LUIGI -PET ROBOT CAR

Microcontroller Based Hardware Project

Faculty Of Information Technology

University Of Moratuwa

September 2024

Group No 07

Name	Index No.
Nimhan R.D.S.	224135V
Gunawardana T.U.D.	224063X
Kahanda M.C	224098H
Gamage G.G.P.T.	224056E
Kumari J.K.A.V	224115K

Table of Contents

Introduction 5
Problem in brief 6
Literature review 7
Aims and objectives 8
Design and Analysis 9
Proposed Solution 9
System block diagram 10
Schematic Diagram 11
PCB Design 12
The 3D view 13
Components 14
Raspberry Pi 4 Model B 14
How it works 14
ESP32S Wi-Fi Bluetooth Dual Mode Microcontroller Module 15
How it works 15
TTP223 1-Channel (Red) Digital Capacitive Touch Sensor Module 17
How the touch Sensor works 17
LCD module Pi TFT 3.5-inch (320×480) Touchscreen Display Module 19
How the LCD module works 19
HC-SR04 4Pin Ultrasonic Sensor Module 21
How it works: 21
Camera Module v1.3 5MP 1080p 720p 22
How it works 22
MPU-6050 Triple Axis Analog Accelerometer Gyroscope Module 24
How it works 24
USB Microphone Dongle Raspberry Pi PC 25
How it works 25
Thumb Joystick Module 26
How it works 27
LED 5mm 28
How It Works 28
PAM8403 3W Stereo Audio Amplifier Board 29

How It Works 29
4 Ohm 3W Speaker 30
How It Works 30
MG90S Metal Gear Servo Motor 31
How It Works 31
L298N Motor Driver Module 32
How It Works 33
DC Motor 12V 34
How It Works 34
Li-ion Battery BMS Charger Protection Board 35
How It Works 35
3.7V 3200mA and 1800mA 18650 Li-ion Rechargeable Battery 37
How It Works 37
LM2596S 3-40V to 1.5-35V 4A DC to DC Adjustable Step-Down Buck Module 38
How It Works 38
Testing and Implementation 40
Unit Testing 40
Integration Testing 41
System Testing 42
Implementation 43
Individual Contribution 45
Nimhan R.D.S. - 224135V 45
Gunawardhana T.U.D. 224063X 49
Rock-Paper-Scissors Game 55
Voice Assistant 55
Gamage G.G.P.T. 224056E 56
Kumari J.K.A.V. 224115K 59
Kahanda M.C. 224098H 64
Reference 69

Table Of Figures

Figure 1 System Block Diagram 10
Figure 2 Luigi's Schematic Diagram 11
Figure 3 Luigi's Remote Controller Schematic Diagram 11
Figure 4 Luigi's Remote Controller PCB Design 12
Figure 5 Luigi's Inner PCB Design 12
Figure 6 Luigi's 3D view 13
Figure 7 Remote controller 3D design 13
Figure 8 Raspberry Pi 4 Model B 14
Figure 9 ESP32S Wi-Fi Bluetooth Dual Mode Microcontroller Module 15
Figure 10 TTP223 1-Channel (Red) Digital Capacitive Touch Sensor Module 17
Figure 11 LCD module Pi TFT 3.5-inch (320×480) Touchscreen Display Module 19
Figure 12 HC-SR04 4Pin Ultrasonic Sensor Module 21
Figure 13 Camera Module v1.3 5MP 1080p 720p 22
Figure 14 MPU-6050 Triple Axis Analog Accelerometer Gyroscope Module 24
Figure 15 USB Microphone Dongle Raspberry Pi PC 25
Figure 16 Thumb Joystick Module 26
Figure 17 LED 5mm 28
Figure 18 PAM8403 3W Stereo Audio Amplifier Board 29
Figure 19 4 Ohm 3W Speaker 30
Figure 20 MG90S Metal Gear Servo Motor 31
Figure 21 L298N Motor Driver Module 32
Figure 22 DC Motor 12V 34
Figure 23 Li-ion Battery BMS Charger Protection Board 35
Figure 24 Li-ion Rechargeable Battery 37
Figure 25 LM2596S 3-40V to 1.5-35V 4A DC to DC Adjustable Step-Down Buck Module 38
Figure 26Unit Testing 41
Figure 27 Integration Testing 42
Figure 28 System Testing 43
Figure 29 Implementation 44
Figure 30 Nimhan R.D.S. - 224135V (Schematic diagram) 45

Figure 31 Luigi's Function Menu 45
Figure 32 Tkinter code 46
Figure 33HTML,CSS and Js code 46
Figure 34 Function to emotional sound and video 47
Figure 35 Double tap detection function 48
Figure 36 Video footages 48
Figure 37 Gunawardhana T.U.D. 224063X (Schematic diagram) 49
Figure 38Raspberry Pi OS setup 50
Figure 39 Wireless Configuration setup 51
Figure 40 VNC Configuration 51
Figure 41 Touch Calibration 52
Figure 42 Resolution Change and Rotation 53
Figure 43 Output Audio 53
Figure 44 Input audio 54
Figure 45 Gamage G.G.P.T. 224056E (Schematic Diagram) 56
Figure 46 Code for spitting the string 57
Figure 47 Code for controlling the servo angle. 57
Figure 48 Code for controlling the DC motors and LEDs 58
Figure 49 Kumari J.K.A.V. 224115K (Schematic diagram) 59
Figure 50 Code for detecting the shake and changing the color theme 60
Figure 51 Code for detecting the shake and wake up mode 61
Figure 52 Code for activate the fear reaction 62
Figure 53 Code for activate the safety feature 63
Figure 54 Buck Converter 63
Figure 55 Kahanda M.C. 224098H (Schematic Diagram) 64
Figure 56 Code for identify user commands and send them to a Raspberry Pi via Bluetooth 65
Figure 57 PCB designs 66
Figure 58 3D Designs 66
Figure 59 Games made by Tkinter 67
Figure 60 threading 68

–

Introduction

Luigi is our creative pet robot project meant to bring life to people working long hours at the desk. This is an interactive car with many fun features and built around a microcontroller. The touch screen interface is its main feature, enabling users to interact with Luigi in the form of giving it instructions on the various tasks. Its user interface turns Luigi from a simple pet robot into a fun and responsive workplace partner. One characteristic feature Luigi boasts of is that it can be driven like any other remote-controlled car. A rather fun and engaging way for users to control Luigi is by having him on the ground while being controlled with a joystick. Luigi has more than just the ability of self-movement; it also has an ultrasonic sensor that can determine when it is being raised higher than 10 cm above the floor. It responds to this by turning on its servos, creating a faint vibration-a pet reacting, in a way, to being lifted. There is still even more to this interactive experience. Luigi is made to respond with physical contact as well, more precisely on its roof. Its screen is filled with the changing expressions of emotion once a touch has been present; this is something that makes it seem somehow a little more relatable and real. This form of emotional feedback makes interacting with Luigi even more engaging and natural, building up a close relationship between the human and robot. Beyond that, Luigi can take selfies with a camera module, adding an interesting and modern touch to his powers. Its ability to show time makes it functional for users working through the night to keep track of the time. Luigi is equipped with a speaker that can play all kinds of sounds upon request for added entertainment. At the end of the day, Luigi is no desk toy; he is, in fact, a multi-functional buddy designed to away boredom and to create a feeling of contact and companionship. Put all together, the haptic elements, mobility, emotive display, and multimedia features make Luigi unique and compelling on any desktop.

Problem in brief

During modern times, when work is highly digitized and sedentary, professionals often must work at their workstations for long periods, often in isolation. Thus, time consumed sitting at their desk might be uninteresting, lonely, and de-motivating, especially those jobs that may be highly repetitive or require little interpersonal communication. Traditional desk arrangements lack the required dynamism and interactive feel that can break the monotony of the workplace. These often are comprised of little more than very basic office supplies and technology. As such, workers may find it difficult to remain focused, motivated, and productive in general. Lack of engagement or feeling of belonging would eventually result in mental fatigue, which decreases productivity and dampens the overall being of the individual. In addition, lack of involvement will tend to make the day at work even longer and exhausting with consideration of the remote and solitary environment of work. It clearly seems something needs to be done to get these issues resolved.

Apart from being a source of entertainment, the solution should ensure a social participatory experience that brings about bonding and makes work less boring and more joyful. This is where our creative pet robot, named Luigi, comes in handy. Luigi offers various kinds of interactive features, well beyond mere functionality, that completely revolutionize the experience of working on your workstation. Luigi is intended for multiple purposes: to enrich users' experience, reduce boredom, and allow for a more connected, engaging workday through touch-responsive interactions, emotional feedback, mobility, and multimedia. With Luigi, our goal is to make a desk friend that extends the work environment for reasons of entertainment, while building up a sense of connection and reducing feelings of isolation that are often associated with long amounts of desk time. Luigi tries to enhance emotional and mental health in people's lives at the modern workplace because these are problems being addressed.

Literature review

1. Companion Robots in Modern Workspaces

   Companion robots are becoming more popular as people work longer hours in solitary settings, especially at desks. A 2004 study by Shibata et al. on the therapeutic robot PARO emphasized the advantages of robots in lowering stress and encouraging relaxation via emotional engagement. The purpose of these robots is to mimic social presence, which can enhance users' mental health (Breazeal, 2003). Luigi fits within this paradigm by offering a responsive and dynamic interface that fosters emotional interaction in addition to physical companionship.

2. Interactive Robotics Displays

   It has been investigated whether touch screens in robotics can improve user engagement. Real-time human interaction with robots is made possible by touch displays, which offer a more user-friendly interface (Park, et al., 2011). This tendency is supported by Luigi's touch screen implementation, which lets users interact and provide commands to the robot. It does this by substituting a recognizable interface for intricate control systems, making the robot more user-friendly for non-experts.

3. Robot Mobility and Sensor Systems

   For many robotic systems, mobility is an essential component. Studies on mobile robots, such those conducted in 2004 by Siegwart and Nourbakhsh, highlight how crucial real-time control and navigation are to improving user experience. Users may direct Luigi's movement with its remote-control feature, which encourages playfulness and a sensation of control. An additional layer of interaction is provided by the ultrasonic sensor that measures elevation. This is consistent with the results of Fox et al. (2005), who emphasized the role that sensor systems play in increasing robot reactivity to external stimuli.

Aims and objectives

Aims:

Our main goal is to develop a strong bond between robots and human beings by the use and advancement of modern technology. This incorporates interactivity by ability and behavior within design to create an experience of having a living being with you in your working environment or home. We intend to enhance productivity, improve mental health, and create a comfort zone around the user.

Objectives:

1. To design a mechanical robot that may provide the people with an immersive experience as if interacting with a living being.
2. To design in such a way so that the user can provide comfort in an atmosphere where the user can move their body freely without much stress and can spend time relaxing.
3. In response to a big, tiring day at work, trigger relaxation, and provide emotional support through the interaction with the robot.
4. Allow the users to have interactive games with this robot and make their leisure time more productive or provide them relief from their hectic routines of life.
5. Provide diversion to the user's minds and help them by providing solace to overcome the hardships of their troubled life.
6. To let the user have an interactive environment without requiring them to leave their working or sitting areas.
7. It would be designed to invent interactive technology that can respond to human behavior to go beyond traditional robot toys and remote-controlled bots.
8. Create an affordable toy for both professionals and children in the context of this modern high-value creation era.

Design and Analysis

Proposed Solution

We have developed a desktop table pet called "LUIGI" using a few advanced technical components. The compact and attractive design allows users to place it on their desk or nearby, enhancing coziness in their surroundings. Whenever the robot is touched or moved, it responds by depicting various emotions and making weird sounds. Further, we have integrated interactive games into the robot, enhancing the user's real-time bonding with the pet. Besides that, users can easily manage the movements of the robot using the joystick controller. The touch sensor instills the act of petting a living pet with sounds and emotive expressions through the eyes of it, specific to each situation. This would include the fact that, for its safety mechanism, it is designed to act quickly if it detects an obstacle in front of it. Such an active response mechanism would enhance its behavior in general, much like the behavior of an actual pet.

Our robot can play games like rock-paper-scissors to build better rapport with the human user. This will be dynamic for each scenario presented in those games, hence making it even more interactive. LUIGI can also be used as a rover to entertain users with its remote control for an enhanced user experience. It has buttons for the on and off headlights, along with a horn, too, for better interaction of the user with their robotic pet. The robot also provides options like a stopwatch and showing date and time for more user-friendliness. The sleep mode of LUIGI allows us to conserve the available power for the robot to operate without disturbing the user since it also displays the time while in sleep mode.

