SMART PARKING(IOT)

NAME: MATHANKUMAR H

NM: AU2272120024

EMAIL: wwwmathanrasig55@gmail.com

PHASE:3

TABLE OF CONTENTS

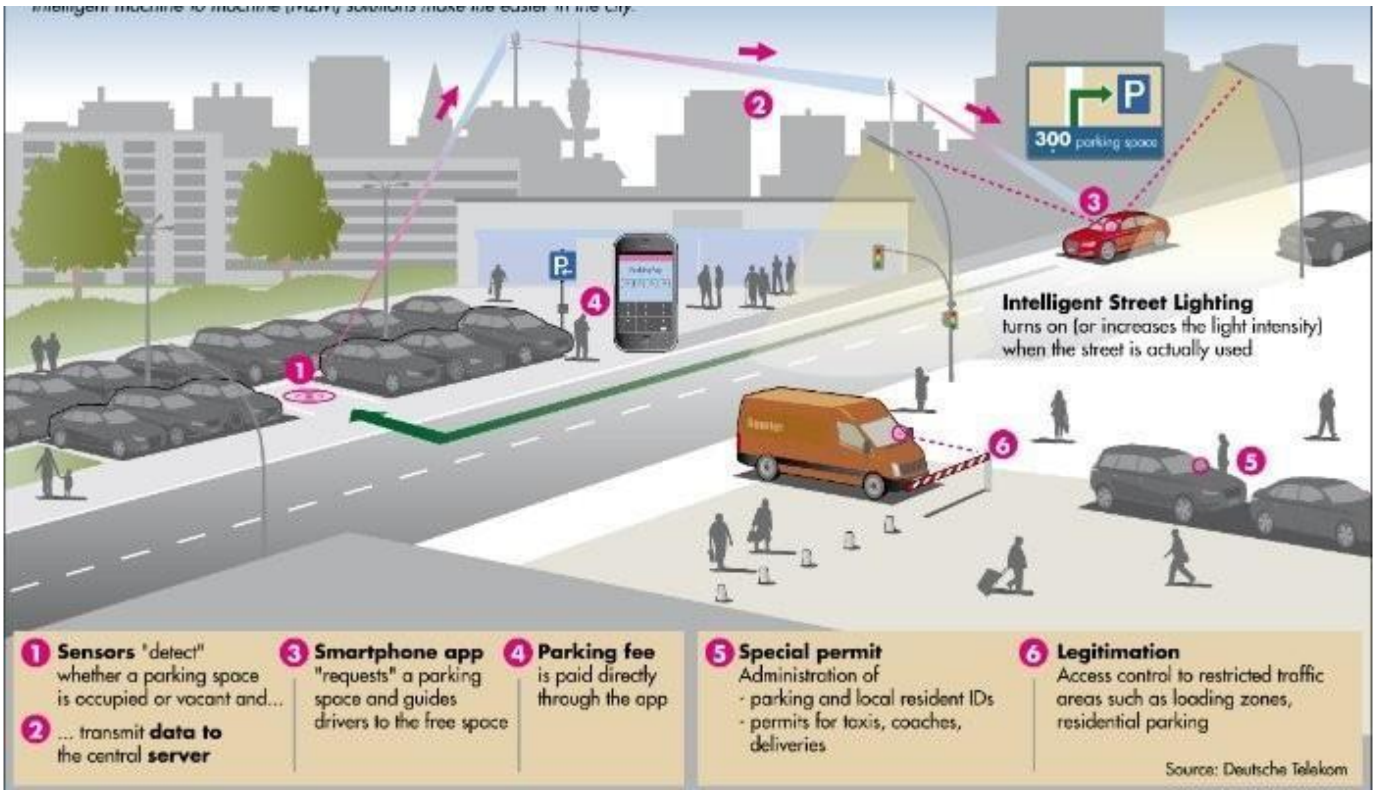
Introduction 3

Development Part 1 4

Program Script.......................................7

Conclusion...14

INTRODUCTION:



Building the IoT sensor system with Raspberry Pi integration for parking space occupancy detection involves several steps.

