SMART PARKING SYSTEM

Done by

Balaji M

Praveenkanth.M

Likkith Kumar M

Rohith P

INTRODUCTION:

Welcome to the future of parking! The IoT Smart Parking System leverages cutting-edge technology to make parking easier and more efficient than ever before. Say goodbye to the frustration of finding a parking spot – this system provides real-time insights and solutions for a hassle-free parking experience

PROBLEM DEFINITION:

In today's conventional parking systems, one of the major challenges is the lack of real-time information regarding the number of vehicles parked and the availability of parking slots. This absence of visibility creates several problems:

1. Inefficient Space Utilization: Without accurate data on parking occupancy, valuable parking spaces often remain unused while drivers struggle to find a spot, leading to wasted resources and

congestion.

2. Frustration and Time Wastage:Drivers spend valuable time circling parking lots, adding to traffic congestion and frustration. This inefficiency also contributes to increased fuel consumption and environmental impact.
3. Revenue Loss for Operators: Parking operators miss out on potential revenue as they cannot optimize pricing based on demand. They also lack data for effective management and planning.
4. Environmental Impact: The increased time spent searching for parking leads to greater emissions and environmental harm, impacting air quality and sustainability.
5. Inadequate Security: Limited visibility makes it challenging to monitor and ensure the security of parked vehicles.

In light of these issues, a Smart Parking System based on IoT and AI technologies emerges as a compelling solution to transform the parking experience and address these pressing challenges

DESIGN THINKING:

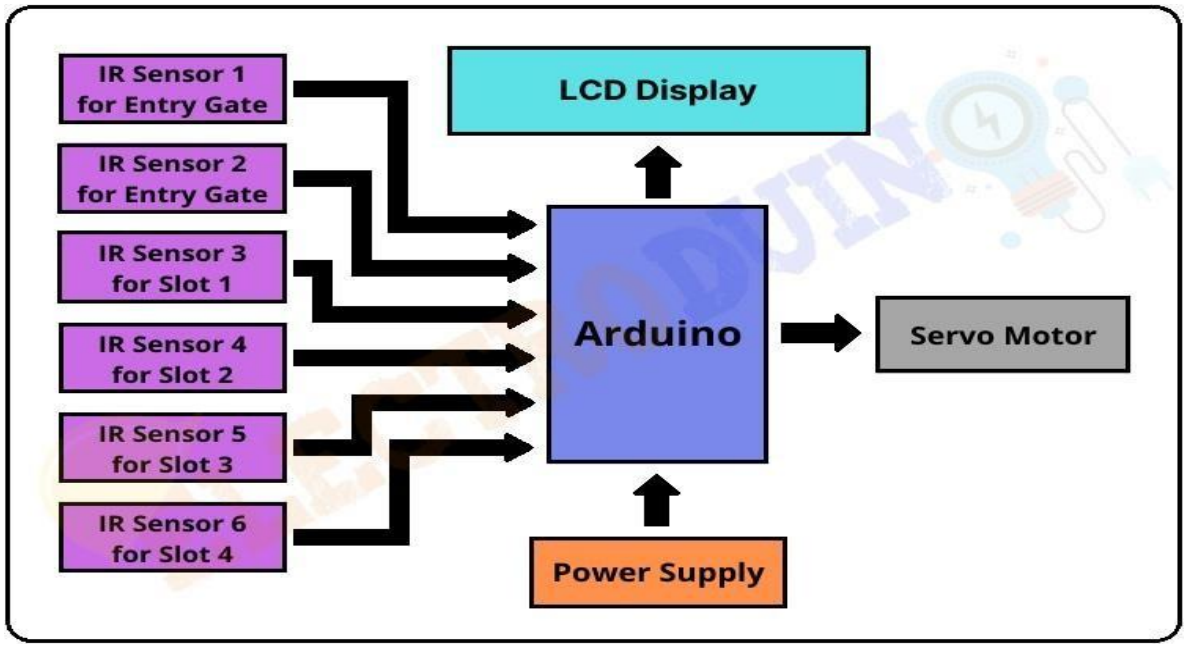
The idea is to deploy IoT sensors in each parking space across the campus that can detect the presence of vehicles. These sensors will continuously monitor the occupancy status and transmit the data to a central server. Users can access this data through a mobile application, which will display real-time parking space availability, help users find parking spots, and provide navigation to the available spaces.

