Uludağ University Journal of The Faculty of Engineering, Vol. 28, No. 2, 2023 RESEARCH

DOI: 10.17482/uumfd.1247989

AN IOT BASED APPLICATION ON THE DETECTION OF CAR PARKING AREA BY AN ARTIFICIAL INTELLIGENCE APPROACH

Oğuzhan KARAKAYA [*]
Fikret Mert GÜLTEKİN [*]
Mert BAŞKAYA [*]
Yunus DOĞAN [*]

Received: 05.02.2023; revised: 09.07.2023; accepted: 14.07.2023

Abstract: Nowadays, smart cities solutions have been becoming popular day by day. The problem of finding a parking place is one of these implementations. Our study appeals to users who want to park their vehicle quickly and who want to find out whether the parking place is available to park before searching for a parking place. This application aims to find a solution to the problem of searching for a place to park. We have found a novel solution to this problem by using image processing methods and utilizing photographs of streetside parking places in the most effective way. In addition, an Android software has been developed through which we can deliver the results obtained with the image processing module to users via a mobile application. As a result, a parking IoT determination system has been implemented in this study with 92% accuracy value. Thanks to this system, vehicle owners who want to park their vehicles on the side of the street will be able to learn whether the parking places in a street are available in advance and they will be able to save their time by preferring more available streets.

Keywords: Smart Cities, Decision Support Systems, Image Processing, Mobile Applications, Internet of Things, Artificial Intelligence

Bir Yapay Zeka Yaklaşımı ile Araba Otopark Alanı Tespiti Üzerine Bir IoT Uygulaması

Öz: Günümüzde akıllı şehir çözümleri her geçen gün popüler hale gelmektedir. Park yeri bulma sorunu da bu uygulamalardan biridir. Çalışmamız hızlı bir şekilde aracını park etmek isteyen ve park yeri aramadan önce park yerinin müsait olup olmadığını öğrenmek isteyen kullanıcılara hitap etmektedir. Bu uygulama park edecek yer arama sorununa çözüm bulmayı amaçlamaktadır. Görüntü işleme yöntemlerini kullanarak, yol kenarındaki park yerlerinin fotoğraflarından en etkin şekilde yararlanarak bu soruna yeni bir çözüm bulunmuştur. Ayrıca görüntü işleme modülü ile elde edilen sonuçları bir mobil uygulama üzerinden kullanıcılara ulaştırabileceğimiz bir Andorid yazılımı gerçekleştirilmiştir. Sonuç olarak bu çalışmada %92 doğruluk değerine sahip bir otopark IoT belirleme sistemi gerçekleştirilmiştir. Bu sistem sayesinde araçlarını cadde kenarına park etmek isteyen araç sahipleri, bir caddede park yerlerinin müsait olup olmadığını önceden öğrenebilecek ve daha müsait caddeleri tercih ederek zamandan tasarruf edebilecektir.

Anahtar Kelimeler: Akıllı Şehirler, Karar Destek Sistemleri, Görüntü İşleme, Mobil Uygulamalar, Nesnelerin İnterneti, Yapay Zeka

[*] Dokuz Eylul University, Department of Computer Engineering, 35390, Buca/İzmir
Corresponding Author: Yunus DOĞAN (yunus@cs.deu.edu.tr)

1. INTRODUCTION

Individual transportation vehicles are more comfortable and every year millions of cars are produced and released to the market. However, the number of cars and the parking places for vehicles (such as indoor parking) do not increase proportionally. Many car owners prefer to park on the side of the road, as there are few parking areas or parking areas are far from the place, they want to reach but it can be parked in a single row here. In this case, the owner of the car looks for a parking place to park the car, without knowing whether the streets are available for parking and this causes a waste of time. In addition, this situation causes vehicles to consume more fuel. It does not only harm the person financially but also causes environmental pollution due to the exhaust gas released into the environment.

Finding a parking place in the places visited is indispensable for car owners. People have always tried to find a solution to the problem of parking vehicles. Almost all of these solutions have been implemented using technology.

Various places such as large shopping malls use devices that detect whether the parking lots around them are full. These devices are usually located in front of or at the top of the places where each vehicle will be placed and give information about the occupancy of the parking lot before customers enter the park. However, it is not possible to put a vehicle detection device in every empty space on the streets and avenues. It is also more effective for people to know if the parking area is empty before they go to that area. So, we formed our project according to these requirements. It is supposed that photographs are taken by cameras that were previously built on public transportation vehicles and test vehicles. However, we used photographs that were taken by our smartphones. Thus, empty parking areas are detected. The processed data is sent into a database that can be updated continuously in the background. Through a mobile application, users are able to know which street is more suitable for parking before going to that street. In this way, it prevents traffic congestion on that street and stops wasting time and fuel. It also stops nature pollution.

We developed an easily accessible mobile application to solve problems and meet the needs of the project. This application was built based on the data which was obtained in the image processing module.

Difficulty in accessing data: Users want to access the information of the available parking areas quickly and in its most up-to-date form and they also need a clear interface.

The purpose of the project is to decrease the time and fuel waste while searching for a parking place, the air pollution due to unnecessary fuel wastage, and to alleviate traffic congestion due to looking for a parking place in a mobile application that shows users available parking places in the map. While doing this project, it is so important for us to stay stuck to up-to-date technologies and follow them. This project utilizes image processing, the Yolo algorithm (uses neural network), and IoT (in future works). We decided to do this project because, in the increasingly populated world, we want to contribute to humanity in the smart city field.

In this parking system (Rahman et al., 2019), users can check whether the parking area is suitable to park or not without arriving at the parking area. This project consists of Arduino Mega 2560 and ultrasonic sensors (Jo et al., 2019). It is checked via ultrasonic sensors whether a parking lot is suitable or not. If there is an empty lot, the Arduino chip acknowledges this information and opens the main entrance to the parking area. The Arduino board is connected to a website and it gives the parking lots information to the website (Selvaraj et al., 2017). This system cannot be used by medium-scale shopping malls, or movie theaters because it is a high-cost system. At many public places, the system only shows the availability but it cannot show the exact slot but this parking system gives the exact slot to the users.

In this parking system (Nisha et al., 2020), the users enter the car park with the RFID cards which belong to each registered user. This system provides a manageable payment system and

also users can display tickets. It provides the closest parking lots. The proposed methodology interfaces RFID technology with an IoT device and with a website.

