Controls)	**Power: 256 µs**	02:05:43.137 -> L: 256 µs
	Pause: 256 µs	02:08:23.777 -> L: 256 µs
	Resume: 256 µs	02:10:06.955 -> L: 256 µs
	Volume Up: 176 µs	02:10:55.171 -> L: 176 µs
	Volume Down: 184 µs	02:11:35.420 -> L: 184 µs
	Channel Next: 260 µs	02:12:31.679 -> L: 260 µs
	Channel Previous: 216 µs	02:13:13.899 -> L: 216 µs
	0: 260 µs	02:14:22.270 -> L: 260 µs
	1: 256 µs	02:15:06.489 -> L: 256 µs
	2: 256 µs	02:16:00.727 -> L: 256 µs

	3: 256 µs	02:16:52.981 -> L: 256 µs
	4: 256 µs	02:17:50.288 -> L: 256 µs
	5: 264 µs	02:18:29.457 -> L: 264 µs
	6: 256 µs	02:20:45.062 -> L: 256 µs
	7: 256 µs	02:21:26.276 -> L: 256 µs
	8: 256 µs	02:22:05.455 -> L: 256 µs
	9: 260 µs	02:24:06.990 -> L: 260 µs
Response time (Voice Controls)	**Power: 57896 µs**	16:41:08.232 -> RT: 57896 µs
	Pause: 57748 µs	16:41:09.324 -> RT: 57748 µs
	Resume: 57788 µs	16:41:10.562 -> RT: 57788 µs
	Volume Up: 233564 µs	16:44:32.188 -> RT: 292140 µs
	Volume Down: 291880 µs	16:47:03.613 -> RT: 291880 µs
	Channel Next: 57776 µs	16:48:02.945 -> RT: 57776 µs
	Channel Previous: 62284 µs	16:49:46.043 -> RT: 62284 µs
	0: 57860 µs	16:53:58.497 -> RT: 57860 µs
	1: 57924 µs	16:54:00.214 -> RT: 57924 µs
	2: 57856 µs	16:54:02.024 -> RT: 57856 µs
	3: 57836 µs	16:54:04.006 -> RT: 57836 µs
	4: 57864 µs	16:54:12.252 -> RT: 57864 µs
	5: 57772 µs	16:54:14.489 -> RT: 57772 µs
	6: 57820 µs	16:54:21.770 -> RT: 57820 µs
	7: 57720 µs	16:54:24.532 -> RT: 57720 µs
	8: 57772 µs	16:54:26.419 -> RT: 57772 µs
	9: 57800 µs	16:54:28.451 -> RT: 57800 µs
Latency (Voice Controls)	**Power: 12 µs**	16:41:08.185 -> L: 12 µs
	Pause: 8 µs	16:41:09.246 -> L: 8 µs
	Resume: 8 µs	16:41:10.528 -> L: 8 µs
	Volume Up: 234044 µs	16:44:32.142 -> L: 234044 µs
	Volume Down: 233828 µs	16:47:03.580 -> L: 233828 µs
	Channel Next: 4 µs	16:48:02.868 -> L: 4 µs
	Channel Previous: 8 µs	16:49:46.008 -> L: 8 µs
	0: 8 µs	16:53:58.418 -> L: 8 µs
	1: 16 µs	16:54:00.182 -> L: 16 µs
	2: 8 µs	16:54:01.978 -> L: 8 µs
	3: 8 µs	16:54:03.974 -> L: 8 µs
	4: 8 µs	16:54:12.174 -> L: 8 µs
	5: 8 µs	16:54:14.396 -> L: 8 µs

	6: 4 µs	16:54:21.723 -> L: 4 µs
	7: 4 µs	16:54:24.454 -> L: 4 µs
	8: 8 µs	16:54:26.387 -> L: 8 µs
	9: 8 µs	16:54:28.404 -> L: 8 µs
Response time (App Button Controls)	**Power: 57892 µs**	17:44:40.505 -> RT: 57892 µs
	Pause: 55736 µs	18:07:08.777 -> RT: 55736 µs
	Resume: 59140 µs	18:07:58.504 -> RT: 59140 µs
	Volume Up: 280764 µs	18:08:53.269 -> RT: 280764 µs
	Volume Down: 286252 µs	18:09:26.209 -> RT: 286252 µs
	Channel Next: 54852 µs	18:10:17.956 -> RT: 54852 µs
	Channel Previous: 55972 µs	18:28:43.769 -> RT: 55972 µs
	0: 53660 µs	18:29:29.508 -> RT: 53660 µs
	1: 55812 µs	18:30:13.218 -> RT: 55812 µs
	2: 54816 µs	18:30:52.920 -> RT: 54816 µs
	3: 55836 µs	18:31:44.683 -> RT: 55836 µs
	4: 53596 µs	18:32:08.327 -> RT: 53596 µs
	5: 54712 µs	18:33:15.129 -> RT: 54712 µs
	6: 54756 µs	18:34:12.906 -> RT: 54756 µs
	7: 55932 µs	18:34:53.595 -> RT: 55932 µs
	8: 54848 µs	18:35:25.267 -> RT: 54848 µs
	9: 55952 µs	18:36:22.061 -> RT: 55952 µs
Latency (App Button Controls)	**Power: 84 µs**	17:44:20.812 -> L: 84 µs
	Pause: 80 µs	18:07:08.731 -> L: 80 µs
	Resume: 100 µs	18:07:58.470 -> L: 100 µs
	Volume Up: 224964 µs	18:08:53.236 -> L: 224964 µs
	Volume Down: 229308 µs	18:09:26.174 -> L: 229308 µs
	Channel Next: 280 µs	18:10:17.883 -> L: 280 µs
	Channel Previous: 384 µs	18:28:43.723 -> L: 384 µs
	0: 168 µs	18:29:29.420 -> L: 168 µs
	1: 176 µs	18:30:13.172 -> L: 176 µs
	2: 180 µs	18:30:52.846 -> L: 180 µs
	3: 224 µs	18:31:44.592 -> L: 224 µs
	4: 232 µs	18:32:08.237 -> L: 232 µs
	5: 244 µs	18:33:15.083 -> L: 244 µs
	6: 252 µs	18:34:12.860 -> L: 252 µs
	7: 292 µs	18:34:53.563 -> L: 292 µs
	8: 304 µs	18:35:25.233 -> L: 304 µs

	9: 308 µs	18:36:21.973 -> L: 308 µs
Response time (App Voice Controls)	**Power: 57920 µs**	17:46:20.996 -> RT: 57920 µs
	Pause: 55612 µs	19:17:43.754 -> RT: 55612 µs
	Resume: 59112 µs	19:18:25.446 -> RT: 59112 µs
	Volume Up: 280688 µs	19:19:10.421 -> RT: 280688 µs
	Volume Down: 286248 µs	19:19:46.346 -> RT: 286248 µs
	Channel Next: 54784 µs	19:22:14.429 -> RT: 54784 µs
	Channel Previous: 51428 µs	20:07:06.704 -> RT: 51428 µs
	0: 53492 µs	20:07:53.416 -> RT: 53492 µs
	1: 55728 µs	20:08:42.160 -> RT: 55728 µs
	2: 54640 µs	20:09:15.863 -> RT: 54640 µs
	3: 55752 µs	20:10:01.117 -> RT: 55752 µs
	4: 53468 µs	20:10:58.474 -> RT: 53468 µs
	5: 54708 µs	20:11:49.219 -> RT: 54708 µs
	6: 54680 µs	20:12:41.965 -> RT: 54680 µs
	7: 55740 µs	20:13:22.653 -> RT: 55740 µs
	8: 54712 µs	20:13:56.375 -> RT: 54712 µs
	9: 55832 µs	20:14:34.035 -> RT: 55832 µs
Latency (App Voice Controls)	**Power: 84 µs**	17:46:20.950 -> L: 84 µs
	Pause: 88 µs	19:17:43.661 -> L: 88 µs
	Resume: 100 µs	19:18:25.411 -> L: 100 µs
	Volume Up: 224904 µs	19:19:10.380 -> L: 224904 µs
	Volume Down: 229372 µs	19:19:46.272 -> L: 229372 µs
	Channel Next: 260 µs	19:22:14.397 -> L: 260 µs
	Channel Previous: 332 µs	20:07:06.672 -> L: 332 µs
	0: 144 µs	20:07:53.384 -> L: 144 µs
	1: 180 µs	20:08:42.128 -> L: 180 µs
	2: 180 µs	20:09:15.817 -> L: 180 µs
	3: 192 µs	20:10:01.085 -> L: 192 µs
	4: 240 µs	20:10:58.427 -> L: 240 µs
	5: 252 µs	20:11:49.173 -> L: 252 µs
	6: 240 µs	20:12:41.919 -> L: 240 µs
	7: 252 µs	20:13:22.620 -> L: 252 µs
	8: 272 µs	20:13:56.284 -> L: 272 µs
	9: 284 µs	20:14:34.002 -> L: 284 µs

