Prototype of a parking system with path recommendation

Tuan Linh Dang
linhdt@soict.hust.edu.vn
Hanoi University of Science and Technology
HaNoi, VietNam

Tran Sy Dat
dat.ts183885@sis.hust.edu.vn
Hanoi University of Science and Technology
HaNoi, VietNam

Thuy Ha Hoang
ha.ht184251@sis.hust.edu.vn
Hanoi University of Science and Technology
HaNoi, VietNam

Trong Nghia Nguyen
nghia.nt184297@sis.hust.edu.vn
Hanoi University of Science and Technology
HaNoi, VietNam

Tuan Minh Vu
minh.vt194334@sis.hust.edu.vn
Hanoi University of Science and Technology
HaNoi, VietNam

ABSTRACT

With the growth of population and vehicles, parking is a significant problem people face in modern life, especially in developed countries. It is often time-consuming and inconvenient for the driver to find a parking space in the parking lot, especially during rush hour. Therefore, this paper suggested a parking prototype that uses the YOLOv5s algorithm to detect vehicles and empty slots, and the Deep SORT algorithm will track the detected cars in moving frames. In addition, this prototype may use the BFS algorithm to recommend the path to the nearest available parking space for a user through a developed user interface. Our experimental results with different video situations showed that the proposed prototype achieved 0.906 mAP for average accuracy and an operating speed of 18 FPS for ten vehicles using Nvidia GTX 1080 GPU in the demo application.

CCS CONCEPTS

• **Computing methodologies → Object detection and tracking**; **Supervised learning**; • **Computer systems organization → Neural networks**; **Realtime system**.

KEYWORDS

Object detection, Object tracking, Parking system, Path recommendation, YOLO, DeepSORT, BFS

ACM Reference Format:
Tuan Linh Dang, Tran Sy Dat, Thuy Ha Hoang, Trong Nghia Nguyen, and Tuan Minh Vu. 2022. Prototype of a parking system with path recommendation. In *The 11th International Symposium on Information and Communication Technology (SoICT 2022), December 1–3, 2022, Hanoi, Vietnam.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3568562.3568611

1 INTRODUCTION

Object tracking is a significant subdomain of computer vision that works with the video stream of cameras. Object tracking aims to identify the target object's position in the video frame. Object tracking has recently been used widely in some domains, primarily vehicle surveillance. The smart parking system (SPS) is widespread in this domain.

SPS is the system applying technologies to manage moving vehicles in the parking lot. The traditional SPS often uses sensors in each parking space to detect vehicles. Next, this sensor will send the data to the management system. If that parking is taken by a car, the management system will store all that vehicle's information, such as time and license plate number. Nowadays, with the increase in population and vehicles, the solution to parking issues has become more critical. Therefore, SPS was an absolute necessity. SPS aims to save users fuel and time and avoid vehicle congestion. With SPS, drivers can effortlessly find parking spaces in the parking lot.

For the last few years, many SPS have been proposed. A widely used solution is to use the Internet of Things. A famous study for this approach is IoT-based SPS [19]. However, this method requires a large number of sensors which is facing difficulties in finance and human resources. Besides, some other proposals suggest designing smart parking systems based on cameras with different views or sensors and some deep learning algorithms. These systems [12, 15] will notify the user about available spaces in the parking lot. Nevertheless, the similarities of these above systems can not suggest the path from the user's location to these spaces. In practice, the most common parking management systems, especially in basements of shopping malls or apartment buildings, only have the function of calculating the number of available spaces. A particular method is Basement Parking System [4]. This automatic system uses sensors in each parking space, then sends information about available parking spaces via the led board. Although there may have some direction boards or traffic signs, the driver can not know the exact location of the empty slots by visualization, causing difficulties if the parking lot is too large.

The previous studies suffer two main issues: they take a lot of time and effort and can not recommend a path to the parking slot. To deal with these problems, the goal of our proposed prototype is to help the driver finds the shortest path to the free slot with few resources. There will be three main phases in our SPS. In the first phase, our proposed prototype will use object detection algorithms to identify the location of parking spaces and moving vehicles in the parking lot. Next, we use object-tracking algorithms to track vehicles from the entrance. These state-of-the-art deep learning algorithms above can give good results in bad conditions and complex backgrounds. We not only avoid using complex sensors for each slot but also balance speed and accuracy properties. Then, our shortest path algorithm will suggest the optimal path to the parking space for users. This approach may solve the drawbacks of the previous study. In addition, our proposed prototype also allows the administrator to modify the parameters of the parking lot. The data transformation between users and the system is processed via socket.

Another problem encountered when researching this problem is data scarcity. The datasets dedicated to training the empty slot detection and car detection with the top-view in the parking lot to cover the whole context of the parking lot are few or almost none published by the other authors. Therefore, it is necessary to collect a dataset that meets this problem's requirements and be made public so that further studies can also use it.

Our manuscript has three main contributions. First, we collected data for the training object detection model in the parking lot. In addition, we built a complete parking prototype that suggests the path to the nearest empty slot for users and provided a demo application for the users.