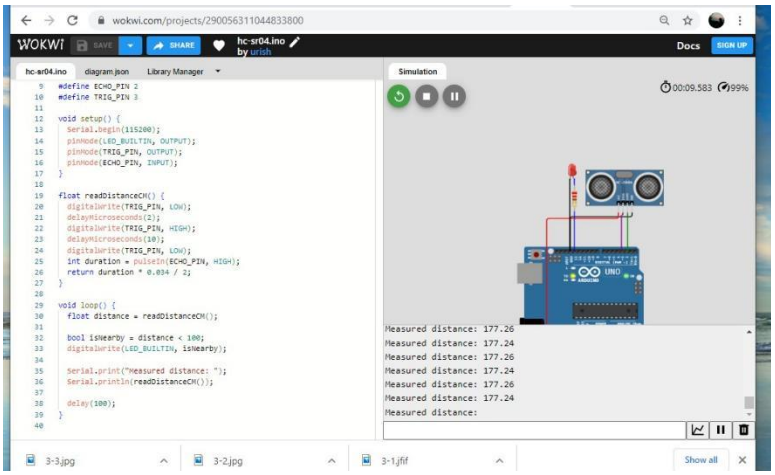
In this overview, I'll guide you through the initial setup. Keep in mind that the specific hardware and software requirements may vary based on your project and sensor choices.

DEVELOPMENT PART 1

Hardware and Components:

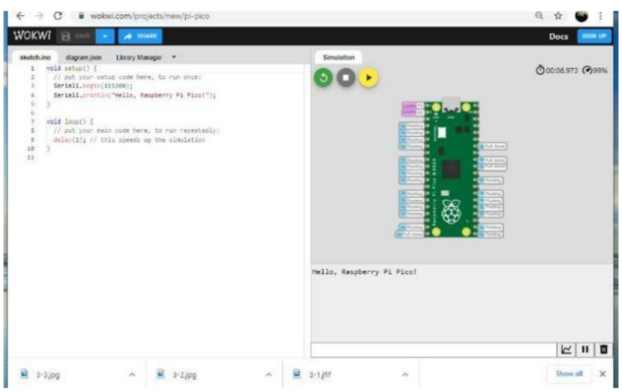
Ultrasonic Sensors:

You'll need ultrasonic sensors to detect parking space occupancy. These sensors measure distances using sound waves and are suitable for this application.



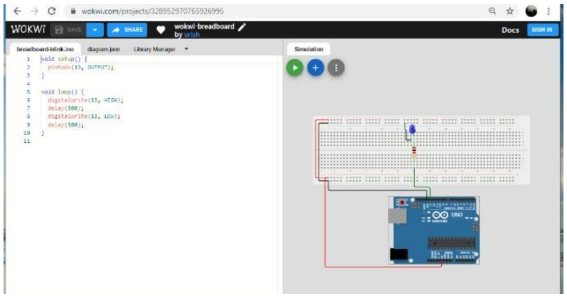
Raspberry Pi:

Use a Raspberry Pi (e.g., Raspberry Pi 3 or 4) as the central controller to collect sensor data and communicate with the IoT platform.



Breadboard and Jumper Wires:

These are essential for connecting the sensors and Raspberry Pi.



Software Setup:

Raspberry Pi OS:
Install the Raspberry Pi OS on your Raspberry Pi. Make sure it's up-to-date with the latest software updates.