Max IR signal range	*Estimated 5 meters+ (15 feet+)*	15 feet+ as it was the max distance it could be tested. Good max range as common room sizes is around the same size.
Max Bluetooth connectivity range	*Estimated 12 meters+ (40 feet+)*	40 feet+ as it was the max distance it could be tested. Stated online that the HC-05 module can have a range of up to 100 meters (or approximately 328 feet)
Voice control range (Main Device) using Elechouse microphone	*0.5 meters (1.5 feet)*	Not a good range as you had to be very close to the microphone for it to pick up the voice.
Voice control range (Main Device) using AKG earphones	*0.15 meters (0.5 feet)*	Not a good range as you had to be very close to the microphone for it to pick up the voice.
Voice control range (Mobile App) using phone microphone	*0.15 meters (0.5 feet)*	Not a good range as you had to be very close to the microphone for it to pick up the voice.
Voice control range (Mobile App) using AKG earphones	*0.15 meters (0.5 feet)*	Not a good range as you had to be very close to the microphone for it to pick up the voice.
Voice control accuracy (Main Device) using Elechouse microphone	20 total voice commands for the main device. 20 commands worked the first time. Accuracy = (20 / 20) x 100% Accuracy = ***100%***	Voice control worked as expected here using an Elechouse microphone for the main device.
Voice control accuracy (Main Device) using AKG earphones	20 total voice commands for the main device. 20 commands worked the first time. Accuracy = (20 / 20) x 100% Accuracy = ***100%***	Voice control worked as expected here using AKG earphones for the main device.
Voice control accuracy (Mobile App) using base phone microphone	17 total voice commands for the app. 17 commands worked the first time. Accuracy = (17 / 17) x 100% Accuracy = ***100%***	Voice control worked as expected here using a phone microphone for the mobile app.
Voice control accuracy (Mobile App) using AKG earphones	17 total voice commands for the app. 17 commands worked the first time. Accuracy = (17 / 17) x 100% Accuracy = ***100%***	Voice control worked as expected here using AKG earphones for the mobile app.
Error rate (Both remote/Mobile App)	**Remote button control:** *0/20 (0%)* **Remote voice control:** *0/20 (0%)* **App button control:** *0/17 (0%)* **App voice control:** *0/17 (0%)* 72 total operations (20 + 20 + 17 + 17) 0 errors were detected. Error Rate = (0 / 74) x 100% Error Rate = ***0%***	Each command was tested, and no errors were found for both the main remote or mobile app. Everything worked as expected.

Table 6 – Performance Testing Results

For the main remote, button controls have an average response time of 55454µs and a latency of 256µs, while voice controls have an average response time of 57783µs and a latency of 6.8µs. This suggests that button controls have a faster response time than voice controls, but voice controls have lower latency. For the mobile app, button controls have an average response time of 54740µs and a latency of 2155µs, while voice controls have an average response time of 55394µs and a latency of 7µs. Again, this suggests that button controls have a faster response time than voice controls, but voice controls have lower latency.

From this, it's understood that with voice control the device will respond more quickly to user inputs, and button controls will be executed faster. As lower latency improves user experience by reducing latency between user input and system output, while faster response times improve system efficiency by allowing the system to process more incoming commands.

There's a trend where the response time and latency for the volume up and down button/voice commands are very high and slow, this is because it’s in a for loop so it's outputted 5 times in the program, as it will do volume 5+ or 5-. Another trend was that the higher the number of commands, the longer the command took to execute, as the response time and latency got worse. This is because the higher number commands are further down in the conditional statement as it goes from 0 to 9, and therefore takes longer to reach.

Overall, no errors were found for the main remote and mobile app. All their accuracy was 100% when each input was first declared, with both microphones tested. But the Elechouses microphone had more range than the AKG headphones with a difference of 1 foot, so it will be used from now on. Also, both IR signal and Bluetooth ranges are very large and sufficient as they extended the max testing range.

Survey results involving user experience and satisfaction from qualitative research. This was done using Google Forms sent via Email. This survey was done face-to-face, as they needed to use the device so I could teach them to use it before conducting the survey. Evidence of this is shown in *9.5 Appendix D.*

No.	Question	Participant 1	Participant 2	Participant 3	Participant 4	Conclusion
1.	What TV model did you use the device with? And specify if it was a smart or non-smart TV.	*Non-smart LG television.*	*sky glass smart tv*	*LG Non smart TV*	*Non-smart Tobisha*	*3 non-smart legacy TVs and 1 smart TV was used in this research.*
2.	What mobile phone did you use the app with?	*Huawei P30.*	*samsung s9*	*Blackview*	*Samsung galaxy S20*	*A wide range of mobile phones was used in this research.*
3.	Do you have any visual or physical impairments that affect your experience using TV remotes?	*No.*	*no*	*No.*	*No*	*No participants had any visual or physical impairments. This is a shame, as they are the intended audience before the regular public.*
4.	How convenient did you find it to have both traditional controls and voice control in one remote?	*I thought having both traditional controls and voice control in one remote was really convenient.*	*i loved it because it gave me more options for how I wanted to interact with the tv*	*Normal buttons felt better faster to use but voice control was still cool to have.*	*Good, as I want to use both*	*The majority of participants liked having both traditional button controls and voice control. One participant preferred traditional button controls over voice controls.*
5.	Did you encounter any issues or errors while using the device?	*Nope, I did not get any errors!*	*no*	*No errors.*	*No*	*No participants came across any initial errors which is good, as I did not come across any errors as well with performance testing.*
6.	What additional features would you like to see added to the device?	*More functionality. So more commands, more buttons, more features like customization so people can create their own commands.*	*create a case for it as it is probably the main complaint because it was hard to use without a case*	*Make it wireless so it can be portable. Also give it a cover or case.*	*Add more buttons*	*Participants would like to see if the device had more functionality, like more commands, buttons, a case, wireless connectivity so its portable, and more customizability features.*

7.	What did you like most about the device/app?	*App is responsive with a good layout, similar to a real television remote so its familiar to use. Many voice commands to use.*	*good app worked well on the phone without any issues and i liked the colour of black and white*	*It was very simple to connect it to the remote. The app is good to use.*	*App and device with all features are easy to use*	*Participants liked the fact the device is easy and simple to use, its many voice commands, and the GUI of the app.*
8.	Would you recommend this universal remote to a friend who also has a non-smart TV?	*I don't think I have any friends who has a smart television, but I would still recommend it to them because of the phone app.*	*yes the voice control will help them out a lot and save them time*	*Yeah. Because I personally enjoy using it, so I don't see why not. I would recommend it to multiple people.*	*Yes*	*All participants would recommend the device to a friend.*
9.	What did you dislike most about the device/app?	*It was not wireless so it could not be placed anywhere.*	*awkward to use because there is no case you cannot hold it without a case it can only be used on the table*	*You can't really hold it as there is no cover or case. You should be able to hold the remote. A 3D printed case would be cool.*	*Its not portable.*	*The majority of participants dislikes the fact the device has no case, and that it's not portable as it's not wireless. One suggested a 3D printed case.*
10.	Would you consider buying the device if you saw it in a shop?	*Yes*	*Yes*	*Yes*	*Yes*	*All participants would purchase the device if they found it in a shop.*
11.	Rate your overall user experience with the device	*7*	*9*	*7*	*7*	*Overall average* CSAT score of 7.5.

Table 7 – Survey Results Summary

The device having good user experience and satisfaction is very important as its critical for the success of any product, especially one that is intended for the public. Customers are more likely to recommend and purchase products that are easy to use and meet their needs and expectations. Therefore, it is important to collect user feedback and consider suggestions for improving the device to meet user needs and provide a positive user experience.

Overall, participants rated their experience and satisfaction with the device with a CSAT score of 7.5 which is a good indication that the majority of participants were satisfied with the device. However, there is room for improvement based on the feedback provided by the participants based on their likes and dislikes to future increase user experience and satisfaction. Such as putting it in a case so it's holdable, powering it wirelessly to make the main device portable, adding more buttons, adding more commands, and a feature where users can enter their custom voice commands.

Trends found were that all participants did not find any errors, they would buy the device if they found it in a shop, and they would recommend it to a friend. Also, none of them had any visual or psychical impairments which is a shame since they are the main intended target audience before the normal public.

4.3 Discussion

The results of the qualitative study demonstrate the potential benefits of integrating voice controls, traditional buttons, universal functionality, and mobile app into one device aimed towards legacy non-smart TVs. Throughout this study, participants reported increased convenience and ease of use when controlling the TV with voice commands, traditional buttons, or a mobile app. They preferred the idea to have all of these tools are their disposal and use whatever feel right in that situation, to just only having one option.

