IOT BASED SMART PARKING SYSTEM
PHASE 3
DEVELOPMENT PART-1

Components:

Raspberry Pi Board:

- The Raspberry Pi board includes essential components such as RAM, a processor, GPU, GPIO pins, and interfaces for external devices.
- It also features an Ethernet port, UART (for sensor and peripheral communication), and a power source connector.

Mass Storage:

- To store the necessary program and data for the project, the Raspberry Pi uses an SD flash memory card.
- This SD card serves as the primary storage medium, similar to how a PC boots from a hard disk.

Boot Configuration:

- The Raspberry Pi is configured to boot from the SD card, enabling it to run the project's software and interact with external devices.

Hardware Requirements:

- Key hardware components for the project include the Raspberry Pi board, an SD card preloaded with the Linux OS, a compatible US keyboard for input, a monitor for display, and a power supply.

Optional Hardware Additions:

- Additional hardware components for the project can include:
    - A USB mouse for user interaction.
    - A powered USB hub for expanding USB connectivity.
    - A protective case for the Raspberry Pi.
    - An internet connection, which can be established with a USB Wi-Fi adapter (for the Model A or B Raspberry Pi) or a LAN cable (for the Model B).

ESP32 Microcontroller:

- The ESP32 microcontroller can be connected to the Raspberry Pi via GPIO pins or UART for sensor data processing or wireless communication, enhancing the project's capabilities.

Servo Motor:

- The servo motor can be used to control mechanical movements or devices, and it can be interfaced with the Raspberry Pi for precise control.

Ultrasonic Sensors:

- Ultrasonic sensors can provide distance measurement capabilities to the project. They emit sound waves and measure the time it takes for the waves to bounce back, helping the system detect objects and their distances.
-

Explanation:

Raspberry Pi is a series of small, credit-card-sized single-board computers developed by the Raspberry Pi Foundation. These devices are designed to be affordable, educational, and versatile, making them suitable for various applications, from hobbyist projects to industrial use.

1..Power Pins
•5V Power
•3.3V Power

2.Ground Pin
•Ground (GND)

3.General Purpose Input/Output (GPIO) Pins
• GPIO pins that can be configured as either input or output.

4.Communication Pins
•Serial Communication (UART)
•I2C and SPI

5.PWM (Pulse Width Modulation) Pins

•Some GPIO pins are capable of PWM output, which is useful for controlling servos, motors, and dimming LEDs.

6.Camera and Display Interfaces
•CSI (Camera Serial Interface) and DSI (Display Serial Interface) connectors for connecting cameras and displays, respectively.

7.HDMI and Audio
•HDMI and audio output connectors are available for connecting a monitor or TV and for audio output.

8.Other Specialized Pins
•Depending on the Raspberry Pi model, there may be other specialized pins, such as a 1-Wire interface, the PoE (Power over Ethernet) header, or HAT (Hardware Attached on Top) support.

b. LCD DISPLAY:
The LCD (Liquid Crystal Display) selected for this project is a standard 16x2 character display. It is a widely used and cost-effective screen commonly applied in various electronic applications. Below are the key characteristics:

1. Size and Format: This LCD is designed with a 16x2 format, capable of displaying up to 16 characters in each of its two lines. The display is alphanumeric, suitable for showing letters, numbers, and symbols.

2. Communication: It employs the I2C (Inter-Integrated Circuit) protocol for communication with the control device. This simplifies wiring and minimizes the number of pins required for connection.

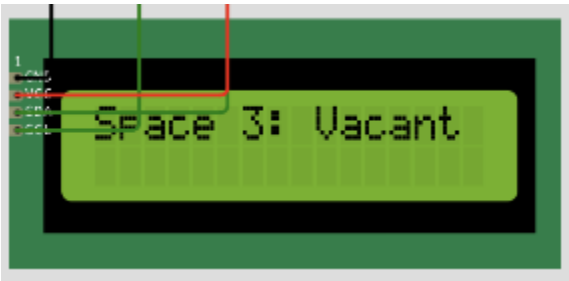
3. Backlight: The LCD features an integrated backlight, offering the option to illuminate the characters for improved visibility, especially in varying lighting conditions.

