Smart Parking Guidance System

Ganesh J Bannur , Ashutosh M Bharadwaj , Allen Joel Lobo, Ishan Kaushik, Dr. K Badari Nath
CSE Department, RV College of Engineering, Bangalore-560059, India
Faculty CSE Department, RV College of Engineering, Bangalore-560059, India

***Abstract*:- With the increasing number of vehicles, parking in community spaces or shared areas such as malls and offices has become an arduous task, involving a lot of time spent searching for a parking spot manually. Ideally, each newly arrived vehicle would go to the nearest available parking lot in the shortest possible time.**

Currently, when a new vehicle arrives at a shared parking lot, there are two possible scenarios: either there are teams of parking attendants who direct them to the available parking lot or the driver of the vehicle has to slowly traverse through the entire parking
space to find an open parking spot.
The first scenario involves a large number of personnel who are required to manage the
parking space, with the number directly in proportion to the parking area, leading to a point where the number of people needed just for directing traffic is very large.
In the second scenario, the driver of the vehicle loses time in search of an open parking space which may or may not even be available.
In this paper we propose a parking guidance system consisting of ultrasonic sensors in each parking space, centrally connected to a cloud server. Drivers can navigate to the nearest free parking spot in the shortest possible time, following the path outlined in the website. The proposed system has a large number of use cases from parking in large malls, arenas and public places to visitor parking in apartments and residential areas.

Keywords: Parking lots,ultrasonic sensors, cloud server

1. INTRODUCTION

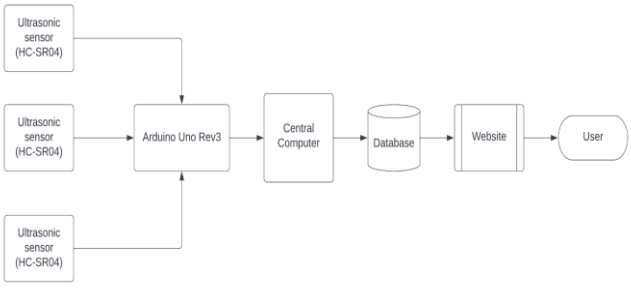
Smart parking guidance systems allow drivers to attain dynamic information on parking in restricted areas such as parking garages. The system uses data from the parking slots as to how many cars have entered the parking lot, how many slots are available and the distance of each slot from the car which is continuously monitored. Once a car enters a new slot, that slot is now closed for new cars which have entered. This helps in assisting drivers in finding parking easily and avoiding time delays and car's blocking each other. The objective is to reduce search time, which in turn reduces congestion in the parking areas.
We develop and implement a prototype of a smart parking guidance system that guides incoming vehicles to the nearest available parking lot through a website. It is required to install distance sensors in each parking lot which will then be connected to a central database from which we can identify which parking lots are occupied and which are free.

2. RELATED WORK

Many attempts at creating smart parking systems exist for various specific applications. For example Yan et al. [1] propose a smart parking system for city-wide roadside parking lots. They focus on the security and privacy of user information collected for reservations and also on the possibility of advertising to users through the same secure network. Geng et al. [2] propose a Smart Parking System for use in an urban environment which assigns and reserves a parking space based on proximity to current location and cost. Their main focus is on efficient utilization of parking space while satisfying user requirements of proximity and cost. Faheem et al. [3] surveyed the existing methods for building intelligent parking systems. They focus on integration of the system into Intelligent Transportation Systems which also involve vehicle guidance, parking reservation, integrated payment and other in-car technologies. The surveyed methods focus on parking systems for managing city-wide public parking spaces. Idris et al. [4] conducted a survey on the different categories of parking systems and also the techniques available for implementing vehicle detection. Parmar et al. [5] study the characteristics which are required in a good parking system and also give techniques to evaluate the effectiveness of a parking system.

While most papers explore parking systems for urban, city-wide parking management, we propose a smart parking system for smaller yet critical and underserved spaces. Our parking system is suitable to be deployed in public parking areas within places such as shopping malls, stadiums and business parks. Even though these are small parking spaces they have the ability to cause traffic congestions of large areas of a city due to high parking demand in very short spans of time (eg. a stadium before a much anticipated sports match). Our system is designed to efficiently locate parking spaces and guide vehicles to them, thus maintaining the flow of vehicles into and out of the space.

3. SYSTEM ARCHITECTURE



A. Methodology

A vehicle enters the parking lot and scans the QR code at the entrance to open the website.

