RASPBERRY PI-BASED REMOTE CONTROLLED ROVER WITH OBSTACLE AVOIDANCE AND CAMERA CAPTURE

PROJECT REPORT

SUBMITTED BY

KENNETH ROGER NELSON – U2112023
ALLEN ANDREWS – U2112009
AAKASH DANYOH A – U2112006

UNDER THE GUIDANCE OF

MR. SUNIL ANTONY
(ASSISTANT PROFESSOR)

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
OF THE AWARD OF THE DEGREE OF

BACHELOR OF SCIENCE IN PHYSICS

DEPARTMENT OF PHYSICS
FATIMA MATA NATIONAL COLLEGE (AUTONOMOUS)
KOLLAM - 691001

DEPARTMENT OF PHYSICS

FATIMA MATA NATIONAL COLLEGE (AUTONOMOUS)
KOLLAM - 691001

CERTIFICATE

This is to certify that the project report entitled **RASPBERRY PI-BASED REMOTE CONTROLLED ROVER WITH OBSTACLE AVOIDANCE AND CAMERA CAPTURE** is a bonafide record of the work carried out by **KENNETH ROGER NELSON, ALLEN ANDREWS, AAKASH DANYOH A** of the Department of Physics, Fatima Mata National College, Kollam, under the supervision and guidance of Dr. Bhagya Uthaman, Assistant Professor, Department of Physics, Fatima Mata National College, Kollam, in partial fulfillment of the requirements for the award of degree of Bachelor of Science in Physics, during the academic year 2020-2023.

Project Guide
SUNIL ANTONY
Assistant Professor
Department of Physics
Fatima Mata National College
Kollam

Head of the Department
Department of Physics
Fatima Mata National College
Kollam

Examiner 1.

Date :

Examiner 2.

Place:

UNDERTAKING

I, hereby undertake that the project work entitled**, RASPBERRY PI-BASED REMOTE CONTROLLED ROVER WITH OBSTACLE AVOIDANCE AND CAMERA CAPTURE** is carried out by us independently under the valuable guidance of **MR .SUNIL ANTONY**, Assistant Professor, Department of Physics, FATIMA MATA NATIONAL COLLEGE, KOLLAM, in partial fulfillment of the requirements for the award of degree of Bachelor of Science in Physics, during the academic year 2021-2024.

KENNETH ROGER NELSON
ALLEN ANDREWS
AAKASH DANYOH A

ACKNOWLEDGEMENT

For any accomplishment or achievement, the prime requisite is the blessing of the "Almighty" with which only was able to complete this work successfully. I bow to the lord with a grateful heart and prayerful mind.

With great pleasure express my most sincere thanks and gratitude to our esteemed teachers and friends whose valuable guidance and encouragement helped us to complete the project successfully.

It is with pleasure that I express our most sincere thanks and gratitude to **MR. SUNIL ANTONY**, Assistant Professor, Department of Physics, FATIMA MATA NATIONAL COLLEGE, KOLLAM, for her inspiring and valuable guidance and keen interest throughout the progress of work.

KENNETH ROGER NELSON
ALLEN ANDREWS
AAKASH DANYOH A

RASPBERRY PI-BASED REMOTE-CONTROLLED ROVER WITH OBSTACLE AVOIDANCE AND CAMERA CAPTURE

ABSTRACT

This undertaking introduces the creation and execution of a remote-controlled rover powered by Raspberry Pi, featuring obstacle avoidance and camera capture functionalities. The rover utilizes a Raspberry Pi single-board computer as its primary control unit, integrating a motor control system for accurate movement. The obstacle avoidance capability is accomplished by incorporating ultrasonic sensors, allowing the rover to detect and maneuver around obstacles in its trajectory.

The capability to control the rover's movement and camera functions remotely is made possible through a user-friendly interface, enabling users to operate it using a computer or mobile device. The addition of a camera capture feature enhances the rover's usefulness by enabling real-time image capture, allowing users to remotely explore and examine environments.

The project involves incorporating Python programming to establish communication between the Raspberry Pi and the remote-control interface. Moreover, OpenCV is utilized for image processing and camera management. The obstacle avoidance algorithm ensures intelligent decision-making, allowing the rover to autonomously navigate intricate surroundings.

This highly adaptable and budget-friendly Raspberry Pi-powered rover offers a practical and efficient solution for remote exploration and surveillance. It is suitable for a wide range of applications, including educational projects, home automation, security, and research. By incorporating obstacle avoidance and camera capture capabilities, the rover becomes even more versatile and valuable in different scenarios. Whether you're a hobbyist or a professional, this rover is an excellent tool to have.

INDEX

FIGURE INDEX

INTRODUCTION

Raspberry Pi-Based Remote-Controlled Rover with Obstacle Avoidance and Camera Capture

In the realm of robotics and embedded systems, the integration of single-board computers has propelled the development of sophisticated projects that leverage the power of computing for enhanced functionality and real-world applications. This project introduces a novel endeavor in the form of a Raspberry Pi-based remote-controlled car, designed to exhibit manual control via keyboard input, autonomous obstacle avoidance using ultrasonic sensors, and video capture through the integration of a camera module.

The primary objective of this project is to create a versatile remote-controlled car that can be operated manually using keyboard controls and autonomously navigate through obstacles employing an ultrasonic sensor. Additionally, the integration of a camera module aims to provide a visual perspective, enabling video capture for monitoring and analysis.

CHAPTER -1

MICROCONTROLLERS

Microcontrollers are compact, integrated circuits that have a processor core, memory, and programmable input/output peripherals. They are designed to carry out specific tasks or functions within embedded systems, making them a essential component in various electronic devices. Microcontrollers play a crucial role in controlling and automating devices such as appliances, industrial machinery, automotive systems, and more.

1.1 KEY CHARACTERISTICS OF MICROCONTROLLERS

- **Central Processing Unit (CPU):** The heart of the microcontroller, responsible for executing instructions.
- **Memory:** Stores program code (Flash or ROM) and data (RAM).
- **Peripherals:** Include input and output components, timers, communication interfaces (UART, SPI, I2C), and analog-to-digital converters.
- **Clock Speed:** Determines how fast the microcontroller processes instructions.
- **Power Consumption:** Critical for battery-operated devices.

1.2 LEADING MICROCONTROLLERS

1.2.1 Arduino:

- Arduino is a popular open-source electronics platform based on various microcontrollers, such as the Atmel AVR and ARM-based processors. Arduino boards are commonly used for prototyping and developing projects due to their simplicity and a vast community of users.
- Example: Arduino Uno

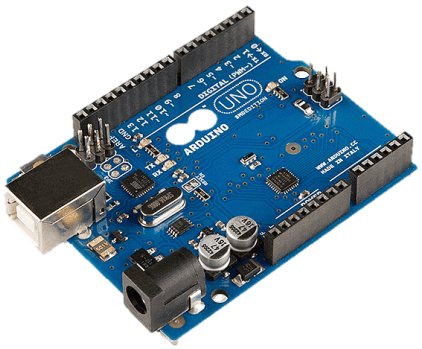

Figure 1 Arduino

1.2.2 Raspberry Pi:

- While Raspberry Pi is often associated with single-board computers, the Raspberry Pi Pico is a microcontroller-based board. It is known for its affordability, ease of use, and integration capabilities.
- Example: Raspberry Pi Pico

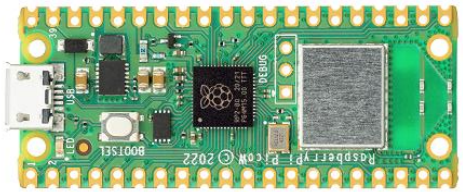

Figure 2 Raspberry pi pico

1.2.3 PIC (Peripheral Interface Controller):

- Developed by Microchip Technology, PIC microcontrollers are common in a variety of applications. They are recognized for their low power consumption, reliability, and a broad range of available models to suit different requirements.
- Example: PIC16F84A

Figure 3 PIC

1.2.4 AVR (Alf and Vegard's RISC processor):

- Developed by Atmel (now a part of Microchip Technology), AVR microcontrollers are known for their efficiency and a rich set of features. They have areas of strength for an in the DIY electronics and hobbyist communities.
- Example: ATmega328P (Used in Arduino Uno)

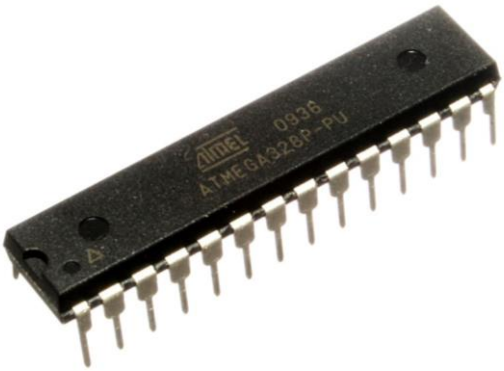

Figure 4 AVR

1.2.5 ARM Cortex-M Series:

- The ARM Cortex-M series is a family of microcontroller cores designed for embedded systems. It offers a large number of models, giving scalability, performance, and energy efficiency.
- Example: STM32 (STMicroelectronics)

