Dissertation on

An Image Feature-Based Method for Detection of Parking Lot Occupancy

Thesis submitted towards partial fulfillment of the requirements for the degree of

Master of Technology in IT (Courseware Engineering)

Submitted by

DEBANJAN KSHETRY

EXAMINATION ROLL NO.: M4CWE22020

UNINVERSITY REGISTRATION NO.: 131239 of 2015-16

Under the guidance of

Mr. JOYDEEP MUKHERJEE

School of Education Technology

Jadavpur University

Course affiliated to

Faculty of Engineering and Technology

Jadavpur University

Kolkata 700032

India

M.Tech. IT (Courseware Engineering)
Course Affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled "**An Image Feature Based Method for Detection of Parking Lot Occupancy"** is a bonafide work carried out by DEBANJAN KSHETRY under our supervision and guidance for partial fulfillment of the requirements for the degree of Master of Technology in IT (Courseware Engineering) from the School of Education Technology, during the academic session 2021-2022.

.....
SUPERVISOR
School of Education Technology
Jadavpur University,
Kolkata – 700032

.....
DIRECTOR
School of Education Technology
Jadavpur University,
Kolkata – 700032

.....
DEAN-FISLM
Jadavpur University,
Kolkata – 700032

Course affiliated to

M.Tech. IT (Courseware Engineering)
Course Affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF APPROVAL

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented in a manner satisfactory to warranty its acceptance as a perquisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made or opinion expressed or conclusion drawn therein but approve the thesis only for purpose for which it has been submitted.

…….

…….

Committee of final examination for

Evaluation of Thesis **……**

……..

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of his **Master of Technology in IT (Courseware Engineering)** studies.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that. As required by this rule and conduct, I have fully cited and referenced all materials and results that are not original to this work.

NAME: DEBANJAN KSHETRY

EXAMINATION ROLL NUMBER: M4CWE22020

THESIS TITLE: **An Image Feature-Based Method for Detection of Parking Lot Occupancy**

SIGNATURE: DATE:

ACKNOWLEDGEMENT

I feel gratified in presenting this thesis at the School of Education Technology, Jadavpur University, Kolkata, in partial fulfillment of the requirements for the degree of Master of Technology in IT (Courseware Engineering).

The contentment and elation that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance, supervision and encouragement crowned my effort with success.

I convey my gratitude to our guide, Mr. Joydeep Mukherjee, for his timely help, encouragement, constructive suggestion, and many more innovative ideas in carrying out this thesis.

I am also grateful to Prof. Matangini Chattopadhyay and, Director of the School of Education Technology, for her support, encouragement and helpful advice. I am really indebted to Dr. Saswati Mukherjee for her constant support during the entire course of the research work. Their advice and support were highly inspirational and helpful.

I would like to take this opportunity to pay my regards and thank to all of classmates at M.Tech IT (Courseware Engineering) motivated me to complete my research work successfully. I do wish to thank all of those who were associated with this research work.

Thanks & Regards,

_____

Debanjan Kshetry,
Class Roll No: 002030402020
Exam Roll No: M4CWE22020
Registration No.:131239 of 2015-16
Master of Technology in IT (Courseware Engineering)
School of Education Technology,
Jadavpur University,
Kolkata- 700032

CONTENTS

CHAPTER 1: INTRODUCTION

1.1. Executive Summary

Personal vehicles have evolved into a need in our daily lives as a result of the economy's ongoing growth. The majority of the working class can now afford the commodity, which offers a comfortable standard of living; nevertheless, on the flip side, numerous issues arise that must be resolved. Numerous studies on traffic congestion analyses have shown that a whopping 70% of cars on the road nowadays are looking for good parking. As a result of the increased time spent on the road by the vehicles, traffic congestion will worsen. When looking for a parking spot, drivers may also favour driving slowly. According to this field's research, cars travel at an average speed of 20 Kmph for an average of 15 minutes while searching for a parking spot, only covering a distance of half a kilometer. Traffic jams are a regular side effect. Accidents are more likely to occur while drivers are looking for parking because they are paying less attention to the road. These issues can only be solved by an advanced parking system. This is why there are a lot of research projects being done in this field globally. Any smart parking system begins with the detection of vacant parking spaces.

There are numerous intricate parking systems in use today, but they are all expensive to design, implement, and maintain. The counter at the

checkpoint is used by the management of many parking lots to count the number of vehicles entering and leaving the lot. Modern technologies precisely locate the open spaces and direct incoming vehicles accordingly. Even though some modern cars come equipped with built-in parking systems, these systems frequently fail to recognize whether a parking place is actually vacant or not.

There are still places where temporary or urgent parking facilities need to be set up despite all these strategies; this work provides a useful, image-based
approach.

Image Processing techniques is used in this thesis to detect and count vacant parking spaces in the Captured Stream Videos.

1.2. Literature Survey

The author [1] has conducted a comparison of the different car slot detection methods used in parking lots. Also demonstrated as a replacement for sensor-based systems are image processing-based system models. Based on optical character recognition, this system for identifying parking spaces is extremely effective and straightforward (OCR). The camera that is mounted in the parking space records an image of the parking area and, using OCR, finds any open spots there.

The author [2] evaluates the state of intelligent transportation today and suggests a design for a video-based parking space detection system, providing a program from the hardware and software platform architecture through the design process for the detection algorithm. Lastly, it presents the findings of on-site field experiments. The design may make some allusions to ongoing and upcoming research in intelligent transportation, or more specifically, to the administration of smart parking lots.

The author [3] used image processing to get videos from a 10 foot-tall camera from the top view of the parking lot. To improve the ability for recognition, the system video data has been recorded under diverse circumstances and temporal changes. Frames are used to separate video. Then, in order to reduce computer complexity, a key frame is extracted from each segment and subjected to additional processing. A radio-controlled toy car's motion is dictated by the main frame subtraction when you drive into or out of the parking space. The parking arena didn't initially have any parking lines. The driver manually enters the parking space and the vehicles intended for parking. The tool automatically generates virtual parking spaces that account for car size. The maximum number of parking spaces in this training model is 14. Each parking lot has been assigned a certain numerical mark. After the parking arena has been divided into virtual blocks, the device will check for the existence of the car in any block. A binary filter is applied to the image, and the automobile is removed using a ROI region. calculates the associated area's ROI interest and sets the minimum number of reserved parking spaces at more than eighty. For

divers, the number of free blocks is displayed in green, while the blocks that are reserved are displayed in red.

The author [4] suggests a image-processing-based approach for giving parking advice and information. The technology under consideration counts the number of parked cars and lists the available parking spaces. Instead of employing electronic sensors buried in the floor, the system identifies autos using photographs. The entrance to the parking lot has a camera installed. It will record video clips. Using image matching, the collected photos are sequentially matched using an image of an automobile as the reference image. Edge detection has been done for this purpose utilising the Prewitt edge detection operator, and the incoming driver is given instructions and information based on the percentage of matches.

The author's [5] goal is to offer an intelligent system for parking space recognition based on image processing approach that captures and processes the brown, rounded picture drawn at a parking lot and produces the information of the empty automobile parking spaces. It will be shown in real time at the display unit that has seven segments. The number of parking spaces currently available in the parking area is displayed on the seven section display. This system proposal has been created on a hardware and software platform.

Parking Guidance and Information (PGI) systems [6]-[8] provide real-time parking information via variable message signs to drivers looking for parking spaces. To gather information about the occupancy of parking spaces, the system makes use of specialised sensors, notably at the

entrance and exit gates of the parking lot. The parking lots of commercial shopping malls and trade centres can swiftly be equipped with these kinds of equipment.

1.3. Objective

Our system's main objective is to

- Build a system that can create or pick parking spaces.
- Detect empty vehicle slot in the picked parking spaces.

In this thesis we are going to use captured stream aerial view videos of parking lot and then create a system to pick parking spaces from the videos. The system is very easy to develop and doesn't always require parking lot to have properly marked parking spaces. The parking spaces can be created manually in the system.

In the second point, we are going to create the main system where we will be using the previous parking space picker system to detect if the parking space marked in the video is vacant or not.

This model will assist in various scenarios such as parking spaces alongside road in any specified block, temporary parking spaces created during any occasion, permanent parking spaces with marked parking spaces and landscaping elements and also parking spaces without slot numbers [1].

1.4. Problem Statement

The current image feature based approach [1] is not very flexible in real life implementation because it cannot detect vacant parking spaces without manually drawn slot numbers in parking spaces.

2. METHODOLOGY AND IMPLEMENTATION

Some of the key concepts will be discussed to understand the details of the algorithm that have been implemented for the detection and counting of vacant parking spaces in the captured stream videos using Image Processing techniques.