PROTOTYPE:

Hardware: You will need ultrasonic sensors or other suitable sensors to detect parking space occupancy.
Backend Server: A server to collect and process data from the sensors, store information about parking space availability, and serve this information to the Android application.
Android Application: An Android app to display real-time parking space availability to users.

WORKING MODEL:
When a user opens the app, they can view the campus map with color-coded parking zones. Tapping on a specific zone reveals detailed information about each parking space in that zone. Users can set up notifications for preferred zones or spaces and receive alerts when a parking spot becomes available.

BLOCK DIAGRAM:

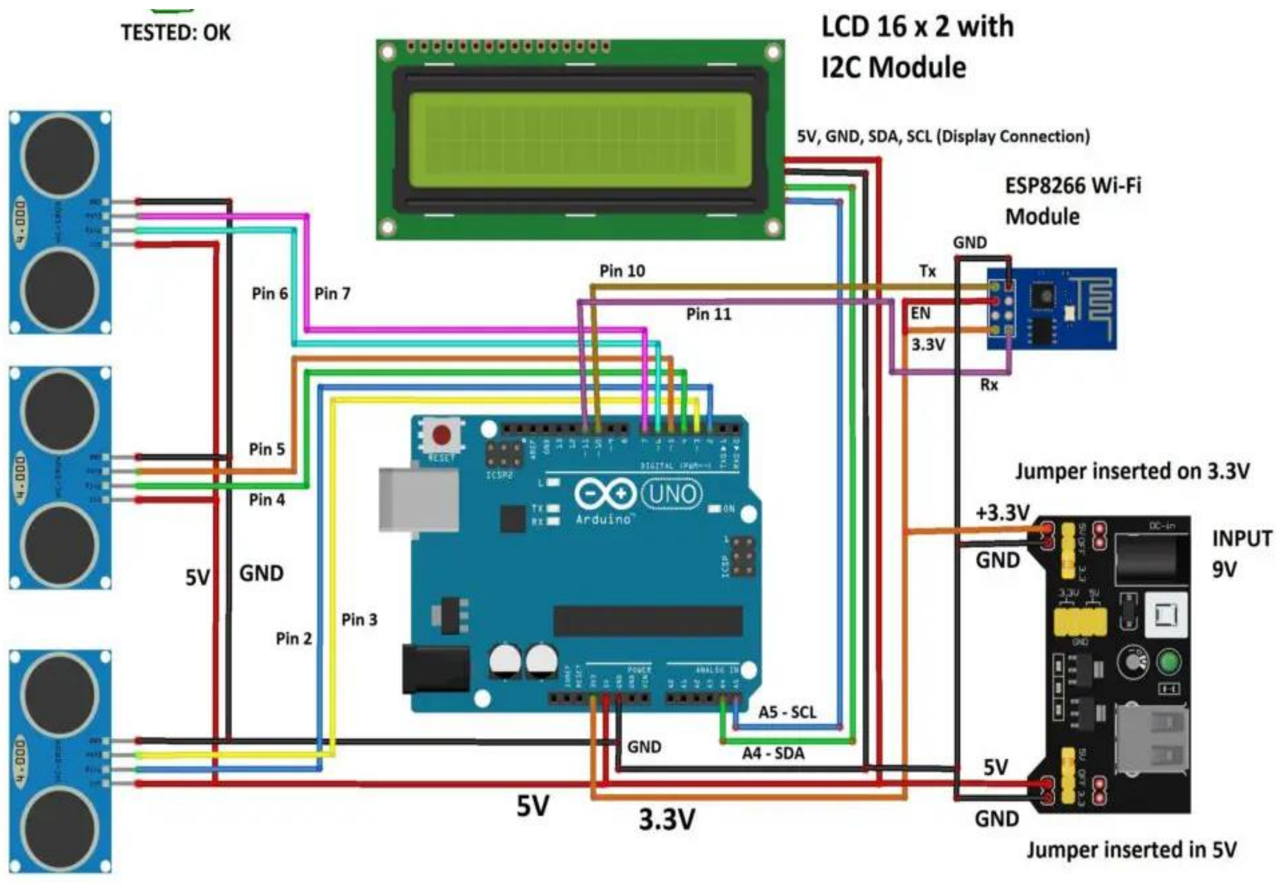


Technology stack :

- Hardware: Use IoT sensors and microcontrollers to detect vehicle presence in parking spaces.
- Connectivity: Connect sensors to the campus network using Wi-Fi.
- Mobile Application (Android): Build an Android app with real-time updates, user authentication, map integration.
- Data Visualization: Create dashboards and charts for administrators to monitor parking occupancy.
- Security: Implement encryption, secure APIs, and user authentication for data protection.
- Testing and Quality Assurance: Thoroughly test the app for accuracy and user experience.

