NAAN MUDHALVAN

**INTERNET OF THINGS PHASE-3**

**SMART PARKING**

TEAM MEMBERS

ASHVIKA D JEMIMAH E RAKSAN S

GOKUL S KAVIRAJAN M THARUN SRIRAM

The problem of parking space saturation is indeed a common issue in many urban areas, leading to congestion, frustration, and various related problems. Using technology to address this problem, as you've mentioned, is a promising approach.  
 Implementing a system that can show the availability of parking spaces in real-time using technologies like Raspberry Pi and IR (Infrared) modules can be a practical solution. Here's how such a system might work:

1. Sensor Placement: Install IR modules or other suitable sensors at each parking space. These sensors can detect the presence of a vehicle and relay this information to a central system.

2. Data Processing: The data from these sensors can be processed by a central system, which could be a Raspberry Pi or a more centralized server. The central system continuously monitors the status of each parking space.

3. Real-time Updates: The system can then provide real-time updates on parking space availability to users through various means, such as a mobile app or electronic signs on the road leading to the parking area.

4. User Interface: Users can access this information through a user-friendly interface, which can show the number of available parking spaces and even guide them to the nearest available spot.

5. Reservations: In addition to availability, the system can also allow users to reserve parking spaces in advance. This can be especially useful for places with high demand for parking.

6. Payment Integration: If required, the system can also be integrated with payment gateways, allowing users to pay for parking through the same app or interface.

**Benefits of such a system:**

1. Reduced Congestion: With real-time information on parking availability, drivers can avoid parking lots that are full, reducing unnecessary traffic.

2. Improved User Experience: Users benefit from a more convenient and less stressful parking experience.

3. Optimized Space Usage: The system can help parking lot operators better manage their resources and potentially reduce the need for excessive parking infrastructure.

4. Data Collection: Over time, the system can collect data on parking utilization, helping city planners make informed decisions about future parking infrastructure.

While this is a practical and innovative approach, it does come with its challenges. Implementing such a system requires a significant upfront investment in sensors and infrastructure. Additionally, there may be privacy concerns related to tracking the movement of vehicles.  
 However, as technology continues to advance and smart city initiatives become more prevalent, solutions like the one you've proposed are likely to play a more prominent role in managing urban challenges like parking congestion.

*IR proximity sensor:* An IR (Infrared) proximity sensor operates by energizing a pair of IR light emitting diodes (LEDs), causing them to emit infrared light. This emitted light travels through the air, and upon encountering an object, it gets reflected back towards the sensor. The strength of the reflected light depends on the proximity of the object. If the object is nearby, the reflected light will be more intense compared to when the object is farther away.

The sensor unit, typically in the form of an integrated circuit (IC), is responsible for detecting this reflected infrared light. When the intensity of the reflected light is sufficiently strong, the IC becomes active. Once the sensing unit is active, it generates a corresponding signal at the output terminal. This signal can then be used to trigger or activate various devices. In the context of your exercise, a small LED is used to illustrate this concept, and it turns on when the sensor becomes active.

*MQTT:* MQTT, short for Message Queuing Telemetry Transport, is an IoT communication standard. This open protocol allows for universal access and implementation in networked applications. MQTT serves as a standardized method for developers to send messages to IoT devices and receive responses. This flexibility enables the development of new control systems and IoT device monitoring tools.Originally introduced in 1999, MQTT has been in use for over two decades. It was initially designed for communication with oil pipeline equipment and falls under the framework of Supervisory Control and Data Acquisition (SCADA), which is intended for industrial communication with factory equipment and process control sensors and controllers.