2.1. ROI Detection

A region of interest (ROI) is a portion of an image or dataset that has been selected for a specific objective. Here we need to detect the ROI using the Parking spaces markers on the parking slot. The camera here needs to be mounted on the roof top of a building from where we can get a clear aerial view of the parking lot and the camera also needs to be static. Here, we used captured stream videos of parking lot from the aerial point of view. Now we need to do some calibration to obtain the ROI, we need to manually get the dimensions of a parking space and then select the parking spaces from x, y coordinates, where the point of selection is generally the top left corner of each parking space. After selecting all the parking spaces, we have detected all the ROI in the parking space. The main reason behind doing this is to get the pixel count in each of this parking spaces after going through proper filtering.

Fig. 1 shows the ROI marked in all the parking spaces in video 1, with pink border, for further detection of car and counting of vacant spaces. We do this by getting a nice and clear frame from the video and using that image to create ROI for each parking space.

Fig. 1 ROI marked for each parking space in the parking lot in video 1

Fig. 2 shows the ROI marked in all the parking spaces in video 2

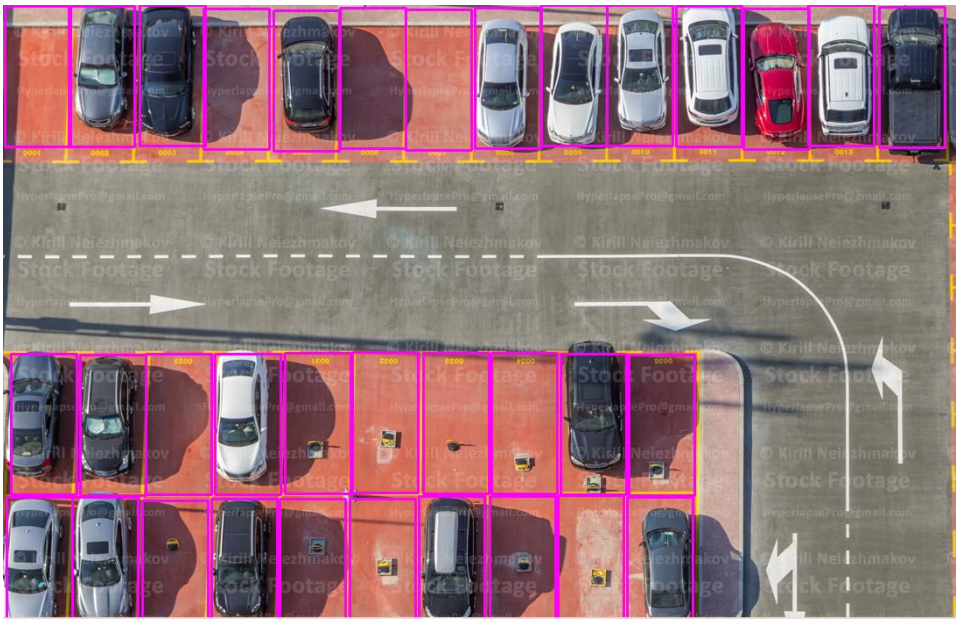

Fig.2 ROI marked for each parking space in the parking lot in video 2

Fig. 3 shows the ROI marked in all the parking spaces in video 3

Fig.3 ROI marked for each parking space in the parking lot in video 3

2.2. Image Acquisition

In the image acquisition module, image capture and storage of digital images from the video cameras are the procedures to be involved. For this, a high-definition camera must be attached to the processing unit which in turn provides the data for the Pycharm software to process in real-time. The camera alignment should be done with extreme care. In order to get a clear top view of the parking lot, the camera should be set up at a good height. Here in this thesis, we will be using captured stream videos which are recorded from a top-down perspective and set up at a good height as shown in Fig. 1, 2 and 3.

2.3. RGB to Grayscale Conversion

Grayscale is a spectrum of apparent-colorless hues of grey. Black is the darkest shade that can exist, and white is the lightest shade that can exist. White denotes total transmission or reflection of light at all visible wavelengths, while black denotes total absence of transmitted or reflected light. Equal brightness levels of the three basic colors—red, green, and blue—represent intermediate shades of grey.

The colored image is converted to grayscale using the Luminosity method which calculates each resultant gray image pixel. The grayscale conversion is necessary because the algorithms used are customized to work only on grayscale images.

Transformations within RGB space like removing the alpha channel, reversing the channel order, conversion from 16-bit RGB color using Equation [1].

$$Y=0.299*R + 0.587*G + 0.114*B \quad \text{....} [1]$$

The conversion from a RGB image to gray is done with:

```
cv2.cvtColor(image, cv2.COLOR_BGR2GRAY);
```

Fig. 4 shows the images of the parking lot with cars converted to grayscale images in Video 1.

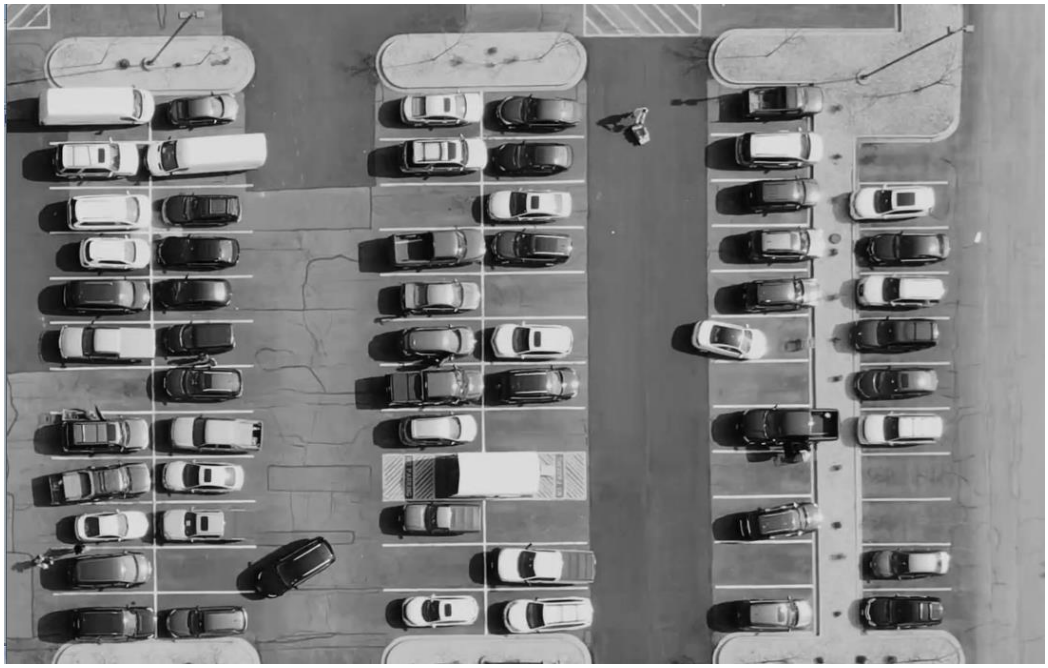

Fig.4 RGB image of a frame converted to Grayscale image in video 1

Fig. 5 shows the images of the parking lot with cars converted to grayscale images in Video 2.

Fig.5 RGB image of a frame converted to Grayscale image in video 2

Fig. 6 shows the images of the parking lot with cars converted to grayscale images in Video 3.

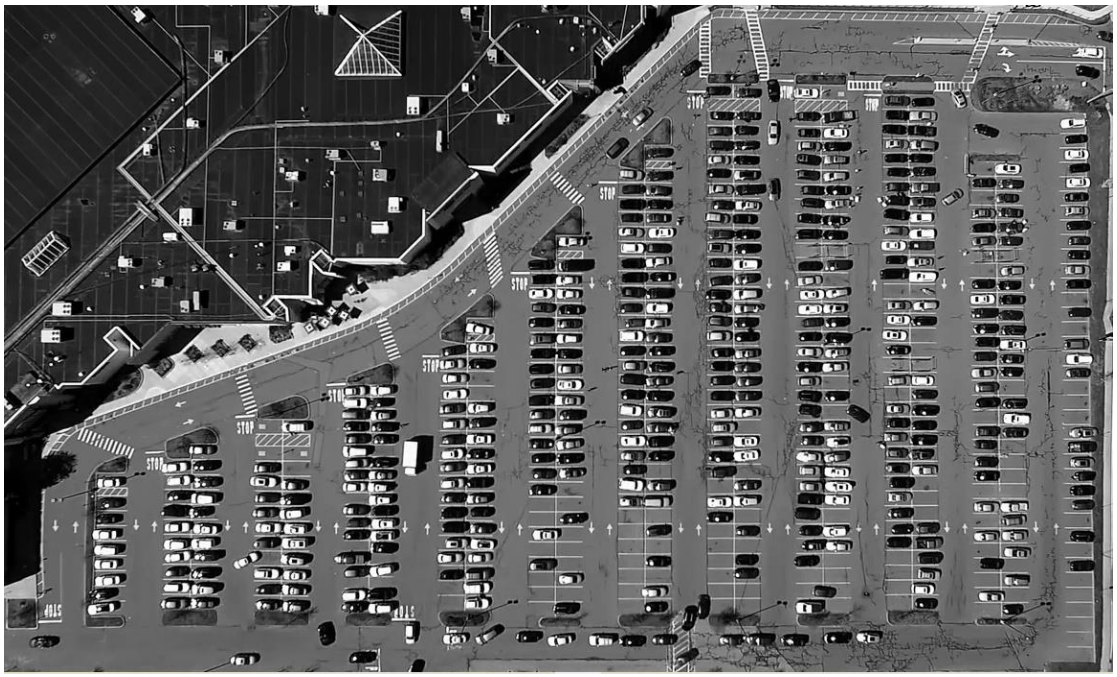

Fig. 6 RGB image of a frame converted to Grayscale image in video 3

2.4. Gaussian Blur

Gaussian blur is the application of a mathematical function to an image in order to blur it. This function can be useful in various cases such as in low light, the resulting image might have a lot of noise, to mute those noises Gaussian blur can be used [9]. It is also useful for reducing chromatic aberration.

