Integrated smart public parking system for Malaysia

Integrated smart public parking system for Malaysia

Low Zhi Qi[1], Douglas Tong Kum Tien[1, a)] and Swee King Phang [1]

[1]School of Engineering, Taylor's University, Malaysia

[a)] douglaskumtien.tong@taylors.edu.my

Abstract. Searching for unoccupied parking spots is a challenge in most modern cities. The increase in the number of vehicles, limited parking spaces and low-speed parking search exacerbated traffic congestion. Consequently, smart parking system solutions have grown in popularity. Smart parking systems were developed in the early 2000s and have advanced to help with parking issues in various countries. In Malaysia however, there is slow adoption of smart parking systems although the need is equally urgent. The aim of this study is to investigate the feasibility of applying a parking spot detection system using cameras installed on lampposts for outdoor public parking in Malaysia. This study aims to demonstrate that a system of lesser complexity can be useful. The proposed solution combined parking spot detection with timing of the parking duration to help drivers monitor their parking time. This can be achieved using image detection and applying filters to identify the occupancy of parking spots. Real-time occupancy data is then exported with an excel file and this data is saved for further study and application development.

1. Introduction

Drivers in Kuala Lumpur spend an average time of 25 minutes to search for a free parking spot and an average of 53 minutes daily in traffic congestion due to open-air public parking search [1]. The queue caused by parking search have caused traffic congestion leading to negative impact on the environment, economic loss, time loss, etc. Irresponsible drivers are tempted to double park at busy areas that causing even more traffic congestion. There are questionable parties who have taken advantage of drivers by charging parking fees in the areas that belong to government at unreasonable prices. Such fees should be collected by local authorities but have been illegally taken by these unauthorized parties.

This study aims to investigate the possibility of an efficient and cost-effective solution through fully integrating public parking availability, detection, and fee charging system. Secondly, to assess the system's feasibility in terms of functionality and cost efficiency for implementation in Malaysia.

Several efforts have been carried out to overcome parking search problems in various cities outside Malaysia. Polycarpou et al. [2] discussed the statistic on drivers' survey about the need for smart parking system. They have provided several indoor parking solutions with different approaches including using ultrasonic sensor or magnetic sensor at each parking spot, cameras to monitor several parking spots, and Internet of Things (IoT) was proposed for further research. Different smart parking concepts were suggested to overcome parking issues such as parking reservation to allow drivers a short reservation time of the parking spot to save time when they arrive, dynamic pricing to increase the parking fee according to the demand, and smart parking deployment, where drivers have access to real-time parking availability information, schedules, and traffic conditions.

Soni [3] proposed a smart parking system using IoT cloud incorporated system for indoor and outdoor parking areas. The system architecture uses Bluetooth to communicate with the application and ultrasonic sensor to detect indoor parking spot occupancy. Outdoor parking uses the bit mounts with attractive sensor module that is able to get Universal Subscriber Identify Module through Bluetooth correspondence. Figure 1 shows a diagram of the system for an open-air public parking area.

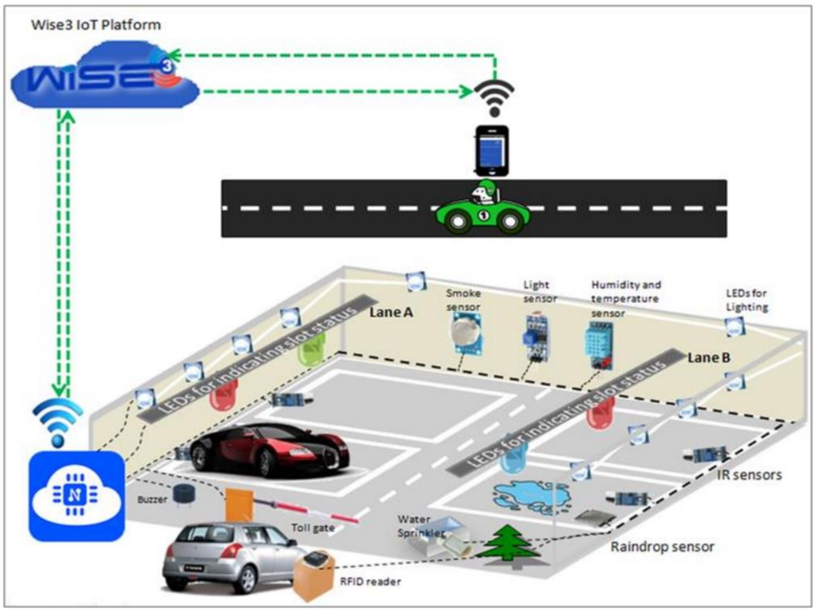


Figure 1. Description diagram of the proposed system [3]

Based on data from multiple cameras, Vítek and Melničuk [4] created a parking monitoring system to identify whether a parking place is occupied. The system is implemented using tiny camera modules built on a Raspberry Pi Zero, a computationally efficient occupancy detection algorithm based on the histogram of oriented gradients (HOG) feature descriptor and support vector machine (SVM) classifier, as well as a convolutional neural network (CNN) of database called PK-Lot, PK-Slots, and FEL-Slot that contains over 10,000 images in various weather conditions. The results is a system with the ability to supply occupancy information with greater than 90% accuracy at a rate of 10 parking spaces per second under a variety of circumstances.