This paper is presented as follows. Section 2 focuses on related works and algorithms of object detection, object tracking, and finding the shortest path. Section 3 demonstrates our proposed parking application and each block's function. Our experimental results are shown in section 4. Finally, section 5 is the conclusion of our manuscript.

2 RELATED WORK

2.1 Existing Smart Parking Systems

The population is increasing at an alarming rate, and the need for transportation is also rising rapidly. That leads to the growth of vehicles, but the number of parking slots is limited. Therefore, creative solutions for the parking system are necessary. Many studies are being introduced to handle this problem.

Previous researches used Radio Frequency Identification (RFID) to manage parking space [21]. RFID was frequently utilized in previous studies for parking management [4]. RFID is the non-contact wireless use of Radio Frequency Electromagnetic Fields to identify and track tags affixed to things automatically. RFID technology uses RFID readers, RFID labels, computers, barriers, and software as its primary components. The program has been implemented for the management, control, transaction reporting, and operation of parking lots throughout the city. RFID readers, labels, and obstacles will govern parking lot check-ins and check-outs. This technology is used to control the entry and exit of cars, but it does not provide information on the number of empty slots available.

For extracting more information about a vehicle, utilizing Internet of Things (IoT)-integrated parking spaces is one of the most common methods. Smart parking management based on IoT contains parking sensors, processing units, mobile applications, and the cloud. The sensors, namely ultrasonic sensors, are used to determine the vacancy of parking slots. These sensors are connected wirelessly to the Raspberry Pi through an ESP8266 chip. The processing unit consists of a Raspberry Pi, an intermediate between sensors and the cloud. Data collected from sensors is sent to the Raspberry Pi via an ESP8266 chip. After that, the Raspberry Pi sends this data to the IBM MQTT Server. Besides, the mobile application is an interface for end users to interact with the systems. This application connects to the IBM MQTT Server and provides information on parking spaces [19]. The limitation of this system is the large number of sensors required to detect an object. It leads to difficulty in deploying and expanding the parking system.

To reduce the number of sensors, a system based on computer vision consists of two phases: detecting the available parking space and notifying users of this status in real-time [15]. In the parking slot detection period, the system also uses many sensors or cameras to spot parking slot occupancy. The data for this status is then sent to the user via the notification system. This system supports real-time monitoring and visualization of the parking spaces; however, it does not navigate the users to the free parking slots. In cases where the parking space is large, users could struggle to find the shortest path. In another study, a distributed network of cameras, plenty of edge devices, and a deep learning model are used to monitor the parking space [12]. For more detail, some cameras with zoom lenses and motorized heads capture the license plate numbers and track the vehicles entering or leaving the parking lot. Other cameras with a wide-angle fish-eye lens detect occupied parking spaces. Deep learning models are used to determine available parking slots based on parked slots. Nevertheless, this system does not suggest the shortest route to the available parking slot. Furthermore, the deep learning system necessitates many edge devices and cameras, making deployment difficult.

To design a suitable parking prototype, we propose to monitor the parking spaces through a camera, mobile application, and deep learning model combined with finding the shortest path algorithm.

2.2 Theoretical background

2.2.1 Object detection. One of the most popular object detection algorithms is you only look once (YOLO). YOLO is an algorithm that uses a convolutional neural network (CNN) to detect objects in real-time [18]. The appearance of YOLO has improved accuracy and processing speed in comparison with other state-of-the-art algorithms. One strength of the YOLO algorithm is to detect an object in one phase, divided into two parts: finding the object's location and assigning a corresponding class to that detected object.

For the operation, the YOLO model takes the image pixel as input and divides it into $S \times S$ grid of cells. The value of S affects the performance of the model. Each cell contains information about only one object by predicting M bounding boxes for that object. A bounding box is a rectangle that captures an object in an image with five elements: (x,y,w,h, confidence), where x and y are the center coordinates of the bounding box, and w and h are the width and height of the bounding box, respectively. The confidence is the confidence score for each bounding box. It is based on conditional class probabilities, which the cell predicts.

YOLO has many versions, and the previous version updates the later version. For example, the fifth version, YOLOv5, was developed by the architecture of YOLOv4. In practice, the YOLOv5 is proposed with different sizes. In details, they are extra-large (v5x), large (v5l), middle (v5m) and small (v5s). These models keep the based architecture of YOLOv5, and among them, YOLOv5s [25] gives the highest speed. This model only contains 27 layers, 7.2 million parameters, and the size of the training weight is about 15 megabytes accounting for 8% of the largest model.

From the above background of ideas and architectures, our proposal did use YOLOv5s to detect objects in one frame. In detail, they are empty slots and vehicles in the parking lot.

2.2.2 Object tracking. The object tracking algorithms aim to follow the moving object in each frame in the video stream. There will be two phases in the tracking operation: the detection phase and the tracking phase. The first phase is object detection which finds the location of object instances in an image frame. Then, the tracking operation will re-identify that object until the end of the video stream.

