

Universal Infrared System using Raspberry Pi

BY

TAN HUNG MHENG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)

COMPUTER ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

© 2025 Tan Hung Mheng. All rights reserved.

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Technology (Honours) Computer Engineering at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr.

Teoh Shen Khang who has given me this bright opportunity to engage in the development of Universal Infrared System using Raspberry Pi. It is my first step to establish a career in computer vision field. A million thanks to you.

I would also like to express my sincere gratitude to my friends for their patience, unwavering support, and encouragement, especially during challenging times. I am also deeply thankful to my parents and family for their constant love and continuous encouragement throughout the course.

ABSTRACT

This project focuses on the development of a Universal Infrared System using Raspberry Pi for academic research and practical implementation in smart home environments. The aim is to provide a clear methodology and hardware design framework that demonstrates how a single-board computer can serve as a centralized controller for multiple household appliances. The Raspberry Pi, equipped with GPIO (General Purpose Input/Output) pins, is used to interact with electronic components and explore Internet of Things (IoT) applications. By integrating the open-source Linux Infrared Remote Control (LIRC) software, the Raspberry Pi functions as both an infrared signal receiver and transmitter. In order to receive IR commands from conventional remote controls, an IR receiver module is connected to the GPIO pins, while IR LED transmitters are used to send commands to the target appliances. Infrared codes from various remotes such as those for lights, fans, and humidifiers are captured, recorded, and stored. These codes can then be retrieved and transmitted via the Raspberry Pi, allowing users to control multiple devices through a single, unified platform.

Area of Study (Maximum 2): Internet of Things

Keywords (Minimum 5 and Maximum 10): Raspberry Pi, Universal Remote, Internet of Things, Databases, User-friendly Graphical User Interface

TABLE OF CONTENTS

TITLE PAGE	i
COPYRIGHT STATEMENT	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	ix
LIST OF TABLES	xii
LIST OF ABBREVIATIONS	xiii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	3
1.2 Project Objectives	3
1.3 Project Scope and Direction	4
1.4 Contributions	4
1.5 Report Organization	5
1.6 Timeline	6
CHAPTER 2 LITERATURE REVIEW	7
2.1 Previous Works on Home Automation	7
2.1.1 Home Automation system using Arduino	7
2.1.2 Home Automation system using RPi and Arduino	9
2.1.3 Home Automation system using LoRa Technology	10
2.1.4 Universal Infrared Remote Control system using ESP8266 microcontroller	11
2.1.5 Design and Construction of Infrared Remote Controller for Multiple Home Appliances	12
2.2 Limitation of Previous Studies	13
2.3 Proposed Solutions	15

CHAPTER 3 SYSTEM METHODOLOGY/APPROACH	16
3.1 Prototype methodology	16
3.1.1 Requirements gathering and analysis	16
3.1.2 Quick design	17
3.1.3 Prototype Development	18
3.1.4 Supervisor Evaluation	18
3.1.5 Review and Refinement	19
3.1.6 Final Project Development	19
3.1.7 Testing and Validation	19
3.1.8 Deployment	19
3.2 System Architecture Diagram	20
3.3 Use Case Diagram and Description	21
3.4 Activity Diagram	23
 CHAPTER 4 SYSTEM DESIGN	 24
4.1 System Block Diagram	24
4.1.1 Top-down system design	24
4.1.2 Flowchart of Universal Infrared System	26
4.2 System Components Specifications	27
4.2.1 Hardware	27
4.2.2 Software	28
4.3 Circuits and Components Design	30
4.3.1 Infrared Receiver Circuit	30
4.3.2 Infrared Transmitter Circuit	30
4.3.3 Breadboard Layout	30
4.4 System Components Interaction Operations	31
4.3.2 Signal Reception (Learning Mode)	31
4.3.3 Signal Transmission (Control Mode)	31

CHAPTER 5 SYSTEM IMPLEMENTATION	32
5.1 Software Setup	32
5.1.1 RPi OS	32
5.1.2 VNC viewer	32
5.1.3 CustomTkinter	32
5.2 Hardware Setup	33
5.3 Setting and Configuration	34
5.3.1 LIRC Installation and Configuration on RPi	34
5.3.2 IR signals recording and transmission using terminal	36
5.3.3 JSON database configuration	39
5.3.4 Reverse Engineering of A/C Remote Control signals	41
5.3.5 Dynamic Code Generation and Transmission of A/C remote	44
5.4 System Operation	46
5.5 Implementation Issues and Challenges	51
5.6 Concluding Remark	52
 CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	 53
6.1 System Testing and Performance Metrics	53
6.1.1 Hardware testing	53
6.1.2 Signal Recording and Storage	53
6.1.3 GUI functionality Testing	54
6.2 Testing Setup and Result	55
6.3 Project Challenges	56
6.3.1 Weak Infrared Transmission and Coverage	56
6.3.2 Signal Interference	56
6.3.3 Overlapping Command Codes	56
6.3.4 Time and Resource Constraint	56
6.3.5 Manual GUI creation for new remotes	57
6.3.6 Compatibility across different appliances	57
6.4 Objectives Evaluation	58
6.5 Concluding Remark	59

CHAPTER 7 CONCLUSION AND RECOMMENDATION	60
7.1 Conclusion	60
7.2 Recommendation	62
7.2.1 Integration of Additional Communication Protocols	62
7.2.2 Mobile Application Support	62
7.2.3 Security and User Authentication	62
7.2.4 Integration with Smart Home Ecosystem	63
7.2.5 Increasing the Type of Appliances	63
REFERENCES	64
APPENDIX	A-1
POSTER	A-2

LIST OF FIGURES

Figure Number	Title	Page
Figure 1.1	The binary code of Volume Up	2
Figure 1.2	Gantt chart for FYP1	6
Figure 1.3	Gantt chart for FYP2	6
Figure 2.1	Block diagram of System Architecture [7]	7
Figure 2.2	Smart home structure [10]	8
Figure 2.3	Block diagram of home appliances control through various controllers [11]	9
Figure 2.4	Circuit diagram of the smart automation system [12]	10
Figure 2.5	Design of IR smart remote using ESP-microcontroller [13]	11
Figure 3.1	Prototyping methodology block diagram	15
Figure 3.2	Concept of smart universal remote	16
Figure 3.3	Simple connection of IR receiver and transmitter with RPi	17
Figure 3.4	System Architecture diagram of Universal Infrared System	19
Figure 3.5	Use Case Diagram	20
Figure 3.6	Activity Diagram of signal recording	22
Figure 3.7	Activity Diagram of signal transmission	22
Figure 4.1	Top-down system design diagram	23
Figure 4.2	Flowchart of Universal Infrared System	25
Figure 4.3	RPi 4 Model B	26
Figure 4.4	IR Digital Transmitter Module	27
Figure 4.5	IR Digital Receiver Module	27
Figure 4.6	VNC viewer	28
Figure 5.1	IP address of RPi	31
Figure 5.2	Paste the IP address and connect to the VNC viewer	31
Figure 5.3	Command to install customtkinter	31

Figure 5.4	Connection of the hardware	32
Figure 5.5	The enclosure designed for the hardware components using Tinkercad	32
Figure 5.6	Output of mode2 command when button is pressed	34
Figure 5.7	Status of lircd when executing “sudo systemctl status lircd”	35
Figure 5.8	Lircd.conf file of fan remote	36
Figure 5.9	Lircd.conf file of light remote	37
Figure 5.10	Lircd.conf file of humidifier remote	37
Figure 5.11	Code snippets of saving the IR signals into JSON database	38
Figure 5.12	Code snippets of exporting signal from JSON to lircd.conf file	39
Figure 5.13	Example of IR signal stored in JSON	39
Figure 5.14	IR signal recorded when executing mode2 command	40
Figure 5.15	Code snippets of decoding IR signal into binary representation	41
Figure 5.16	Analyse the pattern of the A/C signal using Excel	42
Figure 5.17	Default base binary	43
Figure 5.18	Code snippets of modifying the base binary correspond to the specific functions	43
Figure 5.19	Code snippets of encoding binary back to raw signal	44
Figure 5.20	Code snippets of writing the modified raw signal into lircd.conf file	44
Figure 5.21	Welcoming GUI window	45
Figure 5.22	Main menu window	45
Figure 5.23	Window of IR Signal Recorder	46
Figure 5.24	Window shown when “Record New Signal” button was pressed	46
Figure 5.25	Window shown when “Export to lircd.conf” button was pressed	47
Figure 5.26	GUI of fan remote	47
Figure 5.27	GUI of light remote	48

Figure 5.28	GUI of humidifier remote	48
Figure 5.29	GUI of A/C remote	49
Figure 5.30	Latest A/C state stored in JSON file	49

LIST OF TABLES

Table Number	Title	Page
Table 2.1	Advantage and limitations of previous studies	13
Table 5.1	Function Mapping of A/C	43
Table 6.1	Verification Plans	55

LIST OF ABBREVIATIONS

<i>RPi</i>	Raspberry Pi
<i>LIRC</i>	Linux Infrared Remote Control
<i>IR</i>	Infrared radiation
<i>GPIO</i>	General Purpose Input/Output
<i>IoT</i>	Internet of Things
<i>A/C</i>	Air Conditioner
<i>GUI</i>	Graphical User Interface
<i>LIRCD</i>	LIRC Daemon
<i>JSON</i>	JavaScript Object Notation

Chapter 1

Introduction

In today's era of rapid technological advancement, the Internet has brought significant convenience to our daily lives. One of the most prominent developments is the Internet of Things (IoT), which has become a trending topic in the technology industry, engineering circles, and even policy discussions. According to [1], IoT refers to a network of interrelated devices that connect and exchange data with other IoT devices and the cloud. These devices are typically embedded with technologies such as sensors, actuators, and cloud computing, enabling them to transfer data over a network. As a result, smart devices can communicate with each other and with internet-enabled systems without requiring human-to-human or human-to-computer interaction.

As highlighted in [2], devices such as smartphones and gateways create a vast ecosystem of interconnected systems capable of exchanging data and performing tasks autonomously. For instance, IoT can be applied in industrial settings to control machines and processes, in agriculture to monitor environmental conditions, and in logistics to track inventory and shipments. Indeed, IoT is widely used across numerous industries, including manufacturing, transportation, healthcare, and agriculture, playing a vital role in shaping how we live, work, and interact in a world of rapidly growing internet-connected devices.

To illustrate, the agricultural sector has greatly benefited from IoT adoption. According to [3], IoT devices are used to monitor soil conditions, weather patterns, and crop growth. Sensors, for example, measure soil moisture levels, ensuring that crops are irrigated at appropriate times and preventing water wastage. By addressing challenges such as inaccurate soil humidity readings and inefficient irrigation, IoT has significantly improved farming practices and boosted agricultural productivity.

Beyond industries, IoT has also entered our daily lives in the form of the "Smart Home." As described in [4], a smart home refers to a setup where household appliances and systems can be controlled remotely through internet-connected devices such as smartphones or tablets. Many modern systems are also equipped with voice-command features, such as Amazon Alexa, allowing users to manage home functions effortlessly. For instance, a simple verbal instruction like "Alexa, turn on the living room lights" can activate the lights automatically. This feature

is particularly helpful for individuals with mobility challenges, such as the elderly or disabled. Furthermore, smart homes offer benefits like energy efficiency and cost savings through better energy management. According to [5], they also enhance household security by enabling real-time monitoring with smart cameras and motion sensors, providing alerts in case of suspicious activity.

In addition, infrared (IR) remote controls remain an important element in home automation systems. As discussed in [6], IR remotes transmit signals using infrared light, which is invisible to the human eye, from the remote to the target device. These signals consist of pulses that represent specific binary codes corresponding to different commands, such as turning a device on or adjusting the volume. The IR receiver within the device decodes the pulses into binary codes that are processed by its internal system, which then executes the command. For example, the binary sequence “0010010” may represent a “volume up” command; once decoded, the device increases the sound accordingly.

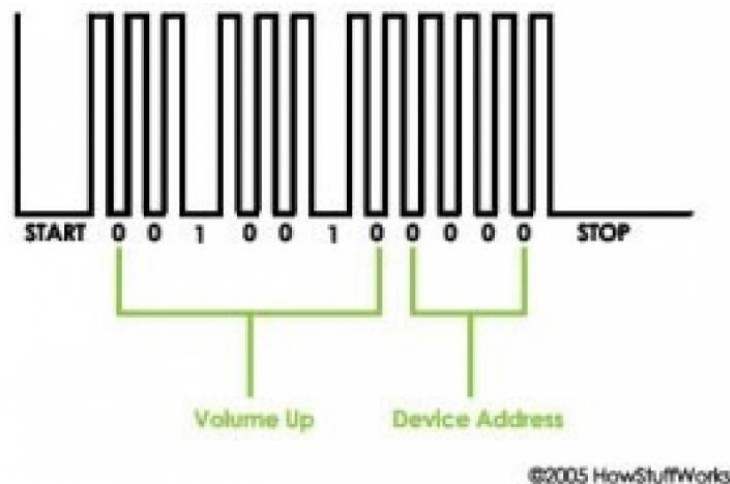


Figure 1.1: The binary code of Volume Up

1.1 Problem Statement and Motivation

In modern homes today, the number of electronic appliances such as refrigerators, air conditioners, and lights has increased drastically. As a result, there are multiple remote controls, each designed to operate a specific appliance, leading to a cluttered living environment. Users are required to keep track of different remotes, increasing the chances of them being misplaced or lost. Furthermore, controlling different appliances with separate remotes can be inconvenient and inefficient. As homes become increasingly connected, the variety and quantity of electronic devices continue to grow, further occupying home spaces and making it more difficult to locate remotes, especially when they become hidden or stuck inside furniture like sofas. To address these challenges, this project is motivated by the need to develop a universal infrared system that enables users to control various devices using a single remote or user interface. In this project, a Raspberry Pi will be utilized to serve as the universal remote, providing a more organized, efficient, and user-friendly solution for modern living environments.

1.2 Project Objectives

The main goal of this project is to develop a universal infrared (IR) system using a RPi that can eliminate the need for multiple remote controls, enabling users to control various appliances with a single remote. The system is designed to capture and record infrared signals from at least three types of electronic appliances, specifically fans, lights, humidifiers and A/C.

The second objective is to design a user-friendly graphical user interface (GUI) that allows users to select and manage different operation modes of the electronic appliances via the Raspberry Pi. The GUI will be developed with a focus on accessibility and ease of use, ensuring that users with varying technical skill levels can navigate and operate the system efficiently.

An additional objective of this project is to implement a structured and scalable IR code management system that supports multiple appliance types. This involves capturing and categorizing infrared signals from various remote controls and storing them in a well-organized format, such as device-specific configuration files or a centralized database. By doing so, the system ensures efficient retrieval and execution of the correct IR codes based on user selection. This approach enhances the flexibility and maintainability of the system, allowing users to

easily add, remove, or update supported devices without modifying the core logic of the control interface.

1.3 Project Scope and Direction

The scope of this project is focused specifically on residential environments, ensuring that the solution addresses the needs and challenges faced by homeowners rather than by commercial spaces such as supermarkets, shops, or other public areas. The universal infrared system is designed to control only electronic appliances commonly found in homes, such as air conditioners, fans, lights, and others. Moreover, this project aims to resolve the problem of managing multiple remote controls by leveraging Internet of Things (IoT) technology, utilizing hardware such as the Raspberry Pi (RPi), and implementing software tools like the Linux Infrared Remote Control (LIRC) libraries.

The project scope includes integrating various hardware components, developing a user-friendly system interface, and designing a centralized solution that enhances the overall user experience in managing household appliances. Special attention is given to capturing, storing, and transmitting infrared signals to ensure the system is robust and adaptable to different home setups. Additionally, the project includes extensive testing and validation in real-world scenarios to ensure the system meets user requirements for reliability, accuracy, and ease of use, aligning with the standards expected of modern smart home solutions.

1.4 Contributions

This project tends to contribute to the field of smart home automation in developing a cost-effective solution by using only Raspberry Pi to serve as the universal infrared remote. Users are allowed to control several devices via a single remote, eliminating the needs of multiples of remote control. Furthermore, this project aims to optimize the user's experience, promotes a more efficient living environment and enhance the home automation. Moreover, throughout this project, it also contributes in decreasing the overall cost and ease user burden as it only requires a universal remote instead of many remote controls.

1.5 Report Organization

The contents of this report are organized across several chapters. Chapter 1 introduces the project, outlining its background, objectives and significance of the project. Chapter 2 presents the literature review, providing an overview and comparison of relevant studies and related works, and lastly proposing the solutions to solve those problems encountered by previous works. Chapter 3 describes the system methodology, including the chosen development model, system architecture, use cases and activity diagrams. Chapter 4 focuses on the system design, covering block diagrams, component specifications, circuit design and system interactions. Chapter 5 details the system implementation, including the hardware and software setup, configurations and overall system operation. Chapter 6 discusses the system evaluation and testing results, performance analysis and challenges encountered, while also assessing whether the project objectives were met. Finally, Chapter 7 concludes the report by summarizing the key findings and offering recommendations for future improvements of the project.

