**SMART PARKING SYSTEM**

Building an IoT-enabled Smart Parking involves several steps,including deploying IoT sensors, collecting data, and developing the necessary software toprocess and transmit the data.To begin building your IoT sensor system( for example:Ultrasonic sensors) with Raspberry Pi integration, follow these steps:

**Gather the Required Components**:

* Pi (any model with GPIO pins)
* IoT sensors (ultrasonic sensors for parking space occupancy)
* Jumper wires Raspberry
* Breadboard (if necessary)
* Internet connectivity (Wi-Fi/Ethernet)

**Setup Your Raspberry Pi:**

* Install a Raspberry Pi operating system (e.g., Raspberry Pi OS).
* Ensure your Pi is connected to the internet.

**Connect Ultrasonic Sensors**:

* Connect the ultrasonic sensors to the Raspberry Pi's GPIO pins. Make sure to wire them correctly (VCC, GND, Trigger, Echo).
* Refer to the sensor's datasheet for specific pin configurations.

**Write Python Scripts:**

* Create Python scripts to interact with the sensors. You can use libraries like RPi.GPIO to interface with GPIO pins.
* Read sensor data (distance measurements) and store it in variables.

**Send Data to Cloud or Mobile App Server:**

* Choose a cloud platform (e.g., AWS IoT, Azure IoT, Google Cloud IoT) or set up your own server.
* Use libraries or SDKs to send sensor data to the selected platform. You may need to create an account and set up appropriate configurations.

**Cloud/Mobile App Integration:**

* On the cloud or mobile app side, set up a receiver or endpoint to accept incoming data from your Raspberry Pi.
* Develop the necessary code to process and visualize the sensor data.

**Testing and Troubleshooting:**

* Test your system to ensure that data is being collected and transmitted correctly.
* Debug and troubleshoot any issues in the setup.

**Security Considerations:**

* Implement security measures to protect your IoT system and data during transmission.

**Scaling and Optimization:**

* Depending on your project's requirements, consider how you can scale the system, optimize code, and enhance its capabilities.

**PROGRAM:**

import RPi.GPIO as GPIO

import time

import json

import paho.mqtt.client as mqtt

# Define the ultrasonic sensor's GPIO pins

TRIG\_PIN = 23

ECHO\_PIN = 24

# Define the MQTT parameters

MQTT\_BROKER = "your-mqtt-broker-url"

MQTT\_PORT = 1883

MQTT\_TOPIC = "sensor\_data"

# Initialize GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup(TRIG\_PIN, GPIO.OUT)

GPIO.setup(ECHO\_PIN, [GPIO.IN](http://gpio.in/))

# Initialize the MQTT client

client = mqtt.Client()

def measure\_distance():

    GPIO.output(TRIG\_PIN, True)

    time.sleep(0.00001)

    GPIO.output(TRIG\_PIN, False)

    while GPIO.input(ECHO\_PIN) == 0:

        pulse\_start = time.time()

    while GPIO.input(ECHO\_PIN) == 1:

        pulse\_end = time.time()

    pulse\_duration = pulse\_end - pulse\_start

    distance = pulse\_duration \* 17150 # Speed of sound = 34300 cm/s

    return round(distance, 2)

def on\_connect(client, userdata, flags, rc):

    print("Connected with result code " + str(rc))

    client.subscribe(MQTT\_TOPIC)

def on\_publish(client, userdata, mid):

    print("Data sent to the cloud")

# Set MQTT callbacks

client.on\_connect = on\_connect

client.on\_publish = on\_publish

# Connect to the MQTT broker

client.connect(MQTT\_BROKER, MQTT\_PORT, 60)

try:

    while True:

        distance = measure\_distance()

        data = {

            "sensor": "ultrasonic",

            "distance\_cm": distance

        }

        payload = json.dumps(data)

        # Publish data to the MQTT topic

        client.publish(MQTT\_TOPIC, payload)

        time.sleep(1) # Adjust the interval as needed

except KeyboardInterrupt:

    GPIO.cleanup()

    client.disconnect()

**OUTPUT:**

Connected with result code 0

Data sent to the cloud

Data sent to the cloud

Data sent to the cloud

The "Data sent to the cloud" message will be printed each time data is successfully sent.The loop will continue until you interrupt it by pressing Ctrl + C. Make sure we have to set up the MQTT broker, and the receiving end (cloud or mobile app server) is subscribed to the specified MQTT topic to receive the data.