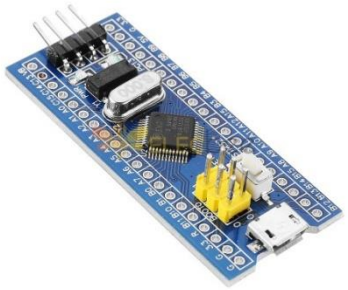

Figure 5 ARM Cortex-M Series

1.2.6 ESP8266 and ESP32:

- Developed by Espressif Systems, these microcontrollers gained a great fame for their built-in Wi-Fi capabilities. They are widely used in IoT (Internet of Things) projects.
- Example: ESP8266, ESP32

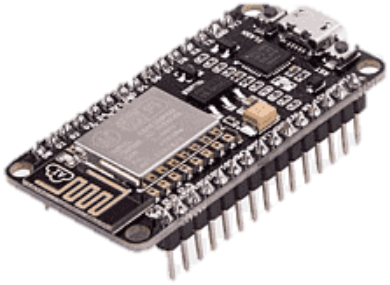

Figure 6 ESP8266

1.2.7 8051 Microcontroller:

- The 8051 microcontroller is a classic model that has been broadly used in various applications since its introduction. It offers a robust architecture and has a big legacy in the industry.
- Example: AT89C51 (Atmel)

Figure 7 8051 Microcontroller

1.3 CHOOSING A MICROCONTROLLER:

The choice of a microcontroller fairly relies upon project requirements, like processing power, memory, I/O capabilities, and cost. Consideration of factors like community support, simplicity of programming, and available development tools also plays a huge part in selecting the right microcontroller for a particular application.

1.4 RASPBERRY PI

Raspberry Pi is a revolutionary single-board computer (SBC) that has transformed the landscape of DIY electronics, computing education, and embedded systems development. Since its introduction in 2012, the Raspberry Pi has gained widespread popularity for its compact size, affordability, and versatility. In this detailed description, we will explore the key aspects of Raspberry Pi, from its hardware components to its applications and impact on various domains.

1.4.1 HARDWARE OVERVIEW

The core of the Raspberry Pi's hardware revolves around a Broadcom system-on-a-chip (SoC) that integrates a processor, GPU, and memory. Essential elements are:

Processor (CPU):

Raspberry Pi models are equipped with different processors. The older Raspberry Pi 1, for example, had an ARM1176JZF-S CPU, whereas the newer Raspberry Pi 4 uses a quad-core ARM Cortex-A72 CPU. The processor's clock speed varies across models, ranging from 700 MHz to 1.5 GHz.

Graphics Processing Unit (GPU):

The Raspberry Pi is equipped with a Video Core VI GPU, which allows for efficient graphics processing. This GPU can support HD video playback and handling 3D graphics, making it well-suited for multimedia applications.

Memory (RAM):

The various Raspberry Pi models come with different RAM options, ranging from 256 MB in the earlier versions to 8 GB in the Raspberry Pi 4. Having an adequate amount of RAM ensures seamless multitasking and enables the use of a wide range of applications.

Input/Output(I/O) Ports:

Raspberry Pi boards come equipped with a variety of I/O ports such as USB ports, HDMI, audio jacks, Ethernet, GPIO (General-Purpose Input/Output) pins, and camera/display connectors. These ports allow for easy connection to external devices, empowering users to explore a wide range of project possibilities.

MicroSD Slot:
The primary storage medium for Raspberry Pi is a microSD card, which holds the operating system (OS) and user data. The use of microSD cards makes the system flexible and easily upgradeable.

1.4.2 OPERATING SYSTEM AND SOFTWARE

Raspberry Pi offers support for a range of operating systems, with the Raspberry Pi OS (formerly known as Raspbian) being the most widely used. This Debian-based Linux distribution is specifically designed for the Raspberry Pi. Additionally, there are other compatible operating systems available such as Ubuntu, Arch Linux, and specialized systems like RetroPie, which is geared towards gaming.

Raspberry Pi OS:
Raspberry Pi OS, created by the Raspberry Pi Foundation, serves as the primary and recommended operating system. It offers a user-friendly interface suitable for both novices and skilled users. The desktop environment is built upon LXDE (Lightweight X11 Desktop Environment), which optimizes performance on lower-end hardware.

Programming Languages and IDEs:
Raspberry Pi is a flexible platform that caters to developers as it offers support for various programming languages. Among these languages, Python stands out for its ease of use and clear syntax. Moreover, Raspberry Pi comes with pre-installed IDEs like Thonny and the robust text editor nano, which greatly aid in coding and script creation.

Application Software:
The Raspberry Pi can operate a wide range of applications, spanning from running web servers with Apache or Nginx to functioning as media centers using Kodi or Plex. Its functionalities also extend to IoT projects, home automation, and educational software.

1.4.3 CONNECTIVITY AND NETWORKING

Raspberry Pi comes with built-in connectivity options, making it suitable for a variety of networking applications.

Ethernet and Wi-Fi:

Most Raspberry Pi models feature an Ethernet port for wired network connections. Additionally, onboard Wi-Fi (in later models) enables wireless networking, enhancing flexibility in project deployment.

Bluetooth:

Bluetooth support allows for easy integration with Bluetooth-enabled devices, such as keyboards, mice, and other peripherals.

USB Ports:

Raspberry Pi boards come equipped with USB ports, facilitating the connection of external devices such as USB drives, keyboards, and webcams.

1.4.4 GPIO PINS AND HARDWARE INTERFACING

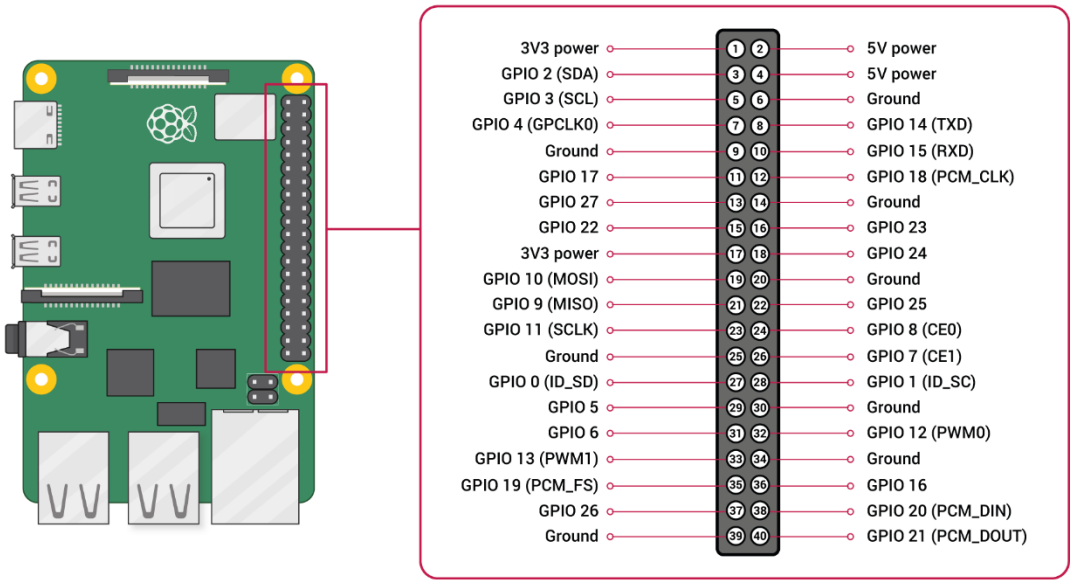

Figure 8 GPIO Pins

A defining feature of Raspberry Pi is its GPIO pins, which enable hardware

interfacing and the connection of external components.

GPIO Pins:
General-Purpose Input/Output pins allow users to interact with the physical world by connecting sensors, LEDs, motors, and other electronic components directly to the Raspberry Pi.

HATs (Hardware Attached on Top):
Raspberry Pi supports HATs, which are add-on boards that extend its capabilities. HATs can include sensors, displays, and other specialized hardware for various applications.

1.4.5 IMPACT ON EDUCATION

The introduction of Raspberry Pi has had a significant impact on computer science education and has greatly encouraged interactive learning.

Affordability:
Due to its low cost, Raspberry Pi has become an affordable option for educational institutions with limited budgets, allowing for its widespread adoption in schools and colleges.

Learning Programming and Electronics:
Raspberry Pi serves as an exceptional platform for teaching programming and electronics. Its GPIO pins offer a hands-on approach for students to explore the world of physical computing.

Learning through Coding and Projects:
By engaging in project-based learning, students have the opportunity to create practical applications and gain valuable experience in real-world programming and problem-solving.

1.4.6 APPLICATIONS

The versatility of Raspberry Pi has resulted in its adoption across various fields and industries.

Home Automation:

Raspberry Pi can be utilized in the development of intelligent home systems, enabling the control of lighting, thermostats, and security cameras.

Media Centers:

Leveraging its GPU capabilities, Raspberry Pi is well-suited for media center applications. By utilizing Kodi and Plex, a Raspberry Pi can be transformed into a robust media player.

IoT Projects:

Thanks to its connectivity options and GPIO pins, Raspberry Pi is an ideal choice for IoT projects. It facilitates the creation of smart devices and sensors.

Robotics:

Raspberry Pi is widely used by robotics enthusiasts for robot control, sensor integration, and the implementation of machine learning applications.