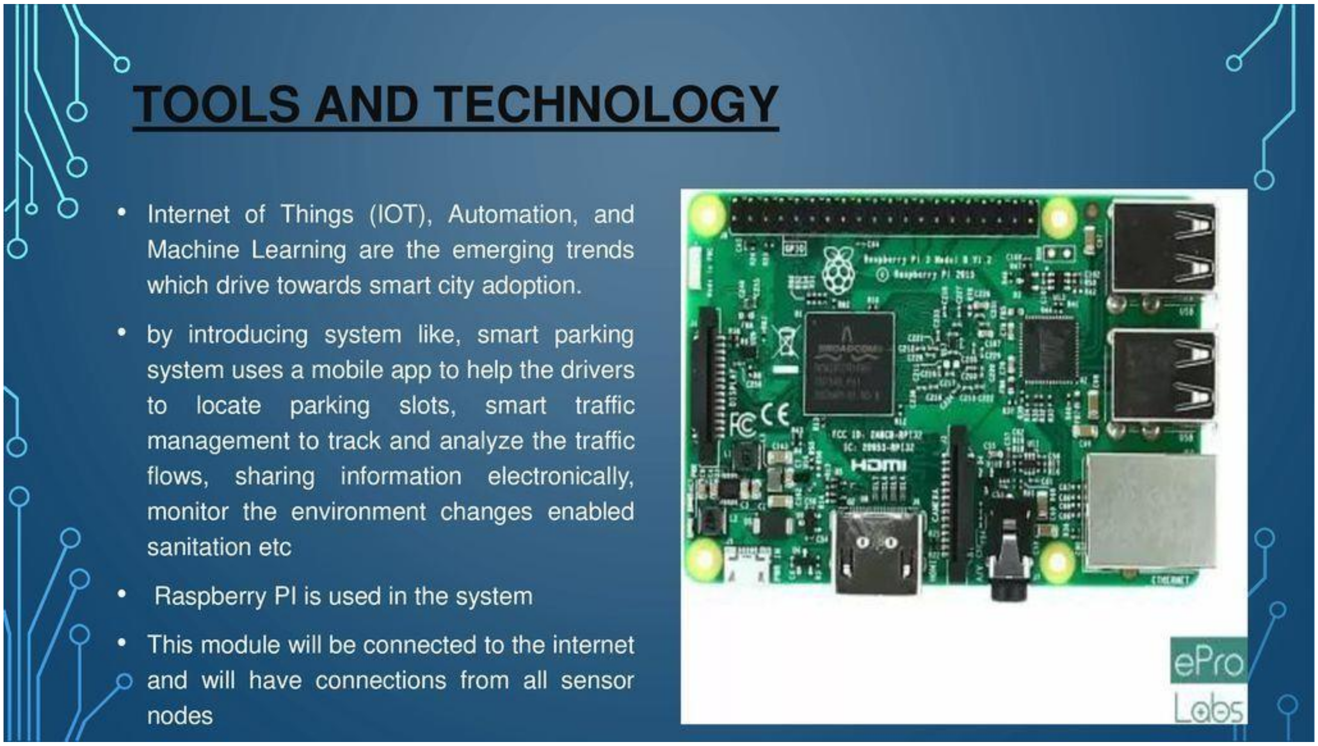
Python:
Python is commonly used for IoT projects. Ensure Python is installed on your Raspberry Pi.

GPIO Library:
You may need to install GPIO libraries to interact with the Raspberry Pi's GPIO pins. Popular options include RPi.GPIO and GPIO Zero.

Programming:

ARDUINO SKETCH FOR THE ESP32 THAT READS THE DISTANCE DATA FROM THE ULTRASONIC SENSORS.

```
 #include <Ultrasonic.h>
Ultrasonic sensor 1(GPIO_TRIGGER1, GPIO_ECHO1);
Ultrasonic sensor2(GPIO_TRIGGER2, GPIO_ECH02);
/I Add more sensors if needed
void setup () {
Serial. begin (115200);
}
void loop() {
long distance 1 = sensor1.read();
long distance2 = sensor2.read();
I Read distances from more sensors if needed
I Process distance data and manage parking spaces here
delay(1000); // Delay for better readability
}
```