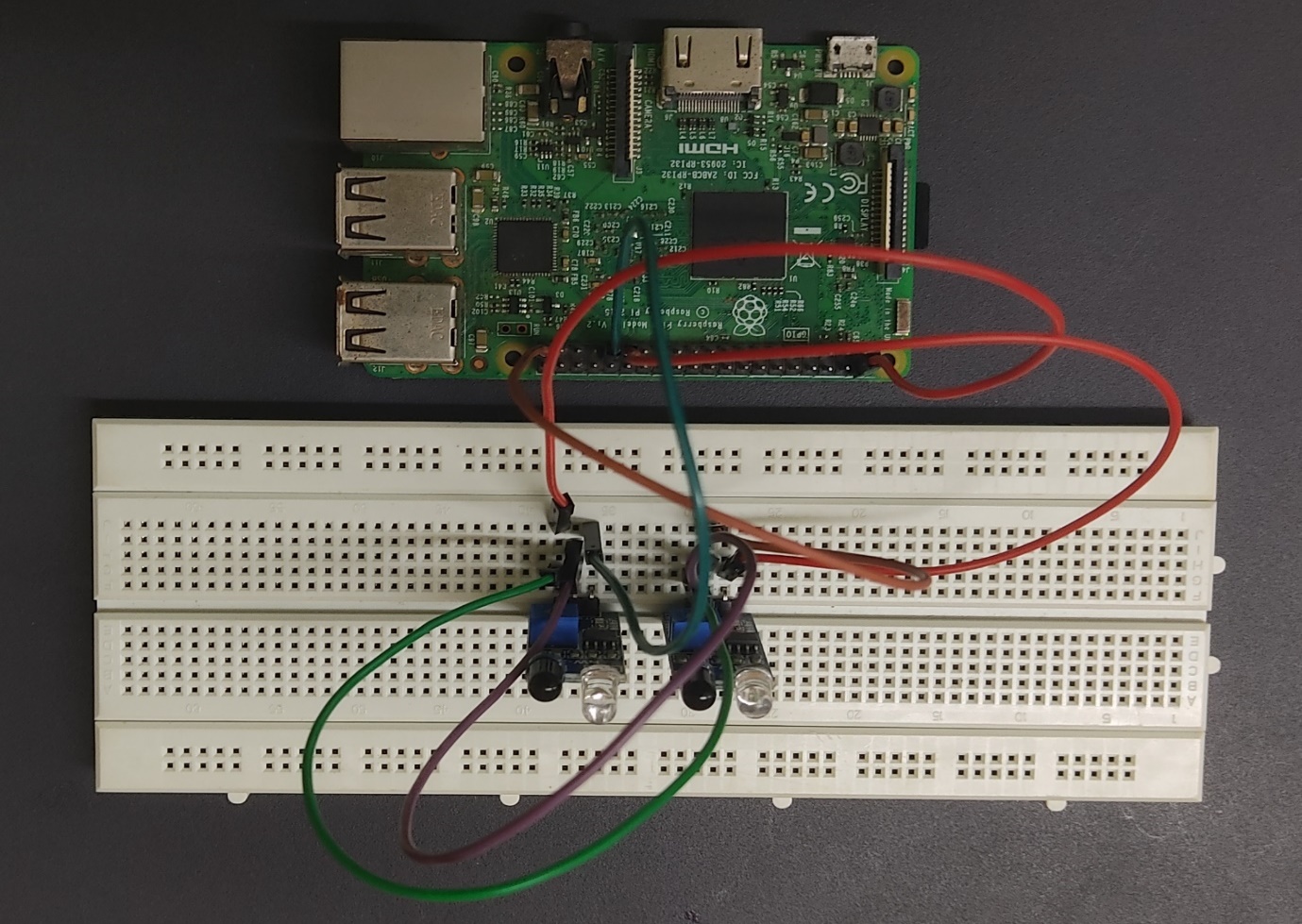
**Hardware Implementation**

MQTT APP SETUP:

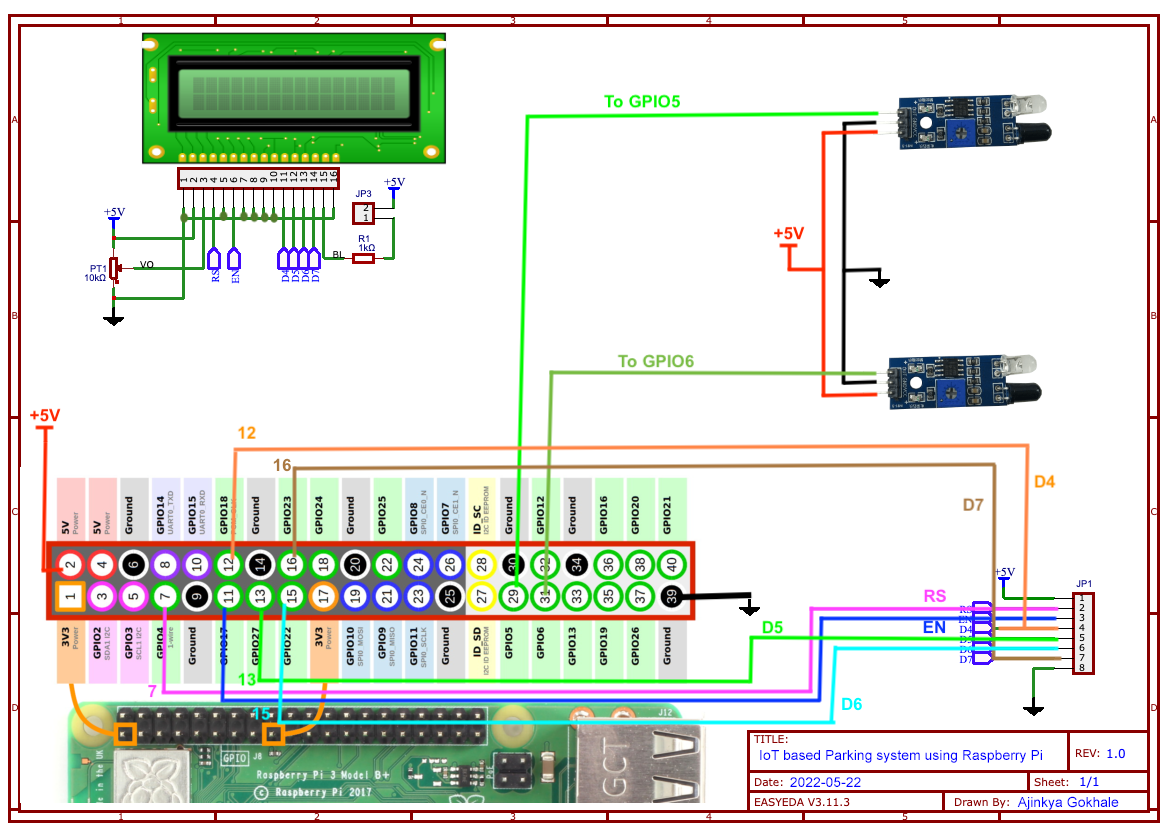
Step 1: Download the MQTT Dashboard App from the Google Play Store.  
  
Step 2: In your MQTT Dashboard App, add a broker using the following parameters.

Step 3: Add a Toggle element to your project and rename it as "slot1". Configure the Toggle settings as follows:  
- ON Value: '1'  
- OFF Value: '0'  
- ON Colour: Red  
- OFF Colour: Green  
- Icon: Car (or any car icon of your choice)

Step 4: With these configurations, the setup is complete.



**sSCHEMATIC:**



**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)

# 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)

Function 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)