The grayscale images in Fig. may include numerous noises or erroneous fluctuations in hue or brightness between pixels. Simply put, the large standard deviation of the pixels in these images shows that there is a lot of variation within pixel clusters. By generating a normal distribution for those pixel values, Because a image is two-dimensional, Gaussian blur uses two mathematical functions (one for the x-axis and one for the y) to create a third function, also known as a convolution. The third function reduces some of the randomness. The amount of smoothing depends on the blur radius that is selected. Each pixel will receive a new value that is based on a weighted average of the pixels in its immediate vicinity, with closer pixels receiving more weight than pixels farther away. The image is made fuzzier as a result of all this arithmetic.

In one dimension, the Gaussian function is shown in Equation [2]:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad \text{.... [2]}$$

Where σ is the standard deviation of the distribution. The distribution is assumed to have a mean of 0.

When working with images we need to use the two dimensional Gaussian function. This is simply the product of two 1D Gaussian functions (one for each direction) and is shown in Equation [3]:

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{.... [3]}$$

The Gaussian Blur works by using the 2D distribution as a point-spread function. This is achieved by convolving the 2D Gaussian distribution function with the image. We need to produce a discrete approximation to the Gaussian function. This theoretically requires an infinitely large convolution kernel, as the Gaussian distribution is non-zero everywhere. Fortunately the distribution has approached very close to zero at about three standard deviations from the mean. 99% of the distribution falls within 3 standard deviations. This means we can normally limit the kernel size to contain only values within three standard deviations of the mean.

Gaussian kernel coefficients are sampled from the 2D Gaussian function. Where σ is the standard deviation of the distribution. The distribution is assumed to have a mean of zero. We need to discretize the continuous Gaussian functions to store it as discrete pixels.

OpenCV-Python provides the cv2.GaussianBlur() function to apply Gaussian Smoothing on the Grayscale image. The syntax used here is,

cv2.GaussianBlur(src, ksize, sigmaX)

src is the Grayscaled image.

ksize is a Gaussian Kernel Size, [height, width]. The height and width should be odd and can have different values.

sigmaX is a kernel standard deviation along X-axis (horizontal direction).

Fig. 7 shows the images of grayscale images after Gaussian Blur is implemented in video 1.

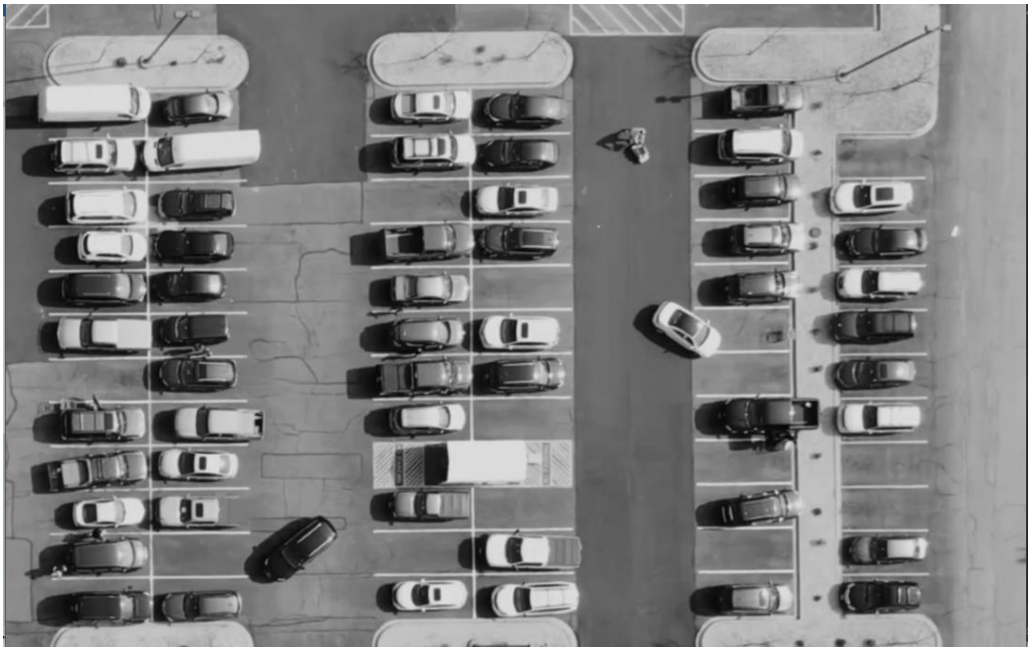

Fig. 7 Grayscale image has undergone Gaussian Blur in video 1

Fig. 8 shows the images of grayscale images after Gaussian Blur is implemented in video 2.

Fig. 8 Grayscale image has undergone Gaussian Blur in video 2

Fig. 9 shows the images of grayscale images after Gaussian Blur is implemented in video 3.

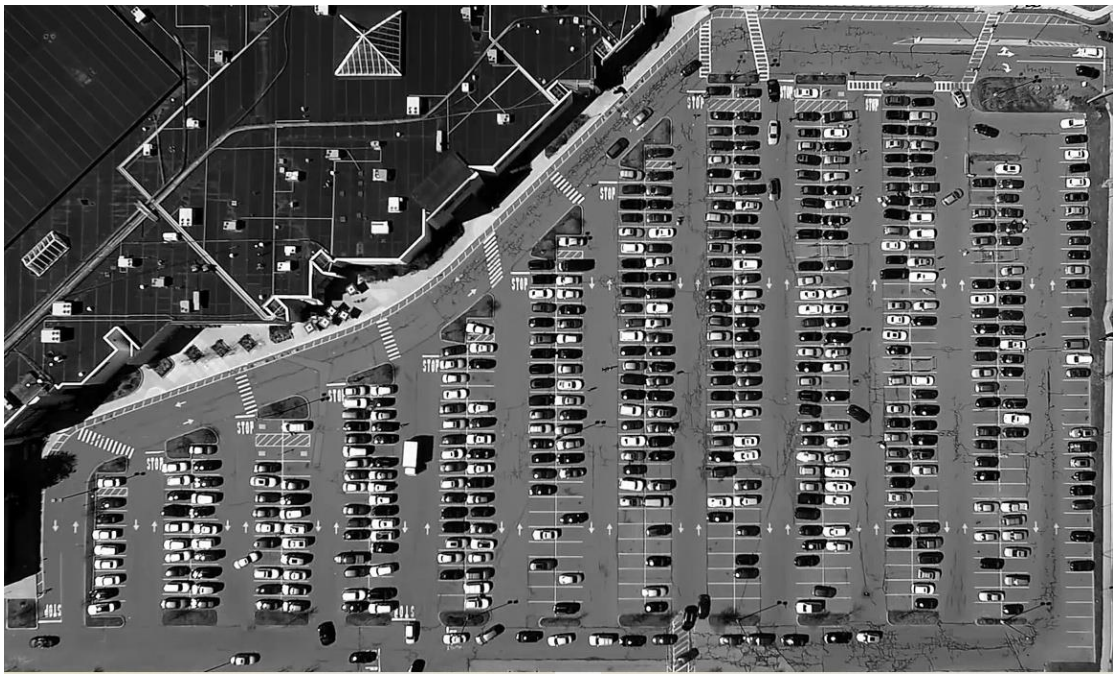

Fig. 9 Grayscale image has undergone Gaussian Blur in video 3

2.5. Adaptive Thresholding

Adaptive Thresholding is the method to determine the Threshold value, T for the local regions of our image. In simple terms, Adaptive Thresholding is used to separate desirable foreground image objects from the background based on the difference in pixel intensities of each region [11]. The common practice is to use either the Arithmetic mean or the Gaussian mean of the pixel intensities in each region. In the arithmetic mean, each

pixel in the vicinity equally contributes to the calculation of Threshold value, T. Additionally, in the Gaussian mean, pixel values farther away from the (x, y)-coordinate center of the region have a less effect on the calculation of Threshold value, T as a whole.

The general formula to compute *T* is shown in Equation [4].

$$T = mean(I_L) - C \ldots. [4]$$

mean is either the arithmetic or Gaussian mean.

I_L is the local sub-region of the image.

C is a constant which we can use to fine tune the threshold value, T.