It consists of three sections: The first section is the IoT section which incorporates Arduino devices together with IR sensors (Ghorpade et al., 2020). The parking lot availability is provided via this section to users. Thus, they can book the slots to park. The second section contains the cloud-based web services. It acts as an intermediary between users and parking lots. The cloud stays up to date as long as the information on the availability of the parking lots is supplied. The third section is the user side. Users get notifications about the parking zone after scanning the RFID card via SMS through the GSM module.

NodeMCU and Arduino UNO are used as microcontrollers in this project (Ch'ng et. al., 2019). The number of parking slots is 12. Infrared (IR) sensor modules, Liquid-crystal display (LCD), and servo motor are controlled by the NodeMCU V3 (Aqeel, 2018) board and Arduino UNO (Smart Car Parking System using Arduino UNO et al., 2018). NodeMCU has a built-in ESP8266 (my2cents et. Al., 2019) inside to connect to the internet. A NodeMCU V3 board and a smartphone are required to connect to the hotspot network in order to receive data. The movement of the entrance and exit gates depends on the NodeMCU V3 board to receive the signal from the IR sensor. Available parking slots are also displayed on the webpage. The users may also scan the QR code before entering the car park, thus they know the exact slot that is empty. The LCD also displays the number of parking slots available. There is no car allowed to enter the car park when the car park is fully parked. If the internet connection cannot be provided, then up-to-date information cannot be supplied to the website. Also, Arduino C language, HTML, Microsoft Azure, and Firebase Realtime Database are used in this project.

In this project, an Android mobile application has been developed. The application aims to make the users find a parking place easily. When users open the application, they select a street and a map shown. They can find out whether there are parking places available on the selected street by looking at the map. There are availability situations written for the streets. They are the low possibility to park, the medium possibility of parking, and the high possibility of parking, and they are shown as red, yellow, and green respectively.

Photographs of the streets are taken and a dataset is created. The image processing is applied to this dataset, the processed data is sent to the database module, and the mobile application accesses the database to obtain up-to-date information, and conditions for available parking are determined in the mobile application module. A model in the Darknet Library which has a high success rate is used to detect the cars in the photographs. As long as the street data is provided to the image processing module, the system works correctly.

The difference between this project and other parking projects is, this project is valid for roadsides, not for specifically determined parking areas. The parking systems in car parks have parking sensors at each parking lot. So, the parking lots are already determined whether it is empty or not and it can be understood via these sensors. Since this system is for roadsides, there is no determined parking lot and it cannot be detected by sensors, because it is not known where people park by the roadsides. This distinguishes this project from others.

We believe this project alleviates traffic congestion, saves people’s time, and stops unnecessary fuel consumption.

The main goal of the project is to create a decision support system to assist car drivers in order to find available parking slots on streets. This system consists of four parts which are the internet of things module, the image processing module, the mobile application module, and the database module.

It is supposed that photographs and location data are obtained from test vehicles but we provide them manually for now. The data is collected in a database. After that this data is processed in the image processing module. After manipulating data, the mobile application receives the processed results from the database and it decides the availability of the streets. The

mobile application module illustrates if selected streets have available parking areas. Available parking areas are kept statistically for each street separately.

In the end, the system depicts a decision support interface on the streets according to available parking area density after obtaining and processing data but the system does not directly show the parking area. The target group is car drivers who want to find parking areas effortlessly.

Python is selected for image processing with the Numpy library, Scipy library, OpenCV library, Python Imaging Library, the Darknet library, and the Yolo algorithm. According to the project (Umesh et al., 2012), Python Imaging Library is a popular and powerful tool with graphical capabilities.

The report (Kempet et al., 2021) provides information that 72.5% of mobile phone users get a share of web traffic originating from Android devices. Android is the mobile operating system due to its popularity and accessibility. The android application is developed by Kotlin programming language. Because Kotlin is a highly supported language by Google and compared to Java, Kotlin fixes some issues of Java such as null safety (Comparison to Java | Kotlin. et al., 2021). MSSQL is used for database operations for the reason that MSSQL is highly stable for each project module.

2. RELATED WORKS

This project (Rahman et al., 2019) proposes improper parking real-time detection, estimates each vehicle's parking lot duration, and parking chargers' automatic collection. It also provides information that a driver's search for a convenient parking lot takes approximately 15 minutes and this causes extra fuel consumption, traffic congestion, and air pollution. This project's solution covers city-wide smart parking management solutions. This solution includes information on parking area availability and the reservation system. Drivers send requests to the central server and this central server provides information on parking suitability all over the city. This information comes from each parking facility via Wi-Fi access points and WLAN or Wi-Fi-integrated laptops/workstations. To understand the availability of parking areas, ultrasonic sensors are used. Additionally, Arduino MEGA 2560 (Siswanto et al., 2019), a led and alarm Ic module is used for displaying per module for taking the license plate of a car. The communication through the local server is provided via 802.11 b/g/n compatible wireless modules. The alarms ring if no desired park happens. This project includes a camera.

This project (Crisostomo et al., 2019) was designed for multi-story parking garages in order to not waste time, and fuel and to prevent traffic congestion. Their proposed system is designed with Python Idle and OpenCV (Boyko et al., 2018) libraries. Thanks to real-time image processing techniques, parking slot availability are updated. Image processing flow consists of image acquisition, image enhancement, image segmentation, and image detection. The system snapshots multiple photographs to cover all the parking areas for every floor. For the available slots, the boxes are marked with green but if it is occupied then the mark turns to red. Their methodology for image processing is the first to determine whether slots or rectangles are full by examining the pixels. This is made by first filtering and blurring the image, then taking the average of pixels and comparing it with the color of the floor or pavement. If there is a significant change in this average, then boxes are marked as red or green. For the count of the parking areas, the numbers are placed in the center of the rectangles. Moreover, they keep the coordinates of the rectangle corners in separate files. However, coordinates are accepted as input for the first time.

In this project (Yusnita et al., 2012), an intelligent Parking Space Detection System Based on Image Processing is designed for the detection of vehicles parked in empty parking spaces in car parks. In this project, a camera is used as a sensor for image detection. The main purpose is to prevent spending time searching for an empty parking space. This system was designed thanks to MATLAB (Morais, et al., 2018). Image processing methods first examine the pixels to determine whether slots or rectangles are filled. This is done by first filtering and blurring the image, then

averaging the pixels and comparing it to the color of the ground or pavement. If there is a significant change in this mean, the boxes change. In this way, it detects whether the parking lot is full or empty. However, this project is for small-vehicle parking.

