Smart Parking System Based on Image Processing and Deep Learning

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 proposes a smart system that utilizes image processing and deep machine learning to address the parking issue for drivers. Finding an available parking slot has become increasingly difficult and time-consuming due to the growing number of vehicles, resulting in parking congestion. To alleviate this problem, the paper suggests implementing a novel camera-based system that can detect available parking spaces in real-time using deep machine learning techniques.

The proposed system is trained on the "COCO" dataset, an open-source dataset containing various objects, including different types of vehicles, using the YOLO v3 algorithm. By utilizing a webcam, the system can detect vehicles within the parking area and accurately count the number of available and occupied parking slots. The webcam is strategically positioned at a static and elevated location to capture the entire parking area. An initial image of the parking area is captured and used as a reference to identify all the parking slots.

During operation, the webcam records a real-time video of the parking area while the system detects vehicles within it. This enables the system to continuously update and provide an accurate count of free and occupied parking slots. The paper demonstrates the step-by-step implementation of the proposed system and presents the obtained results.

Overall, the paper highlights the effectiveness of the proposed system in addressing parking issues through the integration of image processing, deep machine learning, and real-time video analysis. It also emphasizes the potential for further enhancements and advancements in future research.

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

1. Introduction

Recently, there has been a significant increase in the number of vehicles, resulting in traffic congestion. One major contributor to traffic congestion is parking cruising. A recent study [1] shows that between 9% and 56% of traffic is caused by drivers searching for available parking spaces. Another study [2] conducted in New York City reveals that drivers spend an average of 19 minutes each day looking for parking spaces. This cruising for parking wastes valuable time, with an average of 6 minutes spent in the search. In addition to time wasted, this search for parking also has negative environmental impacts, as vehicles emit harmful emissions like CO2, which can adversely affect human health. To address this problem, an image processing-based smart parking system is proposed as a potential solution. This system aims to assist drivers in quickly locating available parking spaces by detecting vehicles in the parking area and counting the number of vacant slots. By implementing such a system, we can reduce traffic congestion, save time, and minimize fuel consumption, ultimately benefiting both individuals and the environment.

One of the proposed smart parking systems [3] utilizes the HSV color segmentation method to obtain the background image of the parking area. The image undergoes preprocessing, including median filters and background detection, followed by background subtraction during the detection process. Various image processing techniques such as blurring, thresholding, and filtering are applied to enhance the detection performance. The system focuses on the region of interest (ROI) corresponding to the parking spaces to detect objects. By comparing the ratio of white pixels in each ROI to the total number of pixels with a threshold value, the system classifies whether the parking space is occupied or not. In another proposed system [4], an artificial vision technique is employed to extract video footage of a specific parking area, which is then processed using MATLAB. Object-Oriented Programming is used to count the available parking slots and determine their status. The results are displayed visually on a website. Additionally, a different proposed system [5] improves upon the YOLO v3 algorithm to detect vehicles and parking spaces using three datasets with images captured in different weather conditions. The YOLO v3 algorithm is enhanced by adding a residual structure to extract features from images containing both parking spaces and vehicles. Four feature maps are employed in the detection process, enabling complex networks to extract additional features. 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 algorithm. Results are shown in Section 4. Finally, Section 5, contains the conclusion and future work.

2. Proposed Smart Parking System and Its Methodology

The figure below illustrates the block diagram of the system. The workflow starts with a selection process, where all parking slots are initially marked as free (represented by the color green). For the detection process, a live video stream of the parking area is captured using a webcam positioned at a high vantage point. The YOLO v3 algorithm, previously trained on the publicly available "COCO" dataset containing various classes, including different types of vehicles, is utilized to detect vehicles in the parking area. Each frame of the video undergoes the detection process.

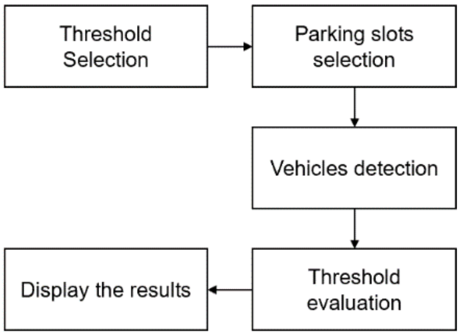

Fig. 1. Block Diagram

2.1 Threshold selection

The threshold value for the area of an occupied parking slot is set to 1600. Additionally, initial threshold values are set for the confidence, score, and Intersection over Union (IOU).