System block diagram

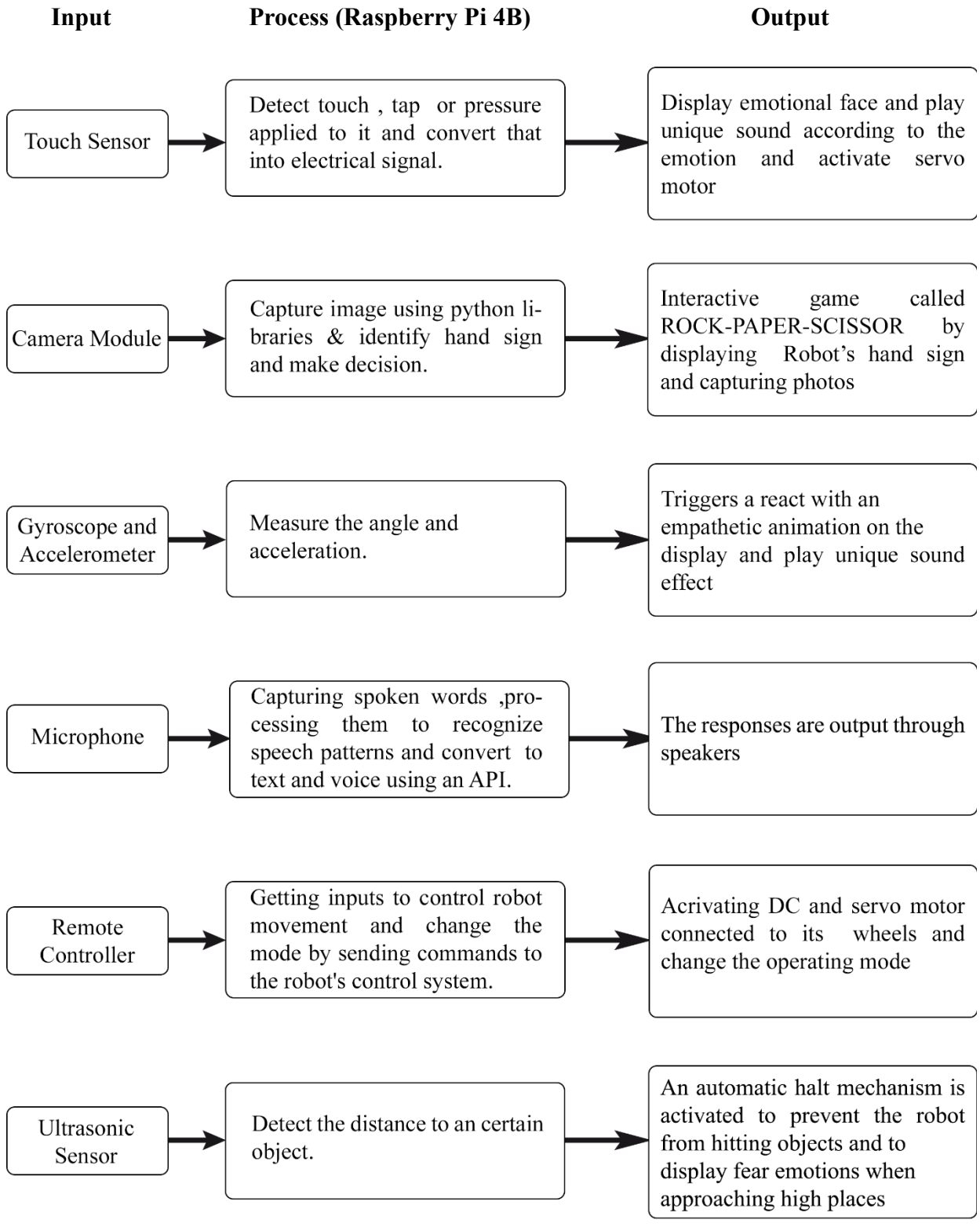


Figure 1 System Block Diagram

Schematic Diagram

- Luigi's Schematic Diagram

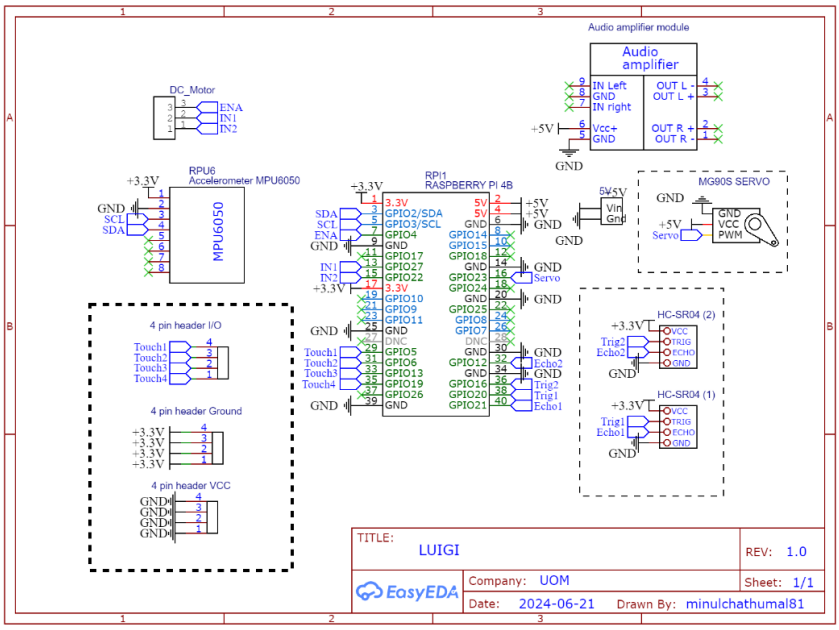


Figure 2 Luigi's Schematic Diagram

- Luigi's Remote Controller Schematic Diagram

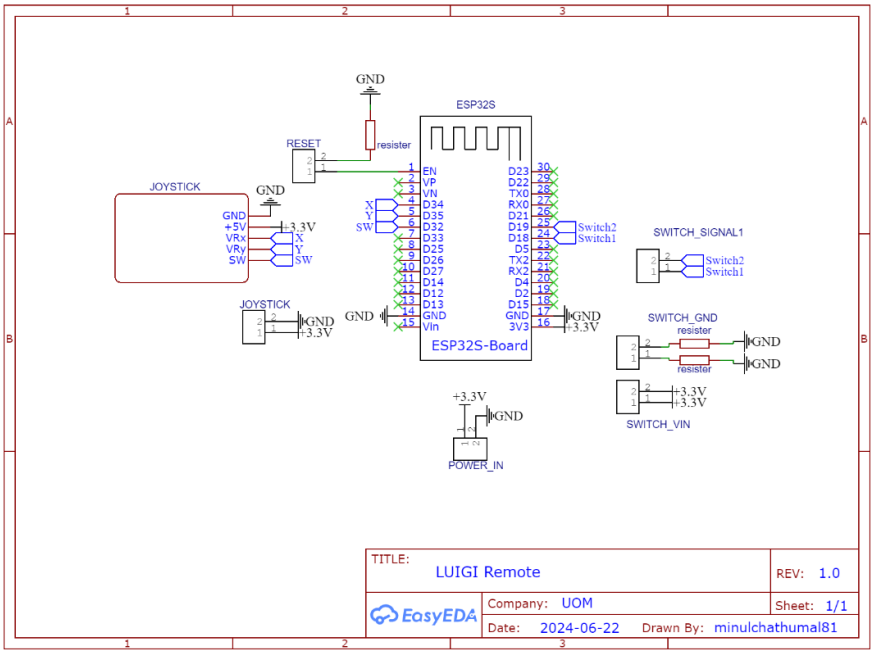


Figure 3 Luigi's Remote Controller Schematic Diagram

- Luigi's Remote Controller PCB Design

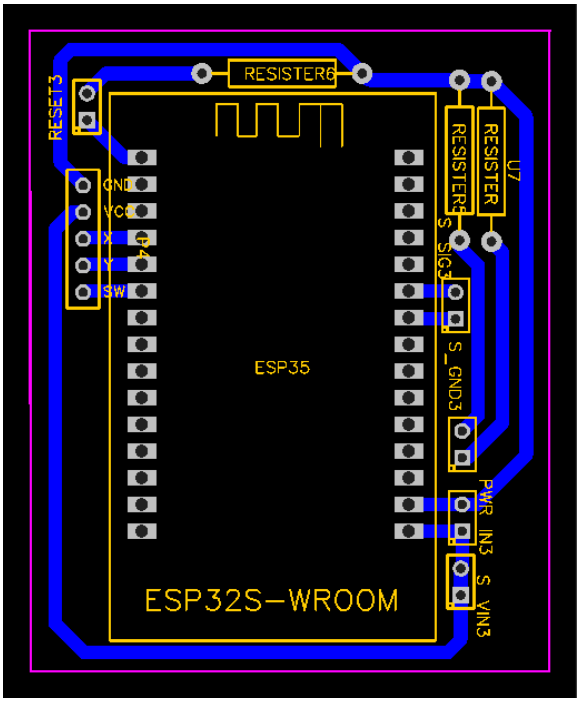


Figure 4 Luigi's Remote Controller PCB Design

- Luigi's Inner PCB Design

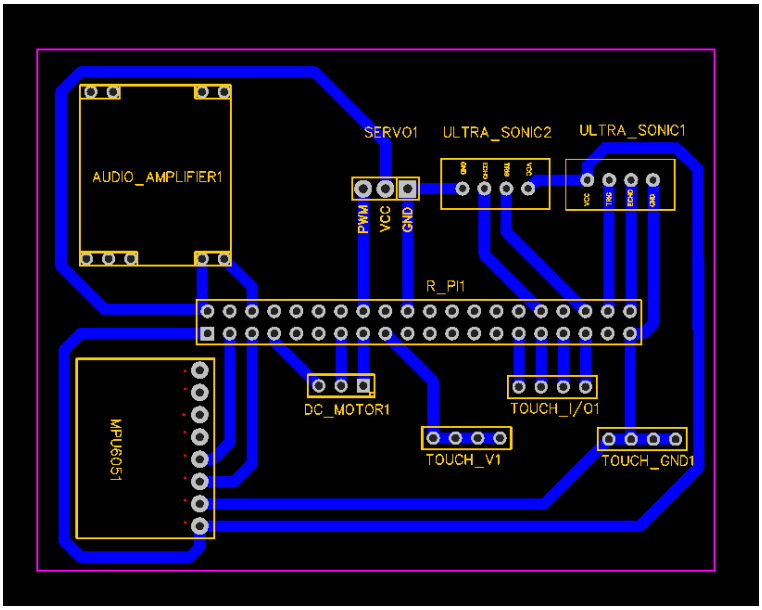


Figure 5 Luigi's Inner PCB Design

The 3D view

Figure 6 Luigi's 3D view

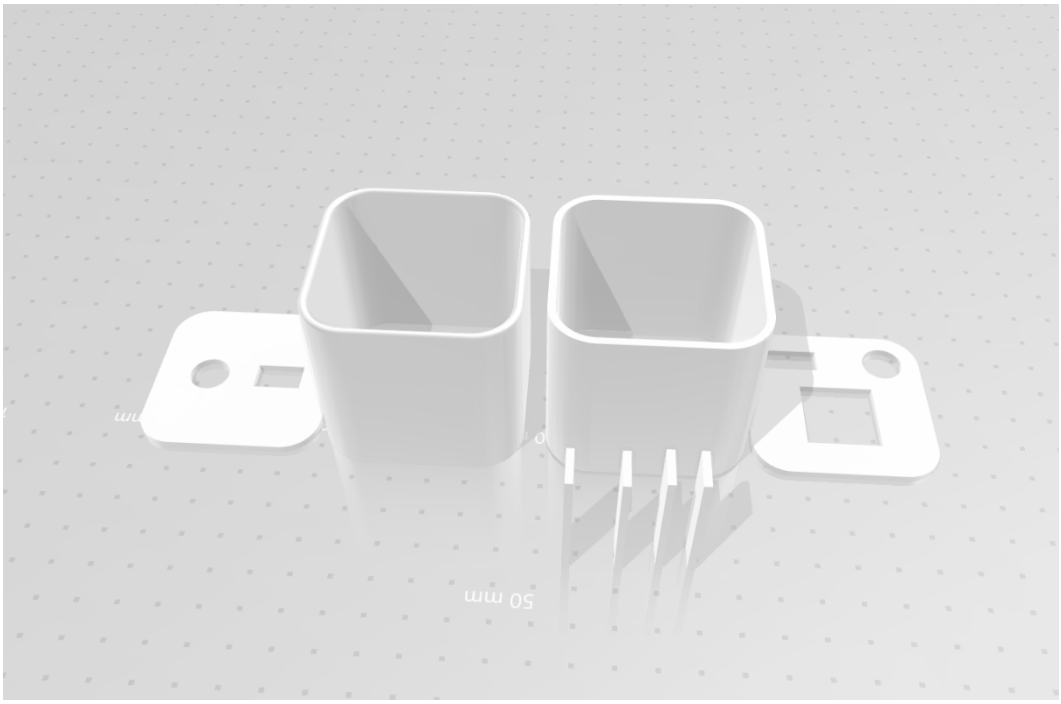


Figure 7 Remote controller 3D design

Components

Raspberry Pi 4 Model B

Figure 8 Raspberry Pi 4 Model B

How it works

The Raspberry Pi 4 Model B is a powerful single-board computer containing all the essential elements that a normal computer would have with regard to general computation, DIY electronics projects, and many uses such as programming, robotics, home automation, and media centers. It integrates a quad-core ARM Cortex-A72 CPU, a Broadcom Video Core VI GPU, and different options for connectivity and expansion.

At its core, the Raspberry Pi 4 is a computer that runs on a Linux-based operating system, by default Raspberry Pi OS, and there is considerable open-source software support. The board is quite versatile, featuring USB, HDMI, and GPIO ports to which peripherals such as keyboards, mice, monitors, and external drives can be attached. It has GPIO pins available for manipulating hardware directly, hence giving it a purposeful use in electronics projects. The Pi 4 Model B also includes built-in Gigabit Ethernet and dual-band Wi-Fi networking, together with Bluetooth for wireless communication with other devices. Major boosted improvements in the Pi 4 are huge bumps in performance, especially for CPU and GPU power, supporting up to two 4K displays simultaneously, courtesy of the new dual micro-HDMI ports. In fact, the Raspberry Pi 4 is not only perfect for learning to program but also for more advanced uses, like AI, IoT projects, and media streaming.

Specification	Value
Processor	1.5GHz Quad-Core ARM Cortex-A72
RAM	2GB
GPU	Broadcom Video Core VI
Networking	Gigabit Ethernet, 802.11ac Wi-Fi (dual band), Bluetooth 5.0
Video Output	2x micro-HDMI (4K output)
Audio Output	3.5mm audio/composite video jack
Storage	microSD card slot
GPIO Pins	40-pin header for general I/O
Power Input	5V DC via USB-C, 3.0A
Operating System	Raspberry Pi OS (Linux-based)
Temperature Range	0°C to +50°C
Weight	46g
Size	85.6mm x 56.5mm x 17mm

ESP32S Wi-Fi Bluetooth Dual Mode Microcontroller Module

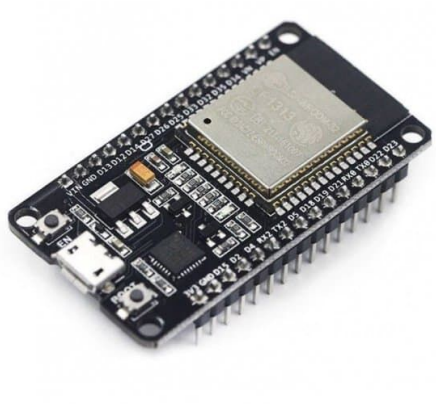


Figure 9 ESP32S Wi-Fi Bluetooth Dual Mode Microcontroller Module

How it works

- The ESP32S Wi-Fi Bluetooth Dual Mode Microcontroller Module is a very efficient, powerful development board that integrates both Wi-Fi and Bluetooth capabilities into one single module; hence, well adapted to IoT projects, smart devices, and embedded

systems. ESP32S is based on the ESP32 microcontroller from Espress if Systems, which features a dual-core processor with multiple peripheral interfaces.

- Microcontroller: The ESP32S has an XMOS XS1-L dual-core 32-bit LX6 microprocessor running up to 240 MHz, coupled with a large volume of processing power. This will enable advanced functions such as real-time data processing, sensor reading, and communications. The onboard processor can be programmed using one of the RTOS or run stand-alone applications in languages like C, C++, or Python using MicroPython.
- Wi-Fi and Bluetooth: It supports both 2.4 GHz Wi-Fi-802.11 b/g/n and Bluetooth-Classical and BLE, meaning Bluetooth Low Energy. ESP32S is a dual-mode device in that it can act like a networking hub for IoT devices. It will mean wireless communication with the internet using Wi-Fi and direct communication with other devices using Bluetooth. It can operate as a Wi-Fi client or access point and act in some applications as a mesh network node.
- GPIO and Peripherals: The ESP32S has several GPIO pins that can be used for input or output purposes, including but not limited to digital input/ output, PWM, ADC, DAC, and I2C/ SPI/ UART communication. Thus, it is easily interfaced with sensors, motors, displays, and generally with any peripheral.
- The ESP32S is a low-power-consuming chip. It houses several power-saving modes that enable it to go into deep sleep for extended periods and only turn on when required to work. This will make this chip useful in applications that are either battery-powered or energy-efficient.
- The development and programming of the ESP32S enjoy huge support in popular development environments, such as Arduino IDE, ESP-IDF, and Platform IO. Since it interactively supports libraries for Wi-Fi, Bluetooth, and peripheral control, programming the ESP32S is easy for beginners and advanced developers.

Specification	Value
Current Consumption	~160mA
Operating Voltage	3.0V to 3.6V
Processor	Xtensa® 32-bit LX6 dual-core (or single-core) processor

Storage	MicroSD card support via SDIO
Networking	IEEE 802.11 b/g/n (2.4 GHz) Bluetooth v4.2 BR/EDR and
GPIO	34- GPIO pins
Input Voltage	5V via USB or 3.3V via Vin pin
Operating Temperature	-40°C to +85°C
GPIO Power Output	40mA
Dimensions	25.5mm x 18mm x 3mm
Weight	~8g

TTP223 1-Channel (Red) Digital Capacitive Touch Sensor Module

Figure 10 TTP223 1-Channel (Red) Digital Capacitive Touch Sensor Module

How the touch Sensor works

- TTP223 1 Channel Capacitive Touch Sensor Module uses capacitive sensing technology to detect touch input. Its two electrodes replace one traditional mechanical button, making it ideal for projects requiring a modern and sleek user interface. When a finger or a conductive object approaches the sensor's touchpad, it will lead to an output signal because the capacitance is changed.
- Capacitive Sensing: The TTP223 module makes use of capacitive sensing. When an electrical field is modified around the touchpad, usually by an outgoing finger or any other conducting object, a change in capacitance coupled is detected by the internal circuitry of the sensor. Triggered by this action, the output toggles its state from LOW to HIGH or vice versa, which again depends on the mode

- Modes of Operation: The TTP223 module can operate in two modes:
  i. Momentary Mode: In this mode, the output remains HIGH only as long as the touch is detected. Once the touch is released, the output returns to LOW.

ii. Toggle Mode: In toggle mode, the output toggles between HIGH and LOW with each touch. The first touch sets the output HIGH, and the next touch sets it LOW, like a toggle switch.

- Sensitivity: TTP223 is sensitive and can easily be adjusted by changing environmental or component groups around it, such as adding/removing a ground plane/shielding.
- Power Efficiency: The module is highly power-efficient with very low consumption in stand-by mode. Hence, it is suitable even for battery-powered devices. Both Low-Power and Fast Mode configurations are supported.

Ease of Use: It is very easy to apply the TTP223 module in one's project. It usually contains three legs for VCC, connecting power; GND, connecting ground; and OUT, digital output. The OUT pin is used to connect with a microcontroller or any other circuits to trigger an action when a touch has been detected.