This project (Balmiki et al., 2020) is to ensure minimization and solve the traffic constraints in parking areas. The system in this project benefits from RFID and IoT in malls and buildings. This system uses RFID in order to provide high identification accuracy, efficient management of parking slots, easy in-and-out access for drivers with the least human intervention, and low deployment, operation, and maintenance cost (Hanis et al., 2021). This SPS checks the availability of parking slots in the nearest parking area. The disadvantage of the project is that the booking of slots for multiple users at the same time is not configured. It is not possible to pre-book the slots. At the entry, there is a 7-meter counter to count the number of cars in the area. The door of the garage is automatically closed when the highest value of the meter exists and it cannot be opened until the next new car enters. After reaching the slot, the vehicle parking card should be read by the RFID reader. Also, the time elapsed in the parking area is provided by reading RFID cards and the fee is calculated.

3. MATERIALS AND METHODS

If we mention the functional requirements of our system, firstly data such as street name, neighborhood name, coordinates of street obtained from Google Maps, etc. should be properly included in the database of the system. Data streams coming at regular intervals should be provided properly to the database. People using the application should be able to easily select the neighborhood and the street, and they should be able to observe the street they go through the map inside the application, to verify this location. The application should calculate availability of parking areas proportionally whether the city, the district, the neighborhood and the street which users choose are full or empty and it should display it with colors such as red, yellow and green.

In the mobile application, a page which is called Info Screen should inform users about using the application. So, users of all levels can understand how to use the application in a simple way. The mobile application should contain a splash screen and it should have a user-friendly design. In this project, we decided to use MSSQL to keep data. The reason why MSSQL is preferred in this project is because it has free versions available, it supports various purpose-optimized engines (storage systems), it provides high performance even with large amounts of data, it is easy to use and it has a user-friendly interface.

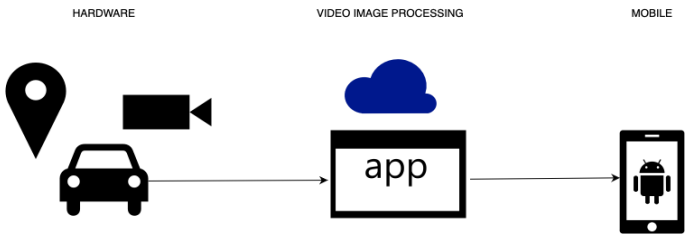


Figure 1:
Architectural view

If we mention the non-functional requirements of our system, at first the application should work stably on all Android operating phones regardless of screen size (provided that very old versions are exceptional). Entering the application should not take too long, it should be possible to open the mobile application within 4 seconds. The cameras where the images are obtained, installed on the vehicles should be mounted in the middle of the vehicle and have a wide angle. After a photograph is taken from a camera, image processing should be done as soon as possible, and a numerical feedback and data is required. The photograph should be taken in as low

resolution as possible but in a useful way, and time and storage space should be saved in transmitting to the database.

The database where the photographs are kept should be cleaned up to date at certain intervals, so that the security gap that may occur from the images can be prevented. To avoid a crash in the application, it should be developed in the most stable way and data retrieval from the database should be at certain intervals. In this way, the possibility of system crashes can be kept minimum. The application interface should be simple enough that users of all levels of knowledge can use it comfortably. Instead of numerical proportions, color graphics and signs should be used, thus a comfortable usage is provided.

3.1. Design of the System

Architectural design of this project is shown above. As mentioned earlier, the project consists of four modules which are the hardware, the image processing, the database (kept in cloud) and mobile modules. Location and photograph information is obtained via public transportations and test vehicles in the hardware module. However, this information is provided manually for now.

This information is sent to the server repeatedly in certain intervals. In the image processing module, incoming data is processed. Location information is related to the photograph. Information about whether a parking place is big enough to park is obtained by measuring the lots using comparing objects. Some information is predefined such as the length of cars. The longitude and latitude values of the streets are taken from Google API (Figure 1).

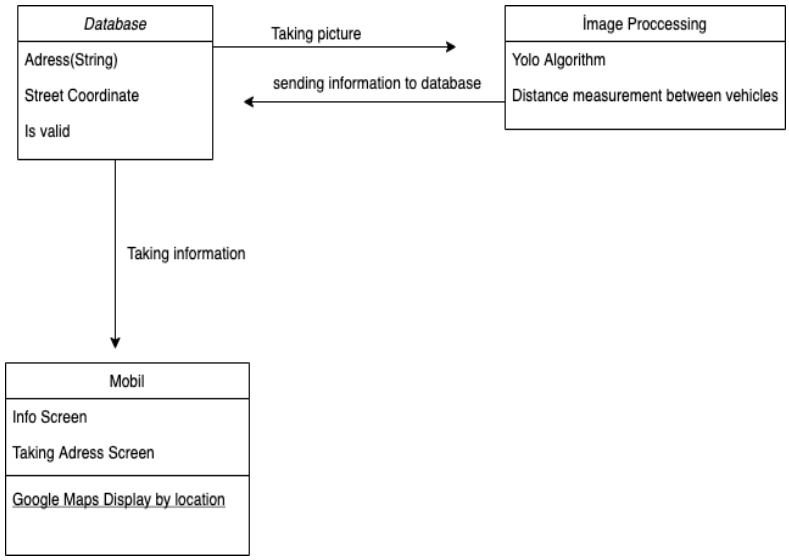


Figure 2:
UML class diagram

Photographs taken from the database are firstly processed by applying the Yolo algorithm. The result whether the car can park or cannot park is sent to the database and the table in the database is updated according to these results. The Mobile part of the system has a minimalist design in a way that makes it easy for users to use. It searches the database according to the address information selected by users and it shows its location in the Google Map. With the red, yellow, and green cursor, users can easily access the occupancy rate of available parking areas (Figure 2).

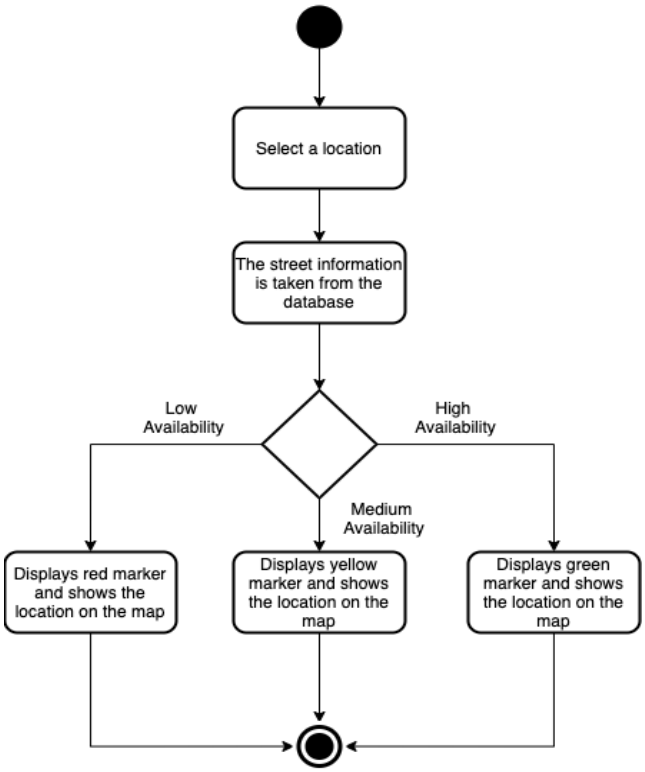


Figure 3:
Activity diagram

The start point of the activity diagram (Figure 3) is selecting a location from the main screen. After the selection, application connects to the database and information is queried. According to the result, different colored markers are printed to the screen.

