Raspberry Pi and Arduino – Comparison and Applications (Robot Car Project: Face Identification)

Jazzal Kandiel, Liza Yousef, Mawa Hijji, Maha alblawi
S21107396, S23108546, S22107735, S22107905
Effat University
ECE 342L: Analog Control Systems
Dr. Nema Salem
April 22 2025

Table of contents

Abstract

This project focuses on building a smart robot car capable of identifying faces using a 5 Megapixel camera module and a Raspberry Pi 4B. The robot uses a motor driver board controlled directly by the Raspberry Pi’s GPIO pins and processes video streams using OpenCV and Python to detect human faces in real time. When a face is detected, the robot moves forward; otherwise, it stops. This system demonstrates the computational power of the Raspberry Pi in handling both hardware control and complex image processing tasks simultaneously.

Introduction

The Raspberry Pi, a powerful single-board computer, allows for advanced robotics projects that require both hardware interfacing and intensive computation like image processing. Unlike microcontrollers, which are limited in processing capabilities, Raspberry Pi can perform real-time video analysis while controlling motors and sensors. This project leverages the Raspberry Pi’s capabilities to create a 4WD robot car that moves intelligently based on face detection using Python and OpenCV libraries. This application highlights the Raspberry Pi’s suitability for embedded AI and robotics tasks without needing a secondary microcontroller like Arduino.

Objective

The objectives of this project are as follows:

- To design and build a 4WD robot car using only Raspberry Pi.

- To implement real-time face identification using a camera module and OpenCV.

- To directly control DC motors through a motor driver board using Raspberry Pi GPIO pins.

- To demonstrate the Raspberry Pi’s ability to handle both hardware control and image processing.

Components

Each component used in the project and its role:

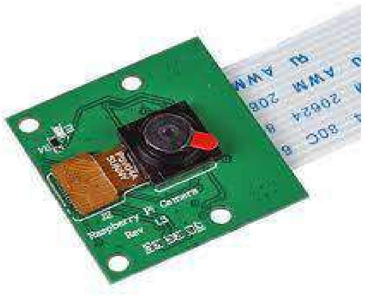
1. Raspberry Pi 4B

- Function: Central processing unit of the robot.
- Role: Runs Python code to detect faces, and controls motor movement based on detection results.
- Why: Combines hardware control with real-time video processing.

2. 5MP Camera Module

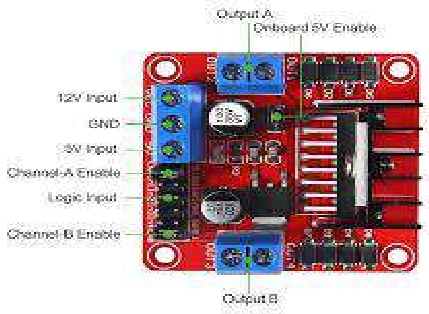
- Function: Captures live video feed.
- Role: Provides input images for face detection algorithms.

- Why: Essential for implementing computer vision.



3. Motor Driver Board (L298N or Motor HAT)

- Function: Interfaces between Raspberry Pi and DC motors.
- Role: Amplifies control signals from Raspberry Pi GPIOs to drive four DC motors.
- Why: Raspberry Pi GPIO pins alone cannot supply the necessary current for motors.



4. 4 DC Motors and 4 Wheels

- Function: Provides mobility to the robot.
- Role: Driven by the motor driver board to allow forward motion when a face is detected.

- Why: Necessary for movement based on detected faces.

5. Chassis

- Function: Structural platform.
- Role: Holds Raspberry Pi, motor driver board, motors, camera, and batteries.
- Why: Provides support and alignment for components.

6. Battery Pack

- Function: Power supply for motors and Raspberry Pi.
- Role: Delivers stable voltage and current to ensure uninterrupted operation.

- Why: Required to make the robot mobile and untethered.

7. Jumper Wires and Connectors

- Function: Electrical connectivity.
- Role: Connect Raspberry Pi GPIO pins to motor driver inputs and power lines.
- Why: Enables communication and control between Raspberry Pi and hardware components.

Connections

The system uses a direct connection between the Raspberry Pi, the motor driver board, the camera module, the DC motors, and the power supply.

The connections are explained below :

1. Camera Module to Raspberry Pi

The 5MP camera module is connected to the Raspberry Pi through the CSI (Camera Serial Interface) port.

To connect it:

- Gently pull up the plastic clip on the CSI port of the Raspberry Pi.
- Insert the camera’s ribbon cable with the shiny metallic side facing toward the HDMI ports.
- Push the clip back down to lock the cable in place.

  No GPIO pins are used for the camera, and it communicates with the Raspberry Pi through a dedicated high-speed interface.

2. Motor Driver Board to Raspberry Pi GPIO Pins

The motor driver board (such as L298N or a Motor HAT) connects to the Raspberry Pi’s GPIO pins to control the movement of the motors.

The control pins of the motor driver are wired to the Raspberry Pi GPIO pins as follows:

- The IN1 pin of the motor driver is connected to GPIO17 on the Raspberry Pi (physical pin 11).
- The IN2 pin is connected to GPIO18 (physical pin 12).
- The IN3 pin is connected to GPIO22 (physical pin 15).
- The IN4 pin is connected to GPIO23 (physical pin 16).

These connections allow the Raspberry Pi to control the rotation direction of the motors by setting the GPIO pins HIGH or LOW through Python programming.

The GND (Ground) pin of the motor driver board is connected to one of the Raspberry Pi’s GND pins (for example, pin 6) to ensure a common ground between the two devices.

This common ground is essential for the motor control signals to be properly recognized.

3. Motors to Motor Driver Board

Each of the four DC motors is connected directly to the output terminals of the motor driver board:

- The left-side motors are connected to the first motor output channels (OUT1 and OUT2).

- The right-side motors are connected to the second motor output channels (OUT3 and OUT4).

If the motors run in the wrong direction (for example, backward when they should move forward), the two motor wires on that side can simply be swapped to correct the direction.

4. Power Connections

The motors require higher current than the Raspberry Pi's GPIO pins can provide.

Therefore, a separate battery pack (such as 7V to 12V) is used to supply power directly to the motor driver board.

The positive terminal of the battery pack is connected to the VCC (or +12V) input on the motor driver board.

The negative terminal of the battery pack is connected to the GND on the motor driver board, and this ground is also shared with the Raspberry Pi to maintain a common ground reference.

The Raspberry Pi itself is powered separately using a 5V portable power bank connected to its USB-C power input port

Part 1 Assembly:

Step 1: Prepare Components and Initial Assembly

Objective: Begin assembling the base structure of the Raspberry Pi Smart Car by preparing components and following the silk print on the PCB.
Components:

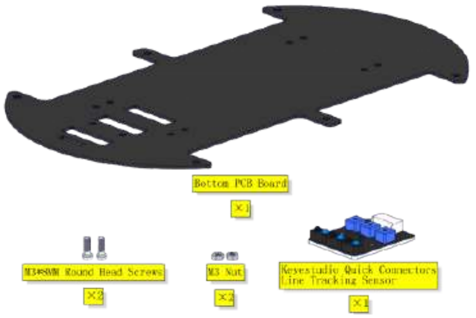
- Main PCB board
- Basic mounting hardware (screws, nuts, standoffs, etc.).
- Other foundational parts as indicated by the silk print.

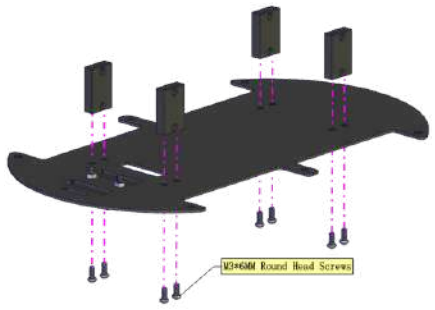
Assembly Process:

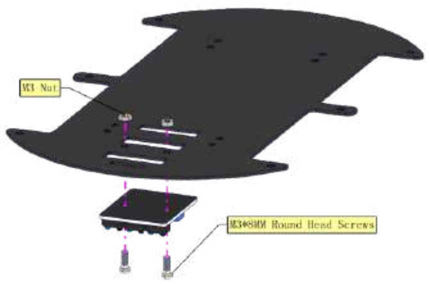
- Gather all components listed in the kit for Step 1. These are typically the main chassis and initial mounting hardware.
- Identify the silk print markings on the PCB. Silk prints are printed guides on the board that indicate where specific components should be placed.
- Align the components with the corresponding silk print markings and secure them using the provided screws or standoffs.
- Ensure all parts are firmly attached without over-tightening, which could damage the PCB or components.

Notes:
- The silk print is critical for correct placement, so double-check alignments before securing.
- This step likely involves setting up the base structure, such as the bottom plate of the chassis, which will support subsequent components like motors and wheels.

Completion:
- The base structure is assembled, with components securely fastened according to the silk print.







Step 2: Assemble Fixed Parts

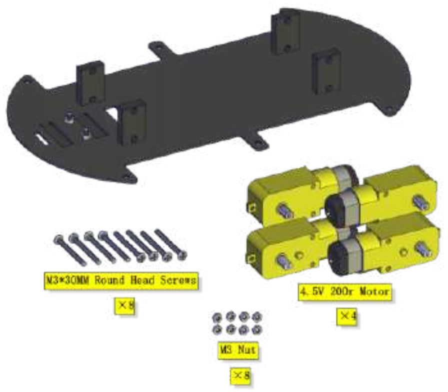
Objective: Attach fixed structural components to the chassis to prepare for motor and wheel installation.
Components:
- Additional structural parts (e.g., brackets, supports, or secondary plates).
- Fasteners (screws, nuts, or spacers).

Assembly Process:

- Collect the fixed parts specified for this step, which may include brackets or supports that reinforce the chassis.
- Position these parts on the chassis, aligning them with pre-drilled holes or silk print guides.
- Secure the parts using the provided fasteners, ensuring stability and proper alignment.
- Check that the fixed parts do not obstruct areas where motors, wheels, or other components will be installed later.

Notes:
- These fixed parts are likely structural reinforcements or mounts for other components, such as motor brackets or upper chassis plates.
- Ensure all connections are tight to prevent wobbling during operation.

Completion:
- The chassis is reinforced with fixed parts, ready for motor installation.



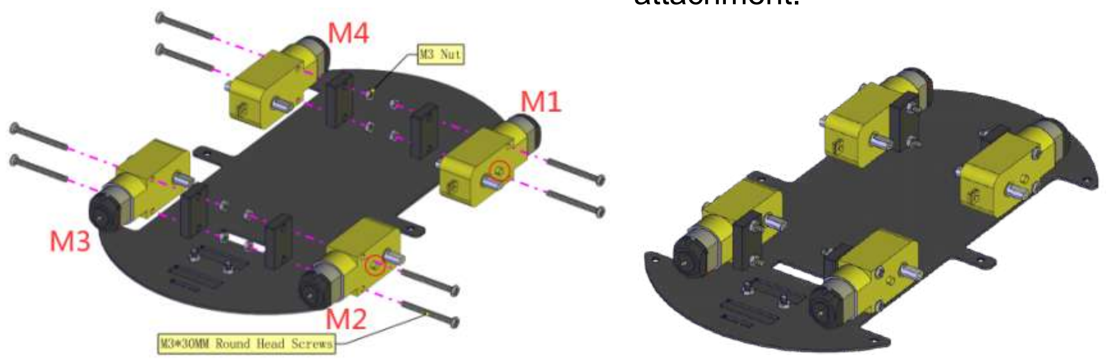
Step 3: Mount Four Motors

Objective: Install the four DC motors onto the chassis, ensuring correct orientation.
Components:
- Four DC motors with attached wires.
- Motor mounting brackets (if not already part of the chassis).
- Screws or fasteners for motor attachment.

Assembly Process:
- Gather the four DC motors and inspect them for consistency (e.g., wire lengths and mounting holes).
- Note the orientation requirement: the wires on all four motors must face inward (toward the center of the chassis).
- Attach each motor to its designated position on the chassis, typically at the four corners of the base plate.
- Secure the motors using screws or brackets, ensuring they are firmly in place and aligned parallel to the chassis edges.
- Verify that the motor wires are oriented inward as specified to ensure proper wiring and clearance later.

Notes:
- Incorrect motor orientation (wires facing outward) could interfere with wiring or wheel movement, so double-check before securing.
- The motors are likely geared DC motors, commonly used in smart car kits for driving the wheels.

Completion:
- All four motors are mounted with wires facing inward, ready for wheel attachment.



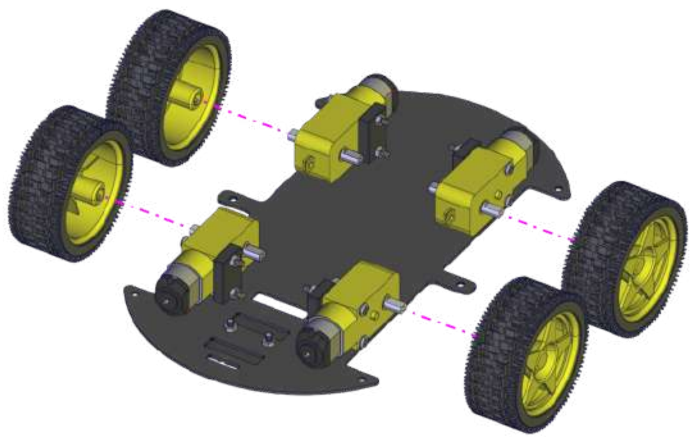
Step 4: Install Wheels

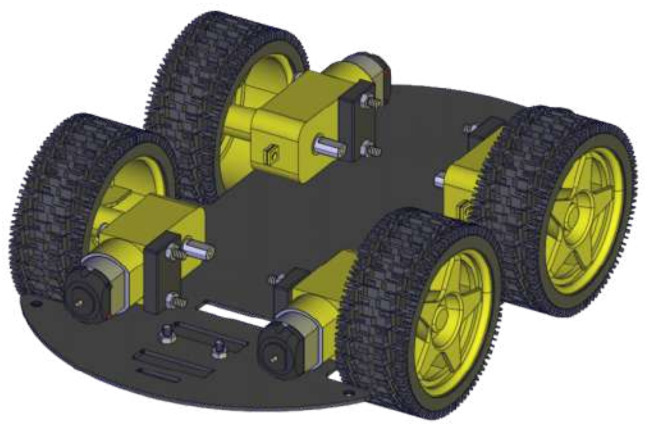
Objective: Attach wheels to the motors to enable movement.
Components:
- Four wheels (typically plastic with rubber tires).
- Couplers or adapters to connect wheels to motor shafts.
- Fasteners (if required for securing wheels).

Assembly Process:
- Collect the wheels and any couplers or adapters provided in the kit.
- Attach each wheel to the shaft of a motor. This may involve:
  - Sliding the wheel onto the motor shaft and securing it with a screw or clip.
  - Using a coupler to connect the wheel to the motor shaft if the wheel and shaft have different shapes (e.g., D-shaped shaft).
- Ensure each wheel is securely attached and rotates freely without wobbling.
- Check that the wheels are aligned perpendicular to the chassis for smooth movement.

Notes:
- If the kit includes a specific tool (e.g., a small wrench) for securing wheels, use it to avoid damaging the components.
- Verify that the wheels are compatible with the motor shafts and that no parts are loose.

Completion:
- The car has four wheels installed, enabling mobility once powered.





Step 5: Assemble Copper Pillars

Objective: Install copper pillars to create mounting points for additional components or boards.

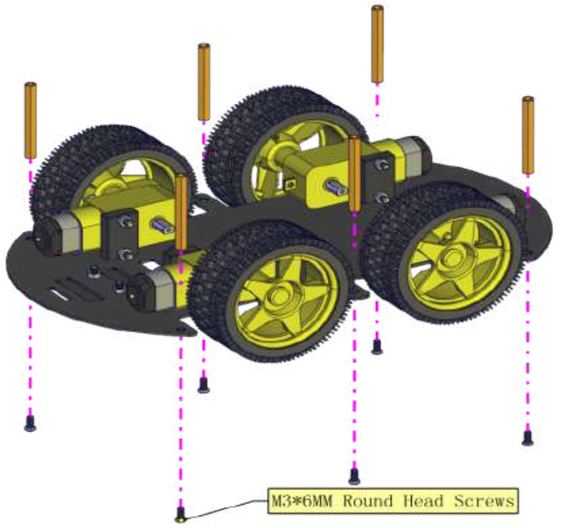
Components:
- Copper pillars (standoffs or spacers).

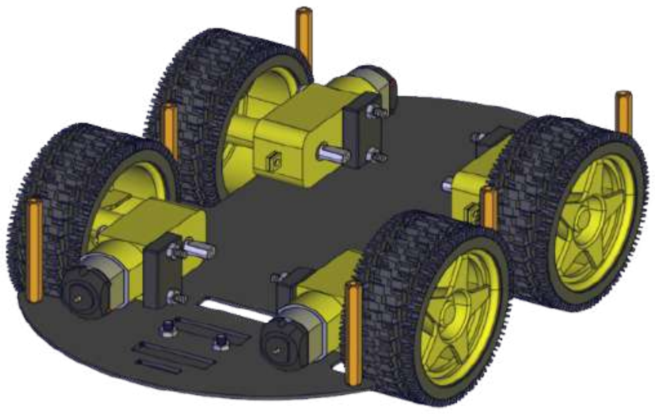
- Screws or nuts for securing the pillars.

Assembly Process:
- Gather the copper pillars and corresponding fasteners.
- Identify the mounting points on the chassis where the pillars will be installed (likely marked by holes or silk print).
- Screw the copper pillars into the designated holes, ensuring they are vertical and securely fastened.
- Check that the pillars are evenly spaced and at the correct height to support the next layer of components (e.g., Raspberry Pi board or sensor mounts).

Notes:
- Copper pillars act as spacers to elevate components, preventing short circuits and providing clearance for wiring.
- Ensure the pillars are not over-tightened, as this could strip the threads or damage the chassis.

Completion:
- Copper pillars are installed, providing mounting points for subsequent components.





Step 6: Assemble the Servo

Objective: Install a servo motor for controlling a component, such as a sensor or camera mount.
Components:

- Servo motor (likely a standard hobby servo, e.g., SG90 or MG90S).
- Servo mounting bracket or holder.
- Screws for securing the servo.

Assembly Process:
- Collect the servo motor and its mounting hardware.
- Attach the servo to its designated position on the chassis, typically using a bracket or direct mounting holes.
- Secure the servo with screws, ensuring it is firmly in place and aligned correctly.
- Do not attach the servo horn (arm) yet, as it may be installed in a later step with specific alignment instructions.

Notes:
- The servo is likely used for precise control, such as rotating a sensor or camera. Its placement should allow for free rotation without obstruction.
- Handle the servo carefully to avoid damaging its gears or wiring.

Completion:
- The servo is mounted on the chassis, ready for further configuration.