1.6 Timeline

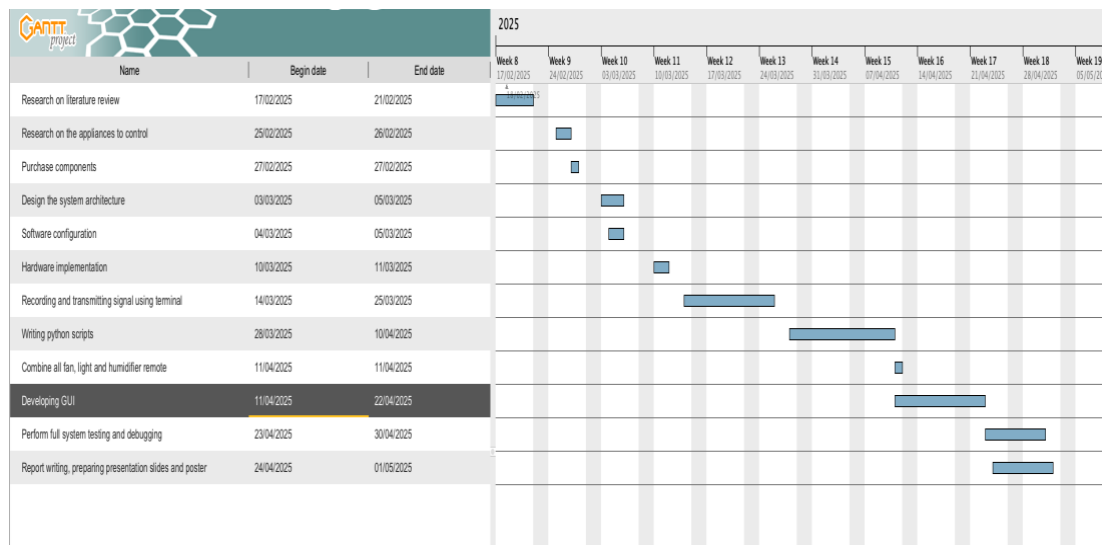


Figure 1.2 Gantt chart for FYP1

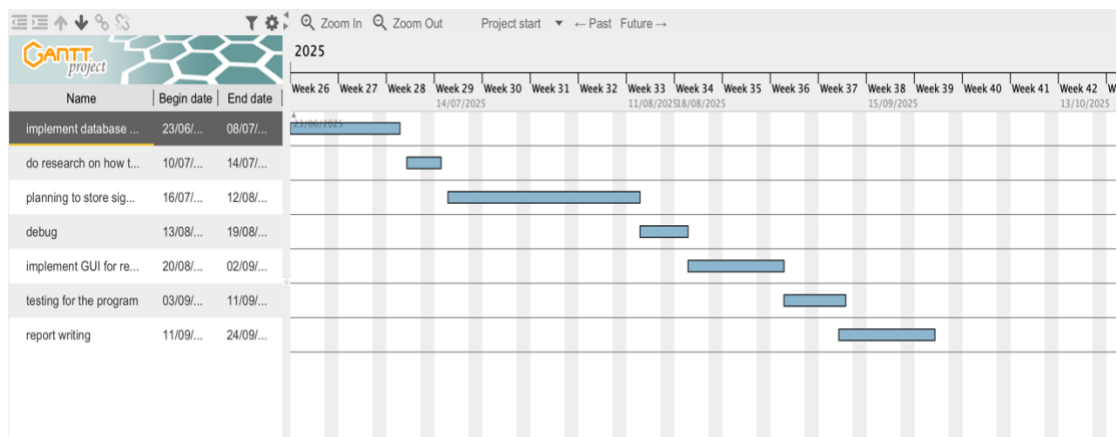


Figure 1.3 Gantt chart for FYP2

Chapter 2

Literature Review

2.1 Previous Works on Home Automation

2.1.1 Home automation system using Arduino

Several recent studies [7, 8, 9] have proposed solutions for home automation, specifically focusing on Arduino-based systems. [7] introduced a home automation system that utilizes Bluetooth technology and an Android application, built around an Arduino microcontroller. The system incorporates an Arduino ATMEGA328 microcontroller, a Bluetooth module (HC-06), and an Android application created using MIT App Inventor 2. This setup enables users to control electrical appliances through a smartphone app or voice commands via Google Assistant. Moreover, the system proposed in [7] offers several advantages over previous approaches. One notable benefit is the voice command feature, which through Google Assistant, enhances accessibility for elderly or physically challenged users who might have difficulty with conventional control methods. Additionally, the use of Bluetooth and Android applications leverages existing smartphone capabilities, minimizing the need for extra components and thereby reducing overall costs.

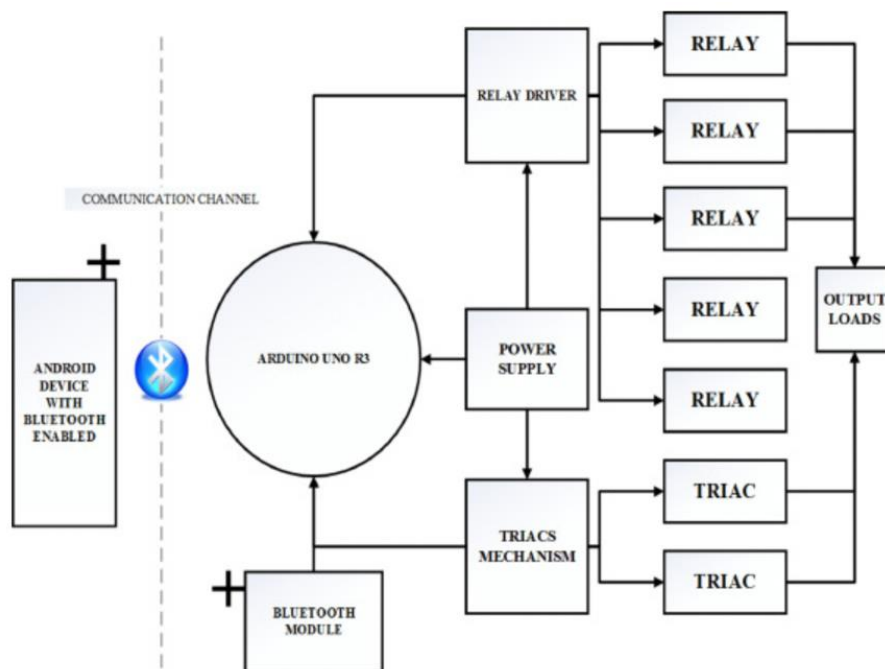


Figure 2.1 Block diagram of System Architecture [7]

In [10], it explores the development and implementation of a cost-effective smart home technology using Arduino FPGA. The authors propose a smart home system that utilizes a set of programmable gadgets built on the Arduino FPGA platform. This system allows users to control various home devices that support IR technology with a single remote control or a smartphone. This research aims to offer a cost-effective and simple solution for establishing a smart home environment, especially for elderly and disabled individuals who may struggle with handling multiple remote controls. The proposed smart home system is based on the Arduino FPGA family, utilizing components such as the Arduino UNO, IR receiver and transmitter modules, and a Bluetooth module (HC-05) to facilitate communication between the Arduino and Bluetooth-enabled devices like smartphones.

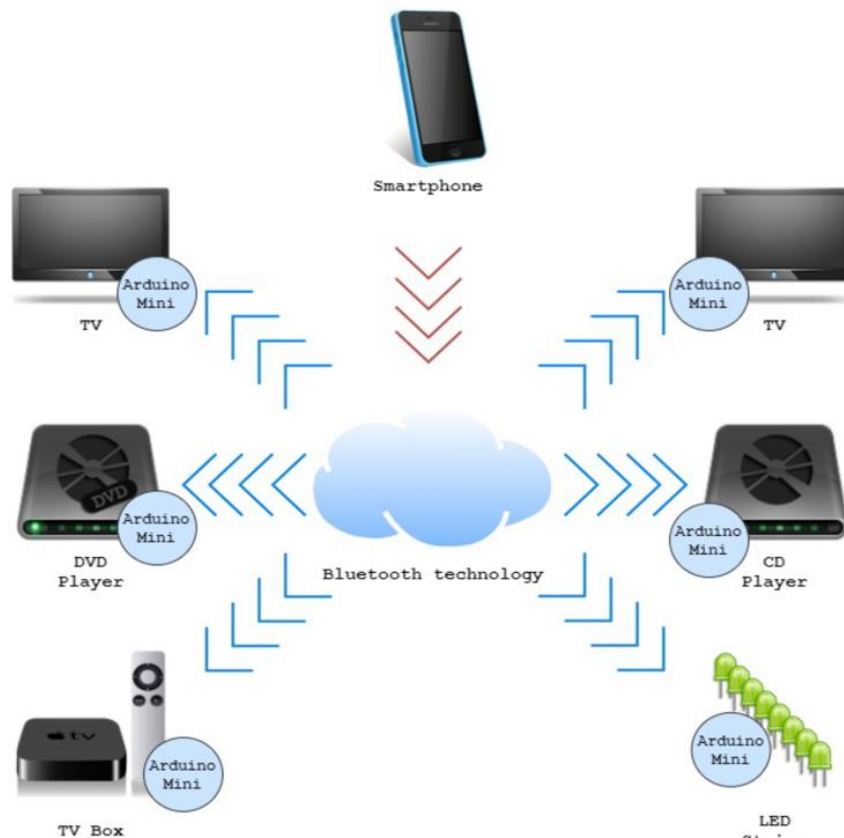


Figure 2.2 Smart home structure [10]

2.1.2 Home automation system using Raspberry Pi and Arduino

In [11], the paper explored the development of an IoT-based platform for controlling multiple home appliances using various methods such as touch panels, Bluetooth, Wi-Fi, and IR remotes. The proposed system utilizes both a Raspberry Pi (RPi) and an Arduino, with the RPi functioning as a server that manages communication between different controllers and appliances through the MQTT protocol. The Arduino acts as a microcontroller, executing commands received from the control inputs and operating the actuators that control the appliances. The project employs the MQTT protocol, a lightweight communication protocol ideal for machine-to-machine interactions. Its effectiveness in low-bandwidth environments makes it particularly well-suited for home automation, where devices need to exchange data without consuming excessive resources. Unlike earlier studies that focused on a single control method, Kolanur et al. [11] incorporated touch-based sensing, smartphone control via Bluetooth or Wi-Fi, and IR remotes into their design, offering users multiple ways to control appliances.

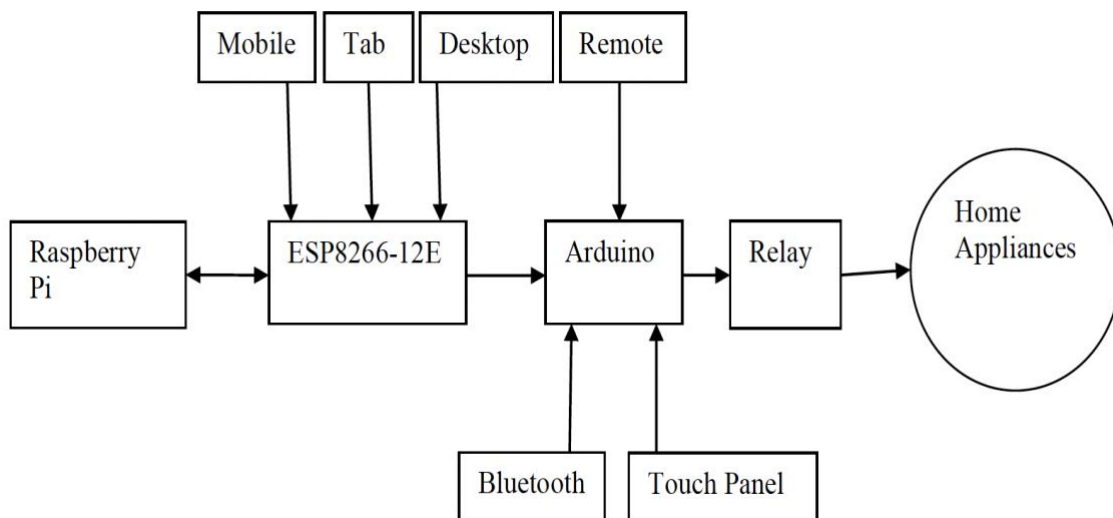


Figure 2.3 Block diagram of home appliances control through various controllers [11]

2.1.3 Home automation system using LoRa Technology

In [12], the authors investigated smart monitoring and control of appliances using a LoRa-based IoT system. LoRa (Long Range) is a low-power wide-area network (LPWAN) technology designed for long-distance communication at a low bit rate. It is commonly used in IoT applications where devices need to communicate across several kilometers while using minimal power, utilizing the Industrial, Scientific, and Medical (ISM) band for communication. In this study, LoRa is integrated with the ESP32 module, which offers both Wi-Fi and Bluetooth capabilities for communication. The system design enables smooth switching between long-range and short-range communication based on the user's location. Additionally, sensors connected to the ESP32 module transmit real-time data to users through a mobile application. The system can detect hazardous situations such as gas leaks and fires and sends notifications to the mobile app, allowing users to respond immediately. Furthermore, users can control multiple devices from a distance of up to 12 kilometers.

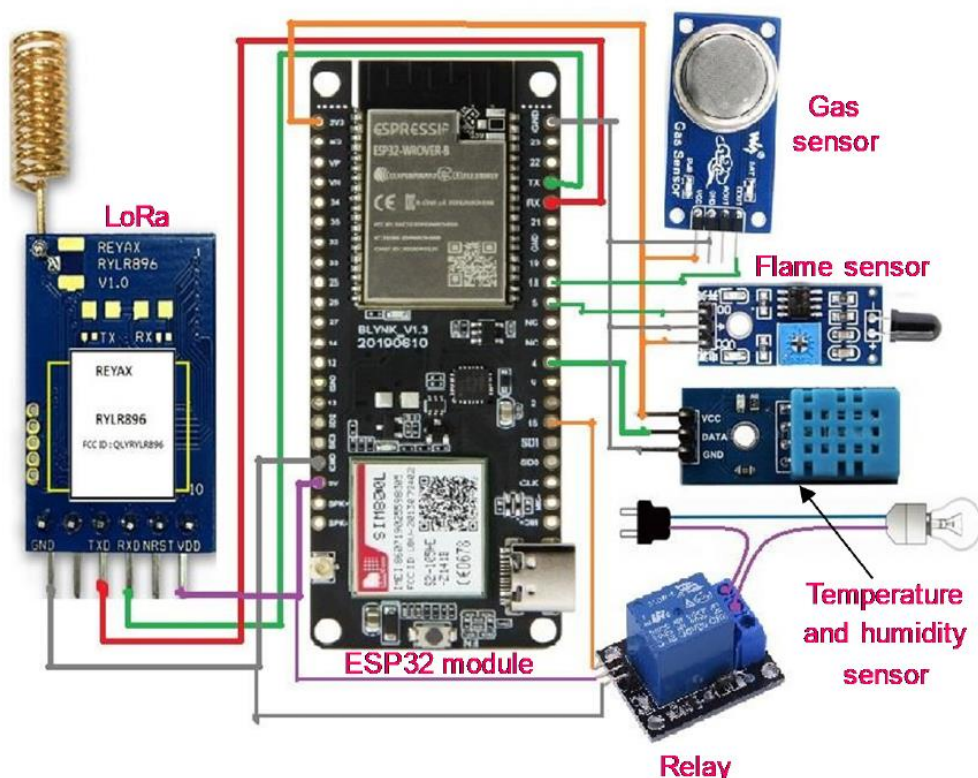


Figure 2.4 Circuit diagram of the smart automation system [12]

2.1.4 Universal infrared remote control system using ESP8266 microcontroller

In [13], Ali and Bao introduced a universal infrared remote control system utilizing the ESP8266 microcontroller. Their design enabled the control of various household appliances by transmitting infrared (IR) signals via the ESP8266 module, which connected to a smartphone application over Wi-Fi. One of its key strengths lies in enabling wireless communication, allowing users to control IR devices remotely via smartphones or other Wi-Fi interfaces, which aligns with smart home trends. Additionally, the use of accessible hardware and open-source tools makes the design low-cost and easily replicable for hobbyists and developers. However, despite these advantages, the system still relies on traditional infrared transmission, which requires a clear line of sight and cannot control devices in separate rooms. Moreover, the paper does not thoroughly address security concerns, as Wi-Fi-based systems can be vulnerable to unauthorized access if encryption and authentication are not properly implemented. Lastly, without feedback mechanisms, users may face uncertainty about whether commands have been successfully executed. Overall, while the design is innovative and practical, it has limitations that should be addressed for more robust real-world applications.

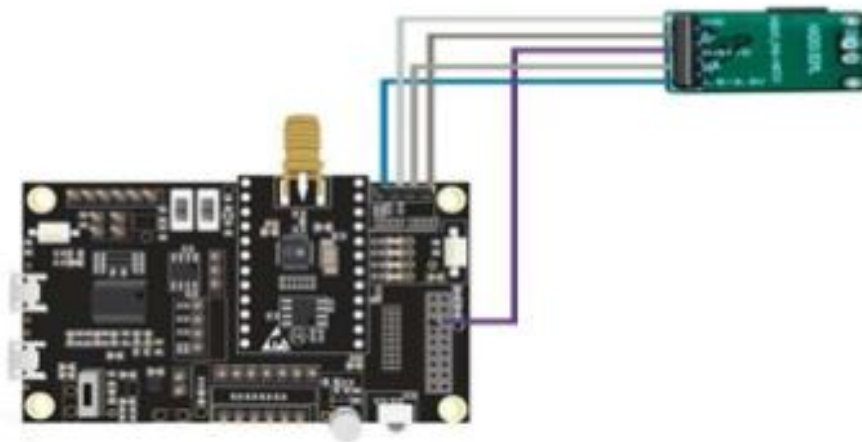


Figure 2.5 Design of IR smart remote using ESP-microcontroller [13]

2.1.5 Design and Construction of Infrared Remote Controller for Multiple Home Appliances

In [14], the authors designed and implemented an Arduino-based infrared remote controller capable of managing multiple household appliances using a single universal remote. The system uses an Arduino UNO connected to an IR receiver module to decode infrared signals from a remote control, which are then mapped to six relays for switching devices such as a TV, fan, motor, and various lights. An LCD display provides real-time feedback by showing the ON/OFF status of each appliance, improving usability. The prototype demonstrates that by learning and decoding IR codes from different protocols, one system can effectively replace multiple remotes at a low cost. This design offers a practical and simple approach to home automation using commonly available components, highlighting the feasibility of local IR-based control for multiple appliances. However, the system is limited to line-of-sight infrared communication and does not incorporate IoT or smartphone connectivity, restricting its scalability and remote accessibility.

2.2 Limitation of Previous Studies

However, there are still some limitations on the previous studies. The advantages and limitations are summarized in below.

Table 2.1 Advantages and limitations of previous studies

Article	Advantages	Limitations
Home Automated System Using Bluetooth and an Android Application [7]	<ul style="list-style-type: none"> • Voice Command Feature • Cost-effective • Integration with smartphones 	<ul style="list-style-type: none"> • Connectivity issue (might fail due to poor internet connection) • Limited to only Android-based platform
Development and Verification of a Smart Remote Control System for Home Appliances [8]	<ul style="list-style-type: none"> • Unified control interface (able to control multiples devices via only a smartphone app) • Cost-effective 	<ul style="list-style-type: none"> • Absence of cloud computing system • Connectivity issue (might fail due to poor internet connection)
A Reasonable Smart Home Technology on the Arduino [10]	<ul style="list-style-type: none"> • Cost-effective • Unified control interface • Ease of deployment 	<ul style="list-style-type: none"> • Bluetooth range constraints (support at limited range of up to 10 meters)
Arduino Based Home Automation System using Android Application [9]	<ul style="list-style-type: none"> • Cost-effective • User-friendly (simple Android app interface) • Unified control interface (able to control multiples devices via only a smartphone app) 	<ul style="list-style-type: none"> • No internet connectivity (unable to control appliances from outside their home using internet) • Reliance on Bluetooth only (unable to function if Bluetooth connection is lost)
Design of IoT based Platform Development for Smart Home Appliances Control [11]	<ul style="list-style-type: none"> • User flexibility (may use WIFI, Bluetooth or touch panel to control) • Efficiency of MQTT (reduce latency) 	<ul style="list-style-type: none"> • Connectivity issue (might fail due to poor internet connection)

Smart Monitoring and Controlling of Appliances Using LoRa Based IoT system [12]	<ul style="list-style-type: none"> • Long range communication (range of up to 12 kilometers) • Support real time to detect the fire or gas leakage 	<ul style="list-style-type: none"> • High power consumption at the receiver end (ESP32 module at the receiver end still led to higher power usage compared to other low-power microcontrollers) • Limited scalability (adding more devices or sensors could lead to network congestion and delay) • Data loss during long distance transmission
Universal infrared remote control system using ESP8266 microcontroller [13]	<ul style="list-style-type: none"> • Leverages of low-cost, low-power chip which makes it easy to integrate IoT features • Cost effective • User-friendly mobile application 	<ul style="list-style-type: none"> • Reliance on IR Line-of-Sight • Consumes more power consumption, not optimized for sleep-modes.
Design and Construction of Infrared Remote Controller for Multiple Home Appliances [14]	<ul style="list-style-type: none"> • Low-cost and simple design using easily available components (Arduino, IR receiver, relays). • Supports multiple appliances (TV, fan, motor, lights) with a single controller. • User-friendly interface with LCD displaying appliance status in real time. • Reusable system since it can learn and decode signals from universal IR remotes. • Reliable control within IR range for direct line-of-sight applications. 	<ul style="list-style-type: none"> • Restricted to line-of-sight communication due to IR technology. • No remote or mobile app access, limiting flexibility for smart home integration. • Scalability challenges, as more appliances may require additional hardware and wiring. • Lacks automation features (e.g., scheduling, sensor-based control). • Not energy-efficient for large-scale smart homes compared to IoT-based alternatives.