- Deployment: Deploy the backend on a cloud platform and publish the Android app on Google Play Store.
- Maintenance and Updates: Continuously update and maintain the application.
- Documentation: Document the entire system, hardware setup, software architecture.

CIRCUIT DIAGRAM



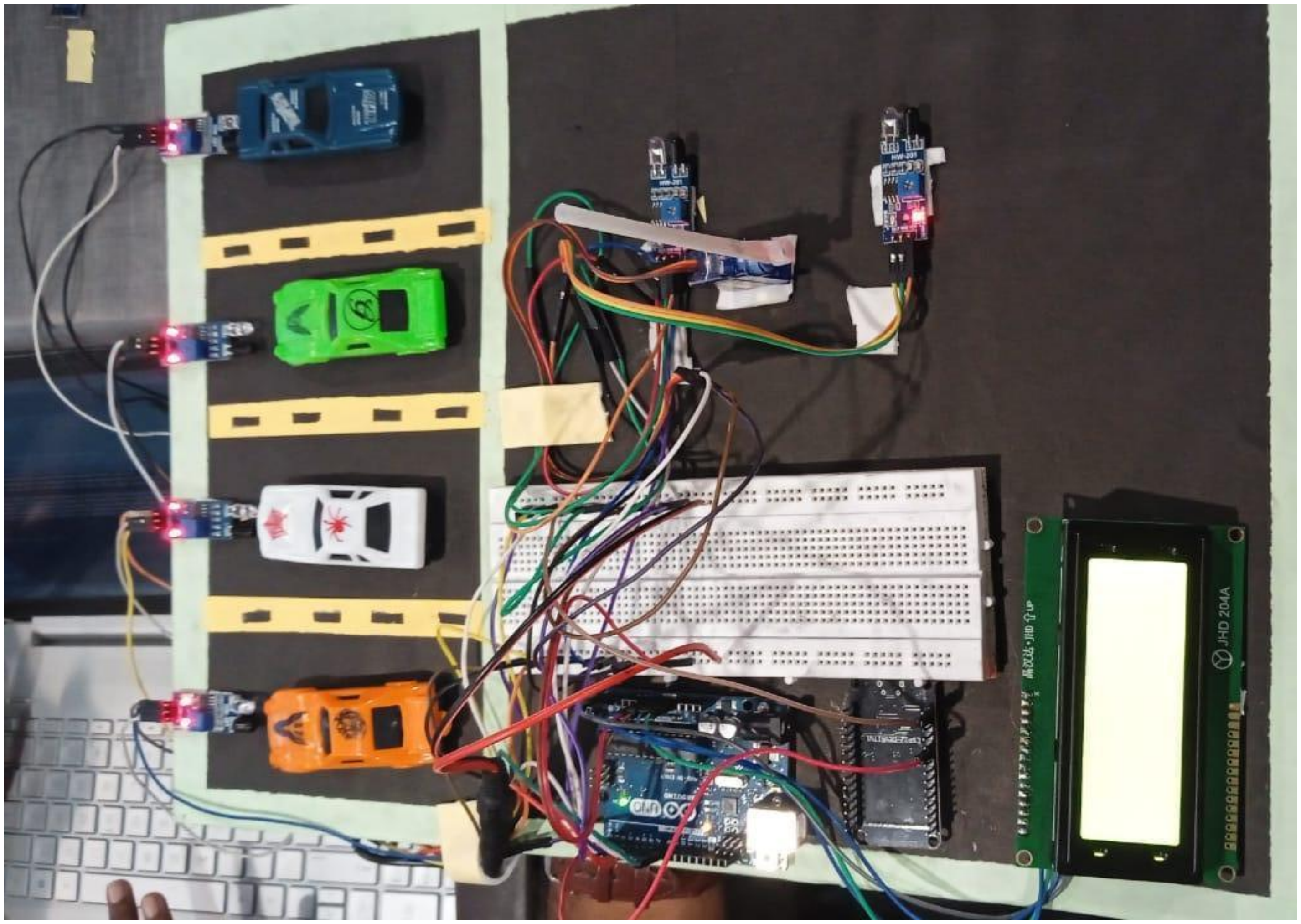
PROGRAMMING FOR ARDUINO

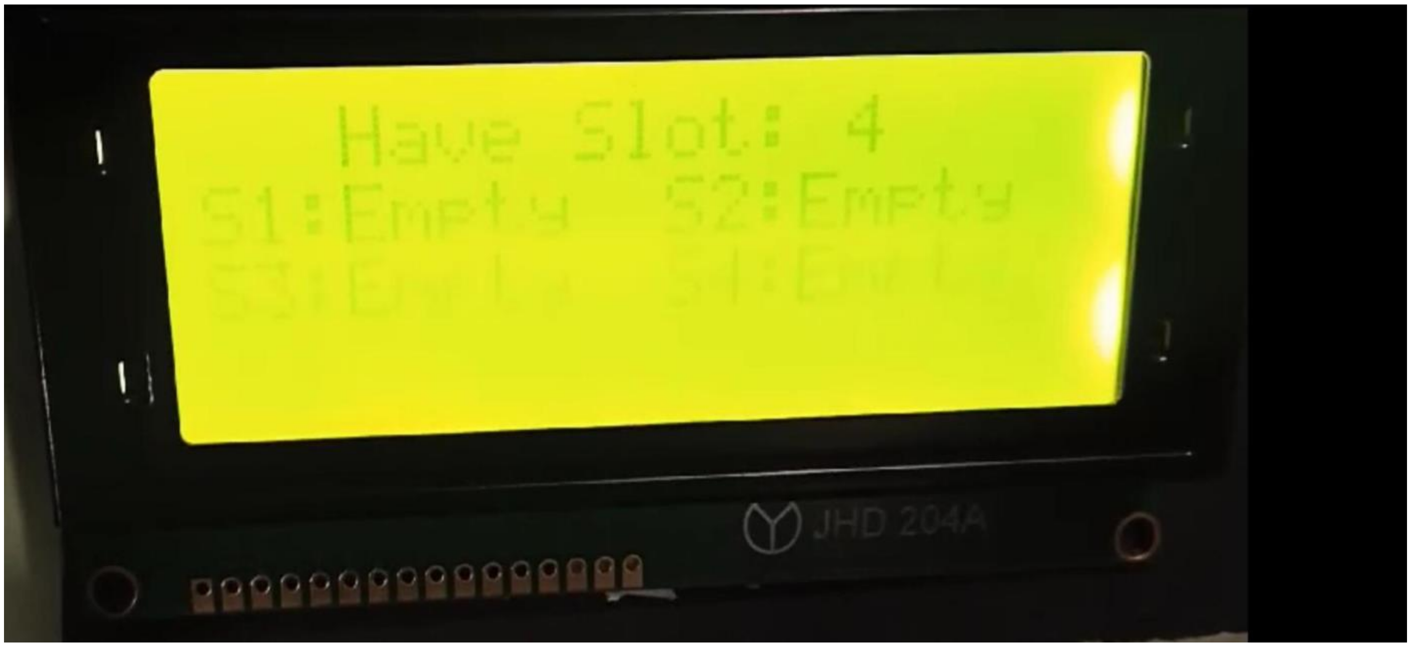
```
#include <Servo.h> //includes the servo library
#include <Wire.h> 
#include <LiquidCrystal_I2C.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

// Your Wi-Fi credentials
Const char* ssid = "Smartpark";
Const char* password = "psv";
Const char* serverAddress = "192.168.0.100";

Void sendDataToFlutterApp(int slot, int isOccupied) {
  WiFiClient client;
  If (client.connect(serverAddress, 80)) {
    String data = "{\"slot\": " + String(slot) + ", \"occupied\": " + String(isOccupied) + "}";
    Client.println("POST /updateParkingSlot HTTP/1.1");
    Client.println("Host: " + String(serverAddress));
    Client.println("Content-Type: application/json");
    Client.print("Content-Length: ");