Step 7: Assemble the Ultrasonic Sensor

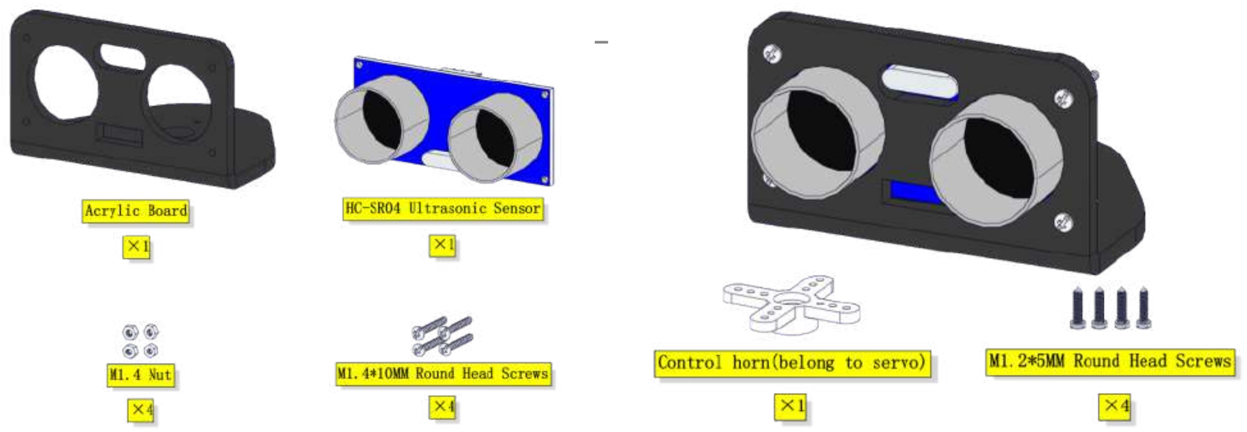
Objective: Install the ultrasonic sensor for distance measurement or obstacle detection.
Components:
- Ultrasonic sensor module (e.g., HC-SR04).
- Mounting bracket or holder for the sensor.
- Fasteners for securing the sensor.

Assembly Process:
- Gather the ultrasonic sensor and its mounting hardware.
- Attach the sensor to its designated mount, which may be attached to the servo from Step 6 to allow rotation.
- Secure the sensor using screws or clips, ensuring its transducers (transmitter and receiver) face outward and are unobstructed.
- If the sensor is mounted on the servo, ensure the servo can rotate the sensor without hitting other components.

Notes:

- The ultrasonic sensor is used for detecting obstacles by measuring distance, so its placement should provide a clear line of sight.
- Check that the sensor's pins (VCC, GND, TRIG, ECHO) are accessible for wiring in later steps.

Completion:
- The ultrasonic sensor is installed, ready for wiring and testing.


Acrylic Board
×1
HC-SR04 Ultrasonic Sensor
×1
M1.4 Nut
×4
M1.4*10MM Round Head Screws
×4
Control horn(belong to servo)
×1
M1.2*5MM Round Head Screws
×4

Step 8: Mount the Cross Servo Horn

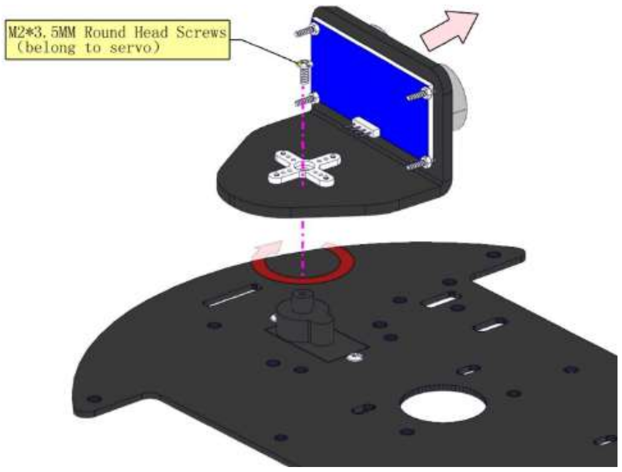
Objective: Attach the cross servo horn to the servo for precise control of a rotating component.
Components:
- Cross servo horn (a cross-shaped arm for the servo).
- Screw for securing the horn to the servo.

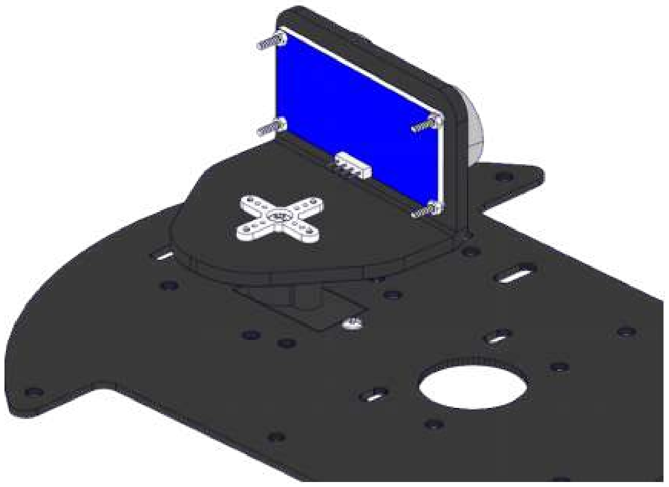
Assembly Process:
- Collect the cross servo horn and the servo it will be attached to (likely the one from Step 6).
- Temporarily attach the horn to the servo shaft without securing it.
- Rotate the horn clockwise until it reaches a "card position" (a point where it stops due to mechanical limits). Do not force it beyond this point to avoid damaging the servo.
- Remove the horn, align the servo shaft to a 90° position (as indicated in the instructions), and reattach the horn.
- Secure the horn with the provided screw, ensuring it is firmly attached.

Notes:

- The “card position” refers to the servo’s mechanical stop, which helps calibrate its rotation range.
- The 90° alignment ensures the servo’s neutral position is correctly set for controlling the attached component (e.g., ultrasonic sensor).

Completion:
- The cross servo horn is attached and calibrated, ready to control the attached component.





Step 9: Assemble on the Board

Objective: Finalize the servo and horn assembly on the chassis.
Components:
- Servo with cross horn (from Step 8).
- Additional mounting hardware (if required).

Assembly Process:
- Ensure the servo from Step 8 is correctly positioned with the cross horn at 90°.
- Mount the servo assembly onto the chassis if it was not already secured in Step 6.
- Secure the servo using screws, ensuring it is stable and the horn can rotate freely.
- Verify that the cross horn’s movement is unobstructed and correctly aligned with the component it controls (e.g., ultrasonic sensor).

Notes:

- This step may involve double-checking the servo's alignment and securing it permanently to the chassis.
- Be cautious not to disturb the 90° calibration of the servo horn.

Completion:
- The servo and cross horn are fully assembled on the chassis, ready for operation.

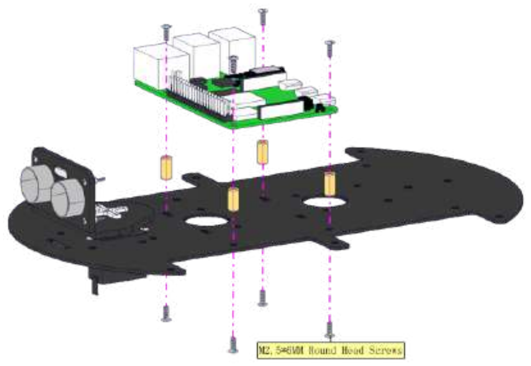
Step 10: Fix the Raspberry Pi Board

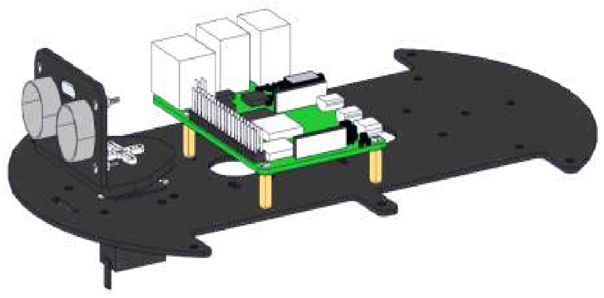
Objective: Mount the Raspberry Pi board onto the chassis.
Components:
- Raspberry Pi board (not included in the kit, typically a Raspberry Pi 3 or 4).
- Standoffs or spacers (likely the copper pillars from Step 5).
- Screws for securing the board.

Assembly Process:
- Gather the Raspberry Pi board and ensure it is powered off and disconnected.
- Position the Raspberry Pi over the copper pillars installed in Step 5, aligning its mounting holes with the pillars.
- Secure the board using screws, ensuring it is firmly attached but not over-tightened, which could damage the board.
- Check that the board's ports (USB, HDMI, GPIO, etc.) are accessible and not obstructed by other components.

Notes:
- The Raspberry Pi is the main controller for the smart car, so ensure it is securely mounted to avoid vibrations during operation.
- Verify that the board is level and stable on the pillars.

Completion:
- The Raspberry Pi is mounted on the chassis, ready for wiring.





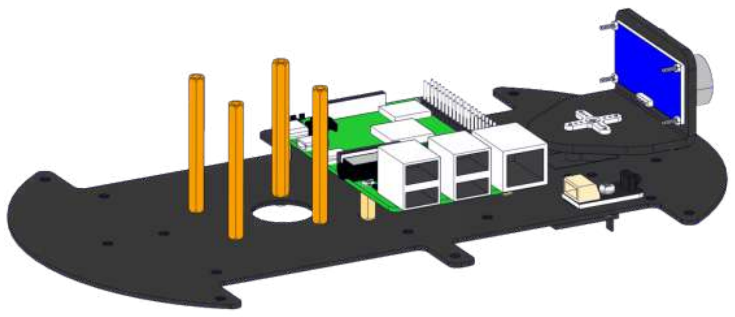
Step 11: Install Additional Components

Objective: Attach additional components to the chassis (specific components not detailed but likely sensors or modules).
Components:
- Unspecified components (possibly additional sensors, brackets, or structural parts).
- Fasteners for securing the components.

Assembly Process:
- Gather the components specified for this step (refer to the kit's manual or GitHub for details).
- Position each component on the chassis according to the instructions or silk print.
- Secure the components using screws or other fasteners, ensuring they are stable and correctly oriented.
- Check for clearance and ensure the components do not interfere with moving parts (e.g., wheels or servo).

Notes:
- Without specific component details, this step likely involves mounting secondary sensors or structural elements.
- Ensure all components are compatible with the Raspberry Pi and the kit's design.

Completion:
- Additional components are installed, furthering the car's functionality.



Step 12: Fix the Battery Holder

Objective: Install the battery holder to power the car.
Components:

- Battery holder (likely for AA batteries or a rechargeable pack).
- Screws or adhesive for securing the holder.

Assembly Process:

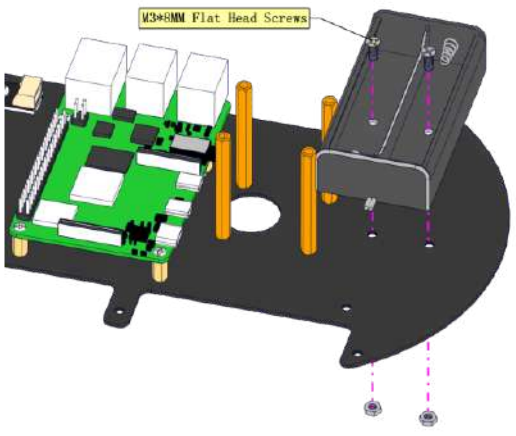
- Collect the battery holder and ensure it is empty (no batteries installed yet).
- Position the holder on the chassis as shown in the figures, typically in a central or rear location for weight balance.
- Secure the holder using screws or adhesive, ensuring it is firmly attached and the battery terminals are accessible.
- Verify that the holder's wiring can reach the power input for the Raspberry Pi or motor driver.

Notes:

- The battery holder powers the motors, Raspberry Pi, and other components, so its placement should optimize weight distribution.
- Ensure the holder is secure to prevent batteries from dislodging during movement.

Completion:

- The battery holder is installed, ready for battery insertion and wiring.



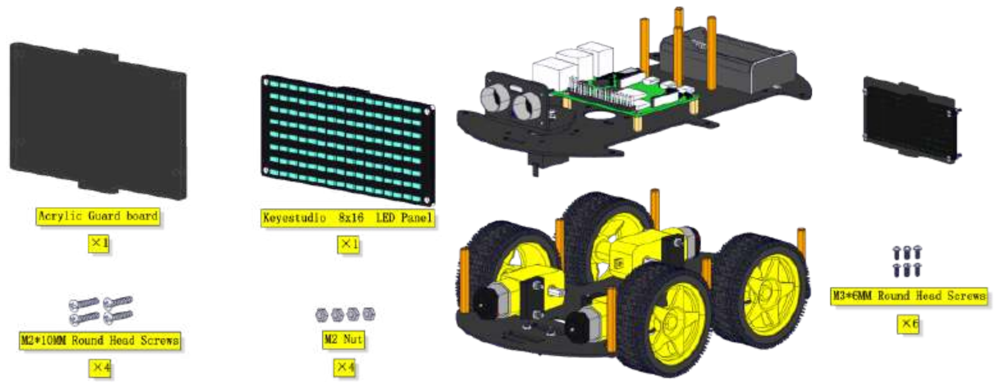
Step 13: Install Additional Components

Objective: Continue installing components (likely more sensors or modules).
Components:
- Unspecified components (refer to the kit's manual or GitHub).
- Fasteners for securing the components.

Assembly Process:
- Gather the components for this step.
- Position and secure each component on the chassis according to the instructions.
- Ensure proper alignment and stability, checking for interference with other parts.
- Verify that any wiring or connectors are accessible for later steps.

Notes:
- This step is likely a continuation of mounting secondary components, such as additional sensors or structural supports.
- Double-check the kit's documentation for specific component details.

Completion:
- Additional components are installed, enhancing the car's capabilities.



Step 14: Wire Up Sensors and Components

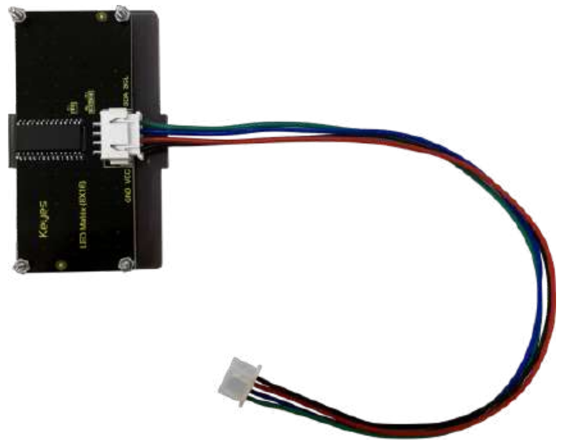
Objective: Connect the 8x16 dot matrix, line tracking sensor, and other components, routing wires through the chassis.
Components:
- 8x16 dot matrix LED panel.
- Line tracking sensor module.

- Wires for motors, servo, tracking sensor, and LED panel.
- Screws for final mounting.

Assembly Process:
- **8x16 Dot Matrix**:
    - Connect the dot matrix to its designated port on the chassis or driver board.
    - Secure the panel to the chassis using screws or brackets, ensuring it is visible and stable.
- **Line Tracking Sensor**:
    - Attach the line tracking sensor to the underside of the chassis, facing downward to detect lines on the ground.
    - Route the sensor's wires through the specified hole in the chassis to keep them organized.
- **Wire Routing**:
    - Pass the wires for the four motors, servo, tracking sensor, and LED panel through the designated gap or holes on the PCB.
    - Ensure wires are neatly arranged to avoid tangling or interference with moving parts.
- **Mounting Screw**:
    - Secure any remaining components or panels with screws to finalize the assembly.

Notes:
- The 8x16 dot matrix is used for displaying patterns or messages, so its placement should be prominent.
- The line tracking sensor enables the car to follow a path, so it must be positioned close to the ground and unobstructed.
- Proper wire management prevents damage and ensures reliable connections.

Completion:
- Sensors and the dot matrix are wired and mounted, with wires neatly routed.



Step 15: Mount Additional Components

Objective: Secure additional components with screws.
Components:

- Unspecified components (likely remaining sensors or structural parts).
- Screws for mounting.

Assembly Process:

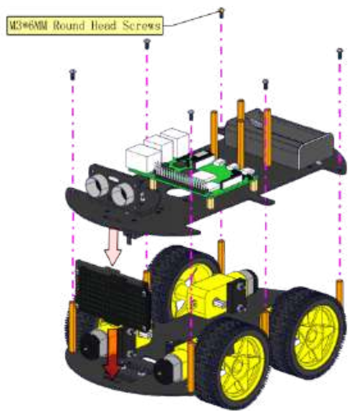
- Gather the components and screws for this step.
- Position each component on the chassis and secure it with screws.
- Ensure all components are stable and correctly aligned.
- Check for clearance and accessibility of wiring or connectors.

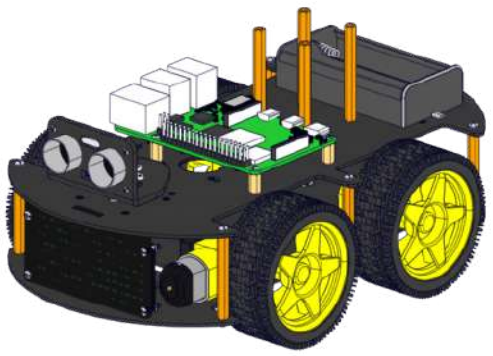
Notes:

- This step likely involves finalizing the mounting of components from previous steps.
- Verify that all components are securely fastened to withstand vibrations.

Completion:

- Additional components are mounted, completing part of the assembly.





Step 16: Install Camera Components

Objective: Install the camera module without disconnecting its ribbon cable.
Components:

- Camera module (likely a Raspberry Pi-compatible camera).
- Mounting bracket or holder.
- Fasteners for securing the camera.

Assembly Process:

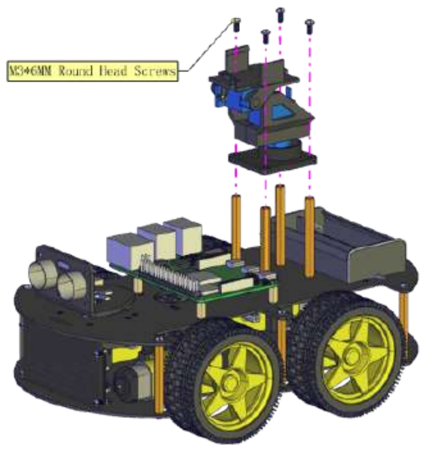
- Gather the camera module and its mounting hardware.
- Carefully position the camera on the chassis, ensuring the ribbon cable remains connected to the Raspberry Pi.
- Secure the camera to its mount using screws or clips, ensuring it is stable and oriented correctly (e.g., facing forward).
- Check that the camera's lens is unobstructed and the cable is not strained.