4. Information Display: In this application, the LCD serves as a visual interface for presenting crucial information, such as the status of monitored spaces. It can display messages like "Occupied" or "Vacant" for each monitored area, providing real-time updates to users.

5. Integration: The LCD connects to the control device via the I2C interface, ensuring straightforward setup and control through the project's code.

6. Versatility: LCD displays are widely used for information presentation in embedded systems, household appliances, and a variety of do-it-yourself (DIY) electronics projects due to their simplicity and reliability.

In this parking space monitoring system project, the LCD display plays a pivotal role in presenting parking space status to users, enhancing the system's usability and convenience. It offers a clear and succinct visual representation of real-time data, ensuring that users can swiftly identify parking availability.
Fig. (d) LCD Display



Ultrasonic Sensors:
In the parking space monitoring system, ultrasonic sensors are employed to detect the presence or absence of vehicles within the monitored areas. These sensors use ultrasonic sound waves, which operate beyond the range of human hearing, to determine the distance between the sensor and an object.

Detection Principle:

- Ultrasonic sensors operate on the principle of emitting high-frequency sound waves and measuring the time it takes for these sound waves to bounce back after hitting an object.
- They calculate the distance to the object based on the speed of sound and the time it takes for the sound waves to return.

Ultrasonic Waves:

- Ultrasonic waves are sound waves with frequencies higher than the upper limit of human hearing, typically in the ultrasonic range above 20 kHz.
- They are well-suited for applications requiring non-contact distance measurement and obstacle detection.

Versatile Applications:

- Ultrasonic sensors have a wide range of applications, such as proximity sensing, object detection, level measurement in tanks, and robotics.
- In the context of parking space monitoring, they play a critical role in detecting the presence of vehicles in parking spaces.

Types of Ultrasonic Sensors:

- Ultrasonic sensors come in different types to serve various purposes:
- Ultrasonic Proximity Sensors: Used for detecting the proximity or presence of objects without physical contact.
- Ultrasonic Distance Sensors: Measure the distance between the sensor and an object with high accuracy.
- Ultrasonic Range Finders: Ideal for measuring distances over a longer range.
- Ultrasonic Obstacle Avoidance Sensors: Commonly used in robotics to avoid collisions with objects.

In the context of the parking space monitoring system, ultrasonic sensors enhance the system's reliability by offering an alternative method for detecting the presence of vehicles. They work effectively in scenarios where IR sensors may have limitations, such as varying lighting conditions or the need for longer detection ranges.
Fig. (d) Ultrasonic Sensor

d. JUMPER WIRES:
1. Jumper wires are essential components in electronics and prototyping.
2. They are simple, insulated wires with connectors on each end, typically used to create

connections between electronic components on a breadboard, a PCB (printed circuit board), or other circuit platforms.
3. Jumper wires come in various forms, each suited for different applications. Here are some common types of jumper wires:
1.Male-to-Male Jumper Wires
2.Male-to-Female Jumper Wires
3. Female-to-Female Jumper Wires
4. Dupont Wires

e. Servo Motor

In the parking space monitoring and management system, a servo motor is strategically placed at the entrance of the parking area to control and regulate access. This servo motor, a rotary actuator, plays a crucial role in managing the entry and exit of vehicles.

Barrier Control:

- The servo motor is utilized to control a barrier or gate at the parking entrance, allowing or denying access to vehicles based on specific conditions.
- It can be programmed to open and close the barrier, providing entry to authorized vehicles, such as patrons with valid tickets or access cards.

Automation and Convenience:

- The integration of a servo motor in the parking system adds automation and convenience. Drivers don't need to manually lift or lower a barrier; instead, it's done automatically, improving the overall user experience.

Security and Access Control:

- The servo motor ensures the security of the parking area by preventing unauthorized access. Only vehicles with the appropriate credentials or permissions can pass through the barrier.
- This contributes to maintaining order and security within the parking facility.