In OpenCV, you can perform Adaptive threshold operation on an image using cv2.AdaptiveThreshold() function. The syntax used here is,

cv2.AdaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)

src is the Grayscaled Blurred image.

maxValue is the output Threshold value.

adaptiveMethod is a variable of integer type representing the adaptive method to be used. This will be either of the following two values ADAPTIVE_THRESH_MEAN_C or ADAPTIVE_THRESH_GAUSSIAN_C, we will be computing the weighted Gaussian mean over the blockSize's area, which gives larger weight to pixels closer to the center of the window.

blockSize is our pixel neighborhood size.

C is a constant, which I mentioned above, which lets us fine tune our Threshold Value.

Fig. 10 shows the images of Gaussian Blurred Grayscale images after Adaptive Threshold is implemented in video 1.

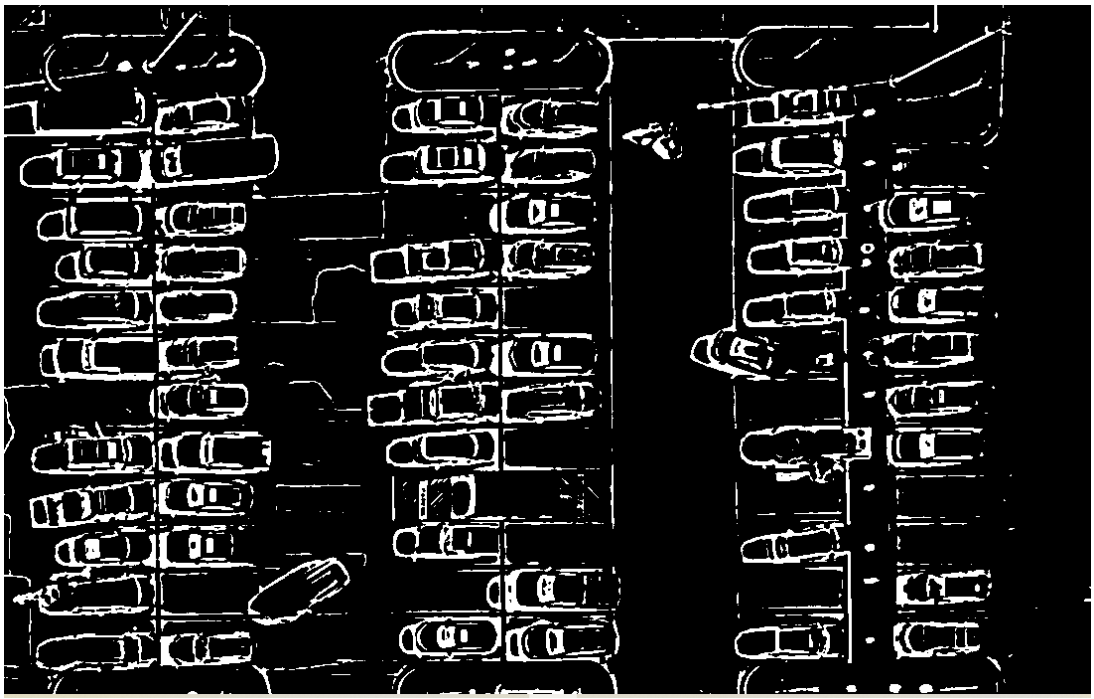

Fig. 10 Grayscale image converted to Inverse Binary in video 1

Fig. 11 shows the images of Gaussian Blurred Grayscale images after Adaptive Threshold is implemented in video 2.

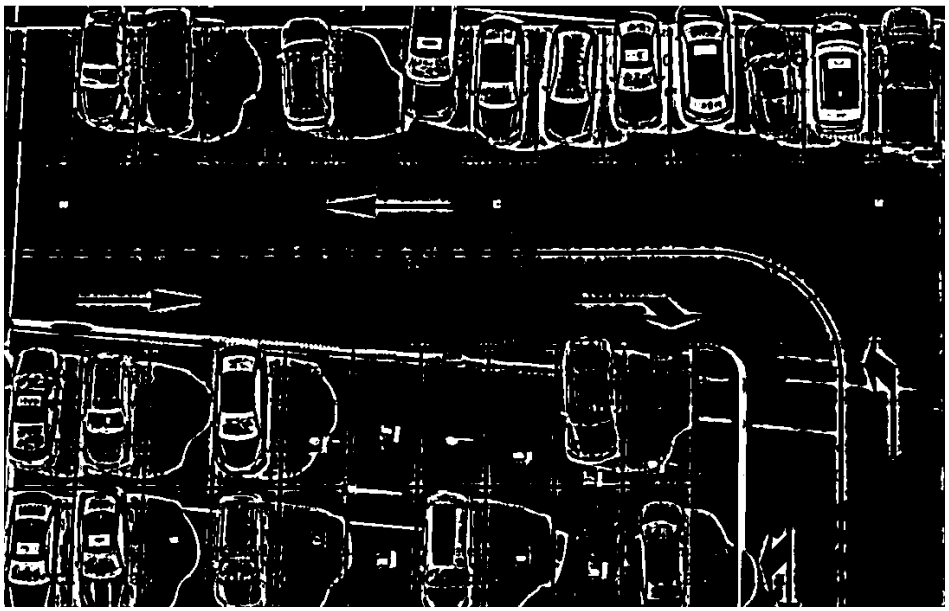

Fig. 11 Grayscale image converted to Inverse Binary in video 2

Fig. 12 shows the images of Gaussian Blurred Grayscale images after Adaptive Threshold is implemented in video 3.

Fig. 12 Grayscale image converted to Inverse Binary in video 3

2.6. Median Blur

The Median blur operation processes the edges while removing the noise. The median of all the pixels in the kernel area serves as the image's new center component in this case, since the median is robust to outliers, the salt-and-pepper noise will be less influential to the median [10].

When applying a median blur, we first define our kernel size. Then, taking into account all pixels in the vicinity of size K times K, where K is an odd integer. For the median, the kernel size must be square.

OpenCV provides the cv2.medianBlur() function to blur the image with a median kernel. The syntax used here is,

cv2.medianBlur(src, ksize)

src is the output threshold image.

ksize is a size object representing the size of the kernel.

Fig. 13 shows the images of output Threshold images after Median Blur is implemented in video 1

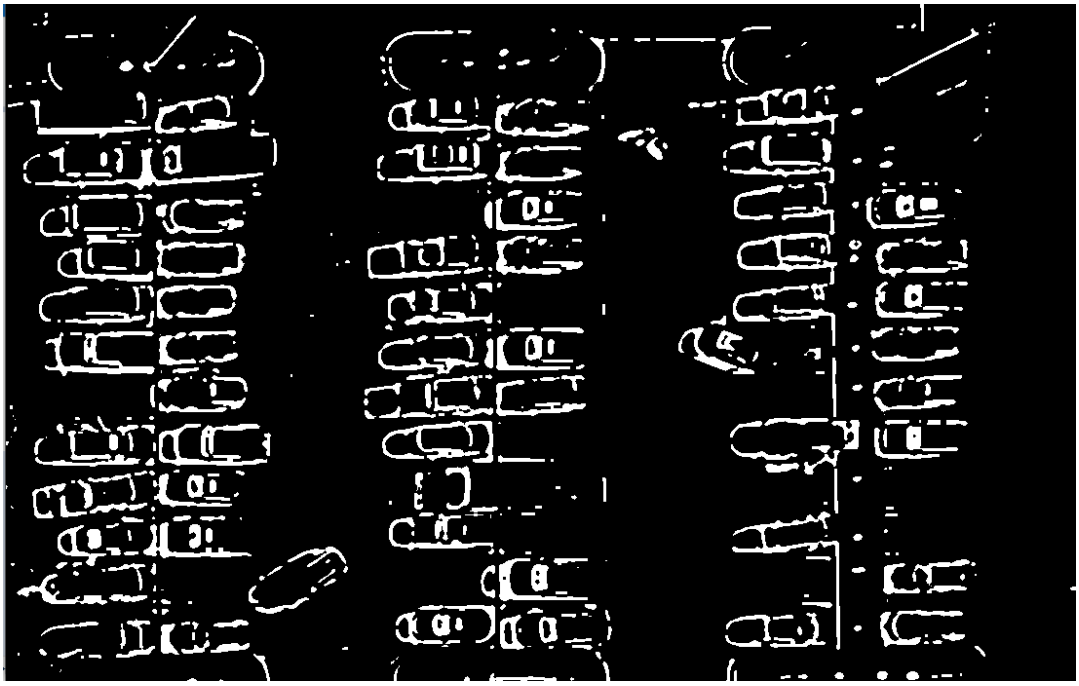

Fig. 13 Resulting image undergoes Median Blur for further noise reduction in video 1

Fig. 14 shows the images of Gaussian Blurred Grayscale images after Adaptive Threshold is implemented in video 2.

