



**SMART  
PARKING**

# IOT BASED SMART PARKING SYSTEM

- The project is a real world's application which can be incorporated to any parking management system. First of all, we will fix the proximity sensors as per the number of slots for parking. These sensors will tell whether any area is empty or not. This will be know to the authority in charge. It can have the report of it's complete management on tips. On the other hand, if the user wants to avail the service at XYZ parking area. Then he will first book his slot at the parking. However, this depends on the availability as well.
- Once the user books the slot with the date and time, he will be the owner of that area for the set period of time. To authenticate whether the user who is entering with the car has his pre booked slot or not, we will use RFID tags. These contain the information of the type of permit that the particular vehicle has been granted. The person enforcing it is equipped with a hand reader that captures the information of the vehicle, such as expiration time and corresponding spot location.

## PROJECT OBJECTIVE:

- ❑ Real-Time Parking Space Management
- ❑ Mobile App Integration
- ❑ Efficient Parking Guidance

# REAL TIME PARKING SPACE MANAGEMENT

Real-time parking space management for smart parking is a crucial component of modern urban planning and transportation systems. It involves the use of technology to efficiently manage and optimize the utilization of parking spaces in urban areas. Here's an overview of how real-time parking space management for smart parking works:

**Sensor Technology:** To monitor parking space availability in real-time, various types of sensors can be used. These sensors can include:

1. **Infrared Sensors:** These sensors detect the presence of a vehicle in a parking space by measuring changes in infrared radiation.
2. **Ultrasonic Sensors:** Ultrasonic sensors use sound waves to detect the presence or absence of vehicles in parking spaces.
3. **Magnetic Sensors:** Magnetic sensors can detect the presence of a vehicle by measuring disturbances in the Earth's magnetic field caused by the metal in the vehicle.
4. **Camera-Based Systems:** High-resolution cameras can be used to capture images of parking spaces and analyze them to determine occupancy.

- ❖ **Data Communication:** Data from these sensors is transmitted in real-time to a centralized parking management system. This can be achieved through wired or wireless communication networks, such as Wi-Fi, cellular networks, or LoRaWAN
- ❖ **Data Processing and Analysis:** The data collected from the sensors is processed and analyzed by the parking management system. Advanced algorithms can determine the occupancy status of each parking space, and this information is updated in real-time.
- ❖ **Interface:** A user-friendly interface is essential for both parking operators and drivers. Users can access real-time parking availability information through mobile apps, websites, or electronic signage located at key points in the city.
- ❖ **Reservation Systems:** In addition to real-time information, smart parking systems often provide the option for drivers to reserve parking spaces in advance, reducing the uncertainty of finding parking.
- ❖ **Navigation and Guidance:** Real-time parking systems can provide drivers with turn-by-turn navigation to available parking spaces. This not only saves time but also reduces traffic congestion as drivers no longer need to circle in search of parking.

# MOBILE APP INTEGRATION

Integrating a mobile app into a smart parking system is crucial for providing a seamless and user-friendly experience for drivers. Below are the key steps and features involved in integrating a mobile app for smart parking:

- ✓ **Real-Time Parking Availability:** The core feature of the mobile app should be to provide real-time information on parking space availability. This information is typically obtained through sensors installed in parking spaces and transmitted to the app.
- ✓ **User Registration and Authentication:** Users should be able to create accounts and log in securely to the app. This allows for personalized features such as reservation history, payment methods, and preferences.



- ✓ **Search and Navigation:** The app should include a search functionality that allows users to find nearby parking spaces. Navigation features can guide drivers to the chosen parking spot using GPS and mapping services like Google Maps or Apple Maps.
- ✓ **Reservation and Booking:** Users should have the option to reserve parking spaces in advance through the app. This feature can include selecting a specific parking location, date, time, and duration of stay. Integration with payment gateways for booking fees is essential.
- ✓ **Payment Integration:** Secure payment options should be integrated into the app to enable users to pay for their parking reservations. This can include credit card, mobile wallets, or even integration with parking passes and subscriptions.
- ✓ **Notifications and Alerts:** The app can send push notifications to users regarding their reservations, upcoming expirations, and important information related to the parking facility.

# EFFICIENT PARKING GUIDANCE

## 1. Communication Infrastructure:

1. Establish a robust communication infrastructure to transmit real-time parking data to a centralized system.
2. Utilize wireless technologies like Wi-Fi, Bluetooth, or cellular networks for data transmission.
3. Ensure redundancy and reliability in data communication to prevent system failures.

## 2. User-Friendly Mobile Apps and Signage:

1. Develop mobile apps that allow users to access real-time parking availability information, make reservations, and navigate to available parking spaces.
2. Install electronic signage at key points in the city to display real-time parking information and directions to available parking facilities.

## 3. Navigation and Routing:

1. Provide turn-by-turn navigation within the mobile app to guide drivers to available parking spaces.
2. Consider integration with popular navigation apps (e.g., Google Maps, Apple Maps) for seamless routing.



## 4. Dynamic Pricing and Payment Integration:

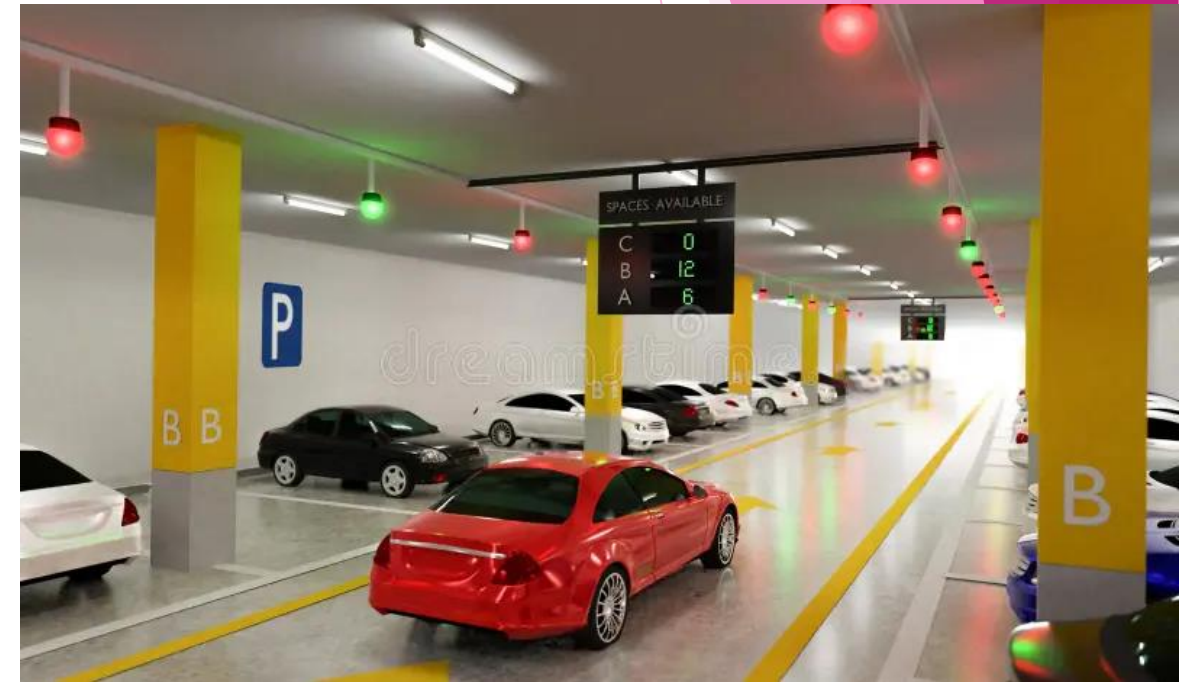
1. Implement dynamic pricing strategies that adjust parking fees based on demand, time of day, location, and other factors.
2. Integrate secure payment gateways within the app to facilitate cashless payments.