2.2 Selection of Parking Slots

Instead of manually specifying the coordinates for irregularly shaped parking slots, rectangular regions are chosen as representations of the slots. These rectangles are selected within the slots, although they may not align precisely with the exact borders of the slots. The coordinates of these rectangles are then supplied to Python software, and after importing the necessary libraries, they are saved into variables. In the visualization, all the parking slots are marked in green to indicate their initial status as free slots.

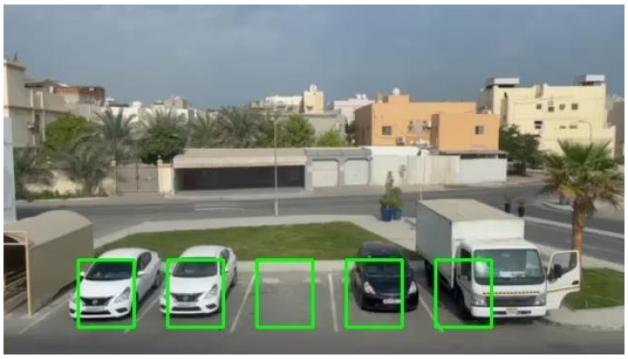

Fig. 2. Parking slots selection

2.3 Vehicle Detection

After loading the configuration and weight files of the pretrained YOLO v3 model, as well as the labels of the objects, a while loop is created to perform object detection on each frame of the video. A 4D blob array is generated to store and process the frames, which are converted into input images. The blob is then set as the input to the network, and all the layer names are retrieved.
Next, a for loop iterates over each output layer of the network. The object detection is refined by focusing only on specific types of vehicles such as cars, buses, trucks, motorbikes, and bicycles, while disregarding other objects. Within the loop, the class IDs, scores, and confidences of the detected objects are extracted. Only objects with a confidence equal to or above the defined threshold value are considered.
The bounding box coordinates, class IDs, and confidences are updated accordingly in case any disregarded objects are encountered. To avoid multiple bounding boxes around a single object, a non-maxima suppression technique is applied. Bounding box rectangles are then drawn around the detected vehicles labeled with the word “busy”, as illustrated in the provided figure.

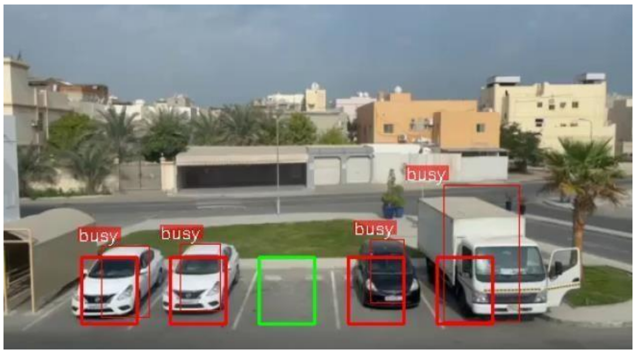

Fig. 3. Vehicles detection

A polygon is created for each bounding box using its coordinates and stored in a variable called "POLYGON". This polygon is then used to calculate the intersection area between each bounding box and each parking slot polygon. If the calculated area exceeds the threshold value for the occupied area, the slot is considered busy and marked in red, while also incrementing the busy counter. Otherwise, if the area is below the threshold, the slot's status remains free, which is the default. The number of free parking slots is determined by subtracting the number of busy slots from the total number of slots, which is set at the beginning.

3. YOLO Algorithm

YOLO stands for "You Only Look Once," and as the name suggests, this algorithm examines the input image only once to make predictions. It achieves this by utilizing a single neural network and performing a forward propagation pass. The network divides the input image into grids, with each grid representing a bounding box. The probabilities for each region serve as the weights for the bounding boxes. To prevent multiple bounding boxes from being drawn around the same object, a technique called "non-maxima suppression" is employed. YOLO consists of 24 convolution layers and two fully connected layers, all of which are organized based on their specific use [6].

4. Results

The proposed system was tested in The Market Mall parking area in Bahrain, and the results were satisfactory compared to other proposed systems described in [3]. However, it is worth noting that the compared system lacks the capability to detect vehicles in the parking area during nighttime. This limitation may be attributed to the dataset used, which does not include images of vehicles in various weather conditions. As a result, the compared system is unable to effectively detect vehicles at night. The figure below illustrates the test results of our model in a parking area with three occupied slots during nighttime.