In a case study in Penang Island by Thinxtra company [5], the Penang state government has implemented Malaysia's first parking system using IoT system. A total of 36,000 individual magnetic sensors were installed in outdoor parking spots. At the same time, the system provided for cashless payment. From the results, there was a reduction of 50% time for parking search and a 35% increase in parking fee revenues. Figure 2 shows a screenshot of the app and figure 3 shows the location of the magnetic sensors used in the system.

The method proposed by Al-Kharusi and Al-Bahadly [7] takes a circular image drawn on the parking lot, analyses it, and outputs data about available parking spots for indoor and outdoor parking. Two approaches were observed: applying point detection with canny operators together with comprehensive analysis method utilizing an aerial view of the parking lot, and edge detection with boundaries condition method for the image detection module.

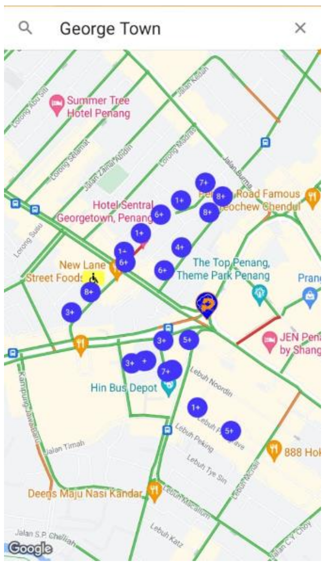


Figure 2. Sample screenshot from the Penang smart parking application user interface

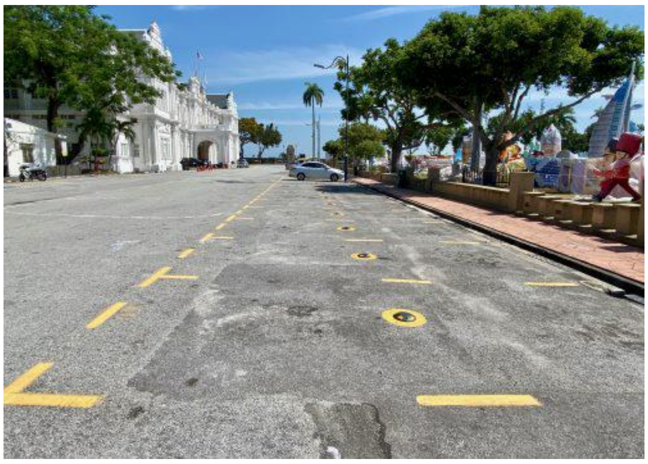


Figure 3. Location of magnetic sensors for the Penang smart parking system [6]

The limitations of the system proposed by Polycarpou et al. [2] are that for indoor parking only theoretical information is available with no statistics given on its efficiency. Research done in the survey of parking issue among drivers provided with preliminary suggestions only. An IoT system using ultrasonic sensor, passive infrared sensor, and Bluetooth module have been proposed by Polycarpou et al. to communicate the occupancy information in both indoor and outdoor parking through drivers' application using Universal Subscriber Identify Module. The efficiency of ultrasonic sensors and IR sensors are limited by the environment in which they are used. Ultrasonic sensors are not able to work accurately when the receiver is covered by obstacles while IR sensors are not able to work under low light condition. This may not be suitable in areas in Malaysia that possess less quality street lightning. This research requires more statistic to prove its functionality.

Deep learning using camera proposed by Vítek and Melničuk [4] shows promise with an accuracy of more than 90%. However, the resources and time to train the system are important factors to consider. Next, the magnetic sensor system for open-air parking installed by Penang government [5] is limited to the quantity of the sensors along with the component cost and maintenance cost as the sensors need to be install at each parking spot. Implementation also requires massive manpower and resources to maintain sensors working condition.

In the approach by Hilal and Ibrahim [7], it shows similar method of image processing to approach the indoor and outdoor parking spots using a camera system developed abroad. This method is limited by the weather conditions due to the filter which uses the morphological processing method.

This research focusses on parking detection through image processing using OpenCV and real-time camera implementation. The system is able to record the parking duration using IP address in Raspberry Pi cameras for purpose of payment. Raspberry Pi is a small single board with Linux operating system and is able to connect with different devices with general purpose input and output pins. In this study specifically, it is connected to the camera input built-in on the board. The Raspberry Pi board is able to run the code with the connection to the camera directly. There are multiple advantages of using Linux operating system but one main advantage is the characteristic of feasibility desktop applications, embedded systems, and server applications [8] which are related to this study.

OpenCV stands for computer vision which is an open-source library for computer vision, machine learning and image processing with real-time operation [9]. It is a Python library that allow to users identify, calculate, check, determine the object in the image frame using different approaches depending on their purpose. With the massive open-source support and online forum online, it is more feasible and cost effective to apply it to this project.

2. Methodology

At the setup phase, starting with the main component of the parking spots detection, camera along with necessary connection to the Raspberry Pi is required to have the image output to the image processing. To connect the camera with Raspberry Pi, several dependencies for Raspberry Pi camera and software for Python are downloaded to run the specific function. Once the video image is collected by the camera, it will be read by the user developed Python code to identify and allow operators to select the parking space orientation in the parking lot manually. Once the orientation of the parking spaces setup is done, threshold value differences between the empty and occupied space will be set by monitoring it with the applied filter. After the setup is done, the system is able to continuously predict the parking occupancy with the developed Python code.

