

SMART PARKING

PROJECT OBJECTIVES:

The basic objective of a smart parking solution is to identify a vehicle's presence or absence in a particular parking space with a high degree of accuracy, and to pass on this data into a system for visualization and analysis – to be available for parking asset managers and/or enforcement officers.

Keeping in mind the objectives mentioned above, the next step is to take into consideration following important features:

- Accuracy of detecting a vehicle presence/absence
- Total cost of solution
- Privacy concerns

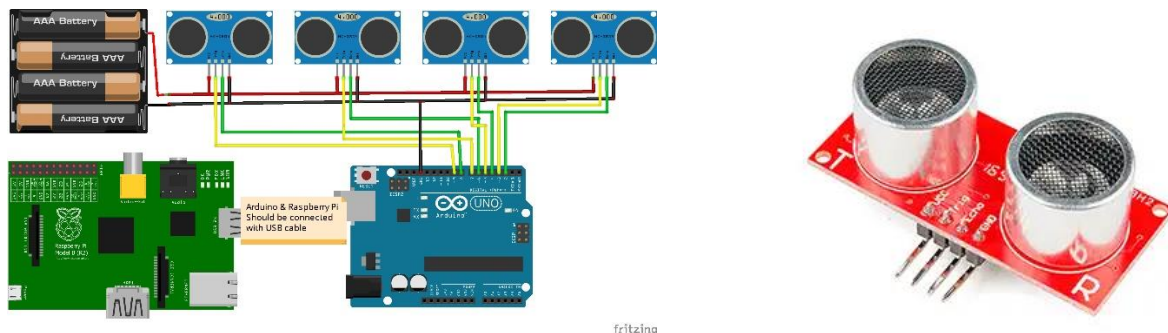
The laser scanner sensors are well known for their accuracy of detecting a vehicle presence – therefore the sensor located at the entrance and exit of a parking area will count with high accuracy the entrance and exit of the vehicles, taking into consideration even two cars stopping very close to each other.

The total cost of solution considering the initial purchasing and installation cost, you will have low maintenance cost and no need of replacing batteries.

The sensors will just count the vehicles without recording any data. The system is not based on cameras, but again on sensors and eventually a display for counting the cars. Therefore, the privacy is totally granted.

Sensor solution is a reliable, and cost-effective modern smart parking solution.

IOT SENSOR SETUP:

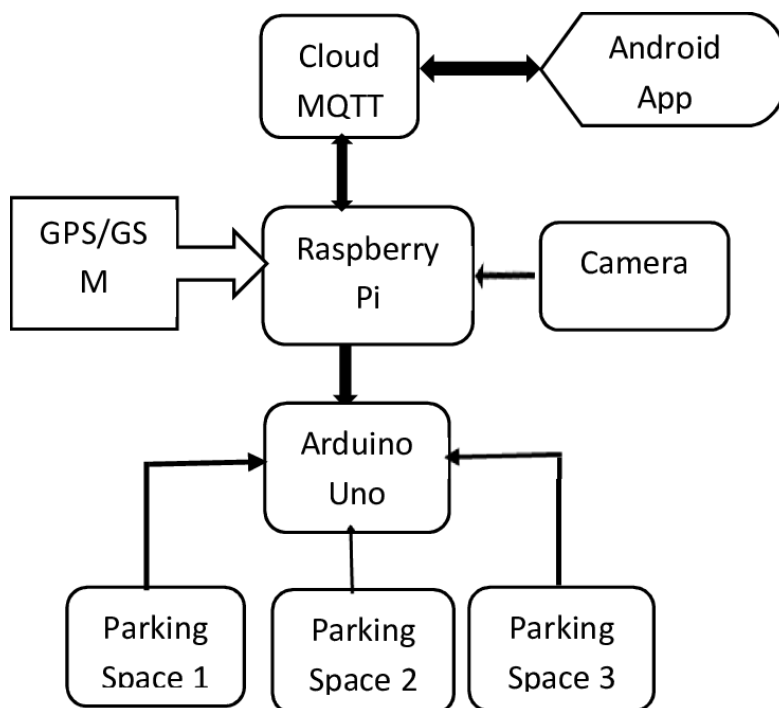


Ultrasonic ranging sensors :

The sensor uses the same time-of-flight method as IR but the emitter sends ultrasound waves in (40-60KHz range) instead of photons. The most widespread use of the sensor is, of course, parking assistance systems in modern cars: the sensors are affixed in rear and front bumpers and help you park by signaling proximity to obstructions in the short range.