The hypotheses stated that the availability of the mobile app is considered a desirable feature for the younger audience, who prefer to use their smartphone as a remote control. Though this hypothesis wasn't directly tested, it's safe to say if its adults enjoyed the experience, then the younger audience would most likely too. Furthermore, as mobile apps can attract younger audiences who prefer to use their phones as remote controls, this supports the larger trend where homes are becoming more automated and simpler, as everything is now in one device the mobile phone. However, not all users may want to use mobile apps to control their TVs, especially older adults as they might prefer the old way. This is why giving users multiple choices is important so everyone can be pleased, thus increasing user experience and satisfaction, which is the main reason the project has 3 different ways of using it.

The qualitative results also support the hypothesis that a device with voice control, traditional buttons, and a mobile app can improve accessibility and convenience when controlling TVs and increase user experience. This is supported as participants rated their experience and satisfaction with the device with a CSAT score of 7.5 which is a good indication that the majority of participants were satisfied with the device. They also stated that they would buy the device if they found it in a shop, and they would recommend it to a friend, meaning they enjoyed the device. Furthermore, the study suggests that the development of mobile apps for traditional TVs represents an opportunity for innovation and market disruption in the TV industry, considering participants seemed satisfied with the app.

The quantitative results also found that for the main remote, button controls have an average response time of 55454µs and latency of 256µs, while voice controls have an average response time of 57783µs and a latency of 6.8µs. This suggests that button controls have a faster response time than voice controls, but voice controls have lower latency. Also that for the mobile app, button controls have an average response time of 54740µs and a latency of 2155µs, while voice controls have an average response time of 55394µs and a latency of 7µs. Again, this suggests that button controls have a faster response time than voice controls, but voice controls have lower latency. It's understood that with voice control the device will respond more quickly to user inputs, and button controls will be executed faster. As lower latency improves user experience by reducing latency between user input and system output, while faster response times improve system efficiency by allowing the system to process more incoming commands. The quantitative results also showed that the volume up and down commands had high response times and latency due to being in a loop that outputted the command five times. Similarly, higher-numbered commands took longer to execute as they were further down in the conditional statement, which may have contributed to the slower response times and latency. In the real world, these results suggest that voice control has lower latency, while button control is faster and more efficient when executing commands in

terms of response time. This information could be important for device manufacturers and designers to consider when developing and optimizing future TV remotes. Users should be able to quickly execute commands such as channel change and change volume without any delays.

5 Conclusions

In conclusion, the qualitative results of the study provide important insight into how integrating voice commands, traditional buttons, and mobile apps into a single device for legacy non-smart TVs, can bring convenience, accessibility, and functionality, while also increasing overall user satisfaction and experience. The disposal of multiple control options allows for a personalized user experience that caters to the preferences of different users. The study suggests that the development of mobile apps for legacy TVs represents an opportunity for innovation in the TV industry.

The study's quantitative results of the study provide insights into the response times and latency of button and voice controls, which can guide device manufacturers and designers to optimize future TV remotes. While button commands were found to have faster response times, voice commands had lower latency, showing the importance of considering both factors when developing a TV remote. The study highlights the impact of user-centered design and the value of incorporating multiple control options into TV remotes to improve user experience and satisfaction.

6 Recommendations for future work

For future work to further improve the project, I would implement features I previously wasn’t able to which was AI/ML/NLP to improve the device's voice recognition in terms of speed and accuracy by using algorithms such as filtering, normalization, vectorization, and feature extraction. A useful feature would be that users can create custom voice commands with their voice, this will make the device work internationally. Another feature could be to overhaul the device to be a multi-device controller so it can work with any IR device, it would have the capability to store multiple IR codes and allow the user to switch between them using voice commands or physical buttons. One recommendation from user feedback was to create a case for the device, which I would do using cheap 3D-printed recycled materials. Another recommendation was to give the device wireless power. Overall, these improvements would enhance the usability and functionality of the device, making it a more appealing option for users with diverse needs and preferences.

7 References

- wrike. (2018). What Is Agile Methodology in Project Management? [2023] from https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/
- Damonlang Lamare. (2023). What is DevOps? DevOps Methodology, Principles & Stages Explained. [2023] from https://www.edureka.co/blog/what-is-devops/
- Jon Terry (2018). WHAT IS LEAN METHODOLOGY? [2023] from https://www.planview.com/resources/guide/lean-principles-101/what-is-lean/
- Jon Terry (2022). What Is Waterfall Methodology? Here's How It Can Help Your Project Management Strategy. [2023] from https://www.forbes.com/advisor/business/what-is-waterfall-methodology/
- Monday (2023). A New Way Of Working. [2023] from https://monday.com/
- Draw.io (2023). [2023] from https://draw.io/
- Creately (2023). Use Case Diagram Tutorial (Guide with Examples). [2023] from https://creately.com/guides/use-case-diagram-tutorial/
- Erica Golightly (2022). How to Create a Flowchart in Excel (With Templates & Examples). [2023] from https://clickup.com/blog/flowchart-in-excel/
- Raluca Budiu (2017). Quantitative vs. Qualitative Usability Testing. [2023] from https://www.nngroup.com/articles/quant-vs-qual/
- Asmo (2018). Agile Methodology: An Overview [2023] from https://zenkit.com/en/blog/agile-methodology-an-overview/
- Isaac Sacolick (2022). What is agile methodology? Modern software development explained. [2023] from https://www.infoworld.com/article/3237508/what-is-agile-methodology-modern-software-development-explained.html

8 Bibliography

- Simon Monk. (2019). Programming Arduino Next Steps: Going Further with Sketches, 2nd Edition. [2022] from https://www-accessengineeringlibrary-com.bcu.idm.oclc.org/content/book/9781260143249?implicit-login=true

- Brian Evans. (2011). Beginning Arduino Programming. [2022] from *https://ebookcentral.proquest.com/lib/bcu/detail.action?docID=883784*

- What is Arduino? (2015). Arduino Documentation. [2022] from *https://search.iczhiku.com/paper/TFzDJhGhd6VMaDsI.pdf*

- Duncan Wilson (2021). How does a remote control work the TV? [2022] from https://www.ucl.ac.uk/culture-online/ask-expert/your-questions-answered/how-does-remote-control-work-tv

- Rosalyn R Porle (2022). Speech-Based Number Recognition Using KNN and SVM. [2022] from https://ieeexplore.ieee.org/abstract/document/9936761

- Khoa N. Van (2018). Text-dependent Speaker Recognition System Based on Speaking Frequency Characteristics. [2022] from https://link.springer.com/chapter/10.1007/978-3-030-03192-3_16

- Sadat Hasan Shehab (2020). Home Automation System Using Gesture Pattern & Voice Recognition For Paralyzed People. [2022] from https://ieeexplore.ieee.org/document/9393142

- Chauhan Naman (2017). Finding the IR Codes of Any IR Remote Using Arduino. [2022] from https://www.instructables.com/Finding-the-IR-Codes-of-Any-IR-Remote-Using-Arduin/

- Jianliang Meng (2012). Overview of the Speech Recognition Technology. [2022] from *https://ieeexplore.ieee.org/abstract/document/6300437*

- Rafizah Mohd Hanifa (2021). A review on speaker recognition: Technology and challenges. [2022] from https://www.sciencedirect.com/science/article/pii/S0045790621000318

- Pratik K. Kurzekar (2014). CONTINUOUS SPEECH RECOGNITION SYSTEM: A REVIEW. [2022] from *https://www.researchgate.net/publication/270899158_CONTINUOUS_SPEECH_RECOGNITION_SYSTEM_A_REVIEW*

- Shweta Singhal (2015). Automatic speech recognition for connected words using DTW/HMM for English/ Hindi languages. [2022] from https://ieeexplore.ieee.org/document/7437908

- Seokyeong Jeong (2018). Fast decoder design of connected word speech recognition for automobile navigation system. [2022] from https://ieeexplore.ieee.org/document/1598387

- Ben Shneiderman (2000). THE LIMITS of SPEECH RECOGNITION. [2022] from https://go-gale-com.bcu.idm.oclc.org/ps/i.do?p=AONE&u=uce&id=GALE|A65277890&v=2.1&it=r

- Priyanka P. Patil (2014). Marathi connected word speech recognition system. [2022] from https://ieeexplore.ieee.org/document/6906687

- V Radha (2012). A Review on Speech Recognition Challenges and Approaches. [2022] from https://d1wqtxts1xzle7.cloudfront.net/30987239/A_Review_on_Speech_Recognition_Challenges_and_Approaches-libre.pdf?1392225529=&response-content-disposition=inline%3B+filename%3DA_Review_on_Speech_Recognition_Challenge.pdf&Expires=1674447635&Signature=FVeO6IB6dlzttB3tFWv1A5tM0eOXA-jxKqOkBgUXUt3CxoF2F1r6tgsIotBqZL1tjEZ3oNpHITO733wL-JEXqhbHjl83Dxy1faId~K~T2vz~hob9iURaxEaTCtBi2ApLgrPLkAR7dPycx~A5PDrO3Pq144zsPC-jw9Mv9m5wbe~UTbWb0tOhUBckLh2X4sX5PkOIp8pwZQuHZ9hShYYnFQpjDuj4tQlYrVMbNitvoAKrLAeOz8mBV5BDCdilC-~2G4DtPL5HVydxyL~O9z0dO3hTr7tEMhVWymRZSaAwHPehxvQ4KkCb85uI2aMAKFe9sn5pBJD49YkZkEjDNCSK6g__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

