

Project Title: Smart Parking

PHASE 1:

Abstract:

Efficient and smart way to automate the management of the parking system that allocates an efficient parking space using internet of things technology. The IoT provides a wireless access to the system and the user can keep a track of the availability of the parking area. With increase in the population of the vehicles in metropolitan cities, road congestion is the major problem that is being faced.

Problem Statement:

By using ultrasonic sensors be able to keep a record of the number of cars parked inside of a parking garage. Consequently, once a car enters a parking garage followed by a parking space, a ping ultrasonic sensor will then be able to determine if a car is parked in the space or not.

Solution Needed:

Smart parking solutions aim to address the growing urban congestion and parking challenges. These solutions leverage technology to optimize parking space utilization and enhance the overall parking experience. One key element is the use of sensors and cameras to monitor parking spot availability in real-time. When drivers access an app or website, they can instantly find vacant spots, reducing the time spent searching for parking. Additionally, smart parking systems often incorporate mobile payment options, allowing users to pay for their parking digitally, reducing the need for physical payment methods like coins or tickets.

Furthermore, these systems can improve city traffic flow by directing drivers to available parking spaces, reducing the congestion caused by circling vehicles. In the long run, data collected from these systems can also help city planners make informed decisions about parking infrastructure and policies. Overall, smart parking solutions not only simplify the parking process for drivers but also contribute to more efficient and sustainable urban mobility.

Phase 1: Problem Definition and Design Thinking

In this part you will need to understand the problem statement and create a document on what have you understood and how will you proceed ahead with solving the problem. Please think on a design and present in form of a document.

Project Definition:

The project involves integrating IoT sensors into public transportation vehicles to monitor ridership, track locations, and predict arrival times. The goal is to provide real-time transit information to the public through a public platform, enhancing the efficiency and quality of public transportation services. This project includes defining objectives, designing

the IoT sensor system, developing the real-time transit information platform, and integrating them using IoT technology and Python.

Design Thinking:

- **Project Objectives:** Define specific objectives such as real-time parking space monitoring, mobile app integration, and efficient parking guidance.
- **IoT Sensor Design:** Plan the design and deployment of IoT sensors in parking spaces to detect occupancy and availability.
- **Real-Time Transit Information Platform:** Design a mobile app interface that displays real-time parking availability to users.
- **Integration Approach:** Determine how Raspberry Pi will collect data from sensors and update the mobile app.

PHASE 2:

Predictive maintenance Algorithm:

Designing predictive maintenance algorithms begins with a body of data. Often you must manage and process large sets of data, including data from multiple sensors and multiple machines running at different times and under different operating conditions.

You might have access to one or more of the following types of data:

- 1) Real data from normal system operation
- 2) Real data from system operating in a faulty condition
- 3) Real data from system failures (run-to-failure data)

For instance, you might have sensor data from system operation such as temperature, pressure, and vibration. Such data is typically stored as signal or time series data. You might also have text data, such as data from maintenance records, or data in other forms. This data is stored in files, databases, or distributed file systems such as Hadoop.

In many cases, failure data from machines is not available, or only a limited number of failure datasets exist because of regular maintenance being performed and the relative rarity of such incidents. In this case, failure data can be generated from a Simulink model representing the system operation under different fault conditions.

Predictive Maintenance Toolbox provides functionality for organizing, labeling, and accessing such data stored on disk. It also provides tools to facilitate the generation of data from Simulink models for predictive maintenance algorithm development.

Example :

Train Detection or Prediction Model:

At the heart of the predictive maintenance algorithm is the detection or prediction model. This model analyzes extracted condition indicators to determine the current condition of the system (fault detection and diagnosis) or predict its future condition (remaining useful life prediction).

Fault Detection and Diagnosis:

Fault detection and diagnosis relies on using one or more condition indicator values to distinguish between healthy and faulty operation, and between different types of faults. A simple fault-detection model is a threshold value for the condition indicator that is indicative of a fault condition when exceeded. Another model might compare the condition indicator to a statistical distribution of indicator values to determine the likelihood of a particular fault state. A more complex fault-diagnosis approach is to train a classifier that compares the current value of one or more condition indicators to values associated with fault states, and returns the likelihood that one or another fault state is present.