From above overview, the system development can be separated into different parts. The camera, set up with the Raspberry Pi board and dependencies. Image detection, based on the application of the filters. Parking prediction, the method of predicting the threshold value to identify the space occupancy. Measurement of parking duration, the timing method used to calculate the parking duration in the parking spot.

2.1. Camera selection

The goal of the study is to live-stream the open-air public parking and process the information to identify the parking spot occupancy. Therefore, Raspberry Pi camera module 2 No IR is selected which operates without IR illuminator that provides ability to work under insufficient light condition during evenings with built-in operations and a set of GPIO pins through Python [10], [11]. Raspberry Pi camera module 2 No IR has a sensor resolution of 3280×24644 pixels with 1.12 μm ×1.12 μm pixel each. Field of view reaches 62.2 degree horizontally and 48.8 degree vertically [12]. It is connected along with Raspberry Pi Board Model 3B+ using the newest operating system version of Debian Linux namely Bullseye. The Linux operating system is able to support the Python programming language.

2.2. Image detection

Once the live-stream video is taken from the camera, the parking spots will be orientated manually to run image processing. The concept for the image detection is to use threshold to set the global value, if the pixel value is smaller than the global value, the pixel will set to 0, vice versa. With this approach, if the parking area is occupied, the pixel value of the vehicle area will return a high due to the brightness. With the calculation and overall monitoring the pixel value, a global value will be set to classify the occupancy of a parking spot.

The system is developed according with the input of Raspberry Pi camera, however due to technical issues, unfortunately the camera failed to function as expected. Therefore, the system used the image and video that are taken by mobile phone to temporarily debug the Python code. Due to the different working principle of mobile phone camera and the Raspberry Pi camera, the size and orientation of the image and video were swapped. Code to resize and reorientate using OpenCV and imutils library was added to ensure the size of the video and image are same before moving to the next step.

Before setting up the threshold, multiple boxes were created to orientate parking spots on the screenshot of the video with the appropriate overall size. Due to the installation camera angle and variety of the parking spots orientation, it caused different views of parking spots to the camera. Therefore, the parking spots is set manually at the setup once for all.

After selecting the car park spot manually, several filters were applied to the video to calculate the pixel after applying the threshold to identify the occupancy including, convert color, Gaussian Blur, adaptive threshold and dilate filter. The coding of each filter along with parameters is shown in figure 4.

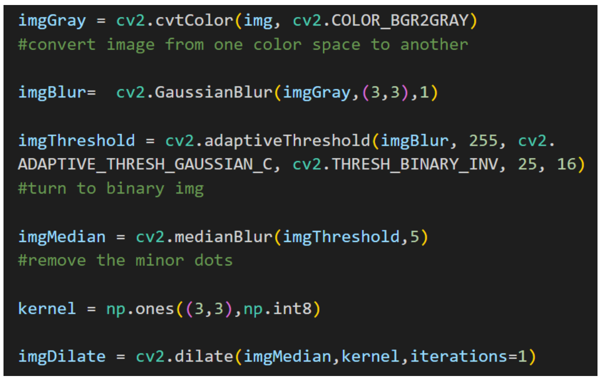
Source image had to be converted to grayscale image [13]. With this, the image has only color range from black to white with pixel value of 0 to 255 that represent the brightness. Gaussian Blur is a type of image smoothing that removes high frequency content from the image by convolving the image with a low-pass filter kernel [14]. It blurs the image to prevent unnecessary pixel factor to affect the image pixel value.

Adaptive threshold defines a global value as a threshold to determines the value for a pixel based on a small region around it [15]. Adaptive thresh gaussian computes the means blocks of pixel values with respect to a content, wherein the constant value is subtracted from the mean of pixels [16]. The output maximum pixel is set to white and black with 255 which represent the maximum brightness of the pixel. If the source image pixel larger than 25, it will set to white, vice versa, the pixel lower than 16, it will set to black. This enables to classify all pixel to white and black for further calculation.

The median was used to replace the central element of the image by the median of all the pixels in the kernel area [17]. This process is done to remove minor pixels for detection accuracy. Dilate filters are morphological filters applied to binary images that increases the boundaries of regions of foreground pixels. Image dilation increases the object area. Kernel is set to 3 × 3 matrix which is a required parameter to convolve the image [18]. It expands the number of pixels to the boundaries of object to further differentiate the pixel number in the area for calculation [19]. The code of filters is shown in figure 4 for better understanding of the filter application.

2.3. Parking prediction

After applying the filters, the image is cropped into the size of the manually selected space for pixel counting. When running the code, the occupied space and free space have different pixel values of hundreds which enable the classification of the status. 900 is the threshold value between occupied and free parking space. Therefore, if the pixel value exceeded 900, the parking space will be shown as occupied and vice versa.



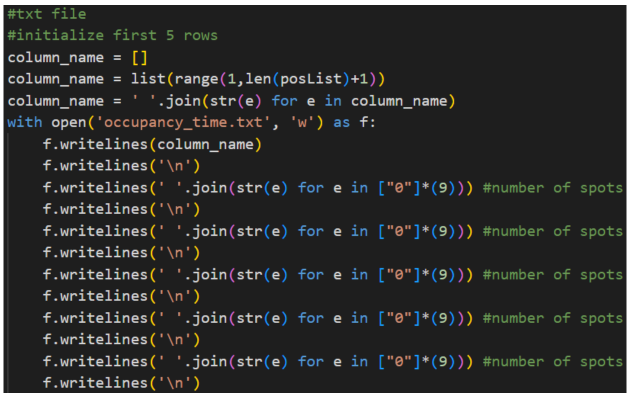
```
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#convert image from one color space to another

imgBlur=  cv2.GaussianBlur(imgGray,(3,3),1)

imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.
ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 25, 16)
#turn to binary img

imgMedian = cv2.medianBlur(imgThreshold,5)
#remove the minor dots

kernel = np.ones((3,3),np.int8)

imgDilate = cv2.dilate(imgMedian,kernel,iterations=1)
```

Figure 4. Filter code for image processing.