- B. P. Lathi (2010). *Modern Digital and Analog Communication Systems.* [2022] from https://edisciplinas.usp.br/pluginfile.php/5251120/mod_resource/content/1/B.%20P.%20Lathi%2C%20Zhi%20Ding%20-%20Modern%20Digital%20and%20Analog%20Communication%20Systems-Oxford%20University%20Press%20%282009%29.pdf

- W.A. Smith (2022). Read an Analog Input with Arduino. [2022] from https://startingelectronics.org/beginners/arduino-tutorial-for-beginners/read-an-analog-input-with-arduino

- Ze-bin Wu (2019). Vector quantization: a review. [2022] from https://link.springer.com/content/pdf/10.1631/FITEE.1700833.pdf?pdf=inline%20link

- Kunjabihari Swain (2021). LI-Care: A LabVIEW and IoT Based eHealth Monitoring System. [2022] from https://link.springer.com/content/pdf/10.1631/FITEE.1700833.pdf?pdf=inline%20link

- Brian Evans. (2011). Beginning Arduino Programming. [2022] from *https://ebookcentral.proquest.com/lib/bcu/detail.action?docID=883784*

- Zilong Jiao. (2018). Analyses on Methods for Noise Reduction of Weak Current Signals. [2022] from *https://ieeexplore.ieee.org/document/9044990*

- Onur Toker. (2019). A Python Based Testbed for Real-Time Testing and Visualization using TI's 77 GHz Automotive Radars. [2022] from *https://ieeexplore.ieee.org/document/9062830*

- Li Lee (1996). Speaker normalization using efficient frequency warping procedures. [2022] from https://ieeexplore.ieee.org/abstract/document/541105

- Ghita K (2018). Mobile Apps Engineering Design, Development, Security, and Testing. [2022] from https://www.google.co.uk/books/edition/Mobile_Apps_Engineering/iouADwAAQBAJ?hl=en&gbpv=1&dq=mobile+apps+engineering:+design,+development,+security,+and+testing+ebook&pg=PP1&printsec=frontcover

- Hwansoo Kang (2015). Application Study on Android Application Prototyping Method using App Inventor. [2022] from *https://sciresol.s3.us-east-2.amazonaws.com/IJST/Articles/2015/Issue-19/Article26.pdf*

- Ivano Malavolta (2016). Web-Based Hybrid Mobile Apps: State of the Practice and Research Opportunities. [2022] from https://ieeexplore.ieee.org/document/7832988

- Anshuman Kamboj (2021). Development of Android App-based Portable Water Quality Testing Device using Arduino. [2022] from https://ieeexplore.ieee.org/document/9596398

- Siu-Cheung Kong (2019). Computational Thinking Education. [2022] from https://library.oapen.org/bitstream/handle/20.500.12657/23182/1006971.pdf?sequence=1#page=39

- Jignesh Patoliya (2015). Arduino controlled war field spy robot using night vision wireless camera and Android application. [2022] from https://ieeexplore.ieee.org/abstract/document/7449624

- Seree Khunchai (2019). Development of Smart Home System Controlled by Android Application. [2022] from https://ieeexplore.ieee.org/abstract/document/8790919

- Ján Hurtuk (2017). The Arduino platform connected to education process. [2022] from https://ieeexplore.ieee.org/abstract/document/8118531

- Terena Bell (2021). What is NLP? Natural language processing explained. [2022] from https://www.proquest.com/docview/2521822106?accountid=10749&parentSessionId=B6NZXQqLBpuWI7%2FMl%2FSQRg8PUbkTgGbTQFGG%2F%2BzrQQM%3D&pq-origsite=primo

- Claire D. Costa (2020). Python Libraries for Natural Language Processing. [2022] from https://towardsdatascience.com/python-libraries-for-natural-language-processing-be0e5a35dd64

- Steve Emms (2019). 8 Excellent C++ Natural Language Processing Tools. [2022] from https://www.linuxlinks.com/excellent-c-plus-plus-natural-language-processing-tools/

- Nitin Hardeniya (2016). Natural Language Processing Python and NLTK. [2022] from https://www.pdfdrive.com/natural-language-processing-python-and-nltk-e158232635.html

- Inés Roldós (2020). Major Challenges of Natural Language Processing (NLP). [2022] from https://monkeylearn.com/blog/natural-language-processing-challenges/

- Hina Magsi (2018). Analysis of signal noise reduction by using filters. [2022] from https://ieeexplore.ieee.org/abstract/document/8346412

- Jacob Benesty (2005). Study of the Wiener Filter for Noise Reduction. [2022] from https://link.springer.com/chapter/10.1007/3-540-27489-8_2

- S Kshipra Prasadh (2017). Efficiency analysis of noise reduction algorithms: Analysis of the best algorithm of noise reduction from a set of algorithms. [2022] from https://ieeexplore.ieee.org/document/8365318