1. The website automatically updates itself with the latest information in the database. So it will show the path to the closest free slot in.
2. The driver follows the directions to the slot and parks there. If another car takes the parking slot before the driver reaches it then the website will automatically update to show a new path.
3. Once the driver parks in the spot, the ultrasonic sensor will detect the vehicle and the Arduino will update the database within five seconds. The path will automatically update for all other drivers in the parking lot.
4. When a vehicle leaves the slot, the ultrasonic sensor will detect the free slot and the Arduino will again update the database within five seconds.

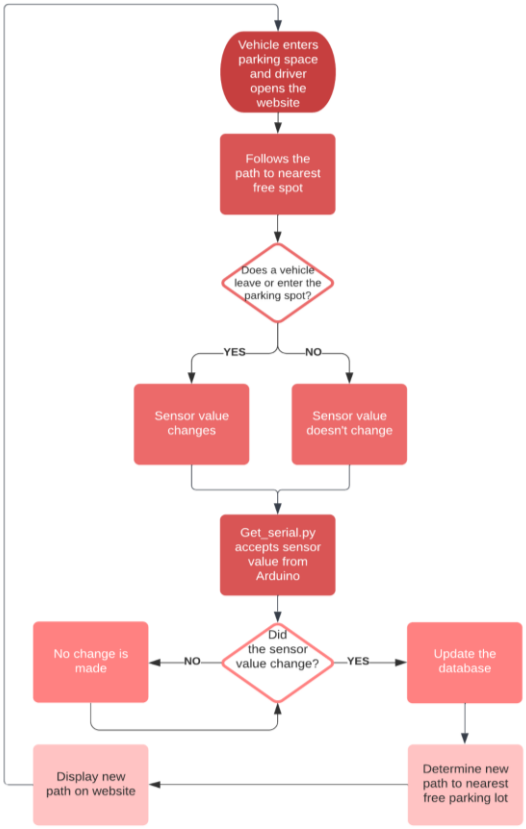


Fig 2. Flowchart

B. Hardware and Software Tools Used

A small-scale prototype of the proposed smart parking guidance system was developed, using an Arduino Uno Rev 3. An ultrasonic sensor (HC-SR04) is used to check the status (empty/full) of the parking slot. We detect that the slot has been occupied by a car when the distance measured by the sensor is less than the distance to the floor. The sensor is attached to any point off the ground such as a pillar or the ceiling of the parking lot. This can be done because the sensor has a range of upto 4 meters and will be able to detect a car on the ground.

The arduino is used to receive distance data from the ultrasonic sensor and detect if the slot is full and send the information to a computer which uploads it to the cloud database. The scripts on the computer to receive data from the Arduino and upload it to the database are in Python.

The cloud database and website are hosted on Firebase.

4. DESIGN AND IMPLEMENTATION

A. On-Site Hardware and Software Working:

At each parking spot we have an ultrasonic sensor (HC-SR04) that is used for detecting whether a parking lot is occupied or not. When the distance measured by it is less than a certain value it means that there is a car in the slot. The sensor is connected to an Arduino Uno.

The ultrasonic sensor is mounted at the top, bottom or back of each parking spot depending on which is most convenient. Top mounting should be the most convenient as it does not get in the way of cars or pedestrians.

Fig 3. Schematic view of the sensors

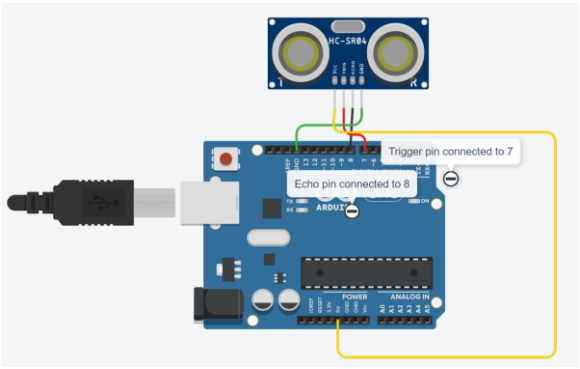


Fig 4:Circuit diagram of sensor and Arduino Uno made using TinkerCad

The Arduino is connected to a computer which reads the Arduino data and updates the cloud database every five seconds.The Arduinos can be powered by connecting them to a common 9V, 2A supply.

The device sends measurements to the cloud where they are stored in Firebase as a sensor state.

A sensor detects a parked car by measuring the distance to the nearest obstacle. The state can be “occupied” if the distance is less than X, “free” if the distance is more than X. If the sensor detects an obstacle- in this case- a vehicle parked in the lot, it returns a value of 1. If there is no obstacle/vehicle, it returns a value of 0.

