**PUBLIC TRANSPORTION OPTIMIZATION**

**IoT – PHASE – 3**

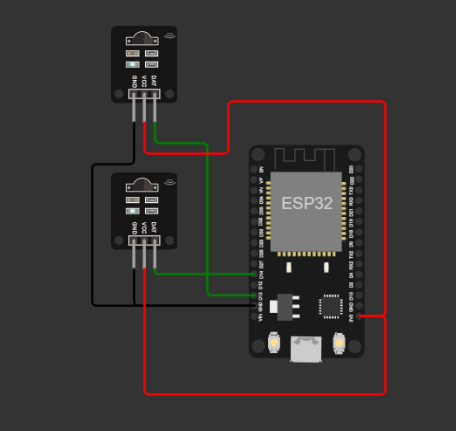
**DEVELOPMENT - 1**

***OBJECTIVES***

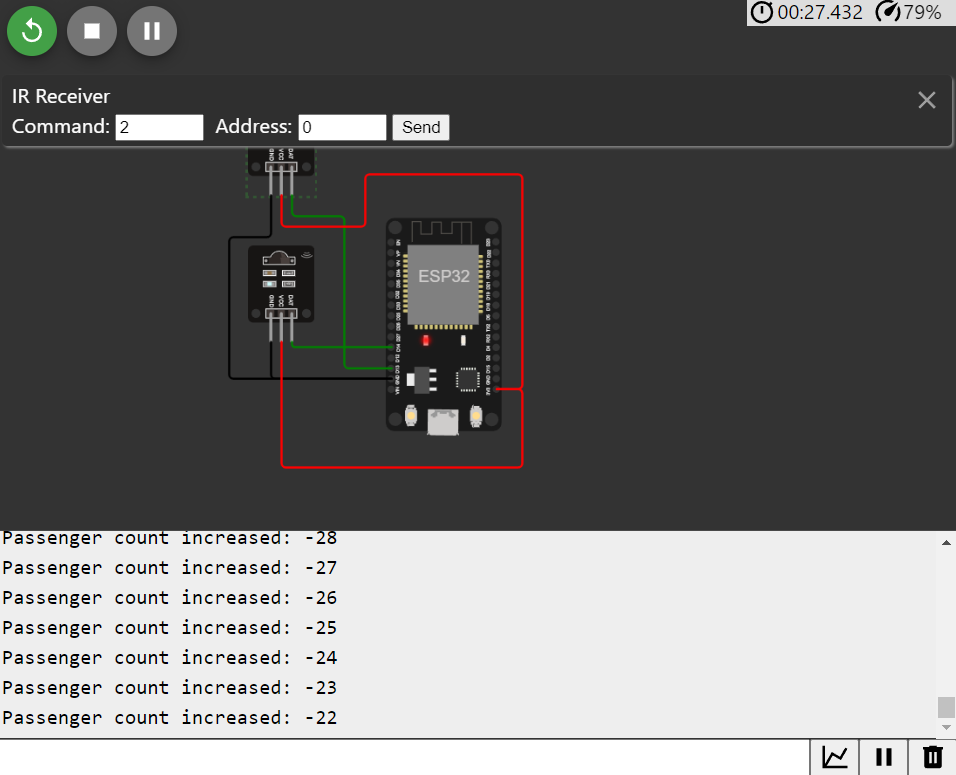
* ***To keep track of the count of people in the vehicle.***
* ***To track the position of the vehicle in real-time.***

***Setting up the counter device:***

|  |
| --- |
| **CODE**  const int irSensor1 = 13; // IR Sensor 1 connected to GPIO 13  const int irSensor2 = 14; // IR Sensor 2 connected to GPIO 14  int passengerCount = 0;  int sensor1State = 0; // Current state of IR Sensor 1  int sensor2State = 0; // Current state of IR Sensor 2  int lastSensor1State = 0; // Previous state of IR Sensor 1  int lastSensor2State = 0; // Previous state of IR Sensor 2  void setup() {  pinMode(irSensor1, INPUT);  pinMode(irSensor2, INPUT);  Serial.begin(115200);  }  void loop() {  // Read the current state of IR Sensor 1 and IR Sensor 2  sensor1State = digitalRead(irSensor1);  sensor2State = digitalRead(irSensor2);  // Check for a change in state  if (sensor1State != lastSensor1State) {  if (sensor1State == HIGH) {  passengerCount++;  Serial.println("Passenger count increased: " + String(passengerCount));  }  lastSensor1State = sensor1State;  }  if (sensor2State != lastSensor2State) {  if (sensor2State == HIGH) {  passengerCount--;  Serial.println("Passenger count decreased: " + String(passengerCount));  }  lastSensor2State = sensor2State;  }  } |



