SMART PARKING SYSTEM

MS.R.PREETHI.AsP/ECE
MENTOR

BY:
R.MONIKA.

ABSTRACT

The rapid urbanization and escalating vehicular population in contemporary cities have exacerbated the perennial challenge of parking management. To address this issue, we propose an innovative IoT-based Smart Parking System designed to revolutionize urban mobility. This project leverages cutting edge sensor technologies, real-time data processing, and user-friendly interfaces to provide a seamless parking experience for both drivers and parking operators. The rapid urbanization and escalating vehicular population in contemporary cities have exacerbated the perennial challenge of parking management. To address this issue, we propose an innovative IoT-based Smart Parking System designed to revolutionize urban mobility. This project leverages cutting edge sensor technologies, real-time data processing, and user-friendly interfaces to provide a seamless parking experience for both drivers and parking operators.

INTRODUCTION

The smart parking system project aims to develop a cutting-edge solution to address the challenges associated with urban parking management.

This system leverages advanced technologies to optimize parking availability, enhance user experience, and improve overall traffic flow within urban areas.

PROPASAL

- ➢ INTEGRATION APPROACH
- ➢ COMPONENTS
- ➢ REAL TIME TRANSMIT INFORMATION PLATFORM
- ➢ PYTHON CODE

Integration Steps:

1. **Sensor Installation**:
   - Install sensors in each parking space to detect vehicle presence. These sensors will send data to the microcontroller.
2. **Microcontroller Setup**:
   - Connect sensors to the microcontroller and program it to read data from the sensors. Process this data to determine the availability of parking spaces.
3. **Communication Setup**:
   - Establish a communication link between the microcontroller and the central server. This can be achieved using Wi-Fi, Bluetooth, or GSM modules.
4. **Data Transmission**:
   - Configure the microcontroller to send parking availability data to the central server at regular intervals or whenever there's a change in status.
5. **Data Processing**:
   - The central server receives the data and processes it. It updates the parking availability status in real-time.
6. **Database Management**:
   - Store parking information in a database, including space availability, location, and any user-related data (like reservations).

7. **User Interface Development**:
    - Create a mobile or web application for users to access parking information. The app should display real-time availability, location, and potentially allow for reservations.
8. **User Authentication and Payment (Optional)**:
    - Implement user authentication for tracking and managing reservations. Integrate a payment gateway if paid parking is involved.
9. **Notifications**:
    - Configure the system to send notifications to users, such as alerts when a parking space is about to become available.
10. **Feedback and Reporting**:
    - Allow users to provide feedback on their parking experience through the application. This data can be valuable for system optimization.
11. **Security**:
    - Implement security measures to protect the system from unauthorized access or tampering.
12. **Testing and Optimization**:
    - Thoroughly test the system to ensure all components are working as expected. Optimize the system for performance, considering factors like response time, data accuracy, and power consumption.
13. **Deployment**:
    - Install the system in the target parking area, ensuring all components are properly connected and configured.
14. **Maintenance and Monitoring**:
    - Regularly monitor the system for any issues and perform maintenance tasks as needed. This might include sensor calibration, software updates, or hardware replacements.

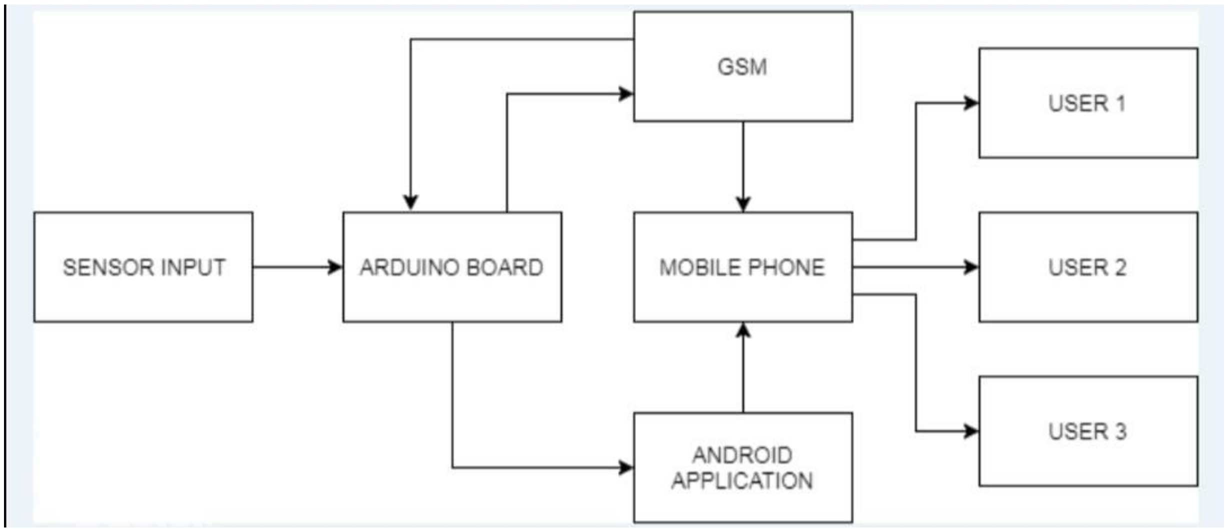
Components Needed:

1. **Parking Sensors**:
    - Ultrasonic or infrared sensors to detect the presence of vehicles in parking spaces.
2. **Microcontroller/Single Board Computer**:
    - Raspberry Pi, Arduino, or similar, to control the sensors and handle data processing.
3. **Communication Module**:
    - Wi-Fi, Bluetooth, or GSM module for transmitting data to the central server.
4. **Central Server**:

- A cloud-based server for data processing, storage, and management.

5. **Mobile/Web Application**:
    - User interface for drivers to access parking information.

BLOCK DIAGRAM



COMPONENTS

➢ This smart parking system project consists of Arduino, six IR sensors, one servo motor, and

one LCD display.

➢ Where the Arduino is the main microcontroller that controls the whole system.

➢ Two IR sensors are used at the entry and exit gates to detect vehicle entry and exit in the parking area.

- And other four IR sensors are used to detect the parking slot availability.
- The servo motor is placed at the entry and exit gate that is used to open and close the gates.
- Also, an LCD display is placed at the entrance, which is used to show the availability of parking slots in the parking area

SENSORS

- Infrared (IR) sensor is also a part of the WSNs technology and it is commonly used in developing a smart parking system.

- IR sensor is used to detect obstacles by emitting radiation.

- It is also known as the general-purpose proximity sensor.

- IR sensor can sense or measure the heat and the motion of an object.

ARDUINO BOARD

- The Arduino Uno is used to create a smart car parking system.

- The device uses IR sensors mounted in the parking slots to detect empty slots and assists the driver in finding parking in a new city.

- The system lacks a payment mechanism as well as guide technology that can automatically find available parking spaces.

Working:

Sensor Integration :

Infrared sensors are installed at parking spaces to detect the presence of vehicles. These sensors can measure the distance between the sensor and the vehicle.

Data Collection:

Arduino microcontrollers are used to collect data from the sensors. The sensors send signals to the Arduino to indicate whether a parking spot is occupied or vacant.

Data Processing :

The Arduino processes the sensor data to determine the status of each parking space. It calculates if a vehicle is parked or not by analyzing the distance measurements from the sensors.

Display:

The system often includes LED displays or LCD screens to show real-time information about parking space availability. Green lights might indicate vacant spaces, and red lights occupied ones.

Source Code:

```
import colorama

from termcolor import colored

options_message = """
Choose:
1. To park a vehicle
2. To remove a vehicle from parking
3. Show parking layout
4. Exit
"""


class Vehicle:
    def init(self, v_type, v_number):

        self.v_type = v_type

        self.v_number = v_number

        self.vehicle_types = {1: 'c', 2: 'b', 3: 't'}

def str(self):

 return self.vehicle_types[self.v_type]

class Slot:

def init(self):

        self.vehicle = None  @property

    def is_empty(self):

        return self.vehicle is None

class Parking:

 def init(self, rows, columns):

        self.rows = row

  self.columns = columns

        self.slots = self._get_slots(rows, columns)

def start(self):

        while True:

            try:

                print(options_message)

                option = input("Enter your choice: ")
```

