Government Of India

Ministry Of Defence

Defence Research & Development Organisation(DRDO)

Defence Research & Development Laboratory (DRDL)

P.O Kanchanbagh Hyderabad- 500058

Summer Internship Project Report

On

“Object Detection and Distance Monitoring System”

Tenure :- 4 weeks (1st July-31st July)

Submitted By

MANCHIKATLA PAVITHRA

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

160122749305

Under the Guidance of

SHRI MAYANK MISHRA , SC, ‘E’

INDEX

TOPIC	PAGE NO
1 . INTERNET OF THNGS	**1**
1.1 INTRODUCTION	
1.2 IOT COMPONENTS	
1.3 IOT APPLICATIONS	
2 . BASICS OF ELECTRONICS	**4**
2.1 COMPONENTS OF ELECTRONICS	
2.2 ELECTRONIC DEVICES	
2.3 ANALOG AND DIGITAL DATA	
3 . IOT DEVICES	**9**
3.1 SENSORS	
3.2 ACUTATORS	
4 . MICROCONTROLLERS	**13**
4.1 ARDUINO	
4.2 NODEMCU(ESP8266)	
4.3 RASPBERRY PI	
5 . OBJECT DETECTION AND DISTNACE MONITIORING USING RASPBERRY PI	**27**

TOPIC	**PAGE NO**
6 . HARDWARE COMPONENTS	28
7 . SOFTWARE REQUIREMENTS	29
8 . PROJECT SETUP	30
9 . CODE	31
10 . OUTPUT	39
11 . ADVANTAGES	42
12 . APPLICATIONS	42
13 . FUTURE SCOPE	43
14 . CONCLUSION	44

1. INTERNET OF THINGS(IOT)

1.1 Introduction:

IoT is network of interconnected computing devices which are embedded in everyday objects, enabling them to send and receive data.

The Internet of Things, is a network of physical devices. These devices can transfer data to one another without human intervention. IoT devices are not limited to computers or machinery. The Internet of Things can include anything with a sensor assigned a unique identifier (UID). The primary goal of the Internet of Things is to create self-reporting devices that can communicate with each other (and users) in real time.

Four Key Components of IOT:

- Device or sensor
- Connectivity
- Data processing
- Interface

1.2 IoT Components:

1. **Sensors and Actuators**:
    - **Sensors**: Devices that detect and measure changes in the environment, such as temperature, humidity, light, motion, and more.
    - **Actuators**: Components that perform actions based on the data received from sensors, like opening a valve, turning on a motor, or adjusting lighting.
2. **Connectivity**:
    - **Wireless Technologies**: Wi-Fi, Bluetooth, Zigbee, LoRaWAN, and cellular (3G, 4G, 5G).
    - **Wired Technologies**: Ethernet, Modbus.
3. **Edge Devices**:
    - **Microcontrollers**: Low-power devices like Arduino, ESP8266, and ESP32.

- **Single-board Computers**: More powerful devices like Raspberry Pi, BeagleBone.

4. **Gateways**:

- Bridge the communication between IoT devices and the cloud, often providing data aggregation, preprocessing, and protocol translation.

5. **Cloud Platforms**:

- **Data Storage**: For collecting and storing large amounts of IoT data.
- **Data Analytics**: Tools for analyzing data to extract meaningful insights.
- **Machine Learning**: For creating predictive models and automating decision-making.

6. **User Interfaces**:

- **Mobile Apps**: For real-time monitoring and control.
- **Web Dashboards**: For data visualization and management.

7. **Security**:

- **Encryption**: Ensuring data privacy during transmission.
- **Authentication**: Ensuring only authorized devices and users can access the network.

1.3 IoT Applications

1. **Smart Home**:

- **Home Automation**: Control of lighting, heating, air conditioning, and appliances.
- **Security Systems**: Smart locks, cameras, and alarm systems.
- **Energy Management**: Monitoring and optimizing energy usage.

2. **Industrial IoT (IIoT)**:

- **Predictive Maintenance**: Monitoring machinery to predict failures before they happen.
- **Supply Chain Management**: Tracking inventory and optimizing logistics.
- **Industrial Automation**: Robotics and automated systems for manufacturing processes.

3. **Healthcare**:

- **Wearable Devices**: Monitoring vital signs like heart rate, blood pressure, and oxygen levels.
- **Remote Patient Monitoring**: Allowing doctors to monitor patients outside the hospital.
- **Smart Medical Devices**: Connected inhalers, insulin pumps, and other devices for personalized healthcare.

4. **Agriculture**:

- **Precision Farming**: Monitoring soil conditions, weather, and crop health.
- **Livestock Monitoring**: Tracking the health and location of animals.
- **Smart Irrigation**: Optimizing water usage based on soil moisture and weather forecasts.

5. **Smart Cities**:

- **Traffic Management**: Real-time monitoring and control of traffic flow.
- **Public Safety**: Connected surveillance systems and emergency response.
- **Environmental Monitoring**: Tracking air and water quality, noise levels, and other environmental parameters.

6. **Retail**:

- **Inventory Management**: Real-time tracking of stock levels and automated restocking.
- **Customer Experience**: Personalized promotions and smart checkouts.
- **Supply Chain Optimization**: Enhancing logistics and distribution processes.

7. **Transportation and Logistics**:

- **Fleet Management**: Monitoring and optimizing vehicle usage and maintenance.
- **Asset Tracking**: Real-time tracking of goods and shipments.
- **Autonomous Vehicles**: Self-driving cars and drones for delivery and transportation.

The IoT ecosystem is vast and continuously evolving, with new technologies and applications emerging regularly, driving innovation and efficiency across various sectors.

2. BASICS OF ELECTRONICS

Electronics refers to the branch of technology that deals with the design, development, and application of devices and systems involving the controlled flow of electrons through various components and circuits. This encompasses the creation and integration of hardware necessary for IoT devices to sense, process, and communicate data.

2.1 Components of Electronics

Components:

- **Sensors:** Detect and measure physical properties such as temperature, humidity, light, motion, and pressure. Examples include thermistors, accelerometers, and photodiodes.
- **Actuators:** Perform actions based on data received from sensors or commands from a controller. Examples include motors, relays, and solenoids.
- **Microcontrollers and Microprocessors:** Serve as the brains of IoT devices, processing data from sensors and executing programmed instructions. Examples include Arduino, Raspberry Pi, ESP8266, and ESP32.
- **Communication Modules:** Enable wireless communication between IoT devices and other systems. Examples include Wi-Fi (ESP8266, ESP32), Bluetooth (HC-05, HC-06), and Zigbee (XBee).

Circuit Design:

- **Printed Circuit Boards (PCBs):** Provide mechanical support and electrical connections for electronic components. PCBs are designed to fit specific IoT applications, ensuring efficient layout and connectivity.
- **Prototyping Tools:** Such as breadboards and jumper wires, are used for testing and iterating on circuit designs before finalizing them on a PCB.

Power Supply:

- **Battery Management:** Essential for portable IoT devices, involving rechargeable batteries and power optimization techniques.
- **Voltage Regulation:** Ensures that electronic components receive a stable and appropriate voltage level to function correctly.

Microcontroller Programming:

- **Embedded Programming:** Writing software to control microcontrollers and other embedded systems. Common programming languages include C, C++, and Python.
- **Development Environments:** Tools used for writing, compiling, and debugging code. Examples include the Arduino IDE, PlatformIO, and MicroPython.

2.2 Electronic Devices

Electronic devices related to IoT (Internet of Things) play crucial roles in enabling the functionality of IoT systems. Here are some key categories of electronic devices commonly used in IoT applications:

1. Sensors:

- **Temperature Sensors:** Measure temperature (e.g., DHT11, DS18B20).
- **Humidity Sensors:** Measure moisture levels (e.g., DHT22).
- **Pressure Sensors:** Measure atmospheric pressure (e.g., BMP180, BMP280).
- **Proximity Sensors:** Detect the presence of nearby objects (e.g., ultrasonic sensors, infrared sensors).
- **Light Sensors:** Measure light intensity (e.g., LDR, photodiodes).
- **Motion Sensors:** Detect movement (e.g., PIR sensors, accelerometers like MPU6050).
- **Gas Sensors:** Detect gas concentrations (e.g., MQ-2 for smoke, MQ-135 for air quality).

2. Actuators:

- **Motors:** Convert electrical energy into mechanical motion (e.g., DC motors, servo motors, stepper motors).
- **Relays:** Electrically operated switches used to control high-power devices.
- **Solenoids:** Electromagnetic devices used to convert electrical energy into linear motion.

- **LEDs and Displays:** Visual output devices (e.g., LEDs, OLED displays, LCDs).

3. Microcontrollers and Single-Board Computers:

- **Arduino:** Open-source microcontroller platforms for prototyping (e.g., Arduino Uno, Arduino Nano).
- **ESP8266/ESP32:** Microcontrollers with built-in Wi-Fi and Bluetooth capabilities, popular for IoT projects.
- **Raspberry Pi:** Small single-board computers for more complex IoT applications requiring higher processing power.
- **STM32:** Microcontroller series for advanced embedded systems.

4. Communication Modules:

- **Wi-Fi Modules:** Enable wireless networking (e.g., ESP8266, ESP32).
- **Bluetooth Modules:** Enable short-range wireless communication (e.g., HC-05, HC-06, Bluetooth Low Energy modules).
- **Zigbee Modules:** Low-power wireless communication for mesh networks (e.g., XBee).
- **LoRa Modules:** Long-range, low-power communication for IoT applications (e.g., SX1278).
- **Cellular Modules:** Enable mobile network communication (e.g., GSM/GPRS modules like SIM800, NB-IoT modules).

5. Power Supply and Management:

- **Batteries:** Provide portable power sources (e.g., Li-ion, Li-Po batteries).
- **Voltage Regulators:** Ensure stable voltage supply to electronic components (e.g., LM7805, AMS1117).
- **Power Management ICs:** Manage battery charging, power distribution, and power-saving modes.

6. Storage Devices:

- **EEPROM:** Non-volatile memory for storing small amounts of data.
- **SD Cards:** Provide larger storage capacity for logging data in IoT applications.

7. Prototyping Tools:

- **Breadboards:** For building and testing circuits without soldering.
- **PCBs (Printed Circuit Boards):** Custom boards for assembling and integrating IoT devices.

8. Development and Debugging Tools:

- **USB to Serial Converters:** Facilitate communication between microcontrollers and computers (e.g., FT232RL, CP2102).
- **JTAG/SWD Debuggers:** For debugging and programming microcontrollers (e.g., ST-Link, J-Link).

