PUBLIC TRANSPORTATION OPTIMIZATION

Project Name: smart car parking system

Project objectives

The objectives of a smart car parking system project typically include:

1. Efficient Space Utilization: To optimize the use of parking spaces, reducing congestion and increasing parking availability.

2. Convenience for Users: To provide a user-friendly interface for drivers to find, reserve, and pay for parking spots.

3. Real-time Monitoring: To enable real-time monitoring of parking occupancy and issue alerts for maintenance and security.

4. Sustainability: To reduce emissions and fuel consumption by minimizing the time spent searching for parking.

5. Revenue Generation: To maximize revenue for parking operators through efficient use of space and dynamic pricing.

6. Security and Safety: To enhance security and safety through surveillance, lighting, and emergency response features.

7. Data Analysis: To collect data for analysis and optimization of parking operations and city planning.

8. Integration with Urban Infrastructure: To integrate the system with other urban infrastructure like traffic signals and public transportation.

9. Accessibility: To ensure that the system is inclusive and accessible to all, including disabled users.

10. Cost-effectiveness: To implement the system in a cost-effective manner and achieve a return on investment.

These objectives can vary based on the specific goals and requirements of the project.

Device setup

Components:

IR sensor
7805 voltage regulator
LEDs
Raspberry pi
1k ohm Resistor
Bread board
Connecting wires

Circuit Connections:

IR Sensor:
• Connect Vcc of the IR sensor to the output pin of the 7805 voltage regulator.

• Connect GND of the IR sensor to the ground rail of the circuit.

• Connect the OUT pin of the IR sensor to a GPIO pin on your microcontroller (e.g., Raspberry Pi).

7805 Voltage Regulator:

• Connect the input pin of the 7805 to your power source (e.g., 9V battery or a suitable power supply).

• Connect the ground pin of the 7805 to the ground rail of the circuit.

• Connect the output pin (5V) of the 7805 to the Vcc pins of the IR sensor and LEDs.

LEDs:
• Connect the anode (longer leg) of each LED through a current-limiting resistor to the 5V output of the 7805.

• Connect the cathode (shorter leg) of each LED to a separate GPIO pin on your microcontroller (these will be used to control the LEDs).

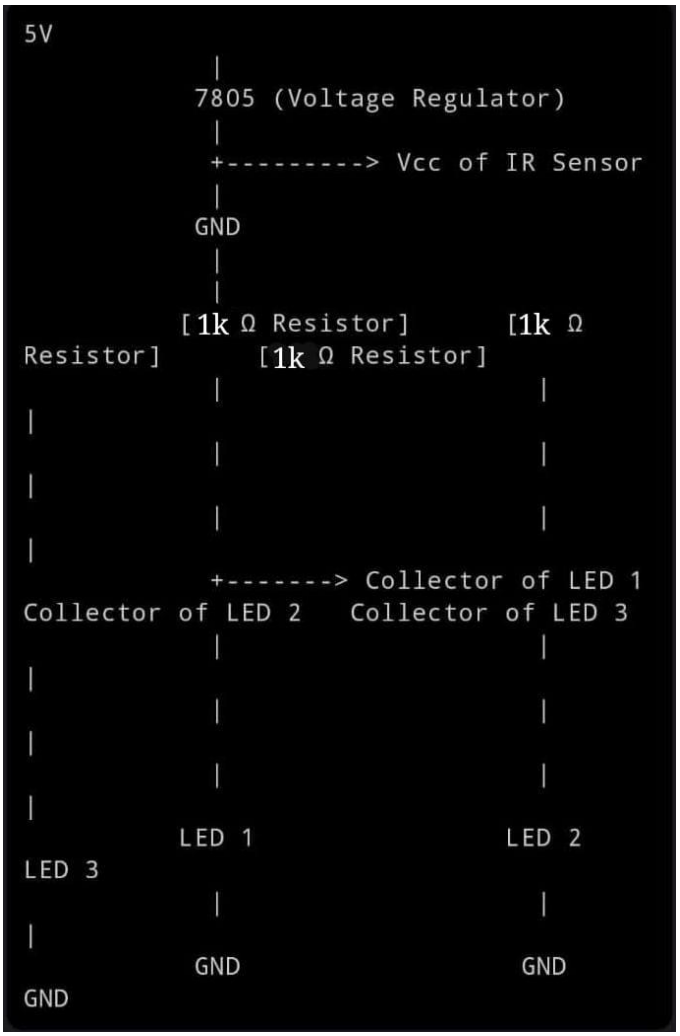
Important Notes:

1. Make sure to use appropriate resistors to limit the current through the LEDs.

2. The value of the resistor will depend on the specific LEDs you're using and the voltage supplied.

3. Double-check the pinout of your IR sensor and LEDs. Different models may have slightly different configurations.

4. If you're using a microcontroller like Raspberry Pi, remember to set up the GPIO pins in your code to match the connections you've made.



Circuit setup

platform development process:

1. Hardware Selection:
- Choose appropriate sensors (e.g., ultrasonic, camera, lidar) to detect available parking spaces and monitor vehicle movement.
- Select communication modules (e.g., Wi-Fi, Bluetooth, LoRa) for data transmission.

2. Software Development:
- Develop a central control software that processes data from sensors and manages the parking system.
- Create a user-facing mobile app or web interface for drivers to find and reserve parking spots.
- Implement algorithms for space detection, vehicle recognition, and reservation management.
- Consider integrating payment processing for parking fees.

3. Data Storage and Processing:
- Set up a database to store information about parking spaces, reservations, and user accounts.
- Implement data analytics to monitor parking space occupancy and usage patterns.

4. Connectivity:
- Ensure robust and secure connectivity between sensors, control software, and user interfaces.
- Consider using cloud services for remote monitoring and updates.

5. Security:
- Implement security measures to protect user data, payment transactions, and the system from unauthorized access.

6. User Experience:
- Focus on creating a user-friendly interface for drivers to easily find, reserve, and pay for parking.
- Consider offering real-time navigation to the reserved parking spot.

7. Integration with Payment Systems:
- Integrate payment gateways for users to pay for parking through the app or website.

8. Maintenance and Updates:
- Plan for regular maintenance and updates to ensure the system's reliability and security.

9. Testing and Quality Assurance:
- Thoroughly test the system for accuracy in space detection, reservation handling, and user interactions.

10. Regulatory Compliance:
- Ensure compliance with local regulations and safety standards, especially if autonomous parking features are included.

11. Scalability:
- Design the platform to be scalable, allowing for the addition of more parking spaces and sensors as needed.

12. User Support and Training:
- Provide customer support for users and training for operators who manage the system.

13. Monitoring and Analytics:
- Implement monitoring and analytics tools to track system performance, user behavior, and revenue generation.

14. Deployment:
- Install hardware components in parking areas and deploy the software to servers and user devices.

15. Feedback and Improvement:
- Continuously gather user feedback and make improvements to enhance the system's efficiency and user satisfaction.

