Smart Public Restroom

Using internet of thing to detect occupancy status and send to a end point

Basic Details of the Team

Team Name : Proj_224080_Team_1

Team Leader Name : Akash.M

Team Based : Smart Public Restroom

Institute Name : Chendu college of engineering and technology

Theme : Smart Public Restroom with Integrated Sensing and Control

Detect occupancy status and send to a end-point

Introduction

A smart public restroom system using Raspberry Pi Pico. Leveraging ultrasonic sensors, the code enables real-time occupancy monitoring and data transmission to a central cloud server. This prototype can be extended to facilitate efficient management of public restrooms, ensuring improved maintenance, and enhanced user experience.

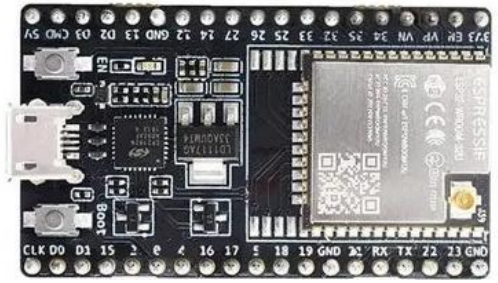
Working Principle:

- By employing multiple ultrasonic sensors, strategically positioned within the public restroom area, the script continuously measures the proximity of individuals to each sensor.
- Upon detecting an individual within a certain range, the sensor registers the occupancy status.
- The Raspberry Pi Pico, equipped with Wi-Fi capabilities, facilitates the seamless transfer of this occupancy data to a cloud server.
- This data can be leveraged for real-time analysis, enabling authorities to monitor restroom traffic, optimize cleaning schedules, and ensure timely maintenance, thereby enhancing the overall user experience and satisfaction.

Components Required

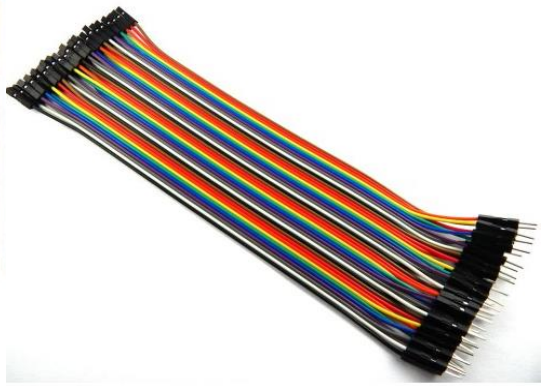
01. Raspberry pi pico

02. Esp32

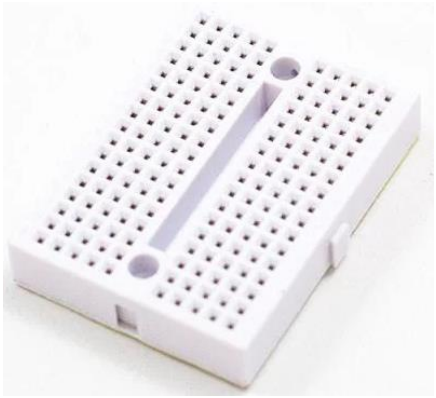


03. Ultra Sonic

04. Jumper cable



05. breadboard



Pin Connection

The First Sensor:

1. VCC Pin: Connect to the 5V pin on the Raspberry Pi Pico.
2. GND Pin: Connect to any of the ground pins on the Raspberry Pi Pico.

The Second Sensor:

1. VCC Pin: Connect to the 5V pin on the Raspberry Pi Pico.
2. GND Pin: Connect to any of the ground pins on the Raspberry Pi Pico.

The Third Sensor:

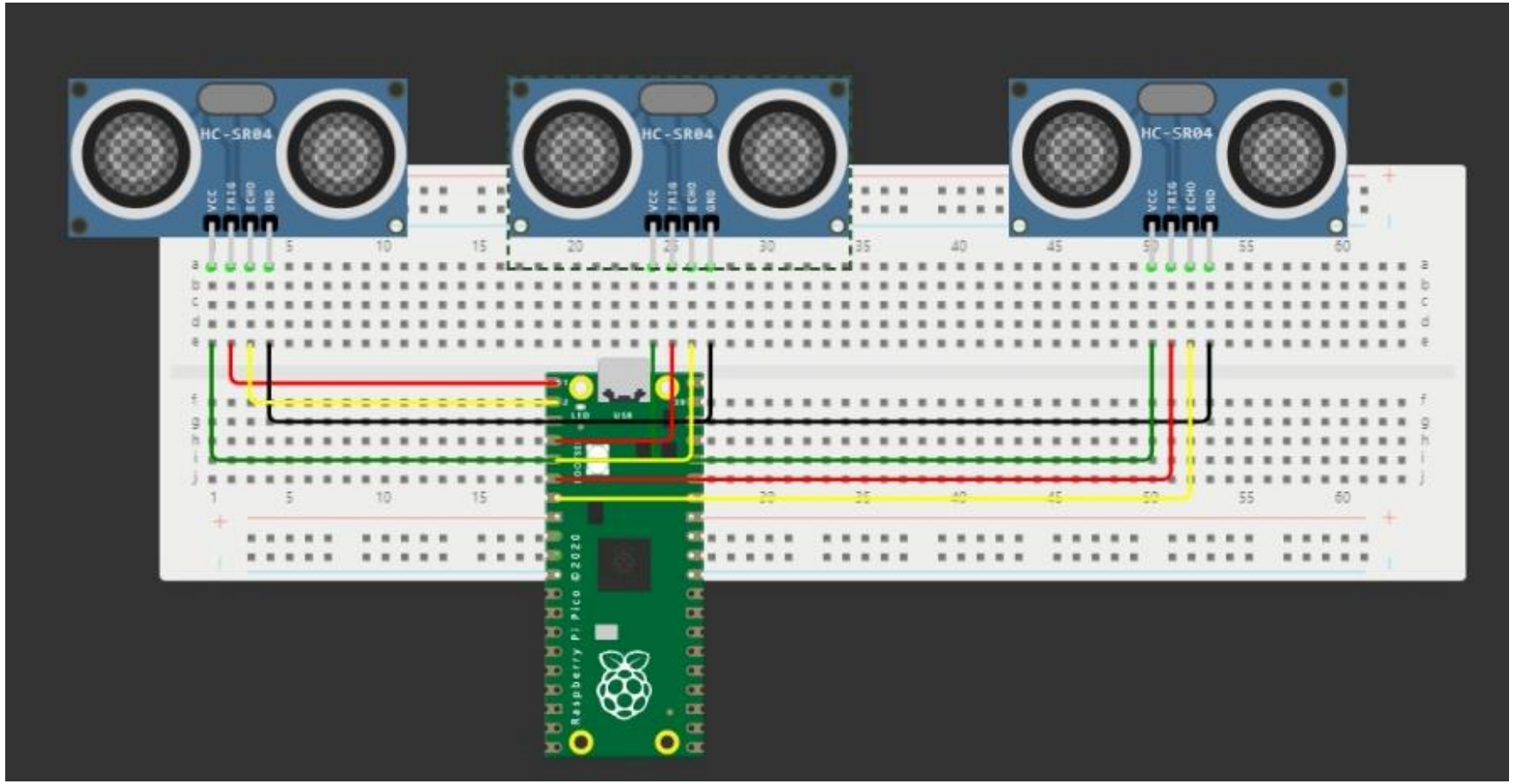
1. VCC Pin: Connect to the 5V pin on the Raspberry Pi Pico.
2. GND Pin: Connect to any of the ground pins on the Raspberry Pi Pico.

Trigger pin: Pin 0
Echo pin: Pin 1

Trigger pin: Pin 2
Echo pin: Pin 3

Trigger pin: Pin 4
Echo pin: Pin 5

Circuit diagram:



Micro Python 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.low()
    time.sleep_us(2)
    trigger.high()
    time.sleep_us(10)
    trigger.low()

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

    timepassed = signalon - signaloff
    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 = urequests.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 explain

- A function send_data_to_beeceptor is defined to send data to a Beeceptor endpoint. It sends data as JSON with a Content-Type header set to application/json. If successful, it prints a success message along with the response text. If an error occurs, it catches the exception and prints an error message.
- There is an undefined variable urequests used in the send_data_to_beeceptor function. You should ensure that the urequests library or module is properly imported or defined before using it.
- A while True loop is used for continuously measuring and sending data from three sensors.
- Inside the loop, there's a list sensor_data used to store the occupancy status of each sensor. It measures the distance for each sensor, calculates occupancy (1 if the distance is less than 10 cm, otherwise 0), and appends the occupancy value to the sensor_data list. The distance measurements are printed.
- After measuring data from all three sensors, the sensor_data list is passed to the send_data_to_beeceptor function for sending the occupancy data to the Beeceptor endpoint.
- Finally, the loop sleeps for 5 seconds (adjustable) before repeating the process. This can be used to control the rate at which data is sent to the Beeceptor endpoint.

Http mocking Rules

Method - POST

Response Header

```
{
 "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 Header

```
{
 "method": "GET",  "path": "/data",
"response": {    "statusCode": 200,
"headers": {      "Content-Type":
"application/json"    },    "body": {
"message": "Sensor data retrieved
successfully",      "occupancy1":
"{{query.occupancy1}}",
"occupancy2": "{query.occupancy2}}",
"occupancy3": "{{query.occupancy3}}",
"occupancy4":
"{{query.occupancy4}}"    }  }}
```

Response Body

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

Usage Scenario

- The provided code serves a pivotal role in monitoring the occupancy status of individual restroom stalls.
- Each stall is equipped with motion sensors that continuously track the presence of individuals.
- This code facilitates real-time data collection by processing the sensor inputs and, when it detects motion, marking the stall as occupied, denoting this with a value of 1.
- When the sensors no longer detect motion, the code signifies an empty stall with a value of 0.
- This information is then relayed to a central system or server using a Beeceptor endpoint, enabling facility managers to monitor restroom stall occupancy in real time.
- In addition, this data can be used to implement smart features, such as displaying real-time stall availability, sending alerts for cleaning and maintenance, or optimizing resource allocation, ultimately enhancing the efficiency and user-friendliness of the public restroom while improving maintenance operations.

Team Members Details

Role In Team	Name	Branch BE	Year
Team Leader	Akash	CSE	3rd
Team Member 1	Ahmed abrarul hag	CSE	3rd
Team Member 2	Gokula Kannan	CSE	3rd
Team Member 3	Dhamodhar	CSE	3rd