Real-time Smart Parking System

JIANJUN WEI
Washington University in St. Louis
U.S.A.
jianjun@wustl.edu

LINXI LI
Washington University in St. Louis
U.S.A.
l.linxi@wustl.edu

WENXUAN ZHU
Washington University in St. Louis
U.S.A.
540516776@qq.com

Abstract

The fast pace life and population explosion in modern society makes finding a parking space not only a time-consuming but also an annoying task. To create an optimized solution, we decide to build a novel prototype to enhance parking efficiency. We have combined camera, senors, Raspberry Pi and various of AWS services together to create the smart parking system that could not only automatically detect car plate number but also give drivers a parking guideline. We restore the real scene in real parking lot, and thus, the design and implementations are based on the real scenario, which makes it easier to be applied. Series of experiments are done to prove the accuracy of the system and the results shows that our system is both effective and efficient. Besides, we provide several possible ways to further enhance this system.

Keywords: real-time, cloud computing, IoT, smart system

ACM Reference Format:
JIANJUN WEI, LINXI LI, and WENXUAN ZHU. 2023. Real-time Smart Parking System. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Finding a parking space is not only time-consuming but also an annoying task. Usually, the process would end up with a disappointed issue to the drivers. People often need to drive around the whole parking lot several times only to find one available space[13]. Traditional parking system could only provide the parking information, which is not "smart" enough and it would be a catastrophe when there is a large parking lot[1].

We could see that Internet of Thing (IoT) starts playing a more and more important role in Smart Cities[4] and various high technology applications have been utilized in this traditional area. In [5], the authors have applied an application of optical Wireless Sensor Network on the parking system. It is cost-efficient and optic WSNs are easy to maintain and to install. However, the accuracy of the detection of the license plate is quite impossible. In [3, 12], the authors have used RFID to make the process of check-in and check-out a fast-manner without stopping the car, which largely alleviate the traffic jam. However, if the RFID is damaged, or more than one tags are read, the accuracy could not be assured. In [8], Bluetooth is applied in the parking system. It is true that it could decentralize the system, but the range of Bluetooth is limited and the connection is to stable enough.

Considering how much pain the traditional parking lot bring to drivers and the IoT technique's popularization, we would like to build a real-time smart parking system that can not only make the whole process fully automated but also give drivers a clear guideline.

Initially, we provide a brief overview of smart parking system and our goals and requirements and the need for IoT devices to be integrated with cloud. Promote, we expand our view about the design in Section 3 and working of the proposed system implementations in Section 4 utilizing Raspberry Pi and several AWS services. By highlighting the key features of our work, we have described the convenience and benefits with IoT. Experiments are shown in Section 5 to prove that our system is reliable and efficient. We put more emphasis on the confidence of the recognition part and latency. To the end of the paper, related work and lessons we have learned are provided in Section 6 and 7. We review the state of the art in smart parking system and view it in a different angle.

2 Goals and Requirements

Our goal is to build a fully functioning smart Parking System that can automatically recognize cars' plate, give drivers a parking guideline, record the status of each car, manage the parking lot space and calculate the parking fee. To achieve this system, specific hardware and cloud services are needed. We need camera, Raspberry Pi, distance sensors, electric motors, resistors and AWS services including S3 Buckets, DynamoDB, AWS lambda, AWS IoT and AWS Recognition.

3 Design

In this section, we would like to show how we design the whole smart parking system. First, we would like to restore the real scene of the parking process in our daily life. And then, the working process of our smart parking system would

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

be shown to show how we make the whole system working in a fluent and correct way.

3.1 Restore the real scene

To restore the scene in the real life, we would like to simulate a real parking lot and imitate the process of getting in and out. As is shown in Figure 1, we could see that there are 4 parking space in our parking lot. We assume that the first two spaces are for the permanent users who have already paid the fee and the rest is for the temporary users who would like to park their cars for just several hours. Also, we build a parking barrier at the entrance of our parking lot, and it will open when the car is allowed to get in the parking lot and close when there is no permission.

Moreover, we set several distance sensors in our system. One of them is set with the camera. This combination is used for capturing the car plate when cars come in. The ultrasonic distance sensor would monitor the distance in real time, and when the distance is less than the threshold we set, the camera would be triggered to capture the image of car plate. Other distance sensors would be set for temporary users. They are used to see whether there is a car in the temporary parking space. This is important and provide useful information for the system since we could know whether or not there is enough space for an incoming temporary car to get in. Checking the parking status in real time would make the whole system in reality.