4. IMPLEMENTATION

The implementation may be summarized as the pseude code below;

- *Importing the necessary Image processing libraries:*
  - *Darknet, Opencv, matploitlib, imutils, numpy, distance and defaultdict*
- *Importing Yolov3 algorithm*
  - *Set the location and name of the cfg file*
  - *Set the location and name of the pre-defined weights file*
  - *Set the location and name of the COCO object class files*
  - *Load the network architecture*
  - *Load the pre-defined weights*
  - *the COCO object classes*
- *Read images from file*
  - *Set the default figure size*
  - *Load the image*
  - *Convert the image to RGB*
  - *Resize the image to the input width and height of the first layer of the network*

- *Measure distance between vehicles*
- *Draw according to parking situation*

4.1. Image Processing Implementation

Respectively, the processes have been implemented;

```
[1] import cv2, os
    import matplotlib.pyplot as plt

    from darknet import Darknet
    import cv2
    import matplotlib.pyplot as plt
    from utils import *
    import imutils
    from imutils import perspective
    from imutils import contours
    import numpy as np
    from scipy.spatial import distance as dist
    from collections import defaultdict
```

Figure 4:
Importing libraries

The libraries like Opencv and Darknet are imported (Figure 4).

```
# Set the location and name of the cfg file
cfg_file = 'cfg/yolov3.cfg'

# Set the location and name of the pre-trained weights file
weight_file = 'yolov3.weights'

# Set the location and name of the COCO object classes file
namesfile = 'data/coco.names'

# Load the network architecture
m = Darknet(cfg_file)

# Load the pre-trained weights
m.load_weights(weight_file)

# Load the COCO object classes
class_names = load_class_names(namesfile)
```

Figure 5:
Importing Yolov3 algorithm

Pre-trained data are determined. The Yolov3 algorithm is used for image processing (Figure 5).

```
from google.colab.patches import cv2_imshow
sizeOfPark=25
carAreaCheck = [0] * sizeOfPark
ilen=0
# Set the default figure size
plt.rcParams['figure.figsize'] = [24.0, 14.0]
IMAGE = 'car3.png'
# Load the image
img = cv2.imread(IMAGE)

# Convert the image to RGB
original_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# We resize the image to the input width and height of the first layer of the network.
resized_image = cv2.resize(original_image, (m.width, m.height))
```

Figure 6:
Reading images from file

An image is read from a file and the image is resized (Figure 6).

```
image = cv2.imread(IMAGE)
box_measures = defaultdict(dict)
width = img.shape[1]
height = img.shape[0]
colors = ((0, 0, 255), (240, 0, 159), (0, 165, 255), (255, 255, 0),(255, 0, 255))
for i, box in enumerate(boxes):
    x1 = int(np.around((box[0] - box[2]/2.0) * width))
    y1 = int(np.around((box[1] - box[3]/2.0) * height))
    x2 = int(np.around((box[0] + box[2]/2.0) * width))
    y2 = int(np.around((box[1] + box[3]/2.0) * height))

    if x2-x1 > 50:
        box_measures["box"+str(i)] = {"top_left": (x1, y1), "top_right": (x2, y1),"bottom_right": (x2, y2),
                                      "bottom_left": (x1, y2), "center": (int((x1+x2)/2),int((y1+y2)/2))}
```

Figure 7:
Car box drawing

Top of square to bottom of square and left to right square are drawn properly and also color arrays are created for drawing (Figure 7).

```
#print(int(refObj[1][0])-int(cXorginLast))
  tempRate  = 150/int(box_array[2][1]-box_array[0][1])
#print(box_array[2][1]-box_array[0][1])



D = (float(int(refObj[1][0])-int(cXorginLast))) * tempRate

(mX, mY) = midpoint((refObj[1][0], refObj[1][1]), (cX, cY))
```

Figure 8:
Measuring distance between vehicles

For the square that allows to mark the cars, the distance between the sides(top of square to bottom of square) is proportional to 150 cm. So, therefore, the net distance between the cars can be found almost accurately (Figure 8).