9. Gateways and Routers:

- **IoT Gateways:** Bridge different communication protocols and manage data flow between IoT devices and the cloud (e.g., Raspberry Pi-based gateways, industrial IoT gateways).
- **Routers:** Provide internet connectivity and manage local networks for IoT devices.

2.3 Analog and Digital Data:

Analog Data:

- Analog data is represented by continuous signals that can take any value within a given range. These signals often correspond directly to physical quantities and vary smoothly over time.

Analog Input:

- An analog input reads a continuous range of values from an external sensor or device. These values are usually converted from a voltage level to a digital value using an Analog-to-Digital Converter (ADC).

Analog Output:

- An analog output generates a continuous range of values. However, many microcontrollers do not have true analog output but use Pulse Width Modulation (PWM) to simulate it. Digital-to-Analog Converters (DACs) are used for true analog output.

Digital Data:

- Digital data is represented by discrete values, typically using binary code (0s and 1s). Digital signals are not continuous and change in distinct steps.

Digital Input:

- A digital input reads a binary state from an external device or sensor. It can recognize only two states: HIGH (1) or LOW (0).

Digital Output:

- A digital output sends a binary signal to control external devices. It can output either a HIGH (1) or LOW (0) state.

3. IOT DEVICES

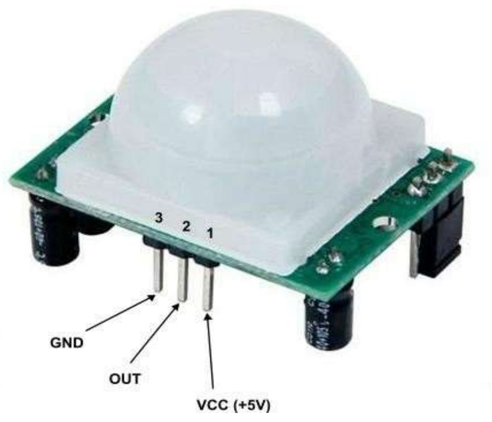
3.1 Sensors:

A sensor is a device that detects and measures a physical property or environmental condition and converts this data into a signal that can be read or interpreted by an observer or by an electronic instrument.

Examples of sensors

PIR SENSOR:

PIR sensor can detect animal/human movement in a requirement range. PIR is made of a pyroelectric sensor, which is able to detect different levels of infrared radiation. The detector itself does not emit any energy but passively receives it.

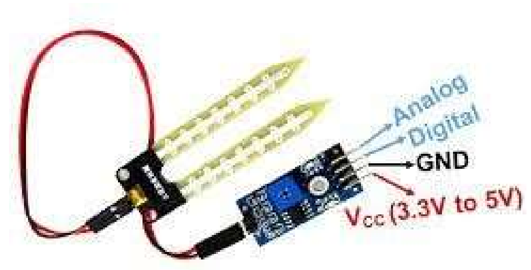


ULTRASONIC SENSOR:

Ultrasonic sensors are electronic devices that determine a target's distance. They work by emitting ultrasonic sound waves and converting those waves into electrical signals. It works by sending sound waves from the transmitter, which then bounce off of an object and then return to the receiver. You can determine how far away something is by the time it takes for the sound waves to get back to the sensor.

SOIL MOISTURE SENSOR:

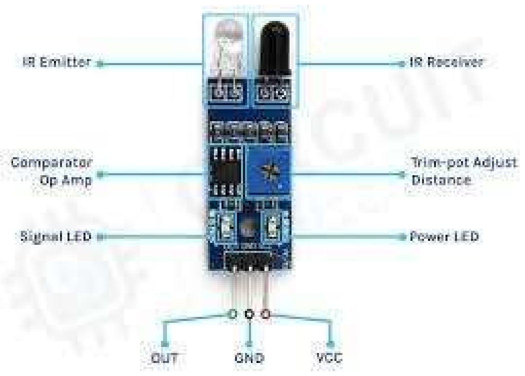
The soil moisture sensor is one kind of sensor used to gauge the volumetric content of water within the soil. As the straight gravimetric dimension of soil moisture needs eliminating, drying, as well as sample weighting. These sensors measure the volumetric water content not directly with the help of some other rules of soil like dielectric constant, electrical resistance, otherwise interaction with neutrons, and replacement of the moisture content.



IR SENSOR:

IR sensors detect infrared radiation emitted by objects in their vicinity. When an object emits heat, it generates infrared radiation that the sensor can pick up. This detection process allows the sensor to identify the presence or absence of an object based on its heat signature.

IR sensors consist of a transmitter and a receiver. The transmitter emits infrared light, which bounces off nearby objects and gets reflected back to the receiver. By measuring the time it takes for the signal to return, the sensor can determine proximity and detect motion.



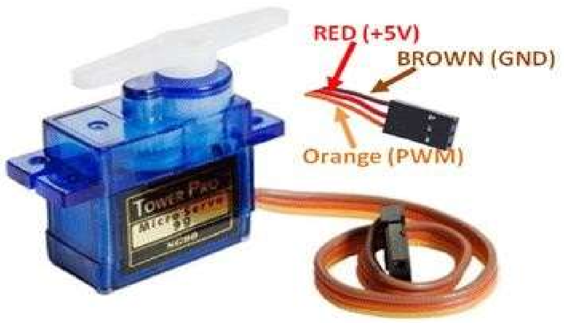
3.2 Actuators:

An Actuator is a device used to move or control a body or mechanism in a linear or rotational direction by applying a control signal. Actuators are present in almost every machine around us, from simple electronic access control systems, the vibrator on your mobile phone and household appliances to vehicles, industrial devices, and robots.

Examples of Actuators

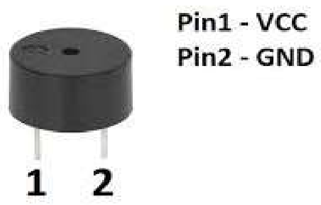
SERVO MOTOR:

A **servo motor** is a type of motor that can rotate with great precision. Normally this type of motor consists of a control circuit that provides feedback on the current position of the motor shaft, this feedback allows the servo motors to rotate with great precision. If you want to rotate an object at some specific angles or distance, then you use a servo motor.



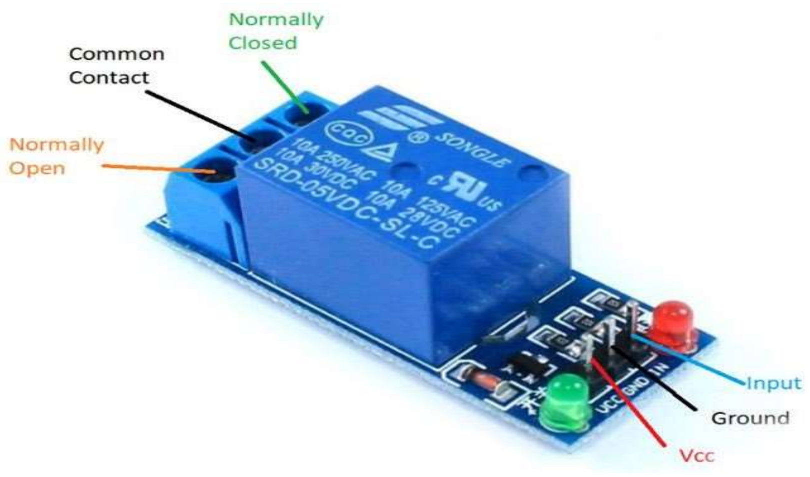
THE BUZZER:

The piezo, also known as the buzzer, is a component that is used for generating sound. It is a digital component that can be connected to digital outputs, and emits a tone when the output is HIGH.



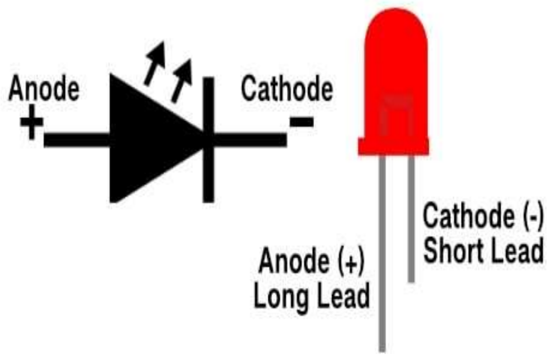
RELAYS:

Relay is an electromechanical device that uses an electric current to open or close the contacts of a switch. The single-channel relay module is much more than just a plain relay, it comprises of components that make switching and connection easier and act as indicators to show if the module is powered and if the relay is active or not.



LED:

LEDs (Light Emitting Diodes) are small lights made from a semiconductor material and metal wire. They light up when electrical current passes through them. LEDs can be different colors that depend on the energy it gives off and how far electrons cross a gap inside the LED. Because LEDs have a positive end and a negative end, they have polarity and will only light up in a circuit when placed in the direction of electrical current flow.