9 Appendices

9.1 Appendix A: Mind Map

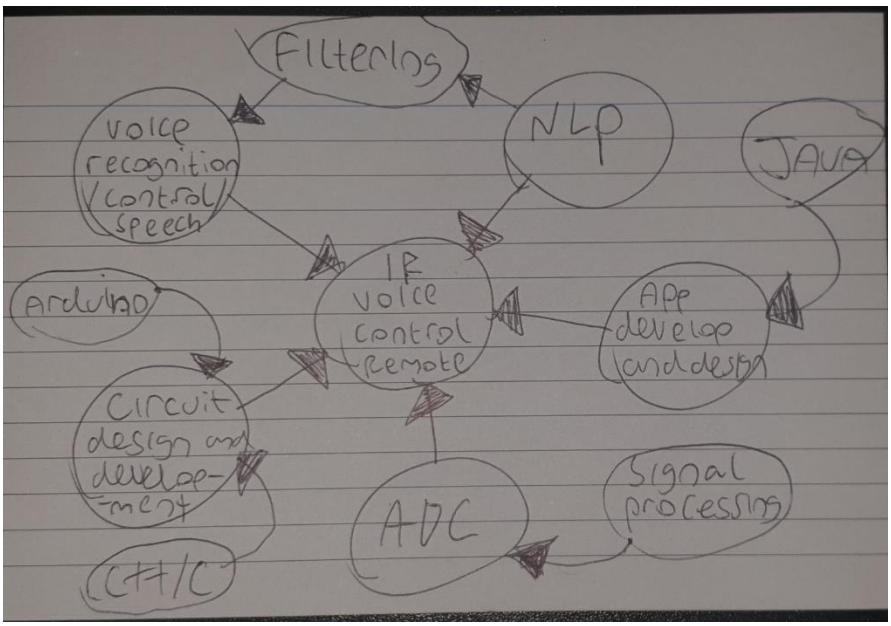


Figure 79 – Mind Map

9.2 Appendix B: Gantt Chart

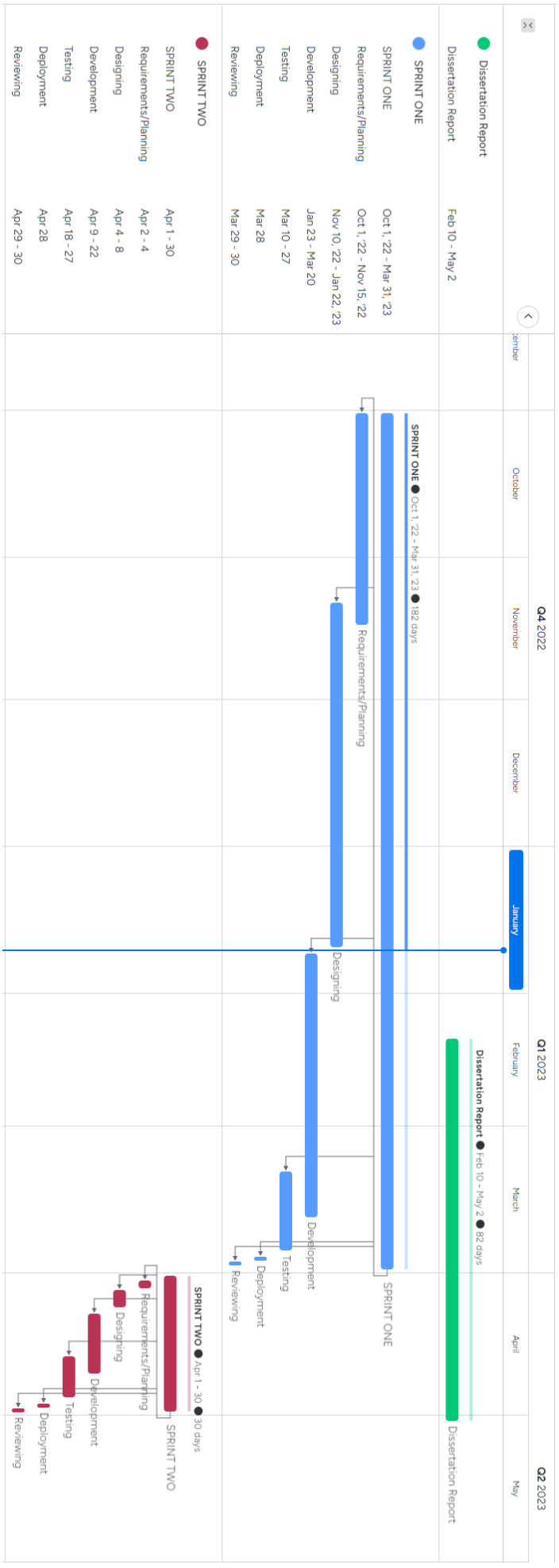


Figure 80 – Gantt Chart

9.3 Appendix C: Device Screenshots

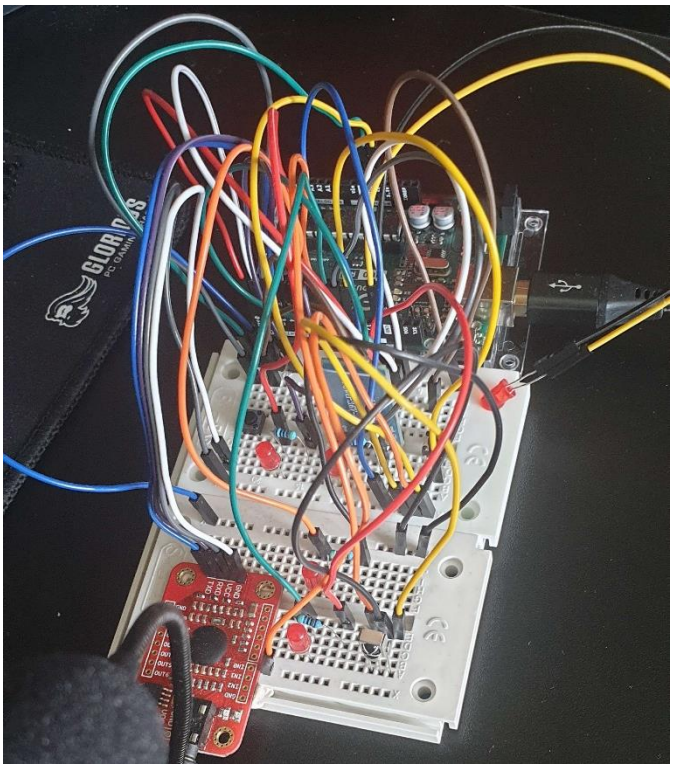


Figure 81 – Device Turned OFF

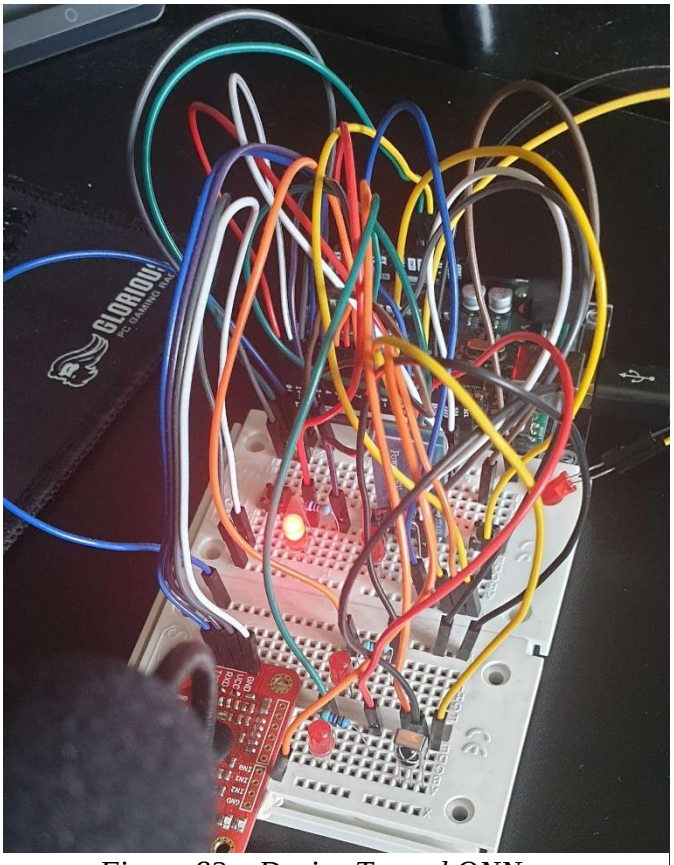


Figure 82 – Device Turned ONN

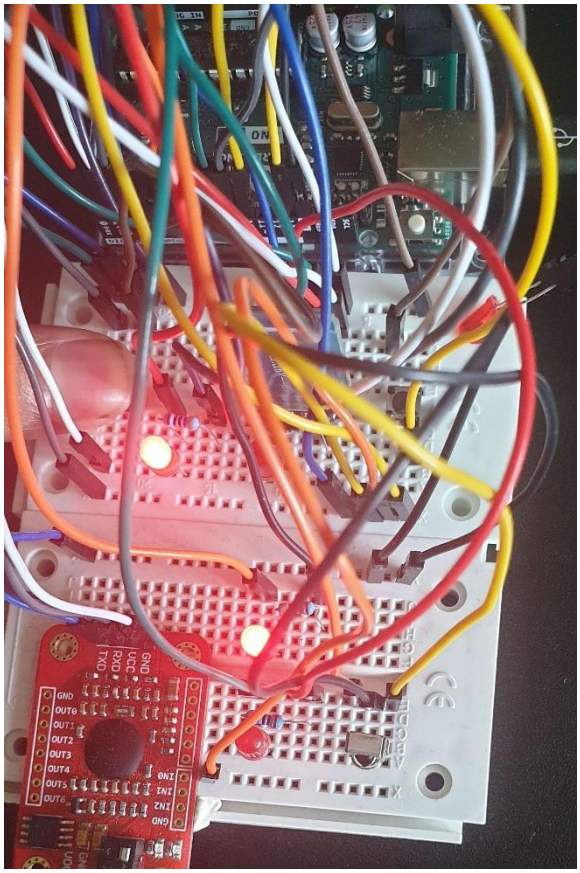


Figure 83 – Mic ONN and Parent Command Active

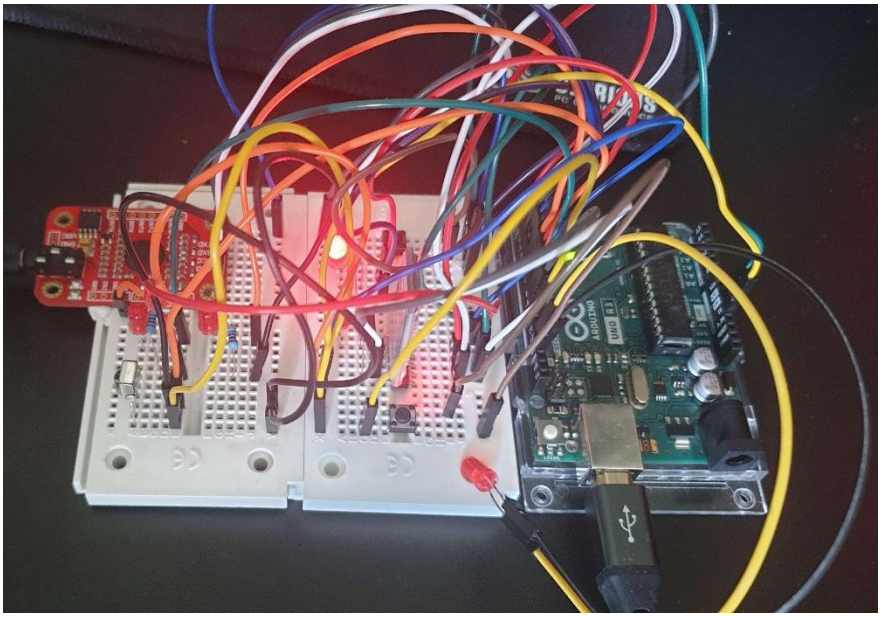


Figure 84 – Device Side Profile 1

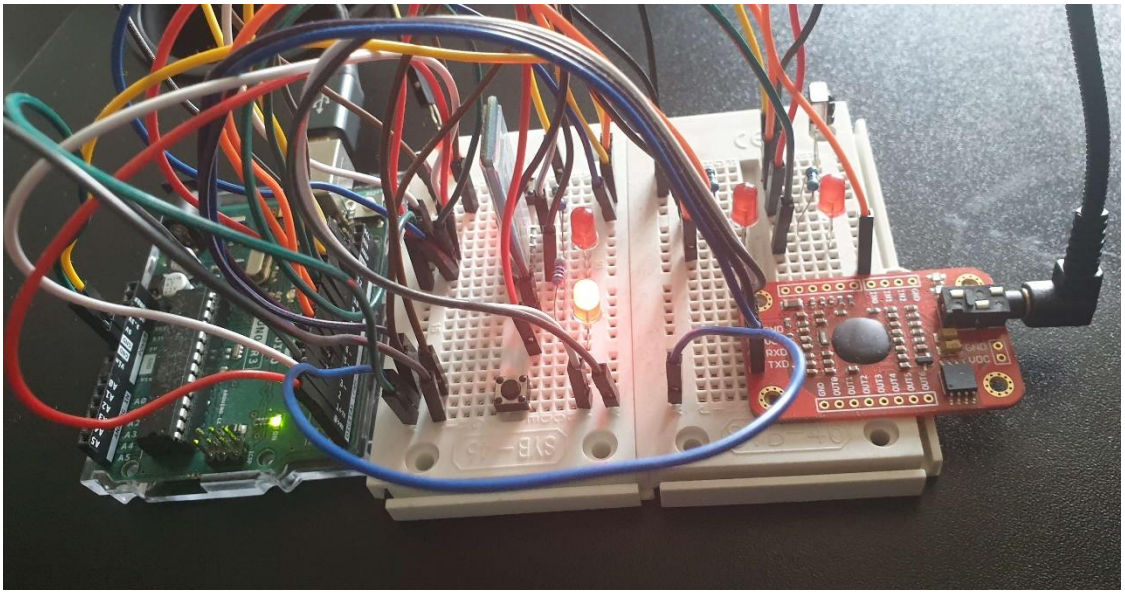


Figure 85 – Device Side Profile 2

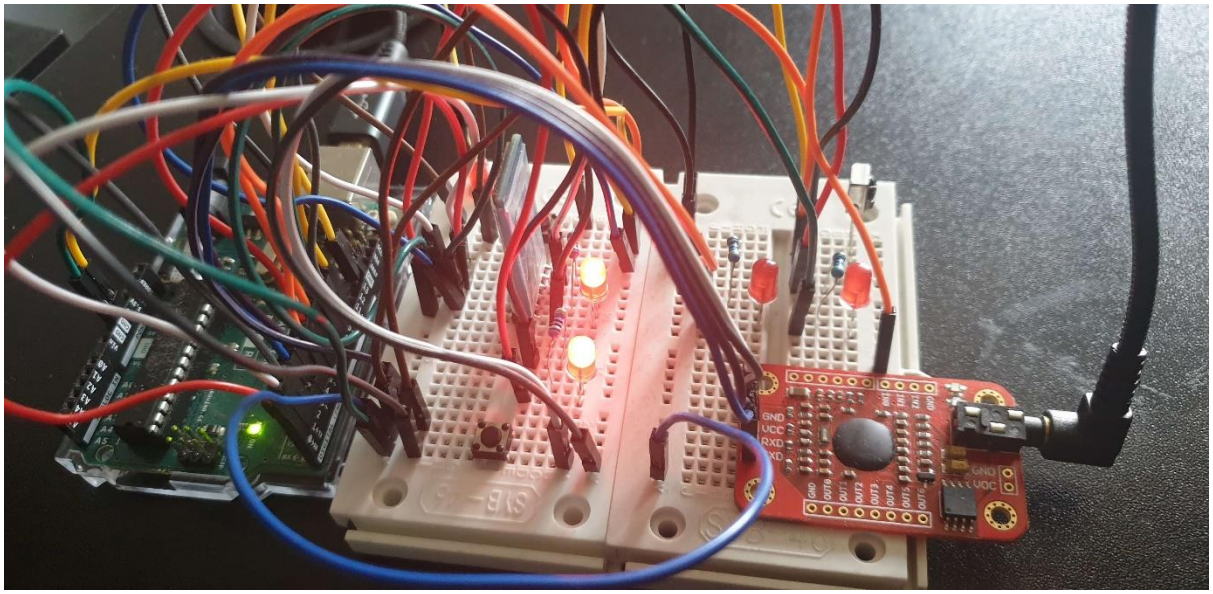


Figure 86 – Device App Connected

9.4 Appendix D: Program Code

```
// used libraries
#include <SoftwareSerial.h>
#include <IRremote.h>
#include "VoiceRecognitionV3.h"
#include <EEPROM.h>

// define voice command variable with their ID
#define POWER (1)
#define PAUSE (3)
#define RESUME (4)
#define VOLUME (5)
#define UP (6)
#define DOWN (7)
#define CHANNEL (8)
#define NEXT (9)
#define PREVIOUS (10)
#define NUMBER (11)
#define ZERO (12)
#define ONE (13)
#define TWO (14)
#define THREE (15)
#define FOUR (16)
#define FIVE (17)
#define SIX (18)
#define SEVEN (19)
#define EIGHT (20)
#define NINE (21)

// set BT object and pins to connect to Blutooth Module
SoftwareSerial BT(0, 1);
VR myVR(3, 4);
uint8_t records[7];  // save record
uint8_t buf[64];

// string that forms message recived from the app
String state;

// for measuring response time and latency
unsigned long startTimeRT;
unsigned long endTimeRT;
unsigned long startTimeLate;
unsigned long endTimeLate;

// variables used for training remote with IR codes so it can be universal
```

```
int numCodesSaved = 0;
unsigned long irCodes[16];
int resetButtonState = 0;

// PIN variables
const byte BUTTON_PIN_MIC = 2;
const byte LED_PIN = 5;
const byte LED_PIN_POWER = 6;
const byte IR_EMITTER = 7;
const byte LED_VALID_COMMAND = 8;
const byte LED_APP_CONNECT = 9;
const byte BUTTON_CODES_RESET = 10;
const byte IR_LED = 13;

// set with IR recivier and IR LED, so IR codes can be sent and recived
IRsend irsend(IR_LED);
IRrecv IR(IR_EMITTER);

// ------------------------------------------------------------------------
----------

// ------ pre-render code for the voice control module that was not written by
me. ------

void printSignature(uint8_t *buf, int len) {
  int i;
  for (i = 0; i < len; i++) {
    if (buf[i] > 0x19 && buf[i] < 0x7F) {
      Serial.write(buf[i]);
    } else {
      Serial.print("[");
      Serial.print(buf[i], HEX);
      Serial.print("]");
    }
  }
}

void printVR(uint8_t *buf) {
  Serial.println("VR Index\tGroup\tRecordNum\tSignature");
  Serial.print(buf[2], DEC);
  Serial.print("\t\t");

  if (buf[0] == 0xFF) {
    Serial.print("NONE");
  } else if (buf[0] & 0x80) {
    Serial.print("UG ");
    Serial.print(buf[0] & (~0x80), DEC);
  } else {
```

```
    Serial.print("SG ");
    Serial.print(buf[0], DEC);
  }
  Serial.print("\t");
  Serial.print(buf[1], DEC);
  Serial.print("\t\t");
  if (buf[3] > 0) {
    printSignature(buf + 4, buf[3]);
  } else {
    Serial.print("NONE");
  }
  Serial.println("\r\n");
}

// -----------------------------------------------------------------------------
----------

// ------ all the code from here on is written by me ------

// saves given variables in the ROM so the values remain after the program is