Usage:

When designing your predictive maintenance algorithm, you might test different fault detection and diagnosis models using different condition indicators. Thus, this step in the design process is likely iterative with the step of extraction condition indicators, as you try different indicators, different combinations of indicators, and different decision models. Statistics and Machine Learning Toolbox and other toolboxes include functionality that you can use to train decision models such as classifiers and regression models.

How IoT sensors detect free parking space :

IoT sensors utilise an ultrasonic wave to determine the distance to something. Each sensor is implanted in the parking space surface and finds the distance to the undercarriage of a vehicle if the parking space is full.

3 possible detection conditions:

- If space is occupied: The distance determined to an object by the sensor is 10 to 50 centimetres, four to 20 inches.
- If space is free: The distance determined to an object by the sensor is more than 50 centimetres, about 20 inches.
- If space is dirty: The distance determined to an object by the sensor is less than 10 centimetres, that is, four inches.

- If the condition is “dirty,” it means the sensor may be covered by something or blocked, and the device needs prompt maintenance and cleaning.

The application operates on AWS IoT and AWS Lambda and displays a driver the free spaces in green, full/occupied spaces in red and sensor malfunctions in yellow.

IoT-based smart parking system configuration :

The number of parking spaces available in a parking lot specifies the software and hardware needs for IoT configuration and system architecture. For large parking lots, it is best to use gateways and the LPWAN protocol for the sensors.

Using the LoRaWAN standard is one of the latest IoT trends and the best way to improve the operating hours of an autonomous system by minimising power usage. As per the specifications of the LoRa Alliance, this decreases the demand to substitute the batteries.

Battery life is increased to five years before replacement.

Sensors for IoT-based smart parking:

Smart parking sensor types have ultrasonic, electromagnetic field detection, and infrared.

Ultrasonic: The sensing accuracy is improved by using ultrasound to know the measurement. The drawback of this type of sensor is the possibility of blockage by dirt.

Electromagnetic field detection: This sensor notices slight changes in the magnetic field whenever metal objects come closer to the sensor.

Infrared: This type of sensor measures the changes happening in the surrounding temperature and identifies the motion.

Parking 4.0: Future and opportunities in smart cities:

Adopting smart parking systems is expected to grow because the technology is beneficial and brings helpful changes in daily life.

Augmented reality technology is of great help for large-scale parking lots as it can create a mapping function overlay on top of authentic images captured by a smartphone. These ARbased outdoor and indoor navigation systems can direct drivers with a virtual path to their parked cars.

Another innovation utilises visual image processing to capture the license number of a vehicle to identify it with the support of Optical Character Recognition technology. Then, it automatically opens the gate to the parking lot, and the system helps the driver to a suitable parking space.

So, we can conclude that the future of smart parking systems is quite promising.

Technologies behind this possible solution are Artificial Intelligence, IoT, Machine Learning, and Augmented Reality. These are also responsible for digital transformation for businesses under the

“Industry 4.0” term. By potentially using these innovations, Parking 4.0 will enhance the efficiency of the parking system by solving urbanisation issues.

PHASE 3 (Development Part 1)

IoT can be used in parking management:

The rapid demand for parking spaces stems from two major trends – urbanization and car ownership. In a short period, the IoT-based parking management system has positively impacted all stakeholders involved in the process.

For instance, drivers can plan trips and commutes keeping slot occupancy in mind. Enforcement agencies can detect and evaluate the gravity of parking rules violations in seconds.

Parking facility managers can optimize the use of the personnel and the available space. Adding to that thought, IoT can be used in parking management in the following ways:

- Create “smart” parking meters that accept payments via credit card or smartphone and provide real-time information on space availability.
- Extend the operating time of an autonomous parking system by consuming less power with the help of LoRaWAN® networking standard.
- Send real-time data from IoT sensors to a cloud server and share it with users to provide them with the details of free-parking spaces.
- Determine the number and position of vacant parking spaces precisely using IoT sensors.
- Reduce congestion and improve traffic flow by using sensors to monitor parking space availability and direct drivers to available spaces.
- IoT gateway and the LPWAN protocol help connect IoT devices and sensors in sizable parking lots.
- The parking management company can use a cloud-based IoT dashboard. It uses gathered sensor data to mine intelligible insights that are visually appealing and provides a clear picture of the parking facility.
- Optimize parking operations and improve security by tracking vehicles entering and exiting a parking facility.
- Mobile applications can display parking data to drivers and send alerts in case of a security breach.
- Monitor environmental conditions in parking garages, such as air quality and temperature, to ensure the safety of occupants.