2.3 Proposed Solutions

In this project, we proposed and implemented a Universal Infrared Sytem using RPi as the central processing unit. Unlike existing solutions that typically rely on Arduino or ESP8266 microcontrollers, RPi was chosen due to its ability to run a full OS and support advanced software libraries. The RPi facilitated the integration of both hardware and software components, allowing infrared signals to be captured through the HX-1838 receiver, stored in configuration files and retransmitted through digital infrared transmitters. The use of LIRC library simplified the processes of recording, decoding and transmitting signals, while a Python-based GUI built using CustomTkinter enabled intuitive user interaction. This combination of hardware and software provided a scalable, flexible and user-friendly solution for controlling multiple appliances such as fans, lights, humidifiers and A/C from a single platform.

Chapter 3

System Methodology

3.1 Prototype methodology

In this project, prototyping model had been used as the design approach to develop a Universal Infrared System using RPi. Prototyping model is used because user's feedback can be gathered, and it is helpful in refining and improving the product. The project was carried out in several distinct phases, including requirements gathering, system design, prototyping, supervisor evaluation, review and refinement, development, testing, and final deployment.

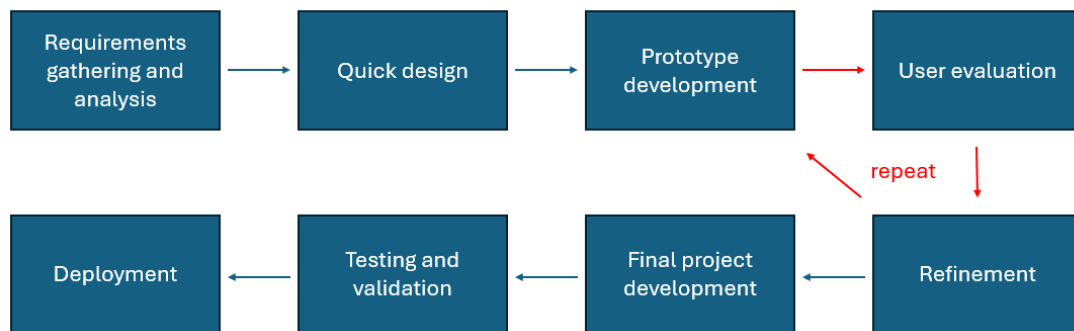


Figure 3.1 Prototyping methodology block diagram

3.1.1 Requirements gathering and analysis

The initial stage of this project is crucial for establishing a clear understanding of how the system works and what the system needs to achieve. During this phase, initial discussions with supervisor will be conducted to collect essential information and requirements. This involves identifying the specific devices to be controlled, such as light, humidifier and fan, and understanding the functionalities desired, such as capturing, storing, and transmitting infrared codes. For the part of the GUI, the system had to be designed in a user-friendly way which is understandable by all of the users. Users are able to see clearly whether which button they press or which device they are controlling. The button layout for each of the remote control was grouped accordingly which prevent users from misunderstanding. Moreover, some icons had been used to represent the remote control, users can quickly identify the devices they are controlling.

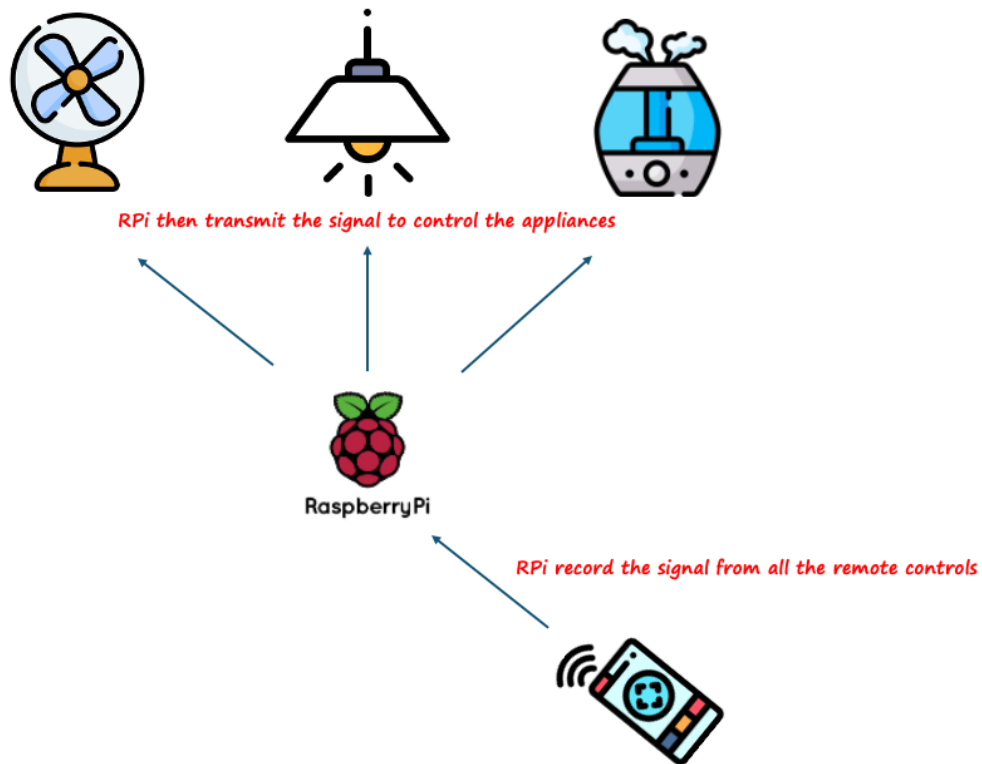


Figure 3.2 Concept of smart universal remote

3.1.2 Quick design

In the quick design phase, a basic structure of the Universal Infrared System is outlined to visualize the core functionality and system flow. The initial design focuses on how the Raspberry Pi will interface with an infrared receiver to capture signals from different household devices such as light, fan, and humidifier. These captured infrared codes will be stored, and organized by device and function. A basic plan for the graphical user interface (GUI) is also proposed, featuring clearly labelled buttons for each device and its corresponding functions to ensure ease of use. The GUI will allow users to easily select a button and send the appropriate infrared command through infrared transmitter modules.

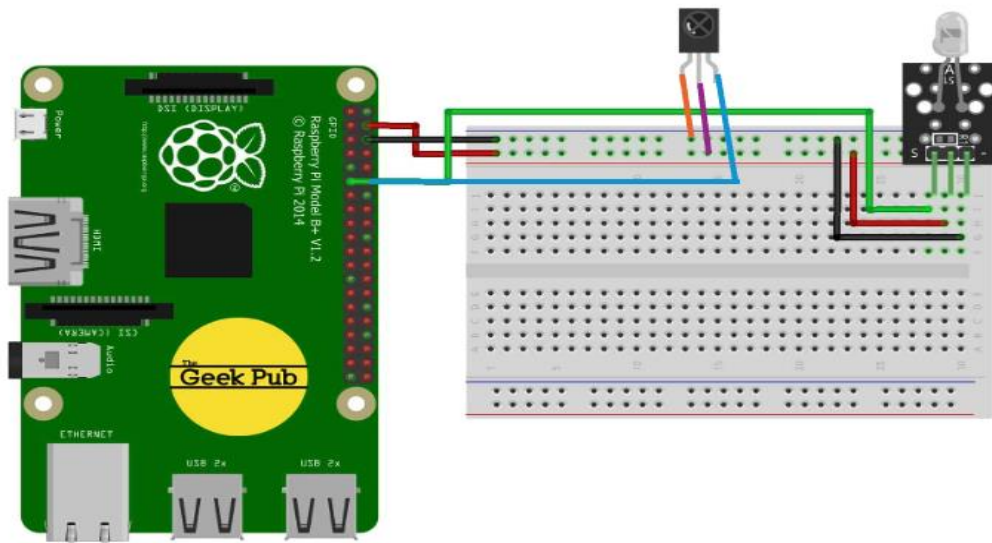


Figure 3.3 Simple connection of the IR receiver and transmitter with RPi

3.1.3 Prototype Development

At this stage, a functional prototype of the Universal Infrared System will be developed, involving both hardware integration and software coding. The prototype will demonstrate key features such as capturing infrared signals from selected devices for instance fan, light and humidifier. Additionally, the system will be capable of retrieving the corresponding infrared codes and transmitting them back to the devices via infrared transmitters. A user-friendly graphical user interface (GUI) will also be incorporated to enable users to control the devices using it.

3.1.4 Supervisor Evaluation

Following the initial prototype development, the system will be presented to the supervisor for evaluation. The supervisor will engage with the prototype by controlling the appliances through the user interface. Throughout this process, the supervisor will assess critical aspects such as system functionality, ease of use, and overall performance. Feedback may be provided on areas like control precision, interface design, and system reliability, which will serve as essential input for further improvements.

3.1.5 Review and Refinement

Based on the supervisor's feedback, the prototype will undergo multiple rounds of refinement to address any identified issues. Improvements may include enhancing the infrared signal transmission range and accuracy, upgrading the GUI, or resolving other technical problems. This refinement phase is vital for ensuring that the system meets all functional and performance requirements. Iterations of supervisor evaluation and system improvement will continue until the prototype closely matches the intended final product.

3.1.6 Final Project Development

Once the supervisor approves the refined prototype, the final version of the Universal Infrared System will be developed. This phase will focus on developing a clear and intuitive GUI, and ensuring precise operation of the universal remote. At this point, the system will be fully functional and ready for use in a smart home environment, capable of controlling multiple electronic devices through a single, streamlined interface.

3.1.7 Testing and Validation

After completing the final development, thorough testing and validation will be performed to confirm that all system components function as intended. This includes unit testing where each hardware part such as the infrared transmitter, receiver, and Raspberry Pi GPIO pins will be individually tested to detect any potential malfunctions and functional testing, which will verify whether the system reliably controls the connected appliances. Additional checks, like measuring response time and validating infrared signal accuracy, will ensure the system meets user expectations and is suitable for real-world deployment.

3.1.8 Deployment

In this final stage, the Universal Infrared System was integrated into a real-world environment to verify its usability and effectiveness. The hardware components were arranged in their designated positions to ensure optimal signal coverage. The GUI was also deployed as the primary user interaction platform, providing intuitive access to control multiple appliances. Besides, the system was operated under typical conditions, such as controlling fan, light and air conditioner in a room environment. This allowed assessment of the system's reliability in everyday scenarios, confirming that recorded signals were transmitted accurately and appliances responded consistently. The deployment step also provided valuable feedback regarding user experience, system responsiveness and areas for potential improvement ensuring that the Universal Infrared System is ready for practical usage

3.2 System Architecture Diagram

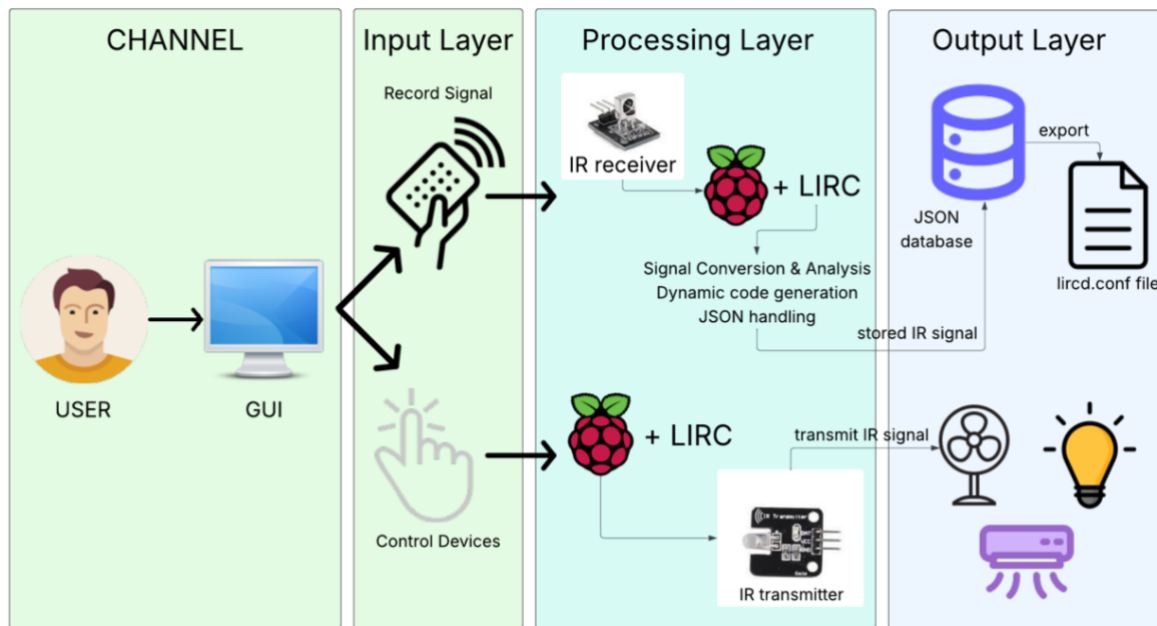


Figure 3.4 System Architecture diagram of Universal Infrared System

The system architecture diagram shown in Figure 3.2.1 is organized into four main layers, Channel, Input, Processing and Output, following the Input-Processing-Output (IPO) model.

1. Channel Layer

The user interacts with the system through GUI. This serves as the entry point, allowing the user to either record new IR signals or control existing appliances.

2. Input Layer

- **Signal Recording:** IR signals from appliance remotes are captured using an IR receiver module.
- **Control Devices:** User selects appliance functions directly from the GUI, which are then mapped to corresponding IR commands.

3. Processing Layer

- For recording, RPi interprets and formats the IR signals into digital codes, which are stored for later use.
- For controlling, RPi retrieves or dynamically generates the appropriate IR codes and prepare them for transmission.

4. Output Layer

The processed data produces two main outputs in different functions:

- Storage output: The recorded signals are stored in a JSON database and can be exported to a lircd.conf file for transmission purpose using LIRC
- Device Control Output: The IR transmitter modules send the formatted signals to the target appliances, enabling remote control operations

3.3 Use Case Diagram and Descriptions

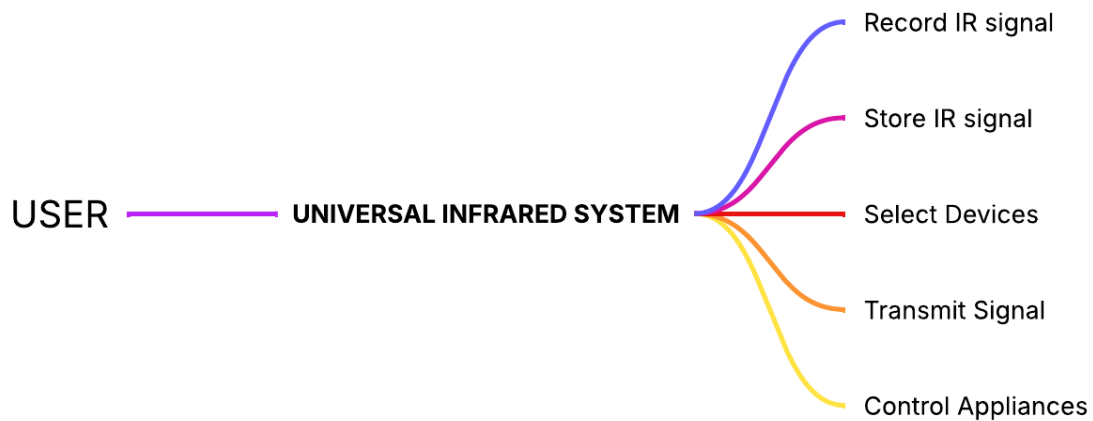


Figure 3.5 Use Case Diagram

The use case diagram above illustrates how user interacts with the Universal Infrared System. The main actor is the USER, who perform several functions through the system, as summarized below:

1. Record IR signal

- Actor: User
- Description: User presses a button on a physical remote, then the system will capture the corresponding infrared signal through the IR receiver.
- Purpose: To acquire the raw IR signal for later use in controlling the specific appliances.

2. Store IR Signal

- Actor: User
- Description: After recording the signal, the system stores the IR signal into a JSON database.
- Purpose: To allow efficient retrieval and mapping of signals to appliance functions.

3. Select Devices

- Actor: User
- Description: User may select the target appliances such as fan, lights, humidifier or air conditioner through the GUI
- Purpose: To specify which device should be controlled by the transmitted IR signal

4. Transmit Signal

- Actor: User
- Description: The user press on the button of the specific functions of the respective appliances, prompting the system to retrieve the corresponding IR signal from the database and transmit it via the IR transmitters.
- Purpose: To send control commands to the selected appliances.

5. Control Appliances

- Actor: User
- Description: The appliance receives the transmitted IR signal and performs the corresponding actions, such as power on or off, changing modes, or adjusting its speed or temperature.
- Purpose: To provide user with functional control of multiples home appliances using a signal universal system.

3.4 Activity Diagram

The activity diagram explains the workflow of the Universal Infrared System. The process can be divided into two main flows, signal recording and signal transmission.