restarted.
// this was used to save the IR codes.
void saveEEPROM() {
  // 2 variables are save being, irCodes and numCodesSaved.
  EEPROM.begin();
  EEPROM.put(sizeof(bool), numCodesSaved);
  EEPROM.put(sizeof(bool) + sizeof(int), irCodes);
  EEPROM.end();
}

// loads the saved variables with intact values from the ROM.
void loadEEPROM() {
  // 3 variables are loaded being, irCodes and numCodesSaved.
  EEPROM.begin();
  EEPROM.get(sizeof(bool), numCodesSaved);
  EEPROM.get(sizeof(bool) + sizeof(int), irCodes);
}

// resets the saved variables in the ROM.
// so different IR codes from different remotes can be configured.
void resetEEPROM() {
  // loops the ROM values and sets them all to 0.
  for (int i = 0; i < EEPROM.length(); i++) {
    EEPROM.write(i, 0);
  }
}

// used for calculating response time for preformance testing
```

```
void startTimerRT() {
  startTimeRT = micros();
}
void endTimerRT() {
  endTimeRT = micros();
}
void calculateRT(unsigned long endTime, unsigned long startTime) {
  unsigned long responseTime = endTime - startTime;
  Serial.print("\nRT: ");
  Serial.print(responseTime);
  Serial.println(" µs");
}

// used for calculating latency for preformance testing
void startTimerLate() {
  startTimeLate = micros();
}
void endTimerLate() {
  endTimeLate = micros();
}
void calculateLate(unsigned long endTime, unsigned long startTime) {
  unsigned long latency = endTime - startTime;
  Serial.print("\nL: ");
  Serial.print(latency);
  Serial.println(" µs");
}


// resets the program which needs to happen if voice control module loses
power, which happends if the mic button is not pressed.
void (*resetFunc)(void) = 0;

// loads the inital batch of commands whith, 3 of them being parent commands
that loads even more commands as well as clearing the inital commands.
// this was to not cause heavy load onto the arduino to load all the commands
at once.
void commandLoad() {
  myVR.clear();
  myVR.load((uint8_t)POWER);    // normal command
  myVR.load((uint8_t)VOLUME);   // parent command
  myVR.load((uint8_t)CHANNEL);  // parent command
  myVR.load((uint8_t)NUMBER);   // parent command
  myVR.load((uint8_t)RESUME);   // normal command
  myVR.load((uint8_t)PAUSE);    // normal command
}

// prints the given IR code to the specified IR LED with "IRsend
irsend(IR_LED);".
```

```
void printIRcode(int HEX_CODE) {
  // measuring latency. calcuation is done BEFORE the output
  endTimerLate();
  calculateLate(endTimeLate, startTimeLate);
  irsend.sendNEC(HEX_CODE, 32);
  // measuring response time. calcuation is done AFTER the output
  endTimerRT();
  calculateRT(endTimeRT, startTimeRT);
}

// function handels all the app logic
void appLogic() {
  myVR.clear();
  BT.begin(9600);
  bool validCommand = true;

  // for calculating reponse time and latency of the apps voice/button
controls
  if (BT.available()) {
    startTimerRT();
    startTimerLate();
  }

  // loops to build the incoming message from the app
  // state string stores the message
  while (BT.available()) {
    char c = BT.read();
    state += c;
  }

  // now checks what message was said to print out coresponding IR code to the
message
  if (state.length() > 0) {
    //Serial.println(state);
    if (state == "enter") {
      digitalWrite(LED_APP_CONNECT, HIGH);
    } else if (state == "exit") {
      digitalWrite(LED_APP_CONNECT, LOW);
    } else if (state == "power") {
      printIRcode(irCodes[0]);