The Raspberry Pi has revolutionized the field of computing and electronics. Its affordability, versatility, and the strong community backing it has enabled people from all walks of life, including students, professionals, and individuals, to delve into uncharted territories and bring forth new innovations. Whether it is used for educational endeavours, do-it-yourself projects, or as a platform for commercial products, the Raspberry Pi has made a significant impact on technology and learning, and its potential for the future is full of promise.

CHAPTER 2

HARDWARE INSIGHT

2.1 RASPBERRY PI ZERO 2W

While microcontrollers like Arduino, PIC, AVR, and others excel in various embedded systems applications, the Raspberry Pi Zero 2 W stands out as a unique single-board computer. Here are some advantages of the Raspberry Pi Zero 2 W over traditional microcontrollers:

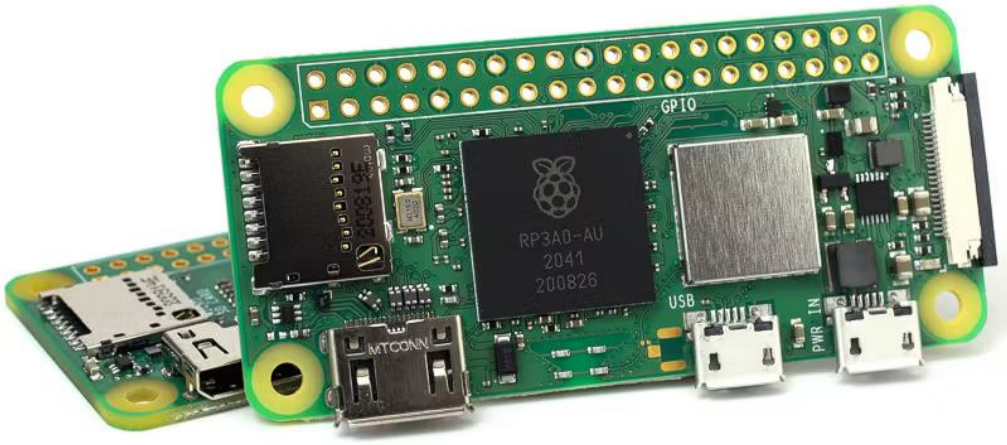

Figure 9 Raspberry Pi Zero 2w

2.1.1 FULL OPERATING SYSTEM:

Raspberry Pi Zero 2 W runs a full-fledged Linux operating system, offering a more complete computing environment. This allows users to carry out complex tasks, run multiple applications simultaneously, and exploit a wide range of programming languages.

2.1.2 MULTITASKING CAPABILITIES:

Unlike most microcontrollers that primarily perform a single program, the Raspberry Pi Zero 2 W can handle multitasking, making it suitable for applications that need simultaneous processing of multiple tasks.

2.1.3 BUILT-IN WIRELESS CONNECTIVITY:

The Raspberry Pi Zero 2 W accompanies built-in Wi-Fi and Bluetooth connectivity. This improves communication with other devices, facilitates IoT (Internet of Things) projects, and takes out the requirement for additional wireless modules.

2.1.4 HIGH PROCESSING POWER:

With a quad-core ARM Cortex-A53 processor running at 1 GHz, the Raspberry Pi Zero 2 W offers appreciably higher processing power compared to traditional microcontrollers. This makes it suitable for more demanding applications, including multimedia and web-related tasks.

2.1.5 VIDEO OUTPUT CAPABILITY:

Raspberry Pi Zero 2 W has a HDMI output, enabling it to connect to monitors or displays. This feature is mostly helpful for projects requiring graphical interfaces or multimedia applications, a feature missing in most microcontrollers.

2.1.6 EXTENSIVE COMMUNITY SUPPORT:

The Raspberry Pi ecosystem has a enormous and active community. This means to extensive online documentation, forums, tutorials, and a vast repository of software and projects. Users can effortlessly track down support, share knowledge, and access a wealth of resources.

2.1.7 USB PORTS FOR EXPANSION:

The Raspberry Pi Zero 2 W has USB ports that can be utilized for connecting additional peripherals, external storage, or other USB devices. This adaptability considers easy expansion and customization of the system.

2.1.8 COST-EFFECTIVE FOR CERTAIN APPLICATIONS:

While the Raspberry Pi Zero 2 W is more high-priced than many traditional microcontrollers, it still provides cost-effective solutions for projects that advantage from its features, such as networking capabilities, multimedia support, and a full operating system.

It's important to note that the choice between a Raspberry Pi and traditional microcontro99llers solely depends on the specific requirements of the project. Microcontrollers are frequently preferred for real-time control and low-power applications, while the Raspberry Pi Zero 2 W excels in projects severs a broader computing environment, connectivity, and multimedia capabilities.

2.2 SETTING UP RASPBERRY PI IN HEADLESS MODE

To set up your Raspberry Pi in headless mode, begin by flashing the preferred operating system (e.g., Raspbian) onto a microSD card using your computer. Upon completion, navigate to the root directory of the microSD card and create an empty file named 'ssh' (without a file extension) to enable the SSH service on the Raspberry Pi. For Wi-Fi configuration, create a file named 'wpa_supplicant.conf' and input the following details:

```
country=country_code
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
 ssid="your_SSID"
 psk="your_password"
}
```

Replace "country_code" and provide your Wi-Fi credentials. Insert the microSD card into the Raspberry Pi, power it up, and it will connect to the Wi-Fi network. Now, you can remotely access your Raspberry Pi using an SSH client with the default username 'pi' and password 'raspberry.' This headless setup eliminates the need for a dedicated monitor, keyboard, or mouse, streamlining your Raspberry Pi project.

2.3 L298N DOUBLE H-EXTENSION ENGINE DRIVER

This double bidirectional engine driver depends on the exceptionally well known L298 Double H-Extension Engine Driver Incorporated Circuit. The circuit will permit you to effectively and freely control two engines of up to 2A each in the two headings. It is great for mechanical applications and appropriate for association with a microcontroller requiring only two or three control lines for each engine. It can likewise be connected with straightforward manual switches, TTL rationale entryways, transfers, and so forth. This board outfitted with power Drove pointers, on-board +5V controller and security diodes.

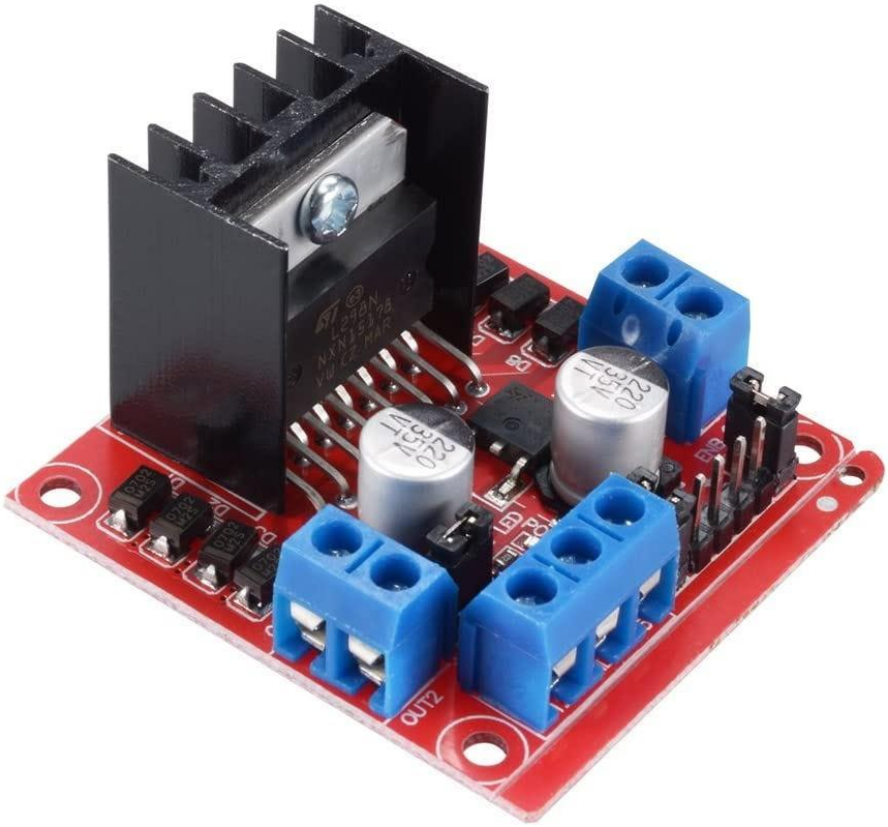

Figure 10 L298n Motor Driver

2.3.1 Brief Data:

- Input voltage: 3.2V~40V dc.
- Driver: L298N Dual H Bridge Dc Motor Driver
- Power Supply: DC 5 V - 35 V
- Peak current: 2 Amp
- Operating current range: 0 ~ 36mA
- Control signal input voltage range :
- Low: -0.3V ≤ Vin ≤ 1.5V.
- High: 2.3V ≤ Vin ≤ Vss.