## 5. Reservation System:

1. Offer users the option to reserve parking spaces in advance through the mobile app.
2. Ensure that reserved spaces are clearly marked and accessible upon arrival.

## 6. Parking Garage Guidance:

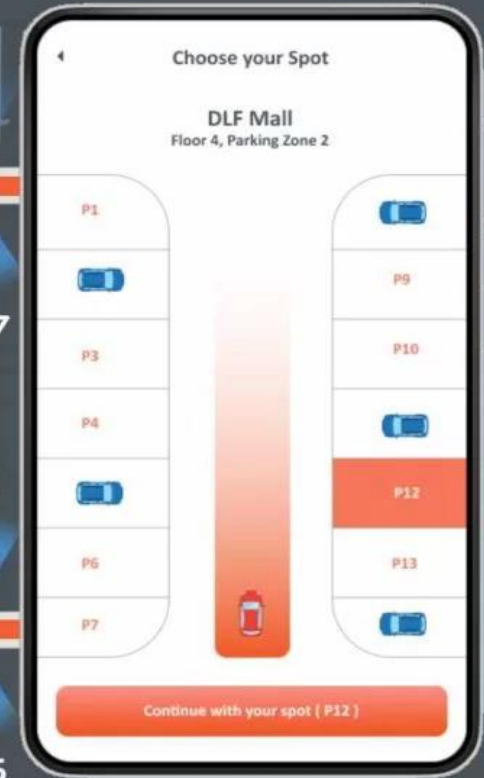
1. Implement guidance systems within multi-level parking garages using LED lights, digital signage, or floor markings to indicate available spaces.
2. Ensure clear and intuitive directional signage to guide drivers efficiently.



# IOT SENSOR DESIGN



## SENSOR BASED PARKING AVAILABILITY SYSTEM



- ▶ IoT (Internet of Things) sensors play a vital role in creating smart parking solutions by providing real-time data on parking space occupancy and availability. These sensors are designed to detect the presence or absence of vehicles in parking spaces and communicate this information to a central system or mobile app. Here are some common types of IoT sensors used for smart parking:

### 1. **Ultrasonic Sensors:**

1. Ultrasonic sensors use sound waves to detect the presence of vehicles. They emit high-frequency sound waves and measure the time it takes for the waves to bounce back after hitting an object (a vehicle in this case).
2. These sensors are typically installed above each parking space or on light poles above a group of spaces.

### 2. **Infrared Sensors:**

1. Infrared sensors detect the heat emitted by vehicles. When a vehicle enters a parking space, it disrupts the infrared radiation pattern, triggering the sensor.
2. These sensors are often installed on the pavement or mounted on nearby structures.



### 3. Magnetic Sensor

1. Magnetic sensors use changes in the Earth's magnetic field caused by the presence of a vehicle to detect occupancy.
2. They are installed in or under the pavement of each parking space.

### 4. surface-installed sensor

1. Some sensors are designed to be mounted on the surface of the pavement and use various technologies, such as pressure-sensitive mats, to detect vehicle presence.
2. They are relatively easy to install and maintain.

### 5. Camera-Based Systems:

1. Camera-based systems use image recognition technology to analyze camera feeds and identify open parking spaces.
2. These systems are often used in conjunction with machine learning algorithms to improve accuracy.

## Designed to fit every parking detection environment

Nwave wireless vehicle sensors streamline parking detection to fully utilize parking spaces and drive profit. Nwave is the smart parking sensor trusted by leading enterprises, municipalities and university campuses worldwide.





## ► **High Accuracy**

- Advanced filtering and noise reduction techniques allow differentiating parking events from electromagnetic interference or false events such as an underground train passing.

## ► **Easy Installation**

- Each parking sensor takes only 30 seconds to affix, with no extra wires or other street furniture required. The network is easily connected to Nwave's cloud-based app or end-user software of choice.

## ► **Long Life**

- Nwave sensors boast especial ruggedness, high load and damage resistance, an unbeatably high battery life of up to 10 years, with no need for service until then battery change is required. These features lead to quick parking detection.

## ► **LoRaWAN Support**

- Allows your project to leverage existing LoRaWAN infrastructure. Nwave sensors successfully tested in all major LoRaWAN regions (US, EU, Australia, Asia) and with all major gateway and LNS providers (TTN, Kerling, Actility, et.al.)



# REAL-TIME TRANSIT INFORMATION PLATFORM

Designing a mobile app interface to display real-time parking availability to users requires careful consideration of user experience, readability, and ease of use. Here's a basic outline of how the interface might look:

## 1. Landing Page:

- The landing page should be clean and visually appealing, with a user-friendly interface.
- Include the app's logo and a simple, welcoming message.
- Provide the option for users to log in or continue as a guest.

## ► 2. Map View:

- The main screen of the app should display a map of the user's current location.
- Mark nearby parking facilities with pins or icons.
- Use color-coding to indicate parking availability (e.g., green for available, red for full).
- Include an option to search for parking at a specific location (e.g., by address or landmark).

## ► 3. Filters and Sorting:

- Allow users to filter parking options by criteria such as price, distance, and type (e.g., street parking, garages, lots).
- Provide sorting options (e.g., by distance, price, availability) to help users find the most suitable parking.