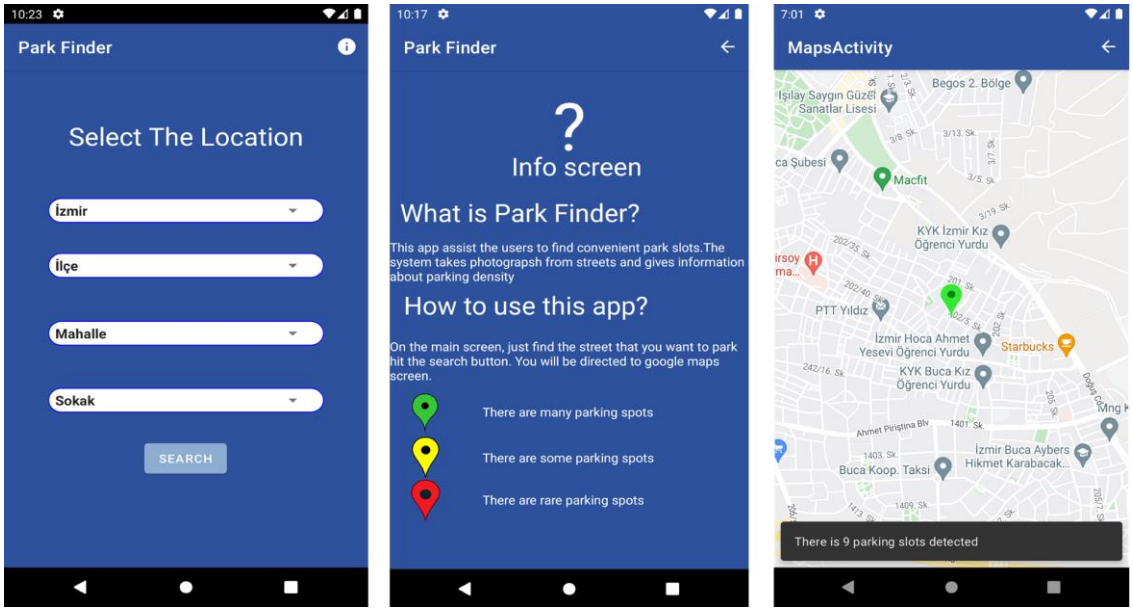
```
if D<0:
  refCoords = np.vstack([box0_array, (cX, cY)])
  objCoords = np.vstack([refObj[0], refObj[1]])
  cv2.rectangle(orig, (refObj[0][0][0], refObj[0][0][1]),
              (refObj[0][2][0], refObj[0][2][1]), (0, 255, 0), 2)
  D = (float(int(cY)-int(refObj[0][0][1]))) * tempOranı

  (mX, mY) = midpoint((refObj[1][0], refObj[1][1]), (cX, cY))

  if D/100 > 3.0: #Success
    cv2.putText(orig, "{:.1f}m".format(D/100), (int(mX), int(mY - 15)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.55, colors[3], 2)

    cv2.putText(orig, "Distance OK  ", (int(mX), int(mY + 15)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.55, colors[3], 2)
    cv2.line(orig, (int(refObj[0][0][1]), int(refObj[1][1])), (int(cXlast), int(cY)), colors[3], 2)
    cv2.circle(orig, (int(refObj[0][0][1]), int(refObj[1][1])), 5, colors[3], -1)
    cv2.circle(orig, (int(cXlast), int(cY)), 5, colors[3], -1)
    carAreaCheck[ilen%sizeOfPark]=1

  else:
    cv2.putText(orig, "{:.1f}m".format(D/100), (int(mX), int(mY - 15)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.55, colors[0], 2)

    cv2.putText(orig, "No  Distance ", (int(mX), int(mY + 15)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.55, colors[0], 2)
    cv2.line(orig, (int(refObj[0][0][1]), int(refObj[1][1])), (int(cXlast), int(cY)), colors[0], 2)
    cv2.circle(orig, (int(refObj[0][0][1]), int(refObj[1][1])), 5, colors[0], -1)
    cv2.circle(orig, (int(cXlast), int(cY)), 5, colors[0], -1)
    carAreaCheck[ilen%sizeOfPark]=0
  ilen+=1
```

Figure 9:
Drawings according to parking situation

The minimum vehicle length is taken as 3 cm (it can be changed manually). If it is less than this distance, it cannot be parked. if it is greater than this distance, it is considered as “it can be parked”. Here reference points are determined, and inter-vehicle lines and reference points are drawn as green in available parking situations and red in no parking situations (Figure 9).

4.2. Mobile Application Implementation

For the Android application, there are three pages. The main activity asks for address information by using spinners. The mobile application works only in Izmir. The second activity is information activity which gives information about the application. The last page consists of a map fragment, and it displays the given address (Figure 10).



a. *b.* *c.*

Figure 10:
General interfaces of the mobile application;
a. *Main Screen* ***b.*** *Information Screen* ***c.*** *Map Screen*

The address is marked with a marker. Apart from the map, there is an information panel, and this consists of Image View and TextView. Image View is just a colored circle. The colors give information about availability. According to the circle color, availability information explained in text view shortly. Behind the map, there are some important structures. To use Google Maps in a mobile application, Google serves API (maps SDK for Android). Also, to convert between addresses and coordinates, Google publishes the Geocoding API. These APIs are commercial APIs.

5. EXPERIMENTAL STUDIES

Our database has 50 photographs to test. It is mentioned in the (Table 1) that the model detected 18 available parking areas but in actuality, there are 22 available parking areas.

There are 28 parking places which are not suitable to be parked. All the parking places which are system found as available to park are detected correctly. Some of the errors which indicates the success of the model, are displayed in the (Table 2).

Table 1: Confusion matrix

	Positive	**Negative**	
True	18	4	Recall = 0.8182
False	0	28	Specificity = 1
	Precision=1.0	Negative predictive value = 0.875	Accuracy = 0.92 F1-score = 0.9

Table 2: Confusion matrix

	Formula	Score
Mean Absolute Error	$\frac{\sum_{i=1}^{n} \lvert e_i \rvert}{n}$	0.08
Mean Squared Error	$\frac{1}{n}\sum_{i=1}^{n}(e_i)^2$	0.08
Root Mean Squared Error	$\sqrt{\frac{\sum_{t=1}^{T}(x_{1,t}-x_{2,t})^2}{T}}$	0.282842712474619
Coefficient of Determination	$R^2 = 1 - \frac{\sum_i (e)_i^2}{\sum_i (y_i - \bar{y})^2}$	0.6753246753246753

This is how a sample photograph taken from a camera is stored in the database (Figure 11).

Figure 11:
An enough distance in a photograph

Reference points and the line are drawn as green because the distance between cars is greater than 3 meters (Figure 12). This is how the sample photograph taken from the camera is stored in the database (Figure 13). Reference points and the line are drawn as red because the distance between cars is greater than 3 meters (Figure 14).

The database name is SmartCityDB and the table name is ParkPlaces. ParkPlaces has six fields. These are id, location, url, addressName, isParkingAvailable and date (Figure 15).

Figure 12:
The enough distance test in this photograph

Figure 13:
An insufficient distance in a photograph

Figure 14:
The insufficient distance test in the photograph

Results | Messages

	id	location	url	addressName	isParkingAvailable	date
10	15	38.3754779,27.188571999999997	car27.png	Buca Atatürk Mahallesi 201/6 Sokak	1	2022-05-17 21:09:29.
11	16	38.3754779,27.188571999999997	car29.png	Buca Atatürk Mahallesi 201/6 Sokak	1	2022-05-17 21:09:36.
12	17	38.3706998,27.195186399999997	car26.png	Buca Kuruçesme 203/25	1	2022-05-21 11:33:06.
13	18	38.3713075,27.193652	car16.png	Buca Kuruçesme 203/3	1	2022-05-21 11:33:06.
14	19	38.3713075,27.193652	car17.png	Buca Kuruçesme 203/3	1	2022-05-21 11:33:06.
15	20	38.3713075,27.193652	car22.png	Buca Kuruçesme 203/3	1	2022-05-21 11:33:06.
16	21	38.3713075,27.193652	car24.png	Buca Kuruçesme 203/3	1	2022-05-21 11:33:06.
18	23	38.3716483,27.193692799999997	car12.png	Buca Kuruçesme 203/6	1	2022-05-21 11:33:06.