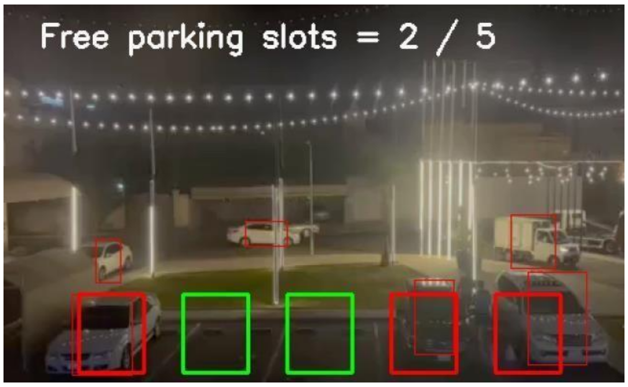

Fig. 4. Vehicles detection during nighttime

Additionally, I tested another smart parking system that determines the availability of parking spaces based on the number of white pixels in each slot. If a parking slot contains a sufficient number of white pixels, it indicates that the slot is occupied by a vehicle. Conversely, if the number of white pixels is low, the slot is considered free. This system performs well during the daytime. However, it exhibited poor performance at night, as depicted in the following figure.

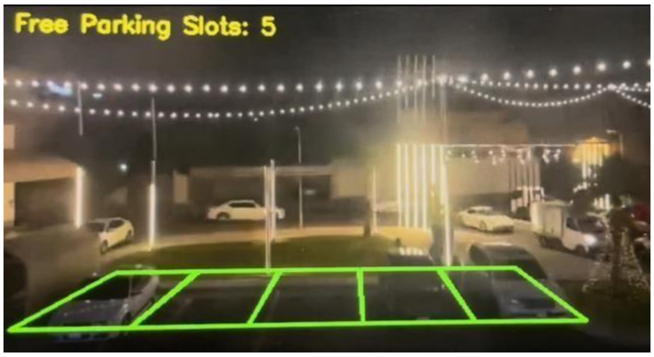

Fig. 5. Fail in detecting vehicles during nighttime

Another limitation of the proposed system, which relies on the number of white pixels, is that it may mistakenly classifya parking slot as occupied when people are passing through the area or when there is a rain spot present in the slot. This is because the image of the slot would have a significant number of white pixels, even if it does not actually contain a vehicle.

Although the confidence values obtained by our proposed system are relatively low in darker conditions, as observed in Fig. 6, the system performs well in detecting vehicles during both day and night, as indicated by the following figures.