## ► 4. Parking Details:

- When a user taps on a parking facility on the map, a pop-up or a new screen should display more information about that facility.
- Include the facility name, address, pricing details, and the number of available spaces.
- Add a "Get Directions" button that integrates with navigation apps

## ► 5. Search Bar:

- Place a search bar at the top of the map view for users to manually search for parking at specific locations.
- Include auto-suggestions as users type.

## ► 6. Menu and Navigation:

- Use a navigation drawer or tabs at the bottom for easy access to various app sections, such as Home, Favorites, Reservations, and Account.
- Include a "Nearby Events" or "Local Attractions" section that users can explore while searching for parking.

## ► 7. User Account and Settings:

- Allow users to create accounts or log in to access additional features like booking reservations and saving favorite parking spots.
- Provide options to customize user profiles and set notification preferences.

## ▶ **8. Real-Time Updates:**

- Ensure that parking availability updates in real-time as users interact with the map.
- Display real-time notifications or alerts for significant changes in parking availability or important updates.

## ▶ **9. Favorites and Reservations:**

- Allow users to save their favorite parking spots and view them easily on a dedicated screen.
- Provide the ability to reserve parking spots directly within the app.

## ▶ **10. Feedback and Help:** - Include a section for users to provide feedback, report issues, or seek assistance. - Provide contact information for customer support.

## ▶ **11. Accessibility and Multilingual Support:** - Ensure that the app is accessible to users with disabilities. - Offer content and support in multiple languages if applicable.

- ▶ **12. Security and Privacy:** - Display clear privacy and data use policies. - Implement security measures to protect user data transactions.
- ▶ **13. App Logo and Branding:** - Use a memorable app logo and consistent branding throughout the interface.
- ▶ **14. App Ratings and Social Sharing:** - Encourage users to rate and share their experiences on social media platforms.
- ▶ **15. Logout Option:** - Provide a clear option for users to log out accounts if they wish.

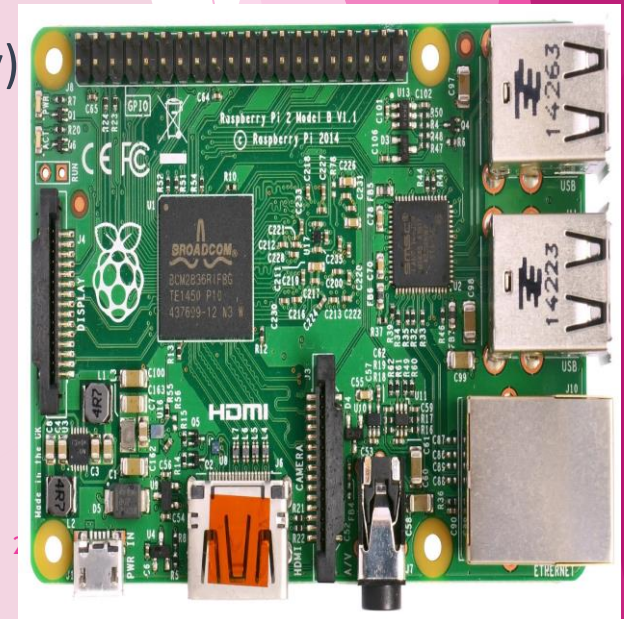


# INTEGRATED APPROACH

To collect data from sensors using a Raspberry Pi and update a mobile app, you'll need to set up the Raspberry Pi as a sensor data collector and implement a communication mechanism between the Raspberry Pi and the mobile app. Here's a step-by-step guide on how to achieve this:

## Hardware and Software Requirements:

- 1. Raspberry Pi:** You'll need a Raspberry Pi board (e.g., Raspberry Pi 3 or 4) with an operating system (e.g., Raspbian) installed.
- 2. Sensors:** Connect the sensors (e.g., ultrasonic, infrared, temperature, humidity) to the Raspberry Pi using appropriate GPIO pins or interfaces. Ensure that you have the necessary libraries or drivers to interact with the sensors.





**3.Mobile App:** Develop a mobile app for iOS or Android using a suitable framework (e.g., React Native, Flutter, or native development tools). Implement the app's user interface and functionality, including data visualization.

**4.Communication Protocol:** Decide on a communication protocol to transmit sensor data from the Raspberry Pi to the mobile app. MQTT (Message Queuing Telemetry Transport) and RESTful APIs are common choices.

► **Steps to Collect Data and Update the Mobile App:**

**1. Sensor Data Collection:**

1. Write Python scripts on the Raspberry Pi to collect data from the connected sensors. Depending on the sensors used, you'll need to read values periodically (e.g., temperature and humidity) or in response to events (e.g., motion detection).
2. Store the collected data in variables or data structures within your Python script.

**2. Data Processing:**

1. Process and format the collected sensor data as needed. You may need to convert sensor readings into meaningful values (e.g., temperature in degrees Celsius) and prepare the data for transmission.

### 3.Communication with the Mobile App:

1. Decide on the communication method between the Raspberry Pi and the mobile app. MQTT is a lightweight messaging protocol suitable for real-time data updates. Alternatively, you can expose a RESTful API on the Raspberry Pi to receive HTTP requests from the mobile app.

### 4.MQTT Communication (Option 1):

1. If using MQTT, install an MQTT broker (e.g., Mosquitto) on the Raspberry Pi.
2. In your Python script, use an MQTT client library (e.g., Paho MQTT) to publish sensor data to MQTT topics.
3. In the mobile app, implement an MQTT client to subscribe to the same MQTT topics and receive real-time updates.

### 5.RESTful API Communication (Option 2):

1. If using a RESTful API, create an API on the Raspberry Pi using a web framework like Flask or Django.
2. Implement endpoints on the Raspberry Pi API to receive sensor data updates from the Python script and provide access to that data.
3. In the mobile app, use HTTP requests (e.g., GET or POST) to send data to and retrieve data from the Raspberry Pi API.

## 6. Mobile App Integration:

- In the mobile app, implement logic to make requests (MQTT or HTTP) to the Raspberry Pi to retrieve sensor data.
- Update the app's user interface to display the real-time sensor data using appropriate widgets (e.g., charts, text fields).
- Implement error handling and data parsing in the app to ensure smooth data retrieval and display.

## 7. Testing and Deployment:

- Test the entire system by running the Raspberry Pi script and launching the mobile app on a test device.
- Debug and refine the system as needed.
- Once tested successfully, deploy the Raspberry Pi and mobile app in your target environment.