Simple Online and Real-time Tracking (SORT) is a well-known multiple-object tracking approach. SORT algorithm uses the Kalman filter to predict the new position of object bounding boxes [8]. However, the biggest problem of SORT is the high number of identity switches. This phenomenon often happens with occlusion and different viewpoints, which causes a decrease in the efficiency of the SORT algorithm. Deep SORT is an extension of the SORT algorithm. The authors of the Deep SORT algorithm use the Kalman Filter, the Hungarian algorithm, and new distance metrics to track the objects. The operation of the Deep SORT algorithm can be summarized as follows [22]. The combination of the YOLO algorithm for object detection and Deep SORT for object tracking has created many potential opportunities for research, such as Vehicle Analysis System [16], People Tracking System [9], and Vehicle Detection and Tracking [10]. This combination is widely used because its robustness and implementation are simple.

2.2.3 Finding shortest path. The single-source shortest path problem is defined as finding the shortest paths in a given graph $G(V, E)$ from a given node v to other nodes. Some basic search algorithms are commonly used, such as breadth-first search, Dijkstra's algorithm, the Bellman-Ford algorithm, the A* algorithm, the Floyd-Warshall [14]

Dijkstra's algorithm is the same as A* when the heuristic function equals zero for every case. In addition, if all edge weights are equal to or greater than zero, it is a specialized and optimized version of Bellman-Ford. Furthermore, breadth-first search is a specialized and optimized version of Dijkstra's algorithm when all edge costs are uniform.

3 PROPOSAL SMART PARKING APPLICATION

An overview of our prototype can be seen in Fig. 1. The proposed prototype includes three main blocks are *Image acquisition block*, *Processing center block*, and *Application block*. Image acquisition block (IAB) captures the parking lot images as input for later blocks. The processing center block (PCB) receives images from IAB as its input. After that, PCB processes images to output the path direction images for the tracked cars. This process consists of three stages: object detection, object tracking, and shortest path finding. Each corresponding stage uses a different algorithm. Specifically, the object detection stage uses YOLOv5s [3] to detect cars and empty slots in the parking lot. The object tracking stage uses Deep SORT [22] to track cars that are being suggested paths. And shortest path-finding stage uses BFS algorithm [5] to find paths to the nearest empty slots of those cars. In the end, the application block (AP) receives output images of the PCB and sends them to user applications. The details of each block are depicted in the three following sections.

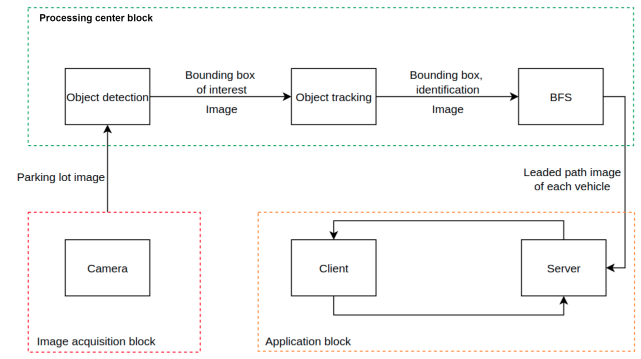

Figure 1: An overview of our proposed prototype

3.1 Image acquisition block

IAB is the block in which a camera is used to capture the parking lot. The camera is placed on the top of the parking lot to ensure that it can capture the entire scene. These captured images are then used as inputs for later blocks.

3.2 Processing center block

PCB consists of three stages: object detection, object tracking, and shortest path finding. Each step is explicitly described below.

Object detection - The task of this block is to detect vehicles and empty locations in the parking lot. These detections are essential in our proposed prototype because it has to determine the location and the number of remaining empty slots and the location of the cars that have just entered the gate to make input for the shortest path-finding algorithm. Besides, the number of empty slots can also be used to display the status information of the parking lot. This helps customers who want to use the parking services consider whether to go to the parking lot. In addition, to limit the false detection of empty locations outside the parking area, we have also developed a tool to determine the coordinates of the parking areas, the moving areas, and the gate areas. The details of this tool can be seen in the section 4.3. After obtaining the coordinates of the parking areas, the system can quickly check for an empty slot whether it belongs to those areas. Hence, our system can produce the exact quantity of the remaining open slots. In this research, we used the YOLOv5s algorithm to detect the empty slots and the cars in the parking lot. YOLOv5s is the most lightweight version of

YOLOv5 that has been released. Because of the small number of layers and parameters, this model can quickly handle the input to give the corresponding output results. However, along with the higher increase in the processing speed, the accuracy of this model is partially reduced. In short, the reasons for using YOLOv5s in this study instead of other versions can be listed as follows. Firstly, YOLOv5s is a lightweight version of YOLOv5. Therefore, the computation complexity required for the training and prediction of the model is relatively small. Secondly, despite the trade-off between speed and accuracy compared to other versions, YOLOv5s still ensures good prediction performance. Among the achieved results provided by the author-published YOLOv5, YOLOv5s achieved 0.45 (AP) on COCO dataset [20] with inference time only 6.4 (ms) for an image on GPU V100. In contrast, the more complicated YOLOv4 version gives the accuracy quite equivalent. The outputs of this stage are the bounding box coordinates, labels, and probability scores of all the objects that belong to two classes, consisting of vehicles and empty slots. Our system can display this information to the user application from the total number of labels that belong to the empty slot class.