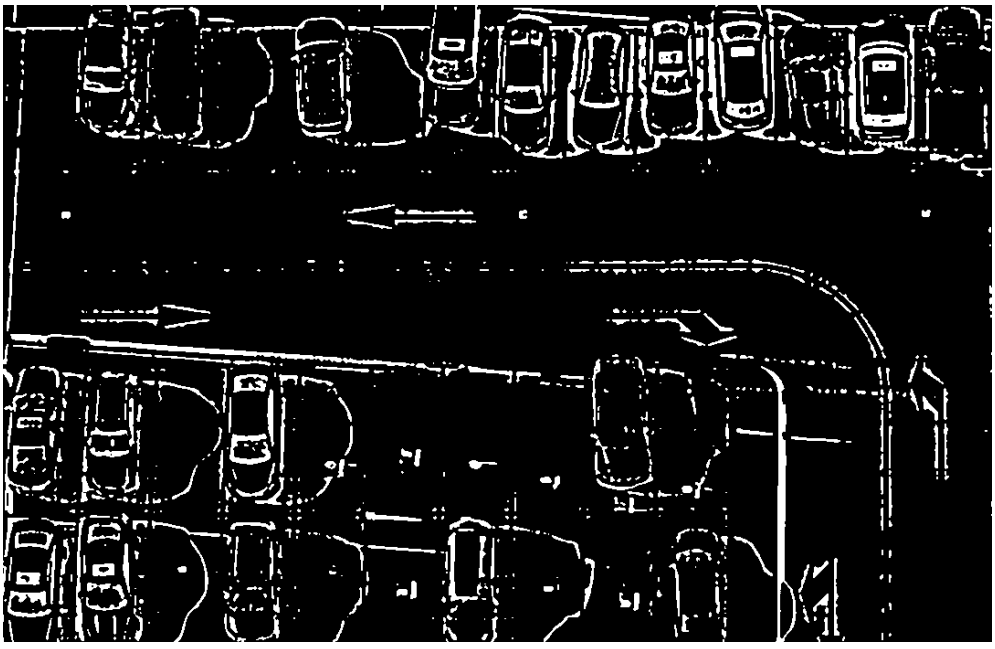

Fig. 14 Resulting image undergoes Median Blur for further noise reduction in video 2

Fig. 15 shows the images of Gaussian Blurred Grayscale images after Adaptive Threshold is implemented in video 3.

Fig. 15 Resulting image undergoes Median Blur for further noise reduction in video 3

2.7. Dilation

The two different kinds of morphological procedures are erosion and dilation. Morphological operations [13], as their name suggests, are a group of procedures that shape-process images. A "structural element" is created based on the input image that has been provided. Any of the two methods can be used to accomplish this. These are intended to reduce noise and smooth out flaws in order to improve the clarity of the image.

Dilation follows convolution with some kernel of a specific shape such as a square or a circle. This kernel has a centre, which is indicated by an anchor point. To calculate the maximum pixel value, this kernel is overlaid over the image. After computation, an anchor is set in the middle of the image. By using this method, the image size grows as the bright parts' surrounding areas expand in size.

OpenCV provides the cv2.dilate() function. The syntax used here is,

cv2.dilate(src, kernel, iterations)

src is the output Threshold image.

kernel is convolved with the image in a matrix of odd size .

iterations is a parameter that will take the number of iterations, which will determine how much to dilate the image.

Fig. 16 shows the images of median blurred images after erosion is implemented in video 1.

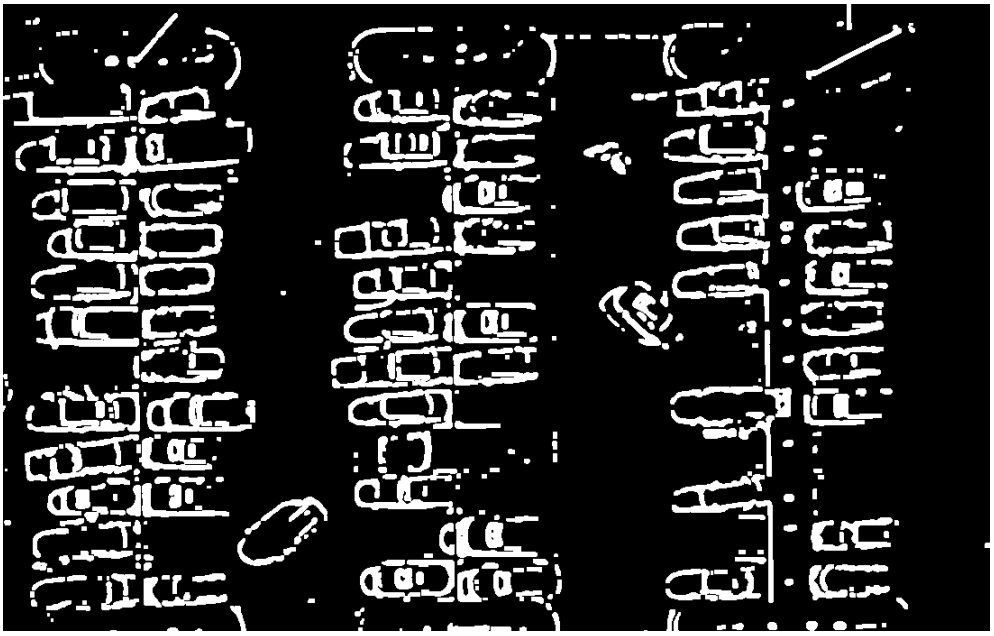

Fig. 16 Median Filtered image undergone dilation in video 1

Fig. 17 shows the images of median blurred images after erosion is implemented in video 2.

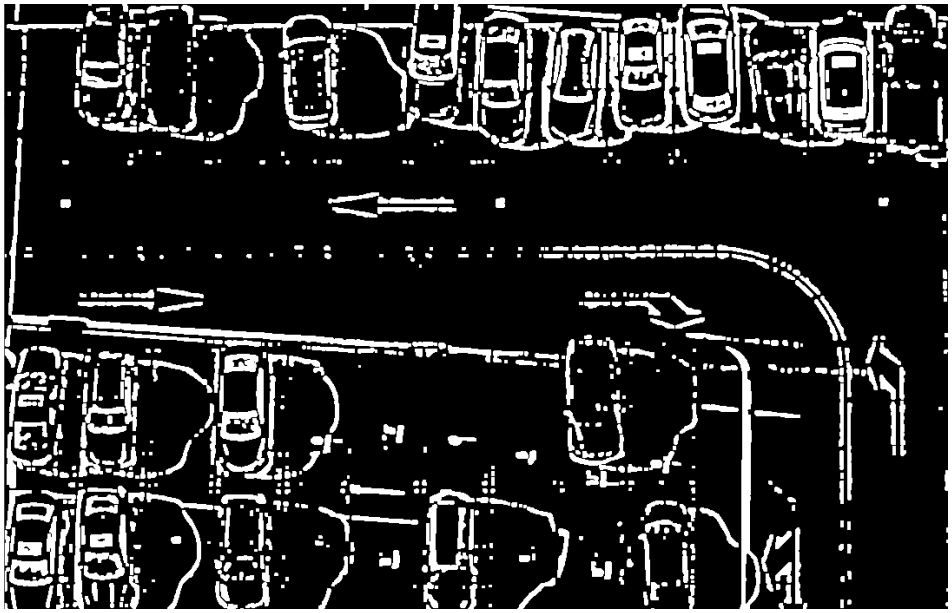

Fig. 17 Median Filtered image undergone dilation in video 2

Fig. 18 shows the images of median blurred images after erosion is implemented in video 3.

Fig. 18 Median Filtered image undergone dilation in video 3

3. ALGORITHM OF THE PROPOSED SYSTEM

The main steps of the proposed algorithm for vacant parking space detection and counting are:

i. The system will receive a live stream of parking lot footage from the camera. Here in our case we have used a recorded stream video.
ii. A frame of the parking lot's video is saved for ROI detection.
iii. Calibration is done in this frame by selecting each parking space as ROI. The ROI's dimension for each parking space is same.
iv. RGB images are converted to Grayscale images in real-time.
v. The Grayscale image is then filtered using Gaussian Blur.
vi. The resultant image is the converted to Binary image and then into inverse binary to get the car in white color and the rest of the background in black using Adaptive Thresholding.
vii. The resulting image is then filtered using Median Blur to remove the small tiny white dots in each ROI.
viii. The resulting converted image which is free of noise is then dilated using the dilation function to make the remaining white color a thick for differentiation.
ix. Number of nonzero pixels in each ROI is calculated, nonzero pixels denote white color.
x. The number of pixels to determine if a parking space is vacant or not is found out manually.

xi. If the number of nonzero pixels is less than the determining number of pixels for vacant spaces, then the parking spaces is shown in green rectangular box.

xii. Whereas If the number of nonzero pixels is greater than the determining number of pixels for vacant spaces, then the parking spaces is shown in Red rectangular box.