Query executed successfully.

Figure 15:
The table in the database

Each record represents a parking place. Id is the specific number for each record which is kept in the table. Location refers to the longitude and latitude value of the location of the street where the park place belongs. Location is obtained from Google Map API. Parking places in the same street have the same location value. When we query according to the location value, in SQL, we obtain all parking places in the relevant street. Url refers to the name of the photograph file and it states the link of the photograph file. Address name represents the address of the street. IsParkingAvailable states whether the parking place is suitable to park or not. Blank images without vehicles parking spaces are defined true. Date states the time when the record is added to the database.

The Python module checks all the parking places by using their photograph file and determines whether the parking places are suitable to park or not and it updates the

isParkingAvailable value in the database according to the result. 0 is not suitable to park and 1 is suitable to park (Figure 16).

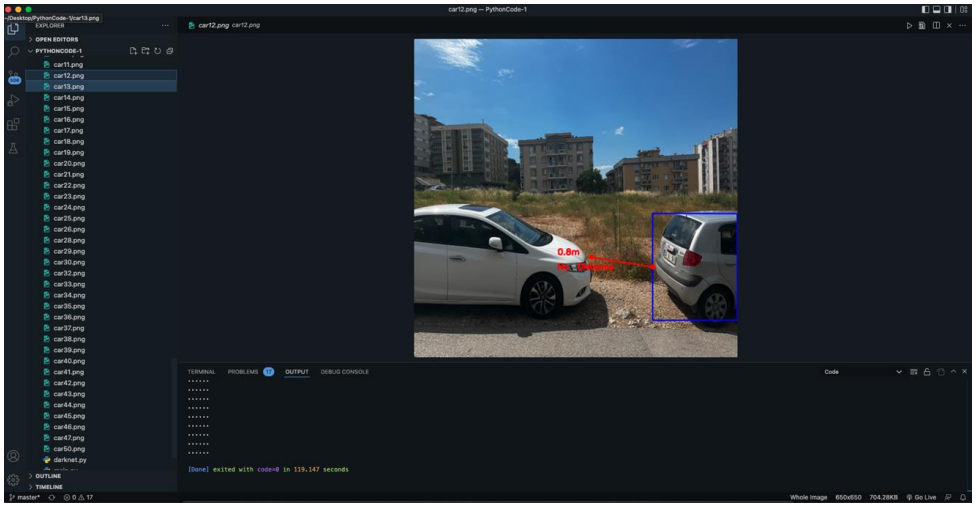


Figure 16:
The Python test screen in the Python module

If the distance between two cars is smaller than 3 meters, therefore it is unsuitable to park (Figure 17).

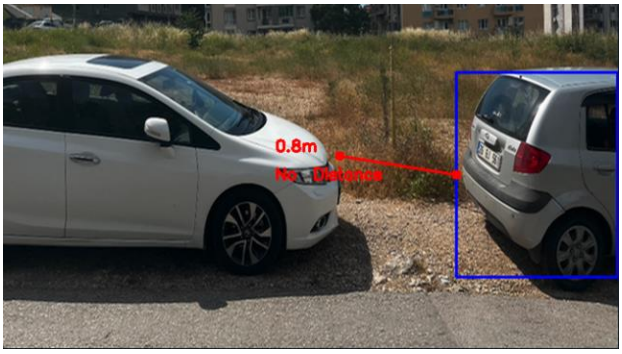


Figure 17:
An example output in the Python module

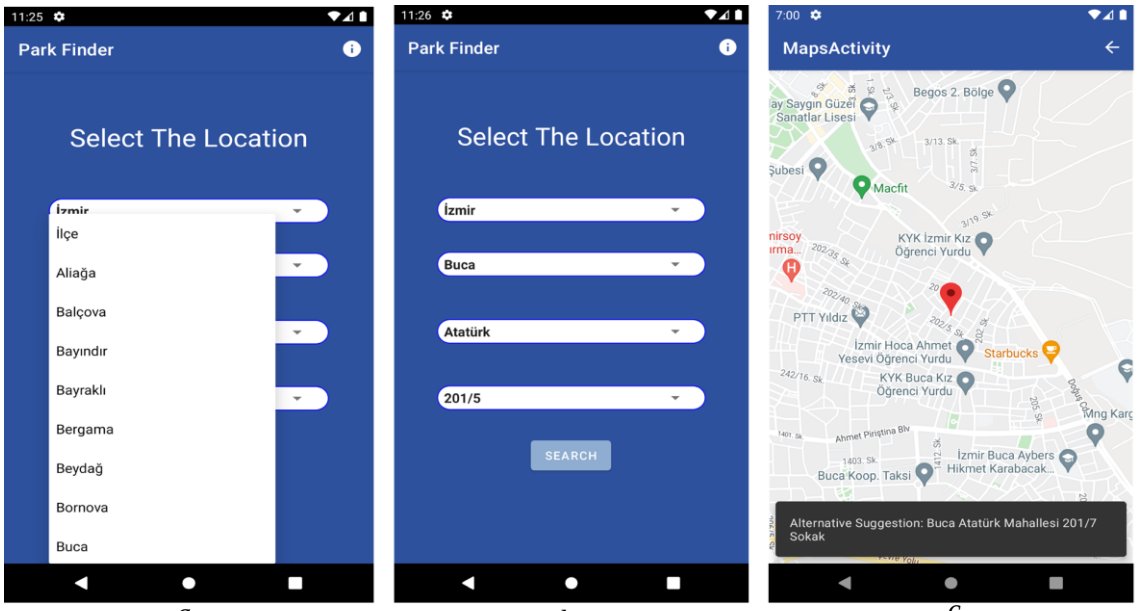
The Python module updates the isParkingAvailable values in the database, according to the result, as below (Figure 18).