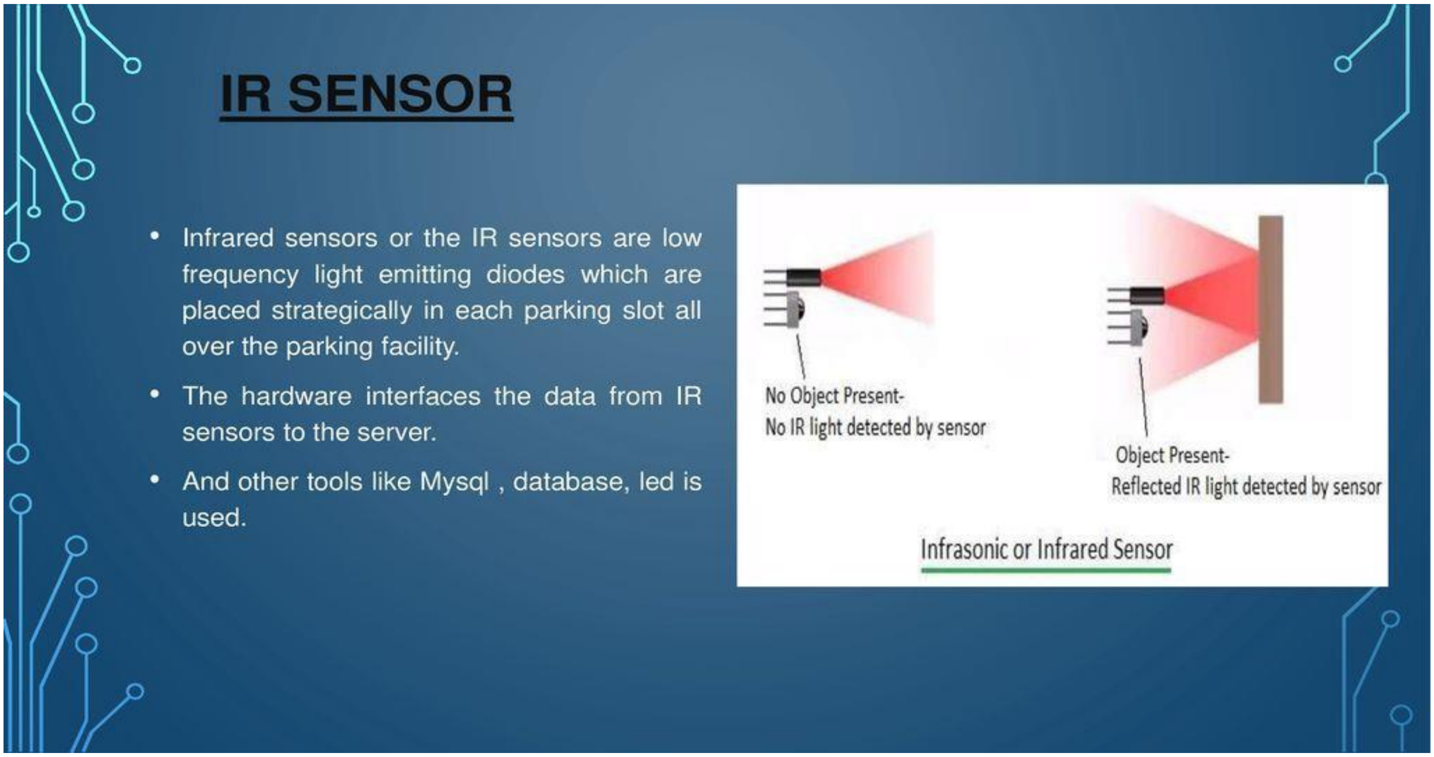
Wiring and Connection:

Connect the ultrasonic sensors to the Raspberry Pi using jumper wires.

Typically, ultrasonic sensors have four pins: VCC (power), GND (ground), TRIG (trigger), and ECHO (echo).

Connect the VCC and GND pins of the sensors to the appropriate power and ground pins on the Raspberry Pi.

Connect the TRIG and ECHO pins of the sensors to GPIO pins on the Raspberry Pi.
Note down the GPIO pin numbers used for each sensor, as you'll need this information in your Python code.



Python Script:

Write a Python script on raspberry pi to collect data from sensors and send it to the cloud or mobile app server.

```
import RPi.GPIO as io
import time
import socket
import sys
from _thread import *
import threading
import cv2
import  imutils
import numpy as np
import pytesseract
from PIL import Image
import time


def check():
    global counter
    ret, frame = cap.read()
    frame = cv2.resize(frame, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)
    print("Taking a photo")
    img = frame
```

```
    img = cv2.resize(img, (620,480))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.bilateralFilter(gray, 11, 17, 17)
    edged = cv2.Canny(gray, 30, 200)
    cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:10]
    screenCnt = None
    for c in cnts:
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.018 * peri, True)
        if len(approx) == 4:
            screenCnt = approx
            break
    if screenCnt is None:
        detected = 0
        print("No Contour detected")
    else:
        detected = 1
    if detected == 1:
        cv2.drawContours(img, [screenCnt], -1, (0, 255, 0), 3)
        mask = np.zeros(gray.shape,np.uint8)
        new_image = cv2.drawContours(mask,[screenCnt],0,255,-1,)
        new_image = cv2.bitwise_and(img,img,mask=mask)

        (x, y) = np.where(mask == 255)
        (topx, topy) = (np.min(x), np.min(y))
        (bottomx, bottomy) = (np.max(x), np.max(y))
        Cropped = gray[topx:bottomx+1, topy:bottomy+1]
        text = pytesseract.image_to_string(Cropped, config='--psm 11')
        print(text)
        if(len(clients)>0 and len(clients[0].license_num)>0):
            if (text == clients[0].license_num or clients[0].license_num in text):
                counter = 1
                print("Detecteddddddddddddddddddddddddddd")
                write(clients[0].conn,"license_status:Received")


def start():
    event = threading.Timer(interval, start).start()
    check()


def write(conn, msg):
    try:
        conn.sendall((msg+'\n').encode('utf-8'))
    except:
        print("Error-sending-msg")
        io.cleanup()
        Sys.exit()
```