Coding implemention

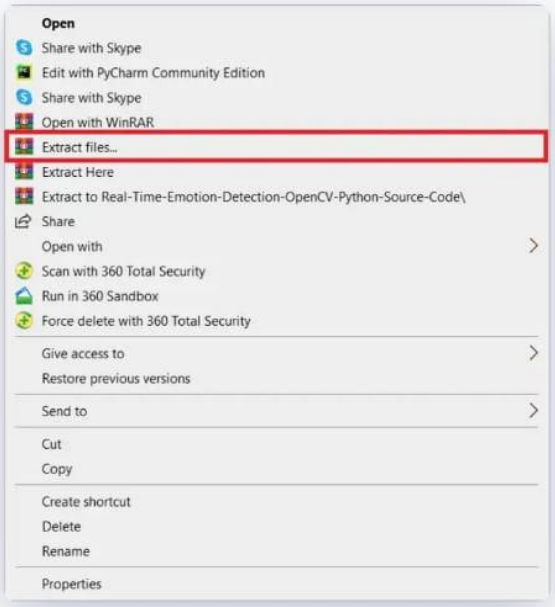
```
import RPi.GPIO as GPIO
import time
# Set up GPIO
GPIO.setmode(GPIO.BCM)
# Define pin numbers
IR_SENSOR_PIN = 17 # Example GPIO pin, adjust as needed
LED1_PIN = 18
LED2_PIN = 19
LED3_PIN = 20
# Initialize GPIO pins
GPIO.setup(IR_SENSOR_PIN, GPIO.IN)
GPIO.setup(LED1_PIN, GPIO.OUT)
GPIO.setup(LED2_PIN, GPIO.OUT)
GPIO.setup(LED3_PIN, GPIO.OUT)
def check_parking_status():
if GPIO.input(IR_SENSOR_PIN) == GPIO.LOW:
return True # Car detected
else:
return False # No car detected
def indicate_parking_status(status):
if status:
GPIO.output(LED1_PIN, GPIO.HIGH) # LED1 ON
GPIO.output(LED2_PIN, GPIO.LOW) # LED2 OFF
GPIO.output(LED3_PIN, GPIO.LOW) # LED3 OFF
else:
GPIO.output(LED1_PIN, GPIO.LOW) # LED1 OFF
GPIO.output(LED2_PIN, GPIO.HIGH) # LED2 ON
GPIO.output(LED3_PIN, GPIO.LOW) # LED3 OFF
try:
while True:
status = check_parking_status()
indicate_parking_status(status)
time.sleep(1)
except KeyboardInterrupt:
GPIO.cleanup()
```

Steps on how to run smart car parking system in python with source code

Project in Python with Source Code

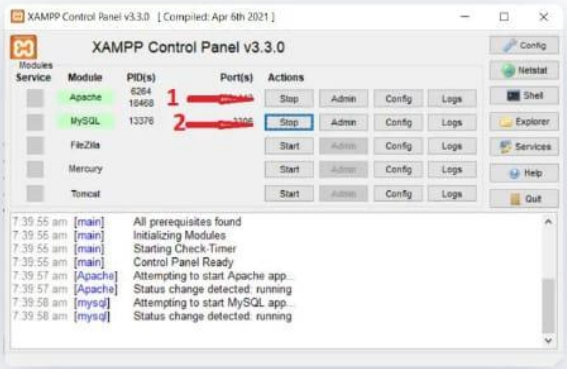
- **Step 1: Download source code.**
  First, download the source code given below.

- **Step 2: Extract file.**
  Next, after you finished download the source code, extract the zip file.



- **Step 3: Open Xampp**

  Next, open xampp and start **“apache”** and **“mysql“**.

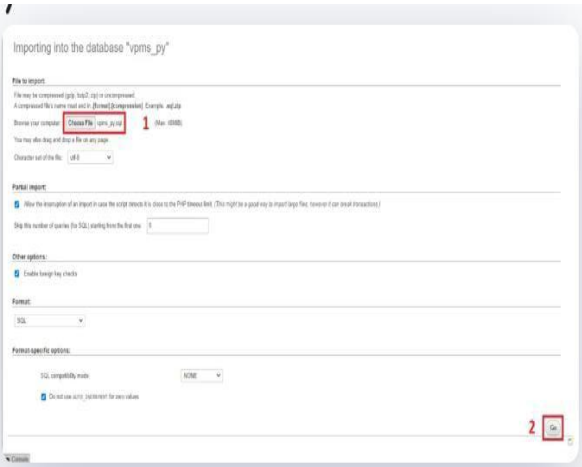


- **Step 4: Create new Database**

  Next, open any browser and type **“localhost/phpmyadmin”**.

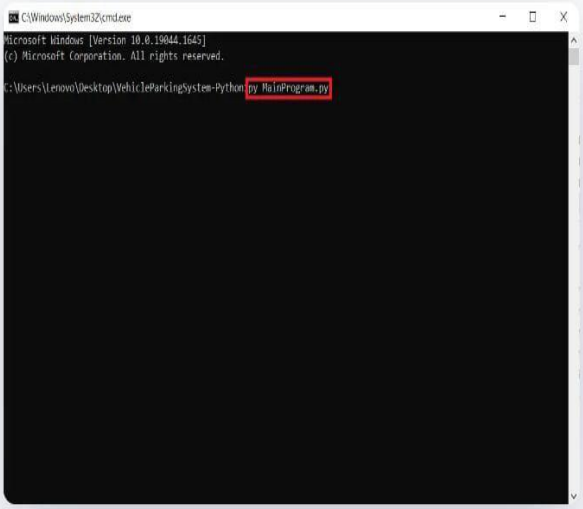
- **Step 5: Import SQL File**

  Next, Import the given sql file to your database you’ve create.