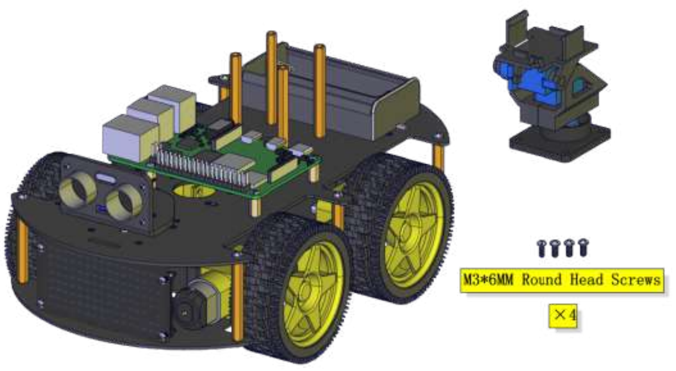
Notes:

- The camera is used for vision-based tasks, such as object detection or navigation, so its placement should provide a clear view.
- Handle the ribbon cable carefully to avoid damaging it or disconnecting it from the Raspberry Pi.

Completion:

- The camera is installed, with its connection intact.





Step 17: Assemble Additional Components

Objective: Install more components (likely related to the camera or sensors).
Components:

- Unspecified components (refer to the kit's manual).
- Fasteners for securing the components.

Assembly Process:

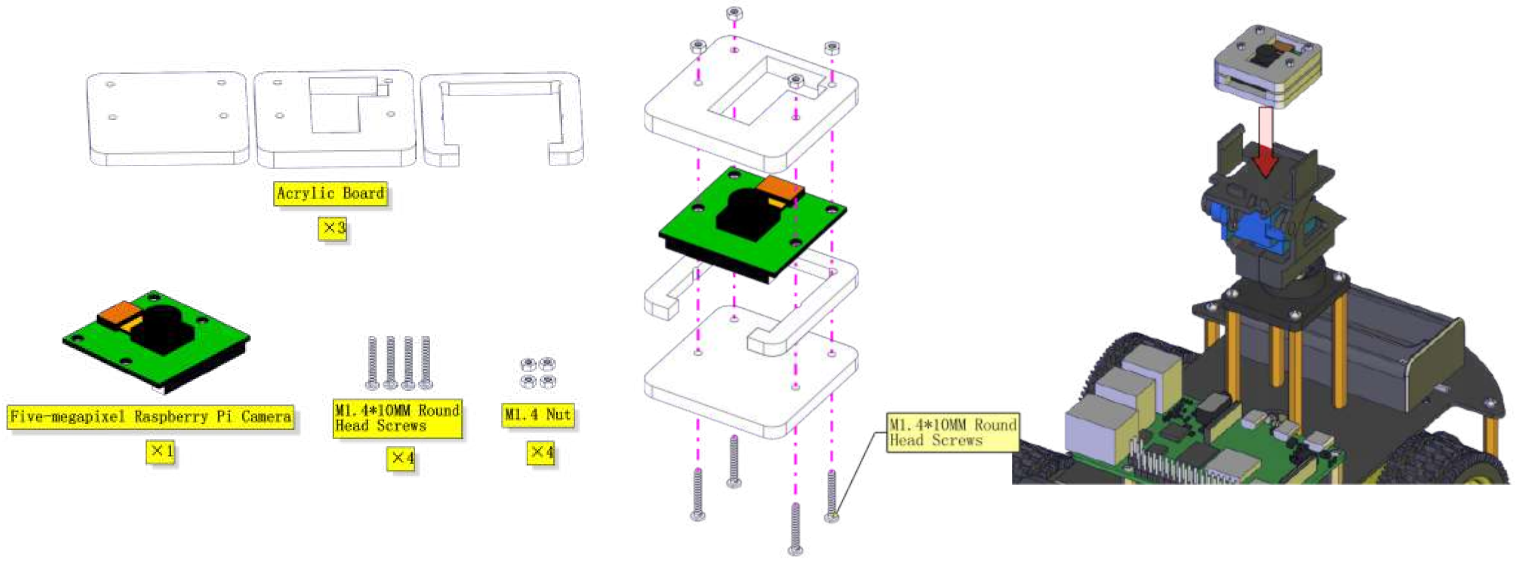
- Gather the components for this step.
- Position and secure each component on the chassis according to the instructions.
- Ensure stability and proper alignment, checking for interference with other parts.
- Verify that any wiring or connectors are accessible.

Notes:

- This step may involve mounting components that support the camera or other sensors.
- Check the kit's documentation for specific details.

Completion:

- Additional components are installed, furthering the car's functionality.


Acrylic Board
×3
Five-megapixel Raspberry Pi Camera
×1
M1.4*10MM Round Head Screws
×4
M1.4 Nut
×4
M1.4*10MM Round Head Screws

Step 18: Install Cooling Fan

Objective: Attach the cooling fan to keep the Raspberry Pi cool.
Components:

- Cooling fan (small DC fan).
- Screws or adhesive for securing the fan.

Assembly Process:

- Gather the cooling fan and ensure it is oriented correctly (logo side facing downward).
- Position the fan over the Raspberry Pi or a designated mounting area on the chassis.
- Secure the fan using screws or adhesive, ensuring it is stable and the airflow is directed toward the Raspberry Pi.
- Check that the fan's wiring is accessible for connection in later steps.

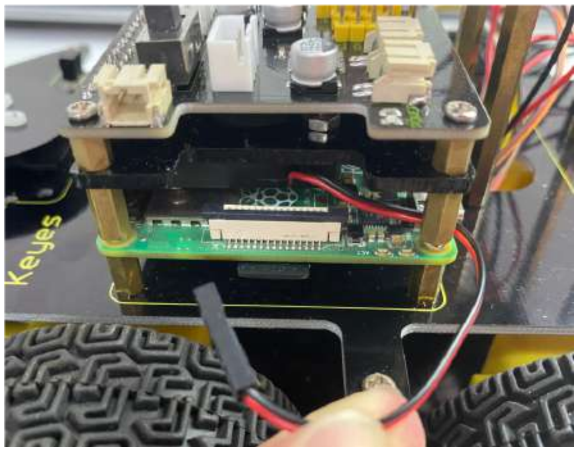
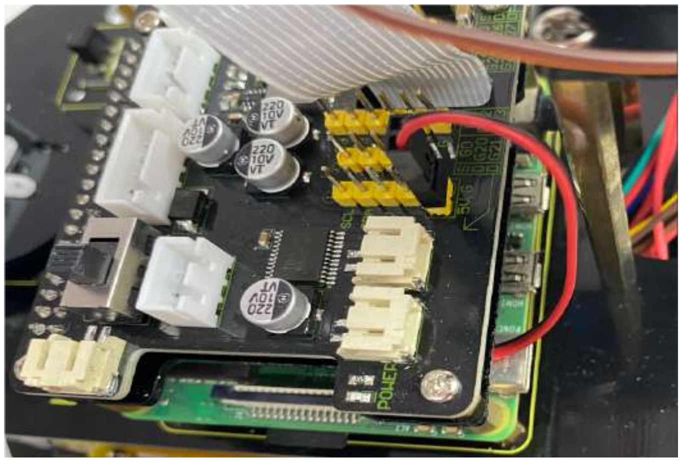
Notes:

- The cooling fan prevents the Raspberry Pi from overheating during operation, especially under heavy processing loads.
- The logo side facing downward likely indicates the direction of airflow, so follow this instruction carefully.

Completion:

- The cooling fan is installed, ready for wiring.

Expansion board	Cooling fan
5V	Red wire
G	Black wire

Step 19: Mount Additional Components

Objective: Install more components (likely final sensors or structural parts).
Components:

- Unspecified components.
- Fasteners for securing the components.

Assembly Process:

- Gather the components and fasteners for this step.
- Position and secure each component on the chassis.
- Ensure stability and alignment, checking for clearance and accessibility.
- Verify that all components are compatible with the car's design.

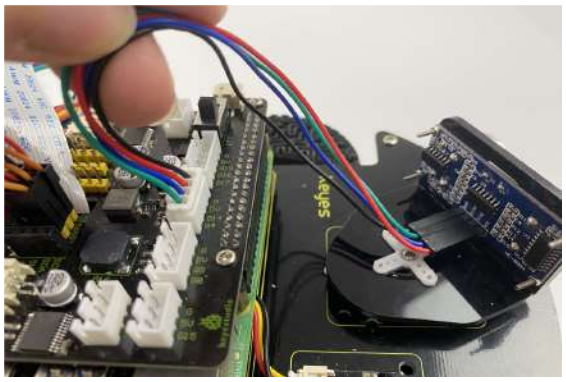
Notes:

- This step likely involves mounting the last few components before wiring.
- Double-check the kit's manual for specific component details.

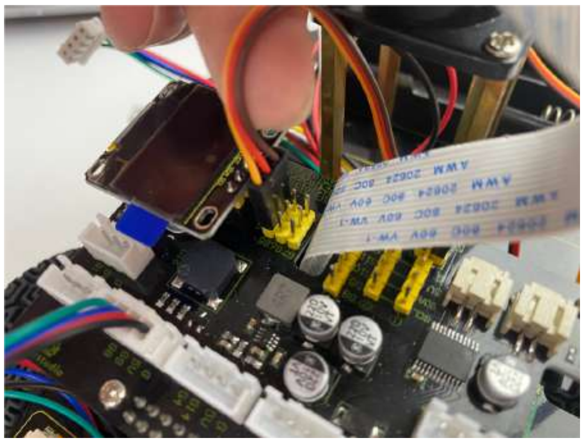
Completion:

- Additional components are mounted, nearing completion of the assembly.

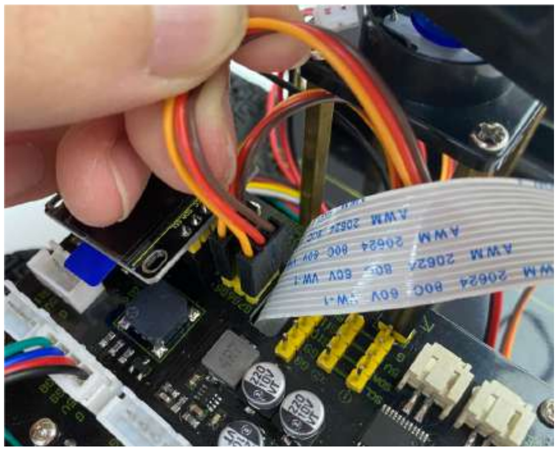
Expansion board	Ultrasonic module
G	Gnd
5V	Vcc
G14	Trig
G4	Echo



Expansion board	Servo for ultrasonic sensor
G5 (S)	Brown wire
5V	Red wire
G	Black wire



Expansion board	Servo on the base
G7 (S)	Brown wire
5V	Red wire
G	Black wire



Step 20: Wire Up the Camera

Objective: Connect the camera's ribbon cable to the Raspberry Pi.
Components:

- Camera module with ribbon cable.
- Raspberry Pi board.

Assembly Process:

- Ensure the camera's ribbon cable is intact and not damaged.
- Locate the camera connector on the Raspberry Pi (typically near the GPIO pins).
- Carefully insert the ribbon cable into the connector, ensuring the contacts face the correct direction (usually toward the board).
- Press down the black plastic latch on the connector to secure the cable.
- Verify that the cable is firmly connected and not loose.

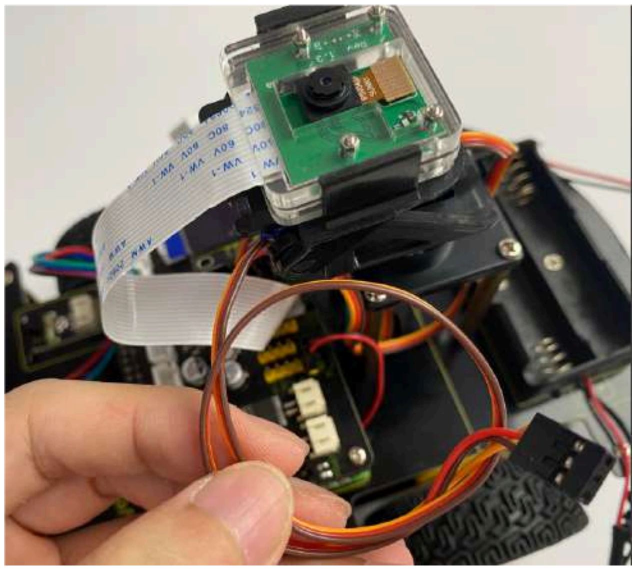
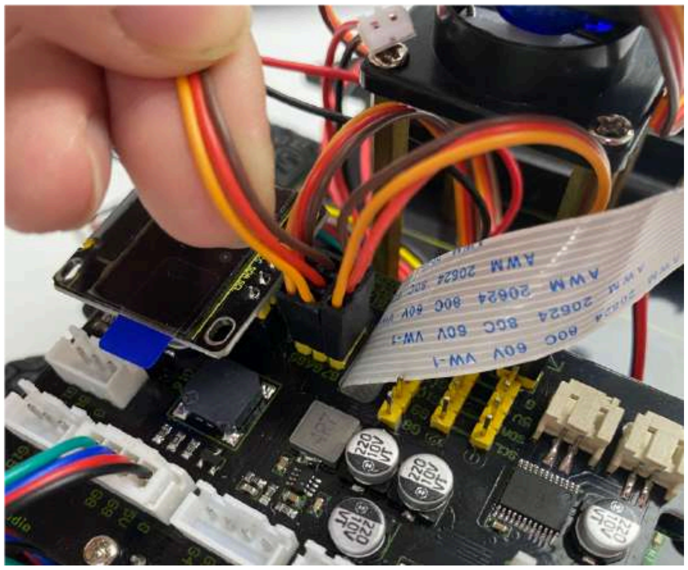
Notes:

- The ribbon cable is delicate, so handle it gently to avoid tearing or misaligning the contacts.
- Ensure the Raspberry Pi is powered off during this step to prevent damage.

Completion:

- The camera is wired to the Raspberry Pi, ready for testing.

Expansion board	Servo controlling the camera
G6(S)	Brown wire
5V	Red wire
G	Black wire

Step 21: Fix the OLED Display Module

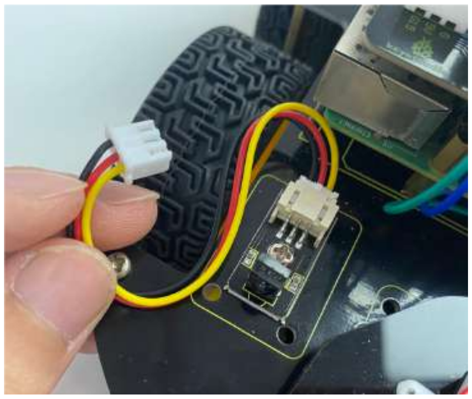
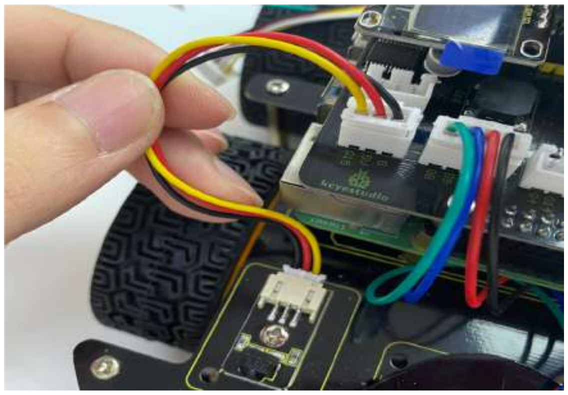
Objective: Install and wire the OLED display for visual output.
Components:

- OLED display module (likely a small I2C or SPI display).
- Mounting hardware or adhesive.
- Connection wires (if not already attached).

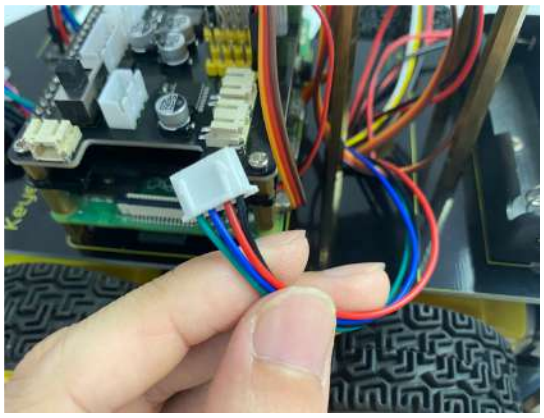
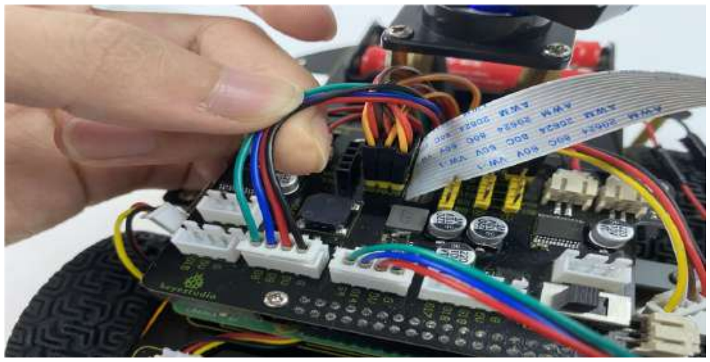
Assembly Process:

- Gather the OLED display and its mounting hardware.
- Position the display on the chassis, typically in a visible location (e.g., front or top).
- Secure the display using screws or adhesive, ensuring it is stable and the screen is unobstructed.
- Connect the display's wires to the Raspberry Pi or driver board (e.g., via GPIO pins for I2C/SPI communication).
- Inputs

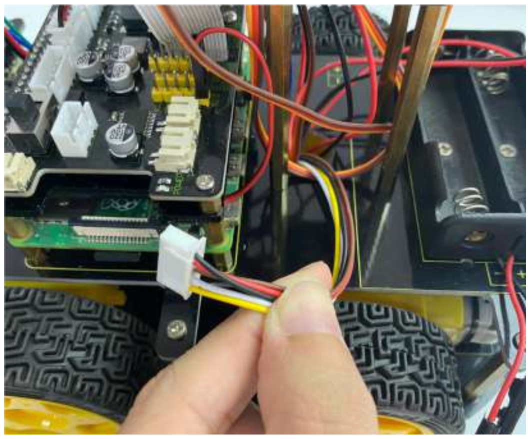
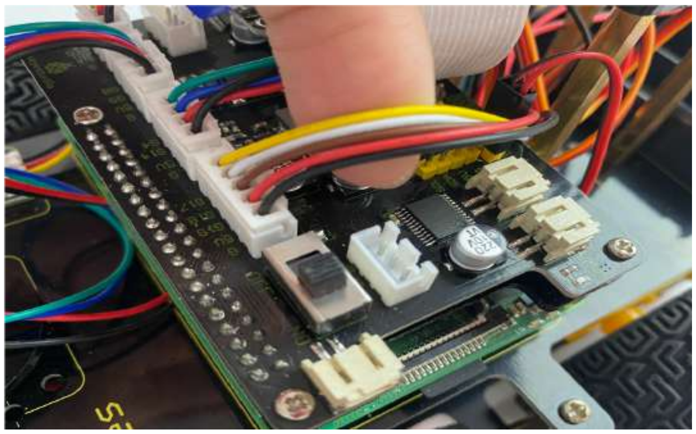
Expansion board	IR receiver module
G	G
5V	V
G15	S