    Client.println(data.length());
    Client.println();
    Client.println(data);
    Client.stop();
  }
}

LiquidCrystal_I2C lcd(0x27,20,4);

Servo myservo;

#define ir_enter 2
#define ir_back  4

#define ir_car1 5
#define ir_car2 6
#define ir_car3 7
#define ir_car4 8
#define ir_car5 9
#define ir_car6 10

int S1=0, S2=0, S3=0, S4=0, S5=0, S6=0;
int flag1=0, flag2=0; 
int slot = 6;

void setup(){
Serial.begin(9600);

pinMode(ir_car1, INPUT);
pinMode(ir_car2, INPUT);
pinMode(ir_car3, INPUT);
pinMode(ir_car4, INPUT);
pinMode(ir_car5, INPUT);
pinMode(ir_car6, INPUT);

pinMode(ir_enter, INPUT);
pinMode(ir_back, INPUT);

myservo.attach(3);
myservo.write(90);

lcd.begin(20, 4);
lcd.setCursor (0,1);
lcd.print("    Car  parking  ");
lcd.setCursor (0,2);
lcd.print("       System     ");
delay (2000);
lcd.clear();

Read_Sensor();

int total = S1+S2+S3+S4+S5+S6;
slot = slot-total;
}

void loop(){

Read_Sensor();

lcd.setCursor (0,0);
lcd.print("   Have Slot: ");
lcd.print(slot);
lcd.print("    ");

lcd.setCursor (0,1);
if(S1==1){lcd.print("S1:Fill ");}
```

```
    else{lcd.print("S1:Empty");}

lcd.setCursor (10,1);