Wider emitter beam than IR sensors which makes the measurement more integral and less prone to random reflections errors

INTEGRATION:



Raspberry Pi 4 Model B:

Raspberry Pi 4 Model B has been chosen for this project mainly due to its powerful processing capabilities as a single-board embedded computer and its ability to connect to the Internet either via Ethernet port or wirelessly using Wi-Fi or Bluetooth. Ease to connect to the Internet is one of the RPi advantages over Arduino, and it also supports a huge variety of programming languages such as Python, Java, C, C++, Perl, Ruby, BASIC; whereas Arduino, however, only accepts either Arduino or C/C++. Raspberry Pi 4 Model B is a direct upgrade over the Raspberry Pi 3 Model B+. It has Broadcom BCM2711 1.5 GHz quad-core 64-bit CPU with a Cortex-A72 processor, which is more efficient and much more powerful than Pi 3's 1.4 GHz processor. The GPU of Pi 4 is capable of running comfortably compare to Pi 3 due to Pi 4 improved clock speed: 500 MHz compared to Pi 3's 400 MHz. The RPi 4 Model B has a better CPU and GPU compared to Model 3 B, which is essential when we have decided on a visual-based detection method using the Pi camera. The Pi camera requires significant processing speed to be able to capture and process the images. Therefore, Raspberry Pi 4 is overall more suitable for our system and also able to give better

performance in terms of video capturing and streaming. 3.1.2. Pi Camera The pi camera is selected for the proposed system

Algorithm:

Sensor and Raspberry Pi Module:

- Step 1: Initialize the System (IR Sensors and raspberry Pi)
- Step 2: Read sensor status and store in the Database
- Step 3: Send Sensor Status to Android Application when requested
- Step 4: Continue through Step 3
- Step 5: Stop

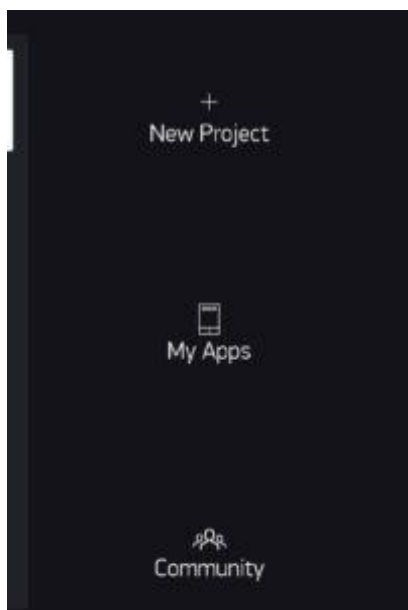
Android Application Module:

- Step 1: Start Android App
- Step 2: Check connectivity with Server (Raspberry Pi)
- Step 3: Request Sensor Status after every time interval
- Step 4: Continue through Step 3
- Step 5: Stop

MOBILE APPLICATION:

Step 1

open the Blynk app & click on the new project.