- **Step 6: Run Project**

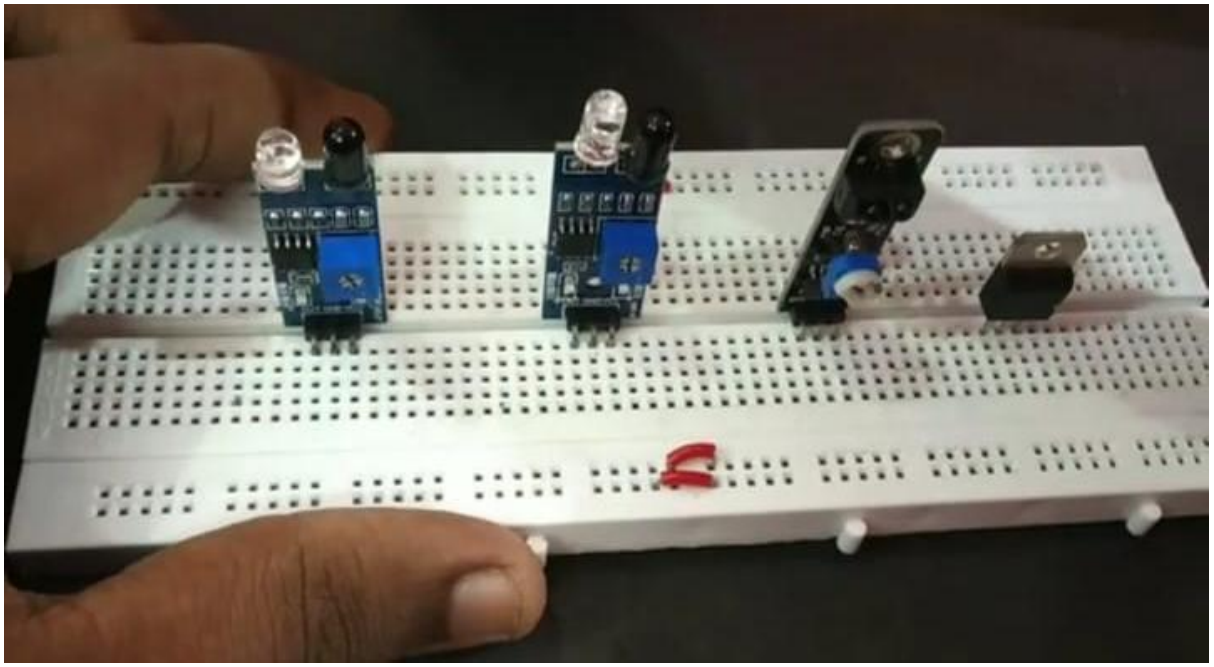
  Last, open the project folder and run the project.