This value is then uploaded to our Firebase server to be stored as a sensor state in a JSON file. The obstacle detection

by the sensor is done continuously, while the sensor state is uploaded every 5 seconds.

```
#define TRIGGER_PIN 7
#define ECHO_PIN    8

int echoTime;
float distance;
int car;

void setup()
{
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  digitalWrite(TRIGGER_PIN, LOW);

  Serial.begin(9600);
}

void loop()
{
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  echoTime = pulseIn(ECHO_PIN, HIGH);
  distance = (echoTime*3.4/100)/2;
  if (distance<10)
  {
    car = 1;
  }

 else
 {
  car = 0;
 }

  Serial.println(car);

  delay(100);
}
```

Fig 5: Arduino Car Detection code

In the Arduino we use pin 7 as the trigger pin and pin 8 as the echo pin for the ultrasonic sensor.
We then open the serial port through which the Arduino communicates with the computer. We set the data rate for the port to the default of 9600 baud.
The trigger pin is used to trigger the ultrasonic sensor to send an ultrasonic pulse and measure the time it takes to come back after reflection. The trigger pin must be held HIGH for 10 microseconds for the sensor to get triggered.
The echo pin is the pin at which the output of the ultrasonic sensor is received. The output of the sensor is the time taken for the ultrasonic signal to reflect off an obstacle and come back to the sensor in seconds.
We use this time to calculate the distance of the obstacle (taking speed of sound as 340m/s)
For our smaller model, if the distance is less than 10cm we detect a car at the spot. For a parking lot in real life we can adjust this threshold distance.

We then print the data onto the serial port which will be read by a script running on the computer.

B. Connecting to cloud based server

```
import serial, sys, time
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db

# Fetch the service account key JSON file contents
cred = credentials.Certificate('/Users/snoopy/Desktop/College/Third Sem/FCSD/EL/smart-parking-system-caf85-firebase-adminsdk-tw3so-627cee23c1.json')

# Initialize the app with a service account, granting admin privileges
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://smart-parking-system-caf85-default-rtdb.asia-southeast1.firebasedatabase.app/'
})

ref = db.reference('/')

with open("which.txt", "r") as f:
    slot_num = f.readline()

start = 0
end = 0

with serial.Serial(port=sys.argv[1], baudrate=sys.argv[2]) as ser:
    while ser.isOpen():
        end = time.time()
        current_data = ser.readline()
        for a_byte in current_data:
            a_byte = a_byte-48

            if(end-start>=5):
                print(a_byte)
                current_ref = ref.child('%s/full' % slot_num)
                current_ref.set(a_byte)
                start=end

            break
```

The python script given above runs on the computer connected to the Arduino. It reads the serial port and gets the Arduino data indicating the presence or absence of a car at the spot.
Which spot the Arduino represents is given in a configuration file.
After connecting to the database, we open the configuration file which defines the spot number that the Arduino board and sensor are located in.
As long as the serial port is open, the script reads the data on it and updates the cloud database every five seconds.
Even though the serial port data updates much faster, we only update the database every five seconds to reduce the amount of data uploaded to the database hence reducing cost. Firebase charges customers for the amount of data uploaded and downloaded from their database.

C. Backend Cloud Server

Firebase is a backend as a service (BaaS) platform provided by Google for creating mobile and web applications. Firebase allows us to quickly build performant and reliable web applications without having to build a scalable and reliable backend by ourselves. It provides all the services we need to develop fast, secure, and extensive web applications

with minimal cost by abstracting the backend as API accessible black boxes.

In Firebase, we first create our own database with the parking slots and paths that the cars can take in the parking space. We link the firebase by granting permission to the python script which imports the necessary data from firebase such as admin details and credentials required to host it on the Firebase Console.


```
    "41": {
        "full": 0,
        "dir": "SSR5"},
    "42": {
        "full": 0,
        "dir": "SSR4"},
    "43": {
        "full": 0,
        "dir": "SSR3"},
    "44": {
        "full": 0,
        "dir": "SSR2"},
    "45": {
        "full": 0,
        "dir": "SSR1"},
    "46": {
        "full": 0,
        "dir": "SSL1"},
    "47": {
        "full": 0,
        "dir": "SSL2"},
    "48": {
        "full": 0,
        "dir": "SSL3"},
    "49": {
        "full": 0,
        "dir": "SSL4"},
    "50": {
        "full": 0,
        "dir": "SSL5"}
```