In addition, we would have a screen to show the real information of the parking system, such as "FULL", which means there is no space for the car comes in, or "Available", which means the car is allowed to the assigned parking space. Besides, we would calculate the parking fee for the temporary users.

Figure 1. Parking lot.

3.2 Working process of the parking system

Figure 2 shows the whole process of our smart parking system. We could see that the core of the the system is the Raspberry Pi, which is used to connect the hard ware and data transfer, and AWS applications, which is set for processing various of data such as car plate recognition and data storage.

When the car comes in, the sensor and the camera would work together to capture the image of car plate. And then, the image would be uploaded to the AWS S3 Bucket through the Raspberry Pi. The AWS Lambda function would be triggered when there is an update in the S3 Bucket and call the AWS Rekognition to detect the word in the image. After we get the real car plate, like "5ALN015" in the Figure 2, we would like to do the data processing through AWS DynamoDB.

First, we would like to do a query to retrieve the data from the database and make a judgement to see whether the user is the permanent one or the temporary one. For the permanent user, we would like to see the which parking space he/she has already bought. However, for the temporary user, we would make a check to see whether there is enough space for the new coming car. A parking space would be assigned if we get a positive response. Also, we would like to calculate the fee for the temporary users. Both temporary users or permanent users' information, such us car plate, entry time, departure time, would be recorded and stored in database.

After finishing the previous steps, the message would be sent back to the Raspberry Pi through AWS IoT and the message would be shown on the screen. The message would tell the users which parking space he/she should be in, parking lot status, and how much he/she should pay.

4 Implementation

In this section, we would like to illustrate how we realize the system. Intuitively, we not only need to write functions for hard wares such as the camera, which is used for capturing the car plate in real time, and the parking barrier, which is controlled by the motor, but also we have to realize the applications such as text detection and data transfer.

4.1 Implementation of endpoint

In our project, we need a Raspberry Pi 3B to collect the information and transfer data. The camera is used to captured the image of cars, and distance sensors are used to detect whether there is a car nearby. There is also an electric motor to control the gate of the parking lot.

4.1.1 Distance sensor. This economical sensor provides 2cm to 400cm of non-contact measurement functionality with a ranging accuracy that can reach up to 3mm. Each HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit. There are four pins: VCC (Power), Trig (Trigger), Echo (Receive), and GND (Ground). To protect the Pins from the voltage, for each distance sensors there would be two resisters. We are able to get the distance using the speed of the sound and the time that sensor receive the sound.

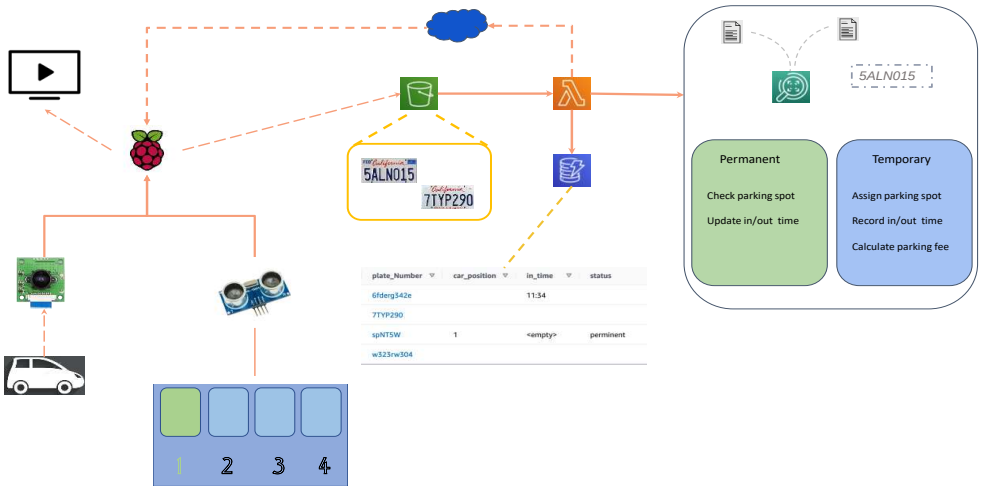

Figure 2. Working process of Smart Parking System.