Ticketing and Payment Systems:

- The servo motor can be synchronized with ticketing and payment systems. When a driver pays for parking or inserts a valid ticket, the servo motor can grant access by raising the barrier.
- Conversely, when the parking time expires or the ticket is invalid, the servo motor lowers the barrier, preventing exit until the required payment is made.

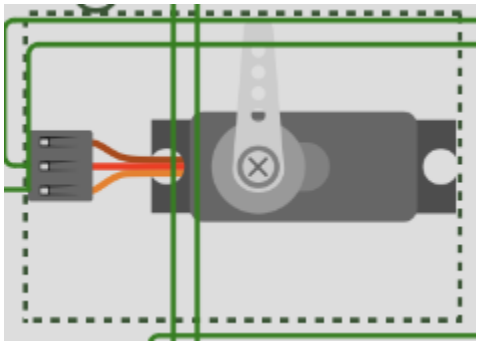
Remote Control and Monitoring:

- In modern parking management systems, the servo motor can be controlled remotely. Parking attendants or administrators can open or close the barrier as needed, even from a central control room.
- Additionally, the system can monitor the status of the servo motor, ensuring its proper functioning.

Emergency Situations:

- In emergency situations, such as fire or evacuation procedures, the servo motor can be overridden to allow free and immediate exit for all vehicles, enhancing safety measures.

Fig. (e) Servo Motor at Parking Entrance



f) ESP32

In the context of the parking space monitoring and management system, the ESP32 microcontroller is an integral component that enhances the system's functionality and connectivity.

Data Processing and Communication:

- The ESP32 microcontroller serves as the central processing unit, responsible for processing data from various sensors and communication with the central monitoring system.
- It can process data from sensors like IR sensors and ultrasonic sensors, providing real-time information on parking space occupancy.

Wireless Connectivity:

- The ESP32's key feature is its built-in Wi-Fi and Bluetooth capabilities. These wireless technologies enable seamless communication between the parking system components and the central server.
- It facilitates remote monitoring, data transmission, and system control.

Sensor Data Fusion:

- The ESP32 can gather data from multiple sensors, combining information from IR sensors, ultrasonic sensors, and other devices.
- This data fusion enables the system to provide accurate and comprehensive information about parking space availability.

Edge Computing:

- The ESP32 supports edge computing by processing data locally, reducing the need for constant communication with the central server. This enhances system responsiveness and reduces network congestion.

Real-time Status Updates:

- Through the ESP32, the system can provide real-time status updates to parking attendants and users, including the availability of parking spaces and navigation to the nearest vacant spot.

Integration with Payment and Access Control:

- The ESP32 can be integrated with payment and access control systems. When integrated with a ticketing system, it can verify payment and grant access to vehicles at the entrance.
- Additionally, it can control barriers or gates through servo motors, automating the entry and exit process.

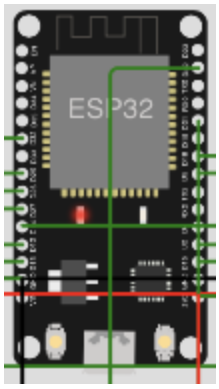
Remote Monitoring and Management:

- The ESP32 enables remote monitoring and management of the parking system. Administrators can access the system's data and control various components from a central control room.
- This ensures efficient management, rapid response to issues, and the ability to adapt to changing conditions.

Scalability and Customization:

- The flexibility of the ESP32 allows for system scalability. Additional sensors and devices can be easily integrated to meet the specific requirements of the parking facility.
- Customization and expansion options make the ESP32 a versatile choice for parking space monitoring systems.

Fig. (f) ESP32 Integration in Parking Space Monitoring System



Software Components:
a. Features of User Interface (Mobile app):
1. User Authentication and Authorization:
Sign-up and login functionality.
Social media login (e.g., using Google, Facebook, etc.).
Password reset and recovery options.
2. User Profile:
Personal information (name, email, profile picture, etc.).
Editable user settings and preferences.
3. Navigation and Menus:
Intuitive navigation menus and UI elements.
In-app navigation bars, tabs, or menus for easy access to different sections.
b. Features of Database:
Databases organize data in a structured manner. This structure can be hierarchical, network, relational, or object-oriented, depending on the type of database management system (DBMS) used.
Databases provide mechanisms to protect and secure the data from unauthorized access.

