Smart Parking System Based on Image Processing

Fatema H. Yusuf and Mohab A. Mangoud

Department of Electrical Engineering, University of Bahrain, Kingdom of Bahrain

Corresponding author: Fatema H. Yusuf (e-mail: 20173010@ stu.uob.edu.bh).

Abstract

This paper proposed a smart parking system that helps drivers in seeking out available parking slots based on image processing. With the increased number of vehicles which leads to the parking congestion, finding an empty parking spaces become a time-consuming task for many drivers specially during rush hours. From this comes the importance of implementing a novel camera-based system that facilities the parking issue to a huge level by detecting the available parking spaces in real-time. The proposed system counts the number of empty slots and detects the vehicles using a single webcam without the need of changing the parking infrastructure. A webcam is positioned in a high place that it can see all the parking slots in the parking area. An image of the parking area is taken as a reference and then all parking slots are selected. A webcam is used to record a video of the parking area while vehicles enter and exit. Both the image and the video are used to determine whether or not the parking space is available. The paper shows the implementation of the proposed system from scratch and then presents the results. The challenges of developing such a solution are also mentioned in addition to possible enhancements for future work.

Keywords: Smart Parking System, Parking Slots, Image Processing, Image Detection

1. Introduction

Nowadays, the number of drivers has been extremely increased and that leads to parking congestion specially in the parking areas while drivers are searching for available parking slots. According to a new study [1], drivers in New York City spend what is equivalent to 19 minutes daily trying to find a vacant parking slot. Parking congestion not only cause delays, but it also increases air pollution due to the hazardous vehicle emissions that negatively affect human health and the environment. Implementing an image processing-based smart parking system is a novel solution to this problem. It assists drivers in finding out vacant parking slots within a short period of time by marking the free spaces and counting them. Consequently, drivers will save time and money as the fuel consumption is reduced.

Few references found in the literature about detecting available parking slots. In [2], a proposed system was implemented to detect parking spaces with the use of image processing in Python. This system detects the available parking spaces by marking them with green outlines when a vehicle enters the parking area. If there is any occupied slot, the camera scans the vehicle's number plate which is then stored in cloud using Tesseract algorithm. Same procedure will be applied when the parking area is full, with only changing the outline color into red. Another camera is used when the vehicle is leaving the lot to scan the number plate. The parking time is then estimated and based on it, a bill is provided. Another proposed system [3] searches for a vacant parking spot depending of the driver's requirements, such as the first point, the desired location, and the distance between the parking slot and the location. If all the parking bays are occupied, the system will inform the driver. If not, then the system will look for the best available parking slot based on the driver's request and provide the parking information to the driver. If the driver is satisfied with the choice, then the reservation process is done, else, the process will be repeated. When the reservation is confirmed, the parking reservation is activated in the database, and the system will charge the parking fees for the first hour. The rest of the paper is divided into the following sections: Section 2 will present the proposed smart parking system and its methodology. Section 3 introduces the used software. Challenges are mentioned in Section 4. Finally, Section 5, contains the conclusion and future work.

2. Proposed Smart Parking System and Its Methodology

The figure shown below represents the block diagram of the system. The work is divided into two parts: selection and detection. In the selector part, the image of the parking area is used as a reference to select all the slots. Whereas in the detector part, live stream video is recorded from a high position with the use of a webcam. The video is segmented into frames and then each frame is exposed to several processes.

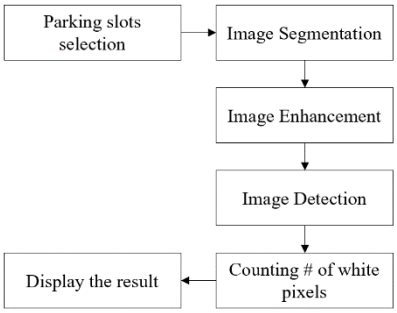

Fig. 1. Block diagram

2.1 Selection of Parking Slots

A picture of the parking area is taken from a high position and is used as a reference to select the parking slots. Then, it is supplied to Python software and saved into a variable after importing the needed libraries as shown in the code below.

```
import cv2
import numpy as np
img=cv2.imread('image0.jpeg')
```