Fig. 19 shows the design flow of our system

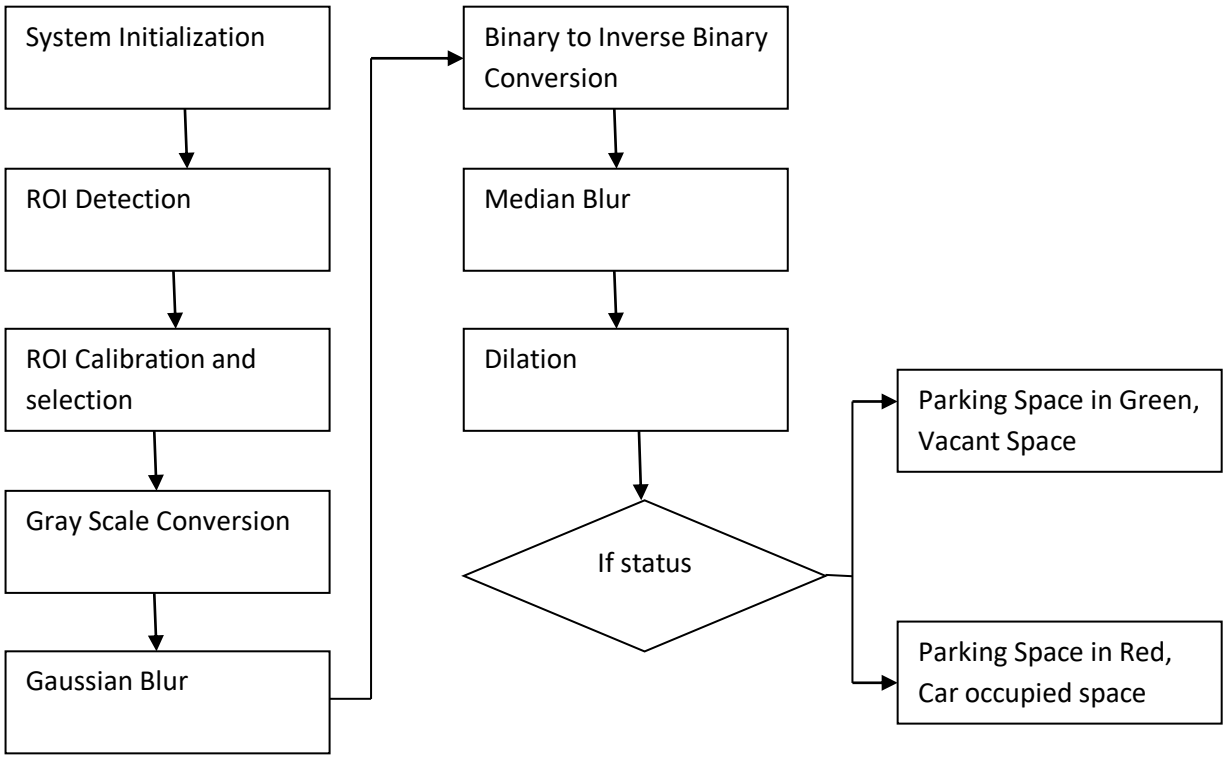

Fig. 19 Block Diagram of the proposed Parking system

4. RESULTS

The captured stream video's frames are processed to get the grayscale images and then it undergoes Gaussian Blur to remove noises and then it is converted into its corresponding binary and then inverse binary to get the car edges in white color and rest black. After that it is properly filtered again, in our algorithm we used median blur to remove the little bits of white dots that was present in the binary image and then applied dilation because sometimes these pixel values they might get a little bit thin so to make it thicker we use dilation so that it easy to differentiate between vacant spaces and car occupied spaces.

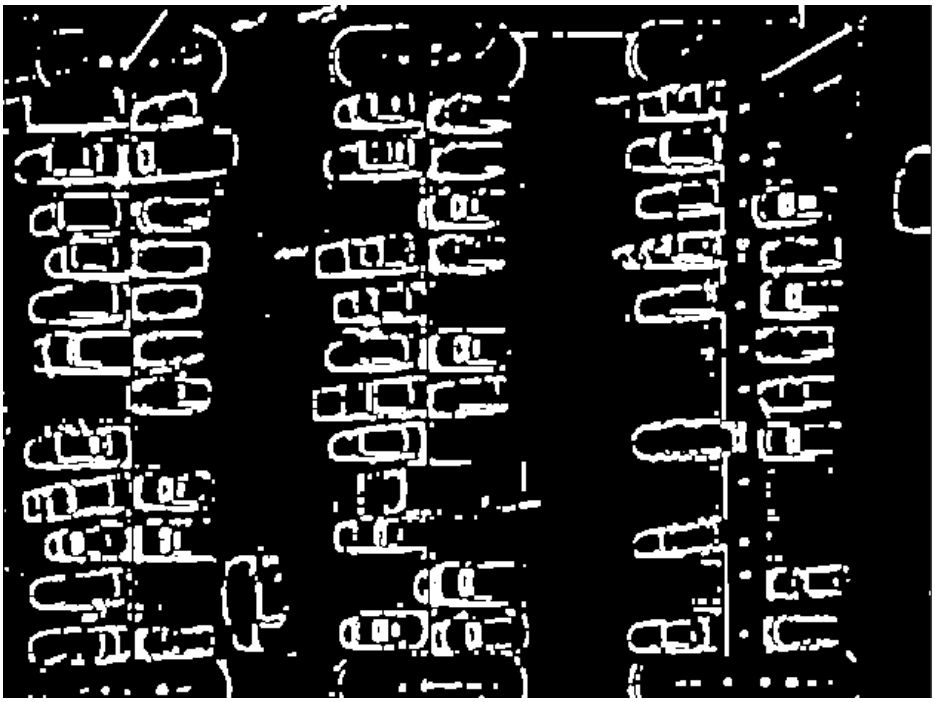

Fig. 20 Filtered Binary image of the parking area in video 1

Now each ROI is checked for pixel count, and try to find out the maximum pixel count shown in a vacant space and minimum pixel count shown in a occupied space and we take a value in between to determine whether the parking space is vacant or not. Generally, using a good camera will show significant difference in pixel count. Here in our video we found the maximum pixel count in one of the vacant space to be somewhere around 600 and minimum pixel count for a occupied space shows somewhere around 1100, so to be on the safe side we have taken 800 to determine if there is a car or not in the vacant space.

Here in the following images of a particular ROI undergoing processing in real-time and giving the desired output as shown with adequate pixel count. Fig. 21, 23, 25, 27 are the binary images of the particular ROI captured in screenshot mode with respect to time respectively. Fig. 22, 24, 26, 28 are the RGB output images of the ROI captured in screenshot mode at the same time as the binary image on the left.

Fig. 21 Binary image of a ROI with car

Fig. 22 Output RGB image with pixel count of the ROI

Fig. 23 Binary image 1 of car leaving the ROI

Fig. 24 Output RGB image 1 with pixel count of the ROI

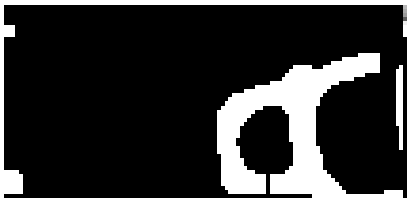

Fig. 25 Binary image 2 of car leaving the ROI

Fig. 26 Output RGB image 3 with pixel count of the ROI

Fig. 27 Binary image of the ROI after car left

Fig. 28 Output RGB image with pixel count of the ROI after car left

4.1. Output Window after Video Image Processing

The system shows the output window with car occupied parking spaces in red and vacant parking spaces in green with pixel count at the bottom left corner of each ROI. At the top left corner of the output window there is shown the total number of vacant spaces available for parking. In video 1 to achieve the following we have first selected ROI for each parking space with the following dimensions, width is 107 and height is 48 and then saved the image where all the parking spaces are picked, then in the main file we convert the frames to grayscale then use Gaussian Blur with kernel size, [3, 3] and sigmax value 1, then we use Adaptive thresholding with maxvalue of 255, blocksize value 25 and C value 16, then we use Median blur with kernel size 5 and lastly

we use dilation with kernel size, [3,3] and number of iterations is 1 to get the best results. Now to check the nonzero pixel count in each ROI and determine the minimum value of pixel count below which vacancy will be detected is 800. A typical output window is shown in Fig. 29