if(S2==1){lcd.print("S2:Fill ");}
    else{lcd.print("S2:Empty");}

lcd.setCursor (0,2);
if(S3==1){lcd.print("S3:Fill ");}
    else{lcd.print("S3:Empty");}

lcd.setCursor (10,2);
if(S4==1){lcd.print("S4:Fill ");}
    else{lcd.print("S4:Empty");}

 lcd.setCursor (0,3);
if(S5==1){lcd.print("S5:Fill ");}
    else{lcd.print("S5:Empty");}

lcd.setCursor (10,3);
if(S6==1){lcd.print("S6:Fill ");}
    else{lcd.print("S6:Empty");}

if(digitalRead (ir_enter) == 0 && flag1==0){
if(slot>0){flag1=1;
if(flag2==0){myservo.write(180); slot = slot-1;}
}else{
lcd.setCursor (0,0);
lcd.print(" Sorry Parking Full ");
delay(1500);
}
}

if(digitalRead (ir_back) == 0 && flag2==0){flag2=1;
if(flag1==0){myservo.write(180); slot = slot+1;}
}

if(flag1==1 && flag2==1){
delay (1000);
myservo.write(90);
flag1=0, flag2=0;
}

delay(1);
```

```
}

void Read_Sensor(){
S1=0, S2=0, S3=0, S4=0, S5=0, S6=0;

if(digitalRead(ir_car1) == 0){S1=1;}
if(digitalRead(ir_car2) == 0){S2=1;}
if(digitalRead(ir_car3) == 0){S3=1;}
if(digitalRead(ir_car4) == 0){S4=1;}
if(digitalRead(ir_car5) == 0){S5=1;}
if(digitalRead(ir_car6) == 0){S6=1;}
}
```