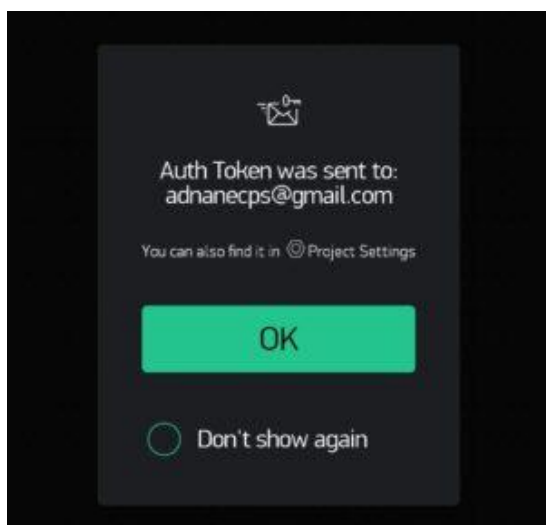
Step 2

Select nodeMCU from the library



Step 3

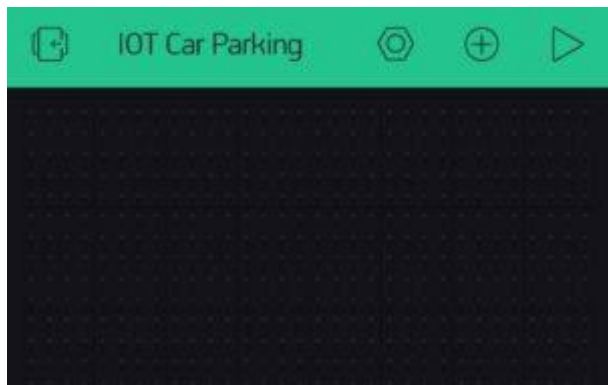
Token will be sent to your Gmail id. copy that token from the Gmail id and paste it into the code which is given above.



Step 4

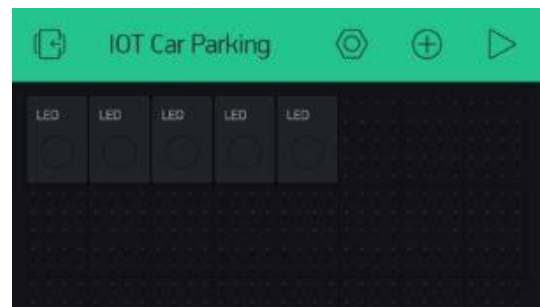
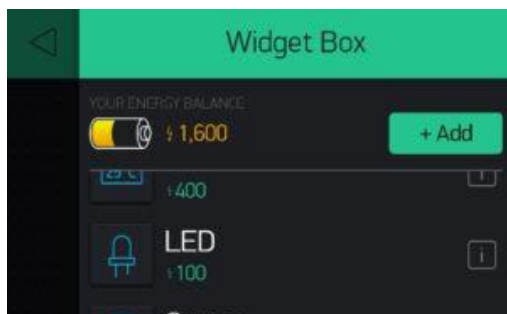
add new button

Click on the plus sign from the given screen in-app.