4.1.2 Raspberry Pi 3b. Raspberry Pi is the core of the whole project and it would have the data collected from three distance sensors and the image captured by the camera. One of the distance sensors is used to detect whether there is a car coming nearby the gate, if so, Raspberry Pi will ask camera to capture a image and upload to the cloud. With the recognition result from the server, Raspberry Pi will give different instruction for the drivers according to the current situation of the parking lot and the type of incoming cars. Raspberry Pi will get the situation of the parking lot using the left two distance sensors. Since we have two temporary parking spaces, we could maintain a variable representing the left empty space. If there is still some empty space, we would open the gate and if there is no empty space but incoming car is a permanent car, we would also open the gate. Otherwise, the parking barrier would not open.

4.2 Implementation with AWS applications

Besides the hard wares, we use AWS applications to help us do the following process. Here we would involve several AWS clouding services including S3 Bucket, IoT, DynamoDB, Rekognition and Lambda.

4.2.1 S3 Bucket. To hold the image getting from the camera, we would like to choose S3 Bucket to store the information. Since the Bucket is a container for objects stored in Amazon S3, we could store large amounts of data in a Bucket to organize and configure access to the data to meet the specific need. Another function for the S3 Bucket in our system is that we could use it as a data backup. When there is a crash for either the hardware or the software, we could retrieve part of the data from the Bucket.

4.2.2 Lambda. All the functions in our system would be written and realized in the AWS Lambda. Functions in Lambda could call different AWS cloud services so that AWS resources could be used easily, and then, we could apply compute to data as it enters or moves through the cloud. In our project, the Lambda function would be triggered when there is an update in S3 Bucket. Then, it would call the AWS Rekognition service to Pick up the newest image storing in the Bucket. After detecting the text in the image and getting the real car plate, DynamoDB application would be called to store all the data. Finally, the Lambda function would send the response to the IoT.

4.2.3 Recognition. We would like to use AWS Rekognition to detect the car plate. It is not hard to see that there is a variety of information on the car plate such as the state name, the car plate number, and the slogan of the state such as "The Empire State" for the state of New York and "THE SUNSHINE STATE" for the state of Florida. It is really helpful that the AWS Rekognition could detect all the text appeared in the image. However, what we really need is the car plate number alone. Thus, we need to remove the noise and filter the information we detect through AWS Rekognition. Here we would like to use regular expression to get the real car plate number. We could see that either the slogan or the name of the state is composed by characters but the car plate is formatted not only by the characters but the numbers. Usually, the length of the car plate would be six or seven. Thus, the regular expression could be set as:

```
^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{4,23}$.
```

After we Pick up the real car plate number, we get the key of our system. And then, we would like to move on and go to the next step. We will first process the data, and then push it into DynamoDB.

4.2.4 DynamoDB. Amazon DynamoDB is a fully managed, key-value NoSQL database which is designed to run

high-performance applications. In our project, DynamoDB would have two functions. First, we store all the parking information in the database. Second, we would like to interact with the database and do some query to process the data we have.

We have created the table in DynamoDB and we set four items. "plate_Number" item is the number of the car plate. Also, the "car_position" item is the parking space the car has already been in, "in_time" and "out_time" item are for the entry time and departure time separately, and "status" is used to differentiate the permanent users from the temporary users.

As is shown in Figure 2, we need to distinguish the permanent user from the temporary user after we recognize the car plate. Thus, we need to do the query in the database since the information of the permanent user would be inserted at the very beginning.

Besides, we need to know whether the car is moving in the parking lot or is going to leave the place. To realize this function, we would like to clear the entry time every time when the car is out. Thus, if we find the "in_time" item is empty, then the car is coming in. We could do it better in the feature to retain the current data as history record and create a new one for the incoming car.

For the permanent user, if the car is going to getting into the parking lot, we would get the car position, which is the car place he/she has already bought, and concatenate it as part of response message. The car information would be printed on the screen as a notice. When the car comes out, we would update the database and record the "out_time" item. Permanent users do not need to pay the parking fee.