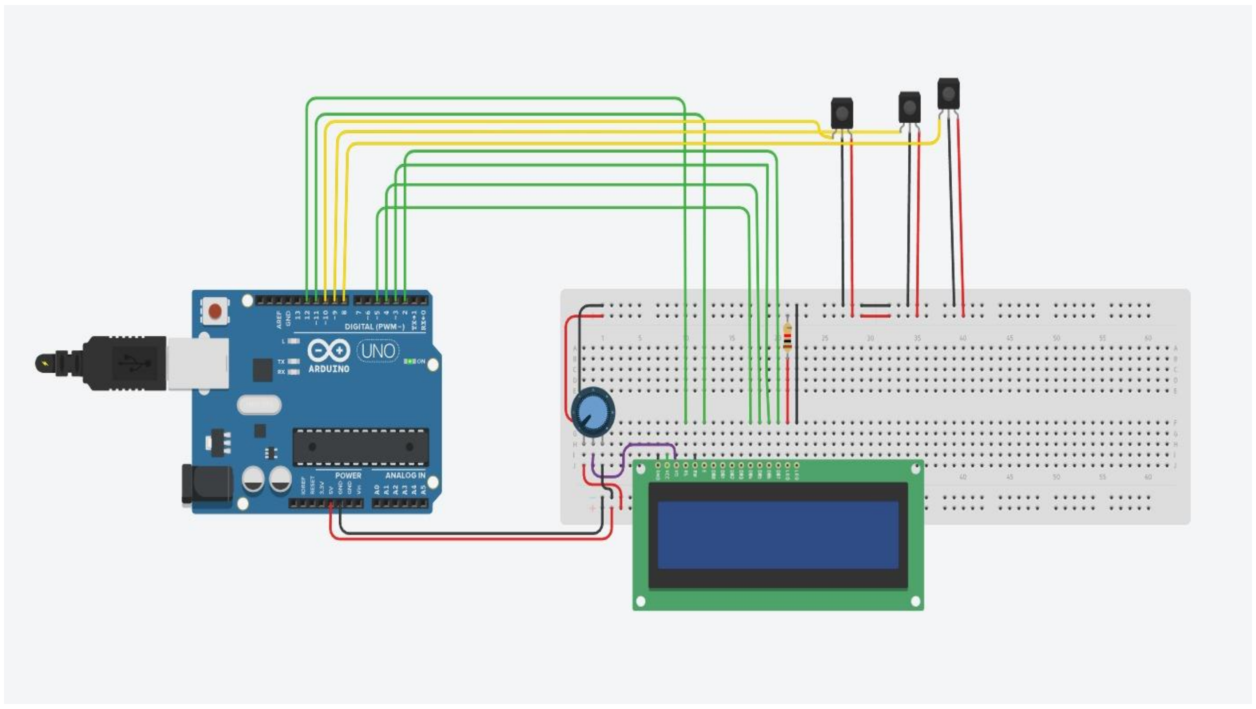
```
             if option == '1':

                 self._park_vehicle()

            if option == '2':

                 self._remove_vehicle()

            if option == '3':

                 self.show_layout()

          if option == '4':

                 break

          except ValueError as e:

              print(colored(f"An error occurred: {e}. Try again.", "red"))

print(colored("Thanks for using our parking assistance system", "green"))

def _park_vehicle(self):

       vehicle_type = self._get_safe_int("Available vehicle types: 1. Car\t2. Bike\t3.
Truck.\nEnter your choice: ")

  if vehicle_type not in [1, 2, 3]:

           raise ValueError("Invalid vehicle type specified")

 vehicle_number = input("Enter vehicle name plate: ")

       if not vehicle_number:

           raise ValueError("Vehicle name plate cannot be empty.")

       vehicle = Vehicle(vehicle_type, vehicle_number)

 print('\n')

       print(colored(f"Slots available: {self._get_slot_count()}\n", "yellow"))
```

```
        self.show_layout()

        print('\n')

  col = self._get_safe_int("Enter the column where you want to park the vehicle: ")

        if col <= 0 or col > self.columns:

  raise ValueError("Invalid row or column number specified")  row =
self._get_safe_int("Enter the row where you want to park the vehicle: ")

        if row <= 0 or row > self.rows:

            raise ValueError("Invalid row number specified")

              slot = self.slots[row-1][col-1]

            if not slot.is_empty:

            raise ValueError("Slot is not empty. Please choose an empty slot.")

  slot.vehicle = vehicle

def _remove_vehicle(self):

        vehicle_number = input("Enter the vehicle number that needs to be removed
from parking slot: ")

        if not vehicle_number:

            raise ValueError("Vehicle number is required.")

  for row in self.slots:

            for slot in row:

                if slot.vehicle and slot.vehicle.v_number.lower() ==
vehicle_number.lower():

                    vehicle: Vehicle = slot.vehicle

                    slot.vehicle = None
                    print(colored(f"Vehicle with number '{vehicle.v_number}' removed
from parking", "green"))

                    return

        else:

            raise ValueError("Vehicle not found.")

def show_layout(self):

        col_info = [f'<{col}>' for col in range(1, self.columns + 1)]

        print(colored(f"|{''.join(col_info)}|columns", "yellow"))
self._print_border(text="rows")

  for i, row in enumerate(self.slots, 1):

            string_to_printed = "|"

            for j, col in enumerate(row, 1):

                string_to_printed += colored(f"[{col.vehicle if col.vehicle else ' '}]",  "red"
if col.vehicle else "green")

            string_to_printed += colored(f"|<{i}>", "cyan")

            print(string_to_printed)

self._print_border()

def _print_border(self, text=""):

        print(colored(f"|{'-' * self.columns * 3}|{colored(text, 'cyan')}", "blue"))

  def _get_slot_count(self):

        count = 0

        for row in self.slots:

            for slot in row:
```