Code:(sketch.ino)

```
#include <ESP32Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h> // Include the LCD library

const int trigPin1 = 27;
const int echoPin1 = 26;
```

```
const int trigPin2 = 2;
const int echoPin2 = 15;

const int trigPin3 = 18;
const int echoPin3 = 5;

const int ledPin1 = 13;
const int ledPin2 = 12;
const int ledPin3 = 14;

const int switchPin = 32;
Servo servo;

long duration;
int distance;

bool isSwitchOn = false;

// LCD initialization (replace the address with the actual I2C
address of your LCD)
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  pinMode(trigPin1, OUTPUT);
  pinMode(echoPin1, INPUT);

  pinMode(trigPin2, OUTPUT);
  pinMode(echoPin2, INPUT);

  pinMode(trigPin3, OUTPUT);
  pinMode(echoPin3, INPUT);

  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
```

```
  pinMode(switchPin, INPUT_PULLUP);
  servo.attach(25); // Attach servo to pin D25
  servo.write(0);   // Initialize the servo position

  // LCD setup
  lcd.init(); // Initialize the LCD
  lcd.backlight(); // Turn on the backlight
  lcd.setCursor(0, 0);
  lcd.print("Hello, World!");

  Serial.begin(9600); // Initialize Serial Monitor
}

void loop() {
  isSwitchOn = digitalRead(switchPin) == LOW;

  if (isSwitchOn) {
    servo.write(90); // Open the servo
  } else {
    servo.write(0); // Close the servo
  }

  digitalWrite(trigPin1, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin1, LOW);
  duration = pulseIn(echoPin1, HIGH);
  distance = duration * 0.034 / 2;

  lcd.clear(); // Clear the previous display content
  if (distance < 200) {
    digitalWrite(ledPin1, LOW);
    lcd.setCursor(0, 0);
```

```
  lcd.print("Space 1: Occupied");
  Serial.println("Space 1: Occupied");
} else {
  digitalWrite(ledPin1, HIGH);
  lcd.setCursor(0, 0);
  lcd.print("Space 1: Vacant");
  Serial.println("Space 1: Vacant");
}
delay(1000);

digitalWrite(trigPin2, LOW);
delayMicroseconds(2);
digitalWrite(trigPin2, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin2, LOW);
duration = pulseIn(echoPin2, HIGH);
distance = duration * 0.034 / 2;

lcd.clear(); // Clear the previous display content
if (distance < 200) {
  digitalWrite(ledPin2, LOW);
  lcd.setCursor(0, 0);
  lcd.print("Space 2: Occupied");
  Serial.println("Space 2: Occupied");
} else {
  digitalWrite(ledPin2, HIGH);
  lcd.setCursor(0, 0);
  lcd.print("Space 2: Vacant");
  Serial.println("Space 2: Vacant");
}
delay(1000);

digitalWrite(trigPin3, LOW);
delayMicroseconds(2);
digitalWrite(trigPin3, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(trigPin3, LOW);
  duration = pulseIn(echoPin3, HIGH);
  distance = duration * 0.034 / 2;

  lcd.clear(); // Clear the previous display content
  if (distance < 200) {
    digitalWrite(ledPin3, LOW);
    lcd.setCursor(0, 0);
    lcd.print("Space 3: Occupied");
    Serial.println("Space 3: Occupied");
  } else {
    digitalWrite(ledPin3, HIGH);
    lcd.setCursor(0, 0);
    lcd.print("Space 3: Vacant");
    Serial.println("Space 3: Vacant");
  }
  delay(1000);
}
```