SOURCE CODE:

Import time

```

Import RPi.GPIO as GPIO

Import time

Import os,sys

From urllib.parse import urlparse

Import paho.mqtt.client as paho

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

","


Define pin for lcd

","


# Timing constants

E_PULSE = 0.0005

E_DELAY = 0.0005

Delay = 1

# Define GPIO to LCD mapping

LCD_RS = 7

LCD_E = 11

LCD_D4 = 12

LCD_D5 = 13

LCD_D6 = 15

LCD_D7 = 16

Slot1_Sensor = 29

Slot2_Sensor = 31

GPIO.setup(LCD_E, GPIO.OUT) # E

GPIO.setup(LCD_RS, GPIO.OUT) # RS

GPIO.setup(LCD_D4, GPIO.OUT) # DB4

GPIO.setup(LCD_D5, GPIO.OUT) # DB5

GPIO.setup(LCD_D6, GPIO.OUT) # DB6

GPIO.setup(LCD_D7, GPIO.OUT) # DB7

GPIO.setup(slot1_Sensor, GPIO.IN)

GPIO.setup(slot2_Sensor, GPIO.IN)

# Define some device constants

LCD_WIDTH = 16 # Maximum characters per line

LCD_CHR = True

```

```
LCD_CMD = False
```

```
LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
```

```
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
```

```
LCD_LINE_3 = 0x90 # LCD RAM address for the 3rd line
```

```
Def on_connect(self, mosq, obj, rc):
```

```
Self.subscribe("Fan", 0)
```

```
Def on_publish(mosq, obj, mid):
```

```
Print("mid: " + str(mid))
```

```
Mqttc = paho.Client() # object declaration
```

```
# Assign event callbacks
```

```
Mqttc.on_connect = on_connect
```

```
Mqttc.on_publish = on_publish
```

```
url_str = os.environ.get('CLOUDMQTT_URL', 'tcp://broker.emqx.io:1883')
```

```
url = urlparse(url_str)
```

```
mqtcc.connect(url.hostname, url.port)
```

```
""
```

```
Function Name :lcd_init()
```

```
Function Description : this function is used to initialize lcd by sending the different commands
```

```
""
```

```
Def lcd_init():
```

```
# Initialise display
```

```
Lcd_byte(0x33,LCD_CMD) # 110011 Initialise
```

```
Lcd_byte(0x32,LCD_CMD) # 110010 Initialise
```

```
Lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
```

```
Lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
```

```
Lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
```

```
Lcd_byte(0x01,LCD_CMD) # 000001 Clear display
```

```
Time.sleep(E_DELAY)
```

```
""
```

```
Function Name :lcd_byte(bits ,mode)
```

Fuction Name :the main purpose of this function to convert the byte data into bit and send to lcd port