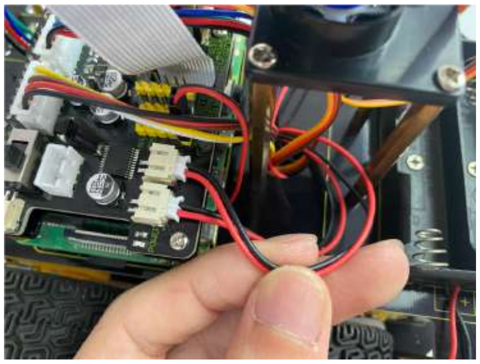
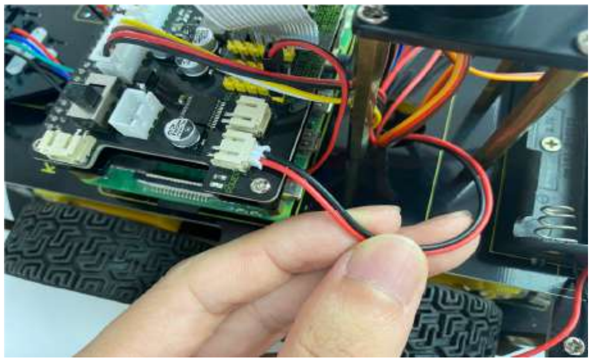
Expansion Board	8* 16 dot matrix LED panel
G	GND
5V	VCC
G8	SDA
G9	SCL

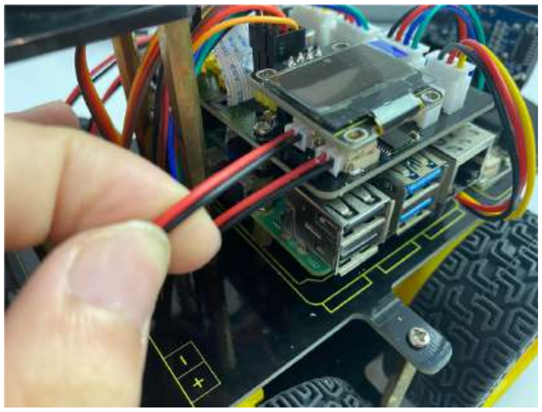
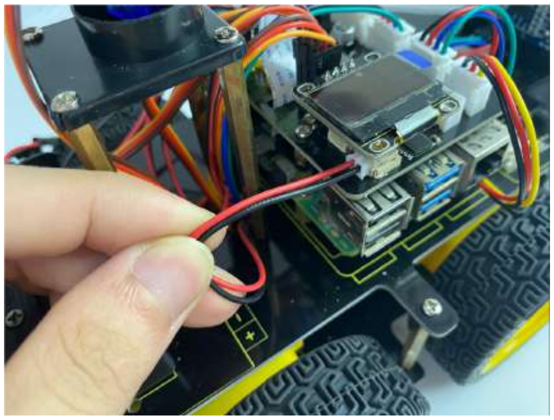
Expansion board	Line tracking sensor
G	S
5V	V
G19	R
G18	M
G17	L

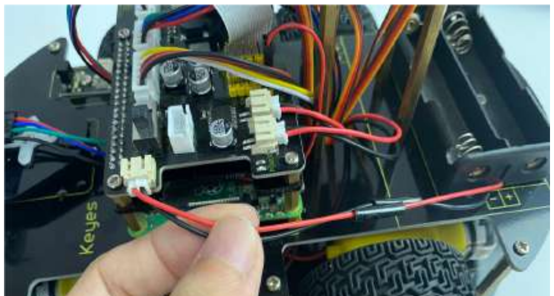
Motor 1: **Motor 2:**

Motor 3: **Motor 4:**

Wire up the power:



Part 2: Image Burning

Introduction

The image burning process involves installing an operating system (OS) onto a microSD card to enable the Raspberry Pi to control the Keyestudio KS0223 Smart Car. This section details the steps to prepare the Raspberry Pi by downloading, burning, and configuring the OS image, ensuring compatibility with the car's components, including motors, sensors, and the camera.

Materials

- Raspberry Pi (e.g., Raspberry Pi 3 or 4, not included in the kit)
- MicroSD card (8GB or larger, Class 10 recommended)
- MicroSD card reader
- Computer (Windows, macOS, or Linux)
- Internet connection
- Raspberry Pi Imager software

Procedure

Step 1: Prepare Materials

- Insert the microSD card into the card reader and connect it to the computer.
- Download and install Raspberry Pi Imager from https://www.raspberrypi.org/software/.
- Ensure the Raspberry Pi is powered off and disconnected.

Step 2: Download the OS Image

- Access the Raspberry Pi OS download page or the Keyestudio GitHub repository (https://github.com/keyestudio/KS0223-Keyestudio-Smart-Car-Kit-for-Raspberry-Pi).
- Download the recommended OS image (e.g., Raspberry Pi OS with Desktop or a custom Keyestudio image, if provided).
- Verify the image file's integrity using a checksum, if available.

Step 3: Burn the Image

- Launch Raspberry Pi Imager.

- Click “Choose OS” and select the downloaded image or Raspberry Pi OS.
- Select the microSD card under “Choose Storage.”
- Click “Write” to burn the image, then verify the write upon completion.

Step 4: Configure the MicroSD Card

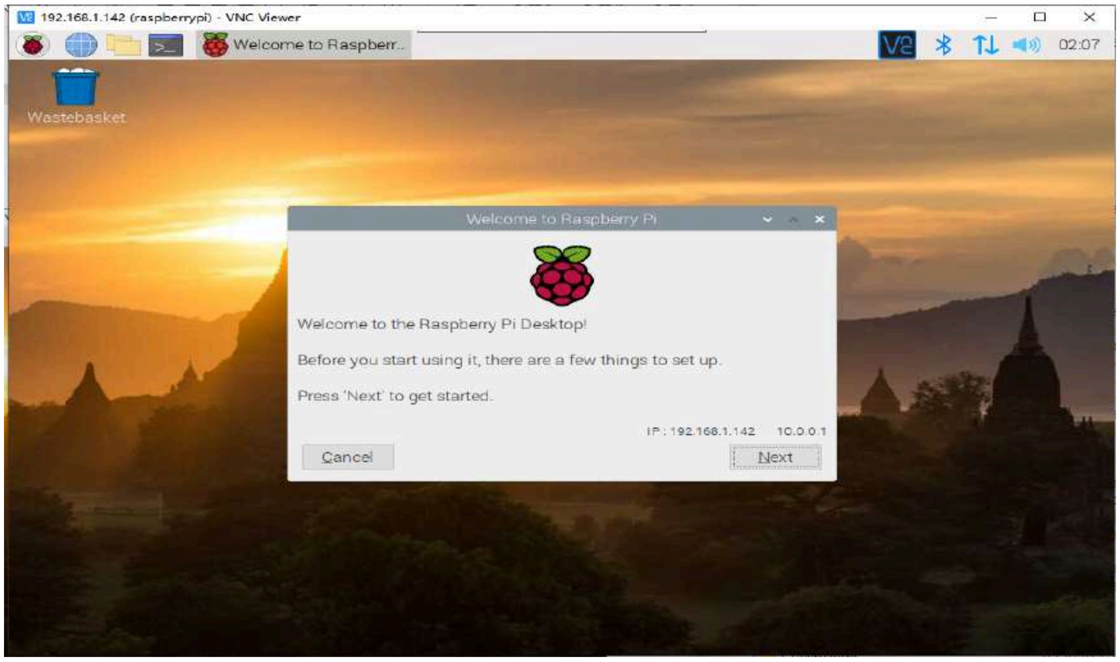
- Eject and reinsert the microSD card to access the boot partition.
- Enable SSH by creating an empty file named ssh in the boot partition.
- Configure Wi-Fi by creating a wpa_supplicant.conf file in the boot partition with the following:

```
country=US

ctrl_interface=DIR=/var/run/ wpa_supplicant GROUP=netdev

update_config=1

network={

   ssid="YOUR_WIFI_SSID"

   psk="YOUR_WIFI_PASSWORD"
```

- }
- Save and eject the microSD card.

Step 5: Boot the Raspberry Pi

- Insert the microSD card into the Raspberry Pi.
- Connect the Raspberry Pi to a monitor, keyboard, and mouse (if using Desktop OS) or prepare for headless setup via SSH.
- Power on the Raspberry Pi.
- Log in using default credentials (username: pi, password: raspberry).
- Confirm the OS boots successfully.



Welcome to the Raspberry Pi Desktop!

Before you start using it, there are a few things to set up.

Press 'Next' to get started.

IP : 192.168.1.142   10.0.0.1

Step 6: Install Dependencies

- Update the OS using:

sudo apt update

- sudo apt upgrade -y
- Install required libraries as specified in the Keyestudio documentation:

sudo apt install python3-pip

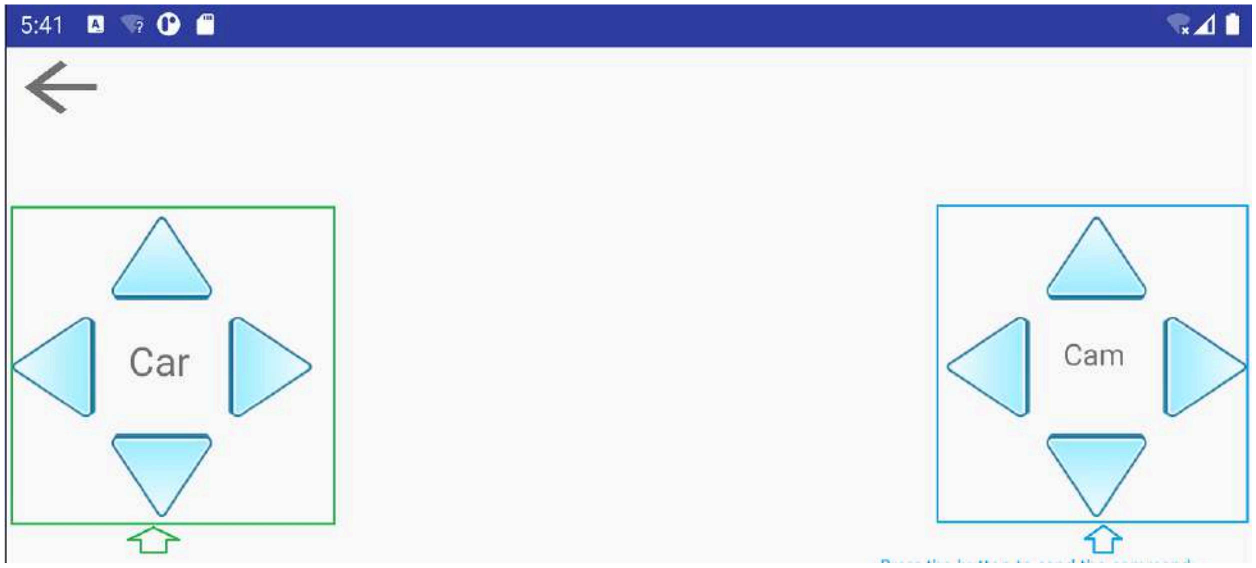
- sudo pip3 install RPi.GPIO
- Download the smart car code from the Keyestudio GitHub repository.
- Test a sample script to verify functionality.

Step 7: Download and Install the Keyes RPi Robot App

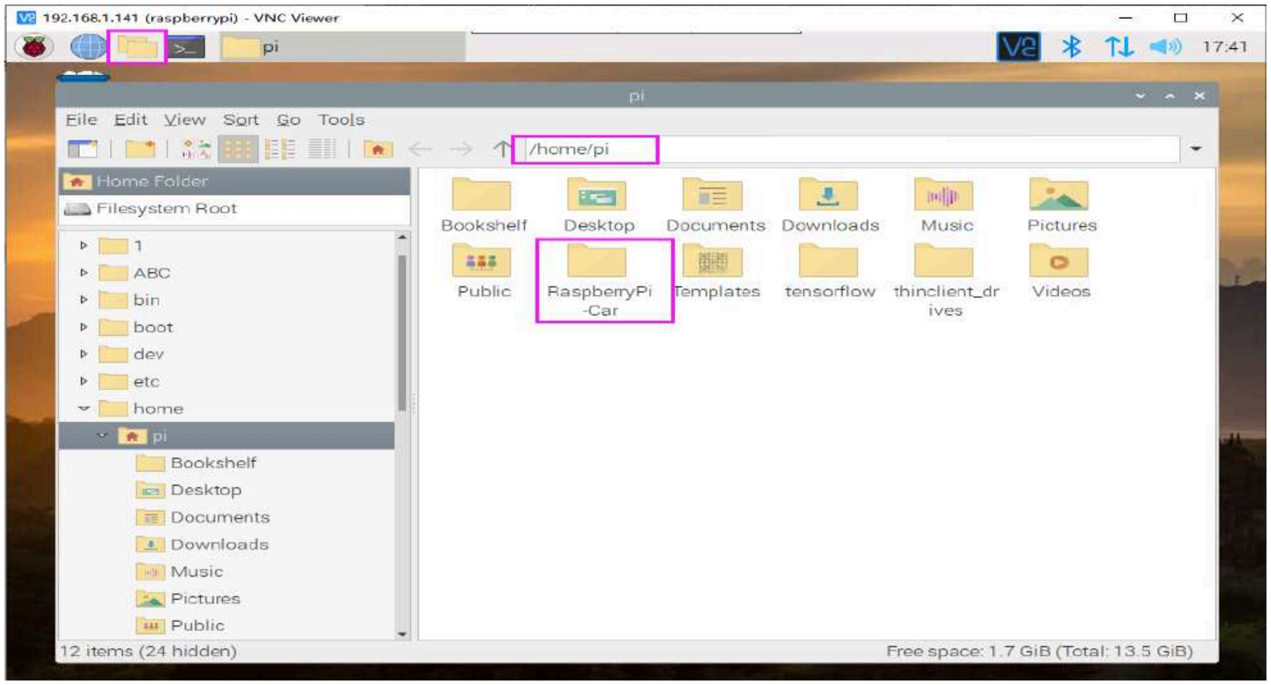
- Locate the keyes RPi Robot.apk file in the Keyestudio kit's provided folder or download the app from Google Play by searching for "keyes RPi Robot."
- Install the app on an Android smartphone.
- Enable the phone's Wi-Fi and connect to the Raspberry Pi's hotspot (Name: picar, Password: picar233).

Step 8: Configure the App Interface

- Determine the Raspberry Pi's IP address using the car's OLED display or WNetWatcher software on the computer.
- Open the Keyes RPi Robot app and enter the car's IP address and port number (50) in the designated fields.
- Click the "ENTER" button to connect.
- Verify the connection: the OLED display should show "connect," and the app should display the camera's monitoring screen.
- If the connection fails, reboot the Raspberry Pi and retry



.



Results

The Raspberry Pi successfully booted with the burned OS image, and the necessary dependencies were installed, preparing it to interface with the smart car's components.

Part 3: Raspberry Pi Hardware Control for Smart Car

Introduction

In this phase of the Keyestudio KS0223 Smart Car project, we conducted experiments to control the hardware components of the smart car using a Raspberry Pi. We programmed and tested individual components, including a passive buzzer, line tracking sensor, ultrasonic sensor, servos, 8x16 LED dot matrix, OLED display, motors, and infrared receiver, and integrated these components to achieve advanced functionalities such as infrared remote control, line tracking, ultrasonic following, and obstacle avoidance. We began by disabling boot-up programs to ensure the test scripts ran without interference, then executed Python scripts using BCM GPIO numbering to interface with each component. This section describes the procedures we performed, the results observed, and a summary of the Python codes used.

Experimental Procedure and Results

We started by disabling the Raspberry Pi’s boot-up programs to prevent conflicts with the test scripts. In the terminal, we accessed the /etc/rc.local file using the command sudo nano /etc/rc.local, commented out enabled scripts by adding a #, saved the changes, and exited. This ensured a clean environment for running the Python scripts.
We first tested the passive buzzer, connected to GPIO 16. Using the script bp1_1buzzer.py, we programmed the buzzer to emit a repetitive “tick, tick” sound, confirming its basic functionality. We then ran bp1_2buzzer_pwm.py, which used PWM to play the “Happy Birthday” tune, demonstrating the buzzer’s ability to produce musical notes. Both scripts were executed from the /home/pi/RaspberryPi-Car/basic_project directory, and we terminated each test by pressing Ctrl + C.
Next, we tested the three-channel line tracking sensor, with pins connected to GPIO 19 (left), GPIO 18 (middle), and GPIO 17 (right). The script bp2_tracking.py read sensor values, displaying 0 when the car was on a surface and 1 when detecting black lines. This verified the sensor’s capability to detect paths, crucial for the line-tracking functionality.
For distance measurement, we used the ultrasonic sensor, connected to GPIO 14 (Trig) and GPIO 4 (Echo). The script bp3_ultrasonic.py measured distances and displayed them in centimeters on the terminal, confirming the sensor’s accuracy for obstacle detection and following tasks.
We then tested the servo motors, which controlled the ultrasonic sensor (GPIO 5) and camera pan-tilt (GPIO 7 and GPIO 6). Using bp4_servo_test.py, we set the ultrasonic sensor servo to a 90° position, and with bp4_2servo.py, we rotated the camera pan-tilt servos through 0° to 180°. These tests demonstrated precise angular control, essential for positioning sensors and the camera.
The 8x16 LED dot matrix, connected to GPIO 8 (SCL) and GPIO 9 (SDA), was programmed with bp5_LED8X16_TM1604.py. We displayed a smile pattern followed by directional arrows (forward, back, left, right), confirming the matrix’s ability to show visual indicators for car movements.
The 0.96-inch OLED display, using I2C communication with a 128x64 resolution, was tested in two configurations. The script bp6_oled_stats.py, run from the /home/pi/RaspberryPi-Car/basic_project/OledModule directory, displayed the Raspberry Pi’s IP address, CPU load, memory usage, and disk usage. The script bp6_2oled_ultrasonic.py showed ultrasonic sensor distances on both the OLED and terminal, validating the display’s utility for real-time data output.
Motor control was achieved using two TB6612 drivers for the four motors, with pins for M2 (GPIO 20, 21, 0), M1 (GPIO 22, 23, 1), M3 (GPIO 24, 25, 12), and M4 (GPIO 26, 27, 13). We executed bp7_motor_test.py to drive the car forward, lifting the car to prevent cable damage, which confirmed motor functionality.