Specification	Value
Current	1.5µA to 3mA
Operating Voltage	2.0V to 5.5V
Operating Temperature	-30°C to 70°C
Response Time	60ms to 220ms
Dimensions	15mm x 11mm
Interface Type	Digital output (high/low signal)
Weight	~2g
Output State	Toggle mode or momentary mode
Touch Area	~9mm x 9mm

LCD module Pi TFT 3.5-inch (320×480) Touchscreen Display Module

Figure 11 LCD module Pi TFT 3.5-inch (320×480) Touchscreen Display Module

How the LCD module works

- Pi TFT 3.5-inch LCD Module is a touchscreen display module for the Raspberry Pi, designed. It provides an interface to operate the Raspberry Pi without a keyboard and mouse, which also excludes a monitor. In addition, it comes with an integrated module of a high-resolution display with a resistive touch screen for both visual output and touch input.

- Display Interface: The Pi TFT 3.5-inch LCD module, in general, connects with the Raspberry Pi via its GPIO header, using SPI for data transfer. The screen will interpret data sent by the display driver across the SPI bus and display the same. This effective utilization of the SPI interface can be used for higher transfer rates, therefore allowing smooth and responsive screen updates.

- Touch Interface: The nature of touch, as may be expected on this module, is resistive because it senses pressure applied to the screen. A touchscreen, therefore, would work quite well with stylus-based input and gives precise control. When this screen is touched, it reads the coordinates of the touch point, which can then be processed by the Raspberry Pi to trigger actions in software.

- Resolution and Size: The standard display is usually 3.5 inches in size, usually with a 480x320 pixels resolution. This will be very clear, colorfully rich visuals for applications involving pictures, videos, or projects needing GUIs, like media players, smart devices, or embedded systems. Its size makes it ideal for projects where space is limited but where a visual interface is required.

- Backlight and Power: The module includes an LED backlight to make sure visibility is easy in low-light conditions. The power to the display and the backlight usually comes directly from the Raspberry Pi. Therefore, it is easy to integrate without needing extra external power sources.

- Compatibility Software: The Pi TFT module supports a handful of libraries and drivers that let it be used with popular programming environments like Python. These libraries handle all communication with the display and touch for you, making setup easy to use in many things.

Specification	Value
Display Type	TFT LCD
Screen Size	3.5 inches
Resolution	320x480 pixels
Touch Type	Resistive or Capacitive
Operating Voltage	5V
Current Consumption	~100mA
Weight	~2g
Operating Temperature	-20°C to 70°C
Dimensions	~85mm x 56mm x 12mm
Pin Header:	26-pin GPIO header

HC-SR04 4Pin Ultrasonic Sensor Module

Figure 12 HC-SR04 4Pin Ultrasonic Sensor Module

How it works:

The HC-SR04 Ultrasonic Sensor is one of the most used distance-measuring sensors, using ultrasonic waves to calculate the distance between a sensor and an object. Using a transmitter and receiver. It works on the simple principle of emitting an ultrasonic sound wave from the transmitter at a resonating frequency of 40 kHz. It then travels through the air, is reflected by an object, and returns to the sensor it has been detected by the receiver. Trigger and Echo Pins: HC-SR04 comes up with four pins: VCC, GND, TRIG or trigger, and ECHO or echo.

By setting the distance, a pulse is sent to the TRIG pin-the sensor then sends an ultrasonic pulse. The ECHO pin then goes HIGH when the pulse is sent and remains HIGH until the reflected sound wave is detected. The duration for which the ECHO pin remains HIGH is proportional to the time it took for the sound wave to travel to the object and back. How the Sensor Works out the Distance: The sensor measures the time of flight of the sound wave to and from the object. Since the speed of sound is already known, approximately 343 m/s in air at room temperature, the distance can therefore be computed with the formula below.

Distance = [Time in microseconds × Speed of Sound] ÷ 2

Specification	Value
Operating Voltage	5V DC
Operating Current	15mA

Ultrasonic Frequency	40kHz
Range	2cm to 400cm
Measuring Angle	15° cone
Trigger Input Signal	10µs TTL pulse
Trigger Input Signal	10µs TTL pulse
Operating Temperature	-15°C to +70°C
Dimensions	45mm x 20mm x 15mm
Weight	~9g

Camera Module v1.3 5MP 1080p 720p

Figure 13 Camera Module v1.3 5MP 1080p 720p

How it works

- Overview The Camera Module v1.3 is a high-definition camera intended to operate with the Raspberry Pi. The sensor on the module has a resolution of 5MP, allowing it to capture still pictures and record video at 1080p or 720p. The module connects to the Raspberry Pi using a CSI, or camera serial interface. It is a compact and effective way to add visual functionality to projects.

- Image Sensor: The OmniVision OV5647 sensor is mounted on the Camera Module v1.3 and has 5 MP resolution. It is designed to capture high-quality still images. Along with this, it also offers different resolution video captures. The sensor comes with a Bayer filter that detects RGB light on the same plane and converts it into digital data which is then processed by Pi into an image or video.

- Connection Interface: The module is connected with the Raspberry Pi via a dedicated camera interface, the CSI connector. The bandwidth of this interface is very high, and it allows the camera to transfer image and video data directly to the GPU of the Raspberry Pi. This makes the camera very effective and efficient for real-time applications, like live streaming or computer vision.

- Lens and Focal Length: There is a fixed-focus lens on the Camera Module v1.3; this means the focal point of the camera is fixed, and not changed by any dynamic adjustment. The module does best in medium-distance settings of about 1 meter and, therefore, could be used for general purposes such as photography, video recording, and simple computer vision.

- Software Integration: The camera will be supported by the camera software stack of the Raspberry Pi, including the raspi-still command to capture still images and the raspivid command for capturing videos. It can also be employed with libraries like OpenCV for applications involving image processing, object detection, and face detection.

Specification	Value
Operating Voltage	3.3V
Operating Current	100mA
Image Sensor Type	OmniVision OV5647
Image Sensor Resolution	5 megapixels (2592 x 1944 pixels)
Field of View (FOV)	50 degrees
Connection	Ribbon cable
Operating Temperature	0°C to +50°C
Dimensions	25mm x 23mm
Weight	~6g

MPU-6050 Triple Axis Analog Accelerometer Gyroscope Module

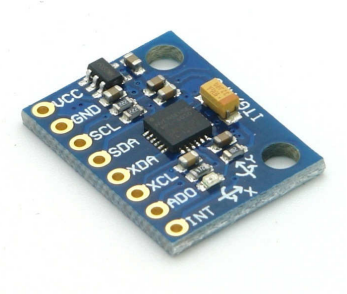


Figure 14 MPU-6050 Triple Axis Analog Accelerometer Gyroscope Module

How it works

The MPU-6050 is an integrated, 3-axis accelerometer and 3-axis gyroscope inside a package, one of the most common sensor modules. A combination like this greatly enhances its capability of measuring linear acceleration and rotational movement, thus making it perfect for applications such as motion detection, orientation tracking, and inertial navigation.

- Accelerometer: The MPU-6050 has a 3-axis accelerometer, which, at any given time, measures accelerations along the X, Y, and Z axes. It measures changes in velocity, as well as static acceleration like gravitational force. In this way, the sensor can find the device's orientation and detect movement or vibration.

- Gyroscope: This module also includes a 3-axis gyroscope, which measures the rate of rotation around the X, Y, and Z axes. It returns data on angular velocity useful in determining the orientation and the rotational motion of the sensor.

- Data Fusion: The MPU-6000 fuses outputs from both the accelerometer and gyroscope to provide complete motion data. This takes input from these components and digitally processes and fuses data using a Digital Motion Processor (DMP). The DMP can be used to calculate orientation, tilt angles, and much more. This digital motion processor reduces computational load on the microcontroller as most of the complex computation is done internally by the DMP.

- Communication Interface: The MPU-6050 communicates with the microcontroller or processor using the I2C, Inter-Integrated Circuit bus. Thus, the communication interfaces

can send and receive data by using only two wires-the data line SDA and clock line SCL-together with power VCC and ground GND. Here, the usage of the I2C interface makes the process of connection rather easy, with multiple devices sharing the same lines of communication.

Specification	Value
Operating Voltage	3.3V or 5V DC
Operating Current	3.9mA
Accelerometer Sensor Type	3-axis (X, Y, Z)
Gyroscope Sensor Type	3-axis (X, Y, Z)
Accelerometer Range	±2g, ±4g, ±8g, ±16g
Gyroscope Range	±250°/s, ±500°/s, ±1000°/s, ±2000°/s
Accelerometer Sensitivity	16384 LSB/g (for ±2g range)
Gyroscope Sensitivity	131 LSB/(°/s) (for ±250°/s range)
Operating Temperature	-40°C to +85°C
Dimensions	20mm x 15mm
Weight	~5g

USB Microphone Dongle Raspberry Pi PC

Figure 15 USB Microphone Dongle Raspberry Pi PC

How it works

A USB microphone dongle is a small module that connects with a USB input on a device and provides an audio input. Often, it will feature a built-in microphone and an ADC; the latter converts

sound waves into digital audio signals. The digital signal gets cross-transmitted via USB to the device it is connected to.

On a Raspberry Pi, it gets detected as an audio input device; you can configure the device using the ALSA sound system or other audio tools of the OS. This would be very helpful in voice recognition, recording audio, and communication applications. The microphone receives its power from the USB connection; thus, it does not require an additional power supply. With its compact form factor, you can easily implement it into portable or embedded systems.

Specification	Value
Operating Voltage	5V DC
Operating Current	50mA to 100mA
Microphone Type	Electret condenser or MEMS
Frequency Response	20Hz to 20kHz
Sample Rate	44.1kHz or 48kHz
Bit Depth	16-bit or 24-bit
Driver Requirements	standard Linux drivers
Operating Temperature	0°C to +50°C
Size	similar to a USB flash drive
Weight	~5g

Thumb Joystick Module

Figure 16 Thumb Joystick Module

How it works

The Thumb Joystick Module has two variable resistors-potentiometers-to detect the position of the joystick in the X and Y axes. As the joystick is moved, the resistance changes in these potentiometers, followed by a change in voltage output at the two pins X and Y. This analog voltage is proportional to the joystick position and can be used to determine the direction taken, the amount of movement, and how much it has been moved.

It's just a switch that closes when the joystick is pressed down. It is a digital output which can be read as HIGH or LOW depending if the button is pressed or not.

The analog outputs are usually connected to an ADC on a microcontroller or microprocessor that will read the position of the joystick. Digital output from the push button can be directly read as a binary state.

Specification	Value
Operating Voltage	5V DC or 3.3V
Operating Current	10mA to 20mA
Interface	Analog output and digital output
Axes	X-Axis: Measures horizontal movement Y-Axis: Measures vertical movement
Sample Rate	44.1kHz or 48kHz
Resolution:	X and Y Axes: 10-bit ADC resolution Push Button: Digital ON/OFF
Potentiometer Resistance	10kΩ to 50kΩ
Size	30mm x 30mm
Weight	~10g

LED 5mm

Figure 17 LED 5mm

How It Works

- The light comes from current passing through the chip, and the LED is a semiconductor light source. The 5mm LED includes a chip in a 5mm diameter plastic housing. There is a certain operating principle of an LED called electroluminescence, which has worked in producing light energy from electrical energy.

- Anode and Cathode: The LED has two leads, with the anode positive and the cathode negative. When current flows through from the anode to the cathode, the LED will emit light.

- Forward Voltage: Depending on color and type, most LEDs require a different forward voltage, often in the range between 1.8 and 3.3 volts, so that the correct functionality is guaranteed.

- Current Limitation: Normally, to prevent the damaging of LEDs, some current-limiting resistor or circuit exists to regulate how much current will flow through an LED.

Specification	Value
Operating Voltage	1.8V to 3.3V
Operating Current	5mA to 20mA
Luminous Intensity	~5000 mcd

Size	5mm diameter
Operating Temperature	-40°C to +85°C
Viewing Angle	15° to 30°

PAM8403 3W Stereo Audio Amplifier Board

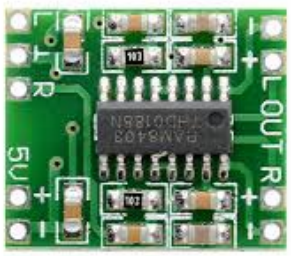


Figure 18 PAM8403 3W Stereo Audio Amplifier Board

How It Works

PAM8403 is a slim, highly efficient Class-D stereo audio amplifier with an operating voltage of 5V and capable of delivering 3W per channel. This integrated circuit is designed to amplify audio signals with high efficiency; therefore, it is suitable for small audio devices powered by batteries. Unlike other traditional amplifier classes that waste a lot of energy in the form of heating, the PAM8403 uses PWM to achieve the amplification. In this approach, the audio signal is converted to a stream of high-frequency pulses. These pulses then fully turn the output transistors on and fully off in rapid succession. This approach minimizes the power loss of the amplifier, and it generates little heat, adding to its suitability for portable applications. It receives the stereo audio signal on input pins and amplifies the signal strength to output pins to drive speakers or headphones. Operation from a low voltage power supply, typically from 2.5V to 5.5V, makes it suitable for common battery voltages. Despite the small size and low power consumption, the PAM8403 is designed to drive clear and powerful audio output with less than 0.1% total harmonic distortion and a signal-to-noise ratio above 80 dB. The module itself may be designed with further features such as volume control or low-pass filters, enabling better quality audio. Overall, PAM8403 is an effective and high-performance solution for compact audio amplification.

Specification	Value
Power Output	3W per channel (4Ω load)
Total Harmonic Distortion (THD)	< 0.1% at 1W (4Ω load)

Signal-to-Noise Ratio (SNR)	> 80 dB
Power Supply Voltage	2.5V to 5.5V
Current Consumption	~1.5A
Efficiency	~90%
Operating Temperature	-40°C to +85°C
Size	30mm x 20mm

4 Ohm 3W Speaker

Figure 19 4 Ohm 3W Speaker

How It Works

This is a 4-ohm, 3-watt speaker intended to convert electrical audio signals to waves of sound, and suitable for small to medium-sized audio. The speaker consists of a diaphragm, voice coil, and magnet assembly. An audio signal applied to the speaker will make its way through the voice coil, whereupon it develops a magnetic field interacting with the permanent magnet. In this case, the movement of the diaphragm back and forth, creating sound waves corresponding to an audio signal, depends on the interaction between the magnetic field and the flow of current.

- Impedance and Power Handling: The impedance of the speaker is "4 ohm," which is just the resistance of the speaker in front of an audio signal. In fact, 4-ohm impedance is very common in many speakers, striking a balance between power handling and efficiency. "3 watts" means that that is the maximum amount of power the speaker can handle without distortion or damage. If this power rating is exceeded, overheating may occur to such a degree that it could potentially damage your speaker.
- Frequency Response: It is the capability of a speaker to produce the sound properly. That shows the range of frequencies that it can accurately produce. While a basic specification

does not provide specific details of the frequency response, for most typical small speakers, the range can be from about 100 Hz to about 20 kHz, with sometimes variation depending on design and construction.

- Efficiency and sensitivity: This is about how much of the electrical power supplied is effectively converted into a sound output. Those speakers that have a higher sensitivity rating produce more sound output for the same amount of power, and hence would be effective for low-power applications.

Specification	Value
Impedance	4 ohms
Power Handling	3 watts maximum
Frequency Response	100 Hz to 20 kHz
Sensitivity	85-90 dB/W/m
Operating Temperature	-20°C to +70°C
Size	50mm to 80mm

MG90S Metal Gear Servo Motor

Figure 20 MG90S Metal Gear Servo Motor

How It Works

The MG90S is a metallic gear micro servo motor applied to precise control in the application field of angular position, speed, and acceleration. Its applications involve robotics, remote-controlled vehicles, and other systems that require very specific movement.

- Mechanism: The MG90S servo consists of a compact DC motor connected with a set of metal gears and with a feedback mechanism using a potentiometer. The rotational motion

derived from the DC motor acts through the gears onto precision angular movement. The potentiometer constantly keeps track of the position of the output shaft of the servo and provides feedback back to the control circuit.

- Control: It works on the principle that a servo motor is controlled by applying a PWM signal at its control input. Depending on the width of the pulse, the angle of the servo's output shaft will set it. Typically, a pulse width of 1 ms is associated with one end of the range of the servo, and a pulse width of 2 ms is associated with the other end. The servo moves to a position that corresponds to the width of the pulse it is receiving. Therefore, by controlling the pulse width, the angle of the servo can be precisely controlled.
- Metal Gears: Compared to plastic gears, the metal gears in the MG90S make the motor more durable and stronger, hence more reliable for applications that demand high torque.

