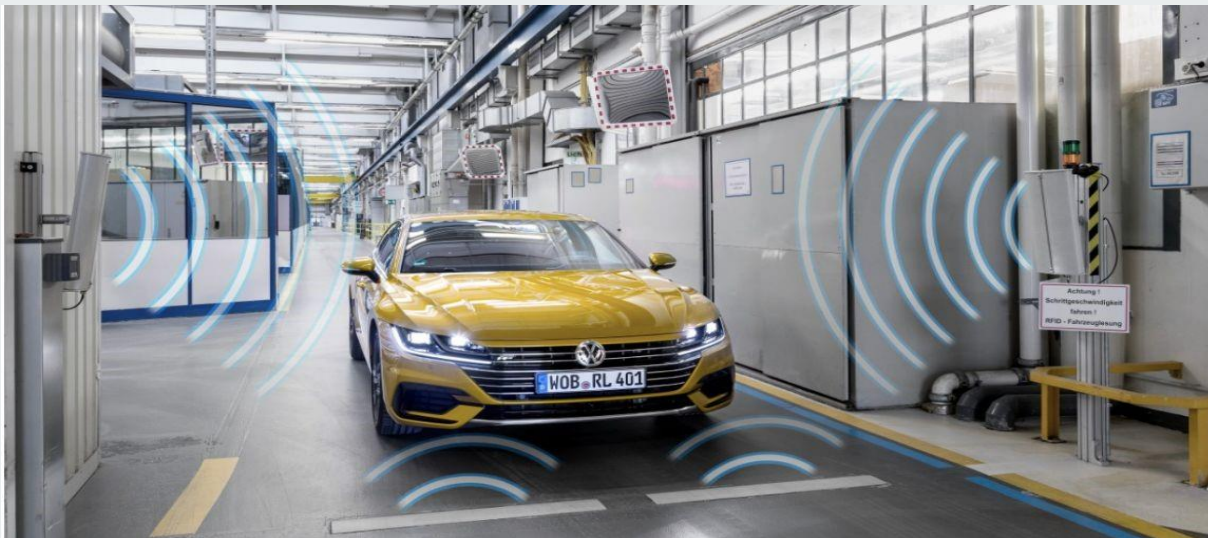




SMART PARKING

Using Internet of Things




Theme: Real-time Occupancy and Data Monitoring and Data Transmission to cloud



Introduction:

A simple implementation of a sensor data collection system using Raspberry Pi Pico. It utilizes ultrasonic sensors to send the occupancy data to a cloud server. This project can be extended for applications such as parking management, occupancy detection, and more.



Working Principle:

The code sets up three ultrasonic sensors, each consisting of a trigger pin and an echo pin, connected to the Raspberry Pi Pico. By emitting ultrasonic waves and measuring the time taken for the waves to bounce back, the sensors determine the distances of nearby objects. If the measured distance is less than a predefined threshold, it is considered as occupancy. The script then sends this occupancy data to a cloud server using the Wi-Fi module, allowing real-time monitoring and analysis of the sensor data for various applications.



Components :

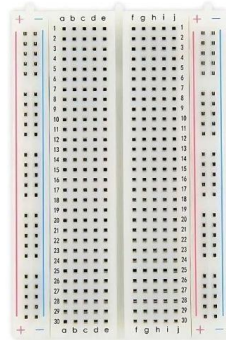
- HC-SR04 Ultrasonic Distance Sensor



- Raspberry Pi Pico



- Mini BreadBoard



- Jumper wires




ESP32



PIN Connection

For the first sensor:

- VCC Pin: Connect to the 5V pin on the Raspberry Pi Pico (usually labeled as VBUS or VSYS).
 - GND Pin: Connect to any of the ground pins on the Raspberry Pi Pico (labeled as GND).
 - Trigger pin: Connect to Pin 0 (GP0) on the Raspberry Pi Pico.
 - Echo pin: Connect to Pin 1 (GP1) on the Raspberry Pi Pico.
- 

For the second sensor:

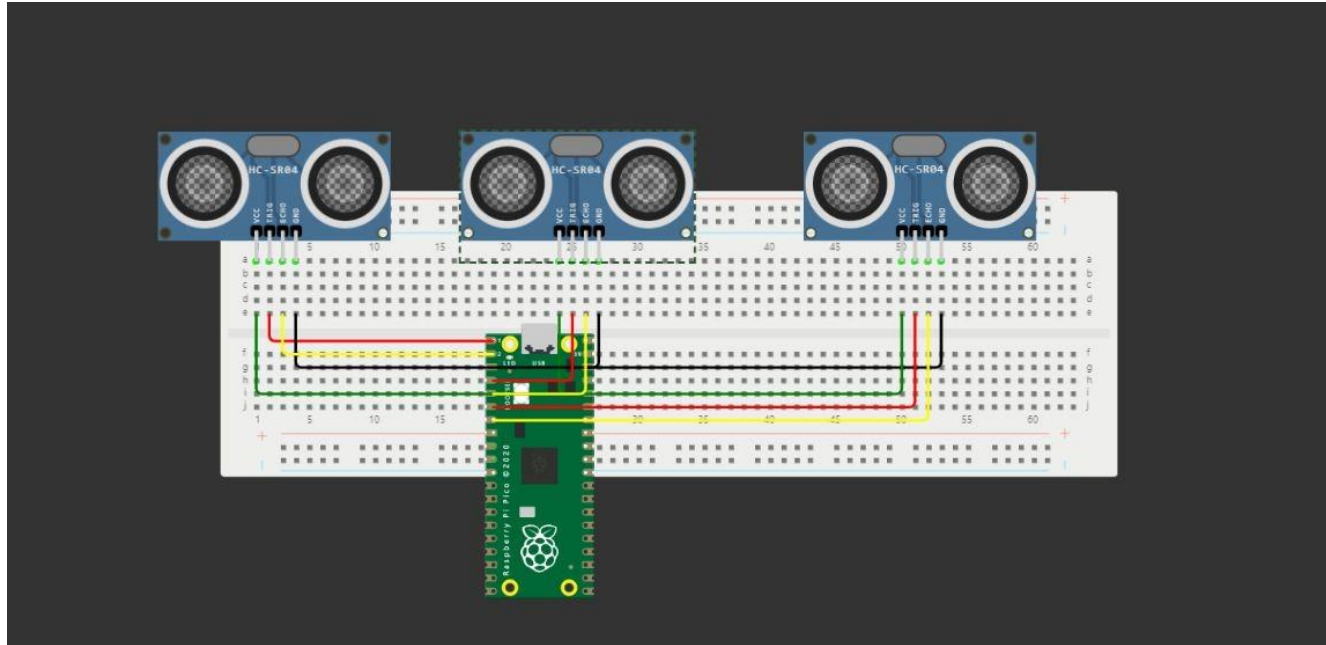
- VCC Pin: Connect to the 5V pin on the Raspberry Pi Pico.
- GND Pin: Connect to any of the ground pins on the Raspberry Pi Pico.
- Trigger pin: Connect to Pin 2 (GP2) on the Raspberry Pi Pico.
- Echo pin: Connect to Pin 3 (GP3) on the Raspberry Pi Pico.

For the third sensor:

- VCC Pin: Connect to the 5V pin on the Raspberry Pi Pico.
- GND Pin: Connect to any of the ground pins on the Raspberry Pi Pico.
- Trigger pin: Connect to Pin 4 (GP4) on the Raspberry Pi Pico.
- Echo pin: Connect to Pin 5 (GP5) on the Raspberry Pi Pico.



Diagram :