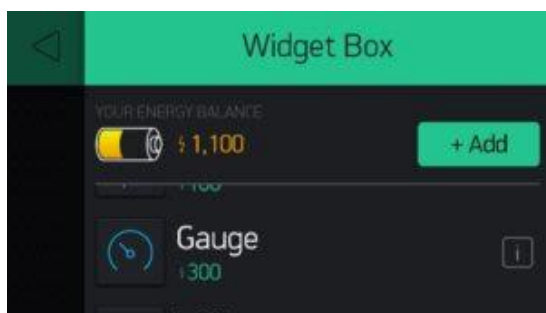
Step 5

add 5 led



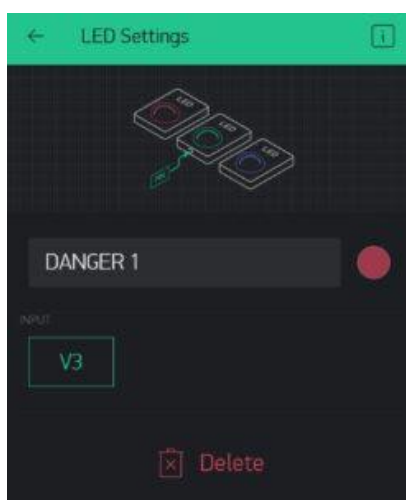
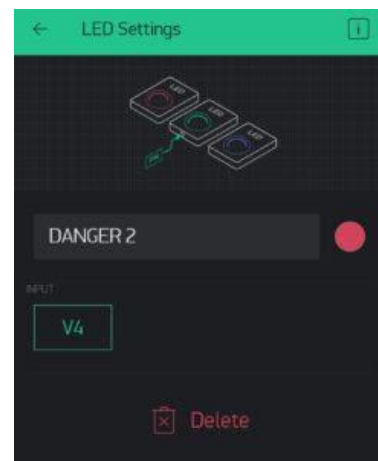
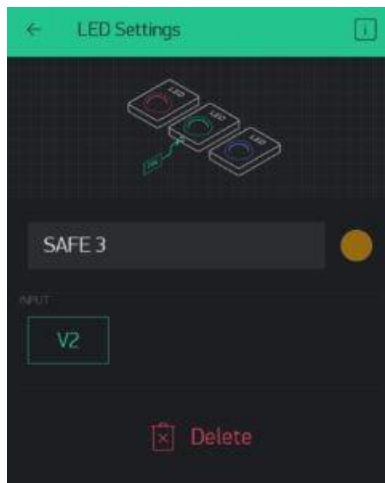
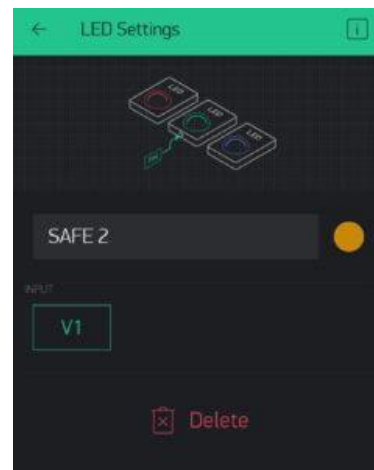
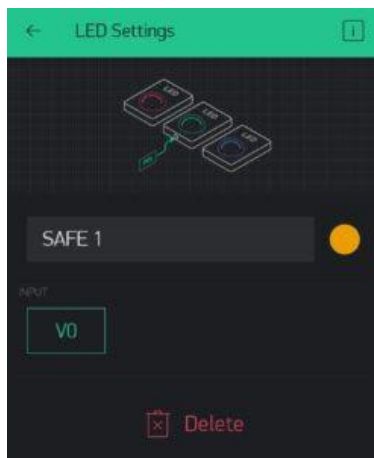
Step 6

add 2 gauge



Step 7

Set led with the name of safe 1 to safe 5 and decide color



Step 8

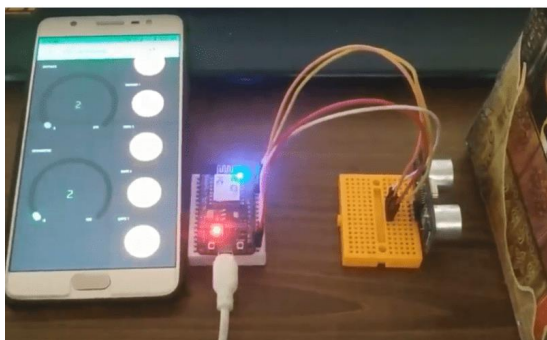
Set gauge from 0 to 100, 255 value.



Now you can see this working in the phone screen.



OUTPUT:



Python 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 = 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()