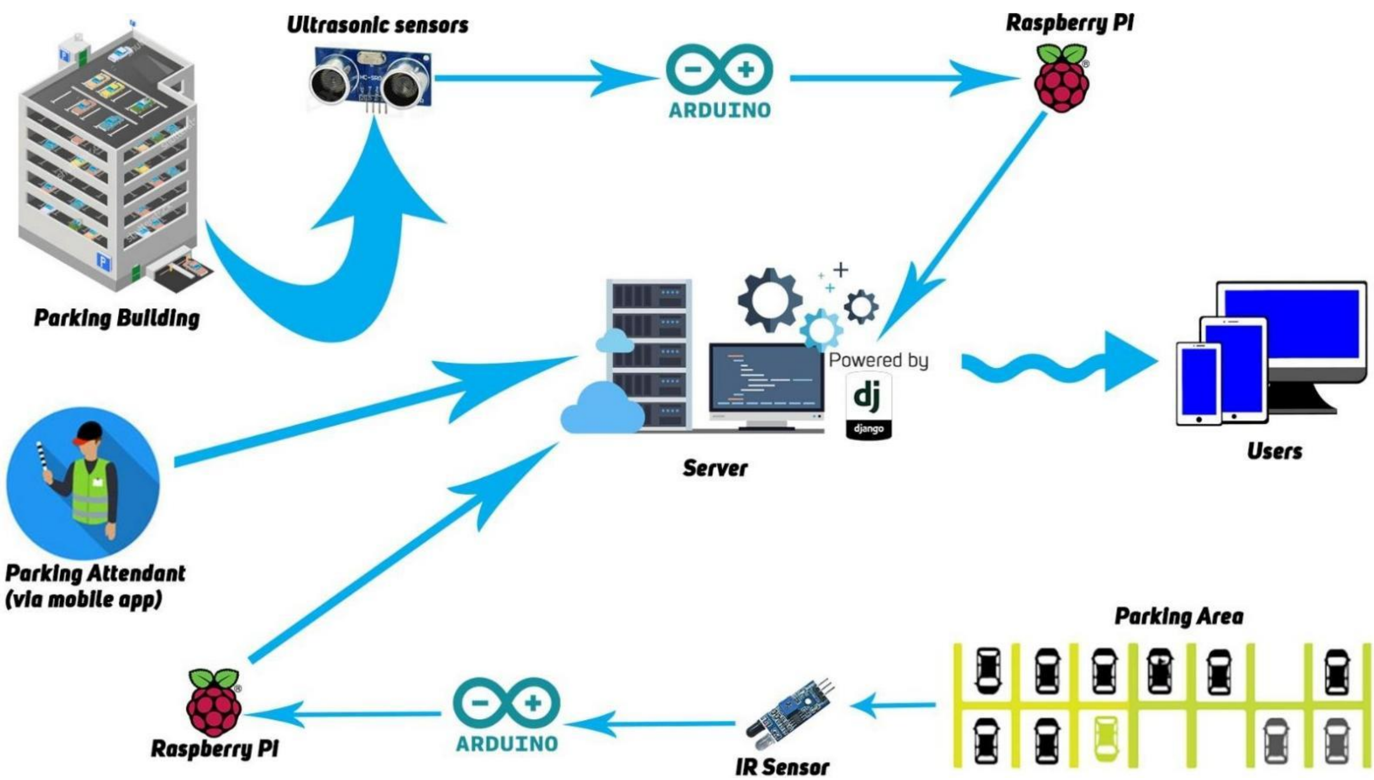
```
def read(conn):
    temp = ''
    while True:
        try:
            data = conn.recv(1024)
            data = data.decode("utf-8")
            if(temp!=data):
                print('client:',data)
                if("license" in data):
                    a = data.split(":")
                    clients[0].license_num = a[1]
                    clients[0].entry_status = True
                    print(clients[0].license_num)
                temp = data

                #start_new_thread(write,(conn, "license_status:Received",))


        except:
            continue




def init(sock):

    try:
        while True:
            try:
                print("waiting for connection")
                conn, client_address = sock.accept()
                print('connection from',client_address)
                client = Client(conn)
                clients.append(client)
                start_new_thread(read,(conn,))
                write(conn,'connected')
                write(conn,'vacant:4')
            except:
                sock.close()
    finally:
        conn.close()
        sock.close()
    conn.close()
    sock.close()



class Client:
```

```
    def__init_(self, conn):
        self.license_num = ''
        self.entry_status = False
        self.payment_status = True
        self.conn = conn
    def setLicenseNum(num):
        self.license_num = num
    def getLicenseNum():
        return self.license_num
    def setEntryStatus(status):
        self.entry_status = status
    def getEntryStatus():
        return self.entryStatus
    def setPaymentStatus(status):
        self.payment_status = status
    def getPaymentStatus():
        return self.payment_status

vacant = 4
counter = 0

interval = 1
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    raise IOError("Cannot open webcam")


servoPIN = 17
io.setmode(io.BCM)
io.setup(servoPIN, io.OUT)
clients = []

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print(socket.gethostname())
server_address = ('192.168.43.244',10000)
print('starting up on %s port %s' % server_address)
sock.bind(server_address)
sock.listen(10)
start_new_thread(init,(sock,))
start()


p = io.PWM(servoPIN, 50)
#GPIO.cleanup()# GPIO 17 for PWM with 50Hz
p.start(2.5) # Initialization

io.setwarnings(False)
io.setmode(io.BCM)

io.setup(12,io.IN)
```

```
io.setup(20, io.OUT)
io.setup(21, io.OUT)

io.setup(6,io.IN)
io.setup(16, io.OUT)
io.setup(26, io.OUT)

io.setup(22,io.IN)
io.setup(1, io.OUT)
io.setup(7, io.OUT)

io.setup(23,io.IN)
io.setup(8, io.OUT)
io.setup(11, io.OUT)

io.setup(18,io.IN)
io.setup(4, io.IN)


#io.cleanup()

temp = list()
temp.append(1)
temp.append(1)
temp.append(1)
temp.append(1)

try:

    while True:
        t = vacant
        vacant = 0


            #start_new_thread(write,(clients[0].conn,'vacant:'+str(vacant),))


        if(io.input(12)==True):
            temp[0]=0
            io.output(20, True)
            io.output(21,False)

        else:
            temp[0]=1
            io.output(20, False)
            io.output(21,True)
            #vacant -= 1
        if(io.input(6)==True):
            temp[1]=0
            io.output(16, True)
```

```
    io.output(26,False)

else:
    temp[1]=1
    io.output(16, False)
    io.output(26,True)
    #vacant -= 1
if(io.input(23)==True):
    temp[2]=0
    io.output(1, True)
    io.output(7,False)

else:
    temp[2]=1
    io.output(1, False)
    io.output(7,True)
    #vacant -=1
if(io.input(22)==True):
    temp[3]=0
    io.output(8, True)
    io.output(11,False)

else:
    temp[3]=1
    io.output(8, False)
    io.output(11,True)

'''if(io.input(18)==True):

    if(counter == 1):
        p.ChangeDutyCycle(6.5)
        time.sleep(5)
else:

    p.ChangeDutyCycle(2.5)

if(io.input(4)==True):
    #start_new_thread(check(),)
    if(counter == 1):
        p.ChangeDutyCycle(6.5)
        time.sleep(5)
else:
    if(counter == 1):
        p.ChangeDutyCycle(2.5)'''
if(counter == 1):
    print("entered")
    p.ChangeDutyCycle(6.5)
    time.sleep(5)
    counter = 0
else:
```

```
        p.ChangeDutyCycle(2.5)


    for i in temp:
        if(i==1):
            vacant+=1
    if(len(clients) > 0):
        write(clients[0].conn,'vacant:'+str(vacant))

except:

    print("Error")
    io.cleanup()
```



Data Transmission:

Integrate a method for the Raspberry Pi to send this occupancy data to your IoT platform.

You can use MQTT, HTTP requests, or other communication protocols based on your chosen platform.

Data Analysis and Integration:

On your IoT platform, set up a way to receive data from the Raspberry Pi.

This platform should handle data storage, real-time analysis, and data visualization.

Continuous Testing and Refinement:

Continuously test your system to ensure the sensors accurately detect parking space occupancy.

Refine your Python script and integration with the IoT platform as needed.

CONCLUSION:

- This script configures a Raspberry Pi to trigger an ultrasonic sensor and send the distance data to an MQTT broker. You will need to adapt the GPIO pin numbers, MQTT settings, and sensor logic to match your specific hardware and cloud or mobile app server setup.

- This is a basic example to get you started. Depending on your specific use case, you may need to implement error handling, data filtering, and additional functionalities. Also, ensure that your cloud or mobile app server is set up to receive and process the MQTT data.