**ESP32 SETUP**

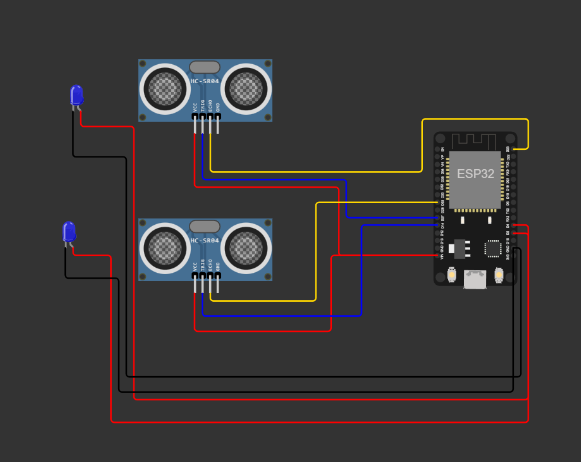


**OUTPUT**

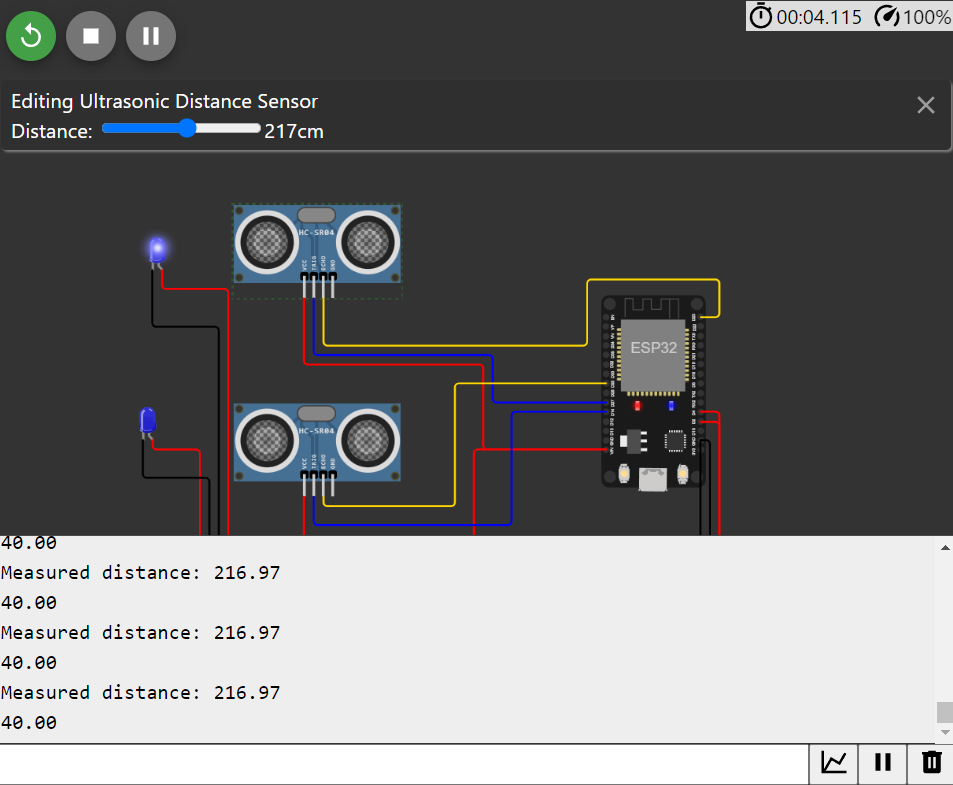
|  |
| --- |
| ***Python script for the same:***  import RPi.GPIO as GPIO  import time  irSensor1 = 13 # IR Sensor 1 connected to GPIO 13  irSensor2 = 14 # IR Sensor 2 connected to GPIO 14  passengerCount = 0  sensor1State = 0 # Current state of IR Sensor 1  sensor2State = 0 # Current state of IR Sensor 2  lastSensor1State = 0 # Previous state of IR Sensor 1  lastSensor2State = 0 # Previous state of IR Sensor 2  # Setup GPIO pins  GPIO.setmode(GPIO.BCM)  GPIO.setup(irSensor1, GPIO.IN)  GPIO.setup(irSensor2, GPIO.IN)  try:  while True:  # Read the current state of IR Sensor 1 and IR Sensor 2  sensor1State = GPIO.input(irSensor1)  sensor2State = GPIO.input(irSensor2)  # Check for a change in state  if sensor1State != lastSensor1State:  if sensor1State == 1:  passengerCount += 1  print("Passenger count increased:", passengerCount)  lastSensor1State = sensor1State  if sensor2State != lastSensor2State:  if sensor2State == 1:  passengerCount -= 1  print("Passenger count decreased:", passengerCount)  lastSensor2State = sensor2State  time.sleep(0.1) # Adjust the delay as needed  except KeyboardInterrupt:  GPIO.cleanup() |