Object tracking - Object tracking is an essential task in our proposed method. It is necessary to track the car when suggesting the shortest path because this helps to determine which path belongs to which vehicle. SORT and Deep SORT are two tracking algorithms that have recently received much attention from researchers. SORT is the simpler and faster version that uses the IoU metric to associate the predicted tracks and the detections with determining the identity of each object. On the other hand, Deep SORT is the algorithm that simultaneously uses three metrics IoU, Mahalanobis distance, and cosine appearance distance. This manuscript used Deep SORT [22] to track cars in the parking lot. The use of Deep SORT instead of more lightweight algorithms can be explained as follows. Firstly, when moving in the parking lot, the cars can overlap in the frame; if only using the IoU metric as the SORT algorithm to associate the tracks and the detections, this can lead to more identity confusion. In addition, if both bounding boxes contain the same object, their feature vectors will be more similar. Deep SORT uses a deep learning model to extract features of all objects and uses those features to calculate the cosine distance between tracks and detection for the association. This makes the association more precise when objects are obscured or entirely out of the frame. Several architectures were proposed to extract features of the objects in Deep SORT. In this study, we used OSNet, a lightweight model, to complete this mission. OSNet was built with multiple streams. Each stream has several convolution layers with different kernel sizes to learn features from various proportions. Secondly, despite the greater computation complexity than SORT, Deep SORT still achieves a fast enough inference time in case the hardware requirements are met.

However, not all cars in the parking lot need to suggest paths, such as the cars in the parking area. The system only needs to suggest paths for cars that have just entered the gate and are in the moving area. Eliminating the number of vehicles to track also increases the speed of the object-tracking algorithm. Therefore, it is necessary only to follow any car that has just entered the gate and any vehicle in the moving area. To check these conditions, after obtaining the detections of vehicles in the parking lot using YOLOv5s in the object detection stage, we still used the tool 4.3 mentioned above. In addition, to overcome the context of a car standing in the moving area for too long which affects the suggestion of paths for the others, a time threshold is also set to eliminate the suggestion for these vehicles.

Shortest path finding algorithm - After tracking the new vehicles that have entered the gate, our prototype will apply the shortest path-finding problems. This problem is often modeled in graphs with vertexes being destinations and edges being the path between them, and then finding a path for each vehicle by finding a path on the graph. We found that the BFS algorithm is an algorithm that meets the requirements set out for the quick time. Besides, the graph which we used in this stage is a graph without weights; the BFS algorithm is more useful than the others.

First, to apply the BFS algorithm, each frame is divided into $n \times m$ grids along the length and width so that each empty slot has at least one grid. Illustrative examples can be seen in Fig. 2. The problem is to find the shortest path from one source to multiple targets, called the B1 problem. Hence, the suggestion for N cars simultaneously becomes the application of B1 with n times consecutively. Another issue that needs to be concerned is that in an actual parking lot, cars can not go through all the areas in the parking lot; the proposed algorithm also has to solve this problem.

Problem B1 can be stated as follows. The frame captures the parking lot and is divided into $m \times n$ grids. Each grid is assigned a value in the set -1, 0, 1 that corresponds to the meaning of impassable, passable, and empty slot. Each source car is marked by a grid in which the car's center belongs to that grid and has a value from two onward. The problem is to find the shortest path from these sources to the destinations so that each destination is only suggested for one source. The order of precedence of suggestions for those sources corresponds to the order of the values assigned to them. From the coordinates of the detected empty slots, including the upper left and bottom right corner coordinates, the system can determine the grids within those areas; those grids are numbered 1 to imply the targets. Similarly, the parking area grids are numbered -1, and the moving area grids are numbered 0. After that, the BFS algorithm is applied to each vehicle. When it comes to the first 1 grid position, the algorithm immediately locks this empty area by filling the value -1 to prevent the suggestion of this empty slot for the others. The suggestion is made the same for all the remaining vehicles. Besides, the algorithm also makes it easy to add path directions to an empty slot pre-selected by the user. In this case, the grids in the selected slot will be marked and locked; the algorithm will implement from the grid that contains the source until the first marked grid of that slot is encountered.

During the process, the algorithm also carried out the trace to trace the path when the first empty slot was found, whereby the algorithm used an array named *Trace* to save the path for each vehicle. Specifically, let (x_{t-1}, y_{t-1}) is the coordinate of the preceding grid according to BFS, the trace to the current grid (x_t, y_t) is stored as follows:

$$Trace[x_t][y_t] = x_{t-1} * m + y_{t-1} \tag{1}$$

Because this study used C++ programming language to implement BFS algorithm, converting from (x_{t-1}, y_{t-1}) to $x_{t-1} * m + y_{t-1}$ helps optimize memory. However, the rest of the system is installed in Python language, so we have used the Pybind11 library [7] to embed the C++ code into the processing stream. Algorithm 1 outlines this shortest path-finding algorithm.