    } else if (state == "pause") {
      printIRcode(irCodes[1]);
    } else if (state == "resume") {
      printIRcode(irCodes[2]);
    } else if (state == "number zero" || state == "number 0") {
      printIRcode(irCodes[3]);
    } else if (state == "number one" || state == "number 1") {
      printIRcode(irCodes[4]);
```

```
    } else if (state == "number two" || state == "number 2") {
      printIRcode(irCodes[5]);
    } else if (state == "number three" || state == "number 3") {
      printIRcode(irCodes[6]);
    } else if (state == "number four" || state == "number 4") {
      printIRcode(irCodes[7]);
    } else if (state == "number five" || state == "number 5") {
      printIRcode(irCodes[8]);
    } else if (state == "number six" || state == "number 6") {
      printIRcode(irCodes[9]);
    } else if (state == "number seven" || state == "number 7") {
      printIRcode(irCodes[10]);
    } else if (state == "number eight" || state == "number 8") {
      printIRcode(irCodes[11]);
    } else if (state == "number nine" || state == "number 9") {
      printIRcode(irCodes[12]);
    } else if (state == "volume up") {
      for (int i = 0; i < 5; i++) {
        printIRcode(irCodes[13]);
      }
    } else if (state == "volume down") {
      for (int i = 0; i < 5; i++) {
        printIRcode(irCodes[14]);
      }
    } else if (state == "channel next") {
      printIRcode(irCodes[15]);
    } else if (state == "channel previous") {
      printIRcode(irCodes[16]);
    } else {
      validCommand = false;
    }
    // state string is reset to nothing so a new message can be created after
new data from the app is loaded.
    state = "";

    // check if input was a valid command.
    if (validCommand) {
      userFeedback();
    }
  }
}

// provides visual feedback to the user of an LED turning on for 2 seconds for
valid and successful actions.
// was also used for testing and debugging
void userFeedback() {
  digitalWrite(LED_VALID_COMMAND, HIGH);
  delay(500);
```

```
  digitalWrite(LED_VALID_COMMAND, LOW);
}

// checkResetButton function that checks if the reset button was pressd, if so
it resets saved ROM variables.
void checkResetButton() {
  if (digitalRead(BUTTON_CODES_RESET) == LOW) {
    Serial.print("\nIR codes Reset");
    resetEEPROM();
    resetFunc();
  }
}

// COMMENT
void waitForButton() {
  myVR.clear();
  Serial.begin(9600);

  // this loops while the mic button is not pressed
  while (digitalRead(BUTTON_PIN_MIC) == HIGH) {
    appLogic();
    checkResetButton();

    // checks for incoming IR signals
    if (IR.decode()) {
      // for calculating reponse time and latency of the main button controls
      startTimerRT();
      startTimerLate();
      // save hex code as string, and then unsigned long
      String inputCode = String(IR.decodedIRData.decodedRawData, HEX);
      unsigned long hexCode = strtoul(inputCode.c_str(), NULL, 16);


      // code trains IR signals
      if (numCodesSaved < 17 && hexCode != 0) {
        irCodes[numCodesSaved] = hexCode;
        numCodesSaved++;
        Serial.print("\nCode captured: ");
        Serial.print(numCodesSaved);
        Serial.print(" - ");
        Serial.print(hexCode);
        userFeedback();
        if (numCodesSaved == 17) {
          saveEEPROM();
          // saves hex code in the ROM
        }
      } else {
        // normal button controls to output hex code through IR LED
```