## 8. Security Considerations:

- Implement security measures, such as encryption and authentication, to protect data transmission between the Raspberry Pi and the mobile app.

By following these steps, you can collect data from sensors using a Raspberry Pi and update a mobile app with real-time sensor data. The choice of communication method (MQTT or RESTful API) depends on your specific project requirements and constraints.

# BENEFITS OF SMART PARKING

- ▶ Smart parking systems offer numerous benefits for both city planners and citizens. These benefits contribute to improved traffic management, reduced environmental impact, and enhanced overall quality of life. Here are some of the key advantages of smart parking:

## 1. Reduced Traffic Congestion:

1. Smart parking systems provide real-time information on available parking spaces, helping drivers find parking quickly. This reduces the need to circle in search of parking, which, in turn, lowers traffic congestion and reduces fuel consumption.

## 2. Time Savings:

1. Drivers spend less time searching for parking, resulting in shorter commutes and reduced stress levels. This can improve overall quality of life and productivity.

## 3. Lower Emissions:

1. Reduced congestion and idling result in lower greenhouse gas emissions. Smart parking contributes to a more environmentally friendly urban environment.

#### 4.Revenue Generation:

1. Cities and parking operators can generate revenue through dynamic pricing models, where parking rates are adjusted based on demand. Additionally, data collected from smart parking systems can be monetized through partnerships or advertising.

#### 5.Enhanced Safety:

1. Smart parking systems often include security features such as video surveillance, well-lit areas, and emergency call buttons, which improve the safety of parking facilities.

#### 6.Improved Accessibility:

1. Accessible parking spaces for individuals with disabilities can be better managed, ensuring that they are available when needed.

#### 7.Enhanced User Experience:

1. Users can access real-time parking information via mobile apps or electronic signs, making their parking experience more convenient and user-friendly.
2. Parking enforcement becomes more efficient with real-time data. Authorities can enforce parking regulations more effectively, reducing illegal parking.



# INNOVATION

---

**CAMERA BASED SOLUTIONS FOR IMAGE  
PROCESSING – TO DETECT PARKING SPACE  
AVAILABILITY**



# ABSTRACT

---

As there are more cars on the road, parking is becoming a more challenging issue in urban areas. Many studies have been done over the past ten years in an effort to create the optimum autonomous. There is an automated system that can park a car for you, but it needs to know which spots are available and which are taken. The approach for parking spot detection proposed in this paper uses image recognition. This study offers suggestions for parking-space occupancy detection, open parking space visualisation, parking data, wireless networking, widely available components, and. The camera may broadcast a live feed of the parking lot to the system. Images are taken each time a car enters or leaves the parking lot.

# INTRODUCTION

---

- In recent years, visual detection has been the research trend of artificial intelligence, especially in the direction of autonomous driving. When detecting distance with radar in traditional parking [1,2,3], there must be a reference vehicle on both sides of the parking space. Furthermore, the position and angle of parking are strictly required. To solve the above problems, visual parking systems [4,5] have been proposed. During these years, many scholars have completed extensive research in the field of vision detection parking systems.
- Target detection has been widely studied. It is mainly designed for specific targets, such as face recognition and pedestrian detection. However, it is not easy to detect other targets [6,7,8]. To address this issue, Suhr et al. [9] proposed an end-to-end single-stage parking space detection method, which used Convolutional Neural Network (CNN) to simultaneously obtain global information and local information, and combined them with the attributes of the parking space to detect the parking space. Dusan et al. [10] utilized active visual ID tags to simultaneously locate and identify vehicles. The tags with 2-D color can be captured by cameras above the parking lot, thus effectively protecting against changes in ambient lighting and helping to reduce costs.