“”

```
Def lcd_byte(bits, mode):
```

```
# Send byte to data pins
```

```
# bits = data
```

```
# mode = True  for character
```

```
#      False for command
```

```
GPIO.output(LCD_RS, mode) # RS
```

```
# High bits
```

```
GPIO.output(LCD_D4, False)
```

```
GPIO.output(LCD_D5, False)
```

```
GPIO.output(LCD_D6, False)
```

```
GPIO.output(LCD_D7, False)
```

```
If bits&0x10==0x10:
```

```
GPIO.output(LCD_D4, True)
```

```
If bits&0x20==0x20:
```

```
GPIO.output(LCD_D5, True)
```

```
If bits&0x40==0x40:
```

```
GPIO.output(LCD_D6, True)
```

```
If bits&0x80==0x80:
```

```
GPIO.output(LCD_D7, True)
```

```
# Toggle 'Enable' pin
```

```
Lcd_toggle_enable()
```

```
# Low bits
```

```
GPIO.output(LCD_D4, False)
```

```
GPIO.output(LCD_D5, False)
```

```
GPIO.output(LCD_D6, False)
```

```
GPIO.output(LCD_D7, False)
```

```
If bits&0x01==0x01:
```

```
GPIO.output(LCD_D4, True)
```



```

If bits&0x02==0x02:
GPIO.output(LCD_D5, True)
If bits&0x04==0x04:
GPIO.output(LCD_D6, True)
If bits&0x08==0x08:
GPIO.output(LCD_D7, True)

# Toggle 'Enable' pin
Lcd_toggle_enable()
""

Function Name : lcd_toggle_enable()
Function Description: basically this is used to toggle Enable pin
""

Def lcd_toggle_enable():
# Toggle enable
Time.sleep(E_DELAY)
GPIO.output(LCD_E, True)
Time.sleep(E_PULSE)
GPIO.output(LCD_E, False)
Time.sleep(E_DELAY)
""

Function Name :lcd_string(message,line)
Function Description :print the data on lcd
""

Def lcd_string(message,line):
# Send string to display

Message = message.ljust(LCD_WIDTH," ")

Lcd_byte(line, LCD_CMD)

For I in range(LCD_WIDTH):
Lcd_byte(ord(message[i]),LCD_CHR)

```

```
Lcd_init()
Lcd_string("welcome ",LCD_LINE_1)
Time.sleep(0.5)
Lcd_string("Car Parking ",LCD_LINE_1)
Lcd_string("System ",LCD_LINE_2)
Time.sleep(0.5)
Lcd_byte(0x01,LCD_CMD) # 000001 Clear display
# Define delay between readings
Delay = 5
```

While 1:

```
# Print out results
```

```
Rc = mqttc.loop()
```

```
Slot1_status = GPIO.input(slot1_Sensor)
```

```
Time.sleep(0.2)
```

```
Slot2_status = GPIO.input(slot2_Sensor)
```

```
Time.sleep(0.2)
```

```
If (slot1_status == False):
```

```
Lcd_string("Slot1 Parked ",LCD_LINE_1)
```

```
Mqttc.publish("slot1","1")
```

```
Time.sleep(0.2)
```

```
Else:
```

```
Lcd_string("Slot1 Free ",LCD_LINE_1)
```

```
Mqttc.publish("slot1","0")
```

```
Time.sleep(0.2)
```

```
If (slot2_status == False):
```

```
Lcd_string("Slot2 Parked ",LCD_LINE_2)
```

```
Mqttc.publish("slot2","1")
```

```
Time.sleep(0.2)
```

```
Else:
```

```
Lcd_string("Slot2 Free ",LCD_LINE_2)
```

```
Mqttc.publish("slot2","0")
```

```
Time.sleep(0.2)
```

SENSORS:

ULTRASONIC SENSOR:

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves. An ultrasonic sensor uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity. It is used to check whether there is a vehicle in a particular slot or not.

Ultrasonic sensor is installed on the ceiling above a parking space and the distance from the ceiling to the ground is measured using a backward echo wave from the ultrasonic sensor. This distance is used as a basic value to determine the vehicle parking status.

IR SENSOR:

IR sensor is an electronic device, that emits the light in order to sense some object of the surroundings. An IR sensor can measure the heat of an object as well as detect the motion. Usually, in the infrared spectrum, all the objects radiate

Working principle of an infrared sensor is similar to the object detection sensor. This sensor includes an IR LED & an IR Photodiode, so by combining these two can be formed as a photo-coupler otherwise optocoupler. The physics laws used in this sensor are Planck's radiation, Stephan Boltzmann & Wien's displacement.