After recognizing the occupancy, all information was saved in a generated text file to record the detection result. The saved result can be carried forward for user interface mobile application along with management interface to supervise the functionality and status. While the car park spot is empty, it will append the status as zero in the new row and vice versa.

2.4. *Measurement of parking duration*

A text file named "occupancy time" was created to save the information from the image processing. Before saving the status of the parking spot, the headers of the table were labelled with the ID given to the parking spaces accordingly. The parking status were saved as 1 and 0 in the list call "pred". Figure 5 shows the code of text file created along with the headers and prediction list.

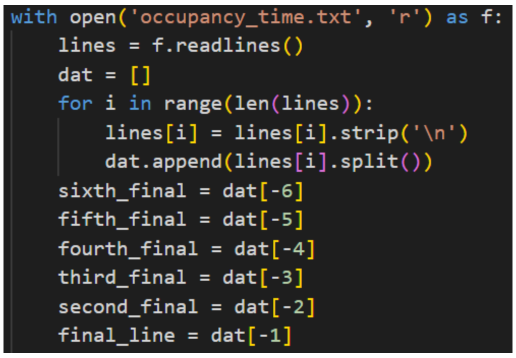


```
#txt file
#initialize first 5 rows
column_name = []
column_name = list(range(1,len(posList)+1))
column_name = ' '.join(str(e) for e in column_name)
with open('occupancy_time.txt', 'w') as f:
    f.writelines(column_name)
    f.writelines('\n')
    f.writelines(' '.join(str(e) for e in ["0"]*(9))) #number of spots
    f.writelines('\n')
    f.writelines(' '.join(str(e) for e in ["0"]*(9))) #number of spots
    f.writelines('\n')
    f.writelines(' '.join(str(e) for e in ["0"]*(9))) #number of spots
    f.writelines('\n')
    f.writelines(' '.join(str(e) for e in ["0"]*(9))) #number of spots
    f.writelines('\n')
    f.writelines(' '.join(str(e) for e in ["0"]*(9))) #number of spots
    f.writelines('\n')
```

Figure 5. Measurement of parking duration text file

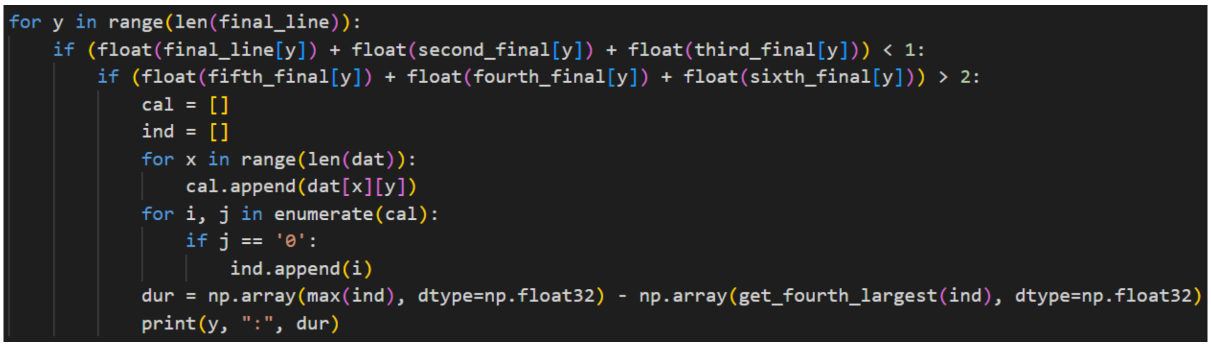
To count the parking duration without distraction of other factors, a total of 6 lists at the ending of the occupied parking status was created to save the occupancy status of the parking spot. The system will only change the parking status from occupied to free when the last 6 to 4 lines are stated as 1, and last 3 to 1 lines are stated as 0 continuously. If the status does not meet the condition stated, the parking spot status will remain as occupied. This function can prevent the system from miscalculating the parking duration of the spot if the pixel values in the parking space is fluctuating between the threshold value. Figure 6 shows the code of created list to prevent duration miscalculation.

If system detected the car leaving the parking space, the system will read the last row of 0 before the status turn to 1. The system will deduct the first row of 0 after turning 1 with the last 0 before change to 1, to find out the parking duration. To explain, the system will find out when the car parked in the space and the time it leaves to calculate the parking duration. Figure 7 shows the code of the system detects the car leaving conditions with the car park duration.



```python
with open('occupancy_time.txt', 'r') as f:
    lines = f.readlines()
    dat = []
    for i in range(len(lines)):
        lines[i] = lines[i].strip('\n')
        dat.append(lines[i].split())
    sixth_final = dat[-6]
    fifth_final = dat[-5]
    fourth_final = dat[-4]
    third_final = dat[-3]
    second_final = dat[-2]
    final_line = dat[-1]
```