- Enable signal input voltage range :
    - Low: -0.3 ≤ Vin ≤ 1.5V (control signal is invalid).
    - High: 2.3V ≤ Vin ≤ Vss (control signal active).
- Maximum power consumption: 20W (when the temperature T = 75 ℃).
- Storage temperature: -25 ℃ ~ +130 ℃.
- On-board +5V regulated Output supply (supply to controller board i.e. Arduino).
- Size: 3.4cm x 4.3cm x 2.7cm

2.3.2 Schematic Diagram:

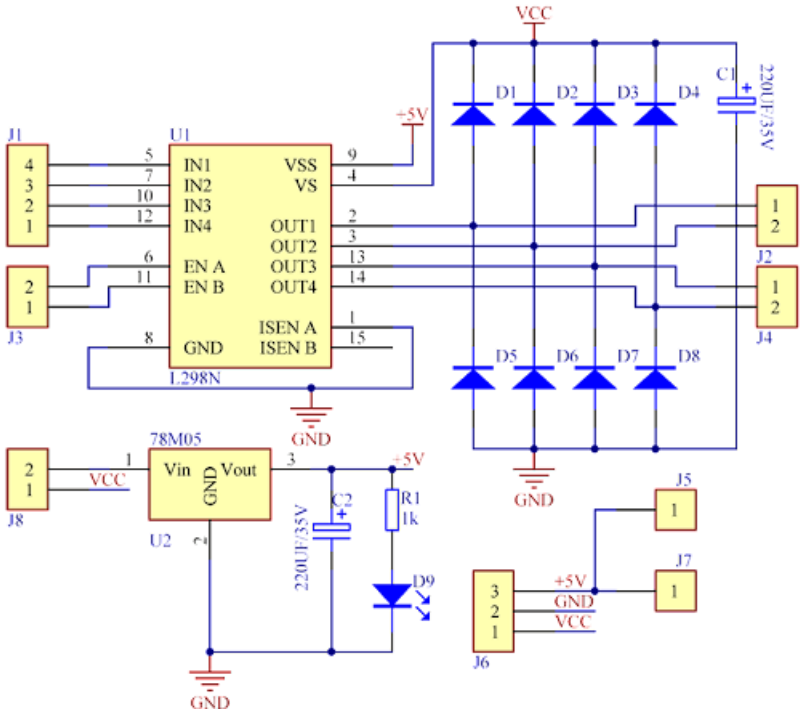

Figure 11 L298n Circuit Diagram

2.3.3 Board Dimensions and Pin Functions:

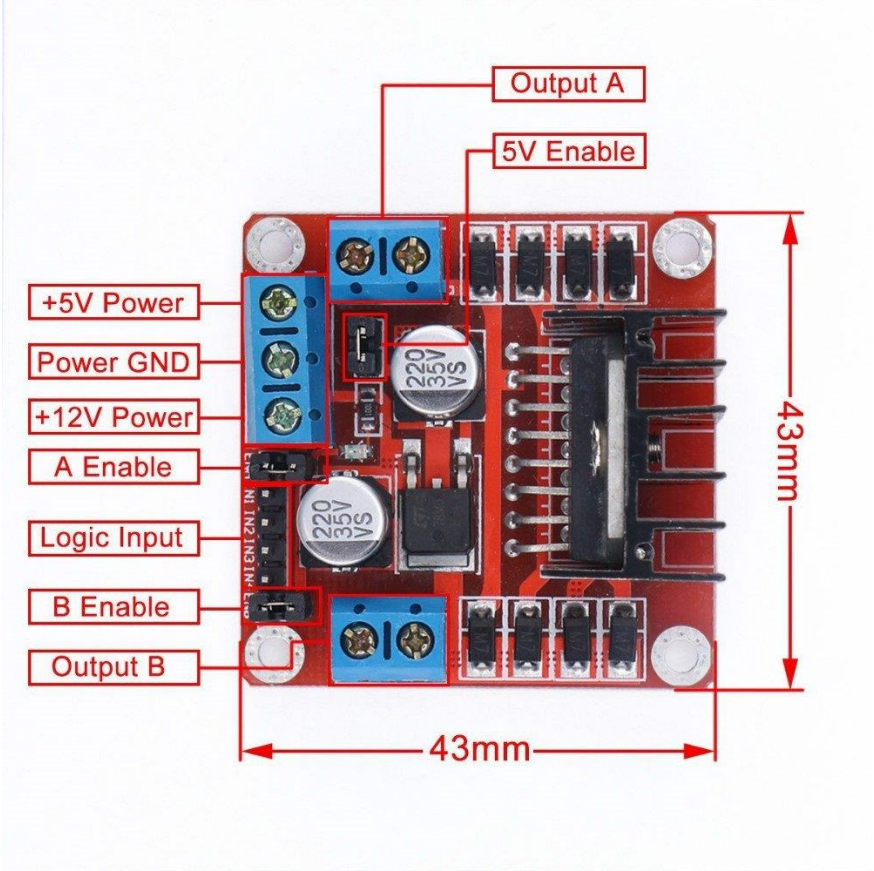

Figure 12 L298n Pins

2.3.4 Connection Example:

Communicating a L298N motor driver, two DC motors, and a Raspberry Pi Zero 2 W incorporates spreading out power, control, and motor affiliations. Coming up next is a point by point model:

Materials:

- Raspberry Pi Zero 2 W
- L298N Motor Driver Module
- Two DC Motors
- Power Source (e.g., Battery Pack)
- Jumper Wires

Power Supply:

- Partner the positive (+) terminal of the battery pack to the VCC2 nail to the L298N module.

- Interface the negative (-) terminal of the battery pack to the GND nail to the L298N module.

- On the other hand, interface an alternate 5V power source to the VCC1 pin for the reasoning equipment. Interface the ground of this source to the GND nail to the L298N.

Motor Affiliations:

- Partner one terminal of the essential DC motor to OUT1 and the other terminal to OUT2 on the L298N.

- Partner one terminal of the resulting DC motor to OUT3 and the other terminal to OUT4 on the L298N.

Raspberry Pi Affiliations:

- Partner GPIO pins on the Raspberry Pi to the IN1, IN2, IN3, and IN4 pins on the L298N module.

- For example, interface GPIO17 to IN1, GPIO18 to IN2, GPIO22 to IN3, and GPIO23 to IN4. Try to plan these GPIO pins in your code.

Engage Pins:

- Partner GPIO pins on the Raspberry Pi to the ENA and ENB pins on the L298N module. These pins control the speed of the motors through PWM.

- For instance, interface GPIO12 to ENA and GPIO13 to ENB.

Ground Affiliation:

- Interface a ground (GND) pin from the Raspberry Pi to the GND nail to the L298N module.

Additional Considerations:

- Ensure that the motors' voltage rating matches the power supply voltage.

- Use outside diodes across the motor terminals to thwart back EMF.

- Present the basic programming libraries on the Raspberry Pi to control the motors through GPIO.

By following these affiliations, you can make a principal plan for controlling two DC motors using the L298N motor driver and the Raspberry Pi Zero 2 W. Persistently counsel the datasheets and subtleties of your parts for accurate information and plans.

Code:

To control the two DC motors connected to the L298N motor driver using a Raspberry Pi Zero 2 W, you can use a programming language like Python. Below is a basic example using the GPIO library.

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins
ENA = 12
IN1 = 17
IN2 = 18

ENB = 13
IN3 = 22
IN4 = 23

# Set GPIO mode and configure pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(ENA, GPIO.OUT)
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)

GPIO.setup(ENB, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)

# Create PWM instances for motor speed control
pwm_motor_a = GPIO.PWM(ENA, 1000)
pwm_motor_b = GPIO.PWM(ENB, 1000)

# Start PWM with 0% duty cycle (motors initially off)
pwm_motor_a.start(0)
pwm_motor_b.start(0)
```

```
try:
    # Move the motors forward for 2 seconds
    GPIO.output(IN1, GPIO.HIGH)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.HIGH)
    GPIO.output(IN4, GPIO.LOW)

    pwm_motor_a.ChangeDutyCycle(50)  # Adjust speed by changing duty cycle
    pwm_motor_b.ChangeDutyCycle(50)

    time.sleep(2)

    # Stop the motors
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.LOW)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.LOW)

    # Move the motors backward for 2 seconds
    GPIO.output(IN1, GPIO.LOW)
    GPIO.output(IN2, GPIO.HIGH)
    GPIO.output(IN3, GPIO.LOW)
    GPIO.output(IN4, GPIO.HIGH)

    time.sleep(2)

