SMART PARKING

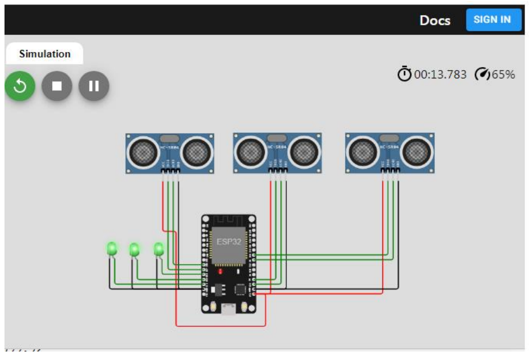
PROJECT OBJECTIVES:

A smart parking project focuses on utilizing technology to optimize parking space management, improve user experience, reduce traffic congestion, and enhance overall efficiency in parking facilities. It involves implementing sensors, data analytics, mobile apps, and smart infrastructure to guide drivers to available parking spots, thereby reducing search time, congestion, and emissions. The project aims to enhance safety, provide real-time information to users, and enable better resource management by collecting and analyzing parking data for informed decision-making. Ultimately, smart parking projects aim to create a seamless, user-friendly, and efficient parking experience while contributing to more sustainable urban environments.

IOT SENSOR SETUP:

In a smart parking system, IoT (Internet of Things) sensors play a crucial role in detecting the presence or absence of vehicles in parking spaces. These sensors help in managing parking spaces more efficiently by providing real-time data on availability, enabling users to find and reserve parking spots easily. Here are some common IoT sensors used in smart parking systems:

Ultrasonic Sensors: These sensors use sound waves to detect the presence of vehicles. They emit ultrasonic waves and measure the time taken for the waves to bounce back. When a vehicle blocks the ultrasonic signal, the sensor registers occupancy



Infrared Sensors: Infrared sensors detect the presence of vehicles by emitting and detecting infrared radiation. When a vehicle interrupts the infrared beam, the sensor identifies that the parking spot is occupied.

Magnetic Sensors: Magnetic sensors detect changes in the magnetic field caused by the presence of a vehicle. They can accurately detect the presence of metal, making them suitable for identifying parked vehicles.

MOBILE APP DEVELOPMENT:

Developing a mobile app for smart parking involves several key steps to create a user-friendly and efficient application. Here's an outline of the development process:

1.Research and Planning

. • Identify Objectives: Understand the main goals of the app. Is it for users to find parking spots, make reservations, pay for parking, or a combination of these?

.Target Audience: Define the user demographics and preferences

•Competitive Analysis: Study existing smart parking apps to identify strengths and weaknesses for differentiation.

2. Design and UI/UX:

- Create wireframes and mockups for the app's user interface.

- Ensure a simple, intuitive, and visually appealing design for ease of use.

3. Technology and Backend Development:

• Choose the appropriate tech stack for the app (native, hybrid, etc.).

• Develop the backend server to handle user data, payments, reservations, and parking spot availability.

4. Integration and Testing:

• Integrate external APIs for mapping, payment gateways, and other required functionalities.

• Thoroughly test the app for functionality, usability, and security.

5. Security and Compliance:

• Implement strong security measures to protect user data and transactions.

• Ensure compliance with relevant data protection laws and regulations.

6. Launch and Marketing:

• Publish the app on relevant app stores (Apple App Store, Google Play Store).

• Plan a marketing strategy to reach potential users.

7. Maintenance and Updates:

• Regularly update the app based on user feedback and technological advancements.

Additional Considerations:

• Collaborate with parking lot owners, local authorities, or IoT sensor manufacturers for real-time parking spot availability.

• Consider including features for user reviews and ratings of parking spots.

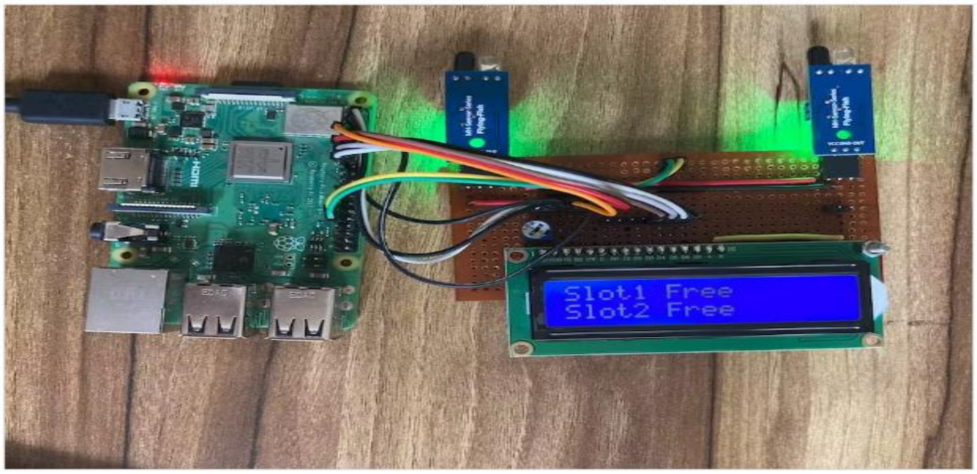
This roadmap is a general guideline; actual development might vary based on specific requirements, user expectations, and the complexity of the app. Smart parking apps can offer a range of benefits to users and parking lot operators, including convenience, efficient space utilization, and improved traffic management.