Figure 6. Lists of parking status



```python
for y in range(len(final_line)):
    if (float(final_line[y]) + float(second_final[y]) + float(third_final[y])) < 1:
        if (float(fifth_final[y]) + float(fourth_final[y]) + float(sixth_final[y])) > 2:
            cal = []
            ind = []
            for x in range(len(dat)):
                cal.append(dat[x][y])
            for i, j in enumerate(cal):
                if j == '0':
                    ind.append(i)
            dur = np.array(max(ind), dtype=np.float32) - np.array(get_fourth_largest(ind), dtype=np.float32)
            print(y, ":", dur)
```

Figure 7. Calculation of parking duration

The overall system flowchart of the methodology aforementioned is shown figure 8. The video is captured by the Raspberry Pi camera and read by the code. At the setup phase, the developer needs to manually orientate the parking spots accordingly for the system to identify the detection area. Then, the status of the selected detection area is being predicted by applying multiple filters. As the total pixel value in the detection area exceed the occupied condition of the parking status, it will predict the parking spot as occupied and save 1 in the txt file corresponding to the ID no. The system works as a while loop, repeating the loop until interrupts occur.

3. Result and Discussion

3.1. Camera testing

The raspberry Pi board along with camera setup was successfully carried out. The camera was able to read the image fed directly from the camera. Ideally, the real-time video will send to Raspberry Pi board and run the code. However, due to the installation limitation, the status of staff parking is recorded through the setup camera and passed to the Python code to run it. The camera took the frame of the image at the rate of 25 per second resulting in slow response which was able to process large data without clashing.

As shown in figure 9, it was possible to take clear images with distance of 2 floors above the parking spot, a height equivalent to a lamppost height. Approximately 6 -10 parking spots with different installation angles can be covered. The actual statistic needs to be confirmed with actual installation on the lamppost.

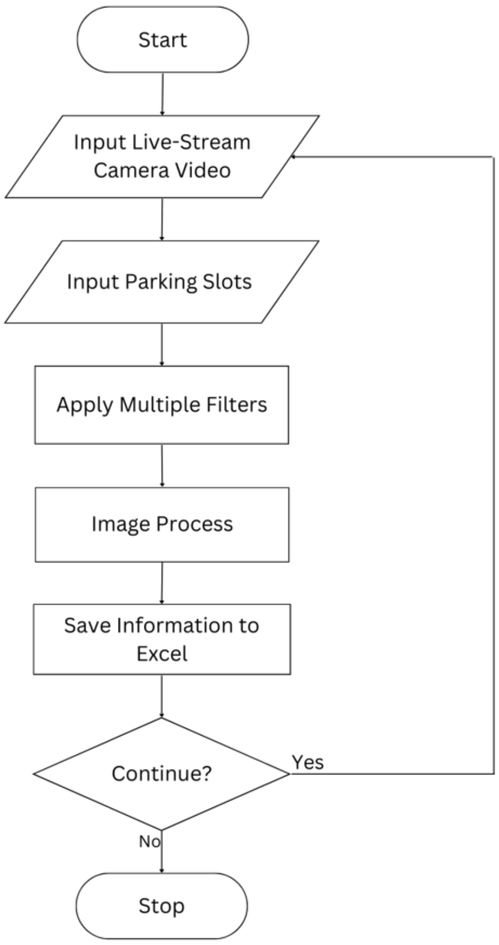


Figure 8. Algorithm flowchart

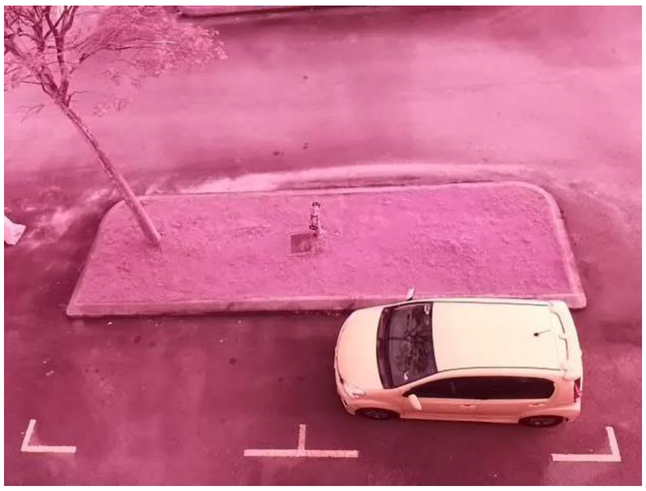


Figure 9. Image capture at level 2 of a building adjacent to the car park

3.2. Parking prediction

The parking picker boxes were allocated accordingly at the parking spot orientation at Taylor's University's staff parking with a width of 260 pixels and height of 110 pixels. The boxes need to be positioned at places that have no obstacles including to the parking spot edge line in the area to ensure detection accuracy. With the line in the detection box, it decreased the pixel value difference gap between occupied and free spot. The oriented parking spot boxes are shown in figure 10.