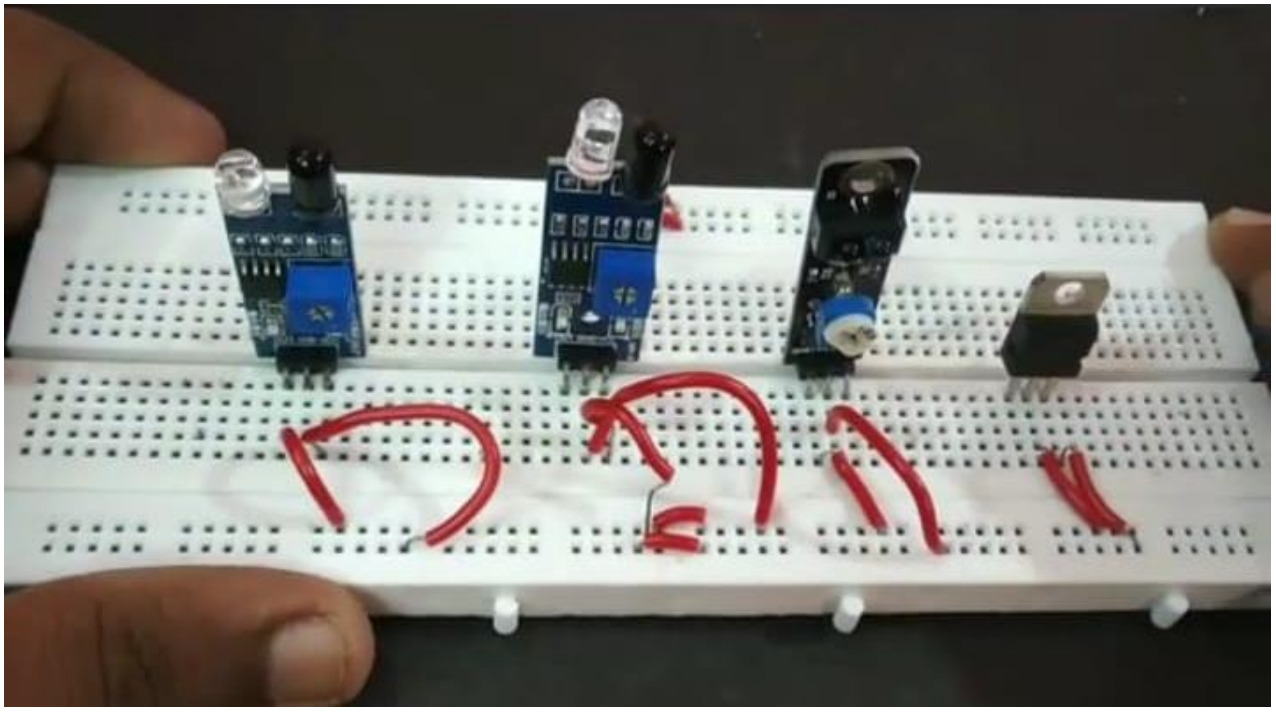


Schematic

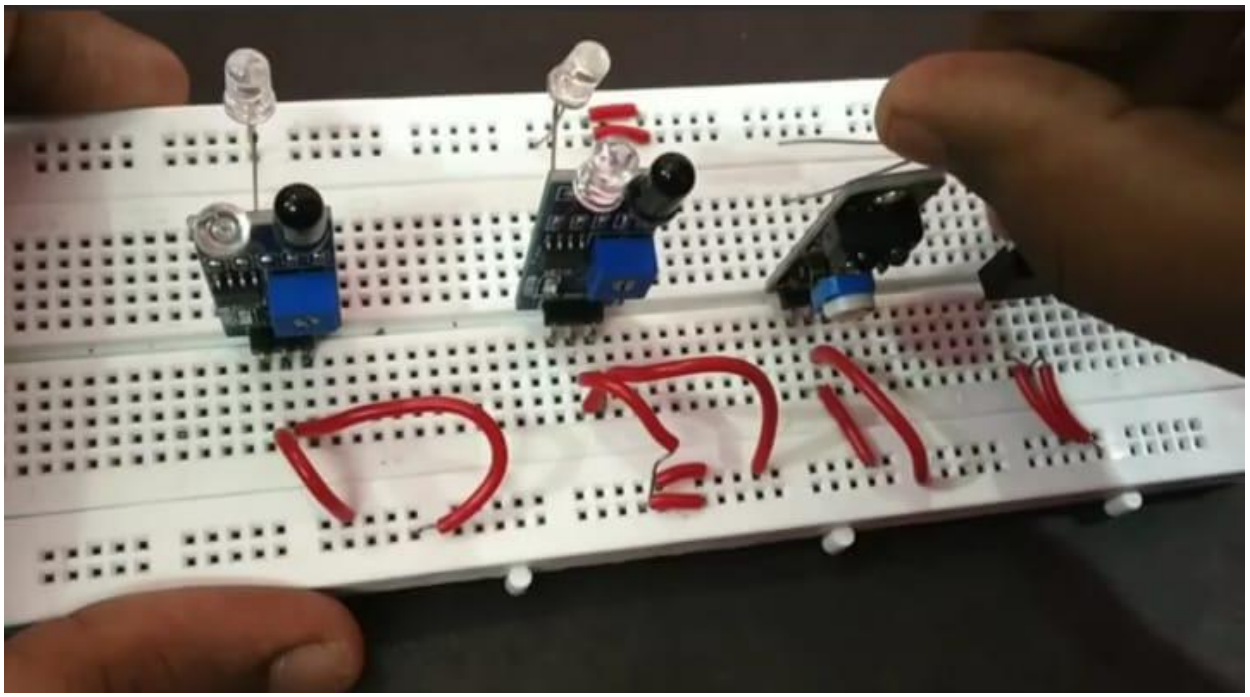
1.Connect the input of the 7805 voltage regulator to your power source (e.g., a battery or power supply).
2.Connect the ground (GND) pin of the 7805 voltage regulator to the system ground.
3.Connect the output (OUT) pin of the 7805 voltage regulator to provide regulated 5V power.
5.Connect the IR sensor's VCC to the regulated 5V output.
6.Connect the IR sensor's GND to the system ground.
7.Connect the IR sensor's OUT pin to a digital input on a microcontroller (not shown in the schematic).
7.Connect the cathode (shorter lead) of the three LEDs to the ground.
8.Connect the anodes (longer leads) of the red, yellow, and green LEDs to separate GPIO pins on a microcontroller (not shown).
8.Connect a 1k ohm resistor in series with each LED's anode to limit current.
9.Connect the common cathode of the LEDs to the regulated 5V output.
This is a basic schematic, and you would need to incorporate a microcontroller and programming to control the LEDs and interpret the IR sensor data for parking status.