4. MICROCONTROLLERS

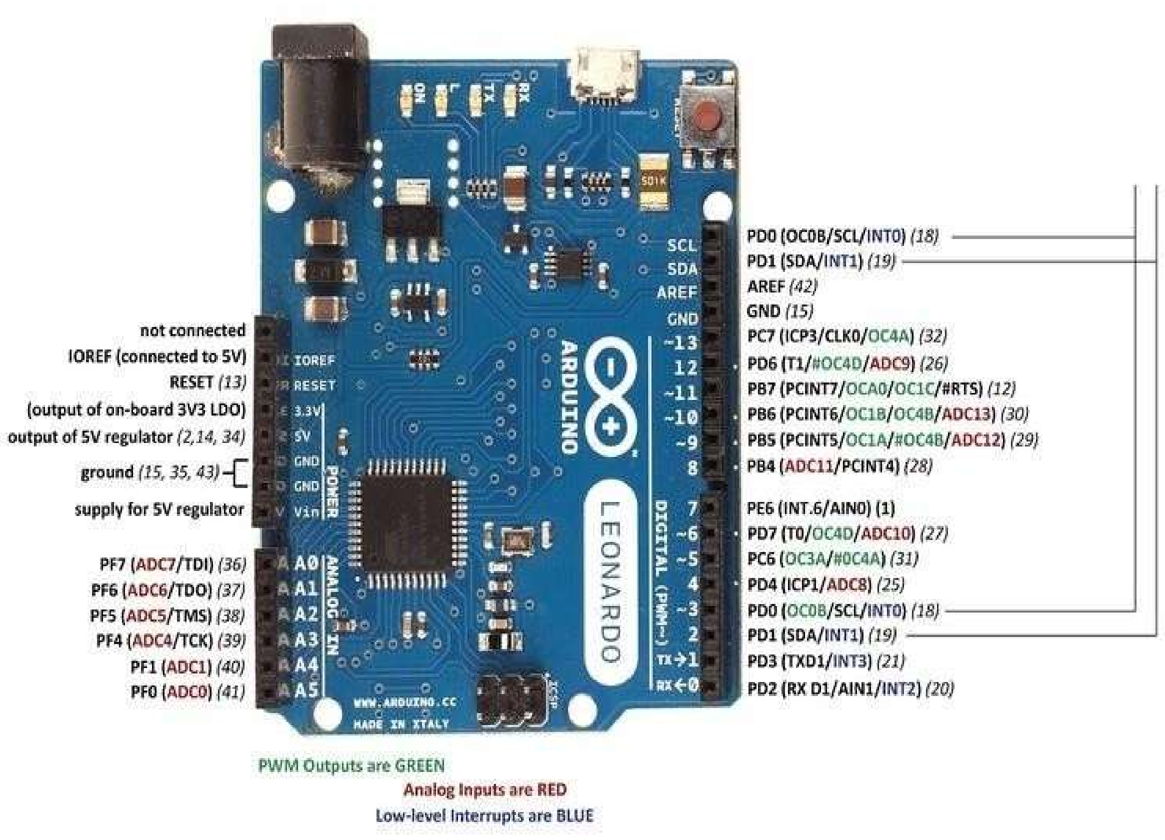
4.1 ARDUINO:

Arduino boards are open-source microcontroller boards that provide a simple platform for building electronic projects.

Arduino boards are versatile tools that bridge the gap between software and hardware, allowing users to create a wide range of interactive projects. Their open-source nature and extensive community support make them an excellent choice for both beginners and experienced developers. Here, we'll cover the general features and different types of Arduino boards, with a focus on the popular Arduino Uno.

ARDUINO UNO:

Arduino UNO is a microcontroller board based on the **ATmega328P**. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.



Arduino IDE:

The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, and Linux) that is written in the programming language C or C++. It is used to write and upload programs to Arduino-compatible boards.

Key Features:

- Sketch: The name given to Arduino programs. It is written in C/C++.
- Serial Monitor: A feature that allows you to communicate with your board via serial communication.
- Library Manager: Enables easy inclusion of various libraries for extended functionalities.
- Board Manager: Allows you to install support for various types of Arduino boards.
- Integrated Editor: Simple and easy-to-use text editor for writing code.

Interfacing:

Interfacing involves connecting the Arduino board to various sensors, actuators, and other devices to interact with the physical world. This can include:

- Digital Read/Write: Interacting with digital sensors and devices (e.g., switches, LEDs).
- Analog Read/Write: Interacting with analog sensors (e.g., temperature sensors, potentiometers).
- PWM (Pulse Width Modulation): Controlling devices that require variable power (e.g., motors, LED brightness).
- Serial Communication: Sending and receiving data from other devices like computers, other Arduinos, or communication modules (e.g., Bluetooth, WiFi).

Structure of an Arduino Sketch:

1. setup() function: This runs once when the Arduino is powered on or reset. It’s used to initialize variables, pin modes, start using libraries, etc.

2. loop() function: This runs continuously after the setup() function. It's used to actively control the Arduino board.

```
void setup() {
  // put your setup code here, to run once:


}


void loop() {
  // put your main code here, to run repeatedly:


}
```

INTERFACING OF ARDUINO UNO WITH PIR MOTION SENSOR:

❖ The Components Used are:-

- Arduino Uno
- PIR Motion Sensor
- LED
- Jumper Wires

❖ Pin Configuration:

The PIR motion sensor consists of 3 pins-VCC, GND, and the Data Output Pin.

1.Connect the VCC pin of the PIR sensor to the 5V pin on the Arduino.

2. Connect the GND pin of the PIR sensor to the GND pin on the Arduino.

3. Connect the OUT pin of the PIR sensor to a digital input pin on the Arduino (e.g., pin 5).

4.connect an LED to pin 12.

❖ <u>Algorithm:</u>

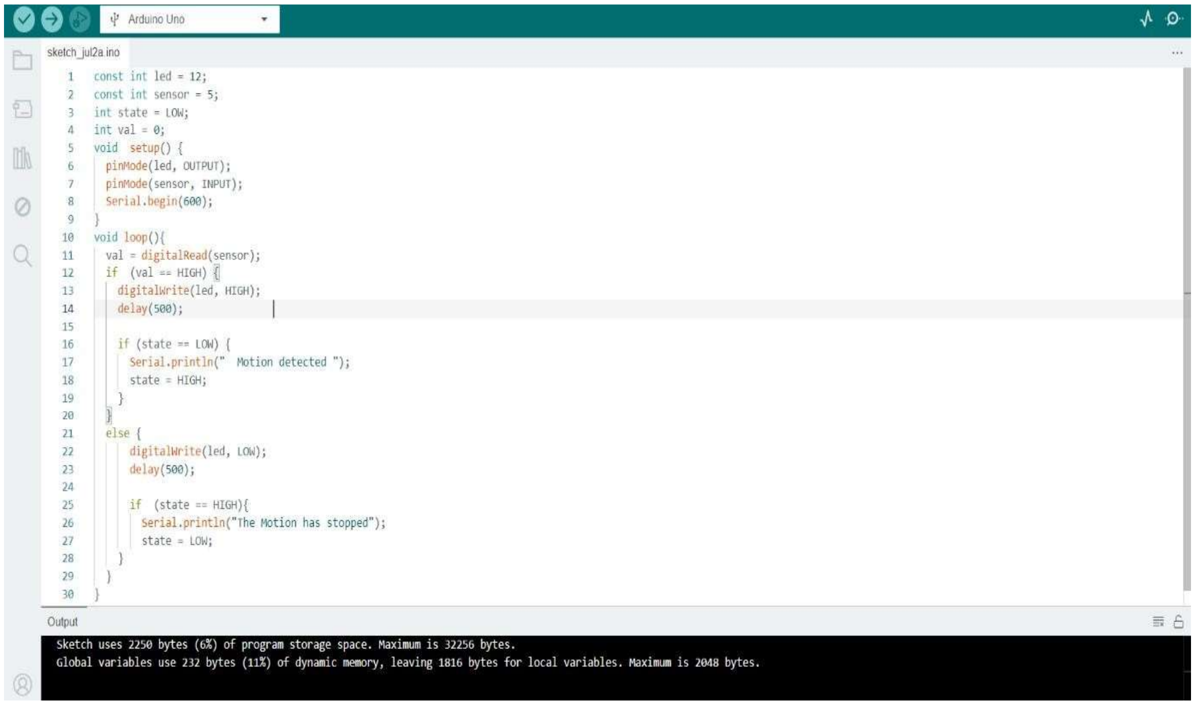
1. Setup

- Initialize the PIR sensor pin as input.
- Initialize the LED pin as output.
- Set up serial communication

2. Loop

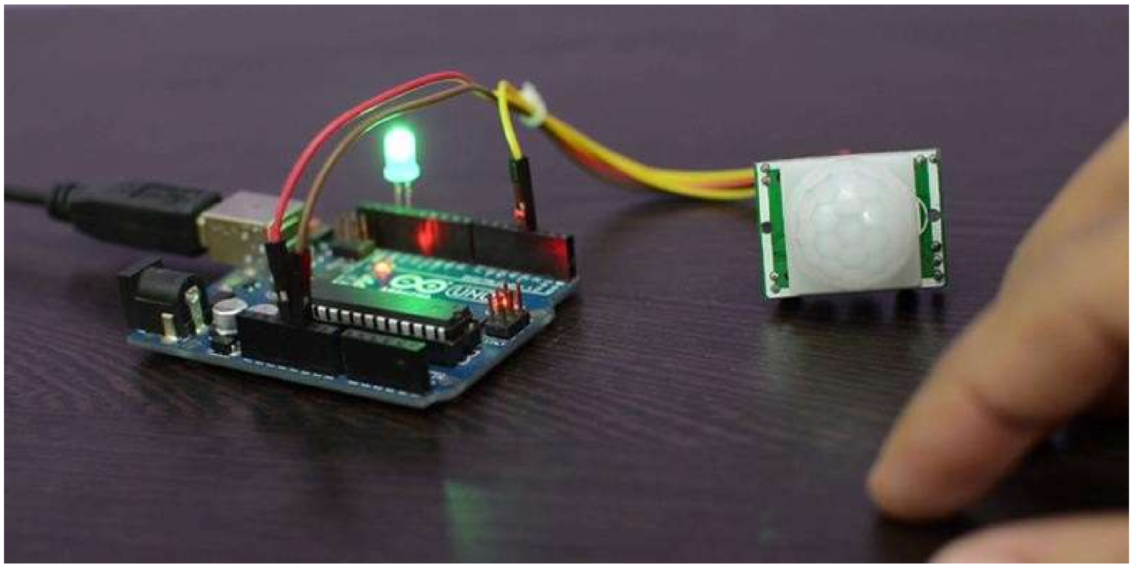
- Read the PIR sensor value.
- If motion is detected (sensor output is HIGH), turn on the LED and print a message to the serial monitor.
- If no motion is detected (sensor output is LOW), turn off the LED and print a message to the serial monitor.
- Add a small delay to stabilize readings.