Diagram.json:

```
{
  "version": 1,
  "author": "Anonymous maker",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": -4.9,
"left": 148.6, "attrs": {} },
    {
      "type": "wokwi-hc-sr04",
      "id": "ultrasonic1",
      "top": -131.76,
      "left": 147.44,
      "attrs": { "distance": "400" }
    },
    {
```

```
    "type": "wokwi-hc-sr04",
    "id": "ultrasonic2",
    "top": -132.9,
    "left": -80.9,
    "attrs": { "distance": "400" }
  },
  {
    "type": "wokwi-hc-sr04",
    "id": "ultrasonic3",
    "top": -132.9,
    "left": 389.5,
    "attrs": { "distance": "400" }
  },
  {
    "type": "wokwi-led",
    "id": "led1",
    "top": -147.6,
    "left": -121,
    "attrs": { "color": "red" }
  },
  {
    "type": "wokwi-led",
    "id": "led2",
    "top": -147.6,
    "left": 109.4,
    "attrs": { "color": "red" }
  },
  {
    "type": "wokwi-led",
    "id": "led3",
    "top": -157.2,
    "left": 349.4,
    "attrs": { "color": "red" }
  },
```

```
    { "type": "wokwi-slide-switch", "id": "sw1", "top": 388.4,
"left": -227.3, "attrs": {} },
    { "type": "wokwi-servo", "id": "servo1", "top": 247.6,
"left": -268.8, "attrs": {} },
    {
      "type": "wokwi-lcd1602",
      "id": "lcd1",
      "top": 534.4,
      "left": 130.4,
      "attrs": { "pins": "i2c" }
    }
  ],
  "connections": [
    [ "esp:TX0", "$serialMonitor:RX", "", [] ],
    [ "esp:RX0", "$serialMonitor:TX", "", [] ],
    [ "ultrasonic3:VCC", "esp:3V3", "red", [ "v0" ] ],
    [ "ultrasonic1:VCC", "esp:3V3", "red", [ "v27.84", "h68.85",
"v161.57" ] ],
    [ "ultrasonic2:VCC", "esp:3V3", "red", [ "v248.34",
"h296.46", "v-61.2" ] ],
    [ "ultrasonic3:GND", "esp:GND.1", "black", [ "v0" ] ],
    [ "ultrasonic1:GND", "esp:GND.1", "black", [ "v7.58",
"h85.62", "v168.49" ] ],
    [ "ultrasonic1:ECHO", "esp:D15", "green", [ "v15.64",
"h87.56", "v153.17" ] ],
    [ "ultrasonic1:TRIG", "esp:D2", "green", [ "v22.89",
"h75.79", "v136.24" ] ],
    [ "ultrasonic3:ECHO", "esp:D5", "green", [ "v0" ] ],
    [ "ultrasonic3:TRIG", "esp:D18", "green", [ "v0" ] ],
    [ "ultrasonic2:GND", "esp:GND.2", "black", [ "v0" ] ],
    [ "led1:C", "esp:GND.2", "black", [ "v0" ] ],
    [ "led2:C", "esp:GND.2", "black", [ "v0" ] ],
    [ "led3:C", "esp:GND.2", "black", [ "v0" ] ],
    [ "led1:A", "esp:D13", "green", [ "v0" ] ],
    [ "led2:A", "esp:D12", "green", [ "v0" ] ],
```