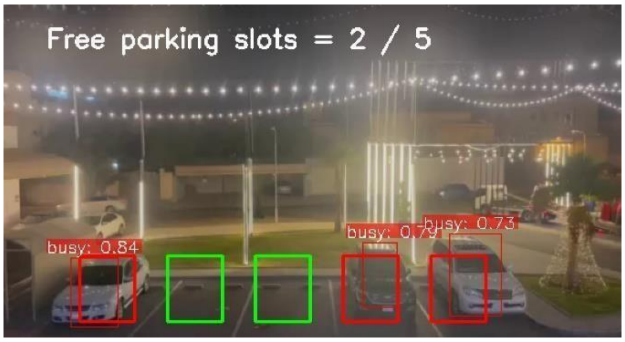

Fig. 6. Parking area with three busy slots

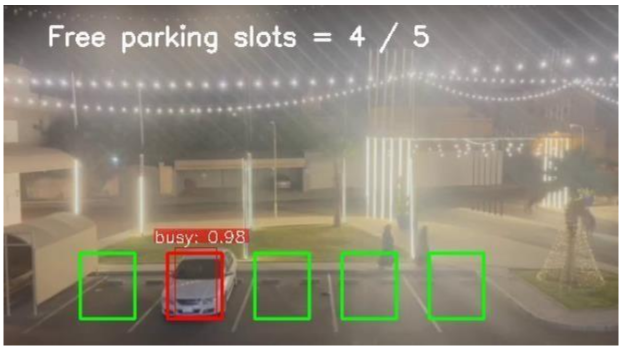

Fig. 7. Parking area with one busy slot

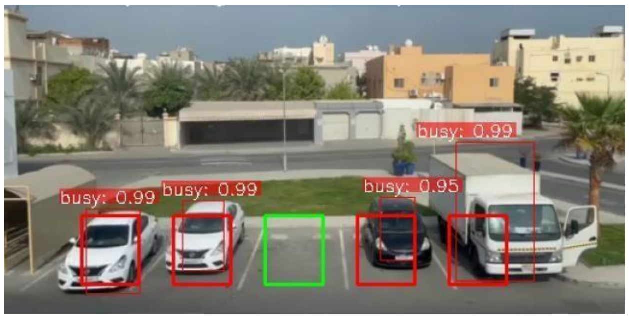

Fig. 8. Parking area with three busy slots

5. Conclusion and Future Work

Implementing the proposed system can pave the way for reducing parking cruising as the number of vehicles continues to increase. Drivers can benefit from the system by saving time, avoiding frustration while searching for parking spaces, and reducing fuel consumption. Additionally, the system contributes to protecting people from respiratory diseases by minimizing harmful emissions and promoting environmental conservation. While the proposed system successfully operates in different conditions and detects various types of vehicles, further improvements can be made. Integration of a user-friendly website to visualize the availability of parking slots and provide directions to the nearest open slot would enhance usability. Additionally, enhancing the system's confidence by incorporating more datasets with varied lighting conditions would be beneficial. Furthermore, automating the parking slot selection process would make the system scalable and more practical.

6. References

[1] Y. Zhu, X. Ye, J. Chen, X. Yan, and T. Wang, "Impact of Cruising for Parking on Travel Time of Traffic Flow," Sustainability, vol. 12, no. 8. MDPI AG, p. 3079, Apr. 12, 2020. doi: 10.3390/su12083079.

[2] 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 Jun. 5, 2023).

[3] A. H. Pratomo, W. Kaswidjanti, A. S. Nugroho, and S. Saifullah, "Parking detection system using background subtraction and HSV color segmentation," Bulletin of Electrical Engineering and Informatics, vol. 10, no. 6. Institute of Advanced Engineering and Science, pp. 3211–3219, Dec. 01, 2021. doi: 10.11591/eei.v10i6.3251.

[4] D. L. Gómez-Ruíz, D. Espejel-García, G. Ramírez-Alonso, V. V. Espejel-García, and A. Villalobos-Aragón, "Implementation of an Available Parking Space Detection System in Hectic Parking Lots," Journal of Transportation Technologies, vol. 11, no. 04. Scientific Research Publishing, Inc., pp. 688–701, 2021. doi: 10.4236/jtts.2021.114043.

[5] X. Ding and R. Yang, "Vehicle and Parking Space Detection Based on Improved YOLO Network Model," Journal of Physics: Conference Series, vol. 1325, no. 1. IOP Publishing, p. 012084, Oct. 01, 2019. doi: 10.1088/1742-6596/1325/1/012084.

[6] S. A. Shakhadri. "Implementation of yolov3: Simplified." Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/implementation-of-yolov3-simplified/ (accessed Jun. 5, 2023).