❖ <u>Code and Output:</u>



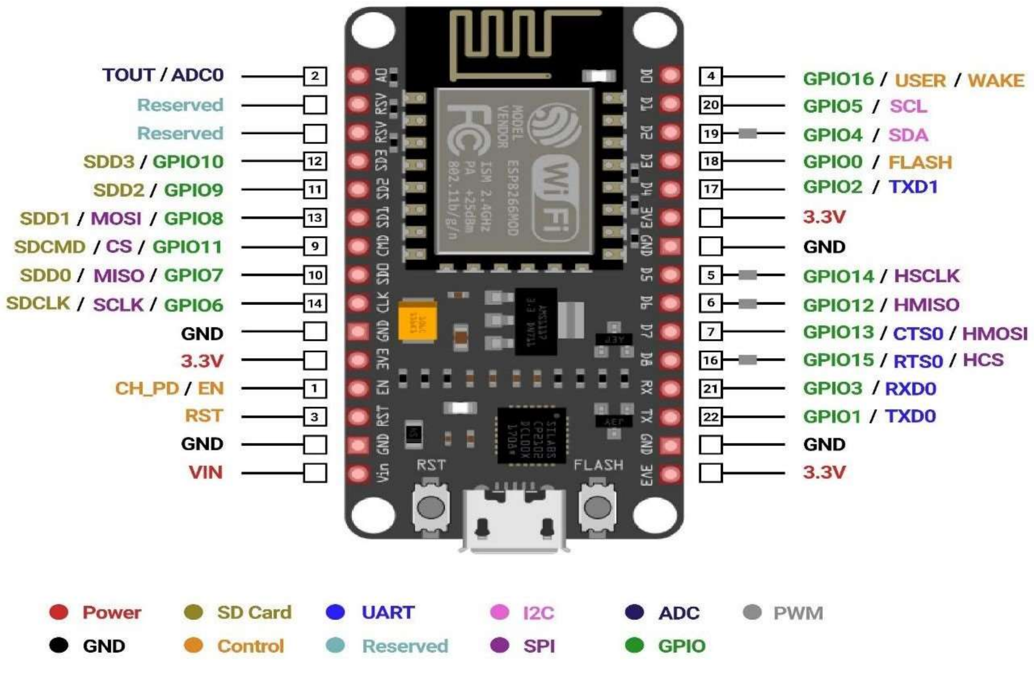
```
const int led = 12;
const int sensor = 5;
int state = LOW;
int val = 0;
void  setup() {
  pinMode(led, OUTPUT);
  pinMode(sensor, INPUT);
  Serial.begin(600);
}
void loop(){
  val = digitalRead(sensor);
  if  (val == HIGH) {
    digitalWrite(led, HIGH);
    delay(500);

    if (state == LOW) {
      Serial.println("  Motion detected ");
      state = HIGH;
    }
  }
  else {
      digitalWrite(led, LOW);
      delay(500);

      if  (state == HIGH){
        Serial.println("The Motion has stopped");
        state = LOW;
    }
  }
}
```

```
Sketch uses 2250 bytes (6%) of program storage space. Maximum is 32256 bytes.
Global variables use 232 bytes (11%) of dynamic memory, leaving 1816 bytes for local variables. Maximum is 2048 bytes.
```



4.2 NODEMCU(ESP8266):

The NodeMCU is a low-cost, open-source IoT platform based on the ESP8266 WiFi module. It is widely used for IoT projects due to its built-in WiFi capabilities and ease of use.The name "NodeMCU" combines "node" and "MCU"(Micro controller unit).



Once Arduino IDE is installed on the computer, connect the board with the computer using the USB cable. Now open the Arduino IDE and choose the correct board by selecting **Tools>Boards>NodeMCU1.0** (ESP-12E Module), and choose the correct Port by selecting **Tools>Port**.

To work with ESP8266 wirelessly and work in mobile we use blynk app in mobile and computer.

<u>INTERFACING OF ESP8266 WITH IR SENSOR:</u>

❖ <u>Components Needed:</u>

1. ESP8266 (NodeMCU)
2. IR sensor
3. Jumper wires

❖ <u>Pin Configuration:</u>

1. Connect the VCC Pin of the IR sensor to the 3.3V Pin of the ESP8266 (or 5V if your sensor operates at 5V, but be cautious with the ESP8266's 5V tolerance).
2. Connect the GND Pin of the IR sensor to the GND Pin of the ESP8266.
3. Connect the OUT Pin of the IR sensor to an Analog Input Pin on the ESP8266 (e.g., A0).

❖ <u>Algorithm:</u>

Initialization:

- Include <ESP8266WiFi.h> library
- Define irSensor as A0

Setup:

- Begin serial communication with Serial.begin(9600)
- Set pinMode of irSensor to OUTPUT

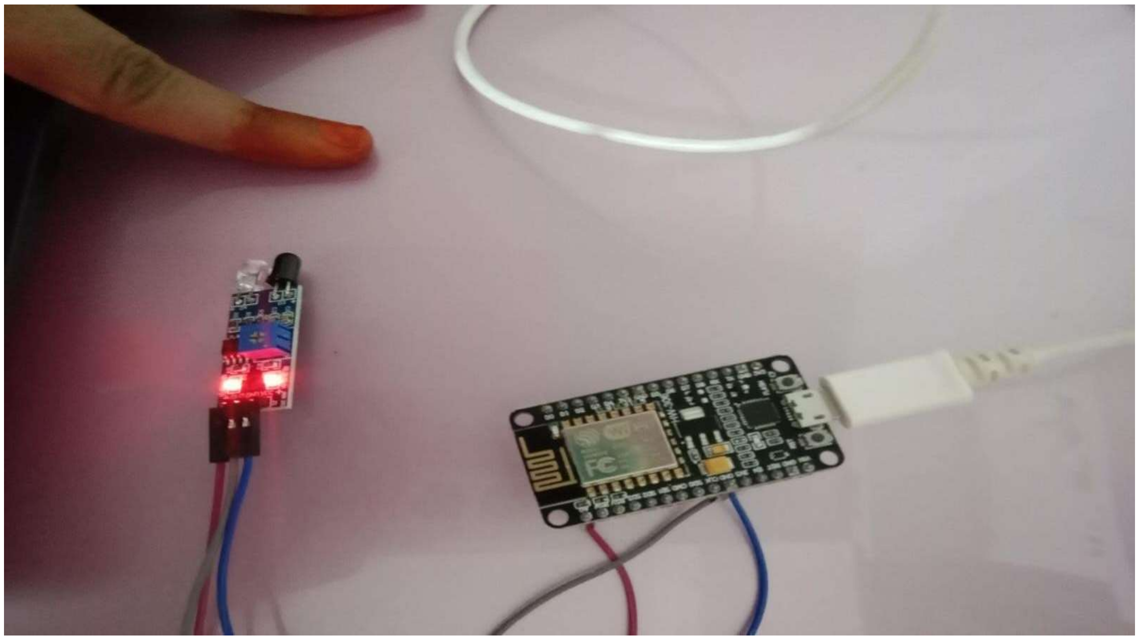
Main Loop:

- Use digitalRead(irSensorPin) to read the digital state from the sensor.
- If the sensor state is HIGH, print IR state.

- Add a delay of 1 second to make the output readable and avoid flooding the Serial Monitor.

❖ Code and Output:

```
#include <ESP8266WiFi.h> // Include the ESP8266 WiFi library

#include <ESP8266WebServer.h> // Include the ESP8266 Web Server library int
IR = A0;


// the setup routine runs once when you press reset:

void setup() {

  // initialize serial communication at 9600 bits per second:

  Serial.begin(9600);

  // make the pushbutton's pin an input:

  pinMode(IR, INPUT);

}


// the loop routine runs over and over again forever:

void loop() {

  // read the input pin:

  int IRState = digitalRead(IR);

  Serial.println(IRState);

  delay(1000);        // delay in between reads for stability

}

// Delay for readability

}
```



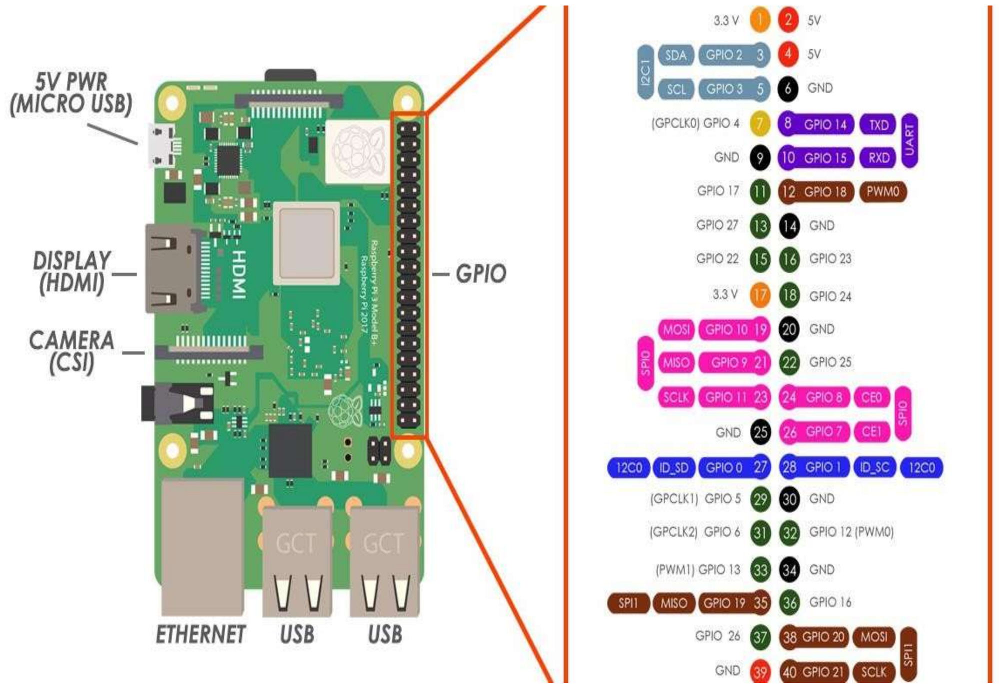
4.3 RASPBERRY PI:

Raspberry Pi is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The original model became more popular than anticipated, selling outside its target market for diverse uses such as robotics, home and industrial automation, and by computer and electronic hobbyists, because of its low cost, modularity, open design, and its adoption of the HDMI and USB standards.

Pin configuration of Raspberry pi:

Raspberry Pi B+, 2, Zero, 3, 4, or 5, you will have 40 pins in total. The super early models, such as the Raspberry Pi B, have just 26 pins.

- **GPIO** is your standard pins that can be used to turn devices on and off, such as an LED.
- **I2C** (Inter-Integrated Circuit) pins allow you to connect to and talk to hardware modules that support this protocol (I2C Protocol). This protocol typically takes up two pins.
- **SPI** (Serial Peripheral Interface Bus) pins can connect and talk to SPI devices. They are similar to I2C but makes use of a different protocol.
- **UART** (Universal Asynchronous Receiver/Transmitter) is the serial pins used to communicate with other devices.
- **DNC** stands for do not connect, this is pretty self-explanatory.
- The **power pins** pull power directly from the Raspberry Pi.
- **GND** are the pins you use to ground your devices. It does not matter which pin you use, as they are all connected to the same line.