Results | Messages

	id	location	url	addressName	isParkingAvailable	date
10	15	38.3754779,27.188571999999997	car27.png	Buca Atatürk Mahallesi 201/6 Sokak	1	2022-05-17 21:09:29.
11	16	38.3754779,27.188571999999997	car29.png	Buca Atatürk Mahallesi 201/6 Sokak	1	2022-05-17 21:09:36.
12	17	38.3706998,27.195186399999997	car26.png	Buca Kuruçesme 203/25	0	2022-05-21 11:33:06.
13	18	38.3713075,27.193652	car16.png	Buca Kuruçesme 203/3	0	2022-05-21 11:33:06.
14	19	38.3713075,27.193652	car17.png	Buca Kuruçesme 203/3	0	2022-05-21 11:33:06.
15	20	38.3713075,27.193652	car22.png	Buca Kuruçesme 203/3	0	2022-05-21 11:33:06.
16	21	38.3713075,27.193652	car24.png	Buca Kuruçesme 203/3	0	2022-05-21 11:33:06.
17	22	38.3713075,27.193652	car25.png	Buca Kuruçesme 203/3	0	2022-05-21 11:33:06.
18	23	38.3716483,27.193692799999997	car12.png	Buca Kuruçesme 203/6	0	2022-05-21 11:33:06.
19	24	38.3716483,27.193692799999997	car13.png	Buca Kuruçesme 203/6	0	2022-05-21 11:33:06.

Figure 18:
Updated isParkingAvailable values in the database

This spinner lists the districts of Izmir. Users can scroll down to see other districts. For this test, 201/5 street was selected to show the information. Also, the map is clickable. Users can tap another location by clicking on the map and see the information about that street as another option. If the number of available parking areas is less than 20% of the total parking areas, the marker turns to red. At the bottom of the maps, the nearest green marked street suggested as a text (Figure 19).

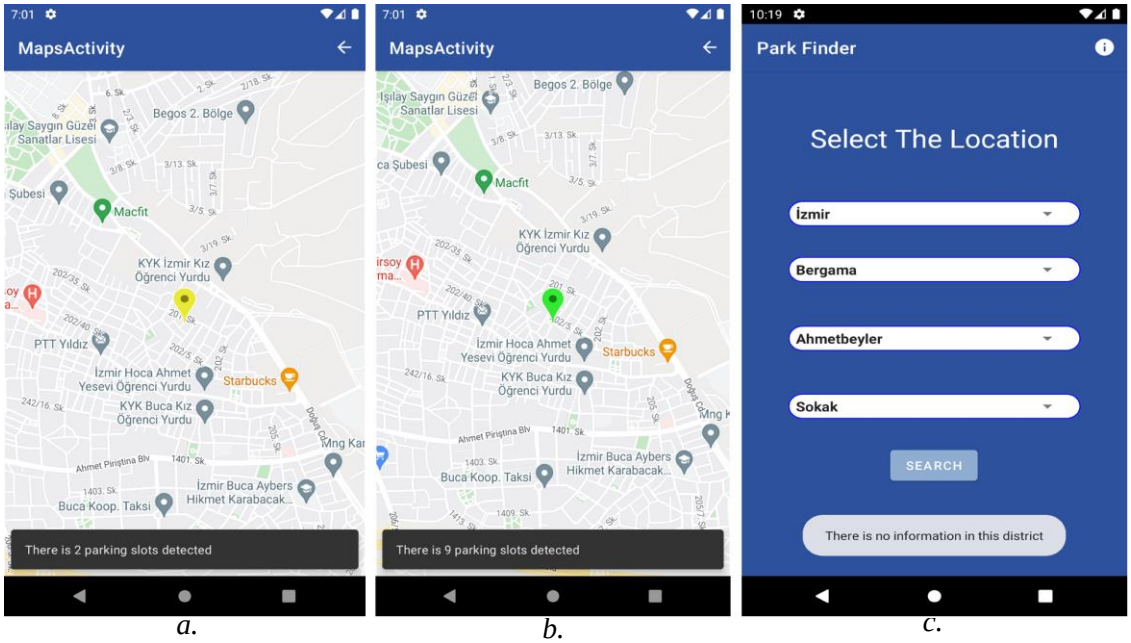


a. *b.* *c.*

Figure 19:
Test interfaces of the mobile application;
a) *Spinner* ***b)*** *Main Screen* ***c)*** *Map Screen having the red marker*

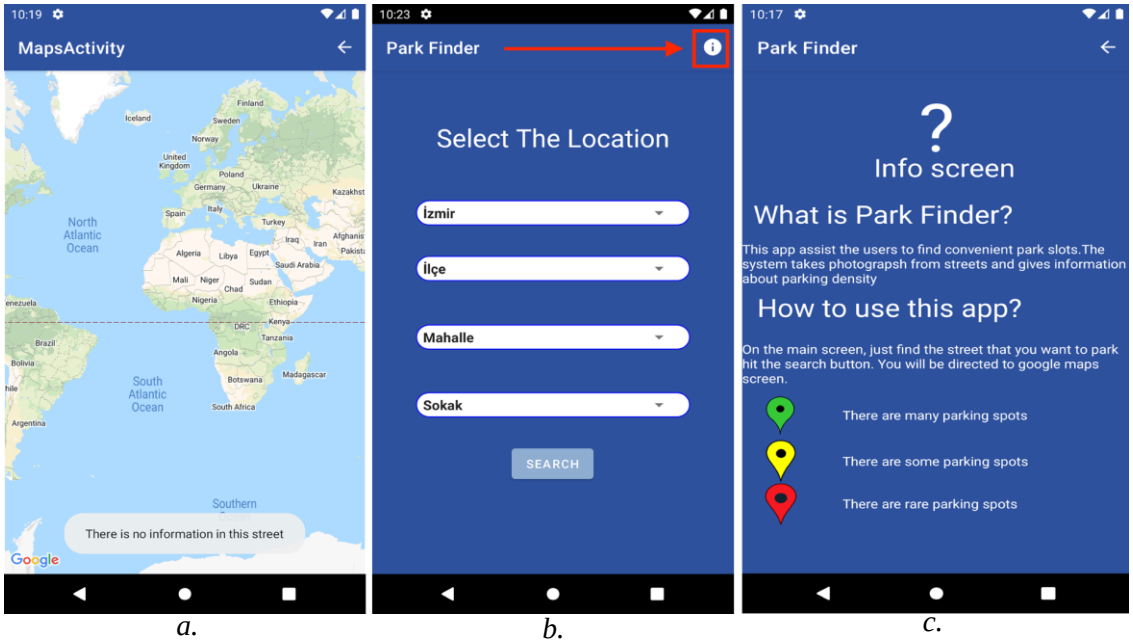
If the number of available parking areas is less than 50% and more than 20% of total parking areas, the marker turns to yellow. At the bottom of the maps, a snack bar displays the up-to-date number of parking available places in that street. If the number of available parking areas is more than 50% of total parking areas, the marker turns to green. In this example, the location is displayed, and the green marker indicates that there are many parking slots in this street. There is

a try-catch structure on spinner adapters. Not all the addresses are saved into the mobile application. Just a few of them are used as pilot addresses. For instance, there is no street information in Bergama district, Ahmetbeyler neighborhood, etc. So, a toast message is shown on the screen (Figure 20).



a. *b.* *c.*

Figure 20:
Test interfaces of the mobile application;
a) *Map Screen having the yellow marker* ***b)*** *Map Screen having the green marker* ***c.*** *Selection screen*



a. *b.* *c.*

Figure 21:
Test interfaces of the mobile application;
a) *Error message* ***b)*** *Direction to information screen* ***c)*** *Information screen*

There may be some cases where the address is fully saved into our application but there is no up-to-date parking information in the database. So, the map camera zooms out a general view of the world and the toast message gives a message about the error.