***Setting up the position detector using distance:***

|  |
| --- |
| **CODE**  #define ECHO\_PIN1 23 //Pins for Sensor 1  #define TRIG\_PIN1 27 //Pins for Sensor 1  #define ECHO\_PIN2 25 //Pins for Sensor 2  #define TRIG\_PIN2 14 //Pins for Sensor 2  int LEDPIN1 = 2;  int LEDPIN2 = 4;  void setup() {  Serial.begin(115200);  pinMode(LEDPIN1, OUTPUT);  pinMode(TRIG\_PIN1, OUTPUT);  pinMode(ECHO\_PIN1, INPUT);  pinMode(LEDPIN2, OUTPUT);  pinMode(TRIG\_PIN2, OUTPUT);  pinMode(ECHO\_PIN2, INPUT);  }  float readDistance1CM() {  digitalWrite(TRIG\_PIN1, LOW);  delayMicroseconds(2);  digitalWrite(TRIG\_PIN1, HIGH);  delayMicroseconds(10);  digitalWrite(TRIG\_PIN1, LOW);  int duration = pulseIn(ECHO\_PIN1, HIGH);  return duration \* 0.034 /2 ;  }  float readDistance2CM() {  digitalWrite(TRIG\_PIN2, LOW);  delayMicroseconds(2);  digitalWrite(TRIG\_PIN2, HIGH);  delayMicroseconds(10);  digitalWrite(TRIG\_PIN2, LOW);  int duration = pulseIn(ECHO\_PIN2, HIGH);  return duration \* 0.034 / 2;  }  void loop() {  float distance1 = readDistance1CM();  float distance2 = readDistance2CM();  bool isNearby1 = distance1 > 200;  digitalWrite(LEDPIN1, isNearby1);  bool isNearby2 = distance2 > 200;  digitalWrite(LEDPIN2, isNearby2);  Serial.print("Measured distance: ");  Serial.println(readDistance1CM());  Serial.println(readDistance2CM());  delay(100);  } |