5V PWR (MICRO USB)
DISPLAY (HDMI)
CAMERA (CSI)
ETHERNET
USB
USB
GPIO
HDMI
GCT
3.3 V 1 2 5V
I2C1 SDA GPIO 2 3 4 5V
SCL GPIO 3 5 6 GND
(GPCLK0) GPIO 4 7 8 GPIO 14 TXD UART
GND 9 10 GPIO 15 RXD
GPIO 17 11 12 GPIO 18 PWM0
GPIO 27 13 14 GND
GPIO 22 15 16 GPIO 23
3.3 V 17 18 GPIO 24
SPI0 MOSI GPIO 10 19 20 GND
MISO GPIO 9 21 22 GPIO 25
SCLK GPIO 11 23 24 GPIO 8 CE0 SPI0
GND 25 26 GPIO 7 CE1
I2C0 ID_SD GPIO 0 27 28 GPIO 1 ID_SC I2C0
(GPCLK1) GPIO 5 29 30 GND
(GPCLK2) GPIO 6 31 32 GPIO 12 (PWM0)
(PWM1) GPIO 13 33 34 GND
SPI1 MISO GPIO 19 35 36 GPIO 16
GPIO 26 37 38 GPIO 20 MOSI SPI1
GND 39 40 GPIO 21 SCLK

Setting up Raspberry pi:

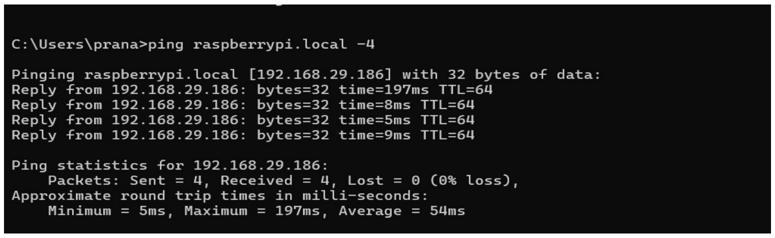
Raspberry Pi comes without any peripheral devices. The first thing to do is to unpack Rasp Pi and protect it with an enclosure. Raspberry Pi can be installed to the protective enclosure without using any tools. The enclosure has plastic clips which are holding the Raspberry Pi in its place. After Raspberry Pi has been installed to enclosure and well protected, all the necessary peripherals can be attached to it. Just like any other computer, Raspberry Pi needs some basic devices such as display which is connected via the HDMI cable, the mouse and the keyboard, and the internet connection cable.

Raspberry Pi OS is a Unix-like operating system based on the Debian Linux distribution for the Raspberry Pi family of compact single-board computers. Raspbian was developed independently in 2012, became the primary operating system for these boards since 2013, was originally optimized for the Raspberry Pi 1 and distributed by the Raspberry Pi Foundation.

Before plugging the power cable, MicroSD-card should be checked if it is flashed and prepared with an operating system. While installing the OS Edit settings as per required and cross check that details thoroughly.

Also it is recommendable to create a backup folder of the MicroSD-card just in case of complications. The MicroSD-card can be checked with a card-reader. The card-reader can be found from most of the laptops and desktop computers. Insert the MicroSD-card into the card-reader and check that there is something stored in the MicroSD-card.Create an empty file without any extension i.e.ssh in microSD card driver when connected to laptop/desktop. If everything looks good, take the MicroSD-card and plug it into the Raspberry Pi. Now the power cable can be connected. Raspberry Pi does not have any kind of power switch so it will start up immediately when the power cable is connected to it.

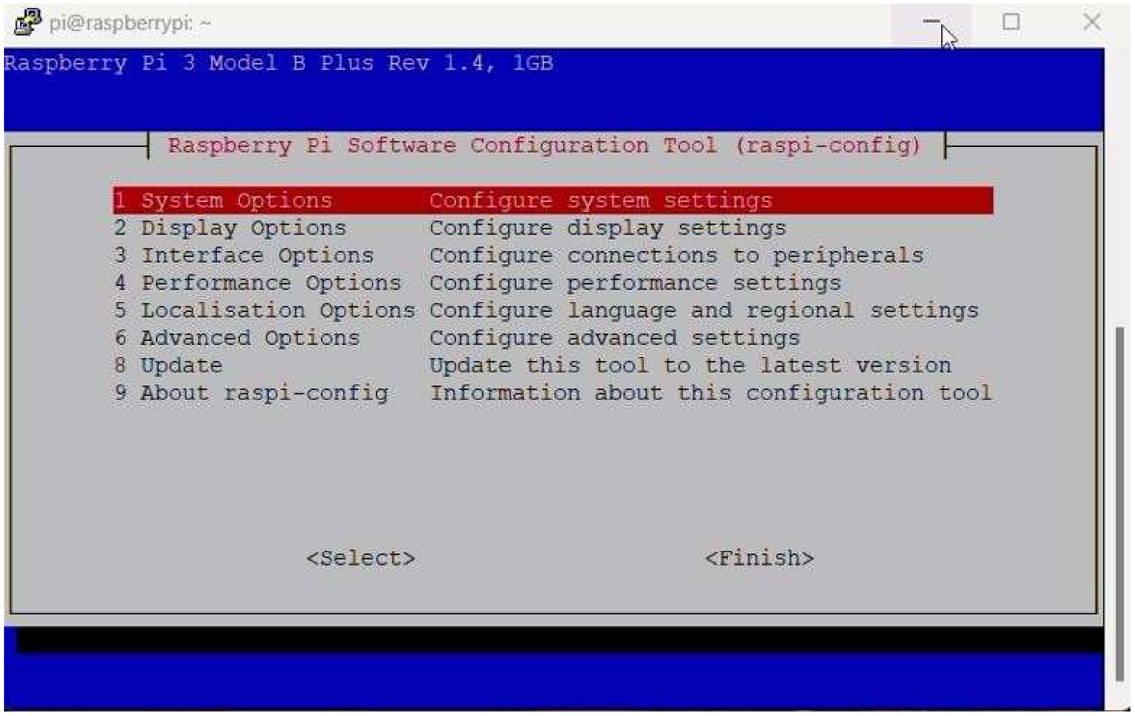
Check whether the Raspberrypi IP Address is found using Command Prompt. Use ping tool to get ip address.



```
C:\Users\prana>ping raspberrypi.local -4

Pinging raspberrypi.local [192.168.29.186] with 32 bytes of data:
Reply from 192.168.29.186: bytes=32 time=197ms TTL=64
Reply from 192.168.29.186: bytes=32 time=8ms TTL=64
Reply from 192.168.29.186: bytes=32 time=5ms TTL=64
Reply from 192.168.29.186: bytes=32 time=9ms TTL=64

Ping statistics for 192.168.29.186:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 5ms, Maximum = 197ms, Average = 54ms
```

If we are connected to desktop or Display directly Raspberrypi OS will be connected and We can run it without any other external environment apps.

If we are connected to laptop then we need to install puTTy app and RealVNC. Putty app will help to connected with raspberrypi without any network issues. Hostname of raspberrypi connected will be “pi”,and set password ,this is setup when installed by the host.Now once logged into raspberrypi using puTTY,use “sudo raspi-config” to set reset configuration settings of raspberrypi and connect with virtual environment . Now go to raspi-config and go to Inerfacing Options->VNC.Enable VNC server here.



```
Raspberry Pi 3 Model B Plus Rev 1.4, 1GB

      Raspberry Pi Software Configuration Tool (raspi-config)

   1 System Options       Configure system settings
   2 Display Options      Configure display settings
   3 Interface Options    Configure connections to peripherals
   4 Performance Options  Configure performance settings
   5 Localisation Options Configure language and regional settings
   6 Advanced Options     Configure advanced settings
   8 Update               Update this tool to the latest version
   9 About raspi-config   Information about this configuration tool

              <Select>                      <Finish>
```

RealVNC is a software where we can connect to other desktop using hostname or ipaddress.Once RealVNC is enabled , now enter the ipaddress or hostname to get connected with Raspberrypi.

If you are using the latest version of Raspberry Pi OS, you can start programming and using the GPIO pins without needing to do any extra configuration.

I recommend updating your Pi to the latest packages before proceeding. If you have not done this, then you can do it by running the following commands.

=>Terminal:

sudo apt update

```
sudo apt upgrade

sudo apt install rpi.gpio  //  install GPIO Package.
```

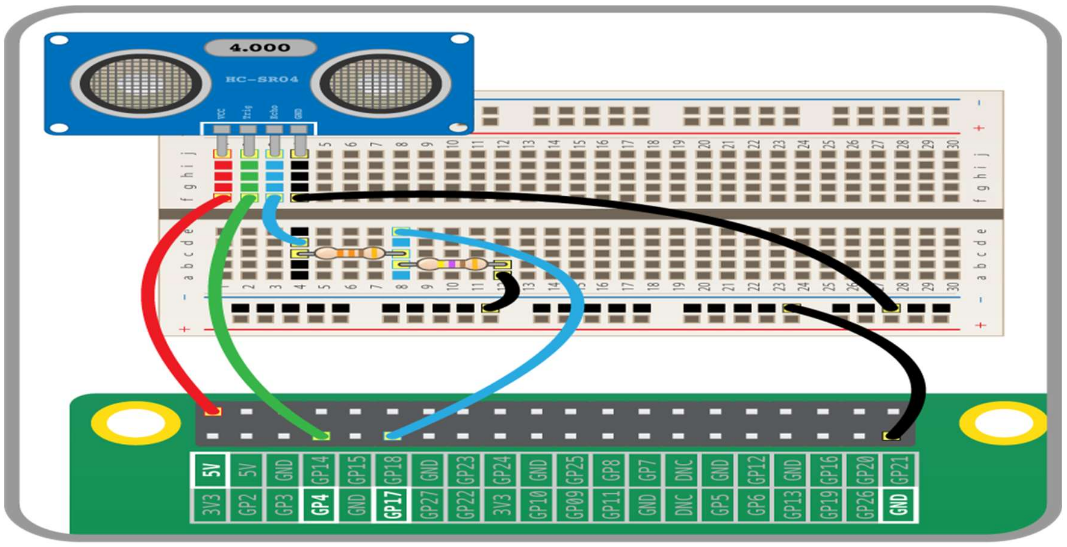
INTERFACING OF RASPBERRY PI WITH ULTRASONIC SENSOR

❖ Components used are:

- Raspberry pi with microSD card
- Ultrasonic sensor
- Breadboard
- Registers

❖ Pin configuration:

- Connect the VCC pin of Sensor to the 5V on RaspberryPi.
- Connect the GND pin of Sensor to the GND pin of RaspberryPi.
- Connect the TRIG pin of Sensor to the GPIO pin (eg.pin 14).
- Connect the ECHO pin of Sensor to the GPIO pin(eg.pin 18) use registers to control flow of current.