RASPBERRY PI INTEGRATION:

Integrating Raspberry Pi into a smart parking system offers a cost-effective and flexible solution to manage and monitor parking spaces. The Raspberry Pi can serve as the central control unit for various tasks and interactions within the smart parking infrastructure. Here's how you can integrate Raspberry Pi into a smart parking system:

1. Hardware Setup:

• **Sensors and IoT Devices**: Use sensors (such as ultrasonic, infrared, or magnetic sensors) to detect vehicle presence in parking spots. Connect these sensor Raspberry Pi's GPIO pins for data



• **LED Displays or Indicators**: Utilize LED indicators to show available or occupied parking spots. sensor data.• Communication Modules: Incorporate Wi-Fi or data transmission.

2. Software Development:

• Data Processing and Storage: Develop software to process data collected from the sensors. Store and manage this data on the Raspberry Pi or a connected database.

• Decision-Making Logic: Create algorithms to interpret sensor data, determine parking spot availability, and control LED indicators accordingly.

• Communication Protocols: Implement communication protocols to enable the Raspberry Pi to communicate with a central server or a user-facing mobile app.

3. Integration with Mobile App:

• API Development: Develop APIs or web services to facilitate communication between the Raspberry Pi and a mobile app.

• Real-Time Data Updates: Allow the mobile app to access real-time parking spot availability information obtained from the Raspberry Pi.

4. User Interface Development:

• Dashboard or Monitoring System: Design a user interface to monitor parking spot availability, occupancy rates, and any other relevant data in real-time.

• User Interaction: Integrate features in the user interface to control or manage parking spaces through the Raspberry Pi system.

5. Remote Control and Management:

• Enable remote control and management of the parking system through the mobile app or a centralized system.

6.Important Considerations:

• **Security**: Implement robust security measures to protect data and ensure secure communication between the Raspberry Pi and the user interface.

• **Scalability**: Design the system to be scalable, allowing for potential future expansion or the addition of more parking spots or functionalities.

Integrating Raspberry Pi into a smart parking system offers the advantage of a customizable and adaptable solution. It provides a cost-effective way to manage parking spaces efficiently while allowing for easy expansion or modification as needed.

CODE IMPLEMENTATION:

Code for smart parking using raspberry pi,

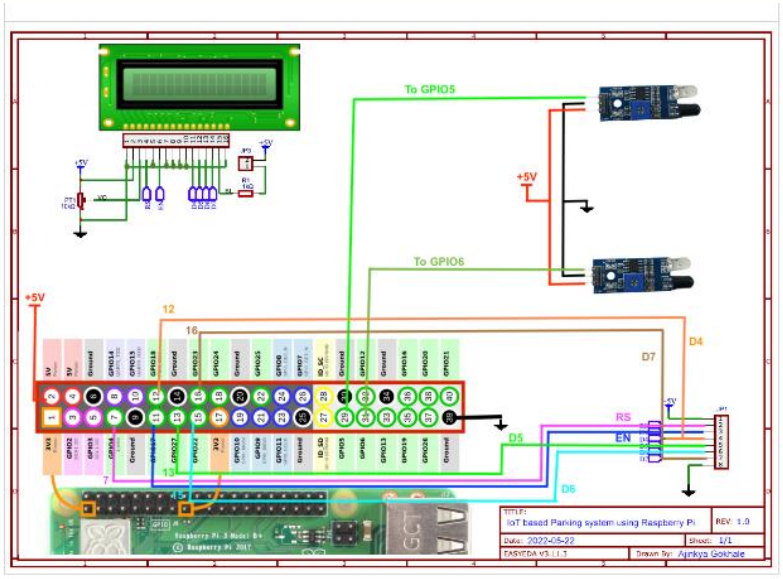
```
import time
import RPi.GPIO as GPIO
import time
import os,sys
from urllib.parse import urlparse
import paho.mqtt.client as paho
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
'''
define pin for lcd
'''
# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005
delay = 1
# Define GPIO to LCD mapping
LCD_RS = 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 DIAGRAM:



REAL TIME AVAILABLITY OF SYSTEM:

A real-time parking availability system can significantly benefit drivers and help alleviate parking issues in several ways:

1. Improved Parking Efficiency:

Time-Saving: Drivers can easily locate available parking spaces without aimlessly circling lots or streets, reducing the time spent searching for parking.

Reduced Congestion: By directing drivers to available spots efficiently, it minimizes traffic congestion caused by cars circulating in search of parking.

2. Convenience for Drivers:

Quick Access: Instant information about available parking spots in the vicinity allows drivers to secure a space promptly upon arrival.

Reservations: Some systems enable advance reservations, ensuring a guaranteed spot, which is especially useful in high-traffic areas or during peak hours.

3. Cost and Resource Savings:

Reduced Fuel Consumption: Less time spent searching for parking leads to decreased fuel consumption and lower emissions, benefiting the environment.

Avoiding Fines or Overpayments: Drivers can avoid fines or overpaying by having accurate information about available parking options.

4. Enhancing User Experience:

Real-Time Updates: Users receive live updates about parking space availability, thus making informed decisions before reaching their destination.

5. Optimized Parking Utilization:

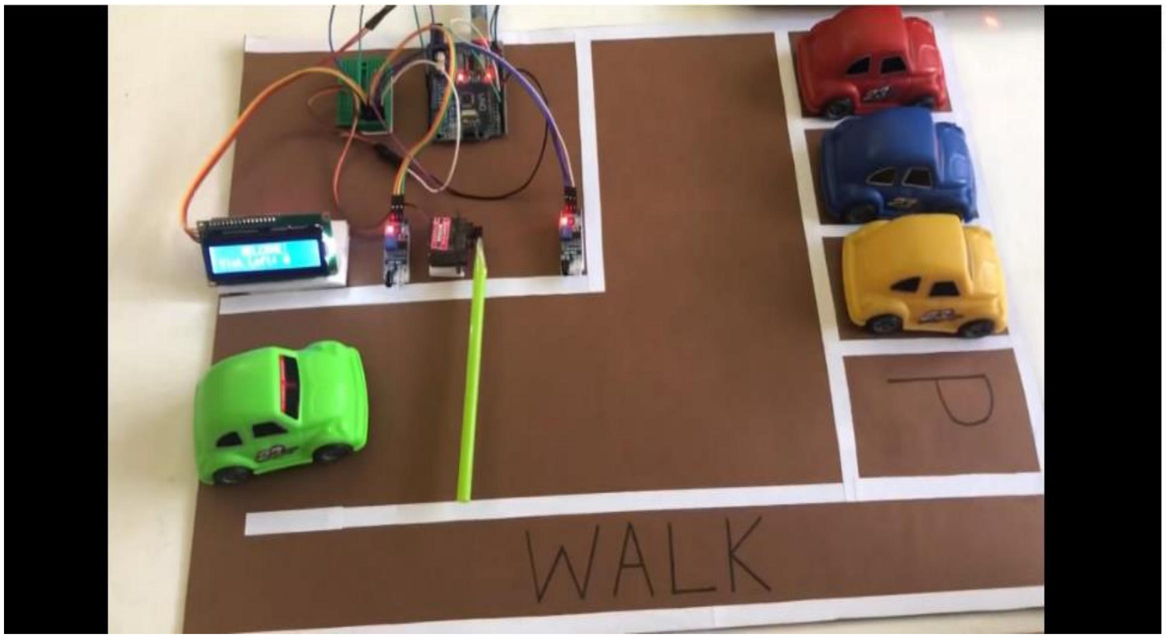
Better Space Management: Real-time data allows parking lot operators to optimize space utilization and plan maintenance based on actual usage patterns.

6. Traffic Management and City Planning:

Reduced Traffic Congestion: Efficient parking contributes to overall traffic management, benefiting the flow of vehicles within a city.

Urban Planning: Data from these systems can aid urban planners in making informed decisions about infrastructure and parking management.

OUR OWN PROJECT FOR SMART PARKING USING ARDUINO:



Our mini project of smart parking using arduino,servo motor,IR sensor and LED Display.we use own components like arduino etc,.and we got an output for this project successfully.

CHALLENGES AND CONSIDERATIONS:

Accuracy and Reliability: Ensuring the system's accuracy and reliability in detecting available spots.

Integration and Adoption: Encouraging widespread adoption and integration across different parking facilities and cities

CONCLUSION:

Automation is a step in the right direction for a future fulfilled in the world of transportation. Smart parking solutions revolutionize urban mobility, offering convenience by swiftly locating available parking, easing traffic congestion, optimizing resource use, and contributing to smarter, more efficient cities. It can be concluded that with correct connection of some simple electrical components, it is possible to create an automatic smart car parking system, thus decreasing aimless driving, fuel and time, as well as making the process of parking considerably simpler.