Specification	Value
Impedance	4.8V to 6.0V
Operating Current	500mA to 700mA
Torque	0.10 seconds/60°
Angle Range	180°
Operating Temperature	-30°C to +60°C
Gear Material	Metal
Size	22.8 x 12.2 x 28.5 mm
Weight	13.4 grams

L298N Motor Driver Module

Figure 21 L298N Motor Driver Module

How It Works

The L298N is an integrated circuit used for both DC and stepper motor speed and direction control. It works on the principle of an H-bridge configuration, allowing a current to pass in either direction through the motor, hence its use for both forward and reverse control. The L298N can drive two DC motors independently or one stepper motor and handle up to 2A current load per motor, with voltages between 5V and 35V.

- Dual H-Bridge: L298N contains two H-bridges-one for each motor. This allows controlling of two motors simultaneously. In an H-bridge circuit, there are four transistors. It can allow a motor to run in both directions. When one pair turns on, the current flows in one direction; when the other pair turns on, the current runs in the opposite direction.
- PWM control can be explained as the application of PWM to the enable pins of the driver. In such a way, the effective voltage applied to the motor will change with the change in the PWM duty cycle, hence its speed. Enable pin: This enables each motor's full turn-on or turn-off.
- Logic Control: The L298N operates on two types of voltages-a logic voltage and a motor voltage. The logic voltage is usually at 5V and feeds the control circuitry while the motor voltage feeds up to 35V into the motors. These inputs are IN1 and IN2 for Motor A, IN3 and IN4 for Motor B, connected via a microcontroller or control system to set up the direction of the motors.
- Heat Management: The L298N is designed with a big heat sink to dissipate the generated heat since it can generate heavy heat in high load conditions or running bigger motors. Normally, the module requires proper ventilation or cooling to keep it running at high current values continuously.

Specification	Value
Operating Voltage	5V to 35V
Operating Current	2A per channel (peak 3A)
Total Power Dissipation	25W
PWM Frequency	Up to 25 kHz
Operating Temperature	-25°C to +130°C
Number of Channels	2 (dual H-bridge)

Size	43mm x 43mm x 26mm
Protection	Overheating, overcurrent, short-circuit

DC Motor 12V

Figure 22 DC Motor 12V

How It Works

This DC motor runs on direct current, converting electric energy into mechanical energy. The most important parts are a rotor commonly called the armature, a stator usually permanent magnets or an electromagnet, brushes, and a commutator. When a 12V power source is applied to the motor, which ought to have been supplied on the armature windings, a magnetic field develops. This magnetic field interacts with the stator magnetic field and creates rotational motion. It is arranged in such a way that there is the flow of current in the right direction to cause continuous rotation with the help of brushes and a commutator.

- Rotor and Stator: The former is the rotating part, while the latter is the stationary part. In a normal Brushed DC motor, the stator is made of permanent magnets or electromagnets, while the rotor consists of coils of wire-also called armature-through which current flows.
- Commutator and Brushes: The brushes come in contact with a commutator, which is a rotary switch that reverses the current in the armature windings at every half rotation of the motor. This switching is done constantly to ensure the torque on the rotor is always in one direction for its continuous rotation.
- Speed Control: The speed of a DC motor is directly proportional to the voltage applied at its terminals. Using a PWM signal allows the programmer to control the average voltage that is supplied to the motor and hence the speed.

- Direction control: The direction of the motor could be changed by reversing the polarity of the applied voltage. As convention goes, this is achieved with an H-bridge circuit implemented in several motor driver boards like the L298N.

Specification	Value
Operating Voltage	12V DC
Operating Current	0.5A to 1.5A
No-Load Speed	3000 to 6000 RPM
Torque	0.1 to 0.3 Nm
Operating Temperature	-20°C to +60°C
Efficiency	60% to 80%
Shaft Diameter	3mm to 5mm
Size	30mm x 50mm to 50mm x 80mm

Li-ion Battery BMS Charger Protection Board

Figure 23 Li-ion Battery BMS Charger Protection Board

How It Works

The Li-ion Battery BMS Charger Protection Board HX-35-01 is designed to manage and protect a lithium-ion battery pack with three 3.7V cells, 3S configuration. This module integrates in one unit several functions: a Battery Management System, maintaining the operation of the battery pack safely during charge and discharge, with protection features.

- Battery Management-the BMS monitors voltage and current of each cell in the battery pack. It works to keep each cell within specified operational limits and tries to prevent

overcharge, over-discharge, or overheating. The BMS performs cell balancing to equalize different voltage levels among the cells internally by redistributing the charge, thus improving performance and operating life of the batteries.

- Charging protection: HX-35-01 has a charging protection circuit to avoid overcharging of the battery pack. In such cases, if any cell reaches maximum safe voltage-usually around 4.2V per cell-the BMS would disconnect the charging circuit to avoid further damage.
- Protection at Discharge: The BMS, during discharge, monitors the voltage of each cell. If this falls to a minimum threshold-usually about 3.0V per cell-the BMS disconnects the load to avoid deep discharge, which affects the life span.

- Overcurrent Protection: Similarly, overcurrent protection is embedded in the BMS to prevent too high of a current draw, which could lead to overheating and/or damage to the battery pack. Should it go higher than the set threshold value (10A here), the BMS would disconnect the load to save the battery.
- Overheating protection: Some of the BMS boards ensure thermal protection to the battery pack by monitoring its temperature. When the temperature exceeds the threshold value specified as safe, the BMS will disconnect either the load or charging circuit to evade overheating and probable hazards.
- Balancing Function: It balances all the cells in a battery pack by equalizing their charge levels. This therefore permits better efficiency and performance of the battery, hence prolonging its life.

Specification	Value
Battery Configuration	3S
Nominal Voltage	3.7V per cell
Maximum Voltage per Cell d	4.2V
Minimum Voltage per Cell	3.0V
Charging Current	10A
Dimensions	50mm x 25mm
Operating Temperature	-20°C to +60°C
Overcharge Protection	Yes

Overcurrent Protection	Yes

3.7V 3200mA and 1800mA 18650 Li-ion Rechargeable Battery

Figure 24 Li-ion Rechargeable Battery

How It Works

These are the specific forms of cylindrical lithium-ion rechargeable batteries known as 18650 Li-ion, finding wide applications in devices, electric vehicles, and energy storage applications. In the two specifications of these batteries, namely 3.7V 3200mAh and 3.7V 1800mAh batteries, the nominal voltage and capacity of the battery, respectively, are stated.

Specification	Value
Nominal Voltage	3.7V
Capacity	3200mAh
Discharge Current	2A to 5A continuous
Charge Voltage	4.2V
Charge Current	1C (3200mA)
Discharge Cutoff Voltage	2.5V to 3.0V
Operating Temperature	-20°C to +60°C
Weight	45g to 50g
Temperature Range	-20°C to +60°C

Specification	Value
Nominal Voltage	3.7V

Capacity	1800mAh
Discharge Current	1A to 3A continuous
Charge Voltage	4.2V
Charge Current	1C (3200mA)
Discharge Cutoff Voltage	2.5V to 3.0V
Operating Temperature	-20°C to +60°C
Weight	40g to 45g
Temperature Range	-20°C to +60°C

LM2596S 3-40V to 1.5-35V 4A DC to DC Adjustable Step-Down Buck Module

Figure 25 LM2596S 3-40V to 1.5-35V 4A DC to DC Adjustable Step-Down Buck Module

How It Works

The LM2596S is a versatile DC-DC step-down/buck converter module that is very useful and, above all, efficient for converting a higher input voltage to a lower output voltage. This module utilizes an integrated circuit known as the LM2596S, one of the most popular choices for regulating power in modern electronic applications.

- Step-down Conversion: This is the primary function of the LM2596S module, whereby it steps down the input voltage into a desired lower output voltage. For example, an inputting of 12V can be adjusted at the module to a lower voltage output of 5V or 9V. This finds its application where a smaller stable voltage is required from a higher voltage source.
- Adjustable Output: The truth is that, typically, one should be able to adjust the output voltage on every LM2596S module in the range from 1.5 to 35V by using a variable resistor

on the module. In this regard, you will have full control over the output voltage. You can, therefore, set any desired output voltage simply by adjusting this potentiometer.

- Current Rating: The module is capable of handling 4A of continuous current. This makes it quite versatile in application, right from simple electronics up to somewhat heavier loads. Again, this current rating insinuates that the module will provide an output with stability while not compromising on efficiency.
- Efficiency: The LM2596S is designed for high efficiency, typically around 80-90%. The implication of this is that the greater part of input power is converted to output power with minimum dissipation into heat energy. The high efficiency helps reduce additional cooling needs and extends battery life in portable applications.
- Thermal Protection: The module has thermal protection so that cases of overheating can be avoided. Thus, on exceeding a certain temperature threshold, the module reduces the output power or switches off to protect itself and the connected load.
- Noise and Ripple: Generally, the LM2596S module has low output ripple and noise. Thus, it ensures that the power supply given to sensitive electronic components is proper and smooth.

Specification	Value
Input Voltage Range	3V to 40V
Output Voltage Range	1.5V to 35V
Output Current	Up to 4A
Efficiency	80-90%
Switching Frequency	150 kHz
Voltage Regulation	Adjustable via potentiometer
Operating Temperature	-40°C to +85°C
Dimensions	55mm x 30mm x 20mm
Weight	~50g

Testing and Implementation

Unit Testing

Unit testing of our project involved testing each component in isolation to ensure that it worked as expected. We first started the critical phase with the TTP223 1-Channel Capacitive Touch Sensor Module, by checking its capability to register a single and double tap correctly. Initially, the sensor was sensitive, so we did some settings on it and implemented debounce logic in the sensor response code to get it right.

Then, we worked on the Pi TFT 3.5-inch LCD Module for its compatibility with the Raspberry Pi, and correct outputs appeared on it. After some driver compatibility issues with this display, which were fixed by updating the firmware and doing some configurations in the system, it worked perfectly. For testing the accuracies of object detection and distance measurement, an HC-SR04 Ultrasonic Sensor Module was used. Initially inconsistent reading for high reflectivity scenarios appeared, which we curbed using an averaging function in our code that would smooth out the sensor placing.

To this end, a set of static and dynamic tests were conducted concerning the measurement capabilities of angular velocity and acceleration by the MPU-6050 sensor module. For the purpose of dealing with calibration drifts, we have set regular intervals for recalibration. Also, with considerations for dead zones and developing the firmware in ways to provide more sensitivity and responsiveness, precision in the thumb joystick to track X and Y coordinates was realized.

Servo Motor Test: In this, the servo motor was tested for its functionality to work at specific angles. This little deviation was removed by fine-tuning the control signals with the addition of compensation through software. Lastly, the motor driver and DC motors were performance-tested to control the speed and direction of movement. Performance variations due to different loads were offset by incorporating a dynamic feedback system for real-time adjustment.

Each of these being cleared out during unit testing lays a concrete foundation for integration and system-level reviews to come, ensuring robust performance and reliability of the whole system. This has helped not only in finding out and rectifying the issues but also in validating each component with respect to its readiness about the subsequent stages of the project development.

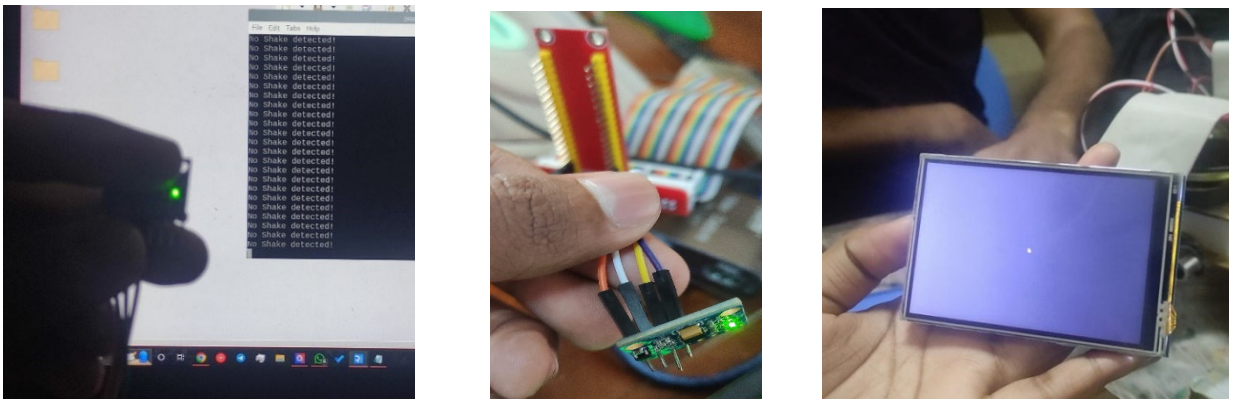


Figure 26Unit Testing

Integration Testing

Integration tests were targeted to make sure that all the different components of the system worked together. It started with integrating touch sensors and the Luigi interface where each of the touch sensors was programmed to trigger an only unique emotional sound and reaction on the interface. This integration was tested well to ascertain those sensors responded correctly and reliably to user input. Next, the ultrasonic sensors were interfaced with the servo motor. This interface allowed the servo to turn on for specified distances read from the ultrasonic sensor. Testing was conducted to verify that the servo engaged when the threshold distances were reached. Similarly, the second ultrasonic sensor was connected to the motor driver for operating a safety device.

This was tested with the intent of validating if the setup in case of distance readings from the sensor indeed triggered the mechanism correctly. The camera module was integrated with the display for recognizing and interpreting user hand gestures. Integration was tested to ensure correct detection of the gestures, which showed everything correctly on the screen. Also, with this display came the MPU6050 for more detailed motion feedback and for increasing the interaction capability, hence enhancing the user experience. The speaker inducted with the microphone would allow the user to interact with Luigi through assistant functions. This component required tests concerning the audio input by the user, which should be processed and responded to appropriately by the system. A joystick connected to the ESP32 was used as a remote controller. Tests would be run on inputs coming from it, ensuring that they were correctly interpreted and processed by the system for smooth remotely controlled functions.

Figure 27 Integration Testing

System Testing

System testing was the thorough checking of the entire setup to ensure everything worked in harmony. First, it started with the design and fabrication of the printed circuit board and attached all the components to it. This whole assembly was then attached to the base of the Luigi securely. Next came the integration between ESP32 and Raspberry Pi. It was achieved in such a manner that communication between the two microcontrollers became smooth.

The display has been used to control these functions, each function having an interface on the screen. In continuation, ESP32 related to the PCB by interfacing it with a joystick and two control buttons, designed for controlling the headlights and horns. These tests proved the correct processing of the joystick inputs, thus making it right to function properly with DC motors and servo motors.

Additionally, there is a battery management system that feeds the two microcontrollers, whose output is preconditioned at appropriate levels by a buck converter. This entire system has been tested to verify the correct operation of the power supply, the communication protocols, and the control functions, ensuring all the parts operate in this type of application according to the specifications defined for the project.