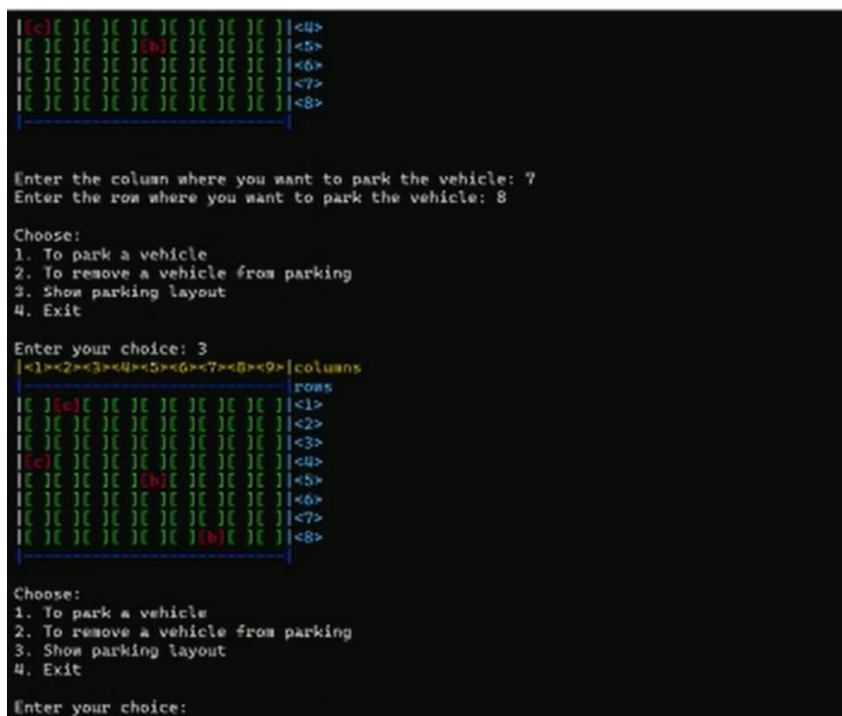
except ValueError:
    print("Rows and columns should be integers only.")

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:



```

[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <4>
[[c]] [[c]] [[c]] [[b]] [[c]] [[c]] [[c]] [[c]] <5>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <6>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <7>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <8>
|-----|

Enter the column where you want to park the vehicle: 7
Enter the row where you want to park the vehicle: 8

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

Enter your choice: 3
<1><2><3><4><5><6><7><8><9>|columns
|-----|rows
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <1>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <2>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <3>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <4>
[[c]] [[c]] [[c]] [[b]] [[c]] [[c]] [[c]] [[c]] <5>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <6>
[[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] [[c]] <7>
[[c]] [[c]] [[c]] [[c]] [[c]] [[b]] [[c]] [[c]] <8>
|-----|

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

Enter your choice:

```

REAL-TIME APPROACHES:

1. Locate Free Spaces

Sometimes drivers may experience difficulties with finding a free spot in an open space. That's when parking app development solutions come in handy. They mark the available spots so that drivers can park their vehicles quickly and effortlessly. Additionally, such apps can offer them turn-by-turn directions to the closest car-park lots and inform them about fees, availability, rules, fines, and more. Moreover, if an app has a parking reservation feature, drivers can pay and book a space in advance.

UCLA parking specialist Donald Shoup has found out that from 15% to 74% of traffic problems result from inability or time-consuming car-park search. Parking app features that allow finding nearby parking lots or reserving them in advance help reduce traffic jams.

2. Multi-Space Parking Meters

Well, dealing with mazes to find a spot that's already taken is challenging. Meanwhile, discovering some fellow settled in two spaces is a whole new level of rage. With a car-park solution at hand, staff can see space allocated for a specific car. In case of double parking, they can notify a driver, resulting in better management.

3. Confusing Rules

Each parking lot operates by its own rules: short-term booking (up to four hours) and long-term booking (more than four hours). Drivers wanting to know how parking lots charge their customers will definitely benefit from using car-park applications.

4. Decrease Air Contamination

Parking app development solutions save drivers from spending too much time looking for a vacant parking spot. Moreover, now they do not have to deviate from straight road direction and cause air pollution.

space, so such platforms can help cities lower the ecological footprint.

Now that we've uncovered the core issues a car parking mobile app development solves, it's time for revealing principles of how a parking service works.

CONCLUSION:

IoT has many different applications, but one of the most exciting is its use in smart parking. IoT-based parking systems are able to better track the availability of parking spots on a given lot, making it easier to find an available parking spot.

It is important to note that not all IoT-based parking systems are the same. For example, some use QR codes to identify available parking spots, while others use sensors to detect when a car leaves a parking spot. The benefits of an IoT-based smart parking system are that it is more creative, efficient, and convenient for both drivers and owners of the parking lot.