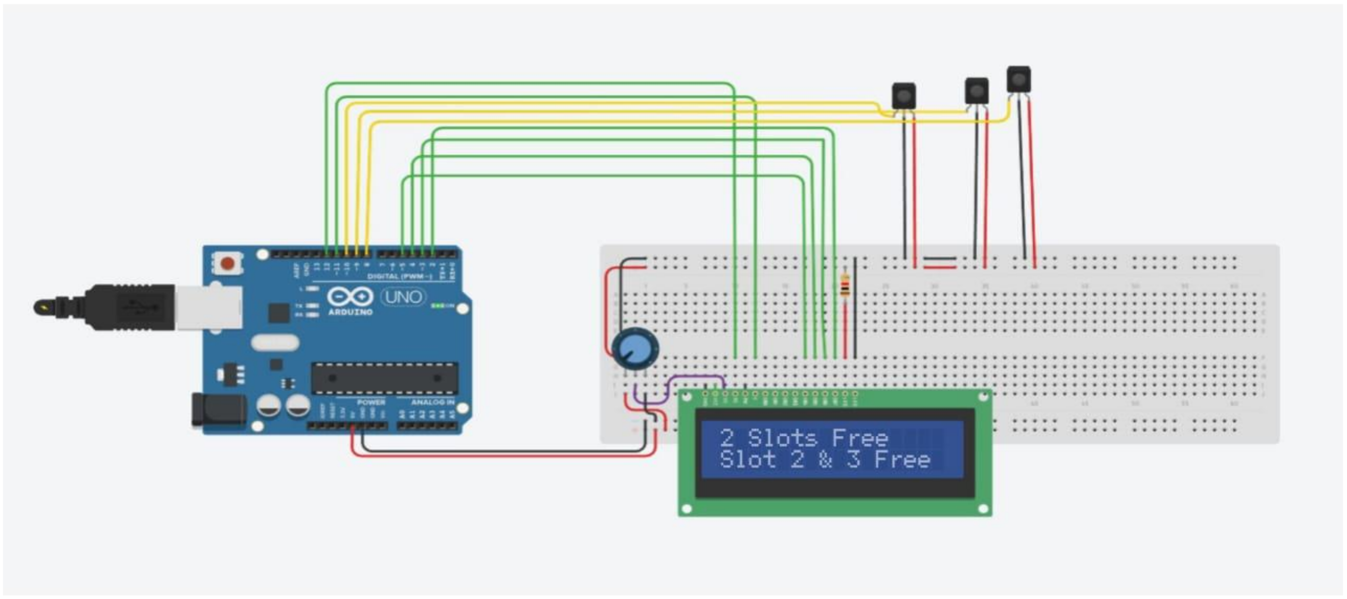
```
            if slot.is_empty:
                count += 1
        return count
 @staticmethod
    def _get_slots(rows, columns):
        slots = []
        for row in range(0, rows):
            col_slot = []
            for col in range(0, columns):
                col_slot.append(Slot())
            slots.append(col_slot)
        return slots
@staticmethod
    def _get_safe_int(message):
        try:
            val = int(input(message))
            return val
        except ValueError:
            raise ValueError("Value should be an integer only")
def main():
    try:
        print(colored("Welcome to the parking assistance system.", "green"))
        print(colored("First let's setup the parking system", "yellow"))

        rows = int(input("Enter the number of rows: "))

        columns = int(input("Enter the number of columns: "))     print("Initializing
parking")

        parking = Parking(rows, columns)

        parking.start()

 except ValueError:

        print("Rows and columns should be integers only.")

  except Exception as e:

        print(colored(f"An error occurred: {e}", "red"))

if name == 'main':

    colorama.init()  # To enable color visible in command prompt

    main()
```

CONCLUSION:

BEFORE SIMULATION:



AFTER SIMULATION:



the Smart Parking System utilizing IoT represents a forward-thinking solution to urban congestion and parking challenges. Through the integration of advanced technologies, it not only streamlines the parking process but also promotes sustainability, efficiency, and improved quality of life for urban residents and visitors alike. As cities continue to grow and evolve, the implementation of such intelligent systems is poised to play a crucial role in shaping the future of urban mobility.