finally:
    # Clean up GPIO settings and stop PWM
    GPIO.cleanup( )
    pwm_motor_a.stop( )
    pwm_motor_b.stop( )
```

This Python script shows moving the connected motors forward, changing their speed, stopping them, and moving them backward.

2.4 US-100 ULTRASONIC SENSOR MODULE

The Ultrasonic sensor module is a helpful way for estimating distances from the objects. This module has a great deal of utilizations like parking sensor, obstacle and terrain monitoring systems, industrial distance measurements, and so on. It has a stable execution and high precision going from 2cm to 450cm. The module sends an ultrasonic signals, eight pulses of 40 kHz square wave from the transmitter; the echo is then picked up by the receiver and the outputs a waveform with a time span corresponding to the distance. The connected microcontroller acknowledges the signal and performs fundamental handling.

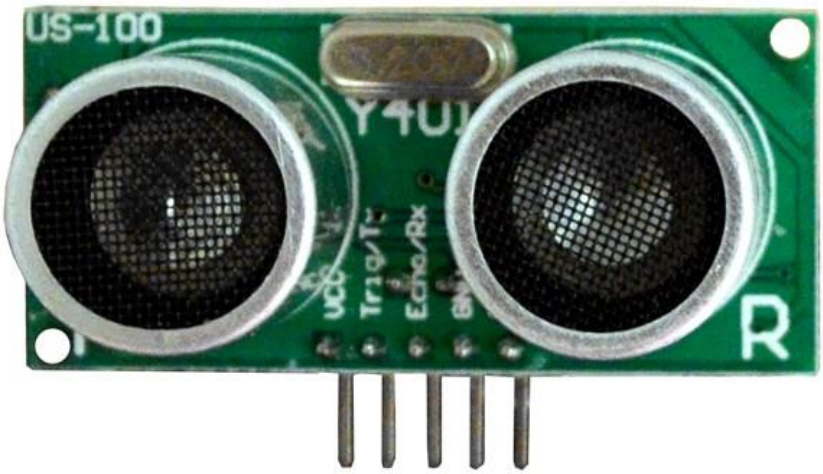

Figure 13 US-100 Sensor

2.4.1 Specifications

- Input voltage: 3.3/5V DC
- Quiescent current: less than 2mA
- Level output: 3.3/5V high
- Level output: at the end of 0V
- Induction angle: not more than 15 degrees
- Detection distance: 2cm-450cm
- Precision: up to 1mm
- Dimensions: 4.4cm x 2.6cm x 1.4cm
- Weight: 43g

2.4.2 Pin Configuration

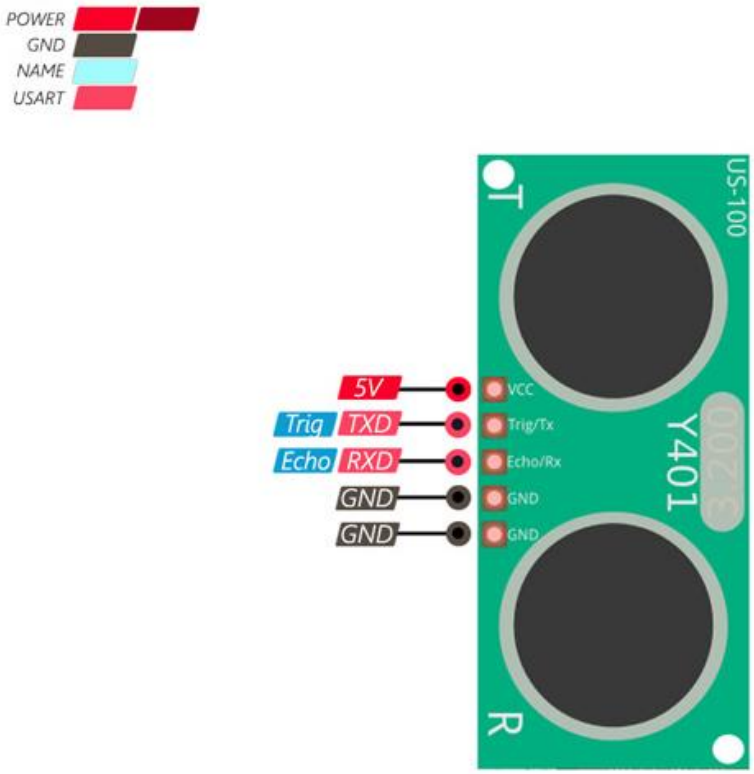

Figure 14 US-100 Pins

- VCC: 3.3/5V DC
- Trig: trigger input
- Echo: pulse output
- GND: ground
- GND: ground

2.4.3 Schematic Diagram

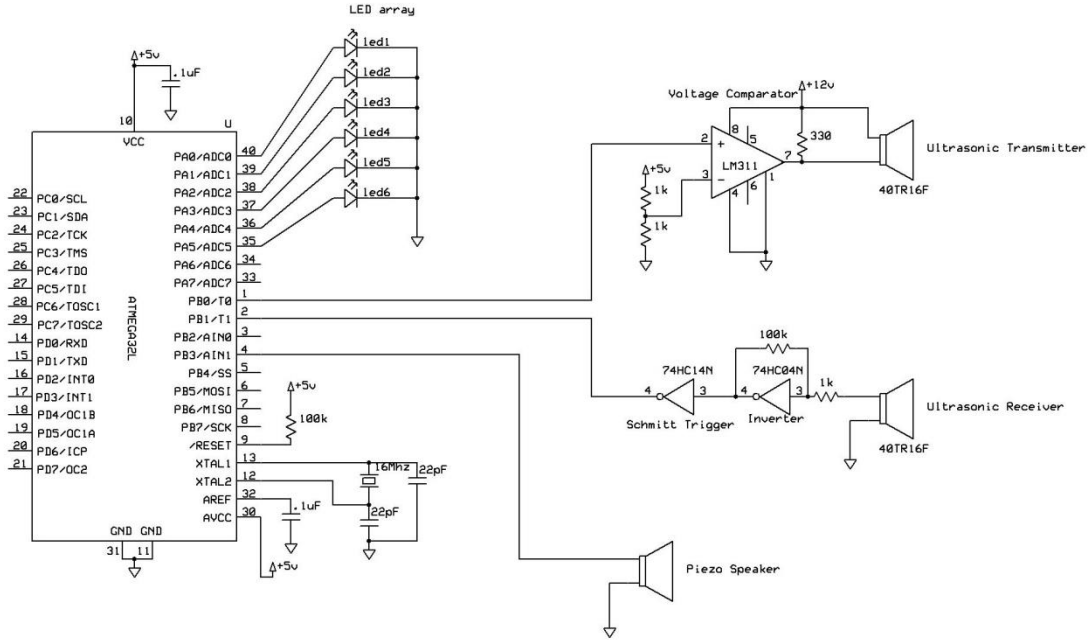

Figure 15 Ultrasonic Sensor Cricuit Diagram

2.4.4 Connection Example

Connecting a US-100 Ultrasonic Sensor to a Raspberry Pi Zero 2 W includes wiring the parts and composing a Python script to read and show the distance measurements. The following is a detailed example:

Materials:

- Raspberry Pi Zero 2 W
- US-100 Ultrasonic Sensor
- Jumper Wires

Connections:

Connect the US-100 Ultrasonic Sensor:

- Connect the VCC pin of the US-100 to the 3.3V power (Pin 1) on the Raspberry Pi.
- Connect the GND pin of the US-100 to the ground (Pin 6) on the Raspberry Pi.
- Connect the Trig pin of the US-100 to GPIO pin 8.

- Connect the Echo pin of the US-100 to GPIO pin 10

Code

```
import serial
import adafruit_us100
import time

uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=1)

us100 = adafruit_us100.US100(uart)