Users can tap address buttons, the search button, and the information button on the top. Information button directs the application to the information screen. The information screen gives information about what is the park finder and short manual (Figure 21).

Finally, when new images of the street arrive, all the previous records of that street in the database are deleted. Also, the data displayed in the mobile application are updated for every minute.

6. CONCLUSION

At the end of the project, the image processing module analyzes many photographs in a street and displays the convention of parking areas. Id, location, url, addressName, isParkingAvailable and date information of photographs were kept in Azure MSSQL database server. Image processing was handled with Python and run under a server provided by Dokuz Eylul University. The mobile application was developed in Android Studio, and it illustrates the information provided by the database. For the future, photographs and location information can be obtained from public transportation and test vehicles as real time information. The mobile application can be published in the Play Store for people to use it. A simple user interface was used, this may be a more user-friendly design in the future. There may be a score system for users. Users with a high score points can update the street parking density information by entering input in the mobile application. Because, who updates the information must be trustable users. Additionally, as future works, these can be said that in this study, the data-set was created by taking photographs manually. It is planned to obtain videos/photos from public transport. At this point, streets without public transport are a constraint. The next studies may address this issue. Also, a single location information is kept for the whole street. This can be diversified.

CONFLICT OF INTEREST

The authors confirm that there is no known conflict of interest or common interest with any institution/organization or person.

AUTHOR CONTRIBUTION

Oğuzhan Karakaya, Fikret Mert Gültekin, Mert Başkaya, Yunus Doğan: determining the concept and design process of the research and research management; data collection and analysis; data analysis and interpretation of results.

REFERENCES

1. Aqeel, A. (2018). Introduction to NodeMCU V3, accessed October, 11, 2022. Retrieved from https://www.theengineeringprojects.com/2018/10/introduction-to-nodemcuv3.html.

2. Balmiki, D., Singhal, M., Singh, A., & Tyagi, D. (2020) A Research on Smart Vehicle Parking System, *International Journal of Scientific Research and Management Studies (IJSRMS)*, 4(7), 124-127. doi:10.55248/gengpi.2022.3.7.15

3. Boyko, N., Basystiuk, O., & Shakhovska, N. (2018). Performance evaluation and comparison of software for face recognition, based on dlib and opencv library, *International Conference on Data Stream Mining & Processing (DSMP)*, Lviv, 478-482. doi:10.1109/DSMP.2018.8478556

4. Ch'ng, S. F. (2019). *Web-based car parking slot monitoring system (Doctoral dissertation, UTAR)*, Faculty of Engineering and Green Technology, Universiti Tunku Abdul Rahman, Malaysia.

5. Crisostomo, C. I. C., Malalis, R. V. C., Saysay, R. S., & Baldovino, R. G. (2019). A Multi-storey Garage Smart Parking System based on Image Processing, *International Conference on Robot Intelligence Technology and Applications (RiTA)*, Daejeon, 52-55. doi:10.1109/RITAPP.2019.8932899

6. Ghorpade, S. N., Zennaro, M., & Chaudhari, B. S. (2020). GWO model for optimal localization of IoT-enabled sensor nodes in smart parking systems. *IEEE Transactions on Intelligent Transportation Systems*, 22(2), 1217-1224. doi:10.1109/TITS.2020.2964604

7. Hanis, H. I. A., & Ramli, S. N. (2021). Attendance Monitoring Application for Students using RFID. *Applied Information Technology and Computer Science*, 2(2), 73-84. doi:10.30880/aitcs.2021.02.02.005

8. https://frightanic.com/iot/comparison-of-esp8266- nodemcu-development-boards/, accessed, september 30, 2022

9. https://github.com/AlexeyAB/darknet/, accessed, september 30, 2022.

10. https://kotlinlang.org/docs/comparison-to-java.html, accessed, october 30, 2022.

11. Jo, Y., Choi, J., & Jung, I. (2014). Traffic information acquisition system with ultrasonic sensors in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 10(5), 961073. doi:10.1155/2014/961073

12. Kemp, S. (2021, September 24). Digital 2021: the latest insights into the 'state of digital.' *WeAreSocialUSA*.

13. Morais, G. A. D., Fujiwara, E., Soares, M. C. P., & Lucimara, G. (2018). Development of an image processing MATLAB algorithm for cell growth analysis. *Revista dos Trabalhos de Iniciação Científica da UNICAMP*, (26). doi:10.20396/revpibic262018557

14. Nandyal, S., Sultana, S., & Anjum, S. (2017). Smart car parking system using arduino uno. *International Journal of Computer Applications*, 169(1), 13-18. doi:10.5120/ijca2017914425

15. Nisha, S. R., Shyamala, C., Pooja, S., Abarnia, A. P., & Shajeetha, M. S. (2020). RFID based smart car parking system using IoT and cloud. *Studia Rosenthaliana (Journal for the Study of Research)*, 12(5), 60-66.

16. Rahman, S., & Bhoumik, P. (2019). IoT based smart parking system. *International Journal of Advances in Computer and Electronics Engineering*, 4(1), 11-16.

17. Selvaraj, K., & Chakrapani, A. (2017). Smart dustbin monitoring system using LAN Server and Arduino. *International Journal of Advances in Computer and Electronics Engineering*, 2(4), 20-23.

18. Siswanto, S., Anif, M., Hayati, D. N., & Yuhefizar, Y. (2019). Pengamanan pintu ruangan menggunakan arduino mega 2560, mq-2, dht-11 berbasis android. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, 3(1), 66-72. doi:10.29207/resti.v3i1.797

19. Umesh, P. (2012). Image processing in python. *CSI Communications*, 23, 2.

20. Yusnita, R., Norbaya, F., & Basharuddin, N. (2012). Intelligent parking space detection system based on image processing. *International Journal of Innovation, Management and Technology*, 3(3), 232-235. doi:10.4236/wjet.2014.22006