❖ Code and output:

Python Script:

```
import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)     # Set GPIO mode

# Define GPIO pins

GPIO_TRIGGER = 14
```

```
GPIO_ECHO = 18
# Set up the GPIO pins
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
def distance():
    # Send a 10us pulse to trigger
    GPIO.output(GPIO_TRIGGER, True)
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)
    # Wait for the echo response
    start_time = time.time()
    stop_time = time.time()
    while GPIO.input(GPIO_ECHO) == 0:
        start_time = time.time()

    while GPIO.input(GPIO_ECHO) == 1:
        stop_time = time.time()
    # Time difference between start and arrival
    time_elapsed = stop_time - start_time
    # Multiply with the speed of sound (34300 cm/s) and divide by 2
    distance = (time_elapsed * 34300) / 2
    return distance
try:
    while True:
        dist = distance()
        print(f"Measured Distance = {dist:.1f} cm")
        time.sleep(1)
except KeyboardInterrupt:
    print("Measurement stopped by User")
```

```
        GPIO.cleanup()
```

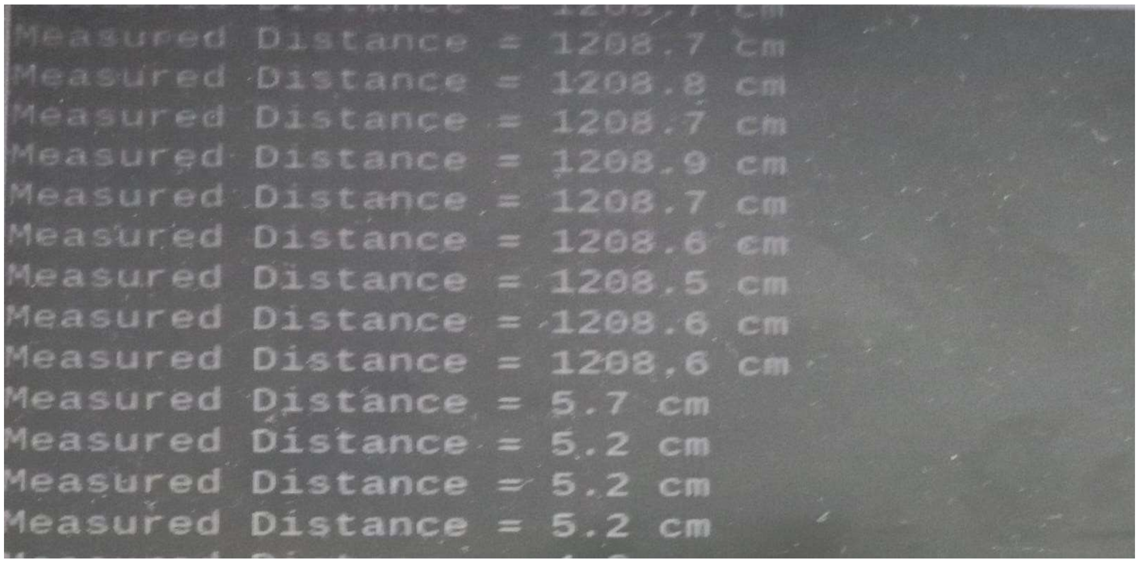
Running the Script:

1. Save the script (e.g., as ultrasonic_distance.py).
2. Run the script:

=>bash:

```
python3 ultrasonic_distance.py
```

OUTPUT:



```
Measured Distance = 1208.7 cm
Measured Distance = 1208.8 cm
Measured Distance = 1208.7 cm
Measured Distance = 1208.9 cm
Measured Distance = 1208.7 cm
Measured Distance = 1208.6 cm
Measured Distance = 1208.5 cm
Measured Distance = 1208.6 cm
Measured Distance = 1208.6 cm
Measured Distance = 5.7 cm
Measured Distance = 5.2 cm
Measured Distance = 5.2 cm
Measured Distance = 5.2 cm
```

5. OBJECT DETECTION AND DISTANCE MONITORING USING RASPBERRY PI

INTRODUCTION:

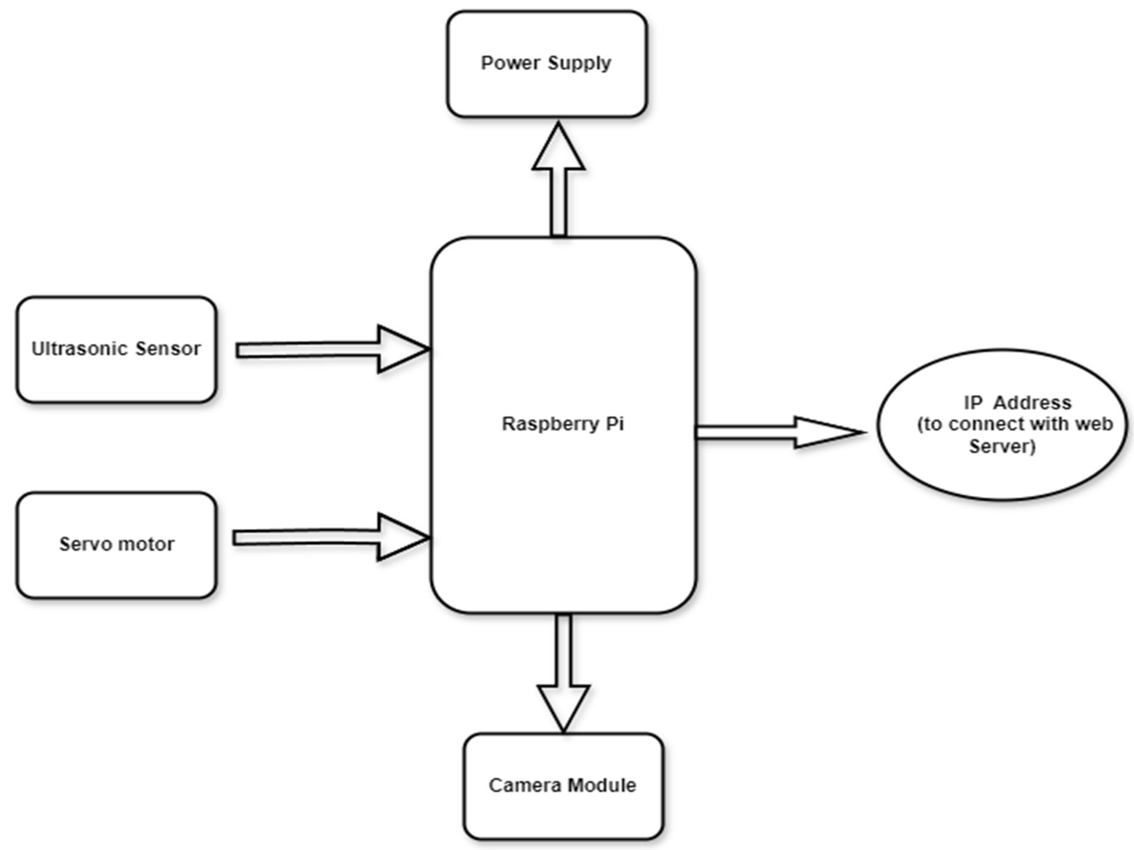
This project involves creating a real-time object detection and distance monitoring system using Raspberry Pi.This system integrates a camera,an ultrasonic sensor,and a servo motor to monitor the environment,detect objects,measure their distances,and store the data in an SQLite database.The Flask web framework is used to serve a live video feed and display distance data on a web server.

The project basically targeted to detect objects and distance of object at the boundary of country to avoid threats and react immediately.This application will live stream the camera to check it out whenever required. This works only when raspberry pi and computer are connected to same network. This project aims to develop an advanced monitoring and control system that integrates ultrasonic distance measurement, servo motor control, and real-time video streaming with object detection. Leveraging a combination of these technologies, the system provides a comprehensive solution for real-time environmental monitoring and automation.

At its core, the system uses an ultrasonic sensor to measure distances and detect the presence of nearby objects. This data is then used to control a servo motor, enabling automated responses based on distance measurements. A real-time video feed is streamed through a web interface, allowing users to monitor their environment and view detected objects directly from their devices.

6. HARDWARE COMPONENTS

Block diagram:



Hardware Requirements:

- **Raspberry Pi** (any model greater than Raspberry Pi2 with GPIO pins and Camera Port): The Raspberry Pi serves as the central processing unit, handling data from the sensors and controlling the peripherals.
- **MicroSD Card(**At least 16GB and preloaded with Raspberry Pi OS)
- **Ultrasonic sensor**(HC-SR04) : Uses the RPi.GPIO library to control the ultrasonic sensor, measuring distances by calculating the time taken for an echo to return.
- **Servo Motor Control**: The servo motor is controlled via PWM (Pulse Width Modulation) to rotate based on the distance measurements.
- **Raspberry Pi Camera Module**: Uses OpenCV to capture and process video frames. The camera is accessed via the /video_feed route in Flask, which streams the video feed.
- **Jumping wires**

7. SOFTWARE REQUIREMENTS:

- **Raspbian OS:** The operating system running on the Raspberry Pi.
- **Python 3:** The programming language used for writing the code.
- **Flask:** A lightweight WSGI web application framework for serving the web interface.

  Install using: pip install flask

- **OpenCV:** An open-source computer vision and machine learning software library for image processing and object detection.

  Install using: pip install opencv-python

- **RPi.GPIO:** A library to control the GPIO pins on the Raspberry Pi.

  Install using: pip install RPi.GPIO

- **SQLite3:** A C library that provides a lightweight, disk-based database. It is often included with Python.
- **Time:** A standard Python library to handle time-related tasks.
- **Threading:** A standard Python library to handle multithreading.
- **Camera Setup**: Ensure the Raspberry Pi camera module is enabled. This can be done using raspi-config:

  Run sudo raspi-config

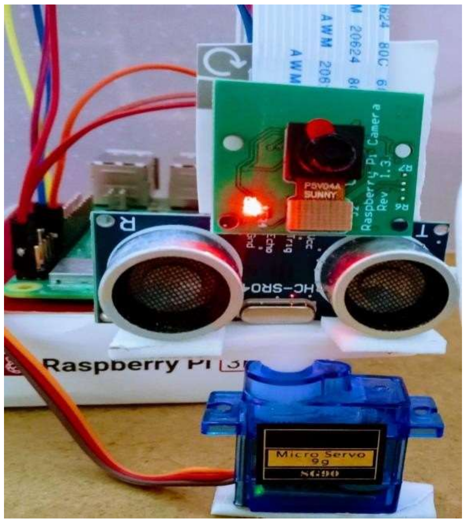
  Navigate to Interfacing Options -> Camera and enable it.