Working model and output





PROGRAMMING FOR APP

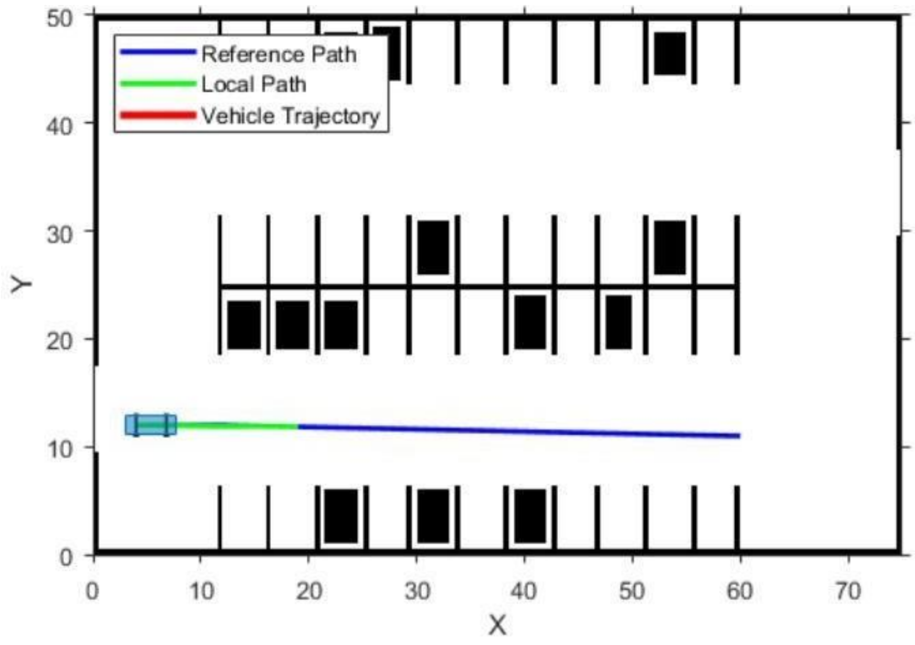
```
Import colorama

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
```

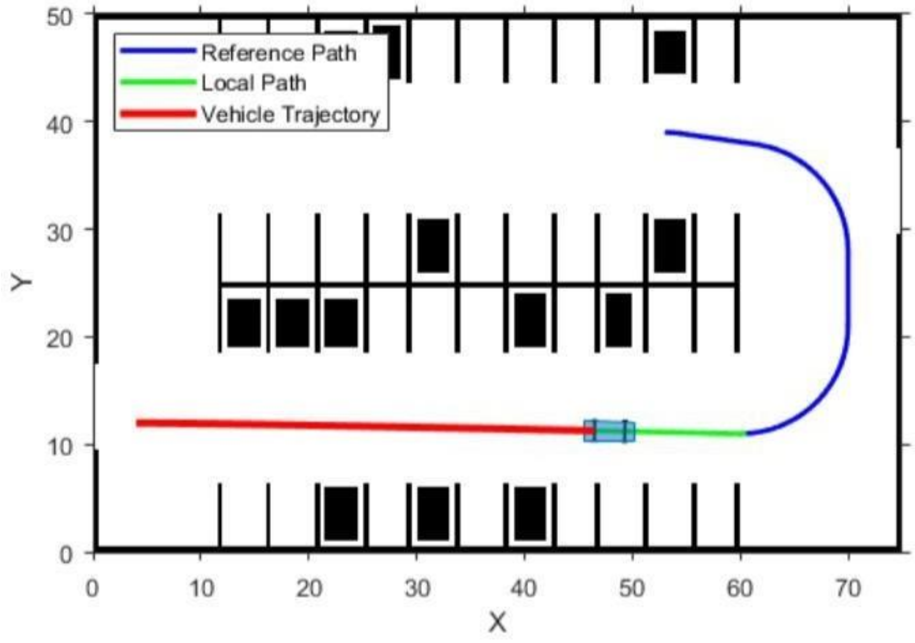
```
def _init_(self, rows, columns):

Self.rows = rows

     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”))
```