while True:
    print("-----")
    print("Temperature: ", us100.temperature)
    time.sleep(0.5)
    print("Distance: ", us100.distance)
    time.sleep(0.5)
```

Explanation:

- The script utilizes the US-100 to measure the distance by sending ultrasonic pulses and measuring the time it takes for the pulses to return.
- The measured distance is then printed to the console, and the loop continues to update the distance every second.

Make sure to run the script with appropriate permissions (sudo) if you encounter permission issues while accessing GPIO pins. This script provides a basic example of reading and displaying distance measurements from the US-100 using the Raspberry Pi Zero 2 W.

2.5 RASPBERRY PI CAMERA MODULE

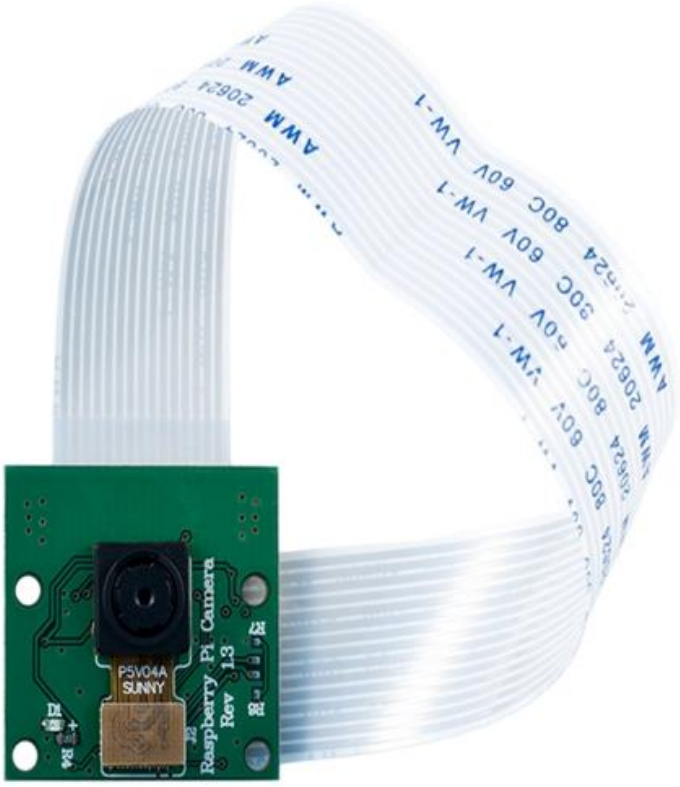

Figure 16 PiCamera

The Raspberry Pi Camera Module has gained popularity as an add-on for the Raspberry Pi single-board computer. It enables users to directly capture images and videos. It is a camera module that is compatible with both the Raspberry Pi Model A and Model B is available. It offers high sensitivity, minimal cross talk, and low noise for capturing high-definition images in a compact and lightweight design. To connect the camera module to the Raspberry Pi board, a CSI connector specifically designed for camera interfacing is used. The CSI bus has the capability to handle extremely high data rates and is solely responsible for transmitting pixel data to the BCM2835 processor.

2.5.1 Schematic Diagram

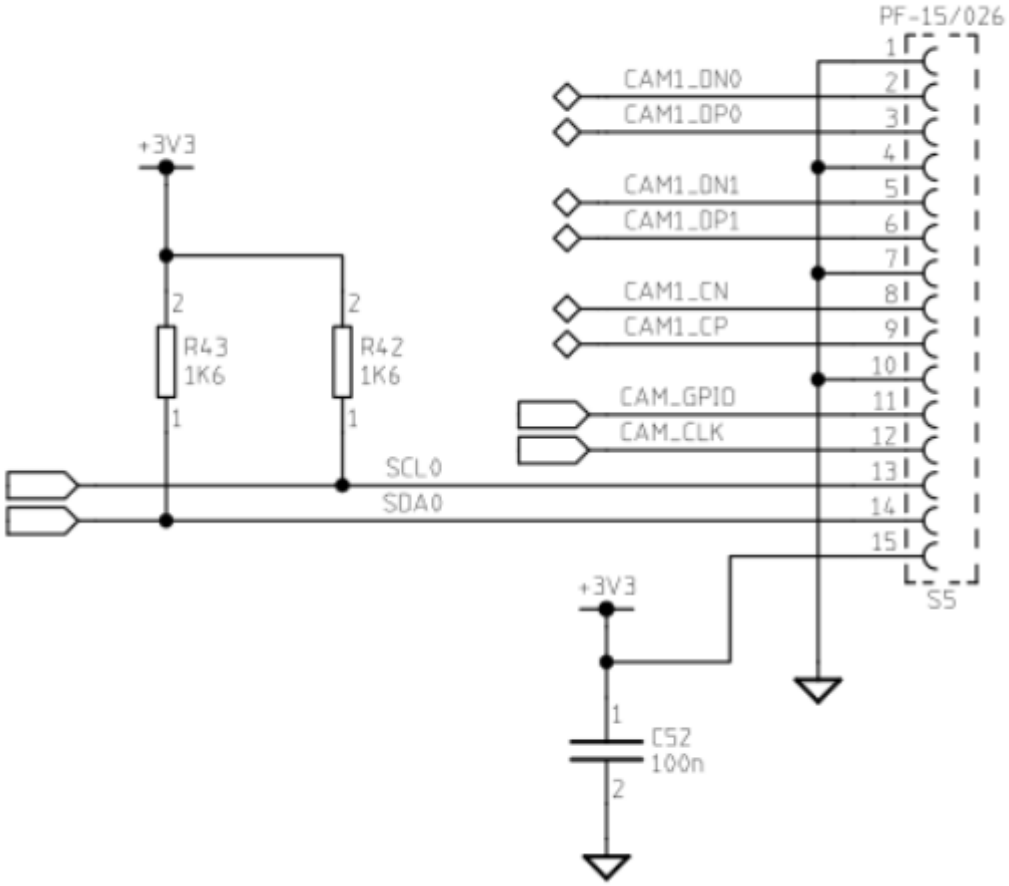

Figure 17 Picamera Circuit Diagram

2.5.2 Connecting the Camera

The flex cable inserts into the connector labelled CAMERA on the Raspberry Pi, which is located between the Ethernet and HDMI ports. The cable must be inserted with the silver contacts facing the HDMI port. To open the connector, pull the tabs on the top of the connector upwards, then towards the Ethernet port. The flex cable should be inserted firmly into the connector, with care taken not to bend the flex at too acute an angle. To close the connector, push the top part of the connector towards the HDMI port and down, while holding the flex cable in place.

2.5.3 Code to Interface With RPi:

```
import time
import picamera

def capture_image(file_path):
    with picamera.PiCamera() as camera:
        camera.resolution = (1920, 1080)
        camera.rotation = 180  # Rotate the image if needed