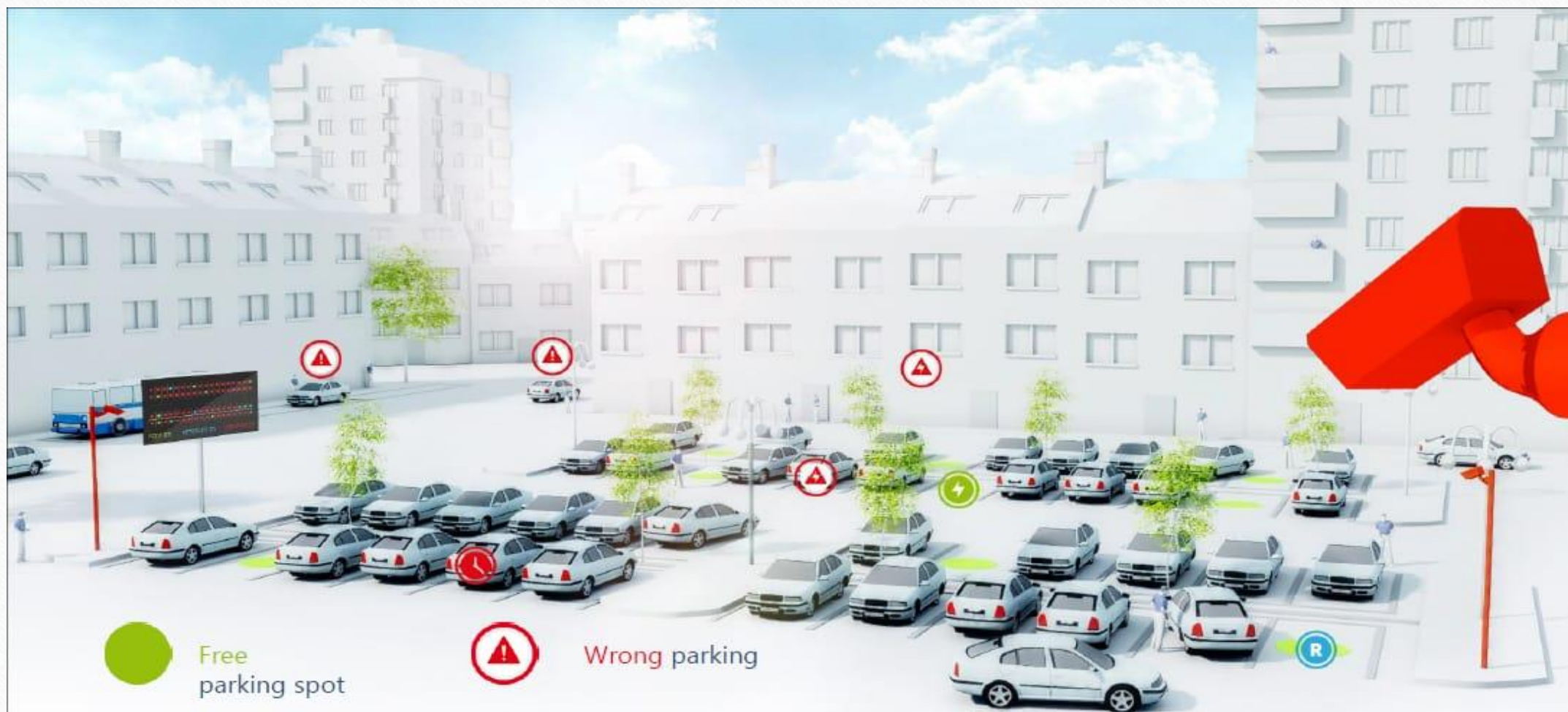


# IMAGE PROCESSING

---

An image is a matrix composed of a pixel. Image processing is to manipulate the pixel matrix. For color images, the color of a pixel can be represented by Formula (4), where  $(x, y)$  represents the coordinates of the pixel, and any color can be represented as the linear sum of three basis vectors. Therefore, the corresponding colors of the specific pixels can be changed by determining the specific coordinates of the pixels and changing the values of the three-color channel corresponding to the pixels. It can be seen from the formula that after the coordinates of the pixel point are determined, the color of the pixel point can be changed by changing the RGB channel value corresponding to the coordinates of the pixel point.

With the model trained by deep learning, some small areas can be detected and extracted from the panoramic aerial view. These areas contain images of only one parking space, which are raw RGB color images. To process color images, we need to adjust the RGB values separately, which greatly increases the computation. At the same time, RGB images cannot represent morphological features, but image colors.



Free  
parking spot



Only electric  
car



Wrong parking



Non-electric  
car



Parking  
time is out



Reserved  
parking spot



# IMAGE ENHANCEMENT

---

There will be a lot of noise in the image because the camera will be disturbed by many factors when taking photos. It will interfere with the identification of parking spaces. The region of interest in the image is enhanced to reduce noise interference and improve the accuracy of recognition. That is to preserve the details of the image and suppress the noise as much as possible, which is necessary to filter. The filtering effect will directly affect the subsequent processing and recognition. In this paper, spatial domain enhancement is selected because it is more suitable for parking detection. Median filtering [25] is better than Gaussian filtering in detail retention and image noise suppression. After a comprehensive comparison, median filtering is finally selected. The principle of median filtering is to use a filter template of a certain size to average the region of a certain size in the image. A  $3 \times 3$  filter template is utilized as an example. The filter template is composed of the pixels to be processed and the 8 pixels around them. The concept of the fuzzy radius is utilized in median filtering. The fuzzy radius of the  $3 \times 3$  template is 1.

# BINARIZATION

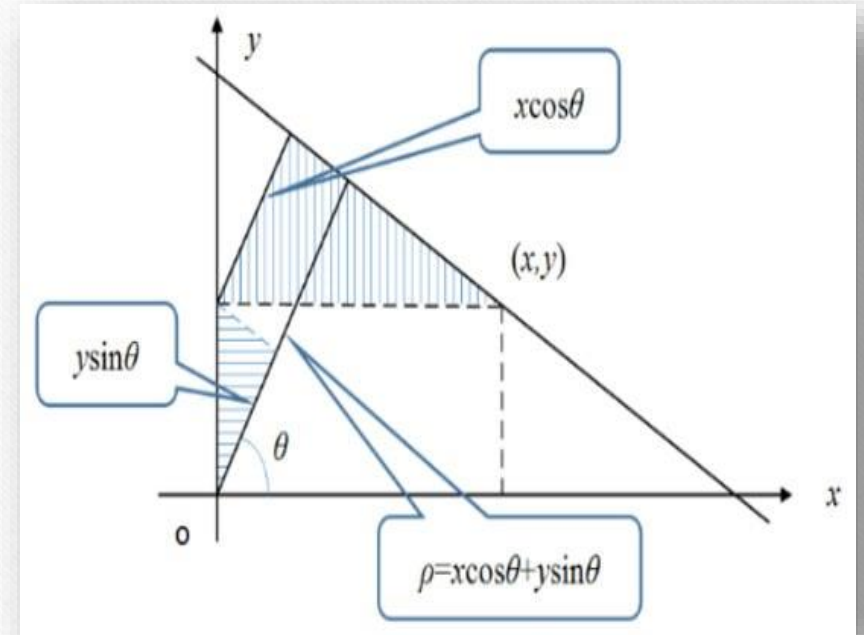
---

Threshold segmentation should be carried out on the image before parking space recognition. The foreground and background should be separated, which is called binarization [26]. The most basic threshold segmentation method is global binarization. It obtains the global threshold through the gray level calculation of the whole image. The pixels higher than this threshold are set to 255. The pixels lower than thresh are set to 0. The formula is shown in (6), where  $x$  represents the gray value of the input pixel,  $f(x)$  represents the gray value of the calculated output pixel, and  $T$  is the threshold value of binarization. The maximum inter-class variance method OSTU [27] is used to calculate the optimal threshold  $T$ .



# HOUGH TRANSFORM

Four vertices need to be positioned after the pre-selected parking space is obtained. In this paper, an improved Hough transform method is adopted to make the decision. Hough transform [31] is a method to transform the straight-line detection problem into the point detection problem in the polar coordinate system. Cartesian and polar coordinates can be converted. Under the rectangular coordinate system of a straight line corresponds to the polar coordinates of  $\theta$  and  $\rho$ . In the figure,  $\theta$  is the angle between the line perpendicular to the line and the X-axis.  $\rho$  is the distance between the origin and the given line.



# CONCLUSION

---

A fisheye camera was used to take real-time pictures of the surrounding environment of the vehicle. At the same time, a faster R-CNN parking detection model was established, which could detect and extract parking spaces from images as the image input of the parking positioning system. Additionally, we addressed the inability of global binarization to handle images with uneven lighting or complex backgrounds by removing background light from the original image. A parking space extraction method based on connected regions was proposed. This method simplified the extraction of parking spaces. It removed interference from irrelevant areas. Finally, the identification and positioning of parking spaces have been realized. The experimental results show that the proposed method can accurately complete the tasks of parking space detection and image processing in the case of uneven illumination or complex background.

# SMART PARKING

**IOT SENSOR SYSTEM AND  
RASPBERRY Pi INTEGRATION**

# SMART PARKING MANAGEMENT SYSTEM USING IOT

## Purpose:

The IoT Smart Parking System is an innovative project aimed at revolutionizing urban parking management. By integrating IoT technology with a Python-based control system, this project addresses the growing challenges of urban congestion and parking inefficiencies.