- Signal Recording

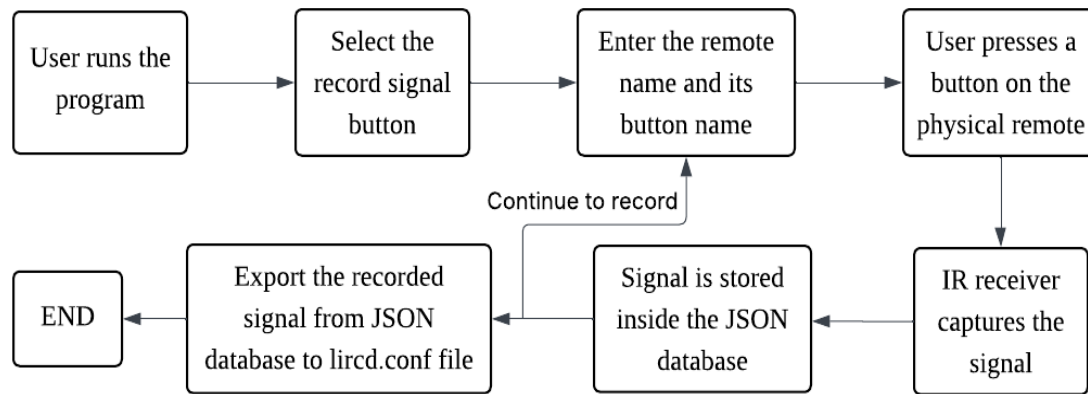


Figure 3.6 Activity Diagram of signal recording

- Signal Transmission

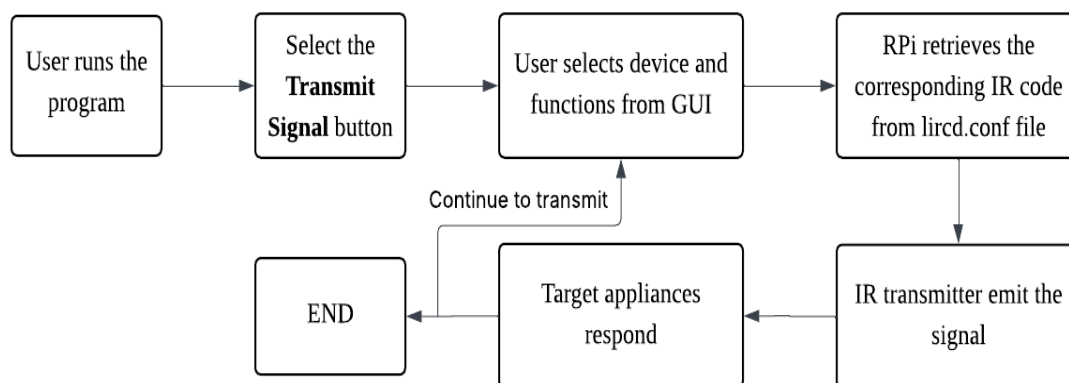


Figure 3.7 Activity Diagram of signal transmission

Chapter 4

System Design

4.1 System Block Diagram

4.1.1 Top-down system design

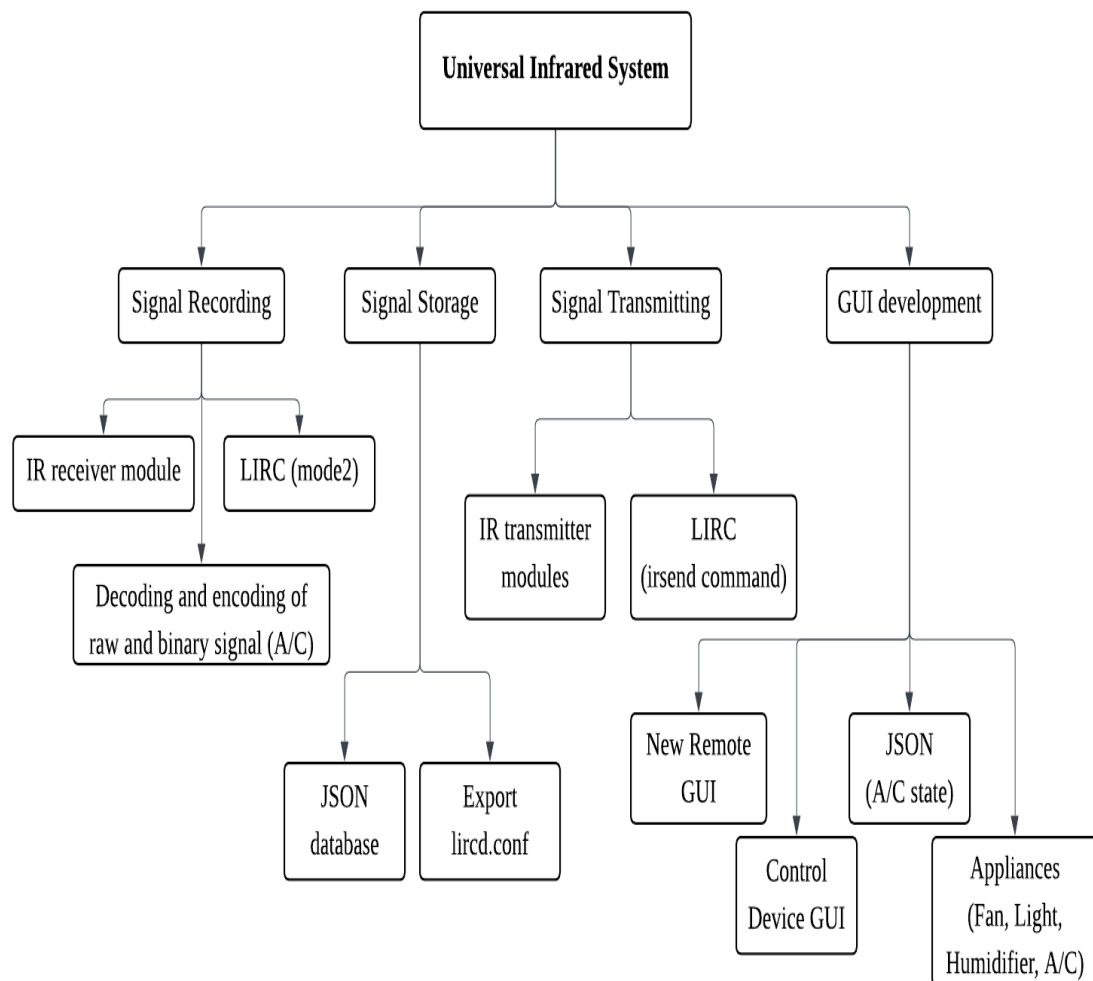


Figure 4.1 Top-down system design diagram

The top-down system design diagram of Universal Infrared System illustrates how the entire system is structured into four main functional modules, Signal Recording, Signal Storage, Signal Transmission and GUI Development. Each of these modules is further broken down into submodules that specify their internal functions.

- **Signal Recording Module**

This module is responsible for capturing infrared signals from different remotes. The IR receiver module detects incoming IR signals and forwards them to the RPi through GPIO18. For the A/C, the LIRC library decodes the raw input, while additional processing converts the captured signals into raw or binary representations

- **Signal Storage Module**

Once signals are recorded, they are stored in a structured format for later use. A JSON database is used to maintain the recorded signals, allowing flexibility in storing multiple remotes and functions. Users also have the option to export signals into lircd.conf file, which is required by LIRC for transmission.

- **Signal Transmission Module**

This module handles the process of sending infrared signals to control appliances. The signals are transmitted via the multiple IR transmitter modules connected to GPIO17m enabling control of appliances.

- **GUI Development Module**

The GUI developed using CustomTkinter, provides user-friendly control of the system. It consists of two main options which is “New Remote” option that allows recording new signals or exporting them to lircd.conf file, and the second options is “Control Devices” option which provides remote-like interface for controlling appliances. Additionally, the GUI supports state persistence for the A/c, which means that the last known configuration of the A/C is saved and restored when the program is restarted.

4.1.2 Flowchart of universal remote

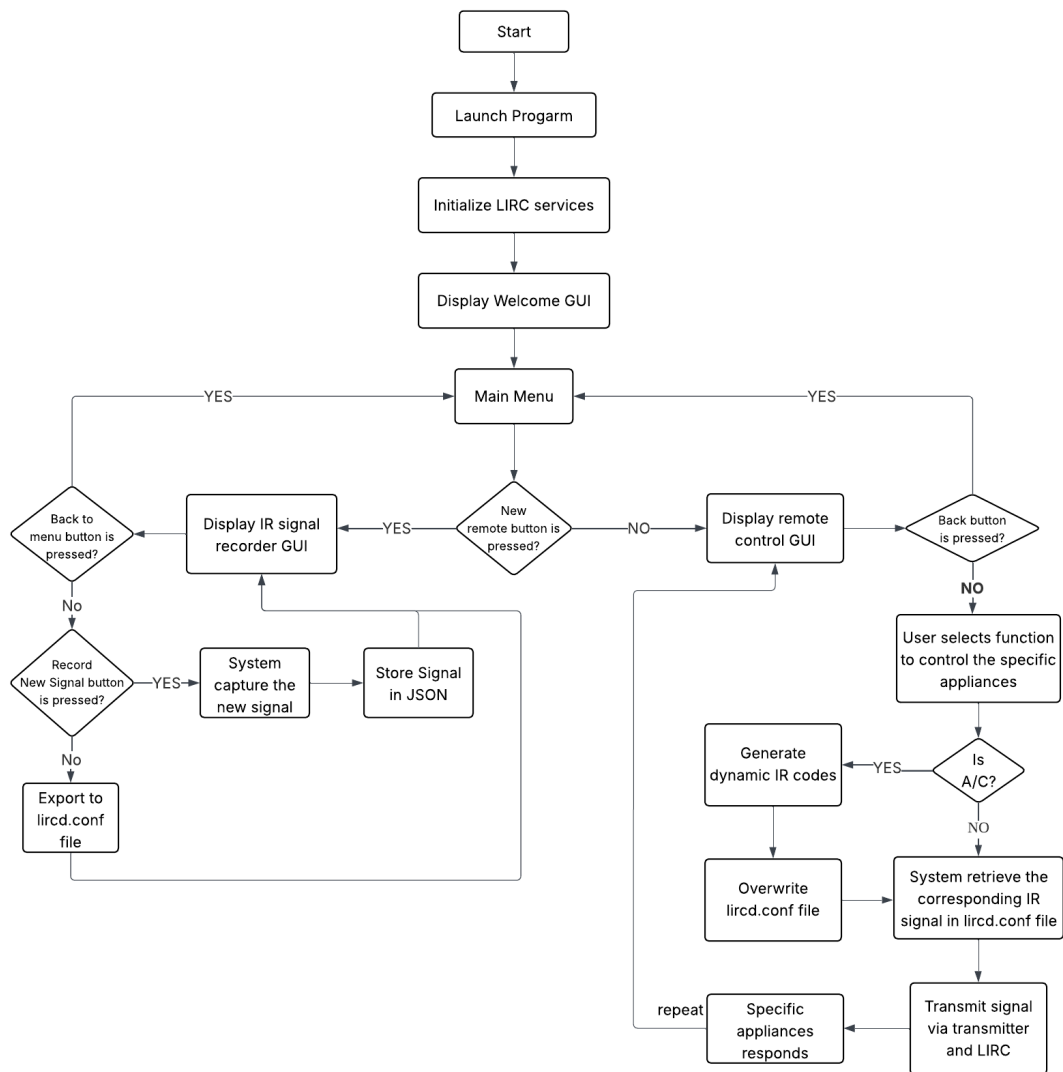


Figure 4.2 Flowchart of Universal Infrared System

The system flowchart shown in Figure 4.1.2.1 illustrates the overall workflow of the universal infrared remote control system. It begins with initializing the LIRC services, followed by displaying the welcome screen and main menu. From the main menu, user can either record new signals or control existing appliances. In the recording branch, the captured signals are first stored in a JSON database and may be exported to the `lircd.conf` file for transmission. In the control branch, the system retrieves the appropriate IR code from the `lircd.conf` file and transmit it through the transmitters. A special case was the A/C, where dynamic code generation was required, thus, the system modifies the `lircd.conf` file before transmission. Finally, the process can be repeated or navigated back to main menu, ensuring smooth user interaction.

4.2 System Components Specifications

4.2.1 Hardware

- Raspberry Pi 4 Model B

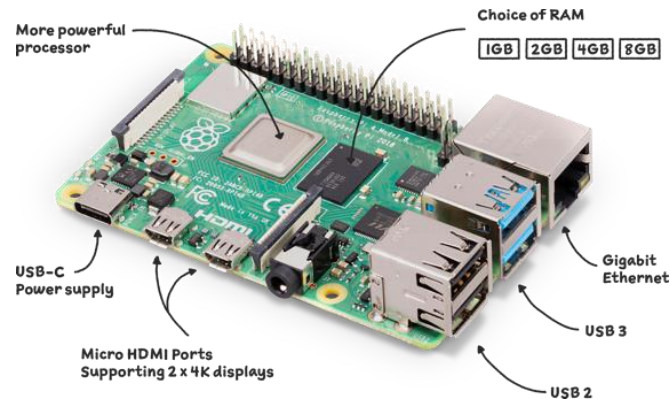


Figure 4.3 RPi 4 Model B

In this project, the central component is the Raspberry Pi 4 Model B, an ARM-based single-board computer that supports various Linux distributions, including Raspberry Pi OS. The Raspberry Pi 4 Model B features a 64-bit quad-core processor running at 1.8GHz, dual-band 2.4GHz and 5GHz wireless LAN, gigabit Ethernet, Bluetooth 5.0, 2 USB 3.0 ports, 2 USB 2.0 ports, and is available in 2GB, 4GB, or 8GB SDRAM variants. It also has 40 GPIO pins and supports Power over Ethernet (PoE) when paired with an additional PoE HAT. [15]

- IR Digital Receiver Module
- 10 IR Digital Transmitter modules
- Laptop
- 2 Breadboards
- Wire cutter and stripper
- Jumper wires
- IR remote controls
 1. Fan remote
 2. Light remote
 3. Humidifier remote
 4. Air conditioner remote

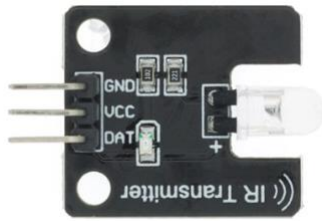


Figure 4.4 IR Digital Transmitter Module



Figure 4.5 IR Digital Receiver Module

4.2.2 Software

- LIRC libraries

LIRC is a software suite used to decode and send infrared signals from various standard remote controls. It offers enhanced flexibility, making it suitable for numerous applications. Its main component, the lircd daemon, interprets signals received by the device driver and communicates this data through a socket. It can also transmit IR signals when supported by the hardware. LIRC's user-space tools allow a computer to be operated via a remote control. [16]

- Python (Geany)

Python is a widely used programming language developed by Guido van Rossum and released in 1991. It is valued for its simplicity and versatility, making it ideal for both rapid prototyping and full-scale application development. Python works across multiple platforms, including Windows, Linux, and Raspberry Pi. As an interpreted language, it allows immediate execution of code, which accelerates the development process. [17]

- VNC Viewer

RealVNC Server allows you to remotely access the graphical desktop of your Raspberry Pi. Even if your Pi is headless (not connected to a monitor) or doesn't have a graphical desktop running, RealVNC Server can still provide remote graphical access by using a virtual desktop. [18]



Figure 4.6 VNC viewer

- JSON database

JSON is a lightweight and human-readable data format widely used for data storage and exchange. It's much more flexible compared to the row-columns format, which is fixed and expensive when it comes to implementing even small schema changes [19]. In this project, JSON is used as the main database to store the raw IR codes captured by the IR receiver. Each recorded IR code is saved with a unique key.

- CustomTkinter

CustomTkinter is an enhanced and modernized version of the standard Tkinter library in Python, designed to build more visually appealing and user-friendly graphical user interfaces (GUI). Unlike the default Tkinter, which has a very basic and outdated appearance, CustomTkinter provides a modern design with features such as dark mode, rounded buttons, sliders, and customizable widgets [20].

- Tinkercad

Tinkercad is an application that allows user to design 3D models and create basic CAD projects. For this project, Tinkercad was used to design an enclosure for the RPi and its associated components, such as breadboards, receiver and transmitter modules.

4.3 Circuits and Components Design

The circuit design for the Universal Infrared System integrates the infrared receiver and multiple infrared transmitters with the RPi 4 Model B. The objective of the design is to ensure accurate reception of infrared signals from remote controls and strong transmission to appliances in different directions.

4.3.1 Infrared Receiver Circuit

The HX-1838 infrared receiver module was used to capture incoming signals. The receiver operates at a carrier frequency of 38 kHz, which is compatible with most household appliance remotes. The module has three pins:

- VCC (5V) – connected to the Raspberry Pi 5V power rail.
- GND – connected to the Raspberry Pi ground.
- OUT (Signal) – connected to GPIO 18 on the Raspberry Pi.

This configuration allows the Raspberry Pi to detect the modulated IR signal and process it through Python scripts.

4.3.2 Infrared Transmitter Circuit

To ensure strong and wide-area coverage of transmitted signals, multiple IR LED transmitters were placed around the breadboard. Each IR LED module includes a driver circuit that ensures stable operation.

- VCC (5V) – connected to the positive power rail.
- GND – connected to the negative rail.
- Signal pin – connected to GPIO 17 of the Raspberry Pi

4.3.3 Breadboard Layout

The breadboard prototype was designed to allow easy connection and testing:

- The HX-1838 receiver is placed near the middle for stable input capture.
- 10 IR LED transmitters are arranged around the edges of the breadboard in different orientations. This layout maximizes the coverage angle, allowing the system to control appliances located in various positions within a room.

4.4 System Components Interaction Operations

The Universal Infrared System is designed to provide seamless interaction between hardware and software components in order to capture, store, and retransmit infrared (IR) signals. The interaction process can be divided into two main operations, which is signal reception and signal transmission.

4.4.1 Signal Reception (Learning Mode)

1. Remote Button Press – When the user presses a button on a physical remote control for example fan remote or light remote, the remote emits a modulated IR signal at a frequency of approximately 38 kHz.
2. Signal Detection – The HX-1838 IR receiver module captures the incoming IR light signal and demodulates it into a digital waveform.
3. Data Transfer to Raspberry Pi – The digital output from the receiver is sent through GPIO 18 of the Raspberry Pi.
4. Processing and Storage – A Python script running on the Raspberry Pi reads the signal data and stores it in a JSON database with a unique identifier for example, FAN_POWER_ON or FAN_SPEED_UP

This process allows the system to “learn”, remember the IR codes from different remotes and store them inside a database in order to retrieve it easily during signal transmission.

4.4.2 Signal Transmission (Control Mode)

1. User Input via GUI – The user interacts with the CustomTkinter GUI on the Raspberry Pi by clicking a virtual button that corresponds to a specific appliance function button.
2. Code Retrieval – The system retrieves the appropriate IR code from the JSON database based on the selected button.
3. Signal Output – The retrieved IR code is transmitted through GPIO 17 of the Raspberry Pi.
4. Infrared Transmission – Multiple IR LED transmitters emit the infrared light simultaneously in different directions to cover a wider area in order to control the respective appliances.
5. Appliance Response – The target appliance receives the IR signal and executes the corresponding action.