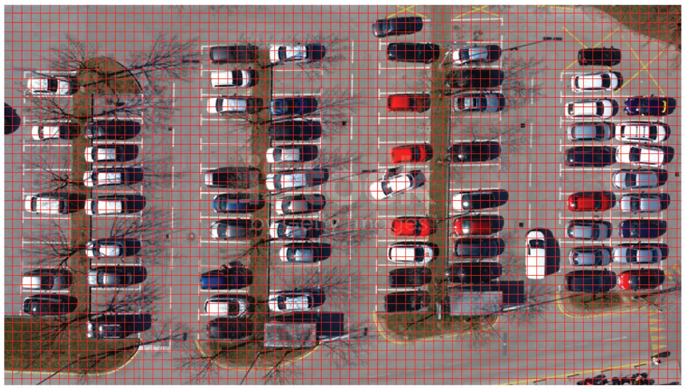
Figure 2: Example of grids on the frame

3.3 Application block

The purpose of building the application block is to provide a demo that is a suitable interface on both web and mobile platforms, supporting displaying the current status of the parking lot and the path to the nearest empty slot over time for new cars to enter the parking lot. The shortest path results of each vehicle returned by the AI system will be sent to those cars separately. Another function supported by this demo app is calculating parking fees when the cars leave the parking lot. This function is implemented based on logging each vehicle's entry and exit times in the database. The application block can be divided into two main components: the client and the server. The client was built using the ReactJS framework by javascript language to design the user interface. Data from the server is sent and received by the client through the socket. The server was built using the NodeJS framework and included many APIs that help the client call and receive data. However, a problem when using the app with the above method is that when two or more people open the app at the same time when a vehicle has just entered the gate, the system cannot determine which app belongs to the user who has entered. Our solution, in this case, is to send a default image showing all the suggested paths of all vehicles at the initial time when opening the application until the system returns the proposed image of each car.

When users enter the gate of the parking lot, if they want to get information about the parking lot and see directions to the nearest empty slot, they should turn on the apps that the system can associate their apps with their corresponding ID to send and receive results during their parking processes. When leaving the parking lot, if users want to consult the fee they have to pay, they can choose the payment function in their apps, and the cost will be displayed on the screen.

Algorithm 1: Shortest Path Finding (BFS)

```
Input: n tracking sources, their corresponding coordinates,
       and showed the path. List of TARGET that are
       available slots and their corresponding coordinates.
Output: paths
1  $queue < int > q$
2  $vector < vector < int >> paths$
3  for $isource \in n\_sources$ do
4      $q.push(cur\_coor\_isource)$
5      $target \leftarrow -1$
6      while $q \neq empty$ do
7          $stop \leftarrow$ False
           /* Position of the current source */
8          $cur\_coor\_isource \leftarrow q.front()$
9          $q.pop()$
           /* iterate all around directions that can go through */
10         for $coor\_dir \in n\_direction$ do
11             $next\_coor \leftarrow cur\_coor\_isource + coor\_dir$
               /* if the first target is found */
12             if $next\_coor = TARGET$ then
13                 $Trace[next\_coor] \leftarrow cur\_coor\_isource$
14                 $target \leftarrow next\_coor$
15                 $stop \leftarrow$ True
16                 break
               /* Push next found coordinate in the queue */
17             $q.push(next\_coor)$
               /* Store the trace to current coordinate */
18             $Trace[next\_coor] \leftarrow coor_dir$
19         end
20         if $stop$ then
21             $break$
22     end
23     $q \leftarrow empty$
       /* Trace the path of isource */
24     $vector < int > path$
25     while $target \neq -1$ do
26         $path.push\_back(target)$
27         $target \leftarrow Trace[target]$
28     end
       /* list paths of all the tracking cars */
29     $paths.push\_back(path)$
30 end
31 return $paths$
```

4 EXPERIMENTS

4.1 Dataset

To our knowledge, the number of datasets for empty slot detection in the parking lots from the top view is minimal. In the past, some authors proposed a dataset [23] which has the top view of slots, but they are zoomed in very close without bounding box labels, so it is hard to use in the detection task. The CARPK dataset [17] is also a top-view dataset but does not contain empty slot labels. The

PK-LOT dataset [13] is a dataset that meets the requirements of this research with about 12416 images collected from two parking lots from three different angles. All the images in this dataset were assigned empty and occupied slot notation. Therefore, we used this dataset for training the empty slot detection. However, it can be seen that although the number of images in this data set is quite large, the diversity of the background and surrounding context is not too abundant, with only two parking lots.

There are currently several published datasets for car detection data, but not all are suitable for object detection. For example, a dataset [24] was published by Linjie Yang, and his partners only provide images with close-ups of cars and do not contain bounding box labels, so it cannot be used in this task. Two other datasets [2], [1] supply bounding box labels, but the capture angle of all images is low compared to the cars.

Therefore, this study decided to use the PK-LOT dataset mentioned above with additional vehicle bounding box labels and collect more data from external parking lots on the Istock website [6] to create a more complete and diverse parking dataset of three available class labels including empty slot, occupied slots, and car, respectively 0, 1, and 2. Specifically, we only randomly took out 3725 images from the PK-LOT dataset to avoid data imbalance. Besides, we collected videos of 30 other parking lots on the IStock website with abundant backgrounds, different light conditions, and multiple camera angles. After that, each video is cut into frames to label the bounding box of objects and their corresponding class. This part of the data also contains three available labels, as mentioned above. For self-collected data, we used several mobile cameras with the same resolution of 12 megapixels to capture top-view videos from some parking lots and then processed them as videos collected from IStock. The final dataset has 4855 images of 32 different parking lots with three available classes. In this research, we only used two class labels, including empty slot and car, to train the object detection model.