Project setup with IoT devices

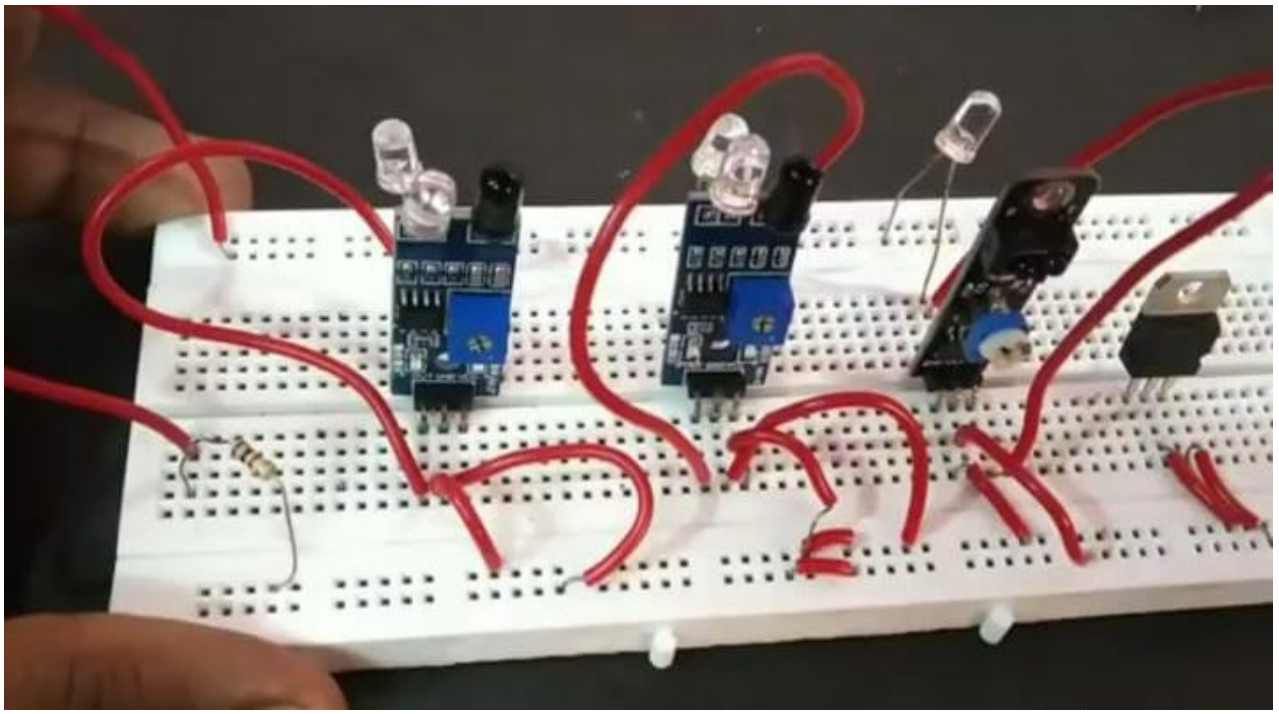




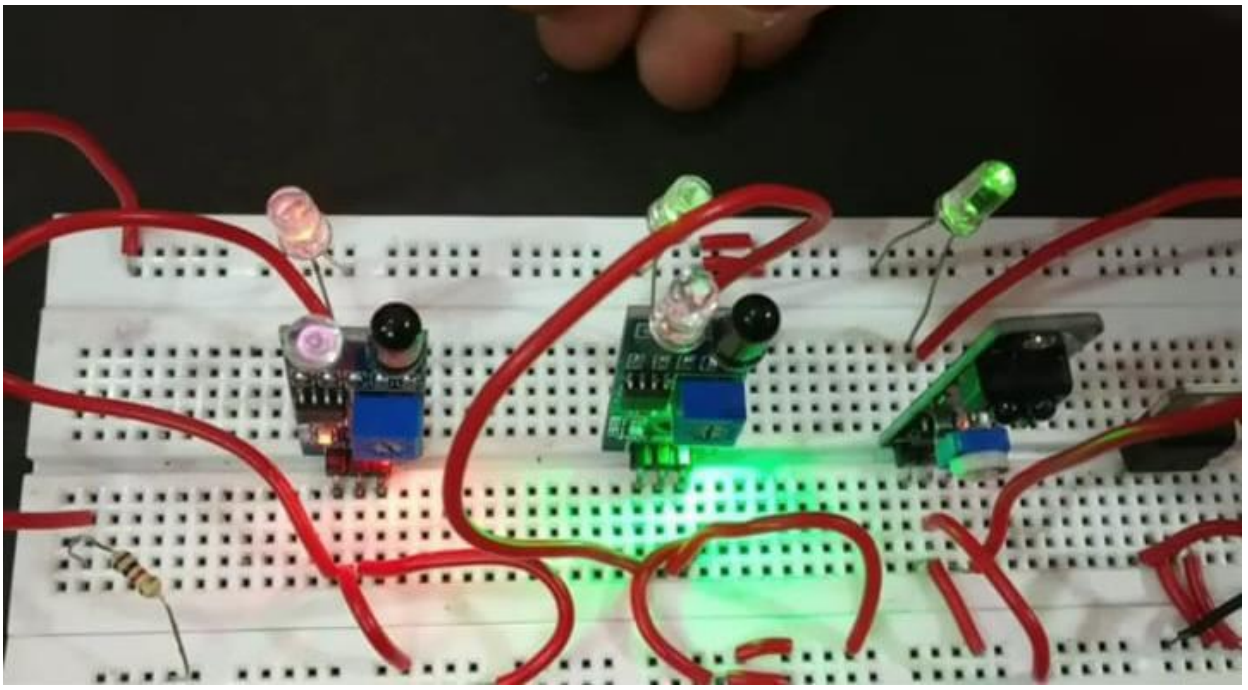
Connecting wires with sensors



Fix LEDs with sensors



Connect resistor in breadboard



Devices are connected

Findings of the smart car parking system

This Python script is designed for a Raspberry Pi using the RPi.GPIO library to monitor an infrared (IR) sensor to detect if a car is parked or not. It uses three LEDs to indicate the parking status.

Here's a summary of what the code does:

1. Sets up the GPIO mode to use BCM numbering.

2. Defines the GPIO pins for an IR sensor and three LEDs.

3. Initializes the GPIO pins for input and output.

4. Defines a function `check_parking_status()` to check the status of the IR sensor and return `True` if a car is detected (IR sensor input is LOW), or `False` if no car is detected (IR sensor input is HIGH).