Chapter 5

System Implementation

5.1 Software setup

5.1.1 RPi OS

First and foremost, RPi OS was installed on a microSD card using the RPi Imager tool. It provided the base operating environment for Python and other required libraries.

5.1.2 VNC Viewer

Then, installation of the VNC viewer was performed to perform remote access with RPi in computer. Before connecting RPi to our computer, it was necessary to enable VNC on RPi by entering the command “sudo raspi config” at the terminal. Then, by entering “hostname -I” in the terminal, the IP address of the RPi is obtained. The IP address is copied and pasted at the VNC viewer to connect.

```
pi@raspberrypi:~ $ hostname -I
192.168.0.132 2001:e68:543a:8681:d35a:a13d:c810:2f72
```

Figure 5.1 IP address of the RPi

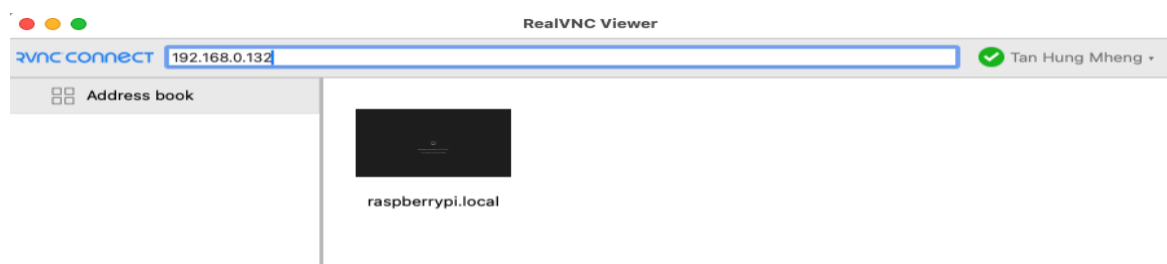


Figure 5.2 Paste the IP address and connect to the VNC viewer

5.1.3 CustomTkinter

Since CustomTkinter were used to create GUI in our project, it had to be installed first before using it. The command “pip install customtkinter” was executed in the terminal.

```
pi@raspberrypi:~/fyp $ pip install customtkinter
```

Figure 5.3 Command to install customtkinter

5.2 Hardware setup

After the software setup was completed, the next step was hardware implementation. The hardware setup involved assembling the RPi 4 Model B with the HX-1838 IR receiver module and multiple IR LED transmitter modules. The IR receiver was connected to GPIO 18 for signal input, while the IR transmitters were connected to GPIO 17. The transmitters were arranged around the breadboard to increase coverage and ensure reliable transmission to appliances in different directions. Besides, the wire had been stripped nicely to avoid distraction during signal transmission.

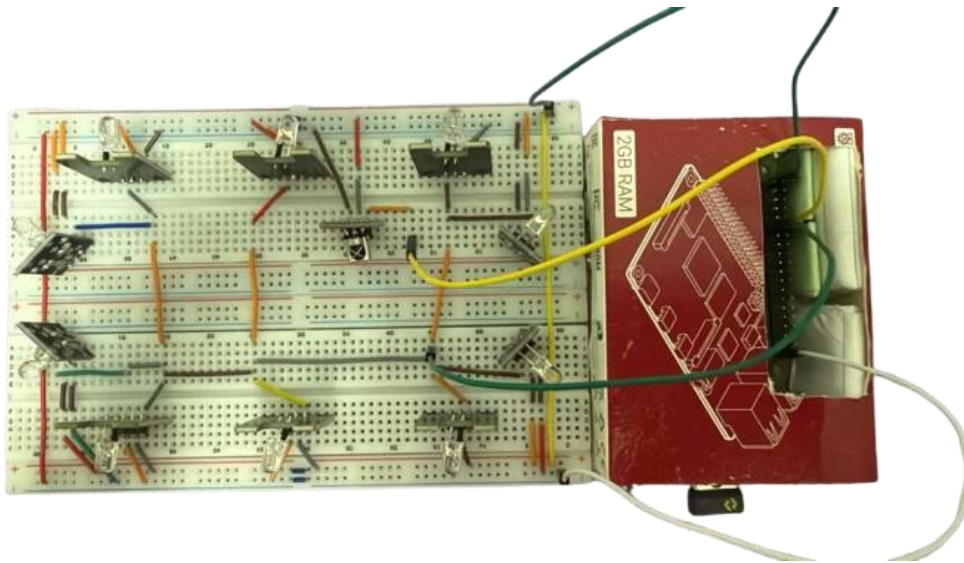


Figure 5.4 Connection of the hardware



Figure 5.5 The enclosure designed for the hardware components using Tinkercad

5.3 Setting and Configuration

5.3.1 LIRC Installation and Configuration on RPi

Before the system could operate, several configurations were required on the RPi to enable IR signal recording and transmission. The setup was carried out using LIRC library and the following steps below summarized the configuration process [21-24].

1. System Update and Library Installation

After setting up our RPi, the system was updated and LIRC package was installed using the following command:

```
$ sudo apt-get update
```

```
$ sudo apt-get install lirc
```

2. Editing LIRC configuration file

The LIRC configuration file was edited to specify the driver and device used by entering the following command:

```
$ sudo nano /etc/lirc/lirc_options.conf
```

Then, the following lines were modified:

```
...
```

```
driver = default
```

```
device = /dev/lirc0
```

```
...
```

3. Enabling GPIO pins for IR recording and transmission

To enable the IR receiver and transmitter modules, the configuration file located at /boot/firmware/config.txt was edited by executing the command below:

```
$ sudo nano /etc/lirc/lirc_options.conf
```

Then, the following lines were uncommented or added:

```
dtoverlay = gpio-ir,gpio_pin=18
```

```
dtoverlay = gpio-ir-tx,gpio_pin=17
```

Note: Manually key in both lines if they are not inside the file initially.

4. Restarting and verifying LIRC services

The LIRC services was restarted to apply changes, then followed by a status check:

```
$ sudo systemctl stop lircd
```

```
$ sudo systemctl start lircd
```

```
$ sudo systemctl status lircd
```

A system reboot was then executed to ensure changes were properly applied.

```
$ sudo reboot
```

Note: It was important to reboot the system every time after editing the configuration file of RPi.

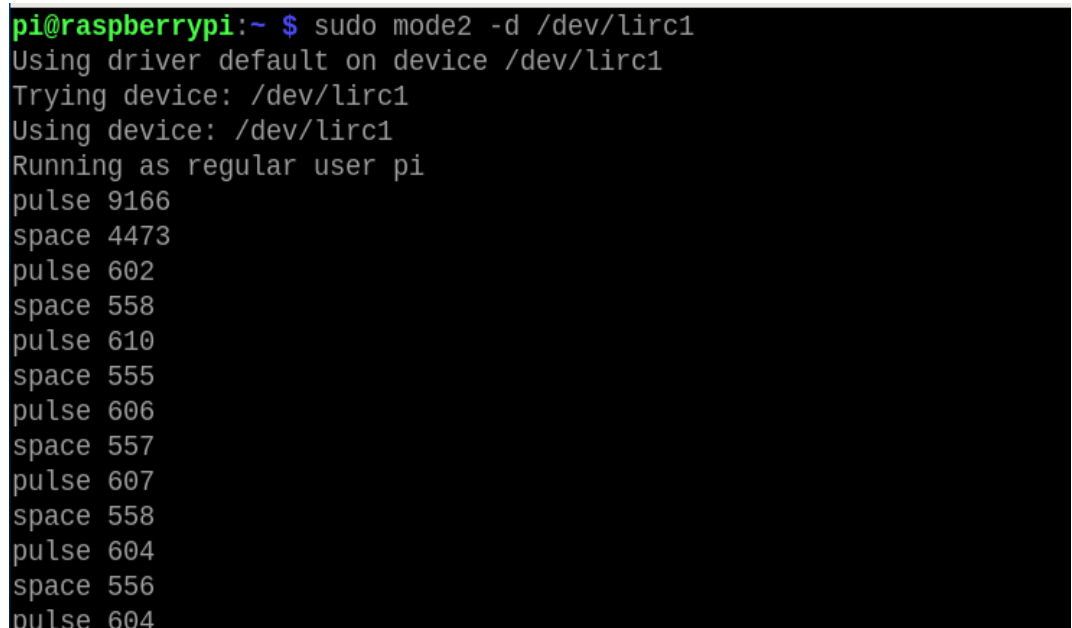
5. Testing the IR receiver

After rebooting, the IR receiver was tested by temporarily stopping the LIRC services and the mode2 command was executed.

```
$ sudo systemctl stop lircd
```

```
$ sudo mode2 -d /dev/lirc1
```

When a button on the remote control was pressed, the RPi successfully detected the incoming IR signal and raw pulse/space patterns will be displayed on the terminal as shown as figure below.



```
pi@raspberrypi:~ $ sudo mode2 -d /dev/lirc1
Using driver default on device /dev/lirc1
Trying device: /dev/lirc1
Using device: /dev/lirc1
Running as regular user pi
pulse 9166
space 4473
pulse 602
space 558
pulse 610
space 555
pulse 606
space 557
pulse 607
space 558
pulse 604
space 556
pulse 604
```

Figure 5.6 Output of mode2 command when button is pressed

5.3.2 IR signals recording and transmission using terminal

1. Once the receiver was confirmed to function correctly, the next step was to record the key buttons of the remote control through the terminal by following the on-screen instructions until a `lircd.conf` file is successfully created by using the following command:

```
$ sudo irrecord --disable-namespace -f -d /dev/lirc1 ~/lircd.conf
```

Note: -- disable-namespace flag allows more flexibility in button naming

2. After recorded successfully, a `lircd.conf` file was generated. Once it had been created, the file was renamed and moved to the specific folder used by LIRC by executing the following commands:

```
$ sudo mv /etc/lirc/lircd.conf /etc/lirc/lircd_original.conf
```

```
$ sudo cp <name of your config file> /etc/lirc/lircd.conf
```

3. Then, the `lircd` service was restarted and the recorded signals in the configuration file were verified by sending the signal using the following commands:

```
$ sudo systemctl restart lircd
```

```
$ sudo systemctl status lircd
```

```
$ irsend SEND_ONCE <remote_name> <command_name>
```

```
● lircd.service - Flexible IR remote input/output application support
   Loaded: loaded (/lib/systemd/system/lircd.service; enabled; preset: enabled)
   Active: active (running) since Wed 2025-04-30 02:31:35 +08; 18h ago
 TriggeredBy: ● lircd.socket
     Docs: man:lircd(8)
           http://lirc.org/html/configure.html
    Main PID: 762 (lircd)
      Tasks: 2 (limit: 1574)
         CPU: 69ms
    CGroup: /system.slice/lircd.service
            └─762 /usr/sbin/lircd --nodaemon

Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Notice: Driver API version: 3
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Notice: Driver version: 0.10.0
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Notice: Driver info: See file:///usr/sh
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Info: lircd: Opening log, level: Info
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Notice: Using systemd-ru
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Warning: Running as root
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Info: Using remote: fan_remote.
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Info: Using remote: light_remote
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Info: Using remote: humidifier_remote.
Apr 30 02:31:35 raspberrypi lircd-0.10.1[762]: Notice: lircd(default) ready, using /var>
```

Figure 5.7 Status of `lircd` when executing “`sudo systemctl status lircd`”

From the figure shown above, it was necessary to check whether the LIRC service were online and the remote name was recognized by LIRC before controlling the appliances. The name of the remote had to match the one recorded during the configuration process. The figure below shown the lircd.conf file of the remote control after being recorded down. Since our project were developed a Universal Infrared System, therefore, three of the lircd.conf file of the respective remote were merged together into a lircd.conf file instead of separating them into 3 files.

```
begin remote

    name fan_remote
    flags RAW_CODES|CONST_LENGTH
    eps      30
    aeps     100

    gap      108936

    begin raw_codes
        name KEY_FAN_POWER
        9155      4525      633      536      606      537
        607      536      608      536      604      539
        606      538      605      539      604      538
        604      1629      635      1626      632      1631
        632      1625      631      1631      631      1625
        631      1630      629      1633      628      1627
        635      534      605      539      607      537
        607      536      604      541      606      560
        581      543      605      540      598      1630
        630      1634      624      1634      631      1629
        628      1638      621      1630      632      1635
        | 624
```

Figure 5.8 Lircd.conf file of fan remote

```

begin remote

  name  light_remote
  flags RAW_CODES|CONST_LENGTH
  eps   30
  aeps  100

  gap   108869

  begin raw_codes

    name KEY_LIGHT_POWER
    9122  4524  631  537  602  539
    603   538  603  538  604  536
    604   538  603  538  604  535
    605  1626  629  1625  631  1627
    629  1625  630  537  603  1629
    629  1627  630  1625  631  1627
    627  1628  627  539  604  537
    605   536  602  538  603  538
    600   541  603  538  601  540
    600  1630  626  1630  627  1627
    621  1627  629  1627  629  1627

```

Figure 5.9 Lircd.conf file of light remote

```

begin remote

  name  humidifier_remote
  flags RAW_CODES|CONST_LENGTH
  eps   30
  aeps  100

  gap   107884

  begin raw_codes

    name KEY_HUMI_POWER
    9159  4479  633  558  606  559
    605   559  607  559  606  559
    607   559  607  559  606  559
    605  1609  633  1611  632  1610
    631  1610  631  1611  630  1611
    629  1612  632  1610  631  559
    603   538  626  559  605  558
    604   560  604  560  604  559
    603   560  604  1611  630  1611
    630  1610  630  1612  630  1611
    631  1611  631  1611  629  1612
    631

```

Figure 5.10 Lircd.conf file of humidifier remote

5.3.3 JSON Database Configuration

In this project, JSON database was primarily used as a storage medium for IR signals during the recording phase. Whenever a new IR signal was captured using LIRC, it was first saved into the JSON file in order to maintain an organized and human-readable record of all the signals of the respective remote. This allowed easier management, updating and verification of the captured codes before they were used for transmission. The JSON database followed a simple key-value structure where each recorded function was assigned a descriptive name as the key, while the corresponding value contained the raw pulse/space timings representing the IR signal code.

```
def save_ir_codes(ir_db):
    with open(DB_FILE, "w") as f:
        f.write("{\n")
        for dev_i, (device, cmds) in enumerate(ir_db.items()):
            f.write(f'    "{device}": {{\n')
            for cmd_i, (cmd, signals) in enumerate(cmds.items()):
                f.write(f'        "{cmd}": [\n')

                for i in range(0, len(signals), 6):
                    chunk = signals[i:i+6]
                    line = "            " + ", ".join(str(x) for x in chunk)
                    if i + 6 < len(signals):
                        line += ",\n"
                    f.write(line + "\n")
                f.write("        ]\n")
            if cmd_i < len(cmds) - 1:
                f.write(",\n")
            f.write("    }\n")
        if dev_i < len(ir_db) - 1:
            f.write(",\n")
        f.write("}\n")
```

Figure 5.11 Code snippets of saving the IR signals into JSON database

Once the required signals were collected, the entries stored inside JSON file could be exported and reformatted into lircd.conf file and signal transmission was ready. This approach provided flexibility as JSON database served as an intermediate database where signals could be continuously recorded and updated. At the same time, compatibility with the LIRC library was preserved by exporting the finalized codes into the lircd.conf format which required for signal transmission.

```

def json_to_lircd_conf(json_file, device_name, output_file):
    try:
        with open(json_file, "r") as f:
            ir_db = json.load(f)
    except Exception as e:
        print(f"Error reading {json_file}: {e}")
        return

    if device_name not in ir_db:
        print(f"Device '{device_name}' not found in database")
        return

    conf_lines = [
        "begin remote\n",
        f"    name {device_name}\n",
        "    flags RAW_CODES|CONST_LENGTH",
        "    eps      30",
        "    aeps      100\n",
        "    gap       108936",
        ""
    ]

    conf_lines.append("        begin raw_codes")

    for cmd_name, signals in ir_db[device_name].items():
        conf_lines.append(f"\n            name {cmd_name}")

        cleaned = []
        for val in signals:
            try:
                cleaned.append(int(val))
            except ValueError:
                continue

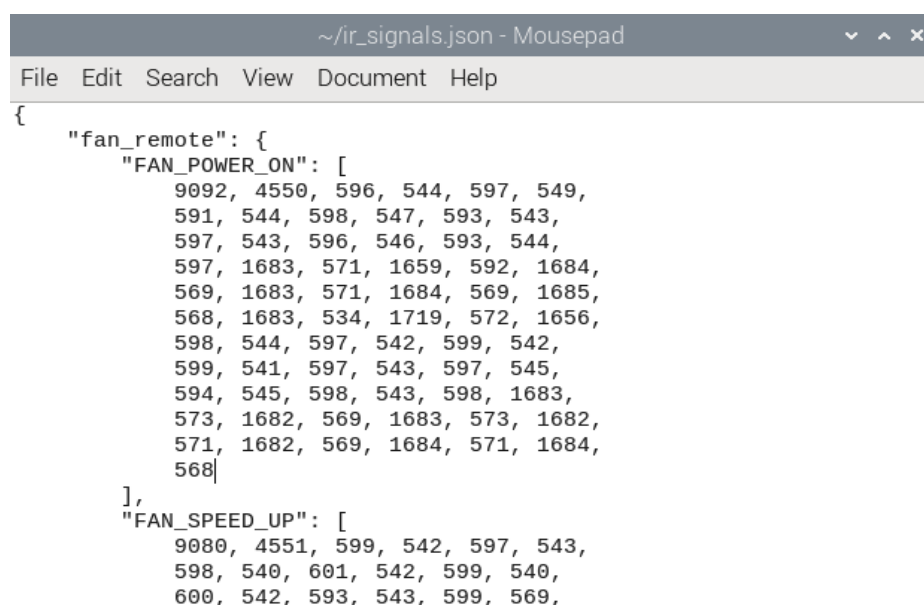
        # Write in chunks of 6 for neat formatting
        for i in range(0, len(cleaned), 6):
            chunk = cleaned[i:i+6]
            line = "        " + " ".join(f"{num:5d}" for num in chunk)
            conf_lines.append(line)

    conf_lines.append("\n        end raw_codes\n")
    conf_lines.append("end remote")

    try:
        with open(output_file, "w") as f:
            f.write("\n".join(conf_lines))
        print(f"lircd.conf written to {output_file}")
    except Exception as e:
        print(f"Error writing {output_file}: {e}")

```

Figure 5.12 Code snippets of exporting signal from JSON to lircd.conf file



```

{
  "fan_remote": {
    "FAN_POWER_ON": [
      9092, 4550, 596, 544, 597, 549,
      591, 544, 598, 547, 593, 543,
      597, 543, 596, 546, 593, 544,
      597, 1683, 571, 1659, 592, 1684,
      569, 1683, 571, 1684, 569, 1685,
      568, 1683, 534, 1719, 572, 1656,
      598, 544, 597, 542, 599, 542,
      599, 541, 597, 543, 597, 545,
      594, 545, 598, 543, 598, 1683,
      573, 1682, 569, 1683, 573, 1682,
      571, 1682, 569, 1684, 571, 1684,
      568
    ],
    "FAN_SPEED_UP": [
      9080, 4551, 599, 542, 597, 543,
      598, 540, 601, 542, 599, 540,
      600, 542, 593, 543, 599, 569,

```

Figure 5.13 Example of IR signal stored in JSON

5.3.4 Reverse Engineering of A/C Remote Control Signals

In contrast to the simple appliances such as fans, lights and humidifier shown above, A/C transmitted a longer and more complex infrared signal each time a button was pressed. Instead of sending only the specific IR signal for a single corresponding function such as adjusting the speed or changing the mode, the A/C remote sent the entire current state of the A/C shown at the GUI of the remote, which includes parameters such as power status, operating modes, temperature and fan speed. For example, when user changes the temperature, only the temperature bits were updated, however, the signal transmitted still includes power, mode, fan speed and swing state. Therefore, a direct button-to-button recording method shown previously using command “sudo irrecord” was not suitable and insufficient. In order to integrate the A/C into the Universal Infrared system, a reverse engineering approach was applied to capture and analyse the IR signals transmitted by the original remote control [25-26]. In this project, the brand of A/C used was Carrier. The processes are shown below.