Fig 6. Sample Database entries

The database is stored in the JSON format. Each slot has 2 fields: the "full" field shows if it is empty or occupied and the "dir" field gives the directions from the entrance to the slot in the form of a string.
For demonstration purposes we can randomize the filled spots in the parking lot in the python script given below and update the database hosted on Firebase.

```
import random
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db

# Fetch the service account key JSON file contents
cred =
credentials.Certificate('/Users/snoopy/Desktop/College/Third Sem/FCSD/EL/smart-parking-system-caf85-firebase-adminsdk-tw3so-627cee23c1.json')

# Initialize the app with a service account, granting admin privileges
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://smart-parking-system-caf85-default-rtdb.asia-southeast1.firebasedatabase.app/'
})

ref = db.reference()

print("Enter number of parking spots that are to be filled out of 50: ")
full=int(input())
data = {}
for i in range(1, 51):
    data[i] = {"full": 0}
for i in range(full):
    data[random.randint(1, 50)]["full"] = 1

for slot_num in data:
    current_ref = ref.child('%d/full' % slot_num)
    current_ref.set(data[slot_num]["full"])
```

The Firebase Console is also realtime, which means everytime we block the sensor which is connected to the Arduino, it sends a signal of certain frequency and baud rate and then updates it in the database connected and hosted on Firebase.

Also, another important feature of Firebase is that it is a cloud hosted platform and is not local to any device.
This ensures that any user who comes into the parking lot can access the website hosted freely and updates keep happening depending on the number of cars in the parking lot.

D. Website

In our website we have a rectangular arrangement of boxes, each box representing a parking spot. When a vehicle enters the parking space there will be a QR code to scan which will take them to the website. In the website, free spots are shown as green boxes, full ones as red boxes and the nearest empty spot is given in black. The path that the vehicle should take to reach the nearest spot is shown along with the parking spot number.
The nearest available spot and the path to it from the entrance are retrieved from the database. The nearest spot and path are dynamically updated every five seconds by syncing with the database.

5. RESULTS AND ANALYSIS

Each ultrasonic sensor returns a value of 0/1 to the server based on whether an obstacle (in this case, a vehicle) is less than 10 cm from the sensor. This value is uploaded to the corresponding slot number in the cloud database, based on the configuration file. In the snapshot given below, the Arduino is reading the output of the ultrasonic sensor, and printing it onto the serial port. The python script reads the value from the serial port and updates the database every 5 seconds. The sequence of 0s and 1s represent the current status (full/empty) of the slot.

Fig 7. Terminal snapshot of the sensor outputs

On our website, the path denoting the way to the nearest free spot keeps updating based on the values received from the sensors. The sensors update the database periodically and the website automatically refreshes the path when a value in the database changes.

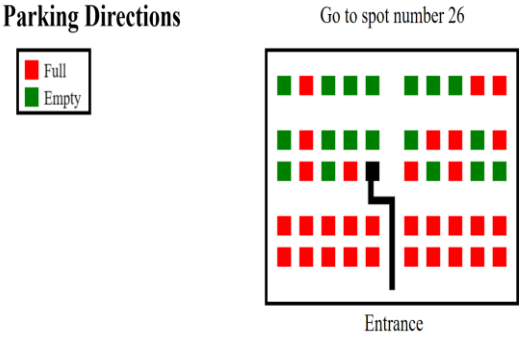
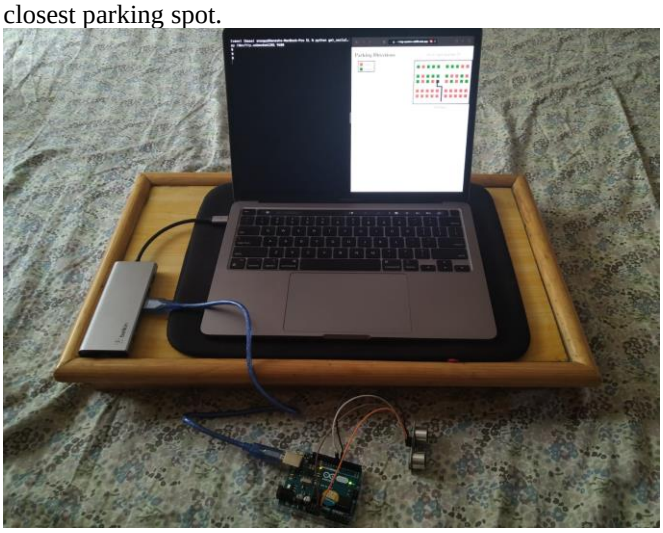


Fig 8. Screenshot of the website

In our website, the vehicle enters whose path has been marked as black and then enters the spot which is free. We have a total of 50 spots shown. As spot 26 is free, hence it goes there. The numbering is done from bottom right and continues in the same line, which is on the other side of the path and then returns back on the opposite side to continue the counting. Hence we get number 26 to the spot. The red spots indicate that the parking is full, and the green spots indicate that the parking is empty. Hence using the shortest distance block in our website code we can send the car to the closest parking spot.