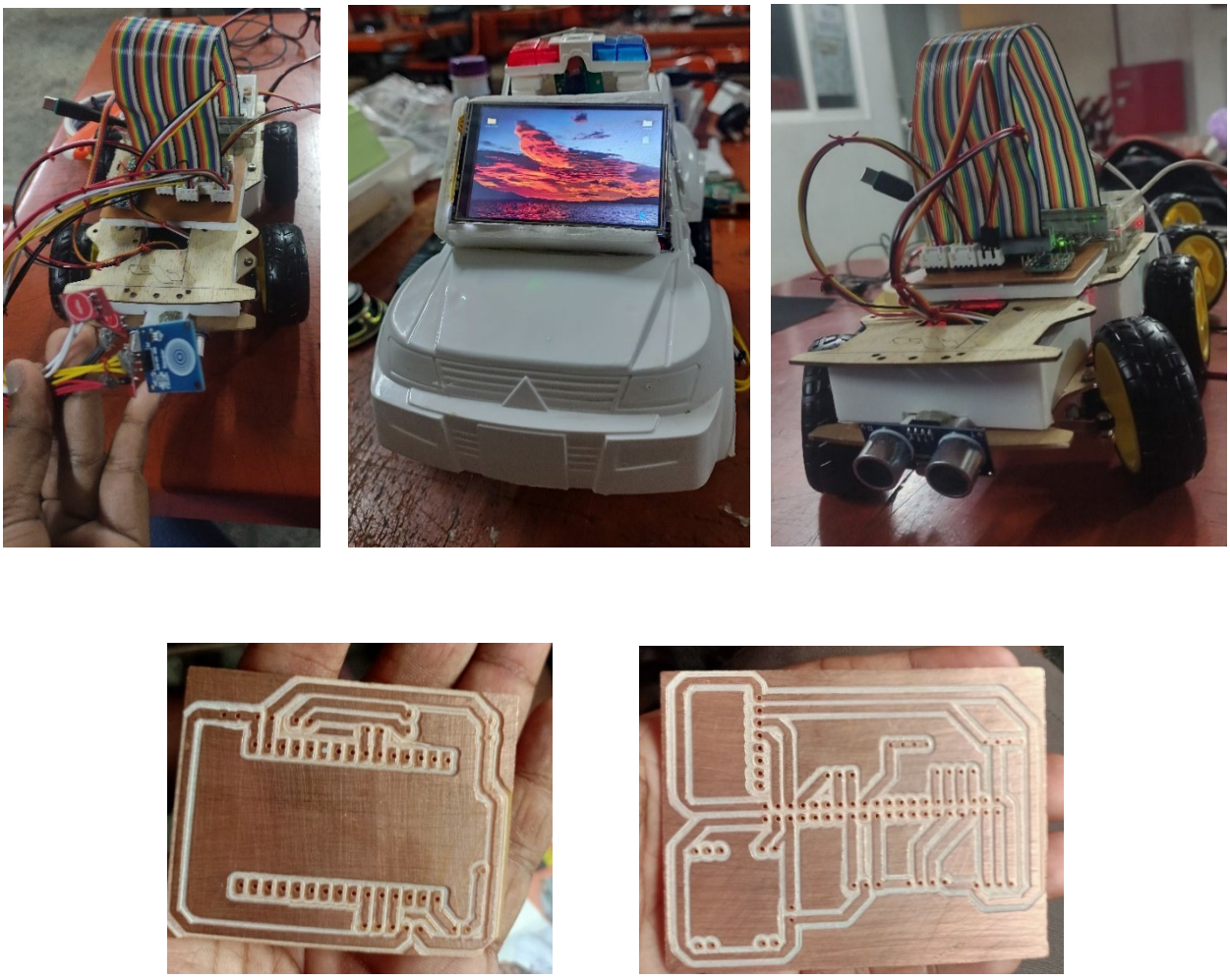


Figure 28 System Testing

Implementation

This stage of implementation included a few other important steps that were needed in bringing the project to the finishing point. Once all components were attached and secured, 3D designs were created and printed that finalized the structure of Luigi and the remote control, providing them with an attractive, polished look.

The system was further tested using a controller by making the necessary adjustments to speed by the motors, whose base was the readings of input. Testing the functionality of distance measurement was extensively made to ensure that safety features operated correctly.

Also, some other sensors, such as touch sensors, ultrasonic sensors, and the camera module, were adjusted for their integration to get reliable performances. The touch sensors must be calibrated for the right emotional response triggering, and ultrasonic sensors are used for distance

measurement so that the servo motor and safety mechanism move correctly. It also tested gesture recognition by the camera module, and indeed, the same conveyed appropriate hand movements.

The final system was checked for smooth operation. That is, all components fitted smoothly into each other to realize the specification of the project. It also updated documentation with instructions on how to set up, calibration, and troubleshooting to support future usage and maintenance.

Figure 29 Implementation

Individual Contribution

Nimhan R.D.S. - 224135V

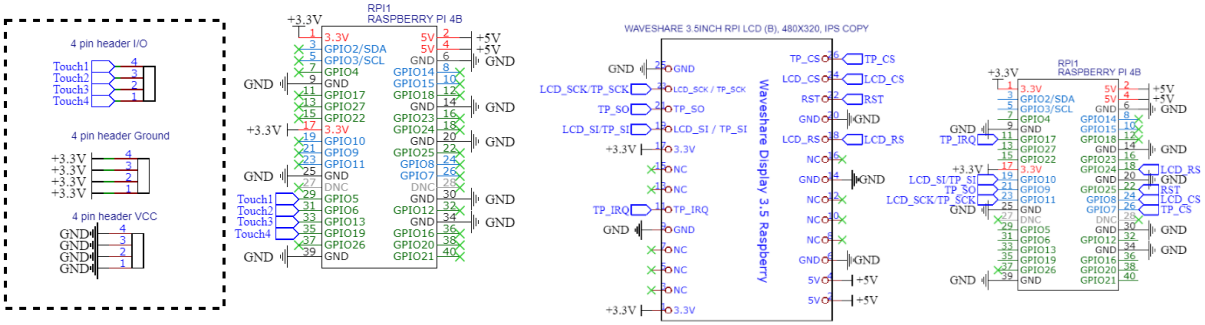


Figure 30 Nimhan R.D.S. - 224135V (Schematic diagram)

My core role in this project was developing the user interface and interaction with the pet robot, which was Luigi. I focused on writing the display output and integrating emotional feedback using sensory and visual elements. Following are the contributions that I provided in detail:

Design of Output Displaying:

I designed Luigi's user interface using the Tkinter library, which is well-suited for creating graphical user interfaces (GUIs). For example, I made it simple to switch between the robot's various functions, prioritizing ease of use. Tkinter's tools allowed me to incorporate intuitive layouts with visually appealing elements like color schemes, buttons, and shapes, ensuring that the design was both functional and aesthetically pleasing.

Additionally, I organized these features so that users could easily navigate and interact with Luigi, making the interface more 'user-friendly.' While the design focused on practicality, it also maintained a beautiful appearance, making the interaction with Luigi not only intuitive but also enjoyable. These design decisions significantly contributed to the overall user experience.

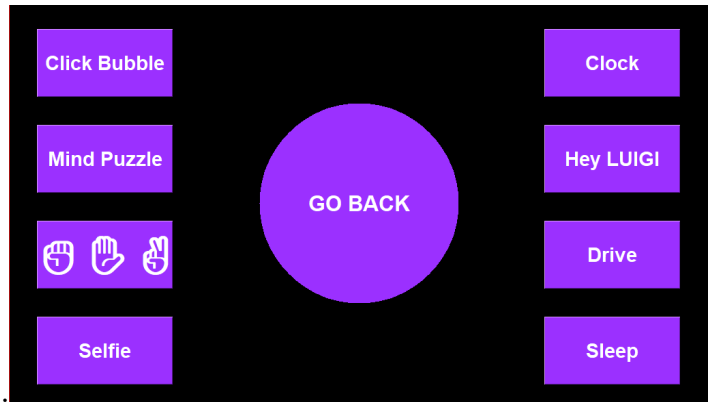


Figure 31 Luigi's Function Menu

```python
# Create the main window
root = tk.Tk()
root.title("Python File Launcher")
root.attributes("-fullscreen", True)  # Set fullscreen mode
root.geometry("1024x768")
root.configure(bg="black")

button_font = ("Helvetica", 22, "bold")
button_font2 = ("Helvetica", 48,"bold")

# Create buttons for the left side
left_button1 = tk.Button(root, text="Click Bubble",font=button_font, command=lambda: open_python_file('/home/pi/Desktop/LUIGI_Final/Functions/Games/Click Bubble.py'), bg="purple1", fg="white")
left_button1.place(x=40, y=130, width=200, height=100)
left_button1.bind("<Enter>", on_enter_button)
left_button1.bind("<Leave>", on_leave_button)

left_button2 = tk.Button(root, text="Mind Puzzle",font=button_font, command=lambda: open_python_file('/home/pi/Desktop/LUIGI_Final/Functions/Games/Mind Puzzle.py'), bg="purple1", fg="white")
left_button2.place(x=40, y=270, width=200, height=100)
left_button2.bind("<Enter>", on_enter_button)
left_button2.bind("<Leave>", on_leave_button)

left_button3 = tk.Button(root, text="✊ ✋ ✌", font=button_font2, command=lambda: open_python_file('/home/pi/Desktop/LUIGI_Final/Functions/Games/RPS/RPS.py'), bg="purple1", fg="white")
left_button3.place(x=40, y=410, width=200, height=100)
left_button3.bind("<Enter>", on_enter_button)
left_button3.bind("<Leave>", on_leave_button)

left_button4 = tk.Button(root, text="Selfie",font=button_font, command=lambda: open_python_file('/home/pi/Desktop/LUIGI_Final/Functions/Selfie/selfie.py'), bg="purple1", fg="white")
left_button4.place(x=40, y=550, width=200, height=100)
left_button4.bind("<Enter>", on_enter_button)
left_button4.bind("<Leave>", on_leave_button)

# Create buttons for the right side
right_button1 = tk.Button(root, text="Clock",font=button_font, command=lambda: open_python_file('/home/pi/Desktop/LUIGI_Final/HTML/digitalclock.py'), bg="purple1", fg="white")
right_button1.place(x=784, y=130, width=200, height=100)
right_button1.bind("<Enter>", on_enter_button)
right_button1.bind("<Leave>", on_leave_button)
```

Figure 32 Tkinter code

Implementation of Expressions of Emotions:

In order to provide emotional feedback to Luigi, I made use of a combination of HTML/CSS, Javascript and the WebView Python library. This allowed for dynamic and emotive display. Each feeling was done with a video and sound combination: happy, sad and love. Since HTML does not support auto-play on audio, I hooked it up with the Pygame library to better orchestrate it through sound. This allowed for perfect synchronization of visual and audio in emotional responses for greater immersion. Designing those emotional expressions to pop out of him had to be drawn in relation to Luigi's personality and thus to create an immediate strong bond between the robot and its user. In such a way, the system would be able to respond to many kinds of input and thus create an emphatic, almost life-like interaction that is very personal for the user.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Video Player</title>
    <style>
        html, body {
            margin: 0;
            padding: 0;
            width: 100%;
            height: 100%;
            overflow: hidden;
        background-color:#000000;
        }

        #videoPlayer {
            width: 100%;
            height: 100%;
        }
    </style>
</head>
<body>
    <video id="videoPlayer" autoplay muted>
        <source id="videoSource" src="" type="video/mp4">
        Your browser does not support the video tag.
    </video>

    <script>
        // Function to set the video source and handle looping
        function setVideoSource(videoPath) {
            var video = document.getElementById('videoPlayer');
            var source = document.getElementById('videoSource');

            // Set new video source
            source.src = videoPath;
            video.load();
            video.play();

            // Loop playback until a new video is set
            video.onended = function() {
                setVideoSource(videoPath); // Loop the same video
            };
        }

        // Initial video source setting
        setVideoSource('');

        // Toggle fullscreen function
        function toggleFullscreen() {
            var video = document.getElementById('videoPlayer');
            if (video.requestFullscreen) {
                video.requestFullscreen();
            } else if (video.mozRequestFullScreen) { /* Firefox */
                video.mozRequestFullScreen();
            } else if (video.webkitRequestFullscreen) { /* Chrome, Safari and Opera */
                video.webkitRequestFullscreen();
            } else if (video.msRequestFullscreen) { /* IE/Edge */
                video.msRequestFullscreen();
            }
        }

        // Listen for messages from Python to change video
        window.addEventListener('message', function(event) {
            if (event.data.type === 'change_video') {
                setVideoSource(event.data.videoPath);
            }
        });
    </script>
</body>
</html>
```

Figure 33HTML,CSS and Js code

Touch Sensors Integration for Emotional Feedback:

I was able to add an interactive layer to Luigi by installing three touch sensors that would allow users to physically elicit emotional responses. Accordingly, I mapped each sensor onto the happiness of, the sadness of, and the love of every emotion. These sensors then sent a video and unique sounds when activated, which I was able to synchronize into an entertaining emotional reaction. This uses a combination of tactile input combined with audio-visual feedback that can help make users' actions much more meaningful and playful while interacting with Luigi. These emotions were in real time to the user for instant feedback and to establish a rapport between Luigi and its users.

```
# Function to handle additional touch sensors
def handle_additional_touch(pin):
    global additional_sound_playing
    try:
        while True:
            if GPIO.input(pin) == GPIO.HIGH:
                if not additional_sound_playing[pin]:
                    print(f"Additional touch sensor on pin {pin} activated!")

                    # Change the video in the webview based on the touch sensor activated
                    video_path = additional_video_paths.get(pin, main_video_path)
                    webview.windows[0].evaluate_js(f"setVideoSource('{video_path}')")

                    # Play the additional sound for 8 seconds
                    additional_sound_playing[pin] = True
                    additional_sounds[pin].play()
                    start_time = time.time()
                    while time.time() - start_time < 8:
                        time.sleep(0.1)
                    additional_sounds[pin].stop()
                    additional_sound_playing[pin] = False

                    # Wait for 10 seconds before returning to main video
                    time.sleep(1)
                    webview.windows[0].evaluate_js(f"setVideoSource('{main_video_path}')")

                # Debounce delay
                while GPIO.input(pin) == GPIO.HIGH:
                    time.sleep(0.01)
            time.sleep(0.01)
    except KeyboardInterrupt:
        print("Exiting program")
    finally:
        GPIO.cleanup()
```

Figure 34 Function to emotional sound and video

Advanced Double-Tap Functionality:

Complementing the emotional touch sensors, I implemented a fourth sensor that detects double taps, used as a method for function changes or exiting any present mode. Because a normal touch sensor could only recognize a single tap, I built a function that counted how many taps happened in 0.5 seconds. If the user tapped twice within the timer, it registered it as a double tap and ran the relative function-twice for switching modes or stopping any current operation. This feature added

versatility to the control system without compromising ease of use, which does not need to include more buttons or complex interfaces.

```python
# Function to start the double tap detection in a separate thread
def start_double_tap_detection():
    try:
        last_tap_time = 0
        tap_count = 0
        DOUBLE_TAP_INTERVAL = 0.5  # Adjust as needed

        def detect_double_tap():
            nonlocal last_tap_time, tap_count
            while True:
                if GPIO.input(TOUCH_PIN) == GPIO.HIGH:
                    current_time = time.time()

                    if (current_time - last_tap_time) < DOUBLE_TAP_INTERVAL:
                        tap_count += 1
                    else:
                        tap_count = 1  # Reset tap count if the interval is too long

                    last_tap_time = current_time

                    if tap_count == 2:
                        subprocess.Popen(['python3', '/home/pi/Desktop/LUIGI_Final/Function.py'])
                        print("Double tap detected! Closing game...")
                        pygame.quit()
                        GPIO.cleanup()
                        os._exit(0)