1. Signal Capture

The Carrier A/C remote control signals were initially captured using the command: `mode2 -m -d /dev/lirc1`. This allowed the RPi to record the raw pulse and space sequences generated during each button press. The captured data consisted of long sequences of varying pulse and space durations, which represented the modulated IR signal. An example of the recorded raw data is shown in figure below. (Note: Ignore the first large number!)

```
pi@raspberrypi:~/fyp $ mode2 -m -d /dev/lirc1
Using driver default on device /dev/lirc1
Trying device: /dev/lirc1
Using device: /dev/lirc1
16777215
```

4409	4369	525	1625	524	547
527	1625	526	1624	524	548
526	547	527	1621	528	575
498	550	523	1625	525	549
526	547	526	1623	527	1624
524	548	525	1626	523	547
605	1571	499	548	525	1622
526	1622	526	1650	498	1624
526	1623	526	1621	528	547
525	1631	517	548	527	547
525	549	525	545	532	543
525	1655	495	1622	523	548
525	1623	526	547	525	549

Figure 5.14 IR signal recorded when executing mode2 command

2. Signal Analysis

The raw pulse and space timing sequences obtained from the Carrier A/C remote were subsequently converted into binary representation and analyzed. This conversion was necessary to simplify the interpretation of the IR protocol and to identify the command structures corresponding to specific commands such as power control, temperature adjustment, fan speed selection, and operations mode. In this step, the binary patterns from different button presses were compared to find which parts of the code controlled the specific functions. For instance, as shown in the figure below, the temperature setting was linked to bits 32 to 35 in the sequence. This method made it easier to match sections of the code with A/C functions such as temperature, mode and fan speed.

```
def decode_ir_signal(durations):
    bits = []
    i = 2 # skip header pulse/space
    while i < len(durations) - 1:
        pulse = durations[i]
        space = durations[i + 1]
        if 400 <= pulse <= 700:
            if 200 <= space <= 900:
                bits.append('0')
            elif 1000 <= space <= 1800:
                bits.append('1')
            else:
                bits.append('?')
        else:
            bits.append('?')

        i += 2
    return ''.join(bits)
```

Figure 5.15 Code snippets of decoding IR signal into binary representation

Power	°C	Mode	fanSpeed	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
OFF	25	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	25	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	24	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
OFF	24	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	30	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	29	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	28	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	27	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	26	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	25	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	24	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	25	DRY	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	23	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	22	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	21	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	20	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	19	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	18	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	17	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
MODE		auto and dry no fan speed	fan no temperature																																
ON	17	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	17	FAN	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	17	DRY	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	17	AUTO	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
FANSPEED																																			
ON	17	COOL	MED	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	17	COOL	LOW	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	17	COOL	HIGH	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
ON	17	COOL	AUTO	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0

Figure 5.16 Analyse the pattern of the A/C signal using Excel

3. Function Mapping

After analyzed the captured binary patterns, the corresponding functions of the Carrier A/C were identified and mapped to its respective IR code. Each binary sequence represented the entire state of the A/C, incorporating parameters such as power status, temperature settings, operating modes and fan speed. The result is shown as table below.

Table 5.1: Function mapping of A/C

Field	Bit position	Remark
Header	0 - 7	Fixed Protocol
POWER	18, 21, 35	POWER OFF → bit 18 = 1, bit 21 = 0, bit 35 = 1 POWER ON → bit 18 = 0, bit 21 = 1, bit 35 = 0
TEMP	32 - 35	0000(17°C), 0001(18°C), 0011(19°C), 0010(20°C), 0110(21°C), 0111(22°C), 0101(23°C), 0100(24°), 1100(25°C), 1101(26°C), 1001(27°C), 1000(28), 1010(29°C), 1011(30°C)
MODE	36 - 39	0000(COOL), 0100(DRY), 1000(AUTO), IF MODE IS FAN, BITS 32-39 = 11100100
SPEED	16 - 18	010(MED), 100(LOW), 001(HIGH), 101(AUTO), 000(If MODE is DRY or AUTO)
SWING	Whole byte	Fixed IR signal → 101100100100110100001111111100001110000000011111
Autoswing	Whole byte	Fixed IR signal → 1011001001001101010101100101001110000000011111
Checksum	8 - 15, 24 - 31, 40 - 47	Checksum byte computed as the bitwise XOR of the preceding 8-bit data fields.

5.3.5 Dynamic Code Generation and Transmission of A/C remote

After the Carrier A/C had been successfully reverse engineered, the next step was to implement the method of dynamically generating and transmitting the IR packets. Unlike simple devices where a single fixed IR code corresponds to each function, the A/C required a complete state packet to be transmitted every time.

To address this requirement, the program was designed to begin with a default binary sequence, representing an initial valid configuration of the A/C. Each of the mapped bit positions was then linked to the corresponding functions in GUI. For example, the temperature bits were assigned to temperature controls button, while the mode and fan speed bits were connected to their respective selection options. Whenever, the user changed a setting through the GUI for example adjusting the temperature or modifying the fan speed, the system executed a function that updated only the relevant bits of the binary sequence while leaving all other fields unchanged. This ensured that the updated binary packet always reflected the current complete state of the A/C.

```
base_binary = '10110010010011010101111101000000100000010111111'
```

Figure 5.17 Default base binary

```
def modify_ac_binary(base_binary, power=None, temp=None, fan_speed=None, mode=None):
    bits = list(base_binary)

    #power
    if power == 'OFF':
        bits[16:24] = list('01111011')
        off_byte1 = bits[16:24]
        bits[32:40] = list('11100000')
        off_byte2 = bits[32:40]

        off_checksum_1 = []
        for i in off_byte1:
            if i == '0':
                off_checksum_1.append('1')
            else:
                off_checksum_1.append('0')
        bits[24:32] = off_checksum_1

        off_checksum_2 = []
        for i in off_byte2:
            if i == '0':
                off_checksum_2.append('1')
            else:
                off_checksum_2.append('0')
```

Figure 5.18 Code snippets of modifying the base binary correspond to the specific functions

Then, the modified binary sequence was then converted back into pulse/space timing data and written into the `lircd.conf` file, which served as the configuration source for the LIRC library. By updating the configuration file, the system ensured that the new packet was recognized as a valid command. Finally, the RPi transmitted the signal through the IR transmitter modules using LIRC. This dynamic signal generation and transmission process guaranteed that the A/C consistently received a full-state signal, allowing it to interpret and execute the command correctly in accordance with its communication protocol.

```
def encode_binary(bits, pulse=520, space_0=550, space_1=1645):
    encoded = []
    for bit in bits:
        encoded.append(pulse)
        if bit == '0':
            encoded.append(space_0)
        else:
            encoded.append(space_1)
    return encoded

def binary_to_raw(binary_string, pulse=520, space_0=550, space_1=1645, header=(4350, 4400)):
    # First part: header + encoded binary
    first_part = list(header) + encode_binary(binary_string, pulse, space_0, space_1) + [pulse]

    # Middle part: large space
    middle_gap = [5200]

    # Final part: repeat header + 3 bits again
    repeat_bits = "101"
    second_part = list(header) + encode_binary(repeat_bits, pulse, space_0, space_1) + [pulse]

    # Combine all parts
    full_signal = first_part + middle_gap + second_part

    return full_signal
```

Figure 5.19 Code snippets of encoding binary back to raw signal

```
def write_lirc_config(raw_signal, remote_name="ac_remote", signal_name="DYNAMIC", filename="universal_remote1.lircd.conf"):
    # Build the remote config block
    new_remote_block = [
        "begin remote\n",
        f" name {remote_name}\n",
        " flags RAW_CODES|CONST_LENGTH",
        " eps 30",
        " aeps 100\n",
        " gap 108693",
        "",
        " begin raw_codes\n",
        f" name {signal_name}\n"
    ]

    for i in range(0, len(raw_signal), 6):
        chunk = raw_signal[i:i+6]
        line = " " + ' '.join(f"{val:5d}" for val in chunk)
        new_remote_block.append(line)

    new_remote_block += [
        "\n end raw_codes\n",
        "end remote\n"
    ]
```

Figure 5.20 Code snippets of writing the modified raw signal into `lircd.conf` file

5.4 System Operation

When the program is launched, user will be greeted with a user-friendly GUI displaying the message “WELCOME TO SMART HOME”, which was developed in Python using CustomTkinter, as shown in the figure below.

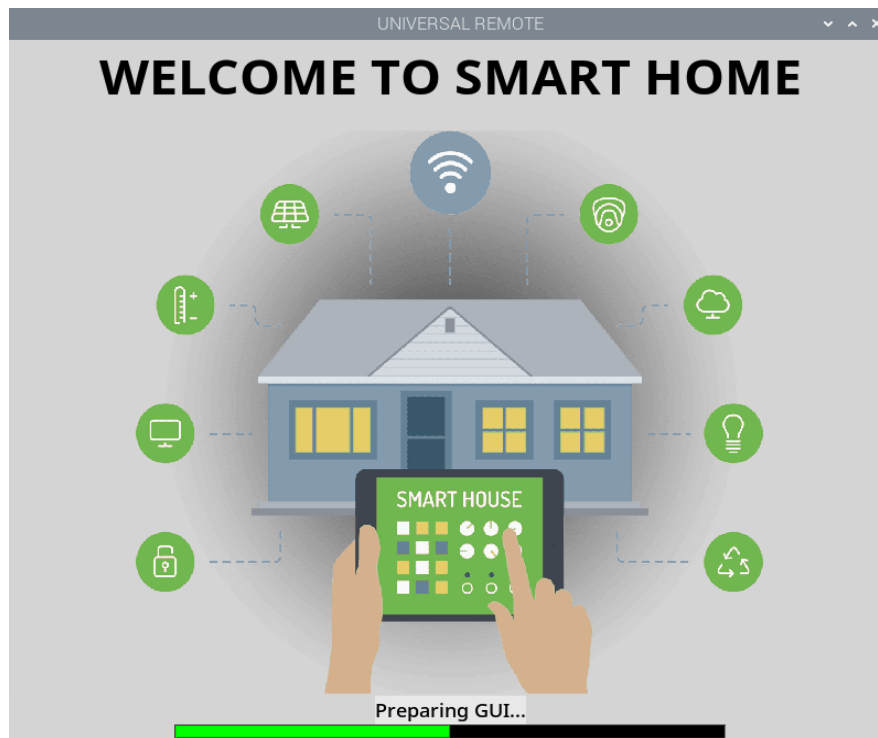


Figure 5.21 Welcoming GUI window was displayed when program was executed

After the welcome screen, the user is redirected to a window where they can choose either to record signals from a new remote (New Remote) or to control the available appliances (Control Devices).

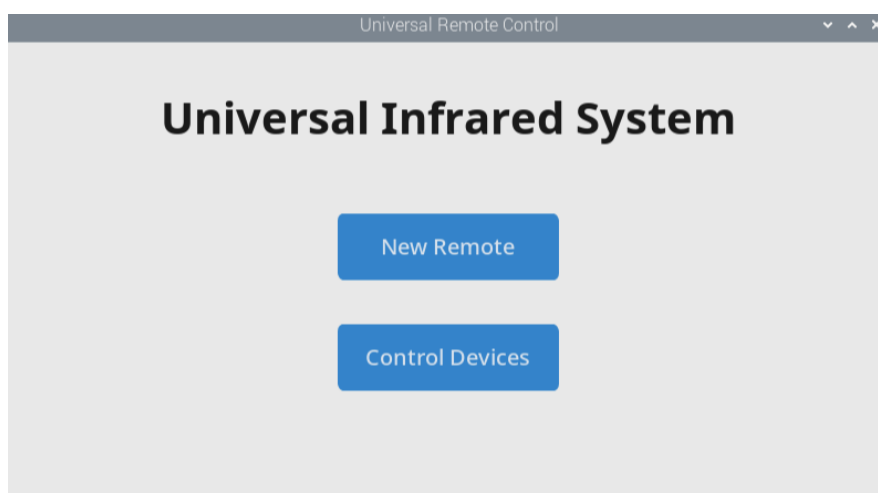


Figure 5.22 Window to let user to choose either record or transmit signal

When the “New Remote” button is pressed, the user is redirected to a new window where they can choose either to record new signal or to export the previously recorded signal to the lircd.conf file. A “Back to Menu” button is also designed to allow user to return back to the window shown in Figure 5.4.2.

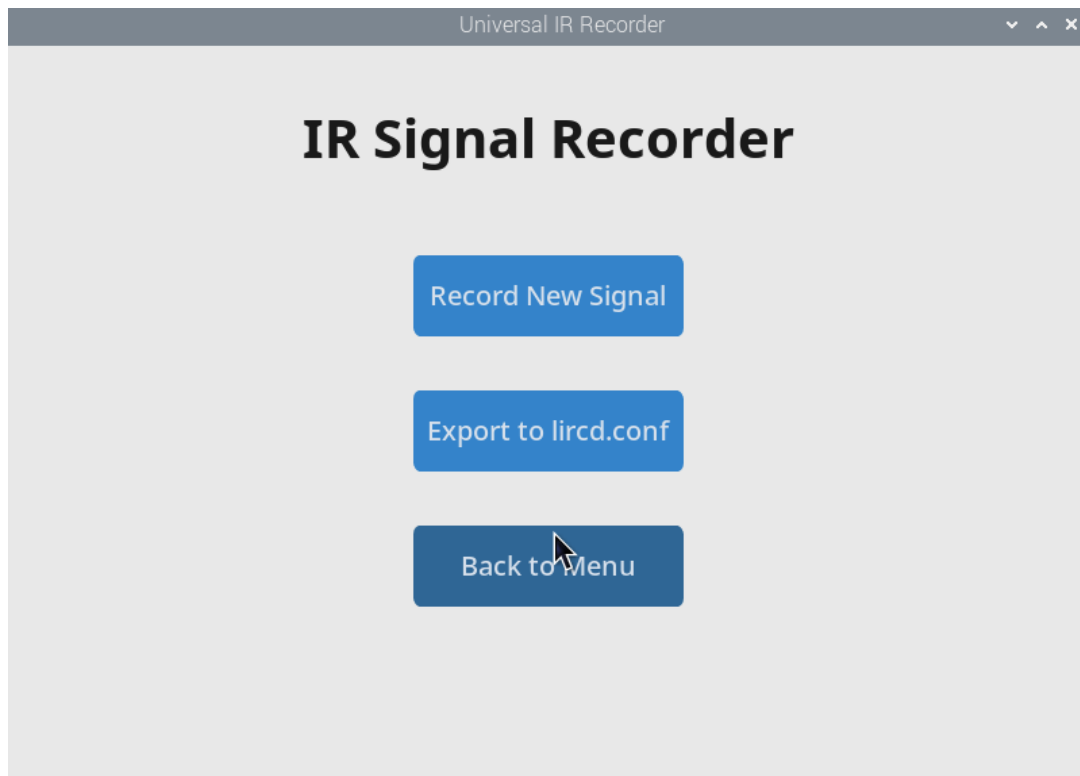


Figure 5.23 Window of IR Signal Recorder

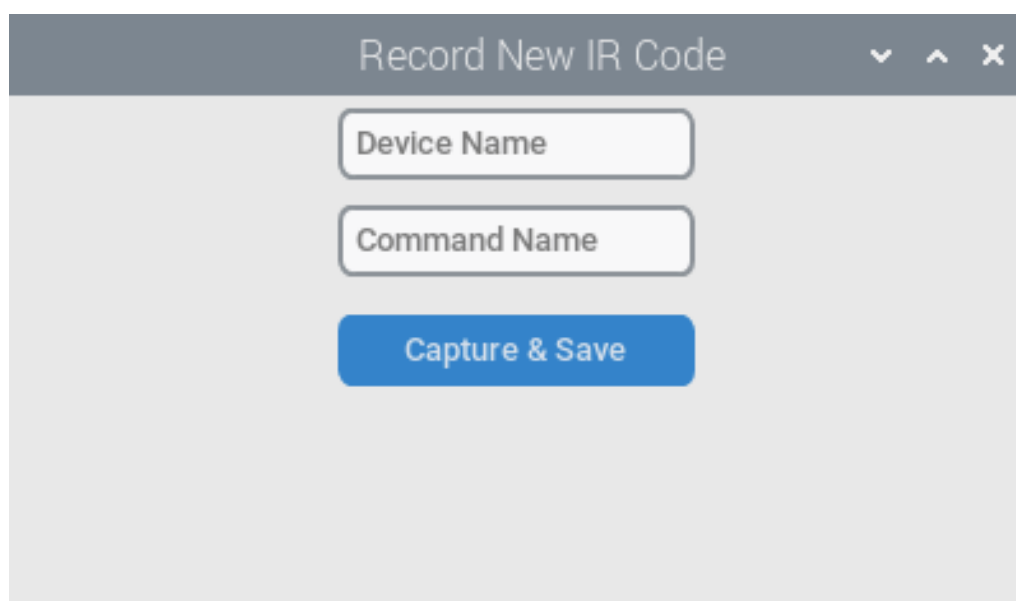


Figure 5.24 Window shown when “Record New Signal” button was pressed



Figure 5.25 Window shown when “Export to lircd.conf” button was pressed

When the “Control Devices” button as shown in Figure 5.4.2 is pressed, the user is redirected to the remote control GUI, which displays 4 remotes such as fan, light, humidifier and A/C.