Senses the incoming car then the signal is transmitted to the DC motor to open the gate. The display is placed at the entrance to show the available parking space. An IR sensor 1 is placed at the entrance to count the entering cars. Once the IR sensor

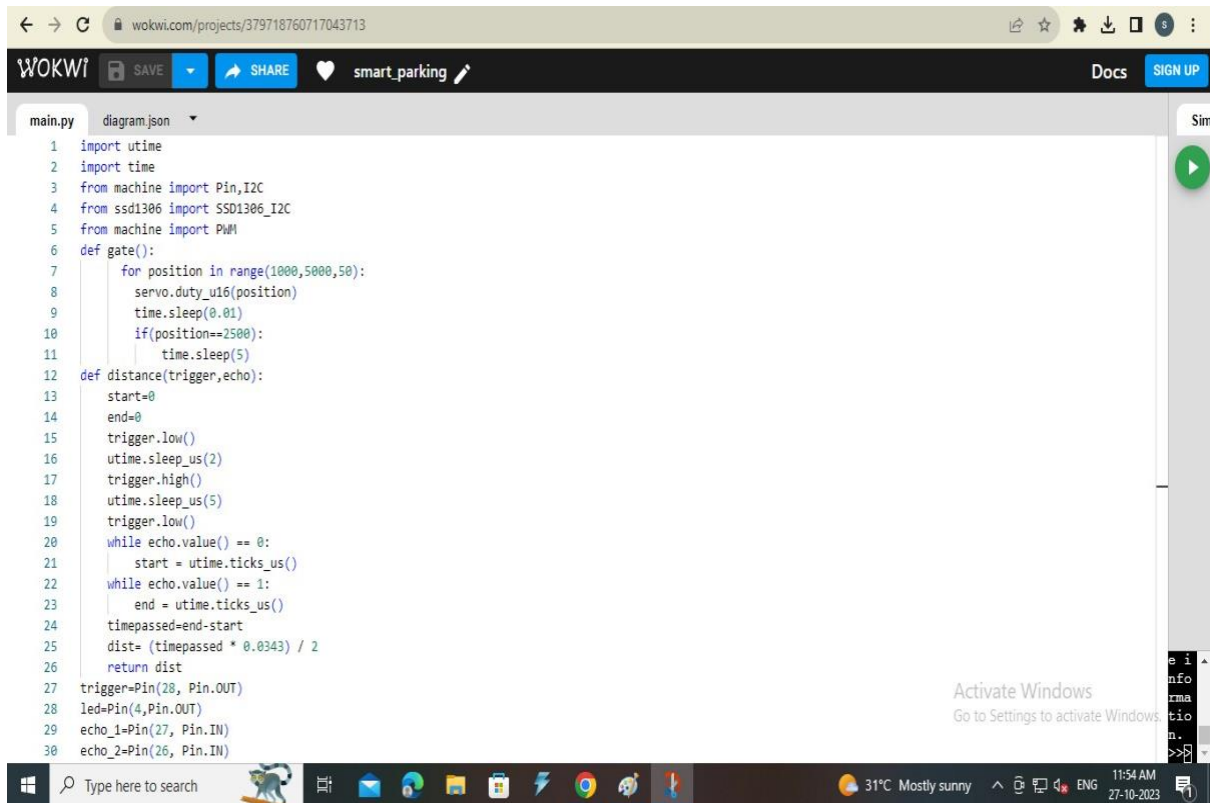
MAGNETOMETER:

Magnetic parking sensors should be placed one-third of the way in front of the parking space. The sensor can detect parked vehicles with high accuracy regardless of parking mode and vehicle type

RADAR:

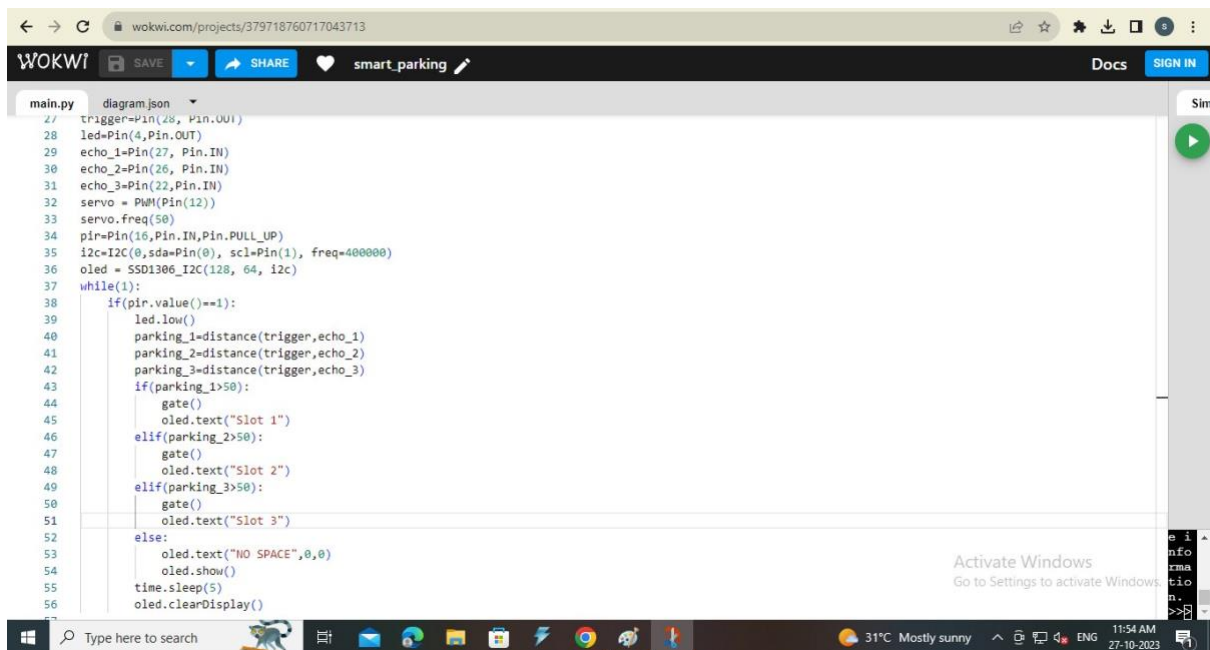
Smart Parking Sensor Technology is a radar sensor device that allows the detection of parking availability indoors and outdoors. Wireless detection of parking spot occupancy. Best accuracy in the market: radar and magnetic technology combination

PHASE 4 (Development part 2)



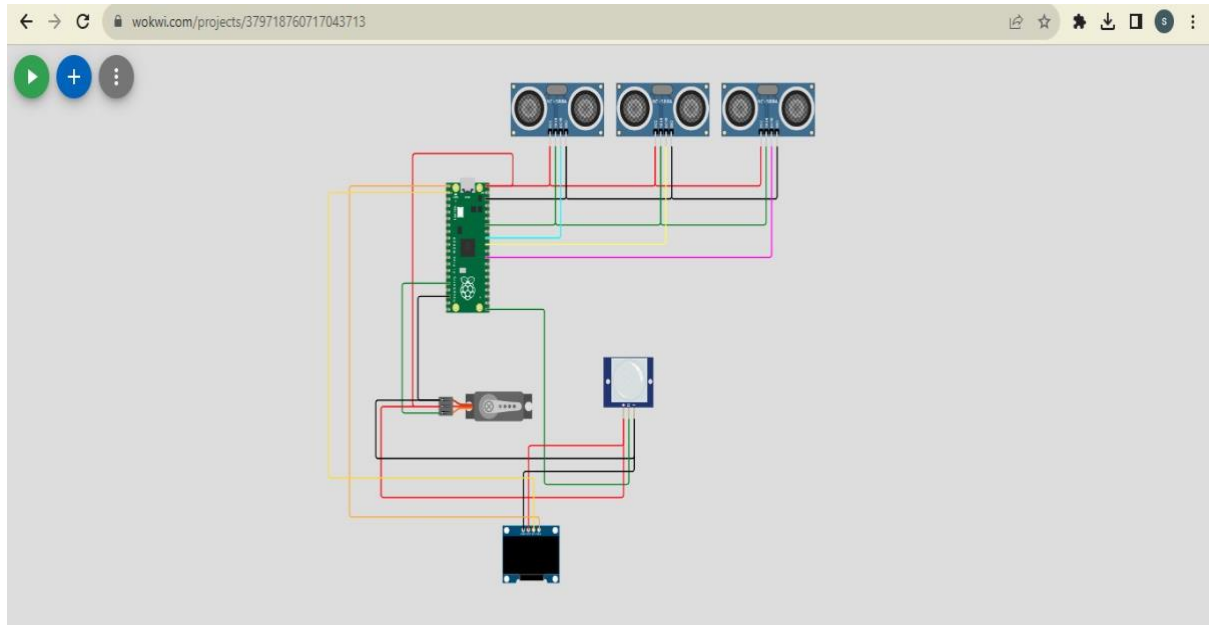
The screenshot shows the Wokwi IDE interface with a project named "smart_parking". The code in `main.py` defines functions for controlling a servo motor and measuring distance using an ultrasonic sensor.

```
1 import utime
2 import time
3 from machine import Pin, I2C
4 from ssd1306 import SSD1306_I2C
5 from machine import PWM
6 def gate():
7     for position in range(1000,5000,50):
8         servo.duty_u16(position)
9         time.sleep(0.01)
10        if(position==2500):
11            time.sleep(5)
12 def distance(trigger,echo):
13     start=0
14     end=0
15     trigger.low()
16     utime.sleep_us(2)
17     trigger.high()
18     utime.sleep_us(5)
19     trigger.low()
20     while echo.value() == 0:
21         start = utime.ticks_us()
22     while echo.value() == 1:
23         end = utime.ticks_us()
24     timepassed=end-start
25     dist= (timepassed * 0.0343) / 2
26     return dist
27 trigger=Pin(28, Pin.OUT)
28 led=Pin(4,Pin.OUT)
29 echo_1=Pin(27, Pin.IN)
30 echo_2=Pin(26, Pin.IN)
```