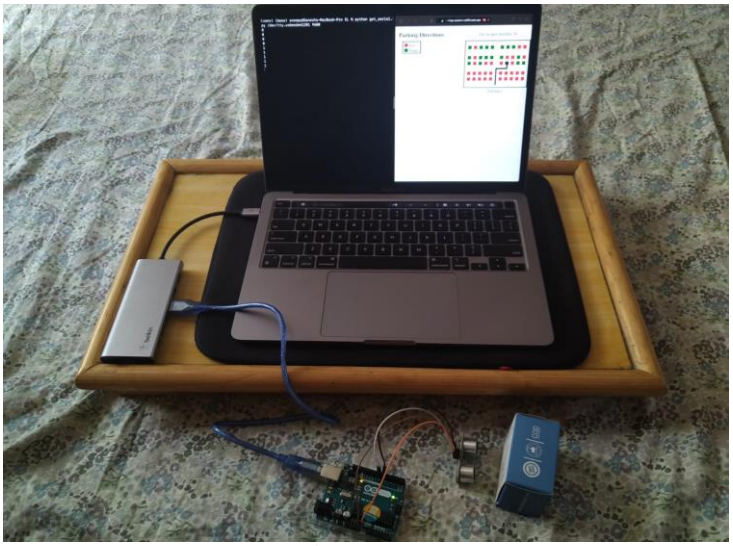


Fig 9. Pictures of the working model

6. CONCLUSION

In this paper we propose an intelligent parking system which is designed to manage parking spaces in urban areas such as shopping malls and stadiums. Most of the literature focuses on city-wide parking management systems as they can be used to ease city-wide congestion and utilize parking spaces all over the city. But urban areas such as malls and stadiums with relatively small parking spaces can also drastically increase the congestion in a part of a city since they receive high traffic inflow in short spans of time during special events.

The prototype built is cheap and scalable to a real parking space by replacing the controlling computer with a suitable board such as an ESP32. Using our design, a full scale parking system can be implemented which can manage incoming vehicles efficiently and prevent customer frustration. This is especially important to places with commercial interests, such as shopping malls.

Future work can be done to prove the feasibility of a full scale system by piloting it in a real parking space. There is also scope for addition of features such as showing real time directions to guide a vehicle to a parking spot, provisions to reserve parking spaces in advance and integrated payment services to pay for parking.

ACKNOWLEDGEMENTS

We are extremely grateful to Dr. Badari Nath for his invaluable guidance, technical assistance and mentorship in the preparation of this paper.

REFERENCES

[1] G. Yan, W. Yang, D. B. Rawat and S. Olariu, "SmartParking: A Secure and Intelligent Parking System," in IEEE Intelligent Transportation Systems Magazine, vol. 3, no. 1, pp. 18-30, Spring 2011, doi: 10.1109/MITS.2011.940473.

[2] Yanfeng Geng, Christos G. Cassandras, A new “Smart Parking” System Infrastructure and Implementation, Procedia - Social and Behavioral Sciences, Volume 54, 2012, Pages 1278-1287, ISSN 1877-0428, https://doi.org/10.1016/j.sbspro.2012.09.842.

[3] Faheem, S.A. Mahmud, G.M. Khan, M. Rahman, H. Zafar, A Survey of Intelligent Car Parking System, Journal of Applied Research and Technology, Volume 11, Issue 5, 2013, Pages 714-726, ISSN 1665-6423, https://doi.org/10.1016/S1665-6423(13)71580-3.

[4] Idris, Mohd & Y.Y, Leng & E.M, Tamil & N.M, Noor & Razak, Zaidi. (2009). Car Park System: A Review of Smart Parking System and its Technology. Information Technology Journal. 8. 10.3923/itj.2009.101.113.

[5] Janak Parmar, Pritikana Das, Sanjaykumar M. Dave, Study on demand and characteristics of parking system in urban areas: A review, Journal of Traffic and Transportation Engineering (English Edition), Volume 7, Issue 1, 2020, Pages 111-124, ISSN 2095-7564, https://doi.org/10.1016/j.jtte.2019.09.003.