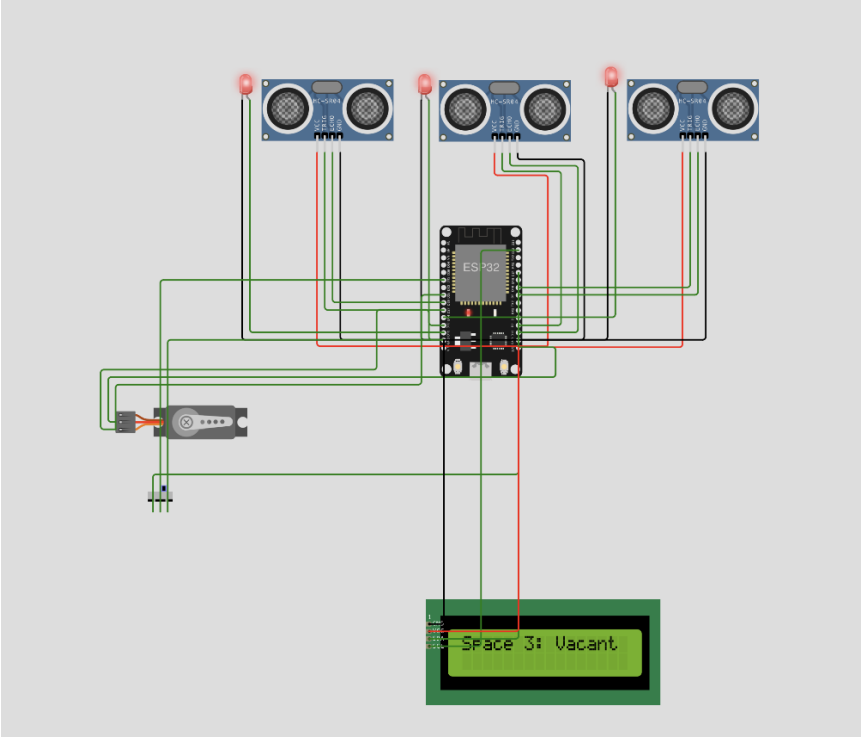
```
    [ "led3:A", "esp:D14", "green", [ "v0" ] ],
    [ "ultrasonic2:TRIG", "esp:D27", "green", [ "v0" ] ],
    [ "ultrasonic2:ECHO", "esp:D26", "green", [ "v0" ] ],
    [ "servo1:PWM", "esp:D25", "green", [ "v-57.4", "h393.6",
"v-172.8" ] ],
    [ "servo1:V+", "esp:3V3", "green", [ "h-9.6", "v-57.5",
"h576", "v-96" ] ],
    [
      "servo1:PWM",
      "esp:GND.2",
      "green",
      [ "h-19.2", "v-76.6", "h355.2", "v-76.8", "h67.2",
"v-19.1" ]
    ],
    [ "sw1:1", "esp:3V3", "green", [ "v-48", "h393.6" ] ],
    [ "sw1:3", "esp:GND.2", "green", [ "v0" ] ],
    [ "sw1:2", "esp:D32", "green", [ "v0" ] ],
    [ "lcd1:SCL", "esp:D22", "green", [ "h67.2", "v0.3" ] ],
    [ "lcd1:SDA", "esp:D21", "green", [ "h0" ] ],
    [ "lcd1:VCC", "esp:3V3", "red", [ "h0" ] ],
    [ "lcd1:GND", "esp:GND.2", "black", [ "h0" ] ]
  ],
  "dependencies": {}
}
```

Raspberry pi Code:

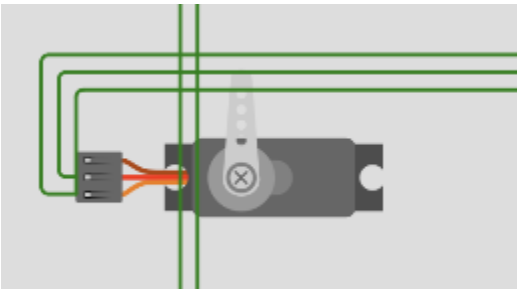
```
import time
import json
import random
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# AWS IoT Core configuration
endpoint = "your-iot-endpoint.amazonaws.com"
```

```
root_ca_path = "path-to-root-ca.pem"
private_key_path = "path-to-private-key.pem"
cert_path = "path-to-certificate.pem"
iot_topic = "sensors_data_topic"

# Initialize the AWS IoT MQTT Client
client = AWSIoTMQTTClient("raspberry-pi-sensor-client")
client.configureEndpoint(endpoint, 8883)
client.configureCredentials(root_ca_path, private_key_path,
cert_path)

# Connect to AWS IoT
client.connect()
while True:
    # Simulate sensor data (replace with your actual sensor
readings)
    sensor_data = {
        "sensor1": random.uniform(0, 100),
        "sensor2": random.uniform(0, 100),
        "sensor3": random.uniform(0, 100)
    }

    # Convert the sensor data to a JSON string
    payload = json.dumps(sensor_data)

    # Publish the sensor data to the AWS IoT Core topic
    client.publish(iot_topic, payload, 1)

    print(f"Published: {payload}")
    time.sleep(5)  # Adjust the frequency of data publishing as
needed

# Disconnect from AWS IoT
client.disconnect()
```