The screenshot shows the continuation of the Python code in `main.py`, which sets up the hardware and enters a main loop to manage parking slots.

```
27 trigger=Pin(28, Pin.OUT)
28 led=Pin(4,Pin.OUT)
29 echo_1=Pin(27, Pin.IN)
30 echo_2=Pin(26, Pin.IN)
31 echo_3=Pin(22,Pin.IN)
32 servo = PWM(Pin(12))
33 servo.freq(50)
34 pir=Pin(16,Pin.IN,Pin.PULL_UP)
35 i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
36 oled = SSD1306_I2C(128, 64, i2c)
37 while(1):
38     if(pir.value()==1):
39         led.low()
40         parking_1=distance(trigger,echo_1)
41         parking_2=distance(trigger,echo_2)
42         parking_3=distance(trigger,echo_3)
43         if(parking_1>50):
44             gate()
45             oled.text("Slot 1")
46         elif(parking_2>50):
47             gate()
48             oled.text("Slot 2")
49         elif(parking_3>50):
50             gate()
51             oled.text("Slot 3")
52     else:
53         oled.text("NO SPACE",0,0)
54         oled.show()
55         time.sleep(5)
56         oled.clearDisplay()
```



SOURCE CODE :

```
import utime
import time
from machine import Pin
from machine import PWM

def gate():
    for position in range(1000,5000,50):
        servo.duty_u16(position)
        time.sleep(0.01)
        if(position==2500):
            time.sleep(5)

def distance(trigger,echo):
    start=0
    end=0
    trigger.low()
    utime.sleep_us(2)
    trigger.high()
    utime.sleep_us(5)
    trigger.low()
    while echo.value() == 0:
```

```

        start = utime.ticks_us()
while echo.value() == 1:
    end = utime.ticks_us()
    timepassed=end-start
    dist= (timepassed * 0.0343) / 2
    return dist

trigger=Pin(28, Pin.OUT)
led=Pin(4,Pin.OUT)
echo_1=Pin(27, Pin.IN)
echo_2=Pin(26, Pin.IN)
echo_3=Pin(22,Pin.IN)
servo = PWM(Pin(12))
servo.freq(50)
pir=Pin(16,Pin.IN,Pin.PULL_UP)

while(1):
    if(pir.value()==1):
        led.low()
        parking_1=distance(trigger,echo_1)
        parking_2=distance(trigger,echo_2)
        parking_3=distance(trigger,echo_3)
        if(parking_1>50):
            gate()
        elif(parking_2>50):
            gate()
        elif(parking_3>50):
            gate()
        time.sleep(5)

```

WOKWI LINK: <https://wokwi.com/projects/379718760717043713>

The readme file gives a detailed information about the working of Smart Parking using IOT:

<https://github.com/AkashKasimani/IOT-Naan-Mudhalvan/blob/main/README.md>