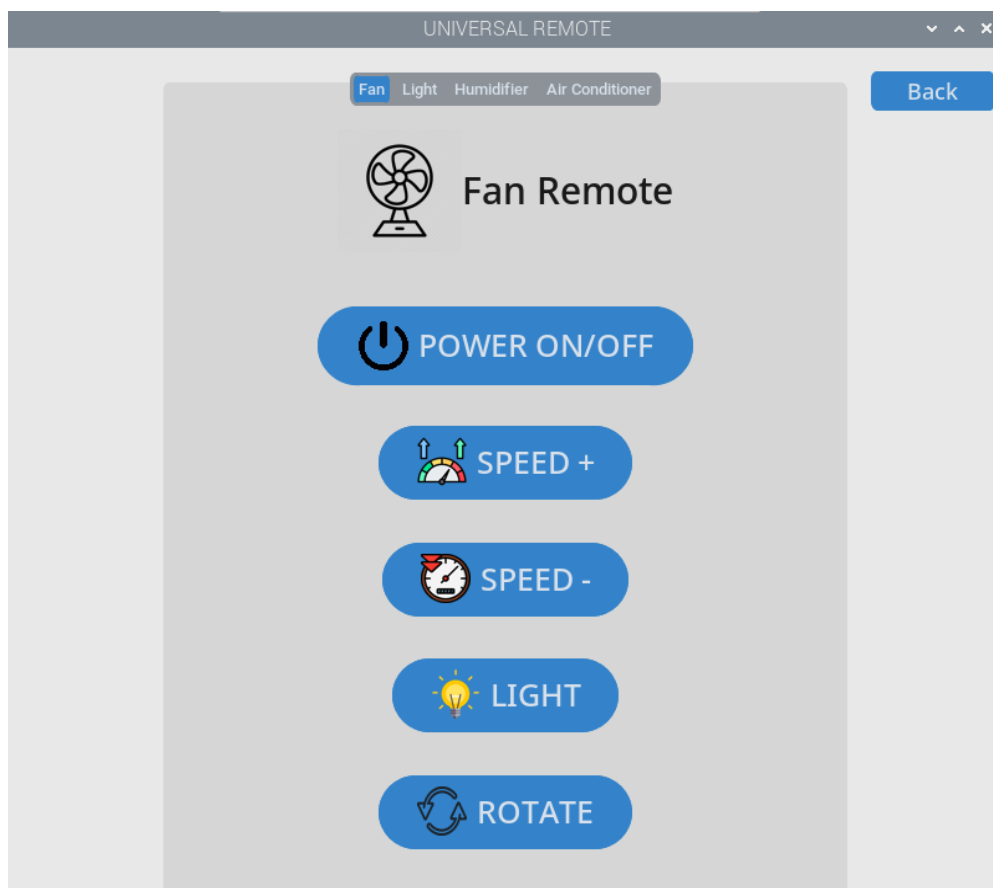


Figure 5.26 GUI of fan remote

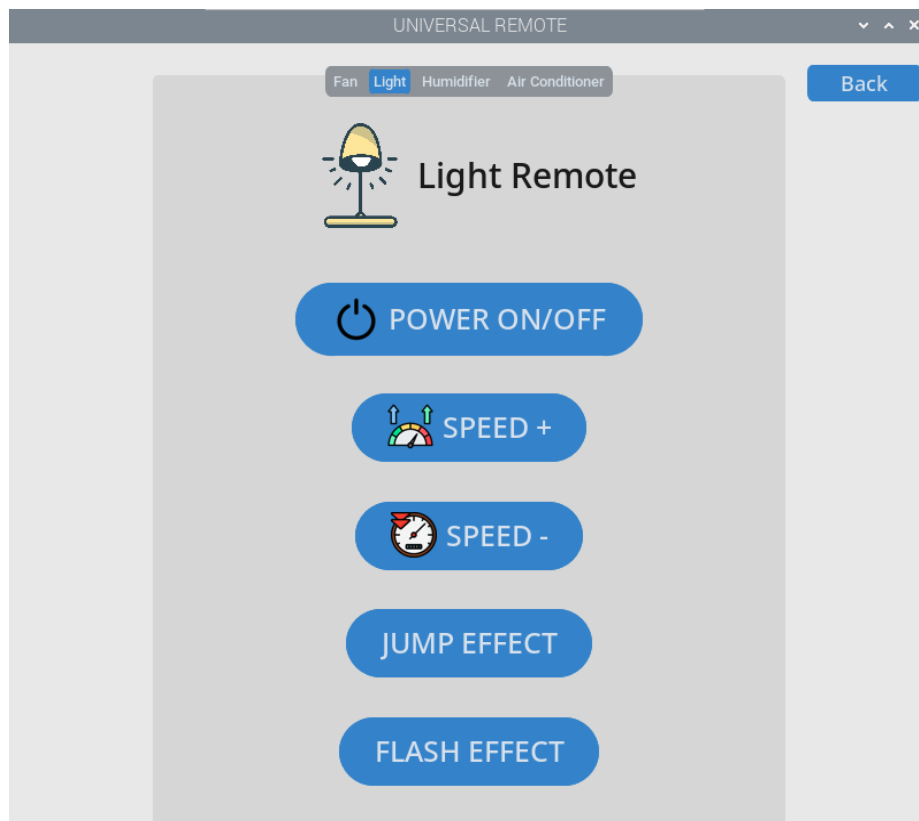


Figure 5.27 GUI of light remote

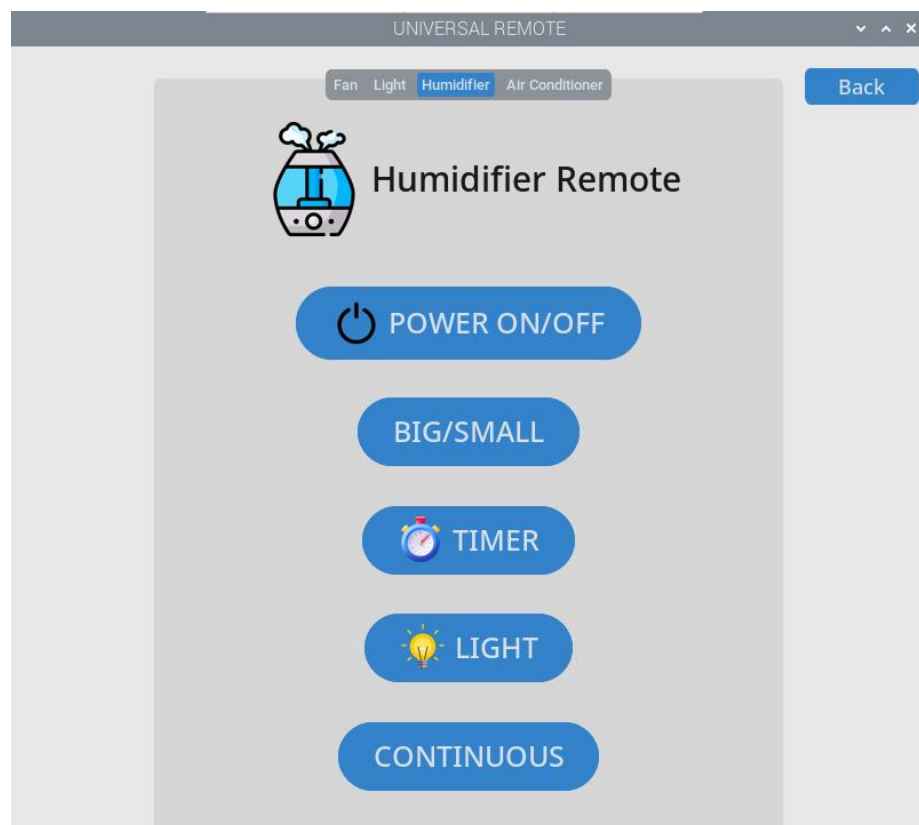


Figure 5.28 GUI of humidifier remote

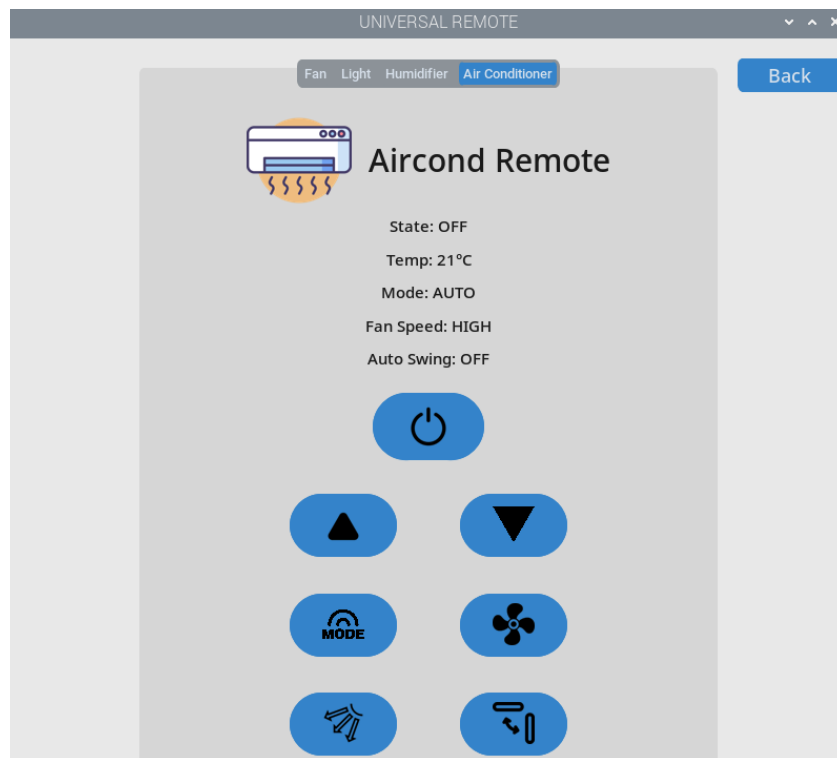


Figure 5.29 GUI of A/C remote

In order to provide a seamless user experience, the system was designed to store the latest operating state of the A/C. Parameter's such as power status, temperature, mode, fan speed and swing setting were recorded and updated dynamically whenever the user made a change through the GUI. These values were then saved into a JSON database as shown in Figure 5.4.10, ensuring that the most recent configuration was retained even if the program was closed. When the system was launched again, the stored state was automatically retrieved from the database and displayed on the GUI, as illustrated in Figure 5.4.9.

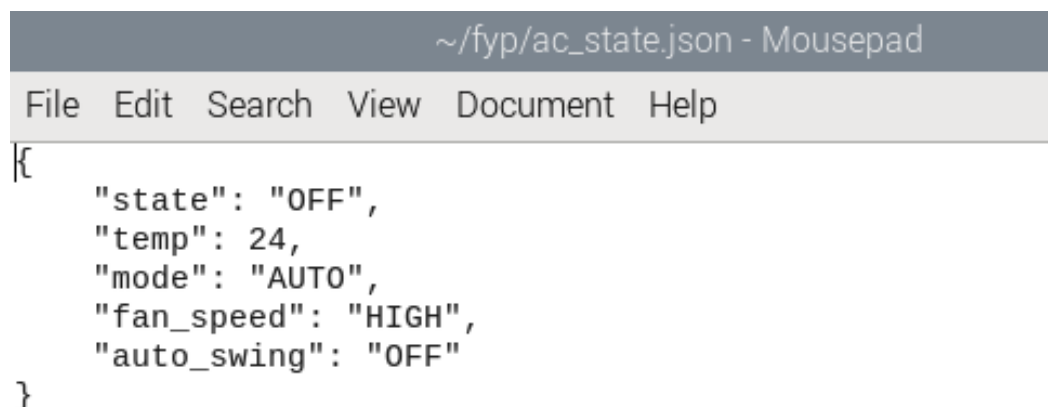


Figure 5.30 Latest A/C state stored in JSON file

5.5 Implementation Issues and Challenges

However, several challenges had been encountered during the implementation process. The first challenge was the different format of infrared codes output when executing the `irrecord` command. The format of the infrared codes for fan remote was in raw codes, whereas for the light remote was in hexadecimal format. Different format of infrared codes can't combine into a same `conf.` file, therefore, `mode2` command was used to obtain the raw code of the light remote in order to combine them into one `conf.` file.

Furthermore, the second challenge that I had encountered during project implementation was the GUI performance lag due to icons. During testing phase, the system experienced noticeable lag whenever user pressed the “Control Devices” button to display the remote control GUI. The delay was traced to the use of custom button icons in the GUI. Since each icon had to be loaded and rendered when the window was opened, it required additional processing overhead on the RPi. This issue occasionally caused the interface to feel less responsive, and the most critical was that it will affect the user experience.

Last but not least, another major challenge encountered during the implementation was the complexity of the A/C remote control signals. The difficulty lay in identifying which sections of this binary sequence corresponded to specific functions as some of the functions may overlapped the bits of another functions. Besides, multiple signals were captured and compared to isolate the bits responsible for each function, making the decoding process time-consuming and error-prone. This complexity made the analysis of the A/C signal one of the most challenging aspects of the project.

5.6 Concluding Remark

In summary, the system implementation involved the successful integration of both hardware and software components to develop the Universal Infrared System. The hardware setup established the foundation by connecting the RPi with the IR receiver and multiple IR transmitter modules, while the software setup provided the necessary libraries and tools, including Python, CustomTkinter and LIRC to support signal processing and GUI development. Through proper configuration, the RPi was able to reliably record, store and transmit IR signals. A user-friendly GUI was also implemented, enabling intuitive control of appliances, such as fan, light, humidifier and A/C.

Although several challenges were encountered during implementation process, including varying signal formats, GUI performance issues and the complexity of decoding A/C function codes, appropriate solutions were applied to ensure stable operation. The completed system was able to perform its intended functions effectively, forming a strong basis for subsequent testing and evaluation presented in next chapter.

Chapter 6

System Evaluation and Discussion

6.1 System Testing and Performance Metrics

In this chapter, system testing was conducted to verify that the Universal Infrared System developed using RPi performs according to the intended design requirements. The evaluation focused on both functional correctness and system performance to ensure that the system is reliable and user-friendly in recording IR signals from new IR remote and in controlling various appliances such as fans, lights, humidifiers and A/C.

The testing process followed the verification plans outlined in Table 6.2.1, covering hardware, software and GUI components. The objective of this stage was to ensure that the system met its requirements in terms of recording, storing, transmitting and managing infrared signals for multiple appliances.

6.1.1 Hardware Testing

The hardware components, including the RPi, IR receiver module and IR transmitter modules, were tested for proper operation. GPIO pin configuration was verified, ensuring that the receiver could detect signals accurately and the transmitters could send signals to control all of the appliances. Transmission range and angle were evaluated, confirming that the system could operate reliably within a distance of 5 meters and in any angles. These tests validated the effectiveness of the physical layer in achieving reliable communication with appliances.

6.1.2 Signal Recording and Storage

Signal recording was tested by capturing raw IR codes from various appliance remotes. The captured data was stored in a structured JSON database. Testing confirmed that the database could retain signal integrity across multiple sessions and after RPi reboots. Retrieval of stored signals was successful, and retransmission resulted in correct appliance responses. This validated the robustness of the storage mechanism and ensured that the system could act as a universal repository for remote codes.

6.1.3 GUI Functionality Testing

The GUI, developed using CustomTkinter, was evaluated for functionality and responsiveness. Separate interfaces were tested for each appliance category, including fan, light, humidifier and A/C. Results confirmed that the button presses, and toggle selections triggered the correct IR signals, with less than one second of delay between input and appliance response. The A/C control module, which dynamically generated signals for mode, fan speed and temperature, was verified to work correctly. Responsiveness test further confirmed that the GUI remained stable with no freezing or lag during extended operation.

6.1.4 Multi-Device Control and Remote Accessibility

The system was tested for controlling multiple appliances sequentially. No conflicts or mis-triggering occurred when switching between devices, demonstrating that the system could manage multiple IR codes effectively. Additionally, remote access testing using VNC Viewer confirmed that the GUI could be accessed and operated from an external device without noticeable performance degradation, thus enhancing user flexibility.

6.1.5 Performance Metrics

Performance metrics were measured to quantify the effectiveness of the system:

- Signal detection accuracy: 100% success in detecting signals from standard IR remotes.
- Signal transmission success rate: 98% success within 3 meters
- GUI responsiveness time: Average of <1 second delay per input action.
- System stability: Operated continuously for over 2 hours without crashes or hardware overheating.
- Database reliability: Signals stored in JSON remained intact after multiple retrievals and Raspberry Pi reboots.
- User usability: Test users with no prior technical experience were able to operate appliances successfully using the GUI.

6.2 Testing Setup and Result

Table 6.1: Verification plans

Test cases	Expected Result	Result
VNC Viewer remote access configuration	Capable of remotely accessing the RPi	Pass
GPIO configuration test	GPIO pins for IR receiver (GPIO 18) and transmitter (GPIO 17) are correctly initialized and accessible via Python	Pass
IR remote recording signal testing	The signals are able to detect by receiver module and record them down by using irrecord or mode2 command.	Pass
Signal storage in JSON	Recorded raw signals are correctly formatted and saved into the JSON database without data loss	Pass
Signal retrieval from JSON	System can correctly fetch the stored IR signal data for a selected appliance	Pass
Signal transmitting testing using terminal	The signals are able to control the appliances via terminal by using irsend command	Pass
Signal transmitting testing using GUI	The signals are able to control the appliances via GUI which developed using CustomTkinter	Pass
GUI responsiveness testing	The buttons and controls should respond correctly and quickly when pressed	Pass
GUI device control – Fan	Fan control buttons in the GUI trigger correct IR signals and operate the fan reliably	Pass
GUI device control – Light	Light control buttons in the GUI trigger correct IR signals and operate the light reliably	Pass
GUI device control – Humidifier	Humidifier buttons in GUI transmit correct IR codes to control the humidifier	Pass
GUI device control – A/C	AC GUI buttons (mode, fan speed, temperature) generate dynamic IR signals and control A/C	Pass
Infrared signal range testing	IR signal can control appliances at a certain distance (up to 3 meters)	Pass
Multi-Device Control Testing	No conflicts when multiple devices are controlled at the same time	Pass
Database integrity	JSON database retains saved signals even after RPi reboot or shutdown	Pass
User usability test	A user with no prior technical knowledge can operate the GUI to control appliances successfully	Pass

6.3 Project Challenges

During the development and implementation of the Universal Infrared system using RPi, several challenges were encountered. These challenges arose from both hardware and software aspects, as well as environmental factors that affect the system performance. The key challenges were summarized as below.