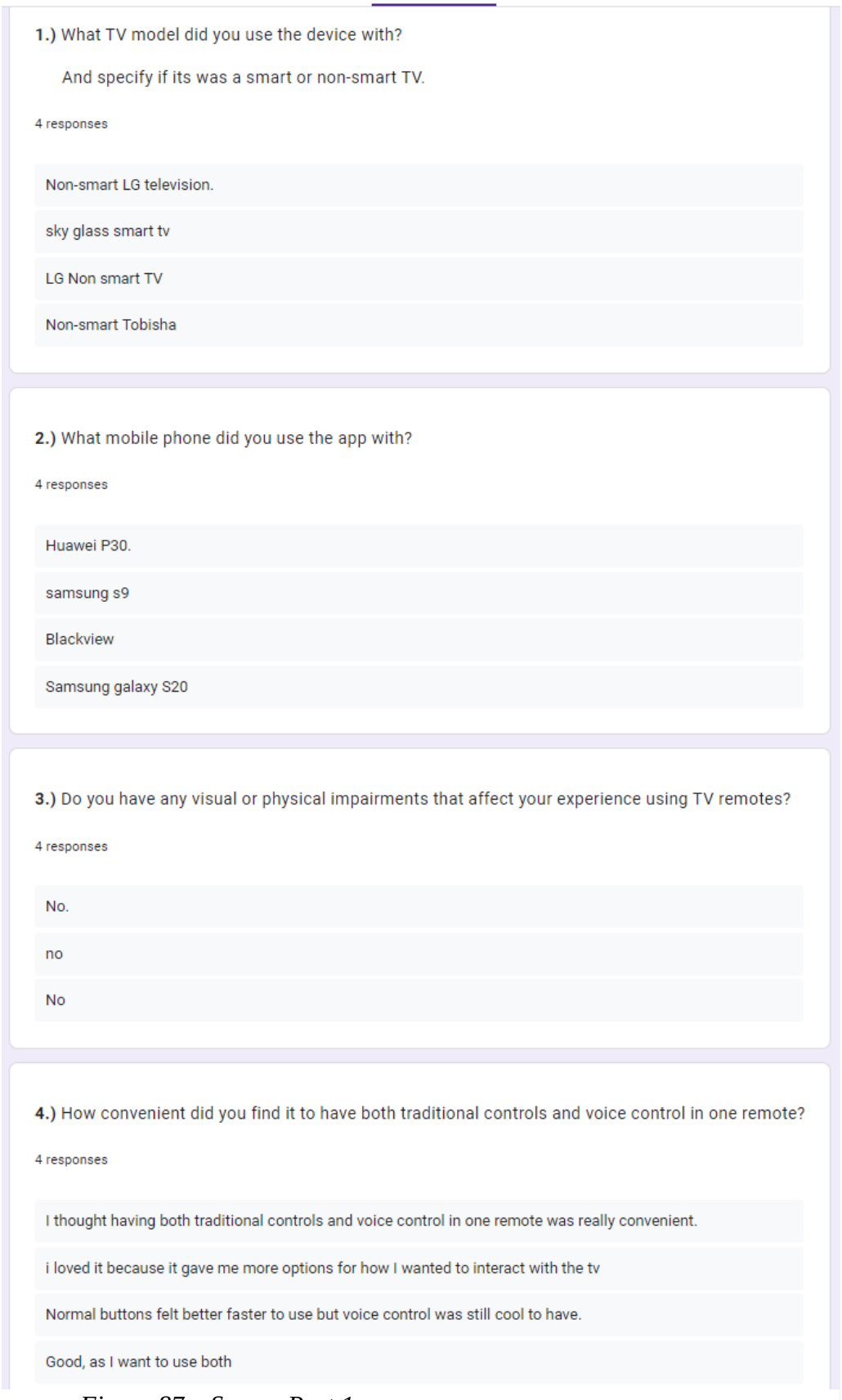
```
        if (hexCode != 0) {
          printIRcode(hexCode);
          Serial.println(hexCode, HEX);
        }
      }
      IR.resume();
    }
  }
}

// ---------------------------------------------------------------------------
----------

// setup
void setup() {
  // loads variables saved in ROM before anything else
  EEPROM.begin();
  loadEEPROM();

  // set all pin modes
  pinMode(BUTTON_PIN_MIC, INPUT_PULLUP);
  pinMode(BUTTON_CODES_RESET, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  pinMode(LED_PIN_POWER, OUTPUT);
  pinMode(LED_VALID_COMMAND, OUTPUT);
  pinMode(LED_APP_CONNECT, OUTPUT);
  pinMode(IR_LED, OUTPUT);

  // shows that the device is on
  digitalWrite(LED_PIN_POWER, HIGH);

  IR.enableIRIn();
  myVR.begin(9600);
  Serial.begin(115200);

  // checks if the mic has power or not
  if (myVR.clear() == 0) {
    Serial.println("Recognizer cleared.");
    commandLoad();
  } else {
    waitForButton();
    resetFunc();
  }
}

// ---------------------------------------------------------------------------
----------
```

```
// loop
void loop() {
  static bool lastButtonState = LOW;
  bool currentButtonState = digitalRead(BUTTON_PIN_MIC);


  if (currentButtonState == HIGH && lastButtonState == LOW) {
    digitalWrite(LED_PIN, LOW);
    resetFunc();  // reset the program
  }
  lastButtonState = currentButtonState;  // update last button state

  bool validCommand = true;

  int ret;
  ret = myVR.recognize(buf, 50);
  if (ret > 0) {
    startTimerRT();
    startTimerLate();
    switch (buf[1]) {
      case POWER:
        printIRcode(irCodes[0]);
        break;
      case PAUSE:
        printIRcode(irCodes[1]);
        break;
      case RESUME:
        printIRcode(irCodes[2]);
        break;
      case NUMBER:
        digitalWrite(LED_PIN, HIGH);
        myVR.clear();
        myVR.load((uint8_t)SEVEN);  // sub command
        myVR.load((uint8_t)EIGHT);  // sub command
        myVR.load((uint8_t)NINE);   // sub command
        myVR.load((uint8_t)ZERO);   // sub command
        myVR.load((uint8_t)ONE);    // sub command
        myVR.load((uint8_t)TWO);    // sub command
        myVR.load((uint8_t)THREE);  // sub command
        myVR.load((uint8_t)FOUR);   // sub command
        myVR.load((uint8_t)FIVE);   // sub command
        myVR.load((uint8_t)SIX);    // sub command
        startTimerRT();
        startTimerLate();
        break;
      case ZERO:
        printIRcode(irCodes[3]);
        break;
```

```
case ONE:
  printIRcode(irCodes[4]);
  break;
case TWO:
  printIRcode(irCodes[5]);
  break;
case THREE:
  printIRcode(irCodes[6]);
  break;
case FOUR:
  printIRcode(irCodes[7]);
  break;
case FIVE:
  printIRcode(irCodes[8]);
  break;
case SIX:
  printIRcode(irCodes[9]);
  break;
case SEVEN:
  printIRcode(irCodes[10]);
  break;
case EIGHT:
  printIRcode(irCodes[11]);
  break;
case NINE:
  printIRcode(irCodes[12]);
  break;
case VOLUME:
  /** turn on LED */
  digitalWrite(LED_PIN, HIGH);
  myVR.clear();
  myVR.load((uint8_t)UP);    // sub command
  myVR.load((uint8_t)DOWN);  // sub command
  startTimerRT();
  startTimerLate();
  break;
case UP:
  for (int i = 0; i < 5; i++) {
    printIRcode(irCodes[13]);
  }
  break;
case DOWN:
  for (int i = 0; i < 5; i++) {
    printIRcode(irCodes[14]);
  }
  break;
case CHANNEL:
  digitalWrite(LED_PIN, HIGH);
```

```
        myVR.clear();
        myVR.load((uint8_t)NEXT);      // sub command
        myVR.load((uint8_t)PREVIOUS);  // sub command
        startTimerRT();
        startTimerLate();
        break;
      case NEXT:
        printIRcode(irCodes[15]);
        break;
      case PREVIOUS:
        printIRcode(irCodes[16]);
        break;
      default:
        Serial.println("Voice command undefined");
        validCommand = false;
        break;
    }
    /** voice recognized */
    printVR(buf);

    // check if input was a valid command.
    if (validCommand) {
      userFeedback();
    }
  }
}
```

9.5 Appendix D: Qualitative Research Survey



1.) What TV model did you use the device with?

And specify if its was a smart or non-smart TV.

4 responses

Non-smart LG television.

sky glass smart tv

LG Non smart TV

Non-smart Tobisha

2.) What mobile phone did you use the app with?

4 responses

Huawei P30.

samsung s9

Blackview

Samsung galaxy S20

3.) Do you have any visual or physical impairments that affect your experience using TV remotes?

4 responses

No.

no

No

4.) How convenient did you find it to have both traditional controls and voice control in one remote?

4 responses

I thought having both traditional controls and voice control in one remote was really convenient.

i loved it because it gave me more options for how I wanted to interact with the tv

Normal buttons felt better faster to use but voice control was still cool to have.

Good, as I want to use both

Figure 87 – Survey Part 1

5.) Did you encounter any issues or errors while using the device?

4 responses

Nope, I did not get any errors!

no

No errors.

No

6.) What additional features would you like to see added to the device?

4 responses

More functionality. So more commands, more buttons, more features like customization so people can create their own commands.

create a case for it as it is probably my main complaint because it was hard to use without a case

Make it wireless so it can be portable. Also give it a cover or case.

Add more buttons

7.) What did you like most about the device/app?

4 responses

App is responsive with a good layout, similar to a real television remote so its familiar to use. Many voice commands to use.

good app worked well on my phone without any issues and i liked the colour of black and white

It was very simple to connect it to the remote. The app is good to use.

App and device with all features are easy to use

8.) Would you recommend this universal remote to a friend who also has a non-smart TV?

4 responses

I don't think I have any friends who has a smart television, but I would still recommend it to them because of the phone app.

yes the voice control will help them out a lot and save them time

Yeah. Because I personally enjoy using it, so I don't see why not. I would recommend it to multiple people.

Figure 88– Survey Part 2

8.) Would you recommend this universal remote to a friend who also has a non-smart TV?

4 responses

I don't think I have any friends who has a smart television, but I would still recommend it to them because of the phone app.

yes the voice control will help them out a lot and save them time

Yeah. Because I personally enjoy using it, so I don't see why not. I would recommend it to multiple people.

Yes

9.) What did you dislike most about the device/app?

4 responses

It was not wireless so it could not be placed anywhere.

awkward to use because there is no case you cannot hold it without a case it can only be used on the table

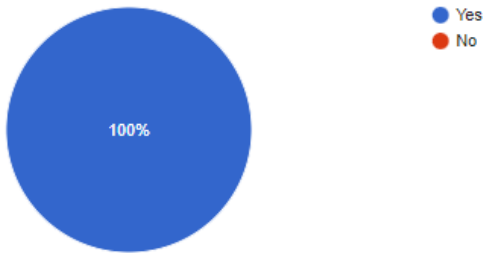
You can't really hold it as there is no cover or case. You should be able to hold the remote. A 3D printed case would be cool.

Its not portable.

Copy

10.) Would you consider buying the device if you saw it in a shop?

4 responses



11.) Rate you overall user experience with my device?

Copy

4 responses

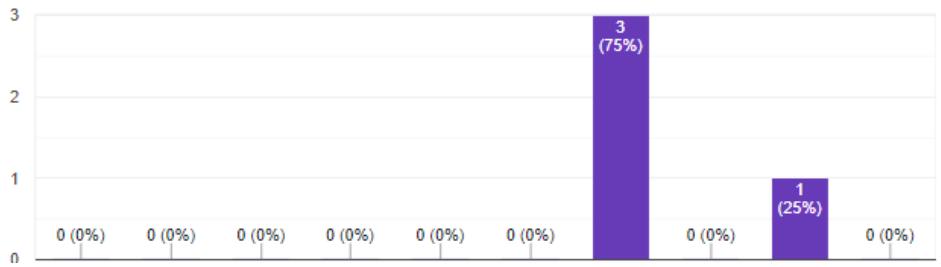


Figure 89 – Survey Part 3