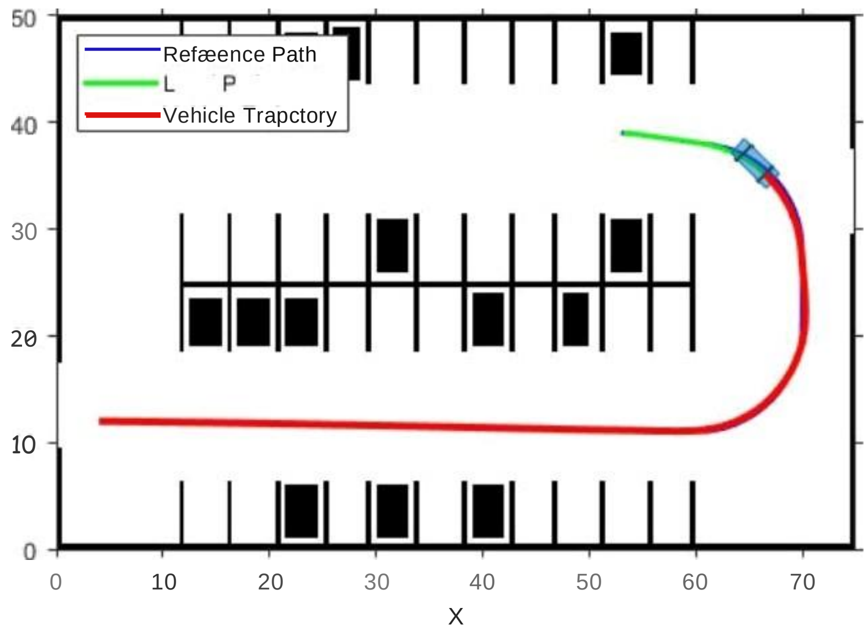
```
        Print(colored("Thanks for using our parking assistance system", "green"))
    Def _park_vehicle(self):
        Vehicle_type = self._get_safe_int("Available vehicle types: 1. Car\t2. Bike\t3. Truck.\nEnter
your choice: ")


        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()
```

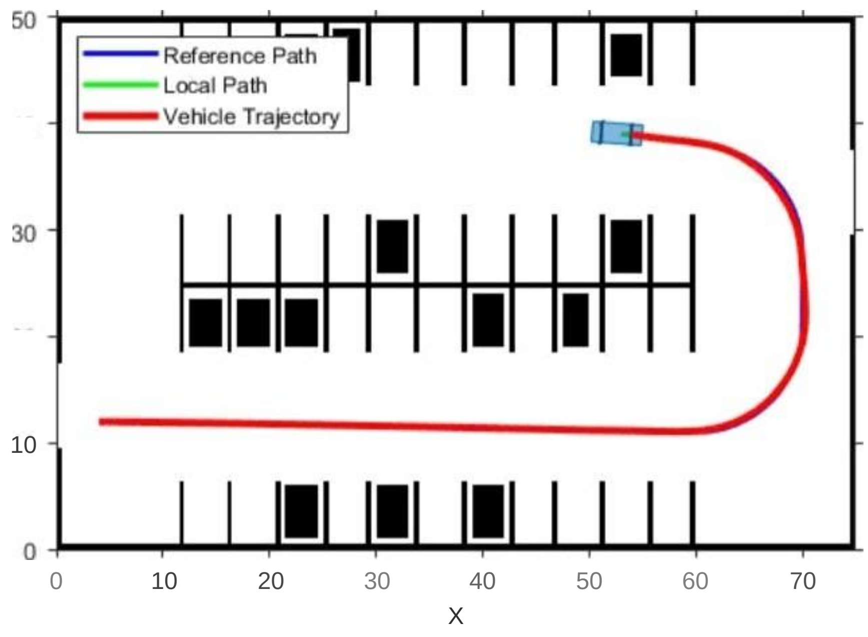
OUTPUT:


Reference Path
Local Path
Vehicle Trajectory
X
Y


Reference Path
Local Path
Vehicle Trajectory
X
Y



Refæence Path
L P
Vehicle Trapctory
50
40
30
20
10
0
0
10
20
30
40
50
60
70
X



Reference Path
Local Path
Vehicle Trajectory
50
30
10
0
0
10
20
30
40
50
60
70
X

Benefits

- Improved User Experience: Users can check parking availability in real-time, reducing frustration and time wasted searching for parking spots.
- Time and Fuel Savings: Users can save time and fuel by quickly finding an available parking space, contributing to environmental sustainability.
- Reduced Traffic Congestion: Efficient parking leads to reduced traffic congestion and a smoother flow of vehicles on campus, making it safer for pedestrians and cyclists.
- Optimized Space Utilization: The application can help campus management optimize parking space usage by identifying underutilized areas, potentially reducing the need for expansion.
- Data Insights: The IoT sensors can collect data on parking patterns, which can be analyzed to make informed decisions about parking space allocation

Conclusion

The IoT Smart Parking System, with its intuitive mobile app, doesn’t just park your car; it parks us firmly in a smarter, more efficient, and eco-friendly urban future. It’s more than innovation; it’s a glimpse of the urban dream realized.