4.2 Experimental setup

YOLOv5s - The object detection algorithm was implemented in the Pytorch framework. YOLOv5s was trained for 300 epochs. Input images were resize to 640×640. We adopted a batch size of 64 in the training stage. The optimizer used was Stochastic Gradient Descent [11] with an initial learning rate of 0.01 and a weight decay of 5×10^{-4}. An IOU threshold of 0.5 was set for the NMS algorithm to remove multiple overlapping bounding boxes of the same object. The above parameters are suggested by default from the original author [3].

Deep SORT - Because of the difficulties in collecting and characteristics of the car re-identification dataset, this study directly used the pre-trained supplied by the author who proposed the OS-Net for feature extraction task in Deep SORT without any further modification. Input images were resized to 256×128. In addition, as mentioned in section 3.2, a threshold time of 300 seconds was set to eliminate tracking of the vehicles standing in the moving area for too long. This threshold can be easily modified by the administrators.

Shortest path finding algorithm - The BFS algorithm was implemented in the C++ programming language. According to our experimental data, because all the parking lots in the data have a number of slots in width or length less than 40, we divided each frame into 40×40 grids. In fact, this division can be changed arbitrarily based on the condition of each parking lot. In order to evaluate this algorithm more thoroughly, our research conducted several experiments with changing parameters, such as the number of grids for each frame and the number of suggested cars each time. These results are presented in the section 4.3.

4.3 Experimental results

Coordinates determination tool - As mentioned in section 3.2, not all objects need to be tracked; the application will only focus on new vehicles that have entered the gate to speed up the inference process. To solve this problem, this research built a tool by the PyQt5 framework to help administrators easily determine the coordinates of the areas, including the parking area, the moving area, and the gate area. The application can easily determine which objects need to be tracked from those coordinates. Fig.3 visualize the user interface of this tool.

Our coordinates determination tool supports selecting coordinates from both videos and images corresponding to the function load video and load image. After clicking on the above function, the image of the parking lot appears in the dialog box; because the camera position is fixed, the frames have the same observation angle from the camera, so the selection function will randomly select one image in the video to display. This image is then used to determine the required coordinates directly. The functions (Gate Area 1, Gate Area 2), Travel Area, and Parking Area are the functions respectively to select the coordinates of the gate, the moving area, and the parking area. The administrator will use the mouse to click on the pixels on the image so that joining them in the order of that click will form the smallest contour around the area to be selected. The administrator can choose many different areas of the gate, parking, and moving in the same parking lot without limit on the number. These coordinates will be stored for use when the system reboots without any further intervention from the administrator. When selecting the coordinates of a gate, the administrator selects *Gate Area 1* first and then selects *Gate Area 2*; these two areas should be adjacent to each other and, in that order form the direction to enter the parking lot from the gate, this is to help the system can confirm the vehicle entering or leaving the parking lot. To determine which vehicle has just entered the gate or check whether the vehicle has entered the parking area or whether the empty slot is in the parking area, or whether the vehicle is in the moving area, the system checks whether the center of that object belongs to the polygon of the area under consideration. In this case, the polygon is created by joining all the points that the administrator selected about that area in order. For convenience, this study used the *Matplotlib.Polygon* library to check if a point is within a polygon. Finally, after clicking the coordinates of any area in the parking lot frame, the administrator can click the *Save* button to save its coordinates. In order to select some different areas that have the same function in the parking lot, the administrator can click the *OK* button between those areas. The tool is also useful in case the camera's position or any parking lot parameter changes, such as changing the parking location or the number of gates. In this case, the administrator can easily modify the coordinates so that the system can run with these new coordinates.

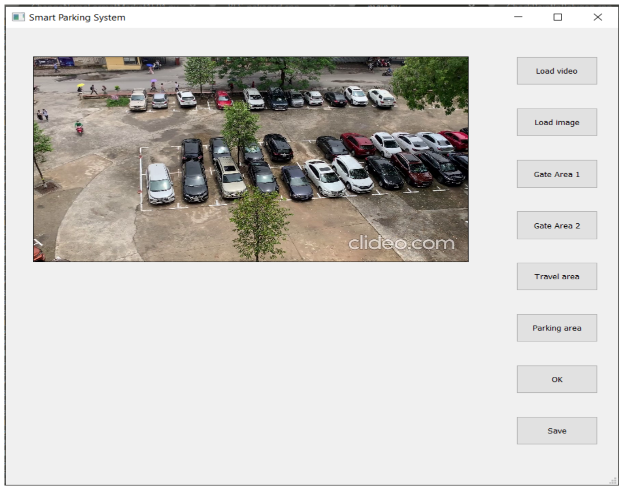

Figure 3: User interface of coordinates determination tool

YOLOv5s - The results of YOLOv5s on the evaluation set are shown in Table 1. It can be seen that the mAP of both empty slots and vehicle labels achieved good results. The average mAP of these two labels reached 90.6%, giving a good detection performance in the actual assessment. However, in some complicated conditions, for example, when other objects largely hide the vehicle because of the camera angle, vehicle detection becomes more challenging, reducing the certainty of YOLOv5s. In addition, other factors, such as poor image resolution and dark or glare lighting conditions, also lessen the detection results. Besides, the time for processing a frame