                    # Debounce delay
                    while GPIO.input(TOUCH_PIN) == GPIO.HIGH:
                        time.sleep(0.01)

                time.sleep(0.01)  # Polling interval

        # Run the double tap detection function in a separate thread
        detect_thread = threading.Thread(target=detect_double_tap)
        detect_thread.start()

```

Figure 35 Double tap detection function

Creation of Video and Audio Footage:

I have created the videos and sound effects for his emotions, creating or changing each one to more specifically fit the emotional context when the touch sensors are being activated. To enhance this emotive output further, all the videos had synchronized sound effects, which helped draw things together and make Luigi's visual reactions more immersive and realistic. Audio was in WAV format, and videos were in MP4 format for the best quality and compatibility during playback.

Figure 36 Video footages

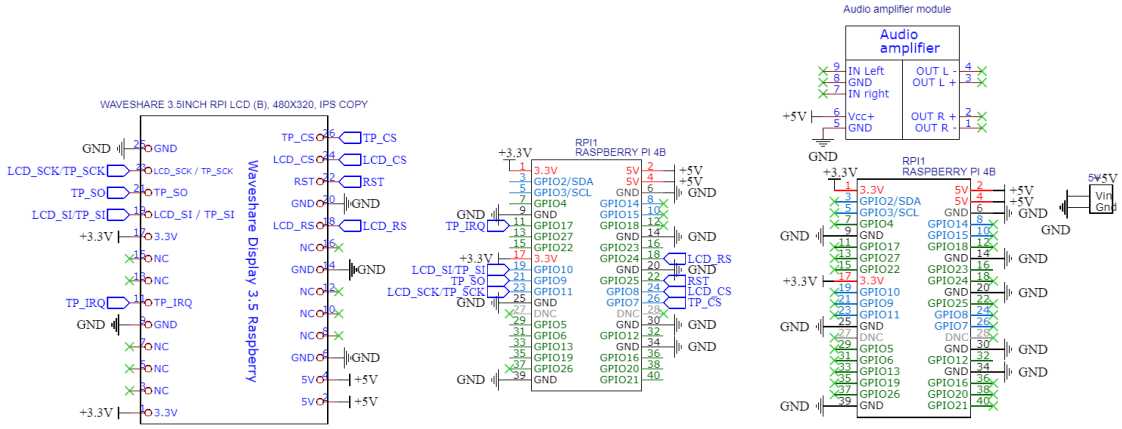


Figure 37 Gunawardhana T.U.D. 224063X (Schematic diagram)

Raspberry Pi Configuration

1. Operating system setup

I was responsible for the entire process of setting up the operating system on the Raspberry Pi. My contributions included:

a. Selecting the OS:
- I evaluated and chose the Raspberry Pi OS (Legacy-Bullseye, 64-bit) to ensure it met the project's requirements.

b. Creating a Bootable SD Card:
- I downloaded the appropriate OS image from the Raspberry Pi website.
- I utilized Raspberry Pi Imager to create a bootable SD card, ensuring that the installation media was correctly prepared.

c. Initial Setup Configurations:
- Inserted the SD card into the Raspberry Pi and completed the initial setup.
- I configured essential settings, including network setup, system updates, and user account creation.

d. Optimization:
- I fine-tuned the OS for optimal performance by adjusting system settings.
- I confirmed that the OS was fully compatible with the required drivers and applications for our project.

For further details on Raspberry Pi OS setup, I referred to the official Raspberry Pi documentation: Getting Started with Raspberry Pi.

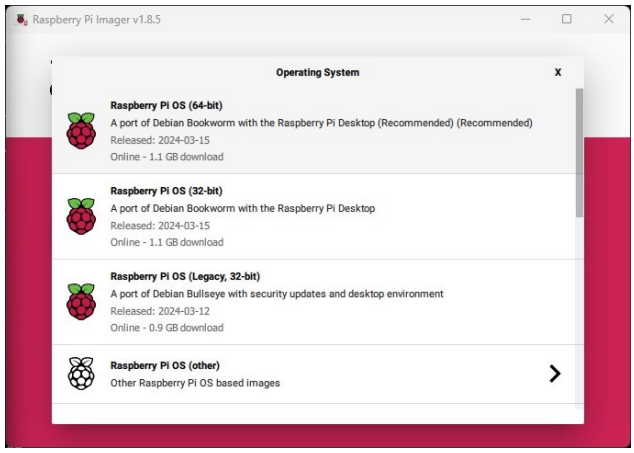


Figure 38Raspberry Pi OS setup

2. Python environment and library setup
   I configured the Python environment by installing the necessary Python version and setting up a virtual environment for the project. Additionally, I installed and managed the required libraries and dependencies, such as OpenCV, NumPy, GPIO, etc. libraries, to support the project's coding requirements.

3. Wireless Configuration setup
   - SSH Configuration
     I set up SSH access to the Raspberry Pi from Windows using Command Prompt.

     - Enable SSH on the Raspberry Pi:
       - Open the terminal on the Raspberry Pi.
       - Type sudo raspi-config and press Enter.
       - Go to Interfacing Options > SSH > Enable.
       - Exit and reboot if needed.
     - Find the Raspberry Pi's IP Address:
       - In the terminal, type hostname -I and note the IP address.
     - Connect from Windows:
       - Open Command Prompt.
       - Type ssh USERNAME@<IP_ADDRESS or HOSTNAME>
       - Press Enter and type the password raspberry when prompted.

       Summary:

       - Use Name: pi
       - Host Name: raspberrypi
       - Password: raspberry

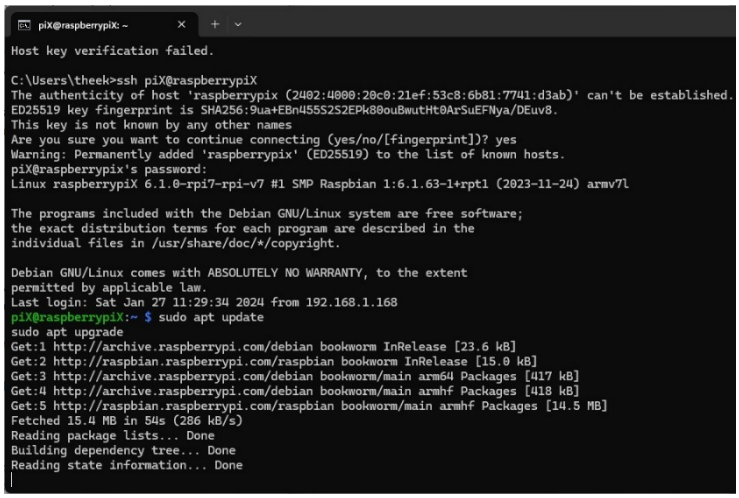


Figure 39 Wireless Configuration setup

- VNC Configuration

  I enabled VNC through the command line to allow remote desktop access, making it easier to manage and control the Raspberry Pi without needing a direct monitor connection.

```
# Installing VNC server
sudo apt install realvnc-vnc-server

# Enabling VNC server through raspi-config
sudo raspi-config
# Navigate to: Interface Options -> VNC -> Enable

# Start VNC server
vncserver

# Check VNC server status
systemctl status vncserver-x11-serviced.service
```

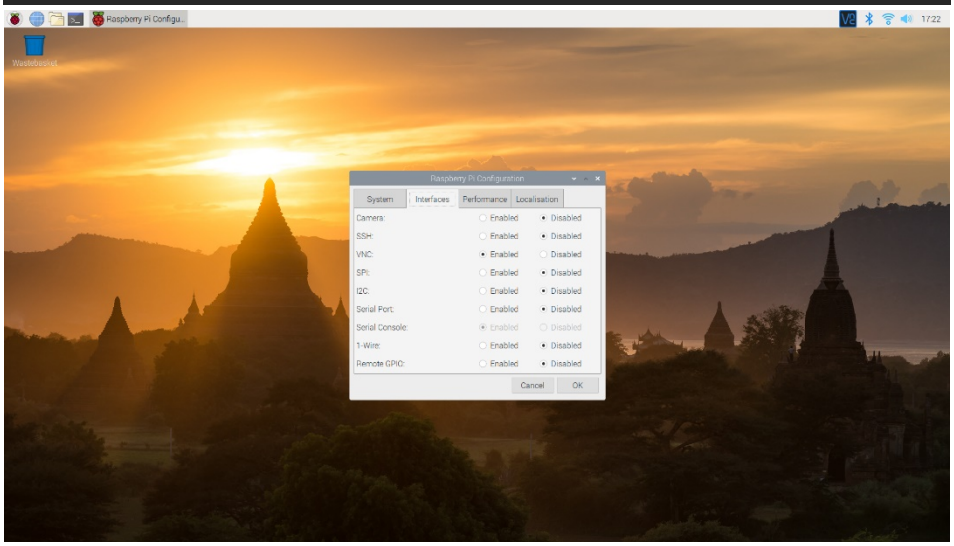


Figure 40 VNC Configuration

Display Configuration

1. Drivers' installation

   I installed the necessary drivers for the 3.5-inch LCD display by following the instructions provided on LCD Wiki and Waveshare. This involved executing specific installation scripts to ensure the display is recognized and functions properly with the Raspberry Pi.

2. Touch Calibration

   I calibrated the touch screen using the xinput-calibrator tool to ensure accurate touch input. This process involved running the calibration utility, which helps align touch points with the display coordinates for precise interaction.

The relationship between rotation Angle and resistance touch parameters		
Type / Angle	2.4, 2.8, 3.2-inch	3.5-inch（MPI3501）
rotate=0	"155 3865 115 3700" "0"	"268 3880 227 3936" "0"
rotate=90	"3700 115 155 3865" "1"	"3936 227 268 3880" "1" (Default Orientation)
rotate=180	"3865 155 3700 115" "0"	"3880 268 3936 227" "0"
rotate=270	"115 3700 3865 155" "1" (Default Orientation)	"227 3936 3880 268" "1"

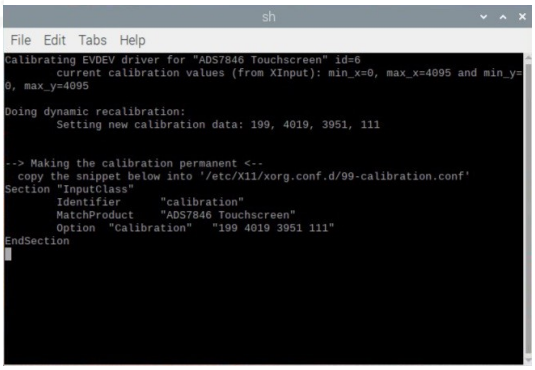


Figure 41 Touch Calibration

3. Resolution Change and Rotation

   I updated the display settings by editing the /boot/config.txt file. I set the resolution to match the display's native settings and adjusted the rotation to ensure the screen was oriented correctly.

```
# Edit the config.txt file
sudo nano /boot/config.txt

# Set HDMI resolution
hdmi_group=2
hdmi_mode=82

# Set screen rotation (e.g., 90 degrees)
display_rotate=1

# Save and exit (Ctrl+X, Y, Enter)
```

Figure 42 Resolution Change and Rotation

Audio Configuration

1. Output Audio

   I configured the audio output to use the 3.5mm jack for sound playback. I adjusted the settings using ALSA utilities to ensure that audio is routed through the 3.5mm jack.

```
# Set the default audio output to 3.5mm jack
amixer cset numid=3 1

# Test audio output by playing a sample sound
speaker-test -c2 -twav -l7
```

Figure 43 Output Audio

- PAM8403 3W Stereo Audio Amplifier Board Integration

I integrated the PAM8403 amplifier with the Raspberry Pi to drive audio output through the 3.5mm jack. I adjusted the audio settings using ALSA utilities to ensure the sound was properly routed and amplified by the PAM8403 board. This setup allows for clear and powerful audio playback from the Raspberry Pi.

2. Input Audio

   I set up the Raspberry Pi to use a USB microphone for audio input. I configured the microphone using ALSA utilities and tested it by recording a short audio clip.

```
# List available audio input devices
arecord -l

# Record a 5-second audio clip from the default microphone
arecord -D plughw:1,0 -d 5 test.wav

# Playback the recorded audio to check the input
aplay test.wav
```

Figure 44 Input audio

Camera Configuration

I configured the camera on the Raspberry Pi by first powering off the device and connecting the camera module to the CSI port, ensuring the ribbon cable was correctly inserted. After powering the Raspberry Pi back on, I opened a terminal and ran **sudo raspi-config**, navigated to **Interfacing Options**, selected **Camera**, and enabled it. After rebooting, I verified the setup by installing the picamera library with sudo apt install python3-picamera and running a simple Python script to capture an image. For further details, you can refer to the Raspberry Pi Camera Guide.

Power Supply System Setup

1. Car power supply

   To power the car, I used three 3.7V 3200mAh rechargeable batteries connected in series, which together provide a combined voltage of 11.1V. This is managed by a 12V Battery Management System (BMS) to ensure a stable and sufficient power supply for the car's operations. The BMS protects the batteries from overcharging and discharging, maintaining the stability and longevity of the power system.

2. Remote power supply

   For the remote control, I used a 3.7V 1800mAh rechargeable battery with a 5V Battery Management System (BMS). This configuration provides reliable power for the remote's functionality.

Soldering Assembling and Design Complex Reduction

I focused on minimizing the space used for assembly and reducing the complexity of wiring. This involved careful soldering and efficient layout design to streamline connections and ensure a clean, organized setup.

Rock-Paper-Scissors Game

I Set up a Rock-Paper-Scissors game using Python with **OpenCV** and **cvzone** for hand gesture recognition and **pygame** for sound effects. The game involves detecting hand gestures through a webcam to identify Rock, Paper, or Scissors.

Voice Assistant

I created "Hey Luigi" a voice assistant that uses Google's **Gemini API** for speech recognition and text generation. It checks for internet connectivity, processes user speech, and provides responses using Gemini's capabilities. The assistant also incorporates text-to-speech through **Google's TTS** services, avoiding **pyttsx3** and **gTTS**. Additionally, it opens a web view for enhanced user interaction.

Gamage G.G.P.T. 224056E

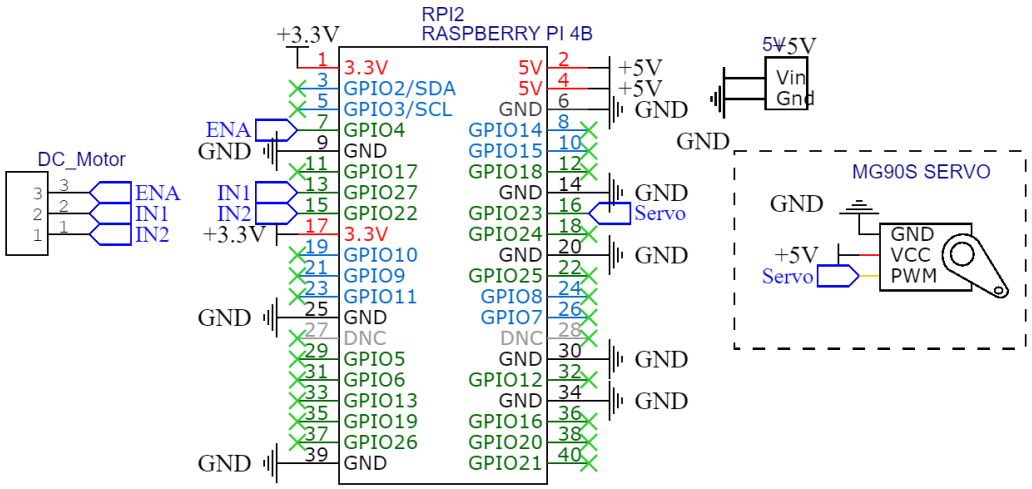


Figure 45 Gamage G.G.P.T. 224056E (Schematic Diagram)

My main responsibility in this project was to design and construct the control system for LUIGI's servo motor, DC motors (via a motor driver), LED headlights, and brake lights. Commands were sent through a joystick controller that was wirelessly connected to the Raspberry Pi via its built-in Bluetooth. Below is a detailed description of my responsibilities and the technical approach I used for the system's successful operation.

The system received input commands from the joystick controller in the form of a string of data, which was transmitted to the Raspberry Pi via Bluetooth.

The process began with splitting this command string. The string was divided into parts, each of which represented a distinct set of control instructions for a component of LUIGI. Each part was assigned to the dedicated variables, which were used in specific functions to control the related functionalities of the robot. This made sure that every movement was managed quickly and effectively, regardless of whether it was light activation or motor control.

Code for spitting the string

```python
# Function to handle Bluetooth data and control motor/servo/LEDs
def bluetooth_loop(sock):
    try:
        while True:
            data = sock.recv(1024)
            if not data:
                break
            data_str = data.decode().strip()
            parts = data_str.split(',')

            if len(parts) == 3:
                command = int(parts[0])
                horn = int(parts[1])
                headlight = int(parts[2])
                print(f"Command: {command}, Horn: {horn}, Headlight: {headlight}")

                set_servo_from_received_number(command)
                set_motor_from_received_number(command)
                led_combined(horn, headlight)
    except Exception as e:
        print(f"Error in Bluetooth loop: {e}")
    finally:
        sock.close()
```

Figure 46 Code for spitting the string

Controlling the servo motor

One of my key responsibilities was the control of LUIGI's servo motor, which was responsible for steering the front wheels. The servo motor was set to a neutral position at an angle of 90°, allowing the robot to move forward or backward. Depending on the commands received from the joystick, the servo motor adjusted its angle to either 45° for a left turn or 135° for a right turn. These precise angle adjustments enabled the rover to execute smooth, controlled turns, simulating a functional steering mechanism.

Code for controlling the servo angle.