Appendix

```
#AIE604 PROJECT
import cv2
import numpy as np
from shapely.geometry import Polygon, box

total=5
CONFIDENCE = 0.1
SCORE_THRESHOLD = 0
IOU_THRESHOLD = 0.33
busy = 0
free = 0
threshold=1600

pts5 = np.array([[390, 230], [390, 290], [440, 290], [440, 230]], np.int32)
pts4 = np.array([[310, 230], [310, 290], [360, 290], [360, 230]], np.int32)
pts3 = np.array([[230, 230], [230, 290], [280, 290], [280, 230]], np.int32)
pts2 = np.array([[150, 230], [150, 290], [200, 290], [200, 230]], np.int32)
pts1 = np.array([[70, 230], [70, 290], [120, 290], [120, 230]], np.int32)

polygon1, polygon2, polygon3, polygon4, polygon5 = Polygon(pts1),
Polygon(pts2), Polygon(pts3), Polygon(pts4), Polygon(pts5)

config_path = r"C:\Users\falyu\PycharmProjects\pythonProject3\yolov3.cfg"
weights_path = r"C:\Users\falyu\PycharmProjects\pythonProject3\yolov3.weights"
labels =
open(r"C:\Users\falyu\PycharmProjects\pythonProject3\coco.names").read().strip
().split("\n")


net = cv2.dnn.readNetFromDarknet(config_path, weights_path)

video_path =
r"C:\Users\falyu\PycharmProjects\pythonProject3\video_at_night.mp4"
output_path = r"C:\Users\falyu\PycharmProjects\pythonProject3\outputVideo.mp4"
cap = cv2.VideoCapture(video_path)

#video properties
fps = cap.get(cv2.CAP_PROP_FPS)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

colors = np.random.randint(0, 255, size=(len(labels), 3), dtype="uint8")
car_class_id,truck_class_id, bicycle_class_id,motorbike_class_id ,bus_class_id
=2, 7, 1 ,3, 5

while True:
    ret, frame = cap.read()

    if not ret:
        break

    h, w = frame.shape[:2]

    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    ln = net.getLayerNames()
    try:
        ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
    except IndexError:
        ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]

    layer_outputs = net.forward(ln)

    font_scale = 0.5
    boxes, confidences, class_ids = [], [], []

    for output in layer_outputs:

        pts = pts1.reshape((-1, 1, 2))
        cv2.polylines(frame, [pts], True, (0, 255, 0), 2)  # lot#1
        pts = pts2.reshape((-1, 1, 2))
        cv2.polylines(frame, [pts], True, (0, 255, 0), 2)  # lot#2
        pts = pts3.reshape((-1, 1, 2))
        cv2.polylines(frame, [pts], True, (0, 255, 0), 2)  # lot#3
        pts = pts4.reshape((-1, 1, 2))
        cv2.polylines(frame, [pts], True, (0, 255, 0), 2)  # lot#4
        pts = pts5.reshape((-1, 1, 2))
        cv2.polylines(frame, [pts], True, (0, 255, 0), 2)  # lot#5


        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            if confidence > CONFIDENCE:
                box = detection[:4] * np.array([w, h, w, h])
                (centerX, centerY, width, height) = box.astype("int")

                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    idxs = cv2.dnn.NMSBoxes(boxes, confidences, SCORE_THRESHOLD,
IOU_THRESHOLD)

    if len(idxs) > 0:
        busy=0
        for i in idxs.flatten():
            x, y = boxes[i][0], boxes[i][1]
            w, h = boxes[i][2], boxes[i][3]

            if class_ids[i] == car_class_id or class_ids[i] == truck_class_id
or class_ids[i] == motorbike_class_id or class_ids[i] == bus_class_id or
class_ids[i] == bicycle_class_id:

                class_id = class_ids[i]
                if class_id < len(colors):
                    color = [int(c) for c in colors[class_id]]

                    scale_factor = 0.5
                    new_width = int(w * scale_factor)
                    new_x = int(x + (w - new_width) / 2)
                    x = new_x
                    w = new_width

                    cv2.rectangle(frame, (x, y), (x + w, y + h),
color=(0,0,245), thickness=1)
                    x, y, w, h = boxes[i]

                    POLYGON = Polygon([(x, y), (x, (y + h)), ((x + w), (y +
h)), ((x + w), y)])

                    intersection = polygon5.intersection(POLYGON)
                    if intersection.area > threshold:
                        busy += 1
                        pts = pts5.reshape((-1, 1, 2))
                        cv2.polylines(frame, [pts], True, (0, 0, 255), 2)  #
lot#5

                    intersection = polygon4.intersection(POLYGON)
                    if intersection.area > threshold:
                        busy += 1
                        pts = pts4.reshape((-1, 1, 2))
                        cv2.polylines(frame, [pts], True, (0, 0, 255), 2)  #
lot#4

                    intersection = polygon3.intersection(POLYGON)
                    if intersection.area > threshold:
                        busy += 1
                        pts = pts3.reshape((-1, 1, 2))
                        cv2.polylines(frame, [pts], True, (0, 0, 255), 2)  #
lot 3

                    intersection = polygon2.intersection(POLYGON)
                    if intersection.area > threshold:
                        busy += 1
                        pts = pts2.reshape((-1, 1, 2))
                        cv2.polylines(frame, [pts], True, (0, 0, 255), 2)  #
lot2

                    intersection = polygon1.intersection(POLYGON)
                    if intersection.area > threshold:
                        busy += 1
                        pts = pts1.reshape((-1, 1, 2))
                        cv2.polylines(frame, [pts], True, (0, 0, 255), 2)  #
lot1

                    text = "busy: {:.2f}".format(confidences[i])
                    cv2.putText(frame, text, (x, y - 5),
cv2.FONT_HERSHEY_SIMPLEX, fontScale=font_scale, color=(255, 255, 255),
thickness=1)

                free=total-busy
    cv2.putText(frame, "Free parking slots = "+str(free)+" / "+str(total),
(40, 40), cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(250, 250, 250),
thickness=2)
    out.write(frame)

    cv2.imshow("Frame", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
```

```
out.release()
cv2.destroyAllWindows(
```