        # Wait for the camera to warm up
        time.sleep(2)

        # Capture an image
        camera.capture(file_path)

if  _name__ == "__main__":
    image_file_path = "captured_image.jpg"
    capture_image(image_file_path)
    print(f"Image captured and saved to {image_file_path}")
```

This script defines a function **capture image** that takes a file path as an argument and captures an image using the Raspberry Pi Camera. The script then calls this function with a specified file path and prints a message indicating where the image is saved.

CHAPTER 3

PYTHON WITH RASPBERRY PI

3.1 PROGRAMMING AND MICROPROCESSORS

Programming microprocessors, particularly on platforms like the Raspberry Pi, requires combining high-level and low-level languages, interacting with hardware, and implementing various applications for various purposes. Professionals, educators, and enthusiasts alike now have much easier access to microprocessor programming thanks to the Raspberry Pi's growing popularity. One kind of central processing unit (CPU) found in a microcomputer is called a microprocessor. These gadgets include the Raspberry Pi, for instance. A compact integrated circuit (CIC) contains the memory, control unit, and arithmetic logic unit (ALU) components. Microprocessors are specifically used by the Raspberry Pi line of single-board computers (SBCs) to provide a compact, reasonably priced computing platform.

3.2 PYTHON PROGRAMMING LANGUAGE

High-level and multipurpose, Python is renowned for its ease of use and readability. Python was developed by Guido van Rossum and was initially made available in 1991. It places a strong emphasis on clear and simple code, which makes it a great option for both novice and seasoned developers. Its syntax promotes the clear and organic expression of ideas, leading to code that is simple to read and update. Python is widely used in many different domains because of its interpreted and dynamic typing, which facilitate quick development and testing.

The language has an extensive standard library that includes modules for a wide range of tasks, from web development and data science to file manipulation and networking. Modular and reusable code is made easier by Python's support for object-oriented programming (OOP), and its vast ecosystem of third-party libraries—hosted on sites like PyPI—allows it to do even more. Because of Python's cross-platform capabilities, code can run smoothly on a variety of operating systems, and the language's vibrant community guarantees ongoing

development and support.

In domains like automation, data science, machine learning, and web development, Python has emerged as a formidable force. Web application development is made easier by frameworks like Django and Flask, and data scientists and machine learning practitioners are empowered by libraries like NumPy, pandas, and scikit-learn. Python's popularity has been fuelled by its readability and versatility, which have established it as the preferred language for a variety of tasks, from small scripts to large-scale projects. The philosophy of the language is embodied in The Zen of Python, a collection of guidelines for writing computer programs in Python that place an emphasis on clarity, simplicity, and usefulness.

3.3 INTERFACING PROJECT LIBRARIES

There are any kinds of Python libraries that has its own purpose with its specific task, of course it even includes libraries that can interface with microprocessos to control specific devices. As such these are the python libraries used for this project:

3.3.1 RPi.GPIO:

A handy interface for managing and interacting with the Raspberry Pi's GPIO (General Purpose Input/Output) pins is offered by the Python library RPi.GPIO, which was created especially for the Raspberry Pi. RPi.GPIO was designed to make it easier for developers to configure, read, and write to GPIO pins on the Raspberry Pi when integrating hardware components with Python programs. With the help of the library's abstraction layer for GPIO operations, users can manage events like button presses and sensor readings, adjust pin modes, and control digital input and output. For projects involving robotics, home automation, and electronics particularly, RPi.GPIO is very helpful as it provides an easy-to-use method for utilizing the Raspberry Pi's GPIO capabilities in Python scripts. RPi.GPIO is an essential tool for integrating Python code with the Raspberry Pi's hardware features because of its adaptability and simplicity of use. Its purpose in this project is to provide precise signals to each hardware components according to make it functional.

3.3.2 Pygame:

Pygame provides an intuitive interface for managing keyboard events, which simplifies the integration of keyboard controls into games and multimedia applications. The pygame.event module gives developers access to the library's

event handling system, which makes it simple to record and handle key presses, releases, and other keyboard-related events. By offering constants for every key on the keyboard, Pygame makes it easier to identify individual keys and makes control integration simple. Developers can effectively ascertain the current state of keys through functions such as pygame.key.get_pressed(), which enable continuous detection and response to pressed keys. Pygame allows you to create controls that can be used for a single key press or multiple key presses at once, giving you flexibility for a variety of input scenarios. Additionally, the library provides features like customization for key repeating, which lets developers adjust the interval and delay for repeated events that are triggered by held keys. All in all, Pygame is a top option for developers who want to create responsive, interactive apps with easy-to-use keyboard controls because of its event-driven architecture and platform independence.
Here Pygame is used as a link between the keyboard controls and the components to control its functions.

3.3.3 Adafruit_us100 and serial:

Adafruit_us100 is a simple Circuit Python library that enables the control of us 100(ultrasonic sensor) for reading distance and temperature. Whereas Python's serial library is a powerful tool for serial communication, allowing developers to connect and exchange data between devices via serial ports. The serial library supports both reading and writing data to serial ports, making it useful for tasks like interfacing with microcontrollers, sensors, and other embedded systems. The ability of the US 100 to interface with the Raspberry Pi's UART pins allows for rapid communication via the connection of receiver and transmitter signals.

3.3.4 PiCamera:

The Python picamera library offers a convenient and all-inclusive toolkit for taking pictures and making videos, and it is specifically made to interface with the Raspberry Pi camera module. The picamera library, designed to work seamlessly with the Raspberry Pi hardware, makes it easier to perform tasks like setting up the camera, taking still photos, and making quality videos. It enables precise customization for various imaging requirements by giving developers control over a number of parameters, such as exposure, frame rate, and resolution. Advanced features like streaming video and rapid image capture are also supported by the library. The picamera library is a crucial tool for projects involving computer vision, photography, and multimedia applications on the Raspberry Pi platform because of its simplicity of use and compatibility with the ecosystem.

3.3.5 Random, Time and Sys

Random library is a useful tool for creating random numbers and making stochastic decisions, the Python random library is essential for applications such as games, simulations, and cryptography. It offers functions for producing random selections from sequences, random integers, and floating-point numbers, which helps programs produce a wide range of unexpected results.
When it comes to time-related Python functionality, the time library is an invaluable tool for tasks like handling timestamps, adding delays, and measuring execution duration. It is useful in situations where exact timing is crucial, like animation, scheduling, or benchmarking, because it offers precise time measurements and easy conversions between various time units.
Through interaction with the Python runtime environment, the sys library gives users access to parameters and functions that are unique to the system. It is frequently used by developers for tasks like interacting with the Python interpreter, retrieving arguments from the command line, and managing configurations pertaining to the system. By acting as a link between the Python script and the underlying system functionalities, sys improves the adaptability of Python programs to various environments and operating systems.

3.4 PROJECT DIAGRAM

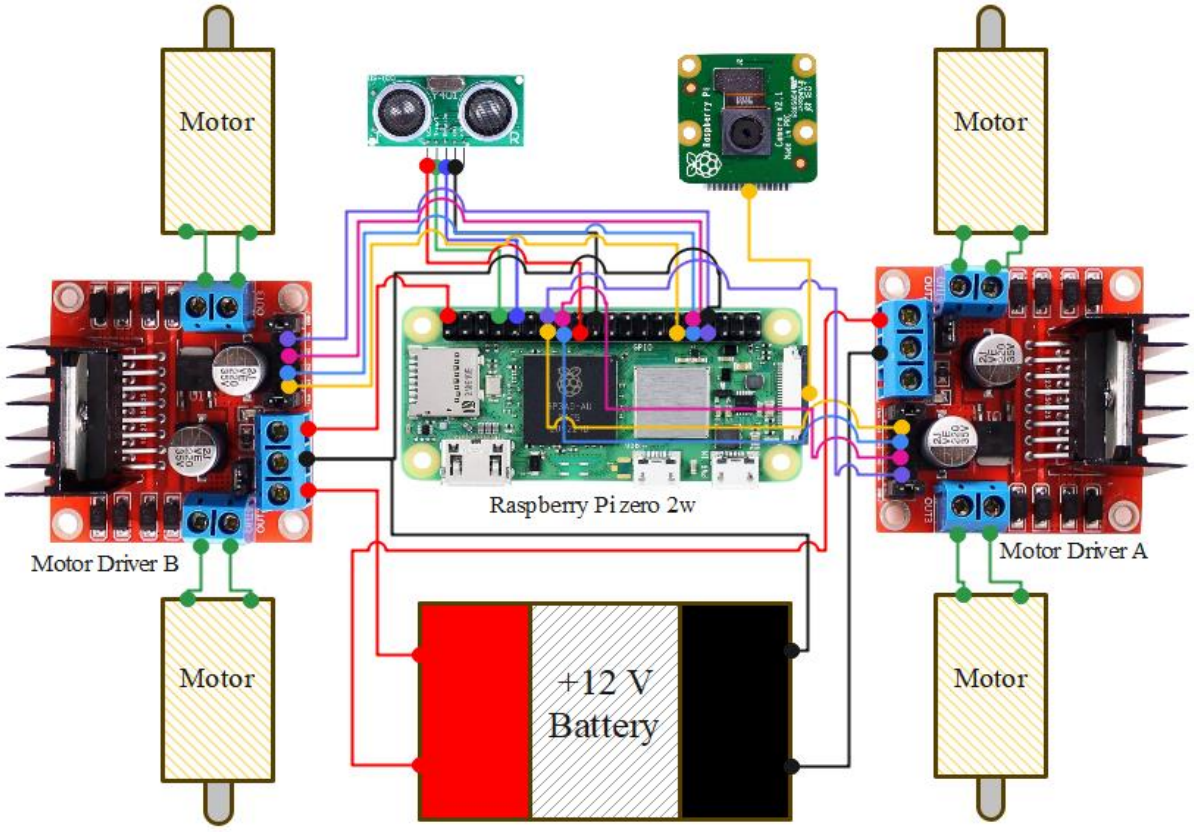

Figure 18 Final Circuit Diagram

Pin connection to Raspberry pi

Motor driver A
in1 = 13
in2 = 15
in3 = 16
in4 = 18
GND = 14

Motor driver B
in1 = 29
in2 = 31
in3 = 32
in4 = 33
GND = 34

US-100
Tx = 8
Rx = 10
Vcc = 17
GND = 20

3.5 CONTROL CODE

'''
import RPi.GPIO as GPIO
import adafruit_us100
import serial
import pygame
from picamera import Picamera
import random
import time
import sys '''

Here necessary libraries for the interfacing is being imported.

'''
GPIO.setmode(GPIO.BOARD)

uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=3)
print(uart)
us100 = adafruit_us100.US100(uart)

pygame.init()

pygame.display.set_mode((500, 500))
pygame.display.set_caption('AI RC') '''

Here we are setting the GPIO board pin to its physical pin numbers after that we are defining the UART pin connection with the us 100 and initiating the pygame module for accessing keyboard controls.

```
'''
in1_a = 13
in2_a = 15
in3_a = 16
in4_a = 18

in1_b = 29
in2_b = 31
in3_b = 32
in4_b = 33

GPIO.setup(in1_a, GPIO.OUT)
GPIO.setup(in2_a, GPIO.OUT)
GPIO.setup(in3_a, GPIO.OUT)
GPIO.setup(in4_a, GPIO.OUT)
GPIO.setup(in1_b, GPIO.OUT)
GPIO.setup(in2_b, GPIO.OUT)
GPIO.setup(in3_b, GPIO.OUT)
GPIO.setup(in4_b, GPIO.OUT)'''
```

Defining connected input pin numbers from the L298n motor driver with the raspberrypi and setting them up as output pins.

```
'''
forward = False
backward = False
left = False
right = False
nw_direction = False
ne_direction = False
sw_direction = False
se_direction = False
rotate_left = False
rotate_right = False

def moveForward():
    GPIO.output(in1_a, GPIO.HIGH)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.HIGH)
```

```
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def moveBackward():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.HIGH)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def moveLeft():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.HIGH)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.HIGH)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def moveRight():
    GPIO.output(in1_a, GPIO.HIGH)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def moveNW():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.LOW)
```

```
    GPIO.output(in3_a, GPIO.HIGH)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def moveNE():
    GPIO.output(in1_a, GPIO.HIGH)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.LOW)

def moveSW():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.HIGH)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.LOW)

def moveSE():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def rotateLeft():
    GPIO.output(in1_a, GPIO.HIGH)
```

```
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.HIGH)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def rotateRight():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.HIGH)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def stop():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.LOW) '''
```

Here the directional flags are set to false and defining the directional function for specific signals from raspberrypi for different movements.

```
'''
camera = PiCamera()
camera.resolution = (1920, 1080)

save_directory = "/home/neko/Desktop/py_projects/rc_prject/py_images"
def captureImage():
    timestamp = time.strftime("%d-%m-%Y_%H:%M:%S")
    image_path = f"{save_directory}image_{timestamp}.jpg"
    print('Capturing Image...')
```

```
    camera.capture(image_path)
    print(f"Image captured and saved to {image_path}") '''
```

Here picamera is initiated and a function for camera capture is defined where path for the jpg file is also defined.

```
'''
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        GPIO.cleanup()
        pygame.quit()
        sys.exit()

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_KP8:
            forward = True
        if event.key == pygame.K_KP2:
            backward = True
        if event.key == pygame.K_KP4:
            left = True
        if event.key == pygame.K_KP6:
            right = True

        if event.key == pygame.K_KP7:
            nw_direction = True
        if event.key == pygame.K_KP9:
            ne_direction = True
        if event.key == pygame.K_KP1:
            sw_direction = True
        if event.key == pygame.K_KP3:
             se_direction = True
        if event.key == pygame.K_KP5:
             stop()

        if event.key == pygame.K_a:
             rotate_left = True
        if event.key == pygame.K_d:
             rotate_right = True

        if event.type == pygame.KEYUP:
             forward = backward = left = right = nw_direction = ne_direction = sw_direction = se_direction = rotate_left = rotate_right = False
             stop()
```

```
if forward:
    moveForward()

if backward:
    moveBackward()

if left:
    moveLeft()

if right:
    moveRight()

if nw_direction:
    moveNW()

if ne_direction:
    moveNE()

if sw_direction:
    moveSW()

if se_direction:
    moveSE()

if rotate_left:
    rotateLeft()

if rotate_right:
    rotateRight() '''
```

These code initiate a for loop the events in a keyboard and keyboard buttons like 7, 8, 9, 4, 5, 6, 1, 2, 3, a, d are the given for the key down events to acquire pressing inputs for the direction north east, forward, north west, horizontal left, stopping, horizontal right, south west, backward, south east, rotate left and rotate right directions respectively. These directions are set to false in key up events to stop the movement and directional functions are called outside the for loop when the key down event set a direction to true.