For the temporary user, if the car hopes to get in the parking lots, we need to make a check first. If there is enough space for a new car, we will give the car permission to get in and assign an parking space for it. However, the car is not allowed to get in if the parking lot is full. When the car leaves the parking space, it is required to pay the parking fee and the amount is calculated according to its entry time and departure time, which are recorded in "in_time" item and "out_time" item. The amount of parking fee would be one part of response message sent back to the Raspberry Pi, and it will finally be shown on the screen.

4.2.5 IoT. AWS IoT is used to transfer the data from the cloud to the Raspberry Pi. After processing data on the cloud service, we would generate the response message and send back to the Raspberry Pi so that the information could be shown on the screen. Also, the control of the parking barrier relies on the response message. Usually, the response message includes the plate number, the status of the car, which is used to see whether he/she is permanent user or not, the car position, the entry and departure time, and the information that shows whether the car is getting in or out the parking lot. These items would be set as json format and send back to the Raspberry Pi. Since the Raspberry Pi subscribe the topic in the cloud, when there is a new message generated and sent, the Raspberry Pi would receive the message and do the following actions.

4.3 Implementation of User Interface

The function of UI is to show useful information to drivers, including whether the parking lot is available, how much he/she need to pay, or which spot the car should go. User interface is implemented by Python Tkinter [11]. Every time when the Raspberry Pi receives messages from AWS cloud, it updates global variables such as "position", indicating which spot current car should go, and "car_park", indicating whether this car is get in or out of the parking lot. User interface updates the parameters, which are set as global variables, to decide which sign should be shown.

5 Experiments

This section presents experiments we conducted to evaluate the efficiency and effectiveness of our smart parking system. The experiments described in this section were performed using Raspberry Pi 3b. L298N and the electronic motor is used to control the parking barrier. PiNG ultrasonic distance sensors are for measuring the distance between moving or stationary objects. The Raspberry Pi Camera Module V2 is equipped to capture the image of car plate. This version has better image quality, color fidelity, and low-light performance, which is easy for beginners, and performs well in taking both high-definition video and still photographs. In addition, resistor ranges from 0 to 10M Ohm are set to protect the whole electric circuit. Moreover, we process the data through AWS cloud services.

The whole process would be shown in the demo video, which is attached in the support materials. Also, we have uploaded all the material on the github[1]. We also set some icons in our experiments, which is shown in Figure 3. The "AVAILABLE" means that there is at least one empty parking space while the "FULL" means that there is no space for the new one. "PASS" icon shows that the car is allowed to get in or out of the parking lot. The last icon indicates the specific parking space the system assigns to the temporary user.

5.1 Latency Measurement

To account for the communication delay in the latency analysis in our smart parking system, we pushed an event back and forth between Raspberry Pi and AWS clouds 10 times, measured each round-trip time for one process, and the time consumption of each part. The results is shown in Table 1.

From the Table 1, we could see that "Text Recog" part is the time consumption of detecting the car plate in Amazon Rekognition part. "Data Process" shows the time we consume

[1]https://github.com/weijj27/CSE520-real-time-system

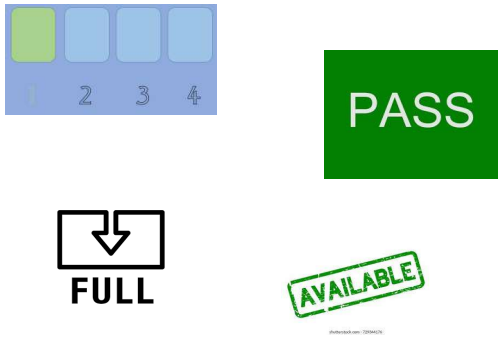

Figure 3. Parking icons.

Table 1. Latency Measurement of each processing part

Latency Measurement				
ID	Text Recog	Data Process	Data Transfer	Time
1	0.91	1.03	6.2	7.22
2	0.87	1.19	6.6	8.1
3	1.44	0.59	5.88	7.98
4	1.8	0.54	7.22	7.95
5	0.87	0.55	4.27	5.01
6	0.88	0.56	5.44	6.92
7	0.84	0.56	5.77	5.90
8	0.90	0.83	5.34	6.05
9	0.92	0.53	4.28	5.04
10	0.95	0.82	4.44	5.25

in database, including the various of queries we need to make the judgment and process the data, storing the new data and picking up all the information we need and generating the response message. "Data Transfer" shows the time we use to transfer the message from the AWS cloud to the Raspberry Pi. "Time" means the total time consumption in one round-trip.