5. Defines a function `indicate_parking_status(status)` to control the LEDs based on the parking status. If a car is detected (status is `True`),

the first LED is turned on, and the other two are turned off. If no car is detected (status is `False`), the second LED is turned on, and the other two are turned off.

6. Sets up an infinite loop to continuously monitor the parking status by calling `check_parking_status()`, and update the LEDs using `indicate_parking_status()`. It also includes a `time.sleep(1)` to introduce a 1-second delay between status checks.

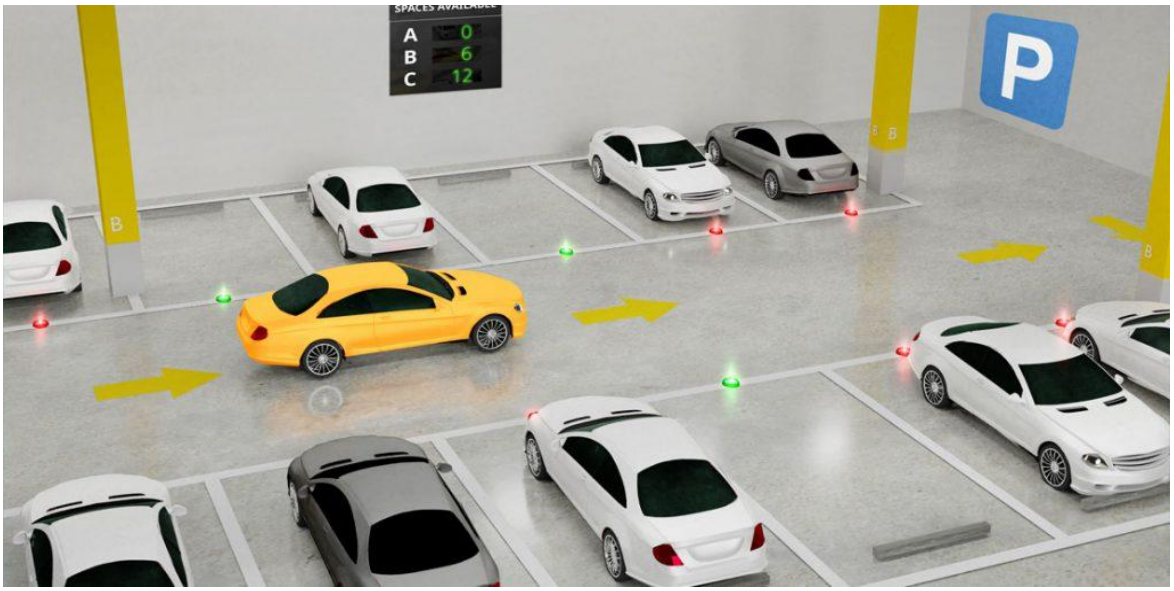
7. It has an exception handler for a keyboard interrupt (Ctrl+C) to ensure GPIO cleanup when the script is terminated.

you would need to run this script on a Raspberry Pi with the appropriate hardware connections and the RPi.GPIO library installed. The LEDs will indicate the parking status based on the input from the IR sensor.

Deliverable:

The IR sensor is used to detect the presence of a car. If a car is detected, LED1 lights up; otherwise, LED2 lights up. LED3 remains off. Make sure to adjust the GPIO pin numbers and connections according to your specific setup.

Remember to use appropriate current-limiting resistors for the LEDs to prevent them from burning out. Additionally, ensure that your power supply can handle the load of the IR sensor and the LEDs.



Conclusion

In conclusion, a smart car parking system offers numerous benefits, including improved efficiency, reduced traffic congestion, and enhanced user experience. By utilizing technologies like sensors, mobile apps, and automation, it streamlines the parking process and contributes to a more sustainable and convenient urban environment. As cities continue to grow and face increasing challenges related to limited parking spaces, the implementation of smart parking systems becomes increasingly important for a smarter, more connected future.

★"A parking revolution: Where innovation meets efficiency."★