```
'''
if event.key == pygame.K_o:
    captureImage() '''
```

This is the script for image capture that is inside the for loop keydown event that calls the captureImage function when button 'o' is pressed.

```
'''
if event.key == pygame.K_p:
    auto_mode = True

    while auto_mode:
        distance = us100.distance
        time.sleep(0.5)
        print(distance)
        if distance <= 25:
            rndir = random.choice(random_rotate_direction)
            stop()
            captureImage()
            time.sleep(1)
            while True:
                distance = us100.distance
                rndir()
                if distance > 25:
                    stop()
                    time.sleep(1)
                    break
                else:
                    moveForward()
                    time.sleep(0.1)
                for event in pygame.event.get():
                    if event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_p:
                            auto_mode = False '''
```

This is the script that initiates a key down event by clicking p button for obstacle detection control that initiates the us 100 sensors for distance calculation. If the distance from an obstacle is less than 25cm it stops the vehicle and rotates it in a random direction for a specified time and continues running forward until there is an obstacle Infront of it. This runs until the p button is clicked again that changes it back to manual control.

```
'''
except KeyboardInterrupt:
    camera.close()
    GPIO.cleanup() '''
```

All the code is set in an infinite loop inside the try block with exception of keyboard interrupt (ctrl + c) that stops the program and cleans the GPIO pins and camera.

3.6 COMPLETE CODE

```
# Importing necessary libraries

import RPi.GPIO as GPIO
import adafruit_us100
import serial
import pygame
from picamera import Picamera
import random
import time
import sys

# Setting up GPIO nummbering to physcial pin numbers
GPIO.setmode(GPIO.BOARD)

# Defining UART pins for ultrasonic sensor
uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=3)
print(uart)
us100 = adafruit_us100.US100(uart)

# Motor driver a pins
in1_a = 13
in2_a = 15
in3_a = 16
in4_a = 18

# Motor driver b pins
in1_b = 29
in2_b = 31
in3_b = 32
in4_b = 33

# Setting motor pins
GPIO.setup(in1_a, GPIO.OUT)
GPIO.setup(in2_a, GPIO.OUT)
GPIO.setup(in3_a, GPIO.OUT)
GPIO.setup(in4_a, GPIO.OUT)
```

```
GPIO.setup(in1_b, GPIO.OUT)
GPIO.setup(in2_b, GPIO.OUT)
GPIO.setup(in3_b, GPIO.OUT)
GPIO.setup(in4_b, GPIO.OUT)

# Initializing pygame
pygame.init()

# Setting up pygame display
pygame.display.set_mode((500, 500))
pygame.display.set_caption('AI RC')

# Setting direction flags
forward = False
backward = False
left = False
right = False
nw_direction = False
ne_direction = False
sw_direction = False
se_direction = False
rotate_left = False
rotate_right = False

# Defining functions for movement
def moveForward():
    GPIO.output(in1_a, GPIO.HIGH)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.HIGH)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def moveBackward():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.HIGH)
```

```
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def moveLeft():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.HIGH)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.HIGH)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def moveRight():
    GPIO.output(in1_a, GPIO.HIGH)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def moveNW():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.HIGH)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def moveNE():
    GPIO.output(in1_a, GPIO.HIGH)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.LOW)
```

```
    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.LOW)

def moveSW():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.HIGH)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.LOW)

def moveSE():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def rotateLeft():
    GPIO.output(in1_a, GPIO.HIGH)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.HIGH)

    GPIO.output(in1_b, GPIO.HIGH)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.HIGH)

def rotateRight():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.HIGH)
    GPIO.output(in3_a, GPIO.HIGH)
    GPIO.output(in4_a, GPIO.LOW)
```

```
    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.HIGH)
    GPIO.output(in3_b, GPIO.HIGH)
    GPIO.output(in4_b, GPIO.LOW)

def stop():
    GPIO.output(in1_a, GPIO.LOW)
    GPIO.output(in2_a, GPIO.LOW)
    GPIO.output(in3_a, GPIO.LOW)
    GPIO.output(in4_a, GPIO.LOW)

    GPIO.output(in1_b, GPIO.LOW)
    GPIO.output(in2_b, GPIO.LOW)
    GPIO.output(in3_b, GPIO.LOW)
    GPIO.output(in4_b, GPIO.LOW)

# Setting up picamera
camera = PiCamera()
camera.resolution = (1920, 1080)

# Providing directory location to save pictures
save_directory = "/home/neko/Desktop/py_projects/rc_prject/py_images"

# Defining function for image capture
def captureImage():
    timestamp = time.strftime("%d-%m-%Y_%H:%M:%S")
    image_path = f"{save_directory}image_{timestamp}.jpg"
    print('Capturing Image...')
    camera.capture(image_path)
    print(f"Image captured and saved to {image_path}")

random_rotate_direction = [rotateLeft, rotateRight]

try:
    # Setting up infinite loop script
    while True:
        # Setting up pygame for acquiring keyboard events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:  # if case for exiting program safely
                GPIO.cleanup()
                pygame.quit()
                sys.exit()
```

```
# Acquiring keydown events from keyboard for diffent direction
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_KP8:
        forward = True
    if event.key == pygame.K_KP2:
        backward = True
    if event.key == pygame.K_KP4:
        left = True
    if event.key == pygame.K_KP6:
        right = True

    if event.key == pygame.K_KP7:
        nw_direction = True
    if event.key == pygame.K_KP9:
        ne_direction = True
    if event.key == pygame.K_KP1:
        sw_direction = True
    if event.key == pygame.K_KP3:
        se_direction = True
    if event.key == pygame.K_KP5:
        stop()

    if event.key == pygame.K_a:
        rotate_left = True
    if event.key == pygame.K_d:
        rotate_right = True

    # Keydown event for image capture
        if event.key == pygame.K_o:
        captureImage()

    # Keydown event for obstacle detection
    if event.key == pygame.K_p:
        auto_mode = True

        # Loop block for directional navigation with US 100
        while auto_mode:
            distance = us100.distance
            time.sleep(0.5)
            print(distance)
            if distance <= 25:
                rndir = random.choice(random_rotate_direction)
```

```
                    stop()
                    captureImage()
                    time.sleep(1)
                    while True:
                        distance = us100.distance
                        rndir()
                        if distance > 25:
                            stop()
                            time.sleep(1)
                            break
                else:
                    moveForward()
                    time.sleep(0.1)
                      # Keydown event to stop obstacle detection and revert to manual
                for event in pygame.event.get():
                    if event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_p:
                            auto_mode = False

            # Acquiring Keyup event to detect stop inputs
            if event.type == pygame.KEYUP:
                forward = backward = left = right = nw_direction = ne_direction = sw_direction = se_direction = rotate_left = rotate_right = False
                stop()

        # if cases for each direction
        if forward:
            moveForward()

        if backward:
            moveBackward()

        if left:
            moveLeft()

        if right:
            moveRight()

        if nw_direction:
            moveNW()

        if ne_direction:
```

```
            moveNE()

        if sw_direction:
            moveSW()

        if se_direction:
            moveSE()

        if rotate_left:
            rotateLeft()

        if rotate_right:
            rotateRight()

# Exception block to stop program with Keyboard interrupt(ctrl+c) and cleaning
up Rpi
except KeyboardInterrupt:
    camera.close()
    GPIO.cleanup()
```

CONCLUSION

In summary, the creation and execution of the Raspberry Pi-based rover that can be controlled remotely and has obstacle avoidance capabilities offer a promising solution for remote exploration and surveillance. The rover's small and efficient design, along with the integration of ultrasonic sensors and user-friendly control interfaces, provides a flexible platform for a wide range of applications. These applications can span from educational projects to practical uses in home automation, security, and research. By incorporating Python programming and OpenCV, the rover is able to adapt and be customized to meet specific needs, further enhancing its usefulness. This project successfully combines affordability, effectiveness, and adaptability, making the rover accessible and invaluable to both hobbyists and professionals who require a versatile and intelligent remote-controlled platform.

REFERENCE

1. https://www.raspberrypi.com/documentation/computers/getting-started.html
2. https://raspberrypi.com/documentation/computers/remote-access.html
3. https://forums.raspberrypi.com/?_gl=1*dptz0i*_ga*MTQ3NTc2MzAxOS4xNzAwMDQ3ODY5*_ga_22FD70LWDS*MTcwNzAzNjUwNS4xLjEuMTcwNzAzNjcyMi4wLjAuMA..
4. https://robocraze.com/blogs/post/what-is-motor-driver
5. https://components101.com/modules/l293n-motor-driver-module
6. https://maxbotix.com/blogs/blog/how-ultrasonic-sensors-work#:~:text=What%20is%20an%20Ultrasonic%20Sensor,information%20about%20an%20object's%20proximity
7. https://projects.raspberrypi.org/en/projects/getting-started-with-picamera
8. https://embeddedschool.in/different-types-of-microcontroller-programming-used-in-embedded-systems/