```python
# Function to set the servo angle based on received number
def set_servo_from_received_number(number):
    global pwm_servo
    if number == 2 or number == 5 or number == 7:
        print("Turn left")
        set_servo_angle(45)

    elif number == 1 or number == 8 or number == 6:
        print("Turn right")
        set_servo_angle(135)

    elif number == 3 or number == 4:
        print("Move straight")
        set_servo_angle(90)


    else:
        GPIO.output(LED_PIN, GPIO.LOW)
        print("Invalid servo command received, no action taken.")

```

Figure 47 Code for controlling the servo angle.

Controlling the DC motors

I was also tasked with managing LUIGI's lighting system, which included two LED headlights and two LED brake lights. The headlights were operated using the joystick controller, while the brake lights were automatically triggered whenever LUIGI reversed.

Controlling of LED headlights and break lights

Headlights: Both LEDs were connected to a single GPIO pin, enabling them to be controlled together. The on and off state of the headlights was determined by commands from the remote controller, which adjusted the GPIO pin's state to HIGH or LOW, turning the lights on or off according to the user's preference.

Brake Lights: In a similar manner, the robot's movement controlled the activation of the brake lights, which were attached to a different GPIO pin. Whenever LUIGI reversed, the brake lights were automatically triggered by setting the corresponding GPIO pin to a HIGH state, offering an interactive mechanism to the user. This was accomplished by activating the brake lights to correspond with the direction order that was sent in the command thread.

Code for controlling the DC motors and LEDs

```
83  # Function to set the motor direction based on received number
84  def set_motor_from_received_number(number):
85      global pwm_motor, safety_active
86      if safety_active:
87          return
88      if number == 4 or number == 5 or number == 6:
89          print("Motor forward")
90          GPIO.output(in1, GPIO.HIGH)
91          GPIO.output(in2, GPIO.LOW)
92          GPIO.output(LED_PIN, GPIO.HIGH)
93      elif number == 3 or number == 7 or number == 8:
94          print("Motor backward")
95          GPIO.output(in1, GPIO.LOW)
96          GPIO.output(in2, GPIO.HIGH)
97          GPIO.output(LED_PIN, GPIO.LOW)
98      elif number == 1 or number == 2 or number == 0:
99          print("Motor not moved")
100         GPIO.output(in1, GPIO.LOW)
101         GPIO.output(in2, GPIO.LOW)
102         GPIO.output(LED_PIN, GPIO.LOW)
103
104     else:
105         GPIO.output(LED_PIN, GPIO.LOW)
106         print("Invalid motor command received, no action taken.")
```

Figure 48 Code for controlling the DC motors and LEDs

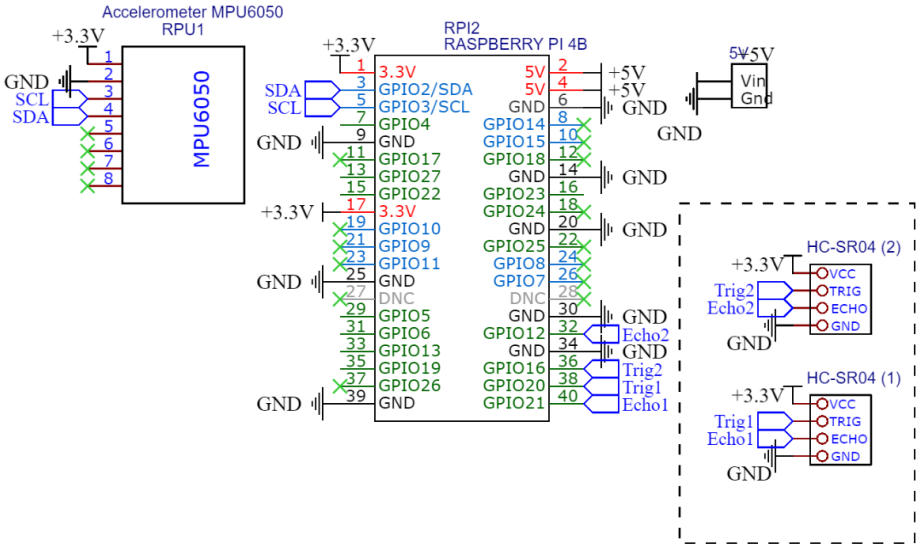


Figure 49 Kumari J.K.A.V. 224115K (Schematic diagram)

In the project of the Luigi pet robot, I am supposed to enhance user interaction and improve user experience by implementing features that avoid damage to the robot and protect Raspberry-pi from high voltage. I reached these goals with an MPU6050 sensor that detects motion, an Ultrasonic sensor for distance measurement, and by using a Buck converter. The exact descriptions of my contributions are described in detail in the following sections.

Shake Detection for Theme Change (Using MPU6050 Accelerometer features)

In this, the aim was to improve user interaction by allowing Luigi to respond to intentional moves.

Detection of the movement of Luigi could be done using an MPU6050 accelerometer. First, I found out that small movements caused noise in the sensor reading. So, I applied a threshold of 2000 on the x, y, and z-axis and filtered out minor, unintentional movements.

Instead of reacting to a single shake, I implemented a three-consecutive-shake detection feature to avoid accidentally triggering it. In that way, only deliberate shaking will be counted as a command.

Upon detecting three shakes, Luigi will change the color theme on its home screen; hence, users can customize the interface based on their preference. This adds to the user experience in terms of interactivity with feedback.

Special reason for choosing accelerometer feature: it is ideal for movement and orientation change detection, ideal to spot the shakes, later to be mapped for commands such as changing color themes.

Code for detecting the shake and changing the color theme

```
# Function to initialize MPU6050
def init_mpu6050():
    bus = smbus.SMBus(1)
    bus.write_byte_data(MPU6050_ADDR, PWR_MGMT_1, 0)

# Function to read a word from MPU6050
def read_word(bus, addr, reg):
    high = bus.read_byte_data(addr, reg)
    low = bus.read_byte_data(addr, reg + 1)
    val = (high << 8) + low
    if val >= 0x8000:
        return -((65535 - val) + 1)
    else:
        return val

# Function to detect shake
def detect_shake(bus):
    threshold = 20000
    accel_x = read_word(bus, MPU6050_ADDR, ACCEL_XOUT_H)
    accel_y = read_word(bus, MPU6050_ADDR, ACCEL_YOUT_H)
    accel_z = read_word(bus, MPU6050_ADDR, ACCEL_ZOUT_H)
    return abs(accel_x) > threshold or abs(accel_y) > threshold or abs(accel_z) > threshold
```

```
# Function to change button colors randomly
def change_button_colors():
    colors = ["darkred", "darkblue", "darkgreen", "darkorange", "purple", "teal", "navy", "maroon"]
    random_color = random.choice(colors)
    buttons = [left_button1, left_button2, left_button3, left_button4, right_button1, right_button2,
               right_button3, right_button4]
    for button in buttons:
        button.config(bg=random_color)
    canvas.itemconfig(circle, fill=random_color)

# Function to check for shakes and change button colors
def check_for_shakes():
    bus = smbus.SMBus(1)
    init_mpu6050()
    shake_count = 0
    while running:
        if detect_shake(bus):
            shake_count += 1
            if shake_count == 3:
                shake_count = 0
                change_button_colors()
                shake_sound.play()
        time.sleep(0.1)
```

Figure 50 Code for detecting the shake and changing the color theme

Wakeup Mode with MPU6050

Here, my goal is to return interactive feedback, which can be used when Luigi is put in power-saving sleep mode.

The shaking of the sleeping-mode Luigi will be detected by the MPU6050 sensor. Upon the detection of this shake, Luigi moves into a sleep mode, through an animation of opening his eyes with an accompanying audio effect, thereby emulating waking up for the robot.

This makes Luigi even more vivid, as it responds appropriately at once with the user's action and increases user experience. Shake detection is employed here to wake up Luigi, making the robot more interactive to react with physical interactions; it encourages users to interact with Luigi more during his idle moments.

Code for detecting the shake and wake up mode

```python
# Function to detect shake in a separate thread
def shake_detection_thread():
    shake_count = 0
    required_shakes = 2
    global sensor_enabled  # Add this line to use the global variable

    try:
        while True:
            if not sensor_enabled:  # Check if sensor readings are disabled
                print("Shake detection stopped.")
                break
            if detect_shake():
                shake_count += 1
                if shake_count >= required_shakes:
                    print("Shake detected! Playing video and audio for 11 seconds.")

                    # Start video playback in a thread
                    video_thread = threading.Thread(target=play_video_for_duration, args=("Video/Wake_Red.mp4", 11))
                    video_thread.start()
                    # Start audio playback in a thread
                    audio_thread = threading.Thread(target=play_audio_in_thread, args=("/home/pi/Desktop/LUIGI_Final/HTML/WAV/yawn.wav", 11))
                    audio_thread.start()
                    # Wait for video and audio threads to finish
                    video_thread.join()
                    audio_thread.join()
                    # Close the webview window
                    if window:
                        webview.windows[0].destroy()
                    # Run another Python script
                    subprocess.run(['python', '/home/pi/Desktop/LUIGI_Final/Home.py'])
                    os._exit(0)  # Terminate the script
                    shake_count = 0  # Reset shake count
            else:
                print("No shake detected.")
                shake_count = 0  # Reset shake count if no shake is detected
            time.sleep(0.1)
```

Figure 51 Code for detecting the shake and wake up mode

Fear Mode and Lift Detection (Using Ultrasonic Sensor)

Here, my goal was to provide interactive and emotional feedback in case Luigi is lifted.The ultrasonic sensor was used to identify when Luigi is lifted beyond 15 cm. In such a case, Luigi goes into fear mode-showing a frightened face and playing the appropriate sound.

Moreover, the front wheels move randomly, simulating such a reaction of fear with the purpose of making users more involved with Luigi's emotional behavior.

Ultrasonic sensors send sound waves through the air and bounce back on the object. What the sensor is measuring is the time for the echo to return. By using the speed of sound in the air ~ 343 m/s, from the following formula the distance to the object is calculated:

Distance= (Speed of Sound × Time Taken for Echo to Return)/2

This is because it travels to the object and back to the sensor, thus the distance is divided by 2. The sensor lifts over 15cm and when it does so, the robot acts accordingly - triggers the fear animation along with the sound.

Code for activate the fear reaction

```
while True:
    if sensor_enabled:
        dist = test_ultrasonic_sensor()

        if dist > 15:
            print(f"Distance: {dist} cm - Moving servo")

            set_servo_angle(38)
            time.sleep(0.03)
            set_servo_angle(90)
            time.sleep(0.03)
            set_servo_angle(142)
            time.sleep(0.03)
            set_servo_angle(90)
            time.sleep(0.03)

            # Start the sound thread if not already running and sound is not already playing
            global sound_thread, sound_playing
            if sound_thread is None or not sound_thread.is_alive():
                if not sound_playing:
                    sound_thread = threading.Thread(target=play_sound)
                    sound_thread.start()

            # Change the video in the webview for ultrasonic condition
            if current_video_playing != ultrasonic_video_path:
                webview.windows[0].evaluate_js(f"setVideoSource('{ultrasonic_video_path}')")
                current_video_playing = ultrasonic_video_path
        else:
            print(f"Distance: {dist} cm - Stopping servo")
            pwm.ChangeDutyCycle(0)
```

Figure 52 Code for activate the fear reaction

Obstacle Detection and Safety Mechanism (Using Ultrasonic Sensor)

Aims at preventing collision detection of objects around it and gives it a stop on the movement of it. I then used an ultrasonic sensor that detects objects in a 10cm radius. Once Luigi detects obstacles, it will perform a reverse for 0.5 seconds to avoid collision.

After reversing, Luigi again checks in 0.5 seconds if the obstacle is still there and, if so, it reverses once more for another 0.5 seconds, keeping in this cycle until the path is clear. Then, I also added a flag system to stop user commands from navigating when it detects an obstacle for the sake of safety. If the obstacle has been cleared, then Luigi resumes accepting commands again.

Code for activate the safety feature

```
187  # Function to monitor ultrasonic sensor and move motor backward if obstacle detected
188  def ultrasonic_monitor():
189      global safety_active
190      while True:
191          distance = measure_distance()
192          print(f"Distance: {distance} cm")
193
194          if distance < 10 and not safety_active:
195              print("Obstacle detected! Moving motor backward.")
196              safety_active = True
197              GPIO.output(in1, GPIO.HIGH)
198              GPIO.output(in2, GPIO.LOW)
199              time.sleep(0.5)
200              GPIO.output(in1, GPIO.LOW)
201              GPIO.output(in2, GPIO.LOW)
202              time.sleep(0.5)
203              safety_active = False
204
205          time.sleep(0.5)  # Add a small delay to avoid excessive CPU usage
206
```

Figure 53 Code for activate the safety feature

Buck Converter for Power Regulation

The goal is to feed the correct voltage into the Raspberry Pi from a 12V battery.

Luigi's battery supplies 12V, whereas the operating voltage of the Raspberry Pi is 5V. A buck converter was implemented to efficiently handle this voltage step-down to 5V from 12V.

A buck converter is a type of DC-to-DC converter that converts higher input voltage to lower, stable output by using a switch-usually a transistor-and an inductor. In such a way, this guarantees that the Raspberry Pi will get the right power without damage; it efficiently uses energy and does not harm the components.

I worked on enhancing Luigi in terms of interaction and safety by using an MPU6050 sensor for motion detection, an ultrasonic sensor for distance measurement, and a buck converter for power regulation. In this way, the user can experience engagement and safety, and Luigi therefore becomes more responsive and life-like.

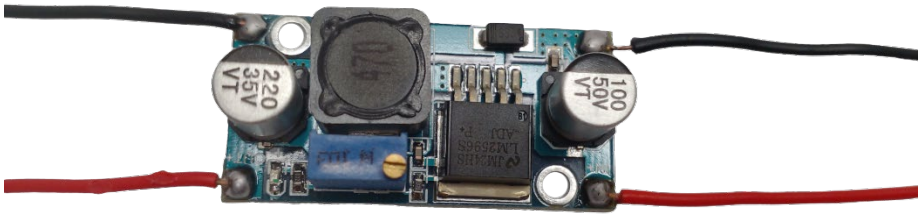


Figure 54 Buck Converter

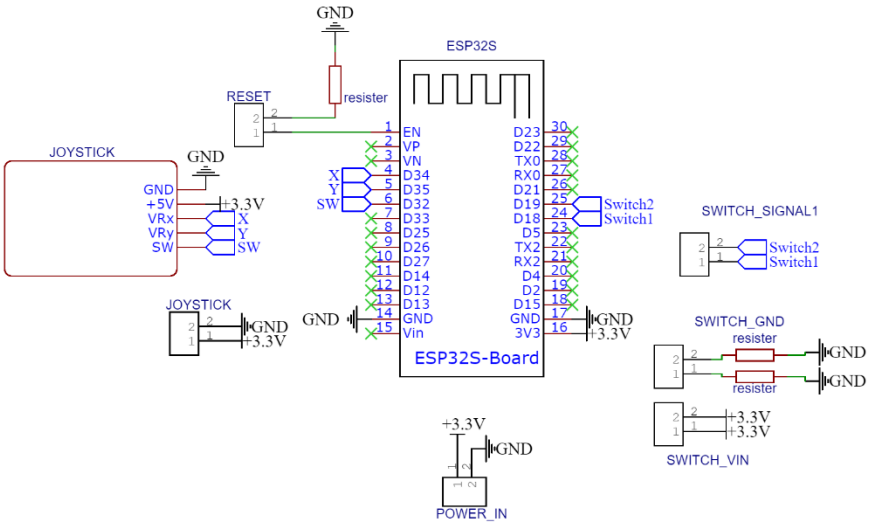


Figure 55 Kahanda M.C. 224098H (Schematic Diagram)

Through the development of this project, I have played a pretty central role in a number of key areas that include navigation system development, 3D designs, PCB design, and final code assembly. Now, in more detail, here is what my tasks exactly were:

Programming and Building the Navigation System:

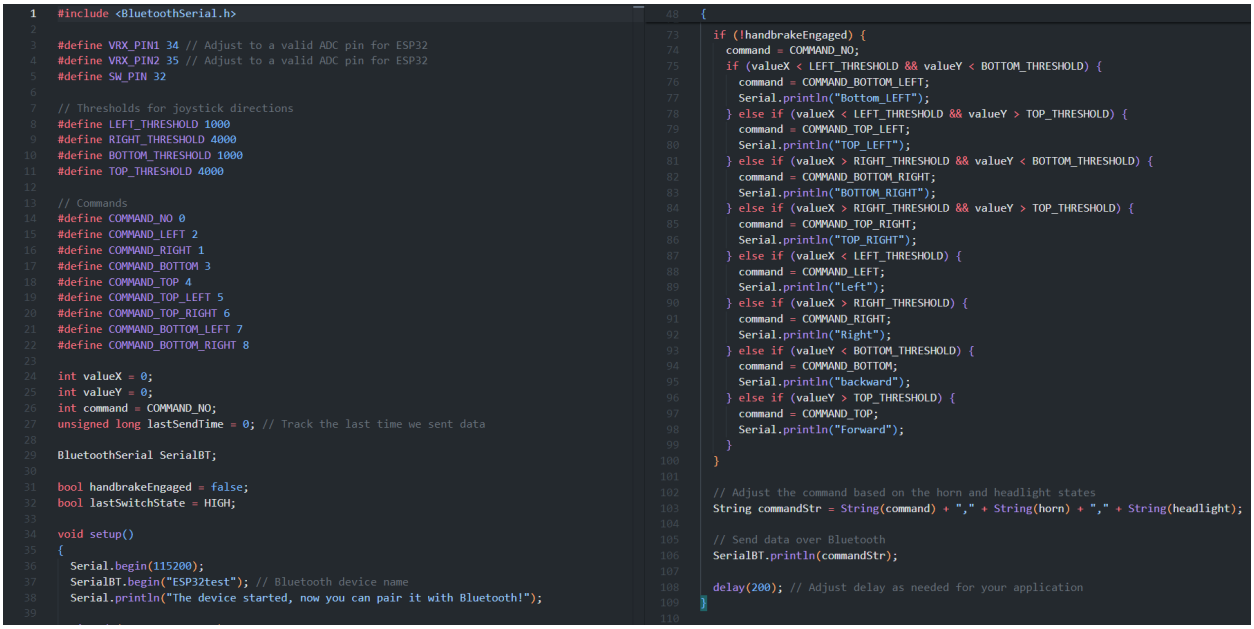
I design the navigation for Luigi, a pet robot, which would be controlled remotely using ESP32. Implementation of joystick-based control capture of X and Y coordinate values to determine the movement directions was performed. Hence, it became possible, through analyzing those values, to set thresholds that would turn continuous input into discrete directional commands.

I created four thresholds to define the directions of movement: front, back, left, and right. These I combined further to give eight directions, including reverse-left, reverse-right, front-left and front-right. In the neutral position of the joystick, it outputs the coordinate (16000, 1700). A value of the Y-coordinate greater than 4000 ensures movement in the forward direction; and less than 1000 it is backward. Similarly, X-coordinate value greater than 4000 indicates right and less than 1000 indicates left. To make it more functional, I added a 3-way switch for the headlights and a push button for the horn. I created an all-encompassing UI with respect to moving and accessory control. Communication between ESP32 and Raspberry Pi was another important part of the system.

After trying several protocols for communications like HTTP and UDP, I have chosen Bluetooth because of the low latency, greater range, and stability in its connections. I utilized the library to

facilitate communication and optimize performance. Besides, to minimize the delay and make the system seamless, I structured the system in a way that all commands, such as directional, horn, and headlight, were sent in one string format. This saved time during transmission and thus ensured speedy execution. Moreover, a safety measure was incorporated using the switch within the joystick as a handbrake. The safety mechanism was necessary due to the fact that Luigi is a desktop pet and it should stay in one place when required.

Code for identify user commands and send them to a Raspberry Pi via Bluetooth