## Program Overview:

The Python program plays a pivotal role in processing and interpreting data received from the sensors. It provides a user-friendly interface for accessing parking availability information in real time. Moreover, it incorporates intelligent algorithms to optimize parking space allocation based on occupancy trends.



## Objectives:

The primary objectives of this project are to:

1. Provide accurate and real-time information on parking space availability.
2. Optimize parking space allocation to reduce congestion and enhance user convenience.
3. Facilitate seamless interaction between users and the smart parking system.

## Key Features:

- Real-time monitoring of parking space occupancy.
- Dynamic allocation of parking spaces based on user demand.
- Integration with mobile applications for user accessibility.
- Data analytics for trend analysis and optimization.

## Design Principles:

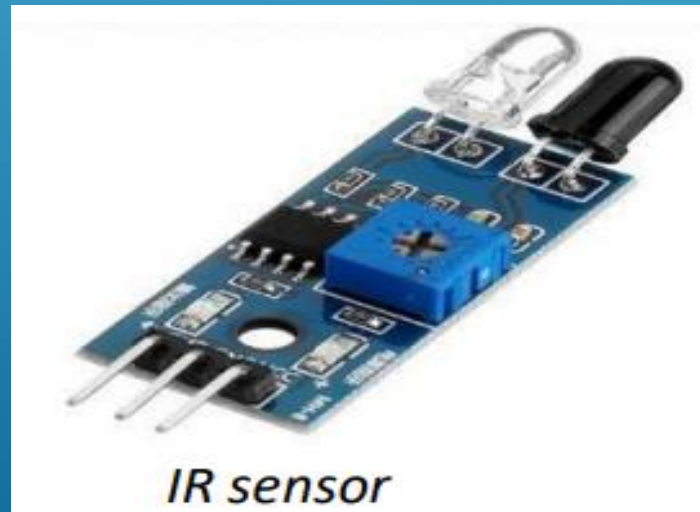
The Python program follows a modular design, allowing for easy scalability and maintenance. It adheres to coding best practices to ensure efficiency and reliability in a real-world urban environment. Design For Smart Parking Management

# Design For Smart Parking Management:

## Components Required:

Hardware:

- Node MCU ESP8266
- IR Sensor- 5 nos
- Servo Motor-2nos



*IR sensor*



*Servo motor*




## CODE:

```
import time from machine
import Pin, Servo, I2C
from ntptime import settime from
umqtt.robust import MQTTClient
# WiFi and MQTT settings
WIFI_SSID = "YourWiFiSSID"
WIFI_PASSWORD = "YourWiFiPassword"
MQTT_BROKER = "io.adafruit.com"
MQTT_PORT = 1883
MQTT_NAME = "aschoudhary"
MQTT_PASS = "1ac95cb8580b4271bbb6d9f75d0668f1"
# Entry and exit sensors
carEnter = Pin(4, Pin.IN)
carExited = Pin(5, Pin.IN)
entrysensor = False exitsensor=False
```

```
# Servo configuration servo_gate = Servo(Pin(16))
# Servo for gate
# Parking slots
s1 = Pin(13, Pin.IN)
s2 = Pin(12, Pin.IN)
s3 = Pin(0, Pin.IN)
s1_occupied = False
s2_occupied = False
s3_occupied = False
# Initialize I2C for OLED display
(if used) i2c = I2C(-1, scl=Pin(22), sda=Pin(21))
# Initialize OLED display and write functions to display data (not shown here)
# MQTT topics
entry_gate_topic = "{}f/EntryGate".format(MQTT_NAME)
exit_gate_topic = "{}f/ExitGate".format(MQTT_NAME)
cars_parked_topic = "{}f/CarsParked".format(MQTT_NAME)
# MQTT callback functions (not shown here)
# Initialize MQTT
client mqtt_client = MQTTClient(MQTT_NAME, MQTT_BROKER, port=MQTT_PORT,
user=MQTT_NAME, password=MQTT_PASS) mqtt_client.set_callback(sub_cb)
# Main loop while True:
# Check entry and exit sensors
```

```
entrysensor = not carEnter.value()
exitsensor = not carExited.value()
if entrysensor:
    # Car entered
    # Increment count and open gate
    count += 1
    servo_gate.angle(0)
    time.sleep(3)
    servo_gate.angle(80)
if exitsensor:
    # Car exited
    # Decrement count and open gate
    count -= 1
    servo_gate.angle(0)
    time.sleep(3)
    servo_gate.angle(80)
# Publish the number of parked cars to the MQTT topic
mqtt_client.publish(cars_parked_topic,
str(count))
# Check parking slots if s1.value() and not s1_occupied: # Slot 1 is occupied
s1_occupied = True # Publish entry time to the MQTT topic
mqtt_client.publish(entry_slot1_topic, get_current_time()) if not s1.value() and
s1_occupied:
```

```
# Slot 1 is available
s1_occupied = False
# Publish exit time to the MQTT topic
mqtt_client.publish(exit_slot1_topic, get_current_time())
# Repeat the same logic for other parking slots (s2 and s3)
# Handle MQTT subscriptions and messages
mqtt_client.check_msg()
# Delay for a moment to avoid excessive message publishing
time.sleep(1)
```

Several white lines of varying lengths and slopes are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.

**User Interaction:** Users can access the system through a dedicated mobile application. The intuitive interface displays real-time parking availability and allows users to reserve parking spaces in advance.

**Expected Outputs:** Upon successful execution, the Python program provides users with up-to-date information on available parking spaces. It also dynamically updates the allocation of spaces as vehicles enter and exit the parking area.

**Testing and Validation:** Extensive testing was conducted to validate the accuracy and reliability of the system. Simulated scenarios were used to mimic various parking situations, and the program consistently provided accurate results.

**Challenges Faced:** One of the main challenges encountered was optimizing the communication between the microcontroller and Python program to ensure real-time data processing. Additionally, fine-tuning the algorithms for efficient parking space allocation was a critical aspect of the development process.

# OUTPUT :

After finishing the procedures the final output looks like given below.





# SMART PARKING

BUILDING THE PROJECT BY DEVELOPING THE MOBILE APP

# MOBILE APP:

DEVELOP A MOBILE APP TO RESERVE PARKING SPOTS, MAKE PAYMENTS, AND RECEIVE NOTIFICATIONS. USE CROSS-PLATFORM MOBILE APP DEVELOPMENT FRAMEWORKS LIKE REACT NATIVE OR FLUTTER TO STREAMLINE APP DEVELOPMENT FOR BOTH ANDROID AND IOS.