Within a loop, the dimensions of the slots are obtained by trial-and-error and the slots are outlined in blue. The following code represents the selection of a single parking slot, and the same steps are repeated for the remaining slots, but only changing the dimensions. This process is illustrated in Fig. 2.

```
while True:
pts = np.array([[230, 530], [30, 670], [245, 670],
[380, 530]], np.int32)
pts = pts.reshape((-1, 1, 2)) cv2.polylines(img, [pts],
True, (255, 0, 0), 2)
#Repeat to select the remaining slots
```

Fig. 2. All parking slots are selected

Parking slot detection consists of three processes which are:

2.2.1 Image Segmentation

In this process, the image of the parking area is segmented into five images, each representing a parking space, and then all are converted to greyscale, as shown in the following figures.

Fig. 3. Cropped image

Fig. 4. Grayscale transformed image

2.2.2 Image Enhancement

To get rid of unnecessary noises, the image is blurred as shown below.

Fig. 5. Blurred image

2.2.3 Image Detection

The edge of the blurred image is detected as shown in the following figure. The number of white pixels that represent in the edge is counted and based on it, the status of the parking slot is determined and then the number of available spaces is displayed.

Fig. 6. Edge detection

3. System Software

The software used for this proposed system is Pycharm with the use of OpenCV library. PyCharm is an Integrated Development Environment (IDE) that has all the required programming functions and tools in Python language [4], including code editor, compiler, and debugger. As mentioned in [5], computer vision (CV) is a branch of computer science by which the computer can understand images and videos, the way they are stored, and how to get data from them. Whereas OpenCV shorts for Open Source Computer Vision Library, by which an enormous programming functions can be used. OpenCV plays an important role in artificial intelligence (AI) specially in the context of image processing. Recently, it has been widely used to detect objects in real-time.

4. Results

The proposed system has been tested in The Market Mall parking area in Bahrain and the results were satisfactory. The figure shown below represents one of the results in which one parking space is available and marked in green.

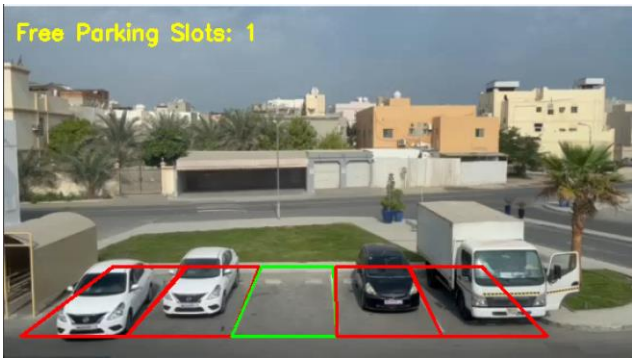

Fig. 7. Parking area with four busy slots and one available slot

Fig. 8 shows that even if a person is walking through the parking space, the status of that space is still considered available because the number of white pixels is not large enough. So, only if there is something as large as the vehicle then the system considers the slot to be occupied.

Fig. 8. A person walks through the parking slot

4. Challenges

In this project, two challenges were faced. The first challenge was the lack of experience in the Python programming language, but by searching and seeking help from experts, this challenge was overcome. The second challenge was the difficulty of taking an image of the parking area from the exact top, so it was not easy to determine the dimensions of the parking spaces. However, on the basis of trial-and-error, approximate dimensions were determined.

7. Conclusion and Future Work

With the increased number of vehicles and parking congestion, implementing the proposed system becomes crucial and beneficial for many drivers, especially during rush hours. The system can help drivers in saving their times, avoiding delays, and save money, as they will not need to fill their vehicles with fuel too frequently. Not only the drivers will benefit from the proposed system, but also the environment as the air pollution will be reduced due to the reduction of car emissions. Although the system is easy to implement, eco-friendly and cost-efficient, it can be further improved by building a mobile application on which the positions of the available slots are displayed besides the location of the parking area providing the shortest path to the nearest available parking slot. Moreover, using an algorithm to detect the available slots automatically rather than manually getting the coordinates of the slots can enhance the proposed system.

References

[1] A. Griffin. "Queens Drivers Have an Easier Time Finding Parking than Most NYC Drivers: Study." LIC Post. https://licpost.com/queens-drivers-have-an-easier-time-finding-parking-than-most-nyc-drivers (accessed Jan. 1, 2023).