******

**ESP32 SETUP**

****

**OUTPUT**

|  |
| --- |
| ***Python script for the same:***  import RPi.GPIO as GPIO  import time  # Pins for Sensor 1  TRIG\_PIN1 = 27  ECHO\_PIN1 = 23  # Pins for Sensor 2  TRIG\_PIN2 = 14  ECHO\_PIN2 = 25  LEDPIN1 = 2  LEDPIN2 = 4  GPIO.setmode(GPIO.BCM)  GPIO.setup(LEDPIN1, GPIO.OUT)  GPIO.setup(TRIG\_PIN1, GPIO.OUT)  GPIO.setup(ECHO\_PIN1, GPIO.IN)  GPIO.setup(LEDPIN2, GPIO.OUT)  GPIO.setup(TRIG\_PIN2, GPIO.OUT)  GPIO.setup(ECHO\_PIN2, GPIO.IN)  def read\_distance\_cm(trig\_pin, echo\_pin):  GPIO.output(trig\_pin, False)  time.sleep(0.2)  GPIO.output(trig\_pin, True)  time.sleep(0.00001)  GPIO.output(trig\_pin, False)  pulse\_start\_time = time.time()  pulse\_end\_time = time.time()  while GPIO.input(echo\_pin) == 0:  pulse\_start\_time = time.time()  while GPIO.input(echo\_pin) == 1:  pulse\_end\_time = time.time()  pulse\_duration = pulse\_end\_time - pulse\_start\_time  distance = pulse\_duration \* 17150  return round(distance, 2)  try:  while True:  distance1 = read\_distance\_cm(TRIG\_PIN1, ECHO\_PIN1)  distance2 = read\_distance\_cm(TRIG\_PIN2, ECHO\_PIN2)  is\_nearby1 = distance1 > 200  GPIO.output(LEDPIN1, is\_nearby1)  is\_nearby2 = distance2 > 200  GPIO.output(LEDPIN2, is\_nearby2)  print("Measured distance Sensor 1:", distance1, "cm")  print("Measured distance Sensor 2:", distance2, "cm")  time.sleep(0.1)  except KeyboardInterrupt:  GPIO.cleanup() |

***To setup a real-time IoT platform***

Setting up an IoT platform for receiving real-time data from IoT devices like people counters and distance detectors involves several steps. In this guide, I'll provide an overview of the process:

***Step 1:*** Choose an IoT Platform

There are various IoT platforms available, both cloud-based and self-hosted. Choose an IoT platform that fits your project's requirements and your level of expertise. Some popular cloud-based IoT platforms include AWS IoT, Google Cloud IoT, and Microsoft Azure IoT. If you prefer self-hosted solutions, you can consider platforms like ThingsBoard or Home Assistant.

***Step 2:*** Set Up Your IoT Platform

Depending on your choice of IoT platform, you'll need to create an account, set up a project, or deploy the platform on your server. Follow the platform's documentation to get it up and running.

***Step 3:*** Configure Your IoT Devices

Each IoT device (ESP32, Arduino, or any other device) should be configured to communicate with your chosen IoT platform. This involves setting up Wi-Fi connectivity and using MQTT or HTTP to send data to the platform.

Here's a high-level overview of the process:

For ESP32:

* Set up your ESP32 to connect to your Wi-Fi network.
* Use an MQTT library or HTTP requests to send data to your IoT platform. You'll need to configure the platform-specific settings (e.g., server address, authentication credentials).

For Arduino:

* Use Ethernet or Wi-Fi shields/modules to connect your Arduino to your network.
* Similarly, use MQTT libraries or HTTP requests to send data to your IoT platform.

***Step 4:*** Define Data Models and Topics

In your IoT platform, define data models that correspond to the data you'll receive from your devices. Create topics or channels where your devices will publish data. You should align these definitions with the types of data your devices will send (e.g., passenger count, distance measurements).

***Step 5:*** Device Code Configuration

Modify the code on your devices (ESP32 and Arduino) to send data to your IoT platform. Use the defined topics and data models. Make sure your devices have internet access (Wi-Fi or Ethernet) and can reach your platform's servers.

***Step 6:*** Data Processing and Visualization

On your IoT platform, set up data processing rules to interpret and process incoming data. You may want to visualize the data using dashboards, graphs, or other visualization tools provided by your platform.

***Step 7:*** Real-Time Monitoring

Configure alerts or notifications to be triggered when specific events occur (e.g., passenger count exceeds a threshold). This will enable you to receive real-time notifications or alarms.

***Step 8:*** Secure Your IoT Devices and Platform

Ensure that your IoT devices and platform are adequately secured. Use secure connections (HTTPS, MQTT over TLS) and implement authentication and authorization mechanisms.

***Step 9:*** Testing and Debugging

Test your setup by sending data from your devices and monitoring the data flow on your IoT platform. Debug and troubleshoot any issues you encounter.

***Step 10:*** Scale and Optimize

As your project grows, consider scaling your IoT platform to accommodate more devices and data. Optimize your code and configurations for better performance and cost-effectiveness.