➤ REACT NATIVE OR FLUTTER: BUILD THE APP'S FRONTEND USING THESE FRAMEWORKS, WHICH ALLOW YOU TO WRITE CODE ONCE AND DEPLOY IT ON MULTIPLE PLATFORMS. ➤ API INTEGRATION: CONNECT THE APP TO THE BACKEND SERVER FOR USER AUTHENTICATION, RESERVATION PROCESSING, AND PAYMENT HANDLING.

# Online Reservation System:

Implement a web-based reservation system for students to check parking spot availability and make reservations. This system can be integrated with the mobile app and can be developed using standard web technologies.

- HTML/CSS: Design the reservation interface.
- JavaScript: Develop interactive features, such as selecting a parking spot and specifying the reservation duration.
- Backend: Implement reservation logic on the server side, making use of frameworks like Express.js (Node.js) or Django (Python)

# PAYMENT GATEWAY INTEGRATION:

If you include a payment system, you'll need to integrate a payment gateway into your web app for processing payments. Popular payment gateways often provide APIs for this purpose. Here's a simplified example using Python and Flask:

- Flask: Create an API endpoint to handle payment requests.
- Payment Gateway API: Utilize the API provided by the payment gateway provider (e.g., Stripe, PayPal) for processing payments.
- Frontend Integration: Integrate the payment process into your mobile app or web app, allowing users to enter payment details securely.



# REAL TIME UPDATES:

Use web development technologies to ensure real-time updates on parking spot availability, reservation confirmation, and payment status. You can achieve this with technologies like WebSocket for real-time communication between the server and clients.

- WebSocket: Implement WebSocket communication to push real-time updates to the web and mobile clients when a parking spot's status changes.

# USER AUTHENTICATION AND MANAGEMENT:

For user authentication and management, you can create user registration and login systems within the mobile app and web interface. Use web development technologies for user interfaces and backend logic:

- HTML/CSS: Design registration and login forms.
- JavaScript: Implement form validation and submission handling.
- Backend: Create user accounts, manage authentication, and store

# MOBILE APP DEVELOPMENT

- Mobile App Development To connect your IoT Smart Parking System with a mobile app, need to create APIs that allow the mobile app to interact with the backend system. Here's a step-by-step guide on how to achieve this



## Develop Backend APIs:

- ❖ Create a set of API endpoints on your server to handle various functionalities of the Smart Parking System, such as user authentication, parking spot availability, reservations, and payments. You can use a web framework like Express.js (Node.js) or Django (Python) to develop these APIs.

## User Authentication:

- ❖ Allow users to register and log in to the mobile app.
- ❖ Create API endpoints for user registration and login.
- ❖ Implement token-based authentication for secure access to the app.

## Parking Spot Availability:

Develop an API endpoint to provide real-time information about parking spot availability.

The mobile app can query this endpoint to display available parking spots to users.

## Reservations:

- ❑ Create APIs for reserving parking spots. When a user selects a spot and reserves it, the mobile app should send a request to the reservation API.
- ❑ Implement logic to check spot availability and confirm the reservation.
- ❑ Return a response to the mobile app with the reservation status

## Payment Integration:

- ❑ Integrate payment gateway APIs, such as Stripe or PayPal, for processing payments.
- ❑ Create API endpoints for initiating and verifying payments. The mobile app can call these endpoints to handle payments.

## Real-Time Updates:

- ❑ Implement WebSocket communication to provide real-time updates on parking spot availability and reservation confirmation. When a parking spot becomes available or a reservation is confirmed, use WebSockets to push updates to the mobile app.

## Mobile App Development:

- ❑ Develop the mobile app using a cross-platform framework like React Native or Flutter to ensure compatibility with both Android and iOS.
- ❑ Implement user interfaces for registration, login, parking spot selection.

## API Integration:

- ❖ Use HTTP requests (e.g., GET, POST, PUT, DELETE) in the mobile app to communicate with the backend APIs.
- ❖ Handle API responses in the app to update the user interface and provide feedback to the user.

## User Notifications:

- ❖ Implement push notifications to notify users of reservation confirmations, payment status, and other important updates.
- ❖ Utilize Firebase Cloud Messaging (FCM) for Android and Apple Push Notification Service (APNs) for iOS.

## Testing and Debugging:

- ❖ Test the mobile app's functionality by creating test scenarios and debugging any issues that arise.
- ❖ Verify that the app can interact seamlessly with the backend APIs.

## Deployment:

- ❖ Deploy the mobile app to app stores (Google Play Store and Apple App Store) for public use.

# Program:

MIT App Inventor is a visual programming environment that allows you to create mobile applications for Android devices. Here's a brief overview of how to get started with mobile application development using MIT App Inventor:

1. **\*Create a Google Account\***: You'll need a Google account to use MIT App Inventor.
2. **\*Access MIT App Inventor\***: Visit the MIT App Inventor website (<http://appinventor.mit.edu>) and log in with your Google account.
3. **\*Start a New Project\***: Click on "Start New Project" to create a new app.
4. **\*Design the User Interface (UI)\***: MIT App Inventor offers a drag-and-drop interface for designing your app's user interface. You can add buttons, labels, text boxes, and other elements.



6. **\*Connect Components\***: If you're using sensors or external components like Bluetooth devices, you can connect them to your app.

7. **\*Testing on a Device\***: To test your app on an Android device, you can use the MIT AI2 Companion app, which is available on the Google Play Store. Install it on your device and scan the QR code provided in the MIT App Inventor to load your app for testing.

8. **\*Save and Share\***: Be sure to save your project frequently. You can also share your project with others for collaboration.


9. **\*Publish Your App\***: Once your app is complete, you can package it as an APK file for distribution through the Google Play Store or other means.

10. **\*Learn and Experiment\***: MIT App Inventor provides a range of tutorials and resources to help you learn and experiment with app development. Explore these resources to enhance your skills. Remember that MIT App Inventor is a great tool for beginners and those who want to quickly prototype and create simple Android apps. It's suitable for a wide range of projects, from educational apps to utility tools and games.