The infrared receiver, connected to GPIO 15, was tested with bp8_ir_remote.py. By pointing an infrared remote at the receiver and pressing keys, we observed the terminal displaying corresponding values (e.g., “Button up”), verifying reliable signal detection.
We integrated the motors, LED dot matrix, and infrared receiver to create an infrared remote-controlled car using bp9_ir_car.py. Pressing remote keys (up, down, left, right, OK) controlled the car’s movement and displayed matching patterns on the LED matrix, demonstrating coordinated system operation.
For the line-tracking car, we used bp10_tracking_car.py, which combined motor control and line tracking sensor inputs. When placed on a tracking map, the car followed black lines, moving forward when the middle sensor detected a line, turning PARP left or right based on side sensors, and stopping when no lines were detected. This confirmed autonomous path-following capability.
The ultrasonic following car was implemented with bp11_follow_car.py. Using the ultrasonic sensor and motors, the car followed a hand, moving forward at distances of 14–50 cm, stopping at 10–14 cm, and moving backward if closer than 10 cm. This validated distance-based tracking.
Finally, we created an obstacle-avoidance robot with bp12_avoid_car.py. The car moved forward if obstacles were beyond 15 cm. If closer, the ultrasonic sensor’s servo rotated left (180°) and right (0°) to measure distances, and the car turned toward the farther side. This demonstrated autonomous navigation around obstacles.

Summary of Python Codes

We utilized 12 Python scripts, each serving a distinct purpose:

- bp1_1buzzer.py: Generated “tick, tick” sounds with the buzzer.
- bp1_2buzzer_pwm.py: Played “Happy Birthday” using PWM.
- bp2_tracking.py: Read line tracking sensor values.
- bp3_ultrasonic.py: Measured distances with the ultrasonic sensor.
- bp4_servo_test.py: Controlled the ultrasonic sensor servo.
- bp4_2servo.py: Controlled camera pan-tilt servos.
- bp5_LED8X16_TM1604.py: Displayed patterns on the LED dot matrix.
- bp6_oled_stats.py: Showed system stats on the OLED display.
- bp6_2oled_ultrasonic.py: Displayed ultrasonic distances on the OLED.
- bp7_motor_test.py: Drove the car’s motors forward.
- bp8_ir_remote.py: Detected infrared remote signals.
- bp9_ir_car.py, bp10_tracking_car.py, bp11_follow_car.py, bp12_avoid_car.py: Integrated multiple components for remote control, line tracking, following, and obstacle avoidance, respectively.

The hardware control experiments successfully demonstrated the Raspberry Pi's ability to interface with the smart car's components, achieving both individual component functionality and integrated autonomous behaviors. The results validated the car's potential for applications in robotics and automation.

Code Listings

1. Passive Buzzer - Tick Sound (bp1_1buzzer.py)

python
```
# -*- coding: utf-8 -*-
import time
import RPi.GPIO as GPIO

buzPin = 16
i1 = 0
i2 = 0
GPIO.setmode(GPIO.BCM)  # use BCM numbers
GPIO.setup(buzPin, GPIO.OUT) # set pin OUTPUT mode

try:
    while 1:  #loop
        while(i1<50):
            GPIO.output(buzPin,GPIO.HIGH)
            time.sleep(0.001)            #wait for 1 ms
            GPIO.output(buzPin,GPIO.LOW)
            time.sleep(0.001)
            i1 = i1 + 1
        time.sleep(0.3)
        while(i2<50):
            GPIO.output(buzPin,GPIO.HIGH)
            time.sleep(0.001)            #wait for 1 ms
            GPIO.output(buzPin,GPIO.LOW)
            time.sleep(0.001)
            i2 = i2 + 1
        time.sleep(1)
        i1 = 0
        i2 = 0
except KeyboardInterrupt:
    pass
GPIO.cleanup() #release all GPIO
```

2. Passive Buzzer - Birthday Song (bp1_2buzzer_pwm.py**)**

```
python
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time

Buzzer = 16  # set the Pin

# Happy birthday
Do = 262
Re = 294
Mi = 330
Fa = 349
Sol = 392
La = 440
Si = 494
Do_h = 523
Re_h = 587
Mi_h = 659
Fa_h = 698
Sol_h = 784
La_h = 880
Si_h = 988

# The tune
song_1 = [
    Sol,Sol,La,Sol,Do_h,Si,
    Sol,Sol,La,Sol,Re_h,Do_h,
    Sol,Sol,Sol_h,Mi_h,Do_h,Si,La,
    Fa_h,Fa_h,Mi_h,Do_h,Re_h,Do_h
]
# delay
beat_1 = [
    0.5,0.5,1,1,1,1+1,
    0.5,0.5,1,1,1,1+1,
    0.5,0.5,1,1,1,1,1,
    0.5,0.5,1,1,1,1+1,
]

def setup():
    GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by physical location
    GPIO.setup(Buzzer, GPIO.OUT)    # Set pins' mode is output
    global Buzz                    # Assign a global variable to replace GPIO.PWM
    Buzz = GPIO.PWM(Buzzer, 440)    # 440 is initial frequency.
    Buzz.start(50)                  # Start Buzzer pin with 50% duty ration

def loop():
    while True:
        print('\n    Playing song 3...')
        for i in range(0, len(song_1)):     # Play song 1
            Buzz.ChangeFrequency(song_1[i]) # Change the frequency along the song note
            time.sleep(beat_1[i] * 0.5)     # delay a note for beat * 0.5s

def destory():
    Buzz.stop()                # Stop the buzzer
    GPIO.output(Buzzer, 1)      # Set Buzzer pin to High
    GPIO.cleanup()              # Release resource

if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # When 'Ctrl+C' is pressed, the child program destroy()
will be  executed.
        destory()
```

3. Line Tracking Sensor (bp2_tracking.py**)**

```
python
import RPi.GPIO as GPIO
from time import sleep

#tracking pin
trackingPin1 = 17
```

```
trackingPin2 = 18
trackingPin3 = 19

GPIO.setmode(GPIO.BCM) # use BCM numbers

GPIO.setup(trackingPin1,GPIO.IN)  # set trackingPin INPUT mode
GPIO.setup(trackingPin2,GPIO.IN)
GPIO.setup(trackingPin3,GPIO.IN)

while True:
    val1 = GPIO.input(trackingPin1) # read the value
    val2 = GPIO.input(trackingPin2)
    val3 = GPIO.input(trackingPin3)
    print("tracking1 = ", val1, "tracking2 = ", val2, "tracking3 = ", val3)
    sleep(0.1)

GPIO.cleanup() # Release all GPIO
```

4. Ultrasonic Sensor (bp3_ultrasonic.py**)**

```
python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

#define GPIO pin
GPIO_TRIGGER = 14
GPIO_ECHO = 4

#set GPIO mode (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # 10us is the trigger signal
    GPIO.output(GPIO_TRIGGER, GPIO.HIGH)
    time.sleep(0.00001)  #10us
    GPIO.output(GPIO_TRIGGER, GPIO.LOW)
    while not GPIO.input(GPIO_ECHO):
        pass
    t1 = time.time()
    while GPIO.input(GPIO_ECHO):
        pass
    t2 = time.time()
    print("distance is %d " % (((t2 - t1)* 340 / 2) * 100))
    time.sleep(0.01)
    return ((t2 - t1)* 340 / 2) * 100


if __name__ == '__main__':   #Program entry
    try:
        while True:
            print("aaaa")
            dist = distance()  #
            print("Measured Distance = {:.2f} cm".format(dist)) #{:.2f},Keep two decimal
places
            time.sleep(0.01)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Measurement stopped by User")
        GPIO.cleanup()
```

5. Servo Control - Ultrasonic Sensor (bp4_servo_test.py**)**

```
python
import RPi.GPIO as GPIO
import time

servoPin1 = 5
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

def init():
    GPIO.setup(servoPin1, GPIO.OUT)
```

```
def servoPulse(servoPin, myangle):
    pulsewidth = (myangle*11) + 500  # The pulse width
    GPIO.output(servoPin,GPIO.HIGH)
    time.sleep(pulsewidth/1000000.0)
    GPIO.output(servoPin,GPIO.LOW)
    time.sleep(20.0/1000 - pulsewidth/1000000.0) # The cycle of 20 ms

try:
    init()
    while True:
        servoPulse(servoPin1, 90)
        '''
        for i in range(0,180):
            servoPulse(servoPin1, i)

        for i in range(0,180):
            i = 180 - i
            servoPulse(servoPin1, i)
        '''
except KeyboardInterrupt:
    pass
GPIO.cleanup()
```

6. Servo Control - Camera Pan-Tilt (bp4_2servo.py**)**

```
python
import RPi.GPIO as GPIO
import time

servoPin2 = 7
servoPin3 = 6
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

def init():
    GPIO.setup(servoPin2, GPIO.OUT)
    GPIO.setup(servoPin3, GPIO.OUT)

def servoPulse(servoPin, myangle):
    pulsewidth = (myangle*11) + 500  # The pulse width
    GPIO.output(servoPin,GPIO.HIGH)
    time.sleep(pulsewidth/1000000.0)
    GPIO.output(servoPin,GPIO.LOW)
    time.sleep(20.0/1000 - pulsewidth/1000000.0) # The cycle of 20 ms

try:
    init()
    while True:
        for i in range(0,180):
            servoPulse(servoPin2, i)
        for i in range(0,180):
            servoPulse(servoPin3, i)

        for i in range(0,180):
            i = 180 - i
            servoPulse(servoPin2, i)
        for i in range(0,180):
            i = 180 - i
            servoPulse(servoPin3, i)

        for j in range(0, 50):
            servoPulse(servoPin2, 90)
        for j in range(0, 50):
            servoPulse(servoPin3, 90)
        time.sleep(2)
except KeyboardInterrupt:
    pass
GPIO.cleanup()
```

7. 8x16 LED Dot Matrix (bp5_LED8X16_TM1604.py**)**

```
python
import RPi.GPIO as GPIO
import time

SCLK = 8
DIO  = 9
# Display pattern data
```

```
smile = (0x00, 0x00, 0x38, 0x40, 0x40, 0x40, 0x3a, 0x02, 0x02, 0x3a, 0x40, 0x40, 0x40,
0x38, 0x00, 0x00)
matrix_forward = (0x00, 0x00, 0x00, 0x00, 0x12, 0x24, 0x48, 0x90, 0x90, 0x48, 0x24,
0x12, 0x00, 0x00, 0x00, 0x00)
matrix_back = (0x00, 0x00, 0x00, 0x00, 0x48, 0x24, 0x12, 0x09, 0x09, 0x12, 0x24,
0x48, 0x00, 0x00, 0x00, 0x00)
matrix_left = (0x00, 0x00, 0x00, 0x00, 0x18, 0x24, 0x42, 0x99, 0x24, 0x42, 0x81, 0x00,
0x00, 0x00, 0x00, 0x00)
matrix_right = (0x00, 0x00, 0x00, 0x00, 0x00, 0x81, 0x42, 0x24, 0x99, 0x42, 0x24,
0x18, 0x00, 0x00, 0x00, 0x00)

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(SCLK,GPIO.OUT)
GPIO.setup(DIO,GPIO.OUT)

def nop():
   time.sleep(0.00003)

def start():
   GPIO.output(SCLK,0)
   nop()
   GPIO.output(SCLK,1)
   nop()
   GPIO.output(DIO,1)
   nop()
   GPIO.output(DIO,0)
   nop()

def matrix_clear():
   GPIO.output(SCLK,0)
   nop()
   GPIO.output(DIO,0)
   nop()
   GPIO.output(DIO,0)
   nop()

def send_date(date):
   for i in range(0,8):
      GPIO.output(SCLK,0)
```

```
        nop()
        if date & 0x01:
            GPIO.output(DIO,1)
        else:
            GPIO.output(DIO,0)
        nop()
        GPIO.output(SCLK,1)
        nop()
        date >>= 1
        GPIO.output(SCLK,0)

def end():
    GPIO.output(SCLK,0)
    nop()
    GPIO.output(DIO,0)
    nop()
    GPIO.output(SCLK,1)
    nop()
    GPIO.output(DIO,1)
    nop()

def matrix_display(matrix_value):
    start()
    send_date(0xc0)

    for i in range(0,16):
        send_date(matrix_value[i])

    end()
    start()
    send_date(0x8A)
    end()

try:
    while True:
        matrix_display(smile)
        time.sleep(1)
        matrix_display(matrix_back)
        time.sleep(1)
        matrix_display(matrix_forward)
        time.sleep(1)
        matrix_display(matrix_left)
        time.sleep(1)
        matrix_display(matrix_right)
        time.sleep(1)
except KeyboardInterrupt:
    GPIO.cleanup()
```

8. OLED Display - System Stats (bp6_oled_stats.py**)**

```
python
import time
import Adafruit_GPIO.SPI as SPI
import Adafruit_SSD1306
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
import subprocess

# Raspberry Pi pin configuration:
RST = None     # on the PiOLED this pin isnt used
# Note the following are only used with SPI:
DC = 23
SPI_PORT = 0
SPI_DEVICE = 0

# 128x64 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)

# Initialize library.
disp.begin()

# Clear display.
disp.clear()
disp.display()

# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
width = disp.width
height = disp.height
```

```
image = Image.new('1', (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0,0,width,height), outline=0, fill=0)

# Draw some shapes.
# First define some constants to allow easy resizing of shapes.
padding = -2
top = padding
bottom = height-padding
# Move left to right keeping track of the current x position for drawing shapes.
x = 0

# Load default font.
font = ImageFont.load_default()

while True:
    # Draw a black filled box to clear the image.
    draw.rectangle((0,0,width,height), outline=0, fill=0)

    # Shell scripts for system monitoring from here :
https://unix.stackexchange.com/questions/119126/command-to-display-memory-usage-
disk-usage-and-cpu-load
    cmd = "hostname -I | cut -d\' \' -f1"
    IP = subprocess.check_output(cmd, shell = True )
    cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell = True )
    cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%sMB %.2f%%\", $3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell = True )
    cmd = "df -h | awk '$NF==\"/\"{printf \"Disk: %d/%dGB %s\", $3,$2,$5}'"
    Disk = subprocess.check_output(cmd, shell = True )

    # Write two lines of text.
    draw.text((x, top),       "IP: " + str(IP),  font=font, fill=255)
    draw.text((x, top+8),     str(CPU), font=font, fill=255)
    draw.text((x, top+16),    str(MemUsage),  font=font, fill=255)
    draw.text((x, top+25),    str(Disk),  font=font, fill=255)
    # Display image.
    disp.image(image)
    disp.display()
    time.sleep(.1)
```

9. OLED Display - Ultrasonic Distance (bp6_2oled_ultrasonic.py**)**

```
python
import time
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

import Adafruit_GPIO.SPI as SPI
import Adafruit_SSD1306
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

#define GPIO pin
GPIO_TRIGGER = 14
GPIO_ECHO = 4
#set GPIO mode (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # 10us is the trigger signal
    GPIO.output(GPIO_TRIGGER, GPIO.HIGH)
    time.sleep(0.00001)  #10us
    GPIO.output(GPIO_TRIGGER, GPIO.LOW)
    while not GPIO.input(GPIO_ECHO):
        pass
    t1 = time.time()
    while GPIO.input(GPIO_ECHO):
        pass
    t2 = time.time()
    print("distance is %d " % (((t2 - t1)* 340 / 2) * 100))
    time.sleep(0.01)
    return ((t2 - t1)* 340 / 2) * 100

import subprocess

# Raspberry Pi pin configuration:
RST = None     # on the PiOLED this pin isnt used
# Note the following are only used with SPI:
DC = 23
SPI_PORT = 0
SPI_DEVICE = 0

# 128x64 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)

# Initialize library.
disp.begin()

# Clear display.
disp.clear()
disp.display()

# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
width = disp.width
height = disp.height
image = Image.new('1', (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0,0,width,height), outline=0, fill=0)

# Draw some shapes.
# First define some constants to allow easy resizing of shapes.
padding = -2
top = padding
bottom = height-padding
# Move left to right keeping track of the current x position for drawing shapes.
```

```
x = 0

# Load default font.
font = ImageFont.load_default()

while True:
    dist = int(distance())  # round numbers
    print("Measured Distance = {:.2f} cm".format(dist)) #{:.2f},Keep two decimal places

    # Draw a black filled box to clear the image.
    draw.rectangle((0,0,width,height), outline=0, fill=0)
    # Write two lines of text.
    draw.text((x, top),       "Distance: " ,  font=font, fill=255)
    draw.text((x, top+8),     str(dist), font=font, fill=255)

    # Display image.
    disp.image(image)
    disp.display()
    time.sleep(.1)
```

10. Motor Control (bp7_motor_test.py**)**

```
python
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

# Control M2 motor
L_IN1 = 20
L_IN2 = 21
L_PWM1 = 0
# Control M1 motor
L_IN3 = 22
L_IN4 = 23
L_PWM2 = 1
# Control M3 motor
R_IN1 = 24
R_IN2 = 25
R_PWM1 = 12
# Control M4 motor
```

```
R_IN3 = 26
R_IN4 = 27
R_PWM2 = 13

GPIO.setmode(GPIO.BCM)  # use BCM numbers
#set the MOTOR Driver Pin OUTPUT mode
GPIO.setup(L_IN1,GPIO.OUT)
GPIO.setup(L_IN2,GPIO.OUT)
GPIO.setup(L_PWM1,GPIO.OUT)

GPIO.setup(L_IN3,GPIO.OUT)
GPIO.setup(L_IN4,GPIO.OUT)
GPIO.setup(L_PWM2,GPIO.OUT)

GPIO.setup(R_IN1,GPIO.OUT)
GPIO.setup(R_IN2,GPIO.OUT)
GPIO.setup(R_PWM1,GPIO.OUT)

GPIO.setup(R_IN3,GPIO.OUT)
GPIO.setup(R_IN4,GPIO.OUT)
GPIO.setup(R_PWM2,GPIO.OUT)


GPIO.output(L_IN1,GPIO.LOW)
GPIO.output(L_IN2,GPIO.LOW)
GPIO.output(L_IN3,GPIO.LOW)
GPIO.output(L_IN4,GPIO.LOW)

GPIO.output(R_IN1,GPIO.LOW)
GPIO.output(R_IN2,GPIO.LOW)
GPIO.output(R_IN3,GPIO.LOW)
GPIO.output(R_IN4,GPIO.LOW)


#set pwm frequence to 1000 caloric
pwm_R1 = GPIO.PWM(R_PWM1,100)
pwm_R2 = GPIO.PWM(R_PWM2,100)
pwm_L1 = GPIO.PWM(L_PWM1,100)
pwm_L2 = GPIO.PWM(L_PWM2,100)
```

```
#set inital duty cycle to 0
pwm_R1.start(0)
pwm_L1.start(0)
pwm_R2.start(0)
pwm_L2.start(0)

while True:
   GPIO.output(L_IN1,GPIO.LOW)  #Upper Left forward
   GPIO.output(L_IN2,GPIO.HIGH)
   pwm_L1.ChangeDutyCycle(50)
   GPIO.output(L_IN3,GPIO.HIGH)  #Lower left forward
   GPIO.output(L_IN4,GPIO.LOW)
   pwm_L2.ChangeDutyCycle(50)
   GPIO.output(R_IN1,GPIO.HIGH)  #Upper Right forward
   GPIO.output(R_IN2,GPIO.LOW)
   pwm_R1.ChangeDutyCycle(50)
   GPIO.output(R_IN3,GPIO.LOW)  #Lower Right forward
   GPIO.output(R_IN4,GPIO.HIGH)
   pwm_R2.ChangeDutyCycle(50)

#stop pwm
pwm_R1.stop()
pwm_L1.stop()
pwm_R2.stop()
pwm_L2.stop()
time.sleep(1)

GPIO.cleanup()  #release all GPIO
```