Table 1: YOLOV5 detection results on

Class	Precision	Recall	mAP@.5	mAP
Empty slot	0.994	0.960	0.979	0.923
Vehicle	0.974	0.947	0.971	0.888
Average	0.984	0.954	0.975	0.906

to give the prediction is quite small. Specifically, YOLOv5s takes about 0.045(s) to produce the result with a corresponding input image, which means the detection speed is about 22 FPS on the Nvidia GTX 1080 GPU. The time spent for training an epoch of YOLOv5s was approximately 51 seconds. The computational complexity was about 15.8 Giga floating-point operations per second (GFLOPs). And the memory complexity when training with the hyper-parameters mentioned in section 4.2 was about 5.6 gigabytes.

Deep SORT - Table 2 shows the experimental prediction speed of the Deep SORT algorithm, including the detection stage on the Nvidia GTX 1080 GPU. It can be seen that the prediction speed of Deep SORT depends largely on the number of vehicles that are tracked. In our experiments, we found that Deep SORT works pretty well because of the following reasons: Our proposed method only concerns vehicles that have just entered the gate and are in the moving area. Therefore, the number of vehicles to be tracked is not too large for the context of an actual parking lot. This makes it easy for the system to track vehicles and avoid being confused with other vehicles of no interest in the moving area, even if their trajectories overlap. However, in some complicated cases where the objects disappear from the frame for a long time, Deep SORT re-identifies them. Thus, the returned results of the latter phase are affected.

Table 2: Inference speed of Deep SORT on GPU GTX 1080

Number of objects	FPS
50	5 FPS
10	18 FPS

BFS - We conducted experiments with a number of different parameter values, based on the context of the available videos, to evaluate the speed of the BFS algorithm. The results of these experiments can be seen in Table 3. It can be seen that when increasing or decreasing the number of grids to ensure the existence of grids in the parking slots, the processing time of BFS also tends to increase or decrease, respectively, although not significantly. In general, the calculation speed is still fast enough. Comparing the running time when suggesting 4 or 12 vehicles simultaneously, the speed for 12 vehicles is somewhat faster than 4 vehicles, which can be explained by the computational speed of the algorithm depending a lot on the layout of the grids on the graph, such as the position of the source relative to the destination and surrounding obstacles.

Table 3: Experimental results of the shortest path finding algorithm. Runtime averaged over the whole video

Number of vehicles	Number of grids	Runtime (second)
3	40	0.0013
	60	0.0025
	80	0.0043
4	40	0.0016
	60	0.0031
	80	0.0049
12	40	0.0018
	60	0.0029
	80	0.0047

Application - The experimental response speed of the application achieves 14 FPS on the Nvidia GTX 1080 GPU when suggesting 5 cars simultaneously. It can be seen that in the processing stream of the application, the object tracking stage is the one that takes the most time. Fig.4 shows an example result of our proposed method with the user interface of the demo app.

From the results obtained above, it can be seen that this system has solved the set objectives. Specifically, the object detection stage combined with post-processing has quickly given the number of empty slots in the parking lot. And the object tracking stage and the BFS algorithm gave each user the path to the nearest empty slot. In addition, a demo of a user interface with experimental functions such as support for viewing expenses to be paid and viewing parking status is also provided. From these functions, the system can completely develop other functions or combine them with the RFID card system to establish a complete parking system and apply them in practice.

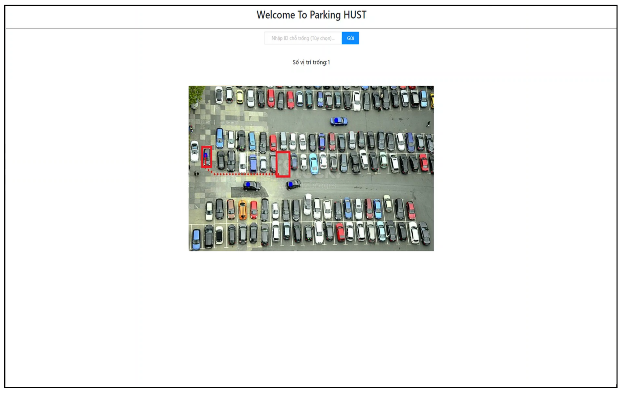

Figure 4: Example result of proposed method

5 CONCLUSION

This manuscript proposes a new multifunctional parking prototype that overcomes the parking obstacles of previous studies. The development process of the proposed prototype can be summarized as follows: The first point concerns vacant slot detection, vehicle detection, and vehicle tracking. Training has been conducted with YOLO, unlike the other studies that used the IoT method. YOLO allows our prototype to obtain good accuracy at high speed before forwarding to the Deep SORT components. From the experiment, the speed of Deep SORT is based mostly on the number of empty slots and tracking vehicles. The second point relates to the shortest path to vacant space. In general, the speed of BFS is still fast enough based on the dividing grid. In addition, this manuscript published collected data for vehicle detection and vehicle tracking in the parking lot and designed a demo application to make clear visualization for users.