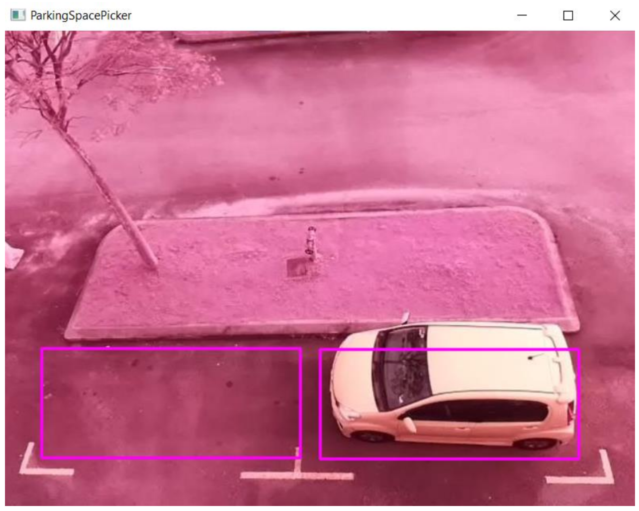


Figure 10. Parking space picker boxes

With the filter applied, it converted the image to grayscale as shown in figure 11.

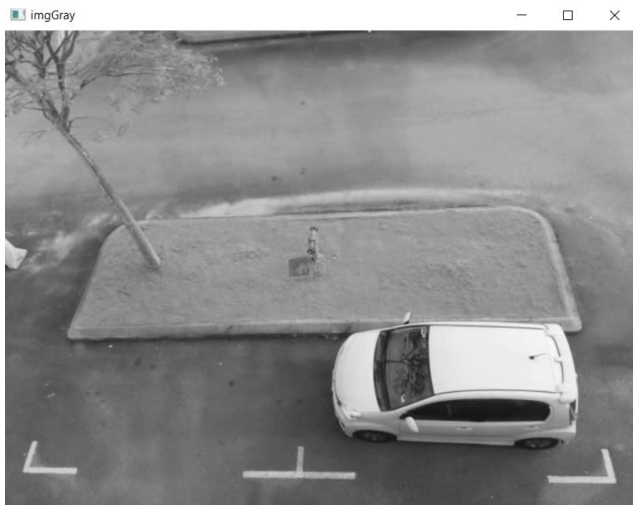


Figure 11. Grayscale image

Next, image is blurred with kernel of 3 × 3 shown in figure 12.

Differences of existence of blur filter is shown after the implementation of adaptive threshold filter. With the blur filter, it ignores the high frequency scattered pixels of the image. Figure 13 is blurred with kernel matrix of 3 × 3, however, figure 14. is blurred with kernel matrix of 5 × 5. As shown, the larger the kernel value, the more blur the image gets. The pixels were less visible with 5 × 5 kernel matrix.

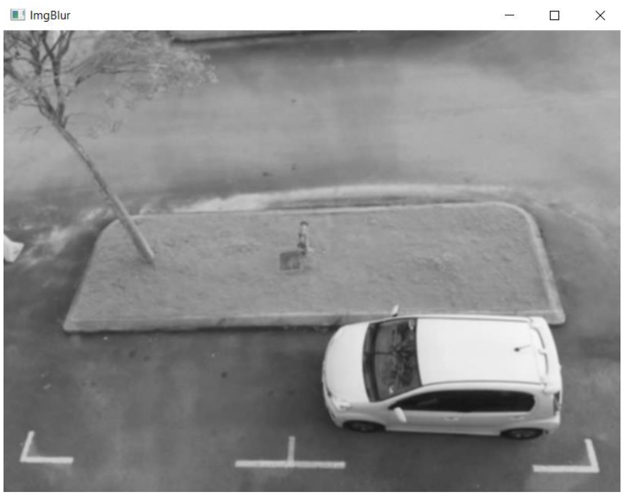


Figure 12. Blurred image

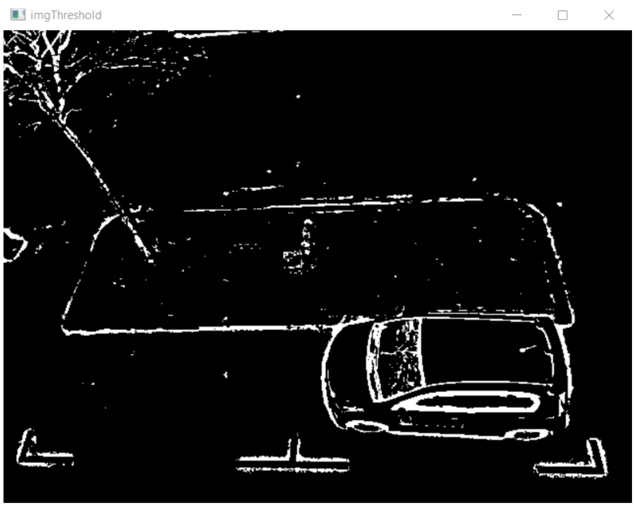


Figure 13. Threshold image with 3×3 kernel

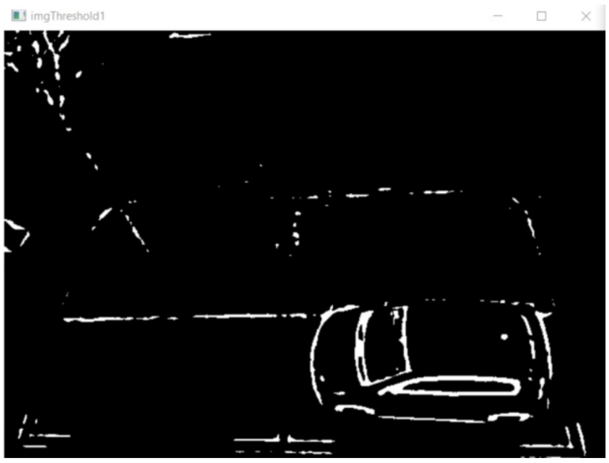


Figure 14. Threshold image with 5×5 kernel

Thirdly, for the adaptive threshold filter with Gaussian method, the threshold binary inverse type with constant threshold value of 16. Figure 15 and figure 16 shows the differences of threshold value of 16 and 18. Constant threshold values with 16 have lesser pixels at the surroundings and thinner line on the detected vehicle. Both results do not show big changes in the result as the pixels in the allocated boxes where detection happens are similar. However, the threshold constant is set to 16 so that the image capture is less affected under insufficient light condition.