11. Infrared Receiver (bp8_ir_remote.py**)**

```
python
import RPi.GPIO as GPIO
import time

PIN = 15;

GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN,GPIO.IN,GPIO.PUD_UP)
print("irm test start...")
```

```
def exec_cmd(key_val):
    if(key_val==0x46):
        print("Button up")
    elif(key_val==0x44):
        print("Button left")
    elif(key_val==0x40):
        print("Button ok")
    elif(key_val==0x43):
        print("Button right")
    elif(key_val==0x15):
        print("Button down")
    elif(key_val==0x16):
        print("Button 1")
    elif(key_val==0x19):
        print("Button 2")
    elif(key_val==0x0d):
        print("Button 3")
    elif(key_val==0x0c):
        print("Button 4")
    elif(key_val==0x18):
        print("Button 5")
    elif(key_val==0x5e):
        print("Button 6")
    elif(key_val==0x08):
        print("Button 7")
    elif(key_val==0x1c):
        print("Button 8")
    elif(key_val==0x5a):
        print("Button 9")
    elif(key_val==0x42):
        print("Button *")
    elif(key_val==0x52):
        print("Button 0")
    elif(key_val==0x4a):
        print("Button #")

try:
    while True:
        if GPIO.input(PIN) == 0:
```

```
        count = 0
        while GPIO.input(PIN) == 0 and count < 200:  # Wait for 9ms LOW level boot 
code and exit the loop if it exceeds 1.2ms
           count += 1
           time.sleep(0.00006)

        count = 0
        while GPIO.input(PIN) == 1 and count < 80:   # Wait for a 4.5ms HIGH level boot 
code and exit the loop if it exceeds 0.48ms
           count += 1
           time.sleep(0.00006)

        idx = 0  # byte count variable
        cnt = 0  #Variable per byte bit
        #There are 4 bytes in total. The first byte is the address code, the second is the 
address inverse code, 
        #the third is the control command data of the corresponding button, and the 
fourth is the control command inverse code
        data = [0,0,0,0]
        for i in range(0,32):  # Start receiving 32BITE data
           count = 0
           while GPIO.input(PIN) == 0 and count < 15:  # Wait for the LOW LOW level of 
562.5US to pass and exit the loop if it exceeds 900US
              count += 1
              time.sleep(0.00006)

           count = 0
           while GPIO.input(PIN) == 1 and count < 40:  # waits for logical HIGH level to 
pass and exits the loop if it exceeds 2.4ms
              count += 1
              time.sleep(0.00006)

           # if count>8, that is, the logical time is greater than 0.54+0.562=1.12ms, that 
is, 
           #the period is greater than the logical 0 period, that is equivalent to receiving 
logical 1
           if count > 8:
              data[idx] |= 1<<cnt    #When idx=0 is the first data  data[idx] = data[idx] | 
1<<cnt   00000001 <<1 == 0000 0010
           if cnt == 7:    #With 8 byte
```

```
                cnt = 0     #Displacement qing 0
                idx += 1    #Store the next data
            else:
                cnt += 1   #The shift adds 1
        #Determine whether address code + address inverse code =0xff, control code +
control inverse code = 0xFF
        if data[0]+data[1] == 0xFF and data[2]+data[3] == 0xFF:
            print("Get the key: 0x%02x" %data[2])  #Data [2] is the control code we need
            exec_cmd(data[2])
except KeyboardInterrupt:
    GPIO.cleanup()
```

12. Infrared Remote Control Car (bp9_ir_car.py**)**

```
python
import RPi.GPIO as GPIO
import time

PIN = 15;
SCLK = 8
DIO  = 9
# Display pattern data
matrix_smile = (0x00, 0x00, 0x38, 0x40, 0x40, 0x40, 0x3a, 0x02, 0x02, 0x3a, 0x40,
0x40, 0x40, 0x38, 0x00, 0x00)
matrix_forward = (0x00, 0x00, 0x00, 0x00, 0x12, 0x24, 0x48, 0x90, 0x90, 0x48, 0x24,
0x12, 0x00, 0x00, 0x00, 0x00)
matrix_back = (0x00, 0x00, 0x00, 0x00, 0x48, 0x24, 0x12, 0x09, 0x09, 0x12, 0x24,
0x48, 0x00, 0x00, 0x00, 0x00)
matrix_left = (0x00, 0x00, 0x00, 0x00, 0x18, 0x24, 0x42, 0x99, 0x24, 0x42, 0x81, 0x00,
0x00, 0x00, 0x00, 0x00)
matrix_right = (0x00, 0x00, 0x00, 0x00, 0x00, 0x81, 0x42, 0x24, 0x99, 0x42, 0x24,
0x18, 0x00, 0x00, 0x00, 0x00)

GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN,GPIO.IN,GPIO.PUD_UP)
GPIO.setup(SCLK,GPIO.OUT)
GPIO.setup(DIO,GPIO.OUT)

print("irm test start...")
# Control M2 motor
```

```
L_IN1 = 20
L_IN2 = 21
L_PWM1 = 0
# Control M1 motor
L_IN3 = 22
L_IN4 = 23
L_PWM2 = 1
# Control M3 motor
R_IN1 = 24
R_IN2 = 25
R_PWM1 = 12
# Control M4 motor
R_IN3 = 26
R_IN4 = 27
R_PWM2 = 13

GPIO.setmode(GPIO.BCM)  # use BCM numbers
#set the MOTOR Driver Pin OUTPUT mode
GPIO.setup(L_IN1,GPIO.OUT)
GPIO.setup(L_IN2,GPIO.OUT)
GPIO.setup(L_PWM1,GPIO.OUT)
GPIO.setup(L_IN3,GPIO.OUT)
GPIO.setup(L_IN4,GPIO.OUT)
GPIO.setup(L_PWM2,GPIO.OUT)
GPIO.setup(R_IN1,GPIO.OUT)
GPIO.setup(R_IN2,GPIO.OUT)
GPIO.setup(R_PWM1,GPIO.OUT)
GPIO.setup(R_IN3,GPIO.OUT)
GPIO.setup(R_IN4,GPIO.OUT)
GPIO.setup(R_PWM2,GPIO.OUT)

GPIO.output(L_IN1,GPIO.LOW)
GPIO.output(L_IN2,GPIO.LOW)
GPIO.output(L_IN3,GPIO.LOW)
GPIO.output(L_IN4,GPIO.LOW)
GPIO.output(R_IN1,GPIO.LOW)
GPIO.output(R_IN2,GPIO.LOW)
GPIO.output(R_IN3,GPIO.LOW)
GPIO.output(R_IN4,GPIO.LOW)
```

```
#set pwm frequence to 1000hz
pwm_R1 = GPIO.PWM(R_PWM1,100)
pwm_R2 = GPIO.PWM(R_PWM2,100)
pwm_L1 = GPIO.PWM(L_PWM1,100)
pwm_L2 = GPIO.PWM(L_PWM2,100)

#set inital duty cycle to 0
pwm_R1.start(0)
pwm_L1.start(0)
pwm_R2.start(0)
pwm_L2.start(0)

def nop():
    time.sleep(0.000001)

def nop2():
    time.sleep(0.01)

def start():
    GPIO.output(SCLK,1)
    nop()
    GPIO.output(DIO,1)
    nop()
    GPIO.output(DIO,0)
    nop()
    GPIO.output(SCLK,0)

def send_date(date):
    for i in range(0,8):
        GPIO.output(SCLK,0)
        nop()
        if date & 0x01:
            GPIO.output(DIO,1)
        else:
            GPIO.output(DIO,0)
        nop()
        GPIO.output(SCLK,1)
        nop()
        date >>= 1
    GPIO.output(SCLK,0)
```

```
def end():
   GPIO.output(SCLK,0)
   nop()
   GPIO.output(DIO,0)
   nop()
   GPIO.output(SCLK,1)
   nop()
   GPIO.output(DIO,1)
   nop()

def matrix_display(matrix_value):
   start()
   send_date(0xc0)

   for i in range(0,16):
      send_date(matrix_value[i])

   end()
   start()
   send_date(0x8A)
   end()

def exec_cmd(key_val):
   if(key_val==0x46):
      print("Button up")
      matrix_display(matrix_forward)
      GPIO.output(L_IN1,GPIO.LOW)  #Upper Left forward
      GPIO.output(L_IN2,GPIO.HIGH)
      pwm_L1.ChangeDutyCycle(50)
      GPIO.output(L_IN3,GPIO.HIGH)  #Lower left forward
      GPIO.output(L_IN4,GPIO.LOW)
      pwm_L2.ChangeDutyCycle(50)
      GPIO.output(R_IN1,GPIO.HIGH)  #Upper Right forward
      GPIO.output(R_IN2,GPIO.LOW)
      pwm_R1.ChangeDutyCycle(50)
      GPIO.output(R_IN3,GPIO.LOW)  #Lower Right forward
      GPIO.output(R_IN4,GPIO.HIGH)
      pwm_R2.ChangeDutyCycle(50)
   elif(key_val==0x44):
      print("Button left")
      matrix_display(matrix_left)
      GPIO.output(L_IN1,GPIO.HIGH)
      GPIO.output(L_IN2,GPIO.LOW)
      pwm_L1.ChangeDutyCycle(100)
      GPIO.output(L_IN3,GPIO.LOW)
      GPIO.output(L_IN4,GPIO.HIGH)
      pwm_L2.ChangeDutyCycle(100)
      GPIO.output(R_IN1,GPIO.HIGH)  #Upper Right forward
      GPIO.output(R_IN2,GPIO.LOW)
      pwm_R1.ChangeDutyCycle(100)
      GPIO.output(R_IN3,GPIO.LOW)  #Lower Right forward
      GPIO.output(R_IN4,GPIO.HIGH)
      pwm_R2.ChangeDutyCycle(100)
   elif(key_val==0x40):
      print("Button ok")
      matrix_display(matrix_smile)
      pwm_L1.ChangeDutyCycle(0)
      pwm_L2.ChangeDutyCycle(0)
      pwm_R1.ChangeDutyCycle(0)
      pwm_R2.ChangeDutyCycle(0)
   elif(key_val==0x43):
      print("Button right")
      matrix_display(matrix_right)
      GPIO.output(L_IN1,GPIO.LOW)  #Upper Left forward
      GPIO.output(L_IN2,GPIO.HIGH)
      pwm_L1.ChangeDutyCycle(100)
      GPIO.output(L_IN3,GPIO.HIGH)  #Lower left forward
      GPIO.output(L_IN4,GPIO.LOW)
      pwm_L2.ChangeDutyCycle(100)
      GPIO.output(R_IN1,GPIO.LOW)  #Upper Right forward
      GPIO.output(R_IN2,GPIO.HIGH)
      pwm_R1.ChangeDutyCycle(100)
      GPIO.output(R_IN3,GPIO.HIGH)  #Lower Right forward
      GPIO.output(R_IN4,GPIO.LOW)
      pwm_R2.ChangeDutyCycle(100)
   elif(key_val==0x15):
      print("Button down")
      matrix_display(matrix_back)
      GPIO.output(L_IN1,GPIO.HIGH)
```

```
        GPIO.output(L_IN2,GPIO.LOW)
        pwm_L1.ChangeDutyCycle(50)
        GPIO.output(L_IN3,GPIO.LOW)
        GPIO.output(L_IN4,GPIO.HIGH)
        pwm_L2.ChangeDutyCycle(50)
        GPIO.output(R_IN1,GPIO.LOW)
        GPIO.output(R_IN2,GPIO.HIGH)
        pwm_R1.ChangeDutyCycle(50)
        GPIO.output(R_IN3,GPIO.HIGH)
        GPIO.output(R_IN4,GPIO.LOW)
        pwm_R2.ChangeDutyCycle(50)
    elif(key_val==0x16):
        print("Button 1")
    elif(key_val==0x19):
        print("Button 2")
    elif(key_val==0x0d):
        print("Button 3")
    elif(key_val==0x0c):
        print("Button 4")
    elif(key_val==0x18):
        print("Button 5")
    elif(key_val==0x5e):
        print("Button 6")
    elif(key_val==0x08):
        print("Button 7")
    elif(key_val==0x1c):
        print("Button 8")
    elif(key_val==0x5a):
        print("Button 9")
    elif(key_val==0x42):
        print("Button *")
    elif(key_val==0x52):
        print("Button 0")
    elif(key_val==0x4a):
        print("Button #")

try:
    while True:
        if GPIO.input(PIN) == 0:
            count = 0
            while GPIO.input(PIN) == 0 and count < 200:  # Wait for 9ms LOW level boot
code and exit the loop if it exceeds 1.2ms
            count += 1
            time.sleep(0.00006)

            count = 0
            while GPIO.input(PIN) == 1 and count < 80:   # Wait for a 4.5ms HIGH level boot
code and exit the loop if it exceeds 0.48ms
            count += 1
            time.sleep(0.00006)

            idx = 0  # byte count variable
            cnt = 0  #Variable per byte bit
            #There are 4 bytes in total. The first byte is the address code, the second is the
address inverse code,
            #the third is the control command data of the corresponding button, and the
fourth is the control command inverse code
            data = [0,0,0,0]
            for i in range(0,32):  # Start receiving 32BITE data
            count = 0
            while GPIO.input(PIN) == 0 and count < 15:  # Wait for the LOW LOW level of
562.5US to pass and exit the loop if it exceeds 900US
                count += 1
                time.sleep(0.00006)

            count = 0
            while GPIO.input(PIN) == 1 and count < 40:  # waits for logical HIGH level to
pass and exits the loop if it exceeds 2.4ms
                count += 1
                time.sleep(0.00006)

            # if count>8, that is, the logical time is greater than 0.54+0.562=1.12ms, that
is,
            #the period is greater than the logical 0 period, that is equivalent to receiving
logical 1
            if count > 8:
                data[idx] |= 1<<cnt    #When idx=0 is the first data  data[idx] = data[idx] |
1<<cnt   00000001 <<1 == 0000 0010
            if cnt == 7:    #With 8 byte
                cnt = 0     #Displacement qing 0
                idx += 1    #Store the next data
            else:
                cnt += 1   #The shift adds 1
        #Determine whether address code + address inverse code =0xff, control code + 
control inverse code = 0xFF
        if data[0]+data[1] == 0xFF and data[2]+data[3] == 0xFF:
            print("Get the key: 0x%02x" %data[2])  #Data [2] is the control code we need
            exec_cmd(data[2])
except KeyboardInterrupt:
    GPIO.cleanup()
```

13. Line Tracking Car (bp10_tracking_car.py**)**

```
python
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

# Control M2 motor
L_IN1 = 20
L_IN2 = 21
L_PWM1 = 0
# Control M1 motor
L_IN3 = 22
L_IN4 = 23
L_PWM2 = 1
# Control M3 motor
R_IN1 = 24
R_IN2 = 25
R_PWM1 = 12
# Control M4 motor
R_IN3 = 26
R_IN4 = 27
R_PWM2 = 13

#tracking pin
```

```
trackingPin1 = 17
trackingPin2 = 18
trackingPin3 = 19

GPIO.setmode(GPIO.BCM)  # use BCM numbers

GPIO.setup(trackingPin1,GPIO.IN)  # set trackingPin INPUT mode
GPIO.setup(trackingPin2,GPIO.IN)
GPIO.setup(trackingPin3,GPIO.IN)

#set the MOTOR Driver Pin OUTPUT mode
GPIO.setup(L_IN1,GPIO.OUT)
GPIO.setup(L_IN2,GPIO.OUT)
GPIO.setup(L_PWM1,GPIO.OUT)

GPIO.setup(L_IN3,GPIO.OUT)
GPIO.setup(L_IN4,GPIO.OUT)
GPIO.setup(L_PWM2,GPIO.OUT)

GPIO.setup(R_IN1,GPIO.OUT)
GPIO.setup(R_IN2,GPIO.OUT)
GPIO.setup(R_PWM1,GPIO.OUT)

GPIO.setup(R_IN3,GPIO.OUT)
GPIO.setup(R_IN4,GPIO.OUT)
GPIO.setup(R_PWM2,GPIO.OUT)


GPIO.output(L_IN1,GPIO.LOW)
GPIO.output(L_IN2,GPIO.LOW)
GPIO.output(L_IN3,GPIO.LOW)
GPIO.output(L_IN4,GPIO.LOW)

GPIO.output(R_IN1,GPIO.LOW)
GPIO.output(R_IN2,GPIO.LOW)
GPIO.output(R_IN3,GPIO.LOW)
GPIO.output(R_IN4,GPIO.LOW)


#set pwm frequence to 1000hz
```

```
pwm_R1 = GPIO.PWM(R_PWM1,100)
pwm_R2 = GPIO.PWM(R_PWM2,100)
pwm_L1 = GPIO.PWM(L_PWM1,100)
pwm_L2 = GPIO.PWM(L_PWM2,100)

#set inital duty cycle to 0
pwm_R1.start(0)
pwm_L1.start(0)
pwm_R2.start(0)
pwm_L2.start(0)
# car forward
def car_forward():
    GPIO.output(L_IN1,GPIO.LOW)
    GPIO.output(L_IN2,GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(30)
    GPIO.output(L_IN3,GPIO.HIGH)
    GPIO.output(L_IN4,GPIO.LOW)
    pwm_L2.ChangeDutyCycle(30)
    GPIO.output(R_IN1,GPIO.HIGH)
    GPIO.output(R_IN2,GPIO.LOW)
    pwm_R1.ChangeDutyCycle(30)
    GPIO.output(R_IN3,GPIO.LOW)
    GPIO.output(R_IN4,GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(30)
# car left
def car_left():
    GPIO.output(L_IN1,GPIO.HIGH)
    GPIO.output(L_IN2,GPIO.LOW)
    pwm_L1.ChangeDutyCycle(50)
    GPIO.output(L_IN3,GPIO.LOW)
    GPIO.output(L_IN4,GPIO.HIGH)
    pwm_L2.ChangeDutyCycle(50)
    GPIO.output(R_IN1,GPIO.HIGH)
    GPIO.output(R_IN2,GPIO.LOW)
    pwm_R1.ChangeDutyCycle(50)
    GPIO.output(R_IN3,GPIO.LOW)
    GPIO.output(R_IN4,GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(50)
# car right
def car_right():
    GPIO.output(L_IN1,GPIO.LOW)
    GPIO.output(L_IN2,GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(50)
    GPIO.output(L_IN3,GPIO.HIGH)
    GPIO.output(L_IN4,GPIO.LOW)
    pwm_L2.ChangeDutyCycle(50)
    GPIO.output(R_IN1,GPIO.LOW)
    GPIO.output(R_IN2,GPIO.HIGH)
    pwm_R1.ChangeDutyCycle(50)
    GPIO.output(R_IN3,GPIO.HIGH)
    GPIO.output(R_IN4,GPIO.LOW)
    pwm_R2.ChangeDutyCycle(50)
# car stop
def car_stop():
    pwm_L1.ChangeDutyCycle(0)
    pwm_L2.ChangeDutyCycle(0)
    pwm_R1.ChangeDutyCycle(0)
    pwm_R2.ChangeDutyCycle(0)

while True:
    val1 = GPIO.input(trackingPin1) # read the value
    val2 = GPIO.input(trackingPin2)
    val3 = GPIO.input(trackingPin3)

    if(val2 == 1):
        car_forward()
        print("forward")
    else:
        if((val1 == 1) and (val3 == 0)):
            car_right()
            print("right")
        elif((val1 == 0) and (val3 == 1)):
            car_left()
            print("left")
        else:
            car_stop()
            print("stop")

#stop pwm
pwm_R1.stop()
```

```
pwm_L1.stop()
pwm_R2.stop()
pwm_L2.stop()
time.sleep(1)

GPIO.cleanup()  #release all GPIO
```