Our measurements showed that "Data Transfer" performs worse than any other part in our experiment. The average time consumption in this part is 5.544 seconds while the "Text Recog" part is 1.038 seconds and "Data Process" part is 0.72 seconds. We also get the average time for the whole process of the 10 experiments is 6.542 seconds. The result shows that "Data Transfer" part occupies nearly 84.7% of the total processing time, which is an obvious bottleneck that could be improved in the feature.

One of the reasons that leads to the high latency is that there might be a block in the AWS message sequence. It is highly likely that the message has already been generated and sent into the message queue. However, nothing triggers the queue immediately until the waiting is out of time. For this situation, we may change another topic to have a try. Also, we need to have deeper research on the configuration of AWS IoT and remove the barrier between Raspberry Pi and message queue. Another reason is that the high latency is caused totally by the high latency of the network. This could be easily solved and improved when we switch into a better network environment and do the experiment to compare the results.

Table 2. Confidience of Text Recognition

ID	Car plate	Recog results	Confidence
1	5ALN015	Full	94.98%
2	XD1E0W	Partial	97.67%
3	M02021	Full	96.17%
4	215BG2	Partial	98.73%
5	BC18351	Full	99.08%
6	JXN646	Partial	99.34%
7	CGL1288	Full	97.65%
8	6WDG928	Full	96.87%
9	6XSU832	Full	99.91%
10	W8TWHAT	Full	100.00%
11	KEW6688	Partial	96.63%

5.2 Recognition Confidence

At the heart of our smart parking system is to detect the correct car plate. To see whether we could get the reliable car plate number, we pushed 11 different car plate images and measured the confidence in our experiment. The result is shown in Table 2.

From Table 2, it is obvious that the confidence of all the car plate is over 94% and the average confidence is 97.90%, which performs well and means the text we detect from the image uploaded in S3 Buckets is highly likely to be correct, and thus, reliable.

The "Recog results" tag shows whether we could recognize the whole car plate. The "FULL" tag means that we have recognized whole car plate while the "Partial" means we do not receive the 100% correct car plate. Although four out of eleven results are partial, this does not mean that we get the wrong results. Reviewing all these partial results, we find that we correctly recognize the part of the car plate but not combine them together. For example, the recognition result for "XD1E0W" shows that we have already get "XD1" and "E0W", and both of them have a relatively high confidence. One of the reasons behind that might be the fact that these there is a huge space in the car plate which seems separate the whole as two different parts. Also, the noise in the background may cause the previous situation. Intuitively, we could solve it by optimize our regular expression. Besides, we could pre-process the car plate image before uploading to the S3 Bucket. When pre-processing the image, we may remove the noise that would disturb the recognition or we could extract the car plate features to make it much easier for the AWS Rekognition to deal with the process.

6 Related Works

Conventional approaches to deploy the parking system is to install the whole package in the local machine. Usually, the system is off-line and hard to transplant, and thus, it would be a huge work to collect parking information and make improvements. Moreover, it is not safe to store the data in only one place, which leads to a big cost for data backup. In contrast, cloud computing[10] delivers various of services through the internet, which makes the lightweight client application possible. Besides, cloud-based storage makes it possible to save files to a remote database and retrieve them on demand, which seems to be a much better choice. In addition, with much powerful computational ability, it would provide much more creative and effective services.

In [7], the authors review the evolution of the smart parking system and they mainly focus on the development of various of sensors and the improvement of detection system. In [2, 9], importance are put on the whole system. cloud computing and IoT are introduced into parking system. The author describes the high-level system architecture and proves the correctness of the proposed model. [14], authors put emphasis on security and intelligence of the system. Since the parking process could be seen as stochastic process, new business promotions can be provided based on the prediction. [6] propose a novel "smart parking" system based on resource allocation and reservations. Through the new method, the average time to find a parking space and the parking cost would be highly reduced, whereas the overall parking capacity is more efficiently utilized.

7 Lessons Learned

The term "real-time system" refers to an information system with hardware and software components that can perform real-time application functions and respond to events within predictable time constraints.