  Reboot the Raspberry Pi.

8. PROJECT SETUP:

1. Prepare the Raspberry Pi:
   - Ensure the Raspberry Pi is powered off before making any connections.
   - Attach the Camera Module to the CSI port on the Raspberry Pi.
2. The Ultrasonic Sensor:
   - Connect the VCC pin of the HC-SR04 to a 5V pin on the Raspberry Pi.
   - Connect the GND pin of the HC-SR04 to a GND pin on the Raspberry Pi.
   - Connect the Trig pin of the HC-SR04 to GPIO 23 (physical pin 16).
   - Connect the Echo pin of the HC-SR04 to GPIO 24 (physical pin 18).
3. Connect the Servo Motor:
   - Connect the VCC pin of the SG90 to a 5V pin on the Raspberry Pi.
   - Connect the GND pin of the SG90 to a GND pin on the Raspberry Pi.
   - Connect the Control pin of the SG90 to GPIO 18 (physical pin 12).
4. Power On the Raspberry Pi:

- Once all connections are secure, power on the Raspberry Pi.
- Ensure that the Camera Module is enabled by running sudo raspi-config, navigating to Interfacing Options -> Camera, and enabling it. Reboot the Raspberry Pi after enabling the camera.
- Open Terminal and install all the libraries required.



9. CODE:

Terminal:

1. **Create the project directory:**

```
mkdir object_detect
cd object_detect
```

2. **Create subdirectories for templates and static files:**

```
mkdir templates
mkdir static
```

3. **Create the Flask application file:**

```
touch app.py
```

4. **Create the HTML template:**

```
touch templates/index.html
```

5. **Python code :** Write code in app.py file

```
import cv2
from flask import Flask, render_template, Response, jsonify
import RPi.GPIO as GPIO
import time
from datetime import datetime
import sqlite3
import threading

app = Flask(__name__)

# Ultrasonic sensor setup
GPIO.setmode(GPIO.BCM)
TRIG = 23
ECHO = 24
```

```
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Servo motor setup
SERVO_PIN = 17
GPIO.setup(SERVO_PIN, GPIO.OUT)
servo = GPIO.PWM(SERVO_PIN, 50)  # 50Hz PWM frequency
servo.start(0)

# SQLite setup
DATABASE = 'distances.db'

def init_db():
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS distances (timestamp TEXT, distance REAL)''')
    conn.commit()
    conn.close()

def log_distance(distance):
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    cursor.execute('INSERT INTO distances (timestamp, distance) VALUES (?, ?)', (timestamp, distance))
    conn.commit()
    conn.close()
```

```
def get_distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)
    start_time = time.time()
    stop_time = time.time()

    while GPIO.input(ECHO) == 0:
        start_time = time.time()

    while GPIO.input(ECHO) == 1:
        stop_time = time.time()

    time_elapsed = stop_time - start_time
    distance = (time_elapsed * 34300) / 2

    log_distance(distance)  # Log distance to database
    return distance

def rotate_servo(angle):
    duty_cycle = 2 + (angle / 18)
    GPIO.output(SERVO_PIN, True)
    servo.ChangeDutyCycle(duty_cycle)
    time.sleep(1)
    GPIO.output(SERVO_PIN, False)
    servo.ChangeDutyCycle(0)
```

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video_feed')
def video_feed():
      return  Response(generate_frames(),  mimetype='multipart/x-mixed-replace;
boundary=frame')

@app.route('/distance')
def distance():
    dist = get_distance()
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    return jsonify(distance=dist, timestamp=timestamp)

@app.route('/history')
def history():
    conn = sqlite3.connect(DATABASE)
    cursor = conn.cursor()
     cursor.execute('SELECT * FROM distances ORDER BY timestamp DESC
LIMIT 10')
    rows = cursor.fetchall()
    conn.close()
    return jsonify(rows)

def generate_frames():
    camera = cv2.VideoCapture(0)
    _, first_frame = camera.read()
    first_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)
    first_gray = cv2.GaussianBlur(first_gray, (21, 21), 0)

    while True:
        success, frame = camera.read()
        if not success:
            break
        else:
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            gray = cv2.GaussianBlur(gray, (21, 21), 0)

            delta_frame = cv2.absdiff(first_gray, gray)
            thresh_frame = cv2.threshold(delta_frame, 30, 255,
cv2.THRESH_BINARY)[1]
            thresh_frame = cv2.dilate(thresh_frame, None, iterations=2)

            contours, _ = cv2.findContours(thresh_frame.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

            if len(contours) > 0:
                largest_contour = max(contours, key=cv2.contourArea)
                if cv2.contourArea(largest_contour) < 500:  # Minimum contour area
to be considered as an object
                    continue
                (x, y, w, h) = cv2.boundingRect(largest_contour)
                cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

                # Center the object in the frame
                cx = x + w // 2
                cy = y + h // 2
                fx = frame.shape[1] // 2
                fy = frame.shape[0] // 2
                dx = fx - cx
                dy = fy - cy
                M = np.float32([[1, 0, dx], [0, 1, dy]])
                frame = cv2.warpAffine(frame, M, (frame.shape[1], frame.shape[0]))

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

def update_distance():
    while True:
        distance = get_distance()
        if distance < 20:
            rotate_servo(180)
            time.sleep(1)
            rotate_servo(0)
        time.sleep(2)

if __name__ == "__main__":
    init_db()
    distance_thread = threading.Thread(target=update_distance)
    distance_thread.daemon = True
    distance_thread.start()
    try:
        app.run(host='0.0.0.0', port=5000)
    except KeyboardInterrupt:
        GPIO.cleanup()
```

6. HTML code: Write html code in index.html file

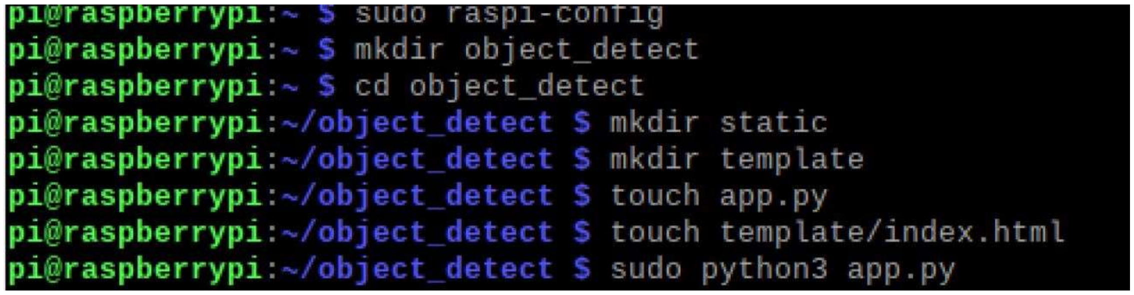
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Live Video Feed</title>
    <script>
        function updateDistance() {
            fetch('/distance')
                .then(response => response.json())
                .then(data => {
                    document.getElementById('distance').innerText = `Distance: ${data.distance.toFixed(2)} cm`;
                    document.getElementById('timestamp').innerText = `Timestamp: ${data.timestamp}`;
                });
        }

        function fetchHistory() {
            fetch('/history')
                .then(response => response.json())
            .then(data => {
                let history = document.getElementById('history');
                history.innerHTML = '';
                data.forEach(row => {
                    let listItem = document.createElement('li');
                    listItem.innerText = `Timestamp: ${row[0]}, Distance: ${row[1].toFixed(2)} cm`;
                    history.appendChild(listItem);
                });
            });
        }

        setInterval(updateDistance, 2000);  // Update distance every 2 seconds
        setInterval(fetchHistory, 10000);   // Fetch history every 10 seconds
    </script>
</head>
<body>
    <h1>Live Video Feed</h1>
    <img src="{{ url_for('video_feed') }}" alt="Video Feed">
    <p id="distance">Distance: </p>
    <p id="timestamp">Timestamp: </p>
    <h2>Distance History</h2>
    <ul id="history"></ul>
</body>
</html>
```

7. Run the Flask application:

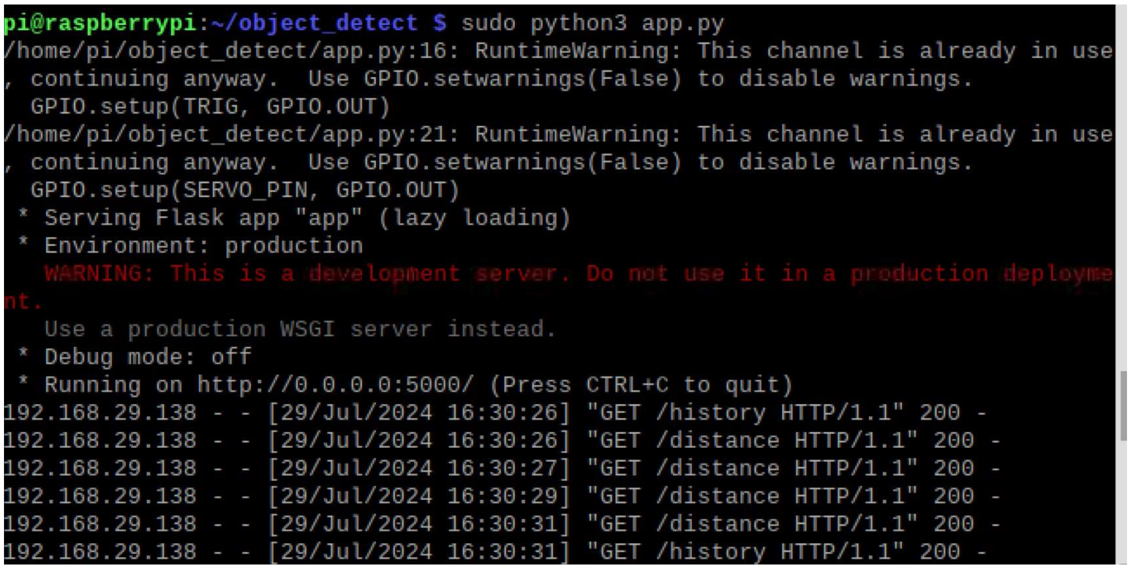
```
sudo python3 app.py
```

10. OUTPUT:

The project output provides a comprehensive system that not only detects objects and measures their distance but also streams a live video feed with object detection capabilities. The integration of these components allows for real-time monitoring and control, with a user-friendly web interface displaying both current and historical data.