MicroPython Code

```
import machine import
```

```
time import urequests
```

```
# Define the pin configuration for each sensor sensor_pins = [(0, 1), (2, 3), (4, 5)]
```

```
BEECEPTOR_ENDPOINT = "https://cloudplatform.free.beeceptor.com"
```

```
# Function to measure the distance from the ultrasonic sensor def
```

```
measure_distance(trigger_pin, echo_pin):    trigger = machine.Pin(trigger_pin,  
machine.Pin.OUT)    echo = machine.Pin(echo_pin, machine.Pin.IN)
```

```
    trigger = machine.Pin(trigger_pin, machine.Pin.OUT)    echo = machine.Pin(echo_pin,  
machine.Pin.IN)
```

```
    trigger.low()    time.sleep_us(2)
```

```
    trigger.high()    time.sleep_us(10)    trigger.low()
```

```
while echo.value() == 0:
    signaloff = time.time()
    while echo.value()
== 1:
    signalon = time.time()

    timepassed = signalon - signal off
    distance = (timepassed *
0.0343) / 2
    return distance
```

```
# Function to send data to Beeceptor endpoint
def send_data_to_beeceptor(data):
    headers = {'Content-Type': 'application/json'}
    json_data = {'occupancy': data}
    try:
        response = requests.post(BEECEPTOR_ENDPOINT, json=json_data, headers=headers)
        print("Data sent successfully")
        print(response.text)
    except Exception as e:
        print("An error occurred while sending data:", e)
```

Main loop to continuously measure and send data from all three sensors while True:

```
sensor_data = []    for trigger_pin, echo_pin in sensor_pins:
    distance = measure_distance(trigger_pin, echo_pin)    print(f'Distance:
{distance} cm from Sensor {sensor_pins.index((trigger_pin, echo_pin)) + 1}')
occupancy = 1 if distance < 10 else 0    sensor_data.append(occupancy)
send_data_to_beeceptor(sensor_data)    time.sleep(5) # Adjust the sleep time as
needed
```



Code Explanation

- 1 - Definition of Pin Configuration: Three sets of trigger and echo pins are defined to connect the ultrasonic sensors to the Raspberry Pi Pico.
- 2 - Distance Measurement Function: The 'measure_distance' function calculates the distance based on the time taken for the ultrasonic signal to return.
- 3 - Data Sending Function: The 'send_data_to_beeceptor' function sends the occupancy data to the specified Beeceptor endpoint.
- 4 - Main Loop: The main loop continuously measures data from all three sensors, determines the occupancy status based on the measured distance, and then sends this data to the Beeceptor endpoint.




HTTP Mocking Rules on End User

Method - POST

Response Headers

```
{
  "method": "POST",
  "path": "/data",
  "response": {
    "statusCode": 200,
    "headers": {
      "Content-Type": "application/json"
    },
    "body": {
      "message": "Received POST data",
      "occupancy1": "{{sensor1}}",
      "occupancy2": "{{sensor2}}",
      "occupancy3": "{{sensor3}}",
      "occupancy4": "{{sensor4}}"
    }
  }
}
```



Method -GET

Response Headers

```
{
  "method": "GET",
  "path": "/data",
  "response": {
    "statusCode": 200,
    "headers": {
      "Content-Type": "application/json"
    },
    "body": {
      "message": "Received POST data",
      "occupancy1": "{{sensor1}}",
      "occupancy2": "{{sensor2}}",
      "occupancy3": "{{sensor3}}",
      "occupancy4": "{{sensor4}}"
    }
  }
}
```

Method

-GET

Response Headers

```
{"Occupancy1:occupancy1"}
{"Occupancy2:occupancy2"}
{"Occupancy3:occupancy3"}
{"Occupancy4:occupancy4"}
```

Usage Scenario

- 1 - Sensor Installation: Install the ultrasonic sensors at strategic locations within the parking lot, ensuring comprehensive coverage of the parking area.
- 2 - Raspberry Pi Pico Integration: Connect the ultrasonic sensors to the Raspberry Pi Pico, ensuring the correct mapping of trigger and echo pins as outlined in the script.
- 3 - Data Collection and Transmission: Run the script on the Raspberry Pi Pico to collect and process real-time occupancy data. The script measures the distance between the sensor and any parked vehicle and sends this data to the cloud platform through the BEECEPTOR endpoint.
- 4 - Cloud-Based Analysis: Receive the occupancy data on the cloud platform, enabling parking lot managers to monitor and analyze parking space utilization trends. This data can be used to identify parking patterns, optimize parking allocation, and provide real-time parking availability updates to users.
- 5 - Optimized Parking Experience: With real-time occupancy data, users can easily locate available parking spaces, reducing the time spent searching for parking.