In our project, the latency is mainly on the data transfer. It is highly likely to be caused by internet network hardware, remote server's location and connection as well as the routers that are located between server and online devices. Although the total latency is limited in nearly 8 seconds, there still a large space to make an improvement.

The latency is a kind of trade-off process and whether the system is good or bad depends on different requirements. For example, in a big market or airport, we need low latency and quick response to avoid the traffic jam. Thus, we could not afford to ignore the help of cloud computing and its powerful computational ability and better services. However, it would be fine to apply the system in a small building with local network.

8 Conclusion and Future Work

Our work represents a promising step toward developing real time smart parking system. By means of the cloud computing services provided by AWS and computing core Raspberry Pi, we would like to restore the real scene and allows applications to obtain optimal utilization to meet the basic parking lot requirements. Our experiment results demonstrate that our application performs well and could easily be extended and transplanted into other platforms. Although the latency for the whole process is a little bit high, it is totally acceptable in the real scene. Besides, the high confidence of our recognition part shows that out result is credible and reliable.

In the future, we would like to improve the design of the database so that it could deal with much more complicated scenarios. Also, we would focus on the data backup in case the services are crashed or down. In addition, we would improve our processing schedule to make it much more robotic and reduce the time consumption in data transfer.

References

[1] Fadi Al-Turjman and Arman Malekloo. 2019. Smart parking in IoT-enabled cities: A survey. *Sustainable Cities and Society* 49 (2019), 101608.

[2] Fadi Al-Turjman and Arman Malekloo. 2019. Smart parking in IoT-enabled cities: A survey. *Sustainable Cities and Society* 49 (2019), 101608. https://doi.org/10.1016/j.scs.2019.101608

[3] G Anusooya, J Christy Jackson, K Sathyarajasekaran, and Kumar Kannan. 2017. RFID based smart car parking system. *International Journal of Applied Engineering Research* 12, 17 (2017), 6559–6563.

[4] Michael Batty, Kay W Axhausen, Fosca Giannotti, Alexei Pozdnoukhov, Armando Bazzani, Monica Wachowicz, Georgios Ouzounis, and Yuval Portugali. 2012. Smart cities of the future. *The European Physical Journal Special Topics* 214, 1 (2012), 481–518.

[5] Jatuporn Chinrungrueng, Udomporn Sunantachaikul, and Satien Triamlumlerd. 2007. Smart parking: An application of optical wireless sensor network. In *2007 International Symposium on Applications and the Internet Workshops*. IEEE, 66–66.

[6] Yanfeng Geng and Christos G Cassandras. 2013. New "smart parking" system based on resource allocation and reservations. *IEEE Transactions on intelligent transportation systems* 14, 3 (2013), 1129–1139.

[7] MY Idna Idris, YY Leng, EM Tamil, NM Noor, Z Razak, et al. 2009. Car park system: A review of smart parking system and its technology. *Information Technology Journal* 8, 2 (2009), 101–113.

[8] Harkiran Kaur and Jyoteesh Malhotra. 2018. A review of smart parking system based on internet of things. *International Journal of Intelligent Systems and Applications in Engineering* 6, 4 (2018), 248–250.

[9] Abhirup Khanna and Rishi Anand. 2016. IoT based smart parking system. In *2016 International Conference on Internet of Things and Applications (IOTA)*. IEEE, 266–270.

[10] Dan C Marinescu. 2022. *Cloud computing: theory and practice*. Morgan Kaufmann.

[11] Alan D Moore. 2018. *Python GUI Programming with Tkinter: Develop responsive and powerful GUI applications with Tkinter*. Packt Publishing Ltd.

[12] Zeydin Pala and Nihat Inanc. 2007. Smart parking applications using RFID technology. In *2007 1st Annual RFID Eurasia*. IEEE, 1–3.

[13] E Cassin Thangam, M Mohan, J Ganesh, and CV Sukesh. 2018. Internet of Things (IoT) based smart parking reservation system using raspberry-pi. *International Journal of Applied Engineering Research* 13, 8 (2018), 5759–5765.

[14] Gongjun Yan, Weiming Yang, Danda B Rawat, and Stephan Olariu. 2011. SmartParking: A secure and intelligent parking system. *IEEE intelligent transportation systems magazine* 3, 1 (2011), 18–30.