[2] S. Bhatt et al., "Smart Parking System Using Image Processing," *IRJET*, vol. 7, no. 6, pp. 1951-1955,

Jun. 2020. [Online]. Available: https://www.academia.edu/44258255/IRJET_Smart_Parking_System_Using_Image_Processing

[3] A. M. Said, A. E. Kamal, and H. Afifi, “An intelligent parking sharing system for green and smart cities based IoT,” Computer Communications, vol. 172. Elsevier BV, pp. 10–18, Apr. 2021. doi: 10.1016/j.comcom.2021.02.017.

[4] “PyCharm: the Python IDE for Professional Developers by JetBrains.” JetBrains. https://www.jetbrains.com/pycharm/ (accessed December 28, 2022).

[5] R. Kulhary. "OpenCV - Overview." GeeksforGeeks. https://www.geeksforgeeks.org/opencv-overview/ (accessed December 28, 2022).

Appendices

Appendix A: Selector Part Code

```
import cv2
import numpy as np
img=cv2.imread('image0.jpeg')
while True:
    pts = np.array([[230, 530], [30, 670],
                                        [245, 670], [380, 530]],
                                    np.int32)
    pts = pts.reshape((-1, 1, 2))
    cv2.polylines(img, [pts], True, (255, 0, 0), 2)
                        # select the remaining parking slots ..
    pts = np.array([[380, 530], [245, 670],
                                        [465, 670], [525, 530]],
                                    np.int32)
    pts = pts.reshape((-1, 1, 2))
    cv2.polylines(img, [pts], True, (255, 0, 0), 2)
    pts = np.array([[525,530], [465,670],
                 [683, 670], [667, 530]],
                np.int32)
    pts = pts.reshape((-1, 1, 2))
    cv2.polylines(img, [pts], True, (255, 0, 0), 2)
    pts = np.array([[667, 530], [683, 670],
                 [900, 670], [810, 530]],
                np.int32)
    pts = pts.reshape((-1, 1, 2))
    cv2.polylines(img, [pts], True, (255, 0, 0), 2)
    pts = np.array([[810, 530], [900, 670],
                 [1070, 670], [960, 530]],
                np.int32)
    pts = pts.reshape((-1, 1, 2))
```

```
cv2.polylines(img, [pts], True, (255, 0, 0), 2) #lot#5
cv2.imshow("Image", img)
cv2.waitKey(1)
```

Appendix B: Detector Part Code

```
import cv2

import numpy as np

cap = cv2.VideoCapture(0)

img=cap.read()

while True:

    ret, frame = cap.read()

    cv2.imshow('frame', frame)

    if cv2.waitKey(1) == ord('q'):

        break

    free=0

    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT): #curent frames = total frames that are in the video

        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)                    #reset frames if they reach the max. frames that the video has (loop the video)

    success, img = cap.read()                                  # to get the frame from the video

    # select the parking slots

    lw, uw, x1, x2, y1, y2, x, w = 95, 65, 98, 15, 235, 280, 60, 80

    for f in range (5):

        pts = np.array([[x1,y1],[x2,y2],[x2+lw,y2],[x1+uw,y1]],np.int32)

        x1,x2 = x1+uw+2 , x2+lw

        pts = pts.reshape((-1, 1, 2))

        crop = img[y1:y2, x:x+w]                               #crop the image into the selected parts

        grayEdge = cv2.cvtColor(crop, cv2.COLOR_BGR2GRAY)          # convert to grayscale

        blur = cv2.GaussianBlur(grayEdge, (3, 3), 0)               # blur

        edges = cv2.Canny(blur, 100, 200)                      # canny edge detection

        pix = cv2.countNonZero(edges)                          # counting the white pixels

        x=x+w

        if pix in range(280, 1000):                            # draw a red rectangle if the parking slot is busy

            cv2.polylines(img, [pts], True, (0, 0, 255), 2)

        else:

            free += 1
```

```
            cv2.polylines(img, [pts], True, (0, 255, 0), 2)                 # draw a green rectangle if the parking slot is busy
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(img, 'Free Parking Slots: ' + str(free) , (10, 30), font, 0.65,             (0,255,255), 2)
        # Display the resulting frame
        cv2.imshow('video', img)
cap.release()
cv2.destroyAllWindows
```