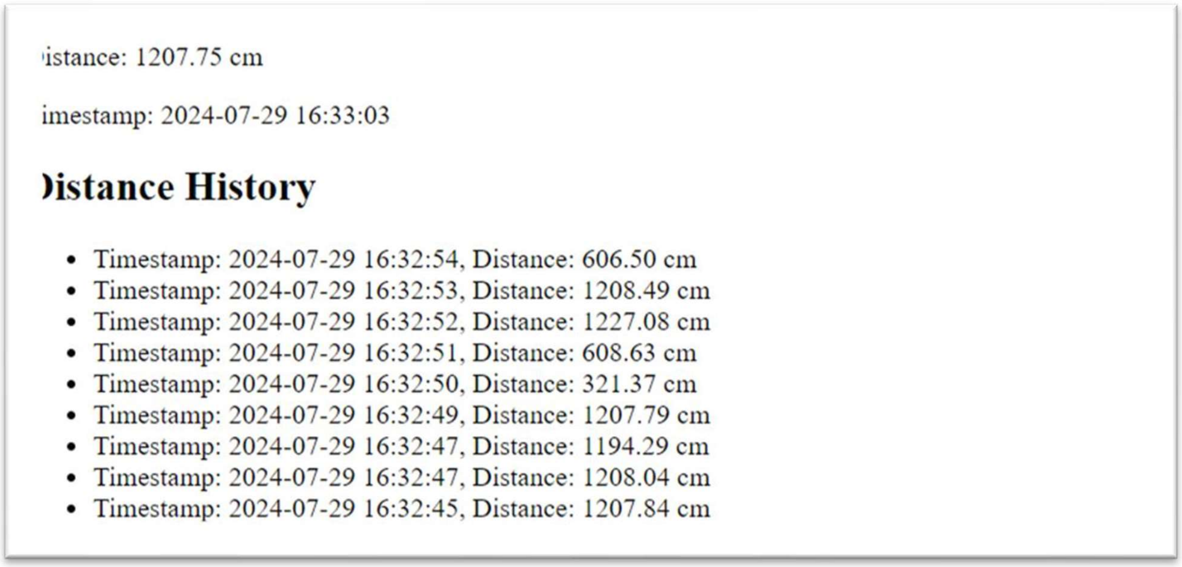


```
pi@raspberrypi:~ $ sudo raspi-config
pi@raspberrypi:~ $ mkdir object_detect
pi@raspberrypi:~ $ cd object_detect
pi@raspberrypi:~/object_detect $ mkdir static
pi@raspberrypi:~/object_detect $ mkdir template
pi@raspberrypi:~/object_detect $ touch app.py
pi@raspberrypi:~/object_detect $ touch template/index.html
pi@raspberrypi:~/object_detect $ sudo python3 app.py
```



```
pi@raspberrypi:~/object_detect $ sudo python3 app.py
/home/pi/object_detect/app.py:16: RuntimeWarning: This channel is already in use
, continuing anyway.  Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(TRIG, GPIO.OUT)
/home/pi/object_detect/app.py:21: RuntimeWarning: This channel is already in use
, continuing anyway.  Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(SERVO_PIN, GPIO.OUT)
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
192.168.29.138 - - [29/Jul/2024 16:30:26] "GET /history HTTP/1.1" 200 -
192.168.29.138 - - [29/Jul/2024 16:30:26] "GET /distance HTTP/1.1" 200 -
192.168.29.138 - - [29/Jul/2024 16:30:27] "GET /distance HTTP/1.1" 200 -
192.168.29.138 - - [29/Jul/2024 16:30:29] "GET /distance HTTP/1.1" 200 -
192.168.29.138 - - [29/Jul/2024 16:30:31] "GET /distance HTTP/1.1" 200 -
192.168.29.138 - - [29/Jul/2024 16:30:31] "GET /history HTTP/1.1" 200 -
```

- To see the webserver copy the ipaddress and paste in in browser. Ipaddress is 192.168.29.138:5000



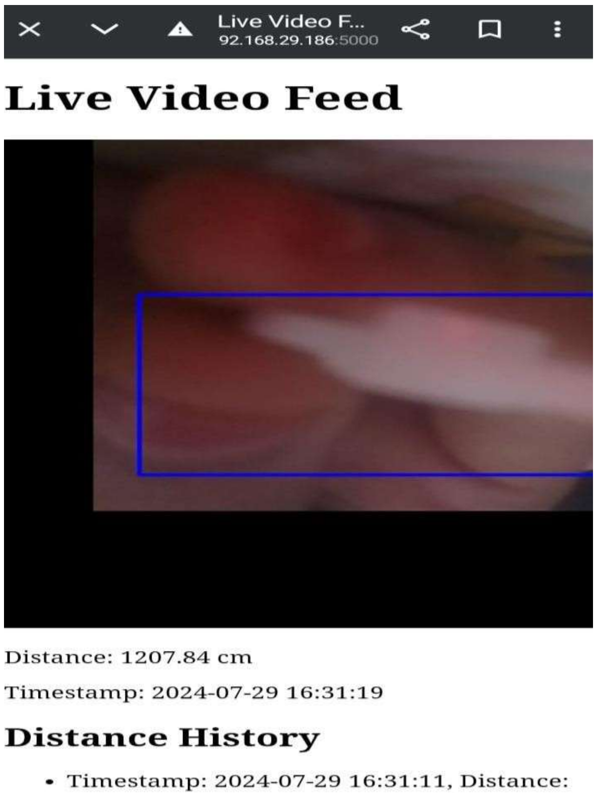
istance: 1207.75 cm

imestamp: 2024-07-29 16:33:03

istance History

- Timestamp: 2024-07-29 16:32:54, Distance: 606.50 cm
- Timestamp: 2024-07-29 16:32:53, Distance: 1208.49 cm
- Timestamp: 2024-07-29 16:32:52, Distance: 1227.08 cm
- Timestamp: 2024-07-29 16:32:51, Distance: 608.63 cm
- Timestamp: 2024-07-29 16:32:50, Distance: 321.37 cm
- Timestamp: 2024-07-29 16:32:49, Distance: 1207.79 cm
- Timestamp: 2024-07-29 16:32:47, Distance: 1194.29 cm
- Timestamp: 2024-07-29 16:32:47, Distance: 1208.04 cm
- Timestamp: 2024-07-29 16:32:45, Distance: 1207.84 cm

•



Live Video Feed

Distance: 1207.84 cm

Timestamp: 2024-07-29 16:31:19

Distance History

- Timestamp: 2024-07-29 16:31:11, Distance:

1. **Ultrasonic Sensor**:
   - Continuously measures the distance to objects.
   - Triggers servo motor rotation if an object is within 20 cm.
   - Logs distance data with timestamps in the database.
2. **Servo Motor**:
   - Rotates to 180 degrees when an object is detected within 20 cm.
   - Provides a visual response to object detection.
3. **Camera Module**:
   - Captures live video feed.
   - Processes the feed for object detection and centers the view on detected objects.
4. **Web Server**:
   - Streams live video feed to connected clients.
   - Displays current and historical distance measurements.

11. ADVANTAGES:

- Live Video Feed: Users can monitor the environment in real-time through a web interface, providing immediate feedback and situational awareness.
- Instant Distance Measurement: The ultrasonic sensor provides continuous, real-time distance measurements, enabling quick detection of nearby objects.
- Enhanced Surveillance: The object detection feature enhances security and surveillance capabilities by highlighting and centering detected objects in the video feed.
- Database Integration: Distance measurements are logged with timestamps in an SQLite database, allowing for historical analysis and record-keeping.
- User-Friendly Interface: The web server displays both real-time and historical distance data, making it easy for users to track changes over time.
- Security Systems: This project can be used in security systems to monitor and detect intruders, triggering alarms or other responses.
- Automation: The ability to measure distances and respond automatically can be applied in various automation systems, such as robotics, where precise object detection and movement are crucial.
- Affordable Components: The project uses readily available and affordable components, such as the Raspberry Pi, HC-SR04 ultrasonic sensor, and SG90 servo motor.
- Open-Source Software: Leveraging open-source libraries and tools reduces software costs and allows for customization and expansion.

12. APPLICATIONS:

1. Home Security and Surveillance
2. Intruder Detection
3. Industrial Automation
4. Smart Home Systems
5. Healthcare and Elderly Care
6. Automotive Applications

13. FUTURE SCOPE:

- Integrate with AI and ML and Implement more advanced AI algorithms for object detection and recognition, allowing the system to identify specific objects, people, or animals.
- Add Dataset of Aircrafts to use this in Defence .
- Expand it and add additional IOT sensor to use in smart home and smart city project.
- Expand the project to be secured and used in Mobile applications.
- Enchance with proper encryption and decryption techniques of data and transmission.
- Add this data to cloud and add feedback mechanisms to develop the system further.
- Provide users with more control over their data, including options for data anonymization and selective sharing.
- Develop a modular system where components can be easily added or removed based on specific needs, making it highly customizable.
- Provide customizable firmware and software options for different applications and user requirements.
- Integrate with renewable energy sources like solar panels to create a sustainable monitoring system.
- Implement more complex automation routines, such as triggering specific actions based on multiple sensor inputs or user-defined rules.
- Use feedback from sensors to dynamically adjust system parameters, such as changing camera angles or adjusting sensor sensitivity.
- Store data in the cloud for long-term storage and advanced analysis, enabling remote access to historical data and advanced data processing.

14. CONCLUSION:

This internship has been a transformative experience, providing both practical skills and theoretical knowledge essential for success in the field of IoT. The projects completed, the challenges overcome, and the lessons learned have collectively prepared us to tackle future IoT endeavors with confidence and creativity. This internship has not only broadened our technical expertise but also inspired a continued commitment to innovation and excellence in the realm of IoT.

This project successfully demonstrates the integration of ultrasonic distance measurement, servo motor control, and real-time video streaming with object detection. It leverages a combination of hardware components like ultrasonic sensors, servo motors, and cameras, along with software frameworks such as OpenCV, Flask, and SQLite, to create a versatile and functional system. The project's ability to detect objects, measure distances, and respond with automated actions, such as servo motor rotation, showcases its potential in various practical applications, including home security, industrial automation, and smart home systems.