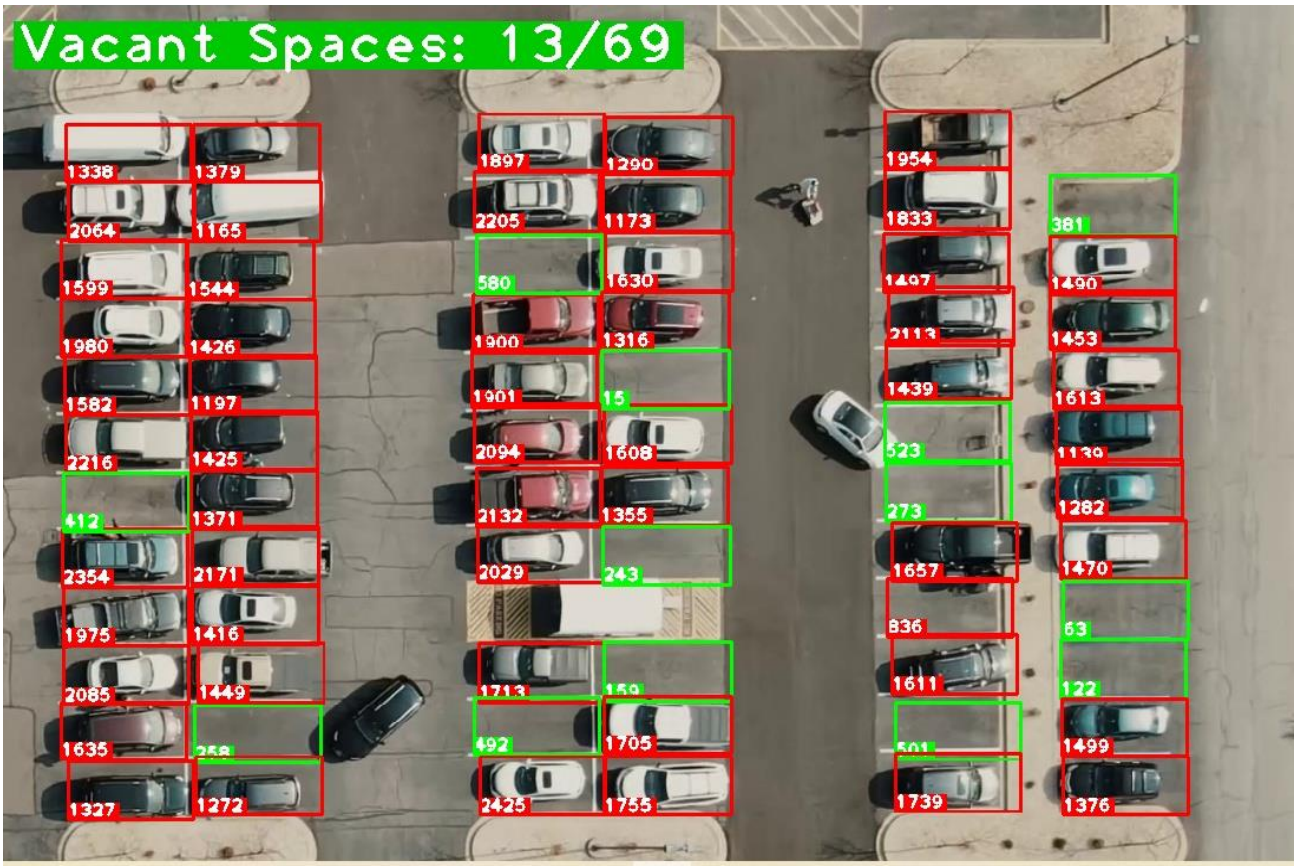

Fig. 29 Output Window in Pycharm which displays the real-time status of the parking spaces in the parking area in video 1

The system shows the output window with car occupied parking spaces in red and vacant parking spaces in green with pixel count at the bottom left corner of each ROI. At the top left corner of the output window there is shown

the total number of vacant spaces available for parking. In video 1 to achieve the following we have first selected ROI for each parking space with the following dimensions, width is 75 and height is 160 and then saved the image where all the parking spaces are picked, then in the main file we convert the frames to grayscale then use Gaussian Blur with kernel size, [3,3] and sigmax value 2, then we use Adaptive thresholding with maxvalue of 255, blocksize value 25 and C value 16, then we use Median blur with kernel size 3 and lastly we use dilation with kernel size, [3,3] and number of iterations is 1 to get the best results. Now to check the nonzero pixel count in each ROI and determine the minimum value of pixel count below which vacancy will be detected is 1800. A typical output window is shown in Fig. 30.

Fig. 30 Output Window in Pycharm which displays the real-time status of the parking spaces in the parking area in video 2

The system shows the output window with car occupied parking spaces in red and vacant parking spaces in green with pixel count at the bottom left

corner of each ROI. At the top left corner of the output window there is shown the total number of vacant spaces available for parking. In video 1 to achieve the following we have first selected ROI for each parking space with the following dimensions, width is 28 and height is 14 and then saved the image where all the parking spaces are picked, then in the main file we convert the frames to grayscale then use Gaussian Blur with kernel size, [1,1] and sigmax value 0, then we use Adaptive thresholding with maxvalue of 255, blocksize value 37 and C value 28, then we use Median blur with kernel size 1 and lastly we use dilation with kernel size, [1,1] and number of iterations is 0 to get the best results. Now to check the nonzero pixel count in each ROI and determine the minimum value of pixel count below which vacancy will be detected is 64. A typical output window is shown in Fig. 31.

. Fig. 31. Output Window in Pycharm which displays the real-time status of the parking spaces in the parking area in video 3

4.2. Results of the Proposed System

The images used for testing purposes show three parking lots with 69, 34 and 546 parking spaces, respectively, and were taken from fixed cameras. Video 1 has a total number of 679 frames, video 2 has a total of 330 frames and video 3 has a total of 1396 frames.

For assessing the rate of parking spaces detection, the tests were performed for each video. One can find below the results from different videos with the total number of frames.

In order to evaluate the system performance, the following rates were calculated: false positive (FP), false negative (FN), true positive (TP), true negative (TN), and accuracy (ACC).

$$\text{ACC} = \frac{TP+TN}{FP+FN+TP+TN}$$

where FP is the total number of available parking spaces labeled as occupied, FN is the total number for occupied parking spaced labeled as available, TP is the total number of available spaces correctly labeled, and TN the total number of occupied spaces correctly labeled.

Table 1. Accuracy results of videos using the proposed image processing algorithm.

Video File	Total No. of Frames tested	FP (%)	FN (%)	TP(%)	TN (%)	ACC (%)
Video 1	679	0.202	0	18.755	81.043	99.798
Video 2	330	0.5	0.29	48.529	50.711	99.24
Video 3	1396	1.074	0.106	18.025	80.795	98.82

4.3. Difference between the existing approach and the proposed approach:

Existing Approach	Proposed Approach
Images are taken by camera when a vehicle enters or exit the parking lot.	Real-time image video processing is done.
Slot numbers needs to be manually marked in parking spaces for system	Slot numbers are not required to be marked

to work	
Binary image contains only nonzero pixels of slot numbers	Binary image contains nonzero pixels of car edges for detection
Optical Character Recognition is used for empty slot detection	Optical Character Recognition is not required
Detection accuracy depends completely on OCR	Detection accuracy depends on proper filtering of nonzero pixels and gives more than 98% accuracy

The video files used for processing and testing are downloaded from the internet. The following source destination of the video files is given below:

Video File 1: https://youtu.be/5G_wMSwrLXc

Video File 2: https://www.youtube.com/watch?v=uVZQSUXYqSQ

Video File 3: https://www.youtube.com/watch?v=yojapmOkIfg&ab

5. CONCLUSION AND FUTURE SCOPE

5.1. Conclusion

A vision-based car parking lot management system is proposed in this paper. The vision-based method makes it possible to manage large area by just using camera from a top-down view. It is consistent in detecting vacant parking spaces due to decrease in nonzero pixel count when a car leaves the parking space. This image processing based empty parking lot identification has made the system simple as well as cheap and efficient and can easy-installed because of the simple equipments.

5.2. Future Scope

Although our research has yielded some results, there are still many places that need to improve. Our future efforts will expect to make a breakthrough in the following areas.

1. The test results have demonstrated that the current algorithm is not ideal in terms of false alarm rates, so one of our future research priorities will be to figure out how to successfully reduce the false alarm rate.

2. Environmental conditions can affect video detecting techniques. Although the current algorithms are somewhat self-adaptive, there is still potential for development. We will work to reduce how much the environment affects the accuracy of the detection.

3. Give full play to the advantage of the hardware algorithm. The hardware algorithm has a speed advantage that the software algorithm cannot be compared with. Further strengthen the advantage is not only for speed considerations, but also to reduce the burden of processor, thus can improve overall system performance.

6. REFERENCES

[1] Athira A. , Lekshmi S. , Pooja Vijayan and Boby Kurian, “Smart Parking System Based On Optical Character Recognition” in IEEE Transactions on Trends in Electronics and Informatics (ICOEI 2019).

[2] Zhang Bin, Jiang Dalin, Wang Fang and Wan Tingting, “Smart Parking System Based On Optical Character Recognition” in IEEE Transactions on Trends in Electronics and Informatics (ICOEI 2019).

[3] T. Lin, H. Rivano and F. Le Mouël, "A Survey of Smart Parking Solutions," in IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 12, pp. 3229-3253, Dec. 2017.

[4] Ms.Sayanti Banerjee, Ms. Pallavi Choudekar, Prof .M.K.Muju, “Real Time Car Parking System Using Image Processing” in IEEE Transactions

[5] R. Yusnita, Fariza Norbaya, and Norazwinawati Basharuddin, “Intelligent Parking Space Detection System Based on Image Processing” in International Journal of Innovation, Management and Technology, Vol. 3, No. 3, June 2012

[6] Y. Ji, W. Guo, P. Blythe, D. Tang, and W. Wang, “Understanding drivers’ perspective on parking guidance information,” IET Intell. Transport Syst., vol. 8, no. 4, pp. 398–406, June 2014.