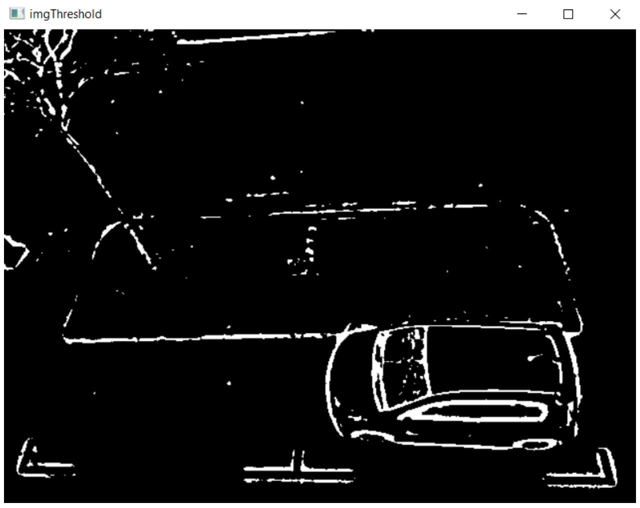


Figure 15. Threshold image with threshold value of 16

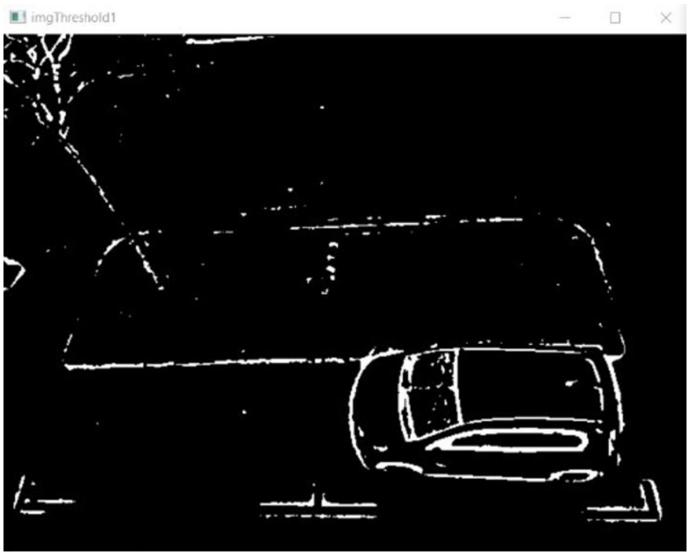


Figure 16. Threshold image with threshold value of 18

The median filter was the fourth filter applied. It removed minor scattered pixels at surroundings to prevent miscalculation of unnecessary pixels that affect the result. As shown in figure 17, only the only the outline of the vehicle was reserved.

The dilate filter thicken the outline of the vehicle with kernel of 3 × 3 and iterations of 1, resulting in higher total pixels for the occupancy prediction. The larger the kernel and iteration values, the thicker outline as shown in figure 18.

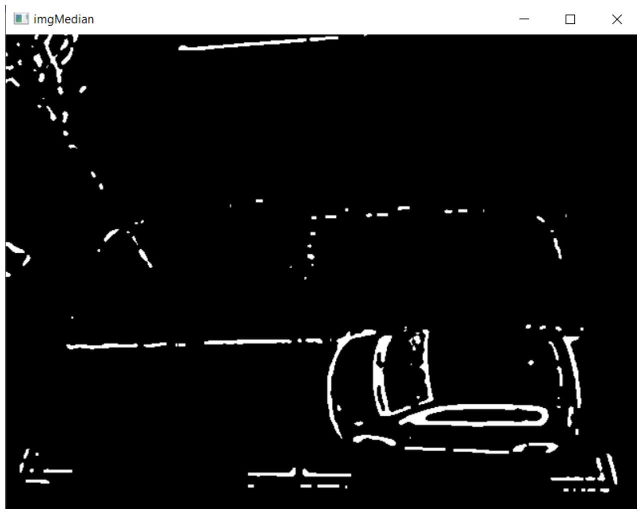


Figure 17. Image with median filter applied

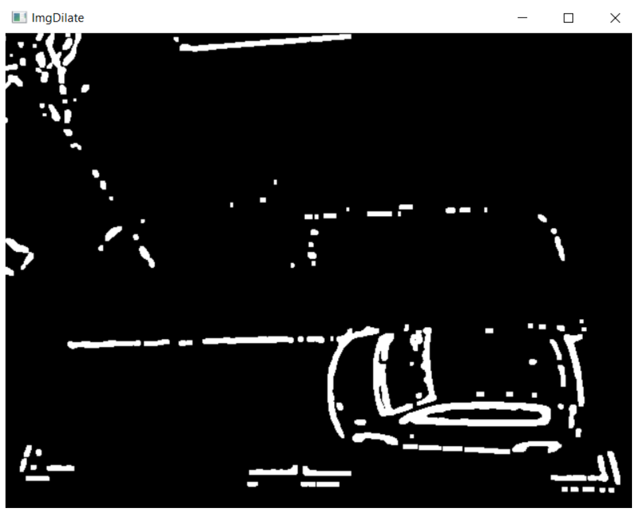


Figure 18. Image with dilate filter applied

The boxes in figure 19 shows the total pixel differences of occupied and non-occupied parking space of the binary image after filtration.