6.3.1 Weak Infrared Transmission and Coverage

One of the main difficulties was the output power of the initial 3 IR transmitter modules were insufficient, especially for appliances that required strong or complex signals such as A/C and for operation at longer distances. Therefore, to address this limitation, the number of IR transmitter modules was increased to 10, arranged in different directions. This enhancement significantly improved the effective signal strength as well as the coverage angle.

6.3.2 Signal Interference

Infrared communication was highly susceptible to interference from ambient light, particularly strong indoor light. During testing phase, it was observed that the system performed better under darker room environment compared to a brighter environment.

6.3.3 Overlapping Command Codes

One of the most critical challenges encountered was the LED light appliance could be unintentionally controlled by the fan remote. This occurred because both devices used standardized IR protocols operating at the same frequency, which caused an overlap in certain command codes. As a result, a command intended for the fan occasionally triggered the LED light, leading to unintended appliance control.

6.3.4 Time and Resource Constraint

The project was limited to a small set of appliances, which includes only mini fan, LED light stands, humidifier and A/C. Due to time and resource constraints, testing could not be extended to a wider variety of brands and models, which may affect the universality of the system in broader applications.

6.3.5 Manual GUI creation for new remotes

Another challenge identified was the limitation in the system's scalability for new appliances. The current design requires user to manually create and configure the GUI whenever a new remote control is added. Since the system does not support automatic button generation based on recorded IR signals, each new device must be manually coded and mapped to its corresponding commands. This process not only increases the setup time but also demands technical knowledge from the user, making the system less convenient for those non-technical users.

6.3.6 Compatibility across different appliances

Different appliances use different IR protocols and encoding schemes. While simple devices such as fans and lights responded well to recorded signals, more complex appliances such as A/C or heaters posed challenges due to multi-byte command structures that included all of the operation settings in a single signal. Therefore, additional effort was required to ensure proper decoding and transmission of these signals.

6.4 Objectives Evaluation

The main objective of this project was to design and implement a Universal Infrared System using RPi which is able to record, store and transmit IR signals to control multiple household appliances. Based on the testing and results obtained, most of the stated objectives were successfully achieved, although some limitations were identified.

First and foremost, the system was able to record raw infrared signals from different appliance remotes using the HX-1838 receiver module. The recorded signals were successfully stored in a JSON-based database which ensured organized and flexible data management. This fulfilled the objective of creating a structured storage system that could be easily extended to support new appliances.

Secondly, the system was designed to transmit stored IR signals using multiple IR transmitter modules connected to a RPi. The results demonstrated a high transmission accuracy with success rate above 90% across all tested appliances. This confirmed that the achievement of the objective to provide reliable transmission performance.

Thirdly, a user-friendly GUI was successfully developed using CustomTkinter, provided a familiar remote-control layout, enabling users to interact with the system through virtual buttons for each appliance. However, one limitation observed was that the system currently not supported to generate the button dynamically, but requires user to manually create the new GUI layouts for the new recorded remote. This partially met the objective of building a user-friendly and extensible interface.

Finally, the system was tested on a range of appliances including fan, light, humidifier and A/C. the overall performance showed that the system was versatile and capable of controlling multiple devices, fulfilling the objective of developing a universal remote. Nevertheless, certain challenges remained, such as overlapping IR command codes between different devices which were unable to solve.

In a nutshell, the project objectives were largely achieved as the system fully demonstrated the ability to store, record and transmit IR signals, while also providing a functional GUI for device control. Although some limitations were encountered, the overall results highlighted the potential of the system as a practical and low-cost universal remote. By consolidating multiple appliance remotes into a single platform, the system also effectively reduces the number of physical remote controls required.

6.5 Concluding Remark

This chapter presented the evaluation of the proposed Universal Infrared System using RPi, covering system testing, performance analysis, challenges encountered and the extent to which the project objectives were achieved. The results demonstrated that the system is capable of recording, storing and transmitting IR signals with a high level of accuracy, while the developed GUI provided an effective platform for user interaction.

Despite minor challenges such as overlapping command codes, susceptibility to environmental lighting and the need for manual GUI configuration when adding new appliances, the system proved to be reliable and functional across different devices, including fans, lights, humidifier and A/C. Overall, the evaluation confirmed that the project objectives were successfully met, establishing the system as a low-cost, flexible and practical universal infrared remote solution.

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

In this project, a Universal Infrared System using RPi was developed to address a common issue in modern households: the inconvenience of managing multiple remote controls for various electronic appliances such as fans, lights, and humidifiers. The motivation behind this project stemmed from the increasingly cluttered home environment due to the growing number of electronic devices, each typically controlled by its own dedicated remote. This situation not only leads to user frustration when remotes are misplaced or lost but also decreases the overall efficiency of daily interactions with smart appliances.

In order to resolve these challenges, the project proposed a centralized solution, a universal infrared remote system built around the RPi platform. The proposed system is capable of capturing, storing, and transmitting infrared signals corresponding to different appliance remotes. The RPi serves as the central controller, interfacing with IR receiver and transmitter modules to learn and resend the required signals. One of the core components of the solution is the development of a user-friendly GUI, implemented using CustomTkinter in Python. The GUI allows users to visually identify and control appliances with clearly labelled buttons and representative icons, promoting an intuitive and accessible experience.

The methodology followed a prototyping approach that included requirement gathering, hardware configuration, software integration, and continuous testing. Through several iterations of feedback and refinement, the system was fine-tuned for better accuracy and reliability. The recorded IR signals were verified and successfully used to control physical devices, demonstrating that the RPi and LIRC library could work together effectively to replicate remote control functionality.

One notable finding during implementation was the unexpected interaction between different appliances due to overlapping infrared codes. For example, it was discovered that a fan remote could unintentionally control an LED light. This issue was caused by both devices operating on similar IR protocols and frequencies, resulting in code conflicts. This highlighted the importance of ensuring unique command mappings and the challenge of IR protocol standardization. Moreover, another challenge involved incompatible formats of IR codes, some remotes outputted raw codes while others used hexadecimal. This required manual adjustments using tools like mode2 and irrecord to maintain consistency and compatibility within configuration files.

A novel aspect of this project is the integration of multiple IR-controlled devices within a single system that includes both the hardware and a complete GUI environment. While prior works have proposed similar systems using Arduino or ESP8266 microcontrollers, the use of RPi provided enhanced flexibility due to its support for high-level operating systems, libraries, and multitasking capabilities. This makes the system more scalable and adaptable for future expansion, such as integrating more appliances or connecting the system with cloud services or voice assistants.

The final product of this project is a fully functional prototype that enables centralized and simplified control of common home appliances using a RPi. It successfully meets the project's objectives by reducing remote clutter, improving accessibility, and providing a cost-effective smart home solution. Testing confirmed that the system performs reliably in various conditions, with effective signal transmission, responsive GUI interactions, and a control range of up to three meters.

In conclusion, the Universal Infrared System using RPi represents a meaningful contribution to the field of smart home automation by combining affordability, functionality, and user-centric design. It demonstrates that a centralized, IR-based control system can effectively replace multiple remotes in a typical home setting and pave the way for more intelligent and unified home control solution.

7.2 Recommendation

7.2.1 Integration of Additional Communication Protocols

The current system focuses exclusively on infrared communication, which limits its use to appliances that rely on IR remotes. However, many modern household devices now operate on alternative protocols such as radio frequency (RF), Bluetooth, and Wi-Fi. Expanding the system to support these additional protocols would significantly broaden its compatibility with a wider range of smart and traditional appliances. For instance, RF support would enable the control of garage doors or ceiling fans, while Bluetooth and Wi-Fi integration would make it possible to connect with smart speakers or IoT-enabled lights. By incorporating these communication standards, the system could evolve from a simple infrared-based controller into a multi-protocol universal remote solution, increasing its versatility and value in a smart home environment.

7.2.2 Mobile Application Support

At present, the system relies on a desktop-based GUI that runs on the Raspberry Pi. While effective, this limits accessibility for end-users who may prefer more portable and intuitive control options. Developing a dedicated mobile application or a responsive web-based interface would allow users to operate appliances directly from their smartphones or tablets. Such an extension would provide greater flexibility, enabling users to control devices remotely within their home network or even from outside the home when combined with internet access. A mobile interface could also improve usability by incorporating touch gestures, voice commands, or quick-access widgets, making the system more user-friendly and convenient for everyday use.

7.2.3 Security and User Authentication

As the system grows to support multiple appliances and potentially integrates with remote access tools like VNC or future mobile applications, security becomes a critical consideration. Currently, the system operates without authentication, which could allow unauthorized users to control connected appliances. Implementing user authentication mechanisms such as passwords, PIN codes, or even biometric verification in a mobile app would help safeguard the system. Additionally, encrypted communication channels could be established for remote access to prevent interception of control commands. Strengthening the system's security

measures not only protects user privacy but also ensures reliability and trustworthiness when deployed in real household environments.

7.2.4 Integration with Smart Home Ecosystems

Smart home adoption is increasing rapidly, with platforms such as Google Home, Amazon Alexa, and Apple HomeKit becoming central to household automation. Integrating the Universal Infrared System with these ecosystems would allow users to control IR-based appliances through voice commands or as part of automated routines. For example, a user could program a voice command such as “movie mode” to dim the lights, turn on the television, and adjust the air conditioner simultaneously. This level of integration would transform the prototype from a standalone solution into a component of a larger smart home ecosystem, significantly enhancing its usefulness and market appeal.

7.2.5 Increasing the Type of Appliances

At present, the system is tested primarily with common household devices such as a fan, light, humidifier, and air conditioner. To enhance its practicality and user adoption, the system should be expanded to support a wider range of appliances. Many home devices, including televisions, set-top boxes, projectors, and audio systems, rely on infrared remotes and can be integrated into the database. By broadening the coverage, the system can serve as a more complete replacement for multiple remote controls.

REFERENCES

- [1] TechTarget, "What is IoT (Internet of Things)? ,\" TechTarget, Sep. 2024. [Online]. Available: <https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT>.
- [2] IBM, "What is the Internet of Things (IoT)?,\" IBM, Sep. 2024. [Online]. Available: <https://www.ibm.com/topics/internet-of-things>.
- [3] G. Writer, "How IoT Soil Condition Monitoring Is Empowering Farmers,\" *IoT For All*, Nov. 06, 2019. [Online] Available: <https://www.iotforall.com/soil-moisture-monitoring>
- [4] D. Ricci, "Motorized Tv Lift: The Total Guide,\" *Maior*, Jan. 11, 2023. [Online]. Available: <https://www.maiortvlift.com/lifestyle/decor-and-design/smart-home-vs-home-automation-understanding-the-key-differences/#how>
- [5] A. Jacobs, "15 Benefits of Smart Home Automation,\" *Jacobs Heating & Air Conditioning*, Aug. 01, 2023. [Online]. Available: <https://jacobsheating.com/blog/benefits-home-automation/>
- [6] Wonderopolis, "How does a remote control work?,\" Wonderopolis, Sep. 2024. [Online]. Available: <https://wonderopolis.org/wonder/how-does-a-remote-control-work>.
- [7] A. E. Amoran, A. S. Oluwole, E. O. Fagorola, and R. S. Diarah, "Home automated system using Bluetooth and an android application,\" *Scientific African*, vol. 11, Mar. 2021, Art. no. e00711, doi: 10.1016/j.sciaf.2021.e00711.
- [8] Y.-N. Lin, S.-K. Wang, C.-Y. Yang, V. R. L. Shen, T. T.-Y. Juang, and W.-H. Hung, "Development and verification of a smart remote control system for home appliances,\" *Computers and Electrical Engineering*, vol. 88, no. 106889, pp. 1–16, Oct. 2020. doi: 10.1016/j.compeleceng.2020.106889.
- [9] Md. W. B. Hafiz, "Arduino Based Home Automation System Using Android Application,\" *ResearchGate*, July 2020. [Online]. Available : <https://www.researchgate.net/publication/353464169>. [Accessed: Sep. 11, 2024]. DOI: 10.13140/RG.2.2.20219.03360.

- [10] D. Lubianov, K. Kasian, and M. Kasian, "A reasonable smart home technology on the Arduino," in *The Fourth International Workshop on Computer Modeling and Intelligent Systems (CMIS-2021)*, Zaporizhzhia, Ukraine, Apr. 2021, pp. 1-6. Available: CEUR-WS.org.
- [11] C. B. Kolanur, R. M. Banakar, and Rajneesh G, "Design of IoT based Platform Development for Smart Home Appliances Control," *J. Phys.: Conf. Ser.*, vol. 1969, no. 1, p. 012052, 2021, doi: 10.1088/1742-6596/1969/1/012052.
- [12] N.-A.-Alam, M. Ahsan, M. A. Based, J. Haider, and E. M. G. Rodrigues, "Smart Monitoring and Controlling of Appliances Using LoRa Based IoT System," *Designs*, vol. 5, no. 1, p. 17, 2021, doi: 10.3390/designs5010017.
- [13] A.-S. A. Ali and X. Bao, "Design and Research of Infrared Remote Control Based on ESP8266," *OALib*, vol. 08, no. 04, pp. 1–14, 2021, doi: <https://doi.org/10.4236/oalib.1107314>.
- [14] K. Win and N. Win, "Design and Construction of Infrared Remote Controller for Multiple Home Appliances," *International Journal of Advances in Scientific Research and Engineering (IJASRE)*, vol. 6, no. 5, pp. 82–87, May 2020, doi: 10.31695/IJASRE.2020.33807.
- [15] Raspberry Pi, "Raspberry Pi 4 Model B Specifications," Raspberry Pi, Sep. 2024. [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>.
- [16] "LIRC - Linux Infrared Remote Control," *Lirc.org*, 2022. <https://lirc.org>
- [17] w3Schools, "Introduction to Python," *W3schools.com*, 2019.
- [18] "RealVNC Connect and Raspberry Pi," *RealVNC Help Center*, Feb. 27, 2024. <https://help.realvnc.com/hc/en-us/articles/360002249917-RealVNC-Connect-and-Raspberry-Pi#setting-up-your-raspberry-pi-0-0> (accessed Apr. 30, 2025).
- [19] MongoDB, "What Is A JSON Database? | All You Need To Know," *MongoDB Resources: Basics – Databases*, 2025. [Online]. Available: <https://www.mongodb.com/resources/basics/databases/json-database> [Accessed: Sep. 20, 2025].

Reference

- [20] T. Schimansky, *CustomTkinter*, version 0.3, PyPI, May 7, 2021. [Online]. Available: <https://pypi.org/project/customtkinter/0.3/> [Accessed: Sep. 20, 2025].
- [21] Rich101101, "Easy Setup IR Remote Control Using LIRC for the Raspberry PI (RPi) - Updated Oct 2021 [Part 1]," *Instructables*. <https://www.instructables.com/Setup-IR-Remote-Control-Using-LIRC-for-the-Raspber/>
- [22] "How to Send and Receive IR Signals with a Raspberry Pi," *DigiKey*, 2021. <https://www.digikey.com/en/maker/tutorials/2021/how-to-send-and-receive-ir-signals-with-a-raspberry-pi>
- [23] C. Mun, "Smart universal remote using mobile for home automation," *Final Year Project Report*, Universiti Tunku Abdul Rahman, 2021. [Online]. Available: <http://eprints.utar.edu.my/id/eprint/4282> [Accessed: Apr. 30, 2025].
- [24] B. Schwind, "Sending Infrared Commands From a Raspberry Pi Without LIRC," *Blog of Brian Schwind*, May 29, 2016. [Online]. Available: <https://blog.bschwind.com/2016/05/29/sending-infrared-commands-from-a-raspberry-pi-without-lirc/> [Accessed: Sep. 20, 2025].
- [25] C. Addis, "Smart Air Conditioner with Raspberry Pi — An Adventure," *Medium*, Aug. 10, 2018. [Online]. Available: <https://medium.com/@camilloaddis/smart-air-conditioner-with-raspberry-pi-an-odyssey-2a5b438fe984> [Accessed: Sep. 20, 2025].
- [26] "Reverse-engineering of an Air-Conditioning control," *Instructables*. [Online]. Available: <https://www.instructables.com/Reverse-engineering-of-an-Air-Conditioning-control/> [Accessed: Sep. 20, 2025].

APPENDIX

Specifications of RPi 4 Model B

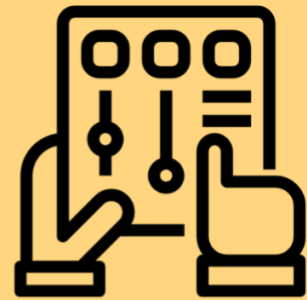
Specification

Processor:	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
Memory:	1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
Connectivity:	<ul style="list-style-type: none"> • 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE • Gigabit Ethernet • 2 × USB 3.0 ports • 2 × USB 2.0 ports.
GPIO:	Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
Video & sound:	<ul style="list-style-type: none"> • 2 × micro HDMI ports (up to 4Kp60 supported) • 2-lane MIPI DSI display port • 2-lane MIPI CSI camera port • 4-pole stereo audio and composite video port
Multimedia:	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
SD card support:	Micro SD card slot for loading operating system and data storage
Input power:	<ul style="list-style-type: none"> • 5V DC via USB-C connector (minimum 3A¹) • 5V DC via GPIO header (minimum 3A¹) • Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature 0–50°C

Poster

UNIVERSAL INFRARED SYSTEM USING RASPBERRY PI

BY: TAN HUNG MHENG 2105223
PROJECT SUPERVISOR: DR TEOH SHEN KHANG



INTRODUCTION

Remote controls are widely used in homes, but managing multiple devices with separate remotes can be inconvenient. This project presents a Raspberry Pi-based Universal Infrared Remote capable of recording, storing, and transmitting IR signals to control appliances such as a fan, light, humidifier, and air conditioner. The project aims to deliver a low-cost, flexible, and user-friendly solution for smart home control.

3

RESULTS

- System successfully controlled Fan, Light, Humidifier, and A/C through a unified interface.
- Dynamic signal generation for A/C supported mode, fan speed
- system are able to store IR signal inside database
- GUI interface worked smoothly with clear button mapping and real-time feedback.

METHODS

2

RPI
Configuration



Hardware
Software
Implementation

Signal recording
and transmitting
using LIRC

Controlling
appliances via
unified GUI

DISCUSSION

This system demonstrated reliable control of multiple appliances using recorded and transmitted IR signals. The A/C posed more challenges due to its complex signal structure, but dynamic packet generation with checksum validation proved effective. The GUI provided a simple and intuitive interface, though scalability could be improved by automating remote button creation.

5

4

CONCLUSION

This project successfully demonstrates the development of a universal infrared control system using Raspberry Pi, capable of replacing multiple remote controls with a unified interface. The system enhances convenience, efficiency, and accessibility in smart home environments and offers a practical step toward affordable and open-source home automation.



**BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)
COMPUTER ENGINEERING**