In the future, we expect to continue collecting data from different environments and viewpoints of the camera to make the original dataset more diverse and investigate multiple situations. The speed of application still depends heavily on the Deep SORT component, so another avenue for future proposals is to improve the Deep SORT algorithm.

ACKNOWLEDGMENTS

This research is funded by Hanoi University of Science and Technology (HUST) under project number T2022-PC-052.

REFERENCES

[1] 2020. Car object detection dataset by Balraj. https://www.kaggle.com/code/balraj98/yolo-v5-car-object-detection. Accessed: 2022-08-28.
[2] 2020. Car object detection dataset by Sshika Maru. https://www.kaggle.com/datasets/sshikamaru/car-object-detection. Accessed: 2022-08-28.
[3] 2021. YOLOv5 published by Ultralytics. https://github.com/ultralytics/yolov5. Accessed: 2022-07-29.
[4] 2022. Basement parking occupancy and location system. https://baigiuxethongminh.vn/chi-tiet-dich-vu/he-thong-bao-cho-trong-va-vi-tri-do-xe-tang-ham. Accessed: 2022-09-04.
[5] 2022. BFS algorithm from Geeksforgeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph. Accessed: 2022-09-04.
[6] 2022. Istock website. https://www.istockphoto.com. Accessed: 2022-08-28.
[7] 2022. Pybind11 library. https://github.com/pybind/pybind11. Accessed: 2022-09-04.
[8] Zongyuan Ge Alex Bewley, Fabio Ramos Lionel Ott, and Ben Upcroft. 2016. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing*. IEEE, 3464–3468. https://doi.org/10.48550/arXiv.1602.00763
[9] Muhamad Izham Hadi Azhar, Fadhlan Hafizhelmi Kamaru Zaman, Nooritawati Md. Tahir, and Habibah Hashim. 2020. People Tracking System Using DeepSORT. In *2020 10th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*. 137–141. https://doi.org/10.1109/ICCSCE50387.2020.9204956
[10] Muhammad Azhad Bin Zuraimi and Fadhlan Hafizhelmi Kamaru Zaman. 2021. Vehicle Detection and Tracking using YOLO and DeepSORT. In *2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. 23–29. https://doi.org/10.1109/ISCAIE51753.2021.9431784
[11] Léon Bottou et al. 1991. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes* 91, 8 (1991), 12.
[12] Harshitha Bura, Nathan Lin, Naveen Kumar, Sangram Malekar, Sushma Nagaraj, and Kaikai Liu. 2018. An edge based smart parking solution using camera networks and deep learning. In *2018 IEEE International Conference on Cognitive Computing (ICCC)*. IEEE, 17–24.
[13] Paulo RL De Almeida, Luiz S Oliveira, Alceu S Britto Jr, Eunelson J Silva Jr, and Alessandro L Koerich. 2015. PKLot–A robust dataset for parking lot classification. *Expert Systems with Applications* 42, 11 (2015), 4937–4949.
[14] Stefan Edelkamp and Stefan Schrodl. 2011. *Heuristic search: theory and applications*. Elsevier.
[15] Robin Grodi, Danda B Rawat, and Fernando Rios-Gutierrez. 2016. Smart parking: Parking occupancy monitoring and visualization system for smart cities. In *SoutheastCon 2016*. IEEE, 1–5.
[16] Fuheng Guo and Yi Xu. 2022. Vehicle Analysis System Based on DeepSORT and YOLOv5. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning and International Conference on Computer Engineering and Applications*. 175–179. https://doi.org/10.1109/CVIDLICCEA56201.2022.9824363
[17] Meng-Ru Hsieh, Yen-Liang Lin, and Winston H. Hsu. 2017. Drone-based Object Counting by Spatially Regularized Regional Proposal Networks. In *The IEEE International Conference on Computer Vision (ICCV)*. IEEE.
[18] Santosh Divvala Joseph Redmon and Ali Farhadi Ross Girshick. 2016. You only look once: Unified, real-time object detection. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 779–788. https://doi.org/10.48550/arXiv.1506.02640
[19] 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.
[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.
[21] Zeydin Pala and Nihat Inanc. 2007. Smart parking applications using RFID technology. In *2007 1st Annual RFID Eurasia*. IEEE, 1–3.
[22] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*. IEEE, 3645–3649.
[23] Zizhang Wu, Weiwei Sun, Man Wang, Xiaoquan Wang, Lizhu Ding, and Fan Wang. 2020. Psdet: Efficient and universal parking slot detection. In *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 290–297.
[24] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. 2015. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3973–3981.
[25] Y. Wu Z. Wang, A. Thirunavukarasu L. Yang, and Y. Zhao C. Evison. 2021. Fast personal protective equipment detection for real construction sites using deep learning approaches, Vol. 21. Sensors, 3478. https://doi.org/10.3390/s21103478