From the detected total pixels value, there is a huge gap between occupied and free parking spots, 5000 and 0. This result is due to the parking spot size detected in the image. When the parking spot size is bigger, it resulted in bigger pixel areas in the image which will contribute to higher pixel values. Hence, the outcome of the total pixel values reaches 5000. Vice versa, smaller size of parking spot will result in smaller total pixel values. On that account, the parking spot is classified as occupied when the condition of the total pixel value in the spot reached above 2000.

With wider detection of parking area, the pixel value will definitely have lesser total pixels, therefore the critical threshold value needs to be determined every new setup.

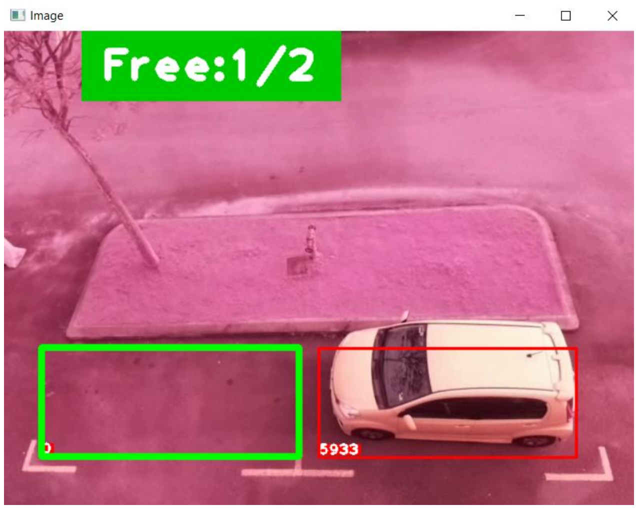


Figure 19. Pixel values of difference parking status

3.3. Measurement of parking duration

With the stabilization of parking status, the result was efficient and accurate. The result of parking duration was saved in a CSV file named 'oneshot.csv'. 2 columns were shown in the file, the first column indicates the parking spot ID and the second indicates the duration in seconds. From the result obtained, the car was parked at the parking spot ID named '0' for a total 92.68 seconds. The parking duration will further append in the new rows to record the parking duration. Additionally, parking fees can potentially be calculated with multiplication and conversion in the code.

4. Conclusion

This paper investigated the feasibility of a parking system by developing a simple system to detect parking occupancy and calculate parking duration using Raspberry Pi camera and OpenCV image processing. The system is intended to help drivers find parking spot at open-air outdoor public parking areas more efficiently. From the result obtained, it appears that this proposed system is feasible and can meet the objectives of parking spot detection and parking duration detection. The accuracy of the system can be further improved with the orientation of camera coverage of the parking spot, and enhanced filter parameters under different lightning and environmental conditions.

The following is the subject for future work. The parking spot detection system should be integrated with an app that drivers can download and use on their mobile phones. Linking the app to the GPS would enable tracking of the vehicle and detection of location changes. The app will recommend several optimal parking spots based on the driver's inputted destination. The recommendations will change in real time should the previous recommended spot be taken by another vehicle before the driver reached it. Once the driver had parked the car, confirmation will be requested to clarify the exact parking location. Parking duration would then be calculated and charged from the app.

References

[1] People in Kuala Lumpur waste 25 minutes of their lives everyday looking for parking *SAYS*
[2] Polycarpou E, Lambrinos L and Protopapadakis E 2013 Smart parking solutions for urban areas *IEEE 14th International Symposium on a World of Wireless, Mobile and Multimedia Networks*
[3] Soni V D 2018 Internet of things based smart parking system using ESP8266 *International Journal of Advanced Research in Electrical Electronics and Instrumentation Engineering* **7** 1839-1843
[4] Vítek S and Melničuk P 2018 A distributed wireless camera system for the management of parking spaces *Sensors* **18** 1-14
[5] The City Councils of Penang launches Malaysia's first IoT-enabled Smart Parking Systems for

more than 36,000 parking spaces *Xperanti*
[6] Smart sensors for designated MBPP parking lots installed *Buletin Mutiara*
[7] Al-Kharusi H and Al-Bahadly I 2014 Intelligent parking management system based on image processing *World Journal of Engineering and Technology* **2** 55–67
[8] Advantages of Linux *Javatpoint*
[9] About OpenCV *OpenCV*
[10] Raspberry Pi NoIR vs normal camera *The Geek Pub*
[11] What is a Raspberry Pi *opensource.com*
[12] Raspberry Pi Documentation – Camera Module *Raspberry Pi*
[13] Python OpenCV | cv2.cvtColor() method *Geeks for Geeks*
[14] Smoothing Images *OpenCV*
[15] Image Thresholding *OpenCV*
[16] What are the different image thresholding techniques and how to implement them? *Analytics India Magazine*
[17] OpenCV - Median Blur *Tutorialspoint*
[18] Python cv2 dilate: The Complete Guide *AppDividend*
[19] OpenCV Erosion and Dilation *Javatpoint*