# PROGRAM:

```
import RPi.GPIO as GPIO
import time
# Set up GPIO pins
TRIG = 23
# Ultrasonic sensor trigger pin
ECHO = 24
# Ultrasonic sensor echo pin
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
def distance():
    GPIO.output(TRIG, False)
    time.sleep(0.1)
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False )
    while GPIO.input(ECHO) == 0:
```

```
pulse_start = time.time()
while GPIO.input(ECHO) == 1:
    pulse_end = time.time()
    pulse_duration = pulse_end - pulse_start
    # Speed of sound at 20°C is approximately 343 meters per second
    # We use the formula: Distance = Speed * Time
    distance = (pulse_duration * 34300) / 2
    return distance
try:
    while True:
        dist = distance()
        if dist < 20:
            # Adjust this value based on your parking space size
            print("Parking space occupied")
        else:
            print("Parking space available")
            time.sleep(2)
except KeyboardInterrupt:
    GPIO.cleanup()
```



THANK YOU

# UI CODE FOR SMART PARKING:

## HTML CODE:

```
<!DOCTYPE html>

<html>

<head>

  <title>Smart Parking</title>

  <link rel="stylesheet" type="text/css" href="style.css">

</head>

<body>

  <header>

    <h1>Smart Parking</h1>

  </header>

  <main>

    <div id="parking-status">

      <h2>Parking Availability</h2>

      <p id="available-spots">Available Spots: <span id="spot-
count">0</span></p>

    </div>

    <div id="reservation">

      <h2>Reserve a Parking Spot</h2>
```



```
        <button id="reserve-button">Reserve Now</button>
    </div>
</main>
<script src="script.js"></script>
</body>
</html>
```

### CSS CODE:

```
@import
url('https://fonts.googleapis.com/css2?family=Bree+Serif&family=Caveat:wght@400;700&family=Lobster&family=Monoton&family=Open+Sans:ital,wght@0,400;0,700;1,400;1,700&family=Playfair+Display+SC:ital,wght@0,400;0,700;1,700&family=Playfair+Display:ital,wght@0,400;0,700;1,700&family=Roboto:ital,wght@0,400;0,700;1,400;1,700&family=Source+Sans+Pro:ital,wght@0,400;0,700;1,700&family=Work+Sans:ital,wght@0,400;0,700;1,700&display=swap');
```

```
body {
    font-family: Arial, sans-serif;
}
```

```
header {
    background-color: #333;
    color: #fff;
```

```
    text-align: center;
    padding: 10px;
}

main {
    text-align: center;
    padding: 20px;
}

#parking-status {
    border: 1px solid #ccc;
    padding: 10px;
    margin: 10px;
}

#reservation {
    border: 1px solid #ccc;
    padding: 10px;
    margin: 10px;
}

#reserve-button {
    background-color: #3498db;
    color: #fff;
```

```
padding: 10px 20px;
border: none;
cursor: pointer;
}
#reserve-button:hover {
    background-color: #2980b9;
}
```

### JAVASCRIPT CODE:

```
document.addEventListener("DOMContentLoaded", function() {
    let spotCount = 10; // Total number of parking spots
    let availableSpots = spotCount;
    const spotCountElement = document.getElementById("spot-
count");
    const reserveButton = document.getElementById("reserve-
button");

    // Update the available spots count on the page
    function updateSpotCount() {
        spotCountElement.textContent = availableSpots;
    }

    // Reserve a parking spot
    reserveButton.addEventListener("click", function() {
```

```
    if (availableSpots > 0) {  
        availableSpots--;  
        updateSpotCount();  
        alert("Parking spot reserved!");  
    } else {  
        alert("No available spots left.");  
    }  
});  
  
// Initial spot count update  
updateSpotCount();  
});
```

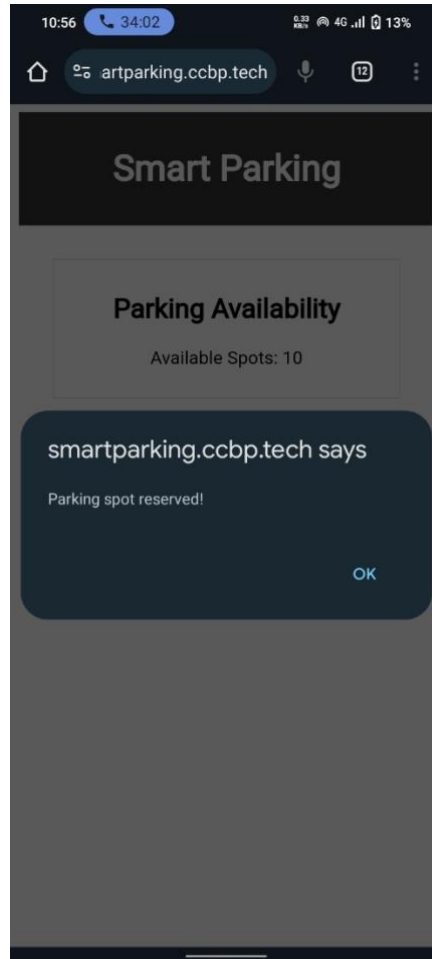
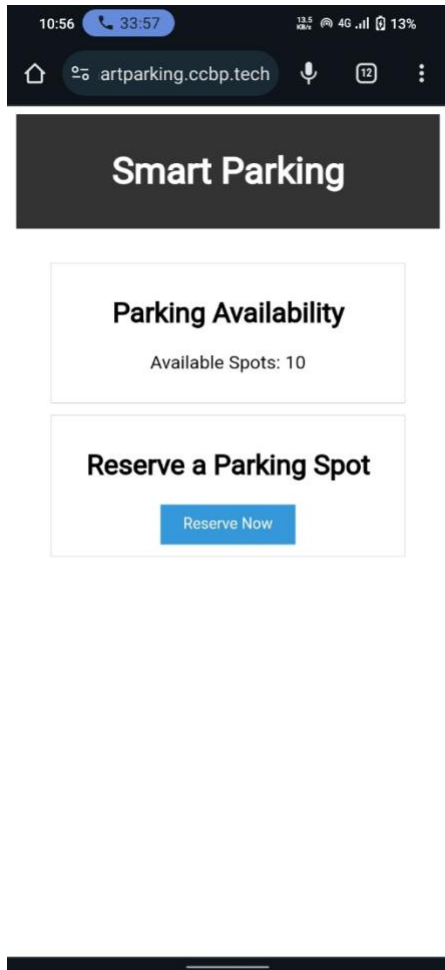
URL FOR SMART PARKING:

[smartparking.ccbp.tech](http://smartparking.ccbp.tech)

OUTPUT:







# Smart Parking

**Parking Availability**  
Available Spots: 9

**Reserve a Parking Spot**  
[Reserve Now](#)