14. Ultrasonic Following Car (bp11_follow_car.py**)**

```
python
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)  # use BCM numbers

#define GPIO pin
GPIO_TRIGGER = 14
GPIO_ECHO = 4
#set GPIO mode (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
def distance():
   # 10us is the trigger signal
   GPIO.output(GPIO_TRIGGER, True)
   time.sleep(0.00001)  #10us
   GPIO.output(GPIO_TRIGGER, False)
   start_time = time.time()  # Log the time the program runs to this point
   stop_time = time.time()  # Log the time the program runs to this point
   while GPIO.input(GPIO_ECHO) == 0:   #Indicates that the ultrasonic wave has been
emitted
       start_time = time.time()  #Record launch time
   while GPIO.input(GPIO_ECHO) == 1:   #Indicates that the returned ultrasound has
been received
       stop_time = time.time()   #Record receiving time
   time_elapsed = stop_time - start_time  #Time difference from transmit to receive
   distance = (time_elapsed * 34000) / 2  #Calculate the distance
   return distance   #Return to calculated distance

# Control M2 motor
L_IN1 = 20
```

```
L_IN2 = 21
L_PWM1 = 0
# Control M1 motor
L_IN3 = 22
L_IN4 = 23
L_PWM2 = 1
# Control M3 motor
R_IN1 = 24
R_IN2 = 25
R_PWM1 = 12
# Control M4 motor
R_IN3 = 26
R_IN4 = 27
R_PWM2 = 13

#set the MOTOR Driver Pin OUTPUT mode
GPIO.setup(L_IN1,GPIO.OUT)
GPIO.setup(L_IN2,GPIO.OUT)
GPIO.setup(L_PWM1,GPIO.OUT)
GPIO.setup(L_IN3,GPIO.OUT)
GPIO.setup(L_IN4,GPIO.OUT)
GPIO.setup(L_PWM2,GPIO.OUT)
GPIO.setup(R_IN1,GPIO.OUT)
GPIO.setup(R_IN2,GPIO.OUT)
GPIO.setup(R_PWM1,GPIO.OUT)
GPIO.setup(R_IN3,GPIO.OUT)
GPIO.setup(R_IN4,GPIO.OUT)
GPIO.setup(R_PWM2,GPIO.OUT)

GPIO.output(L_IN1,GPIO.LOW)
GPIO.output(L_IN2,GPIO.LOW)
GPIO.output(L_IN3,GPIO.LOW)
GPIO.output(L_IN4,GPIO.LOW)
GPIO.output(R_IN1,GPIO.LOW)
GPIO.output(R_IN2,GPIO.LOW)
GPIO.output(R_IN3,GPIO.LOW)
GPIO.output(R_IN4,GPIO.LOW)


#set pwm frequence to 1000hz
```

```
pwm_R1 = GPIO.PWM(R_PWM1,100)
pwm_R2 = GPIO.PWM(R_PWM2,100)
pwm_L1 = GPIO.PWM(L_PWM1,100)
pwm_L2 = GPIO.PWM(L_PWM2,100)

#set inital duty cycle to 0
pwm_R1.start(0)
pwm_L1.start(0)
pwm_R2.start(0)
pwm_L2.start(0)
# car forward
def car_forward():
    GPIO.output(L_IN1,GPIO.LOW)
    GPIO.output(L_IN2,GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(30)
    GPIO.output(L_IN3,GPIO.HIGH)
    GPIO.output(L_IN4,GPIO.LOW)
    pwm_L2.ChangeDutyCycle(30)
    GPIO.output(R_IN1,GPIO.HIGH)
    GPIO.output(R_IN2,GPIO.LOW)
    pwm_R1.ChangeDutyCycle(30)
    GPIO.output(R_IN3,GPIO.LOW)
    GPIO.output(R_IN4,GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(30)
# car back
def car_back():
    GPIO.output(L_IN1,GPIO.HIGH)
    GPIO.output(L_IN2,GPIO.LOW)
    pwm_L1.ChangeDutyCycle(30)
    GPIO.output(L_IN3,GPIO.LOW)
    GPIO.output(L_IN4,GPIO.HIGH)
    pwm_L2.ChangeDutyCycle(30)
    GPIO.output(R_IN1,GPIO.LOW)
    GPIO.output(R_IN2,GPIO.HIGH)
    pwm_R1.ChangeDutyCycle(30)
    GPIO.output(R_IN3,GPIO.HIGH)
    GPIO.output(R_IN4,GPIO.LOW)
    pwm_R2.ChangeDutyCycle(30)
# car stop
def car_stop():
    pwm_L1.ChangeDutyCycle(0)
    pwm_L2.ChangeDutyCycle(0)
    pwm_R1.ChangeDutyCycle(0)
    pwm_R2.ChangeDutyCycle(0)

while True:
    dist = distance()
    print("Measured Distance = {:.2f} cm".format(dist))
    #time.sleep(0.02)
    if dist < 10:
        time.sleep(0.15)
        car_back()
    elif (dist>=10 and dist<=14):
        car_stop()
        time.sleep(0.1)
    elif (dist>14 and dist<= 50):
        time.sleep(0.15)
        car_forward()
    else:
        car_stop()

print("stop")
#stop pwm
pwm_R1.stop()
pwm_L1.stop()
pwm_R2.stop()
pwm_L2.stop()
time.sleep(1)

GPIO.cleanup()  #release all GPIO
```

15. Ultrasonic Obstacle Avoidance Robot (bp12_avoid_car.py**)**

```
python
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)  # use BCM numbers

servoPin1 = 5
```

```
GPIO.setup(servoPin1, GPIO.OUT)
def servoPulse(servoPin, myangle):
   pulsewidth = (myangle*11) + 500  # The pulse width
   GPIO.output(servoPin,GPIO.HIGH)
   time.sleep(pulsewidth/1000000.0)
   GPIO.output(servoPin,GPIO.LOW)
   time.sleep(20.0/1000 - pulsewidth/1000000.0) # The cycle of 20 ms

#define GPIO pin
GPIO_TRIGGER = 14
GPIO_ECHO = 4
#set GPIO mode (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
def distance():
   # 10us is the trigger signal
   GPIO.output(GPIO_TRIGGER, True)
   time.sleep(0.00001)  #10us
   GPIO.output(GPIO_TRIGGER, False)
   start_time = time.time()  # Log the time the program runs to this point
   stop_time = time.time()  # Log the time the program runs to this point
   while GPIO.input(GPIO_ECHO) == 0:   #Indicates that the ultrasonic wave has been
emitted
      start_time = time.time()  #Record launch time
   while GPIO.input(GPIO_ECHO) == 1:   #Indicates that the returned ultrasound has
been received
      stop_time = time.time()   #Record receiving time
   time_elapsed = stop_time - start_time  #Time difference from transmit to receive
   distance = (time_elapsed * 34000) / 2  #Calculate the distance
   return distance   #Return to calculated distance

# Control M2 motor
L_IN1 = 20
L_IN2 = 21
L_PWM1 = 0
# Control M1 motor
L_IN3 = 22
L_IN4 = 23
L_PWM2 = 1
# Control M3 motor
```

```
R_IN1 = 24
R_IN2 = 25
R_PWM1 = 12
# Control M4 motor
R_IN3 = 26
R_IN4 = 27
R_PWM2 = 13

#set the MOTOR Driver Pin OUTPUT mode
GPIO.setup(L_IN1,GPIO.OUT)
GPIO.setup(L_IN2,GPIO.OUT)
GPIO.setup(L_PWM1,GPIO.OUT)
GPIO.setup(L_IN3,GPIO.OUT)
GPIO.setup(L_IN4,GPIO.OUT)
GPIO.setup(L_PWM2,GPIO.OUT)
GPIO.setup(R_IN1,GPIO.OUT)
GPIO.setup(R_IN2,GPIO.OUT)
GPIO.setup(R_PWM1,GPIO.OUT)
GPIO.setup(R_IN3,GPIO.OUT)
GPIO.setup(R_IN4,GPIO.OUT)
GPIO.setup(R_PWM2,GPIO.OUT)

GPIO.output(L_IN1,GPIO.LOW)
GPIO.output(L_IN2,GPIO.LOW)
GPIO.output(L_IN3,GPIO.LOW)
GPIO.output(L_IN4,GPIO.LOW)
GPIO.output(R_IN1,GPIO.LOW)
GPIO.output(R_IN2,GPIO.LOW)
GPIO.output(R_IN3,GPIO.LOW)
GPIO.output(R_IN4,GPIO.LOW)


#set pwm frequence to 1000hz
pwm_R1 = GPIO.PWM(R_PWM1,100)
pwm_R2 = GPIO.PWM(R_PWM2,100)
pwm_L1 = GPIO.PWM(L_PWM1,100)
pwm_L2 = GPIO.PWM(L_PWM2,100)

#set inital duty cycle to 0
pwm_R1.start(0)
```

```
pwm_L1.start(0)
pwm_R2.start(0)
pwm_L2.start(0)
# car forward
def car_forward():
    GPIO.output(L_IN1,GPIO.LOW)
    GPIO.output(L_IN2,GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(40)
    GPIO.output(L_IN3,GPIO.HIGH)
    GPIO.output(L_IN4,GPIO.LOW)
    pwm_L2.ChangeDutyCycle(40)
    GPIO.output(R_IN1,GPIO.HIGH)
    GPIO.output(R_IN2,GPIO.LOW)
    pwm_R1 belülChangeDutyCycle(40)
    GPIO.output(R_IN3,GPIO.LOW)
    GPIO.output(R_IN4,GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(40)
# car left
def car_left():
    GPIO.output(L_IN1,GPIO.HIGH)
    GPIO.output(L_IN2,GPIO.LOW)
    pwm_L1.ChangeDutyCycle(80)
    GPIO.output(L_IN3,GPIO.LOW)
    GPIO.output(L_IN4,GPIO.HIGH)
    pwm_L2.ChangeDutyCycle(80)
    GPIO.output(R_IN1,GPIO.HIGH)
    GPIO.output(R_IN2,GPIO.LOW)
    pwm_R1.ChangeDutyCycle(80)
    GPIO.output(R_IN3,GPIO.LOW)
    GPIO.output(R_IN4,GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(80)
# car right
def car_right():
    GPIO.output(L_IN1,GPIO.LOW)
    GPIO.output(L_IN2,GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(80)
    GPIO.output(L_IN3,GPIO.HIGH)
    GPIO.output(L_IN4,GPIO.LOW)
    pwm_L2.ChangeDutyCycle(80)
    GPIO.output(R_IN1,GPIO.LOW)
    GPIO.output(R_IN2,GPIO.HIGH)
    pwm_R1.ChangeDutyCycle(80)
    GPIO.output(R_IN3,GPIO.HIGH)
    GPIO.output(R_IN4,GPIO.LOW)
    pwm_R2.ChangeDutyCycle(80)

# car stop
def car_stop():
    pwm_L1.ChangeDutyCycle(0)
    pwm_L2.ChangeDutyCycle(0)
    pwm_R1.ChangeDutyCycle(0)
    pwm_R2.ChangeDutyCycle(0)

# Loop several times to make sure the steering gear is turned to the specified Angle
# The initialization Angle is 90 degrees
for g in range(0, 50):
    servoPulse(servoPin1, 90)

while True:
    dist = distance()
    print("Measured Distance = {:.2f} cm".format(dist))
    time.sleep(0.01)
    if dist > 15:    # If the obstacle in front is larger than 15cm
        car_forward()
    else:            # If the obstacle in front is less than 15cm
        dist = distance()
        if dist <= 15:  # Make sure the obstacle in front is less than 15cm
            car_stop()
            # Loop several times to make sure the steering gear is turned to the specified
Angle
            for i in range(0, 50):
                servoPulse(servoPin1, 180)
            time.sleep(0.1)
            left_distance = distance()  # Measure the distance to the left
            time.sleep(0.2)
            for j in range(0, 50):
                servoPulse(servoPin1, 0)
            time.sleep(0.1)
            right_distance = distance()  # Measure the distance to the right
            time.sleep(0.2)
            for k in range(0, 50):
                servoPulse(servoPin1, 90)
            time.sleep(0.3)
            # Compare the distance between the left and right sides
            if left_distance > right_distance:
                car_left()
                time.sleep(0.6)
                #The car stops to prevent the sudden positive and
                #negative rotation of the motor from causing the raspberry PI voltage and
                #current too low malfunction
                car_stop()
                time.sleep(0.1)
            else:
                car_right()
                time.sleep(0.6)
                car_stop()
                time.sleep(0.1)

#stop pwm
pwm_R1.stop()
pwm_L1.stop()
pwm_R2.stop()
pwm_L2.stop()
time.sleep(1)

GPIO.cleanup()  #release all GPIO
```

Part 4: OpenCV Implementation for Smart Car

Introduction

In Part 4 of the Keyestudio KS0223 Smart Car project, we used OpenCV to enhance the smart car's navigation with the Raspberry Pi's camera. We implemented line tracking and object detection, processing real-time video to control the car's motors. This section summarizes our experiments, results, and Python codes.

Experimental Procedure and Results

We confirmed the Raspberry Pi camera was functional and installed OpenCV libraries using sudo pip3 install opencv-python and sudo apt install libatlas-base-dev.

For line tracking, we ran cv_line_tracking.py to detect a black line on a white background. The script converted video frames to grayscale, applied a threshold, and used contour detection to find the line's center. The car adjusted its motors to follow the line, turning left or right as needed. On a tracking map, the car accurately followed the line, outperforming sensor-based tracking from Part 3.
For object detection, we used cv_object_detection.py to track a red object in the HSV color space. The script identified the object's centroid and controlled the car to move toward it, stopping if too close or turning if off-center. The car successfully followed the object, demonstrating effective visual tracking.
Both scripts were run from /home/pi/RaspberryPi-Car/basic_project and stopped with Ctrl + C. Calibration of thresholds and speeds was key for reliable performance.

Summary of Python Codes

We used two Python scripts:

- *cv_line_tracking.py: Detected and followed a black line using the camera, controlling motors for line tracking.*
- *cv_object_detection.py: Tracked a red object, directing the car to follow it.*

Our OpenCV experiments enabled the smart car to navigate autonomously using visual data, showcasing the potential of computer vision in robotics.