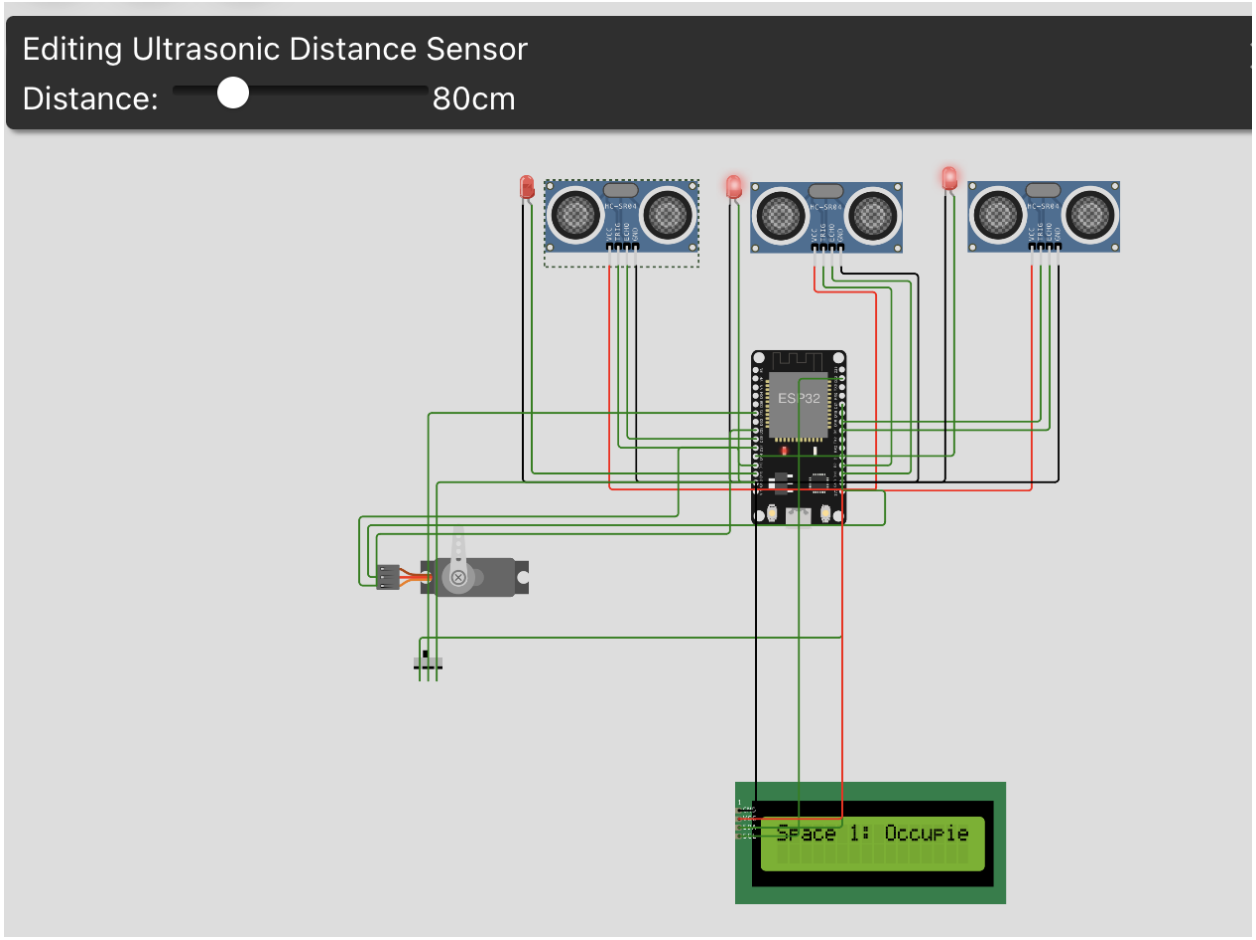
OUTPUT



OPENING OF SERVOMOTOR:



Parking space 1 occupied:



- Light goes off (indicating space unavailable)
- Corresponding output displayed on lcd display