KGiSL INSTITUTE OF TECHNOLOGY

(Approved By AICTE, New Delhi, Affiliate to Anna University

Recognized by UGC, Accredited by NBA(IT)

265, KGISL Campus, Thudiyalur Road, Saravanampatti, Coimbatore-641035.)

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

NAAN MUDHALVAN - INTERNET OF THINGS

SMART PARKING

NAME: ASWITHA.R

REG NO: 711721243014

NM ID: au711721243014

TEAM MENTOR: Mr. Mohankumar M

TEAM EVALUATOR: Ms. Akilandeeshwari M

Phase 3: Development Part 1

Building a smart parking system using IoT sensors and Raspberry Pi integration

Materials and Components Needed:

1. Raspberry Pi (with Wi-Fi and Bluetooth capabilities)
2. IoT Sensors (e.g., ultrasonic or PIR motion sensors)
3. Power source for Raspberry Pi
4. Breadboard and jumper wires
5. LEDs (optional, for visual indicators)
6. MicroSD card with Raspbian OS
7. Internet connection for Raspberry Pi
8. Python programming environment on the Raspberry Pi
9. Cloud platform (optional, for remote data storage and access)

Procedure:

1. Set up Raspberry Pi:

- Install Raspbian OS on the microSD card.
- Configure Wi-Fi and connect the Raspberry Pi to the internet.
- Install necessary libraries for sensor communication (e.g., GPIO for Python).

2. Connect IoT Sensors:

- Connect the IoT sensors to the Raspberry Pi using jumper wires. Depending on the sensor type, you'll need to connect power (3.3V or 5V), ground (GND), and data pins to the appropriate GPIO pins on the Raspberry Pi.
- Ensure you've connected the sensors correctly and securely to the GPIO pins.

3. Write Python Code:

- Developing Python code to interface with the IoT sensors. Use the GPIO library to read sensor data.
- Implementing code to detect parking space occupancy. For example, if you're using an ultrasonic sensor, you can measure distance. If the distance is below a certain threshold, consider the space occupied.
- Add code for real-time data processing.

4. Data Processing and Decision Making:

- Implementing logic to decide if a parking space is vacant or occupied based on the sensor data.
- To set up thresholds and timers to account for sensor noise and transient changes in readings.

5. Display and Communication:

- LEDs or other visual indicators to show the occupancy status of each parking space.
- Set up communication with a user interface (e.g., a website or mobile app) to display the parking availability information to users.

6. Optional: Cloud Integration

- To store and access data remotely, integrating the system with a cloud platform (e.g., AWS, Azure, Google Cloud).
- Send sensor data to the cloud for storage and real-time monitoring**.**

7. User Interface:

- Creating a user-friendly interface for users to check parking availability.
- Display real-time information about parking spaces on a website or mobile app.

8. Testing and Calibration:

- Thoroughly testing the system to ensure it accurately detects parking space occupancy.
- Calibrate the sensors as needed to reduce false positives or negatives.

9. Deployment:

- Install the sensors and Raspberry Pi in the parking facility.
- Ensure reliable power sources and internet connectivity.

10. Maintenance and Updates:

- Regularly maintain and update the system as needed.
- Monitor for hardware failures or connectivity issues.

```
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 3nd 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)
mqttc.connect(url.hostname, url.port)

'''
Function Name :lcd_init()
Function Description : this function is used to initialized 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)
```

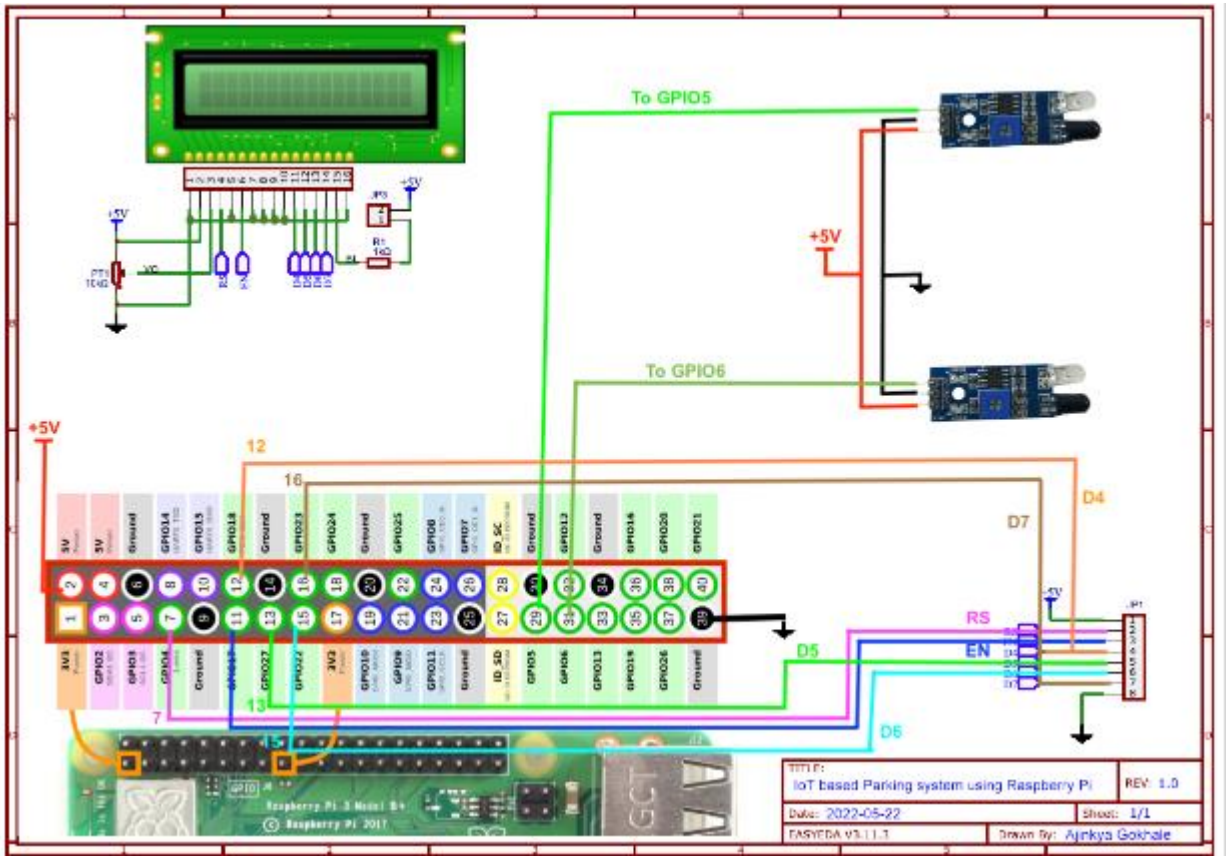
```
  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

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)
```



Schematic for IoT based Smart Parking Sensor

CONCLUSION:

In conclusion, building a smart parking system using IoT sensors and Raspberry Pi integration is a valuable project that offers solutions to urban parking challenges. By following the step-by-step procedure outlined above, we can create a reliable and efficient parking management system.