Code Listings

1. OpenCV Line Tracking (*cv_line_tracking.py****)***

```
python
import cv2
import numpy as np
import RPi.GPIO as GPIO
import time

# Motor pins
L_IN1, L_IN2, L_PWM1 = 20, 21, 0
L_IN3, L_IN4, L_PWM2 = 22, 23, 1
R_IN1, R_IN2, R_PWM1 = 24, 25, 12
R_IN3, R_IN4, R_PWM2 = 26, 27, 13
```

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Setup motors
for pin in [L_IN1, L_IN2, L_PWM1, L_IN3, L_IN4, L_PWM2, R_IN1, R_IN2, R_PWM1,
R_IN3, R_IN4, R_PWM2]:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

# Setup PWM
pwm_R1 = GPIO.PWM(R_PWM1, 100)
pwm_R2 = GPIO.PWM(R_PWM2, 100)
pwm_L1 = GPIO.PWM(L_PWM1, 100)
pwm_L2 = GPIO.PWM(L_PWM2, 100)
pwm_R1.start(0)
pwm_R2.start(0)
pwm_L1.start(0)
pwm_L2.start(0)

def car_forward(speed=30):
    GPIO.output(L_IN1, GPIO.LOW)
    GPIO.output(L_IN2, GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(speed)
    GPIO.output(L_IN3, GPIO.HIGH)
    GPIO.output(L_IN4, GPIO.LOW)
    pwm_L2.ChangeDutyCycle(speed)
    GPIO.output(R_IN1, GPIO.HIGH)
    GPIO.output(R_IN2, GPIO.LOW)
    pwm_R1.ChangeDutyCycle(speed)
    GPIO.output(R_IN3, GPIO.LOW)
    GPIO.output(R_IN4, GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(speed)

def car_left(speed=50):
    GPIO.output(L_IN1, GPIO.HIGH)
    GPIO.output(L_IN2, GPIO.LOW)
    pwm_L1.ChangeDutyCycle(speed)
    GPIO.output(L_IN3, GPIO.LOW)
    GPIO.output(L_IN4, GPIO.HIGH)
    pwm_L2.ChangeDutyCycle(speed)
    GPIO.output(R_IN1, GPIO.HIGH)
    GPIO.output(R_IN2, GPIO.LOW)
    pwm_R1.ChangeDutyCycle(speed)
    GPIO.output(R_IN3, GPIO.LOW)
    GPIO.output(R_IN4, GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(speed)

def car_right(speed=50):
    GPIO.output(L_IN1, GPIO.LOW)
    GPIO.output(L_IN2, GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(speed)
    GPIO.output(L_IN3, GPIO.HIGH)
    GPIO.output(L_IN4, GPIO.LOW)
    pwm_L2.ChangeDutyCycle(speed)
    GPIO.output(R_IN1, GPIO.LOW)
    GPIO.output(R_IN2, GPIO.HIGH)
    pwm_R1.ChangeDutyCycle(speed)
    GPIO.output(R_IN3, GPIO.HIGH)
    GPIO.output(R_IN4, GPIO.LOW)
    pwm_R2.ChangeDutyCycle(speed)

def car_stop():
    pwm_L1.ChangeDutyCycle(0)
    pwm_L2.ChangeDutyCycle(0)
    pwm_R1.ChangeDutyCycle(0)
    pwm_R2.ChangeDutyCycle(0)

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)

try:
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        _, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY_INV)
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
        if contours:
            largest_contour = max(contours, key=cv2.contourArea)
            M = cv2.moments(largest_contour)
            if M["m00"] != 0:
                cx = int(M["m10"] / M["m00"])
                frame_center = frame.shape[1] // 2
                if cx < frame_center - 50:
                    car_left()
                    print("Turning left")
                elif cx > frame_center + 50:
                    car_right()
                    print("Turning right")
                else:
                    car_forward()
                    print("Moving forward")
            else:
                car_stop()
                print("No line detected")
        else:
            car_stop()
            print("No line detected")
        time.sleep(0.1)

except KeyboardInterrupt:
    car_stop()
    pwm_R1.stop()
    pwm_L1.stop()
    pwm_R2.stop()
    pwm_L2.stop()
    cap.release()
    GPIO.cleanup()
```

2. OpenCV Object Detection (*cv_object_detection.py****)***

```
python
import cv2
import numpy as np
import RPi.GPIO as GPIO
import time
```

```
L_IN1, L_IN2, L_PWM1 = 20, 21, 0
L_IN3, L_IN4, L_PWM2 = 22, 23, 1
R_IN1, R_IN2, R_PWM1 = 24, 25, 12
R_IN3, R_IN4, R_PWM2 = 26, 27, 13

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

for pin in [L_IN1, L_IN2, L_PWM1, L_IN3, L_IN4, L_PWM2, R_IN1, R_IN2, R_PWM1,
R_IN3, R_IN4, R_PWM2]:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

pwm_R1 = GPIO.PWM(R_PWM1, 100)
pwm_R2 = GPIO.PWM(R_PWM2, 100)
pwm_L1 = GPIO.PWM(L_PWM1, 100)
pwm_L2 = GPIO.PWM(L_PWM2, 100)
pwm_R1.start(0)
pwm_R2.start(0)
pwm_L1.start(0)
pwm_L2.start(0)

def car_forward(speed=30):
    GPIO.output(L_IN1, GPIO.LOW)
    GPIO.output(L_IN2, GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(speed)
    GPIO.output(L_IN3, GPIO.HIGH)
    GPIO.output(L_IN4, GPIO.LOW)
    pwm_L2.ChangeDutyCycle(speed)
    GPIO.output(R_IN1, GPIO.HIGH)
    GPIO.output(R_IN2, GPIO.LOW)
    pwm_R1.ChangeDutyCycle(speed)
    GPIO.output(R_IN3, GPIO.LOW)
    GPIO.output(R_IN4, GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(speed)

def car_left(speed=50):
    GPIO.output(L_IN1, GPIO.HIGH)
    GPIO.output(L_IN2, GPIO.LOW)
    pwm_L1.ChangeDutyCycle(speed)
    GPIO.output(L_IN3, GPIO.LOW)
    GPIO.output(L_IN4, GPIO.HIGH)
    pwm_L2.ChangeDutyCycle(speed)
    GPIO.output(R_IN1, GPIO.HIGH)
    GPIO.output(R_IN2, GPIO.LOW)
    pwm_R1.ChangeDutyCycle(speed)
    GPIO.output(R_IN3, GPIO.LOW)
    GPIO.output(R_IN4, GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(speed)

def car_right(speed=50):
    GPIO.output(L_IN1, GPIO.LOW)
    GPIO.output(L_IN2, GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(speed)
    GPIO.output(L_IN3, GPIO.HIGH)
    GPIO.output(L_IN4, GPIO.LOW)
    pwm_L2.ChangeDutyCycle(speed)
    GPIO.output(R_IN1, GPIO.LOW)
    GPIO.output(R_IN2, GPIO.HIGH)
    pwm_R1.ChangeDutyCycle(speed)
    GPIO.output(R_IN3, GPIO.HIGH)
    GPIO.output(R_IN4, GPIO.LOW)
    pwm_R2.ChangeDutyCycle(speed)

def car_stop():
    pwm_L1.ChangeDutyCycle(0)
    pwm_L2.ChangeDutyCycle(0)
    pwm_R1.ChangeDutyCycle(0)
    pwm_R2.ChangeDutyCycle(0)

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)

lower_red = np.array([0, 120, 70])
upper_red = np.array([10, 255, 255])

try:
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv, lower_red, upper_red)
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        if contours:
            largest_contour = max(contours, key=cv2.contourArea)
            M = cv2.moments(largest_contour)
            if M["m00"] != 0:
                cx = int(M["m10"] / M["m00"])
                area = cv2.contourArea(largest_contour)
                frame_center = frame.shape[1] // 2
                if area > 1000:
                    car_stop()
                    print("Object too close")
                elif cx < frame_center - 50:
                    car_left()
                    print("Turning left")
                elif cx > frame_center + 50:
                    car_right()
                    print("Turning right")
                else:
                    car_forward()
                    print("Moving forward")
            else:
                car_stop()
                print("No object detected")
        else:
            car_stop()
            print("No object detected")
        time.sleep(0.1)

except KeyboardInterrupt:
    car_stop()
    pwm_R1.stop()
    pwm_L1.stop()
    pwm_R2.stop()
    pwm_L2.stop()
    cap.release()
    GPIO.cleanup()
```

Part 5: App-Controlled Camera for Smart Car

Introduction

In Part 5, we used OpenCV and the “Keyes RPi Robot” app to stream video and control the Keyestudio KS0223 Smart Car’s movement and camera via Raspberry Pi. TCP handled app commands, and UDP streamed video. This section summarizes our experiments, results, and codes.

Experimental Procedure and Results

We disabled boot-up programs in /etc/rc.local using sudo nano /etc/rc.local, commenting scripts with #. OpenCV was pre-installed via Raspberry Pi imager.

We tested TCP with socket_test.py from /home/pi/RaspberryPi-Car. The terminal/OLED showed the Raspberry Pi's IP (e.g., 192.168.1.126). Using the app on a hotspot-connected phone, we entered IP/port (5051). The terminal displayed "accept the client!" and OLED showed "Connect." App buttons triggered outputs (e.g., "go").

With MainControl.py, we controlled car movement (forward, back, left, right, stop) and camera pan-tilt (up, down, left, right) via app, with accurate responses.
We streamed video using FramesSend_test.py via UDP (port 5051, IP 10.0.0.222). The terminal showed frame lengths, and the app displayed real-time video.

Using FramesSend.py, we automated phone IP detection with Scapy. The app streamed video after entering Raspberry Pi's IP (via ip a) and port.

We added MainControl.py and FramesSend.py to /etc/rc.local for boot-up. After sudo reboot, the app controlled the car and streamed video seamlessly.

Summary of Python Codes

We used four scripts:

- *socket_test.py: TCP for app commands.*
- *MainControl.py: Car/camera control.*
- *FramesSend_test.py: UDP video streaming.*
- *FramesSend.py: UDP streaming with IP detection.*

Conclusion

We integrated OpenCV streaming and app control, enabling remote navigation and monitoring, enhancing the car's capabilities.

Figures

[Insert images, e.g., app, OLED, video feed.]

Code Listings

1. TCP Communication (*socket_test.py****)***

python

```
import socket
import time
from OledModule.OLED import OLED

def getLocalIp():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(('8.8.8.8', 80))
    ip = s.getsockname()[0]
    s.close()
    return ip

def cameraAction(command):
    print({"CamUp": "camUp", "CamDown": "camDown", "CamLeft": "camLeft",
"CamRight": "camRight", "CamStop": "camStop"}.get(command, ""))

def motorAction(command):
    print({"DirForward": "go", "DirBack": "back", "DirLeft": "left", "DirRight": "right",
"DirStop": "stop"}.get(command, ""))

def main():
    host = getLocalIp()
    print('ip:' + host)
    port = 5051
    tcpServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcpServer.bind((host, port))
    tcpServer.setblocking(0)
    tcpServer.listen(5)
    oled = OLED()
    oled.setup()
    oled.editArea1(host)
    oled.editArea3('State:')
    oled.editArea4(' Disconnect')
    while True:
        try:
            time.sleep(0.001)
            client, addr = tcpServer.accept()
            print('client connected!')
            oled.editArea4(' Connect')
            client.setblocking(0)
            while True:
                time.sleep(0.001)
                try:
                    data = client.recv(1024).decode()
                    if not data:
                        print('client closed')
                        oled.editArea4(' Disconnect')
                        break
                    motorAction(data)
                    cameraAction(data)
                except socket.error:
                    continue
                except KeyboardInterrupt:
                    raise
        except socket.error:
            pass
        except KeyboardInterrupt:
            tcpServer.close()
            oled.clear()
            port = 0
        except Exception:
            tcpServer.close()
            oled.clear()

main()
```

2. App-Controlled Car and Camera (**MainControl.py**)

python

```
import socket
import time
```

```
from OledModule.OLED import OLED
import RPi.GPIO as GPIO
GPIO.setwarnings(False)

servoPin1, servoPin2, servoPin3 = 5, 6, 7
angle1, angle2 = 90, 90
GPIO.setup(servoPin1, GPIO.OUT)
GPIO.setup(servoPin2, GPIO.OUT)
GPIO.setup(servoPin3, GPIO.OUT)

def servoPulse(servoPin, myangle):
   pulsewidth = (myangle * 11) + 500
   GPIO.output(servoPin, GPIO.HIGH)
   time.sleep(pulsewidth / 1000000.0)
   GPIO.output(servoPin, GPIO.LOW)
   time.sleep(20.0 / 1000 - pulsewidth / 1000000.0)

L_IN1, L_IN2, L_PWM1 = 20, 21, 0
L_IN3, L_IN4, L_PWM2 = 22, 23, 1
R_IN1, R_IN2, R_PWM1 = 24, 25, 12
R_IN3, R_IN4, R_PWM2 = 26, 27, 13
GPIO.setmode(GPIO.BCM)
for pin in [L_IN1, L_IN2, L_PWM1, L_IN3, L_IN4, L_PWM2, R_IN1, R_IN2, R_PWM1,
R_IN3, R_IN4, R_PWM2]:
   GPIO.setup(pin, GPIO.OUT)
   GPIO.output(pin, GPIO.LOW)

pwm_R1 = GPIO.PWM(R_PWM1, 100)
pwm_R2 = GPIO.PWM(R_PWM2, 100)
pwm_L1 = GPIO.PWM(L_PWM1, 100)
pwm_L2 = GPIO.PWM(L_PWM2, 100)
pwm_R1.start(0)
pwm_L1.start(0)
pwm_R2.start(0)
pwm_L2.start(0)

def ahead():
   GPIO.output(L_IN1, GPIO.LOW)
   GPIO.output(L_IN2, GPIO.HIGH)
   pwm_L1.ChangeDutyCycle(80)
```

```
    GPIO.output(L_IN3, GPIO.HIGH)
    GPIO.output(L_IN4, GPIO.LOW)
    pwm_L2.ChangeDutyCycle(80)
    GPIO.output(R_IN1, GPIO.HIGH)
    GPIO.output(R_IN2, GPIO.LOW)
    pwm_R1.ChangeDutyCycle(80)
    GPIO.output(R_IN3, GPIO.LOW)
    GPIO.output(R_IN4, GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(80)

def left():
    GPIO.output(L_IN1, GPIO.HIGH)
    GPIO.output(L_IN2, GPIO.LOW)
    pwm_L1.ChangeDutyCycle(80)
    GPIO.output(L_IN3, GPIO.LOW)
    GPIO.output(L_IN4, GPIO.HIGH)
    pwm_L2.ChangeDutyCycle(80)
    GPIO.output(R_IN1, GPIO.HIGH)
    GPIO.output(R_IN2, GPIO.LOW)
    pwm_R1.ChangeDutyCycle(80)
    GPIO.output(R_IN3, GPIO.LOW)
    GPIO.output(R_IN4, GPIO.HIGH)
    pwm_R2.ChangeDutyCycle(80)

def right():
    GPIO.output(L_IN1, GPIO.LOW)
    GPIO.output(L_IN2, GPIO.HIGH)
    pwm_L1.ChangeDutyCycle(80)
    GPIO.output(L_IN3, GPIO.HIGH)
    GPIO.output(L_IN4, GPIO.LOW)
    pwm_L2.ChangeDutyCycle(80)
    GPIO.output(R_IN1, GPIO.LOW)
    GPIO.output(R_IN2, GPIO.HIGH)
    pwm_R1.ChangeDutyCycle(80)
    GPIO.output(R_IN3, GPIO.HIGH)
    GPIO.output(R_IN4, GPIO.LOW)
    pwm_R2.ChangeDutyCycle(80)

def rear():
    GPIO.output(L_IN1, GPIO.HIGH)
    GPIO.output(L_IN2, GPIO.LOW)
    pwm_L1.ChangeDutyCycle(80)
    GPIO.output(L_IN3, GPIO.LOW)
    GPIO.output(L_IN4, GPIO.HIGH)
    pwm_L2.ChangeDutyCycle(80)
    GPIO.output(R_IN1, GPIO.LOW)
    GPIO.output(R_IN2, GPIO.HIGH)
    pwm_R1.ChangeDutyCycle(80)
    GPIO.output(R_IN3, GPIO.HIGH)
    GPIO.output(R_IN4, GPIO.LOW)
    pwm_R2.ChangeDutyCycle(80)

def stop():
    pwm_L1.ChangeDutyCycle(0)
    pwm_L2.ChangeDutyCycle(0)
    pwm_R1.ChangeDutyCycle(0)
    pwm_R2.ChangeDutyCycle(0)

def getLocalIp():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.connect(('8.8.8.8', 80))
    ip = s.getsockname()[0]
    s.close()
    return ip

def cameraAction(command):
    global angle1, angle2
    if command == 'CamUp':
        angle1 = max(0, angle1 - 1)
        servoPulse(servoPin2, angle1)
    elif command == 'CamDown':
        angle1 = min(180, angle1 + 1)
        servoPulse(servoPin2, angle1)
    elif command == 'CamLeft':
        angle2 = min(180, angle2 + 1)
        servoPulse(servoPin3, angle2)
    elif command == 'CamRight':
        angle2 = max(0, angle2 - 1)
        servoPulse(servoPin3, angle2)
```

```
def motorAction(command):
    actions = {'DirForward': ahead, 'DirBack': rear, 'DirLeft': left, 'DirRight': right, 'DirStop':
stop}
    if command in actions:
        print(command.replace('Dir', '').lower())
        actions[command]()

def setCameraAction(command):
    return command if command in ['CamUp', 'CamDown', 'CamLeft', 'CamRight'] else
'CamStop'

def main():
    oled = OLED()
    oled.setup()
    oled.editArea1('keyestudio')
    oled.editArea3('State:')
    oled.editArea4(' Disconnect')
    time.sleep(2)
    host = getLocalIp()
    print('ip:' + host)
    port = 5051
    tcpServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcpServer.bind((host, port))
    tcpServer.setblocking(0)
    tcpServer.listen(5)
    global cameraActionState
    cameraActionState = 'CamStop'
    oled = OLED()
    oled.setup()
    oled.editArea1(host)
    oled.editArea3('State:')
    oled.editArea4(' Disconnect')
    while True:
        try:
            time.sleep(0.001)
            client, addr = tcpServer.accept()
            print('client connected!')
            oled.editArea4(' Connect')
            client.setblocking(0)
            while True:
```

```
            time.sleep(0.001)
            cameraAction(cameraActionState)
            try:
                data = client.recv(1024).decode()
                if not data:
                    print('client closed')
                    oled.editArea4(' Disconnect')
                    break
                motorAction(data)
                cameraActionState = setCameraAction(data)
            except socket.error:
                continue
            except KeyboardInterrupt:
                raise
        except socket.error:
            pass
        except KeyboardInterrupt:
            tcpServer.close()
            oled.clear()
            print("close")
            port = 0
        except Exception:
            tcpServer.close()
            oled.clear()

main()
```

3. UDP Video Streaming (*FramesSend_test.py****)***

python

```
import cv2
import socket
import time

HOST = '10.0.0.222'
PORT = 5051
WIDTH = 320
HEIGHT = 240

server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
server.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
server.connect((HOST, PORT))
print('sending frames...')
capture = cv2.VideoCapture(0)
capture.set(cv2.CAP_PROP_FRAME_WIDTH, WIDTH)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, HEIGHT)
try:
    while True:
        time.sleep(0.01)
        success, frame = capture.read()
        if success and frame is not None:
            _, imgencode = cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY,
95])
            server.sendall(imgencode)
except Exception as e:
    print(e)
    capture.release()
    server.close()
```

4. UDP Video Streaming with IP Detection (*FramesSend.py****)***

python

```
#!/usr/bin/python
import cv2
import socket
import time
from scapy.all import *

IPData = [0, 1]
def Echo(packet):
    global IPData
    IPData[0] = packet['IP'].src
    print("IP:", IPData[0])

def Stop(packet):
    return True

while (IPData[0] == 0) or (IPData[0] == '10.0.0.1'):
    print("wait..")
    sniff(iface="wlan0", filter="icmp[icmptype] = icmp-echo", count=0, prn=Echo,
stop_filter=Stop)

HOST = IPData[0]
PORT = 5051
WIDTH = 320
HEIGHT = 240

server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
server.connect((HOST, PORT))
print('sending frames...')
capture = cv2.VideoCapture(0)
capture.set(cv2.CAP_PROP_FRAME_WIDTH, WIDTH)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, HEIGHT)
try:
    while True:
        time.sleep(0.01)
        success, frame = capture.read()
        if success and frame is not None:
            _, imgencode = cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY,
95])
            server.sendall(imgencode)
except Exception as e:
    print(e)
    capture.release()
    server.close()
```

Conclusion

This project successfully demonstrated the integration of hardware and software in building an intelligent robot car capable of real-time face detection and autonomous control using the Raspberry Pi platform. By employing Python, OpenCV, and GPIO-based motor interfacing, the system was able to identify human faces through the camera module and respond with motion commands. Through the assembly and configuration of various components—including sensors, motor drivers, and display modules—we validated the versatility and performance of the Raspberry Pi in embedded AI applications. Additionally, by testing individual hardware elements and combining them into advanced functions like line tracking, ultrasonic following, and obstacle avoidance, the project highlighted the effectiveness of modular coding and system integration in robotic systems. Overall, this work reinforces the Raspberry Pi's potential as a cost-effective and powerful platform for learning and developing smart automation and computer vision projects.