```
#include <BluetoothSerial.h>

#define VRX_PIN1 34 // Adjust to a valid ADC pin for ESP32
#define VRX_PIN2 35 // Adjust to a valid ADC pin for ESP32
#define SW_PIN 32

// Thresholds for joystick directions
#define LEFT_THRESHOLD 1000
#define RIGHT_THRESHOLD 4000
#define BOTTOM_THRESHOLD 1000
#define TOP_THRESHOLD 4000

// Commands
#define COMMAND_NO 0
#define COMMAND_LEFT 2
#define COMMAND_RIGHT 1
#define COMMAND_BOTTOM 3
#define COMMAND_TOP 4
#define COMMAND_TOP_LEFT 5
#define COMMAND_TOP_RIGHT 6
#define COMMAND_BOTTOM_LEFT 7
#define COMMAND_BOTTOM_RIGHT 8

int valueX = 0;
int valueY = 0;
int command = COMMAND_NO;
unsigned long lastSendTime = 0; // Track the last time we sent data

BluetoothSerial SerialBT;

bool handbrakeEngaged = false;
bool lastSwitchState = HIGH;

void setup()
{
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); // Bluetooth device name
  Serial.println("The device started, now you can pair it with Bluetooth!");
```

```
{
  if (!handbrakeEngaged) {
    command = COMMAND_NO;
    if (valueX < LEFT_THRESHOLD && valueY < BOTTOM_THRESHOLD) {
      command = COMMAND_BOTTOM_LEFT;
      Serial.println("Bottom_LEFT");
    } else if (valueX < LEFT_THRESHOLD && valueY > TOP_THRESHOLD) {
      command = COMMAND_TOP_LEFT;
      Serial.println("TOP_LEFT");
    } else if (valueX > RIGHT_THRESHOLD && valueY < BOTTOM_THRESHOLD) {
      command = COMMAND_BOTTOM_RIGHT;
      Serial.println("BOTTOM_RIGHT");
    } else if (valueX > RIGHT_THRESHOLD && valueY > TOP_THRESHOLD) {
      command = COMMAND_TOP_RIGHT;
      Serial.println("TOP_RIGHT");
    } else if (valueX < LEFT_THRESHOLD) {
      command = COMMAND_LEFT;
      Serial.println("Left");
    } else if (valueX > RIGHT_THRESHOLD) {
      command = COMMAND_RIGHT;
      Serial.println("Right");
    } else if (valueY < BOTTOM_THRESHOLD) {
      command = COMMAND_BOTTOM;
      Serial.println("backward");
    } else if (valueY > TOP_THRESHOLD) {
      command = COMMAND_TOP;
      Serial.println("Forward");
    }
  }

  // Adjust the command based on the horn and headlight states
  String commandStr = String(command) + "," + String(horn) + "," + String(headlight);

  // Send data over Bluetooth
  SerialBT.println(commandStr);

  delay(200); // Adjust delay as needed for your application
}
```

Figure 56 Code for identify user commands and send them to a Raspberry Pi via Bluetooth

Designing the PCB:

I was responsible for the PCB design for Luigi and the controller. I used the EasyEDA online platform to create schematic diagrams in the connection of components. Following that, with the finalization of the schematics, a single-layer PCB design was prepared with the main consideration of compactness and functionality. The design was done considering component positioning for the flow of signals with minimal interference.

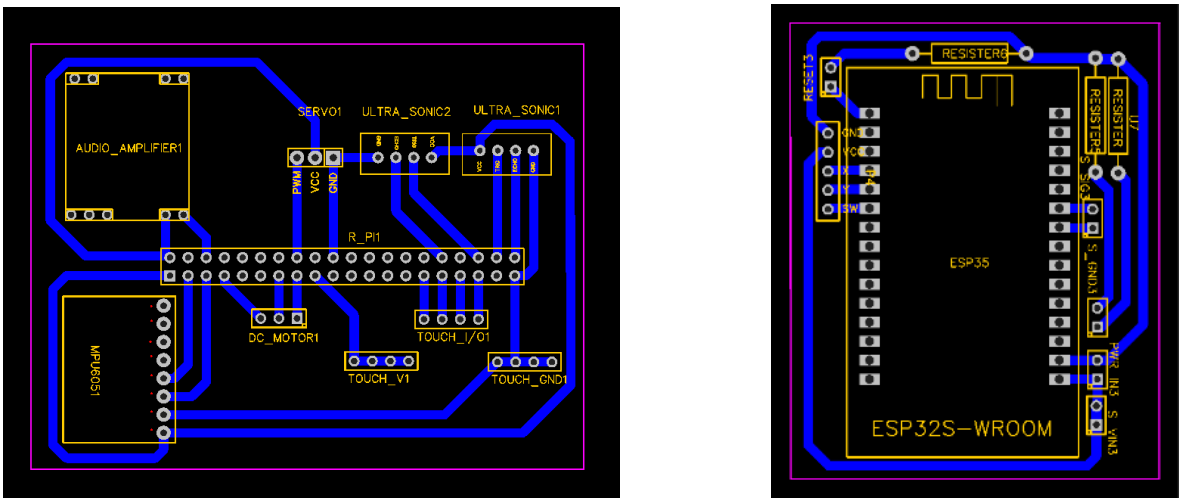


Figure 57 PCB designs

3D Design of the Joystick:

I used Fusion365 for the creation of the 3D model of the joystick to be used for the control of Luigi. This involved the development of a functional and ergonomic design that would be comfortable for the user and good enough to affect control on the robot. The design of the joystick further contained all the buttons and switches required to control additional features such as the headlamps, reset, power button, and horn, ensuring it would be user-friendly.

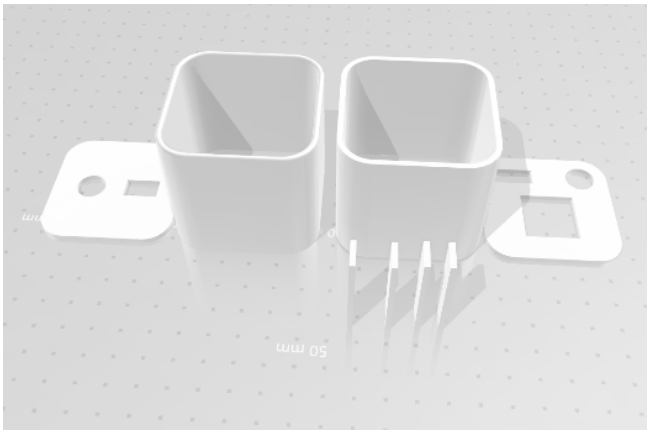


Figure 58 3D Designs

Interactive Games for Enhancing User Engagement

To encourage users to interact more with the pet robot, I developed two interactive games using Tkinter GUI in Python: Mind Puzzle game and a Pop the Bubble game. Tkinter is quite a flexible

library; it allows for the shaping, coloring, and laying out of the shapes according to one's wish. This was quite helpful in developing visually appealing and functional interfaces for both games.

The aim of the mind puzzle game was to improve the user's memory and cognitive ability. I designed a grid of squares, each hiding a color; users were supposed to pick out the pairs of squares that contained the same color. Every time two squares of matching color are picked, they flip face up, and the user can continue playing. This challenge based on memory asks the user to remember positions; hence, it improves cognitive skills through visual recognition and pattern matching.

The second game, "Pop the Bubble," was designed to enhance the user's speed and agility. In this game, there is a bubble circular shape that appears anywhere on the screen. The user should click on that bubble before the timer expires or the bubble disappears to change its position. As time goes by, the seconds allotted to pop the bubble decrease and add more difficulty to the task, thus requiring quicker reflexes.

Both games were designed to be fun for him while developing some of his key cognitive and motor skills in the process. The great flexibility of the Tkinter library had allowed me to build dynamic, colorful interfaces regarding user experience for the robot.

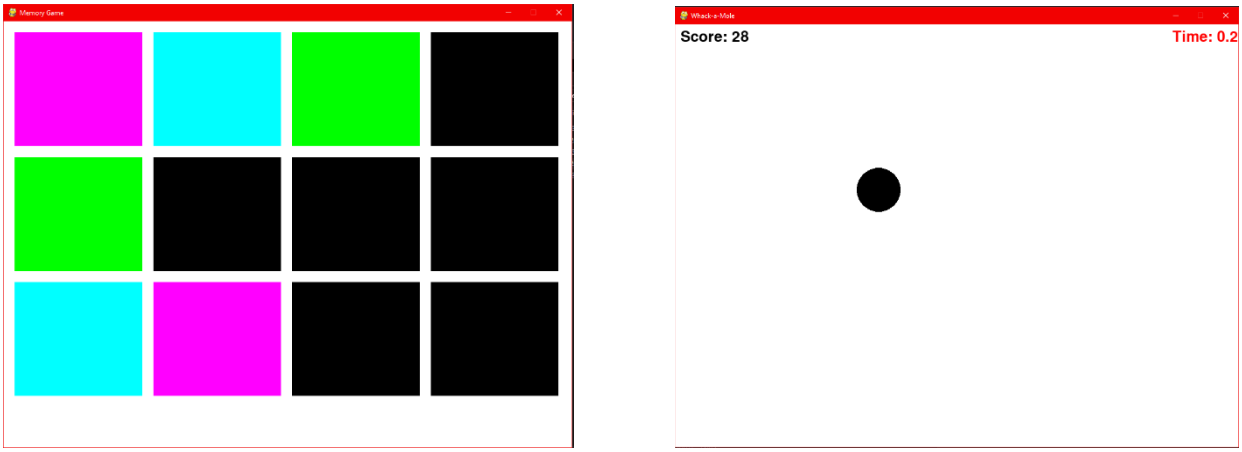


Figure 59 Games made by Tkinter

Final Code Assembly and Optimization:

I took full responsibility for assembling and finalizing the code by integrating the functions developed by my team members into a cohesive system. Since 'Luigi', the pet robot, had many parts, I decided to use threading because several functions, such as sensor readings, motor control, and user inputs, required simultaneous execution in real-time. This was particularly crucial for

tasks that required continuous sensor data processing and motor control. The slightest amount of delay would dampen not just the performance of the robot but, more importantly, the user experience.

To optimize system efficiency, I utilized flags to enable or disable sensor functionality based on their actual usage or otherwise. This allowed sensors, when not in operation, to stay turned off and relieved the system from processing overhead. Different test scenarios were performed, which ranged from realistic scenarios: keeping in control of the movement by the sensor inputs, mode switches, and responding to user commands in real time. These tests helped in the building process of the robot to weed out flaws and fix them to ensure that everything flowed well and that the immediate responses of the robot came about.

Code for creating the threads

```
if __name__ == "__main__":
    try:
        # Start threads for double tap detection, servo control, and additional touch sensors
        detection_thread = threading.Thread(target=detect_double_tap)
        detection_thread.start()

        servo_thread = threading.Thread(target=move_servo)
        servo_thread.start()

        additional_touch_thread_6 = threading.Thread(target=handle_additional_touch, args=(TOUCH_PIN_6,))
        additional_touch_thread_6.start()

        additional_touch_thread_13 = threading.Thread(target=handle_additional_touch, args=(TOUCH_PIN_13,))
        additional_touch_thread_13.start()

        additional_touch_thread_19 = threading.Thread(target=handle_additional_touch, args=(TOUCH_PIN_19,))
        additional_touch_thread_19.start()

        # Start the webview
        start_webview()

        # Wait for the threads to finish (although they will run indefinitely until Ctrl+C is pressed)
        detection_thread.join()
        servo_thread.join()
        additional_touch_thread_6.join()
        additional_touch_thread_13.join()
        additional_touch_thread_19.join()

```

Figure 60 threading

Reference

Breazeal, C. (2003). *Emotion and sociable humanoid robots*. International Journal of Human-Computer Studies, 59(1-2), 119-155. https://doi.org/10.1016/S1071-5819(03)00018-1

Fox, D., Burgard, W., & Thrun, S. (2005). The role of sensory systems in dynamic robot behavior. In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 374-379). https://doi.org/10.1109/ROBOT.2005.1570175

Park, S., Kwon, H., & Han, Y. (2011). Touch screen interfaces in interactive robots: Enhancing human-robot interaction. *Journal of Robotics and Autonomous Systems*, 59(9-10), 724-734. https://doi.org/10.1016/j.robot.2011.05.005

Shibata, T., Wada, K., & Tanie, K. (2004). *Effects of robot-assisted activity for elderly people and nurses at a day service center*. Proceedings of the IEEE, 92(11), 1780-1788. https://doi.org/10.1109/JPROC.2004.835378

Siegwart, R., & Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. MIT Press.

YT. (n.d.). *Raspberry Pi power supply setup*. YouTube. https://www.youtube.com

Alldatasheet. (n.d.). *MPU6050 datasheet*. Alldatasheet. https://www.alldatasheet.com

Alldatasheet. (n.d.). *HC-SR04 ultrasonic sensor datasheet*. Alldatasheet. https://www.alldatasheet.com

Alldatasheet. (n.d.). *ESP32 datasheet*. Alldatasheet. https://www.alldatasheet.com

Alldatasheet. (n.d.). *Touch sensor datasheet*. Alldatasheet. https://www.alldatasheet.com