[7] H. Guan, L. Liu, and M. Liao, “Approach for planning of parking guidance and information system,” J. Highway Transport. Res. Develop., vol. 1, pp. 034, 2003.

[8] F. Caicedo, “Real-time parking information management to reduce search time, vehicle displacement and emissions,” Transport. Res. D, vol. 15, no. 4, pp. 228–234, 2010.

[9] E. S. Gedraite and M. Hadad, "Investigation on the effect of a Gaussian Blur in image filtering and segmentation," Proceedings ELMAR-2011, 2011, pp. 393-396.

[10] Sin Hoong Teoh and Haidi Ibrahim, “Median Filtering Frameworks for Reducing Impulse Noise from Grayscale Digital Images: A Literature Survey” International Journal of Future Computer and Communication, Vol. 1, No. 4, December 2012.

[11] P. Roy, S. Dutta, N. Dey, G. Dey, S. Chakraborty and R. Ray, "Adaptive thresholding: A comparative study," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014, pp. 1182-1186, doi: 10.1109/ICCICCT.2014.6993140.

[12] Francis Huo Yen Chan, F. K. Lam, and Hui Zhu. "Adaptive Thresholding by Variational Method." IEEE Transactions on Image Processing 7:3 (1998), 468–473.

[13] Mary L. Comer, Edward J. Delp III, "Morphological operations for color image processing," J. Electron. Imag. 8(3) (1 July 1999)

[14] Rashid, M. M., Musa, A., Rahman, M. A., Farahana, N.: Automatic parking management system and parking fee collection based on number plate recognition. International Journal of Machine Learning and Computing, vol. 2, no. 2, p. 94 (2012)

[15] Jian, M. S., Yang, K. S., and Lee, C. L. Modular RFID parking management system based on existed gate system integration. WSEAS Transactions on Systems, vol. 7, no. 6, pp. 706–716 (2008)

[16] Tsiropoulou, E. E., Baras, J. S., Papavassiliou, S., Sinha, S., RFID-based smart parking management system. CyberPhysical Systems, vol. 3, no. 1–4, pp. 22–41 (2017)

[17] Wei, L., Wu, Q., Yang, M., Ding, W., Li, B., and Gao, R., Design and implementation of smart parking management system based on rfid and internet. In: International Conference on Control Engineering and Communication Technology, pp. 17–20 (2012)

[18] Bi, Y., Sun, L., Zhu, H., Yan, T., Luo, Z., A parking management system based on wireless sensor network. Acta Automatica Sinica, vol. 32, no. 6, p. 968 (2006)

[19] Vera-Gómez, J. A., Quesada-Arencibia, A., García, C. R., Suárez Moreno, R., Guerra Hernández, F., An intelligent parking management system for urban areas. Sensors, vol. 16, no. 6, p. 931 (2016)

[20] Gandhi, B. K., Rao, M. K., A prototype for IoT based car parking management system for smart cities. Indian Journal of Science and Technology, vol. 9, no. 17, pp. 1–6 (2016)

[21] Sadhukhan, P., An IoT-based E-parking system for smart cities. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1062–1066 (2017)

Appendix:

Code Snippets:

Video File 1:

Frames.py

```
import cv2

vidcap = cv2.VideoCapture('CarPark.mp4')
success,image = vidcap.read()
count = 0

while success:
cv2.imwrite("frame%d.png" % count, image)
success,image = vidcap.read()
print('Read a new frame: ', success)
count += 1
```

ParkingSpacePicker.py

```
import cv2
import pickle

img = cv2.imread('carParkImg.png')



width, height = 107, 48

try:
        with open('CarParkPos', 'rb') as f:
        posList = pickle.load(f)
except:
        posList = []

def mouse_click(events, x, y, flags, params):
        if events == cv2.EVENT_LBUTTONDOWN:
                posList.append((x, y))
        if events == cv2.EVENT_RBUTTONDOWN:
                for i, pos in enumerate(posList):
                        x1, y1 = pos
                        if x1 < x < x1 + width and y1 < y < y1 + height:
                                posList.pop(i)
        with open('CarParkPos', 'wb') as f:
        pickle.dump(posList, f)
```

```
while True:
    img = cv2.imread('carParkImg.png')


    for pos in posList:
        cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), (255, 0,
    255), 2)

cv2.imshow("Image", img)
cv2.setMouseCallback("Image", mouse_click)
cv2.waitKey(1)
```

main.py

```
import cv2
import pickle
import cvzone
import numpy as np

cap = cv2.VideoCapture('carPark.mp4')

with open('CarParkPos', 'rb') as f:
```

```
        posList = pickle.load(f)

width, height = 107, 48



def check_parking_space(imgPro):

        spacecounter = 0

        for pos in posList:
                x, y = pos

                imgCrop = imgPro[y:y + height, x:x + width]
                count = cv2.countNonZero(imgCrop)
                cvzone.putTextRect(img, str(count), (x, y+height-3), scale= 1,
                thickness= 2, offset= 0)

                if count < 800:
                        color = (0, 255, 0)
                        thickness = 5
                        spaceCounter+=1
                        cvzone.putTextRect(img, str(count), (x, y + height - 3),
                        scale=1, thickness=2, offset=1, colorR=(0, 255, 0))
                else:
                        color = (0, 0, 255)
                        thickness = 2
```

```
                cvzone.putTextRect(img, str(count), (x, y + height - 3),
                scale=1, thickness=2, offset=1, colorR=(0, 0, 255))
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color,
            thickness)
        cvzone.putTextRect(img, f'Vacant Spaces: {spaceCounter}/{len(posList)}',
        (15, 2 + height - 3), scale=3, thickness=4, offset=5, colorR=(0, 200, 0))

while True:
        if cap.get(cv2.CAP_PROP_POS_FRAMES) ==
        cap.get(cv2.CAP_PROP_FRAME_COUNT):
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        success, img = cap.read()
        imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
        imgThreshold = cv2.adaptiveThreshold(imgBlur, 255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 25, 16)
        imgMedian = cv2.medianBlur(imgThreshold, 5) kernel = np.ones((3, 3),
        np.uint8)
         imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)

        check_parking_space(imgDilate)

        cv2.imshow("Image", img)
        cv2.waitKey(10)
```

Video File 2:

ParkingSpacePicker.py

```
import cv2
import pickle

img = cv2.imread('carParkImg2.png')

width, height = 75, 160

try:
        with open('CarParkPos2', 'rb') as f:
        posList = pickle.load(f)
except:
        posList = []

def mouse_click(events, x, y, flags, params):
        if events == cv2.EVENT_LBUTTONDOWN:
                posList.append((x, y))
        if events == cv2.EVENT_RBUTTONDOWN:
                for i, pos in enumerate(posList):
                        x1, y1 = pos
                        if x1 < x < x1 + width and y1 < y < y1 + height:
                                posList.pop(i)
```

```
        with open('CarParkPos2', 'wb') as f:
        pickle.dump(posList, f)

while True:
        img = cv2.imread('carParkImg2.png')


        for pos in posList:
                cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), (255, 0,
        255), 2)

cv2.imshow("Image", img)
cv2.setMouseCallback("Image", mouse_click)
cv2.waitKey(1)
```

main.py

```
import cv2
import pickle
import cvzone
import numpy as np

cap = cv2.VideoCapture('carPark2.mp4')
```

```
with open('CarParkPos2', 'rb') as f:
        posList = pickle.load(f)

width, height = 75, 160



def check_parking_space(imgPro):

        spacecounter = 0

        for pos in posList:
                x, y = pos

                imgCrop = imgPro[y:y + height, x:x + width]
                count = cv2.countNonZero(imgCrop)
                cvzone.putTextRect(img, str(count), (x, y+height-3), scale= 1,
                thickness= 2, offset= 0)

                if count < 1800:
                        color = (0, 255, 0)
                        thickness = 4
                        spaceCounter+=1
                else:
                        color = (0, 0, 255)
                        thickness = 2
```

```
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color,
            thickness)
        cvzone.putTextRect(img, f'Vacant Spaces: {spaceCounter}/{len(posList)}',
        (25, 70 + height - 3), scale=3, thickness=4, offset=5, colorR=(0, 200, 0))

while True:
        if cap.get(cv2.CAP_PROP_POS_FRAMES) ==
        cap.get(cv2.CAP_PROP_FRAME_COUNT):
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        success, img = cap.read()
        imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
        imgThreshold = cv2.adaptiveThreshold(imgBlur, 255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 25, 16)
        imgMedian = cv2.medianBlur(imgThreshold, 5) kernel = np.ones((3, 3),
        np.uint8)
         imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)

        check_parking_space(imgDilate)

        cv2.imshow("Image", img)
        cv2.waitKey(50)
```

Video File 3:

ParkingSpacePicker.py

```
import cv2
import pickle

img = cv2.imread('carParkImg3.png')

width, height = 28,14

try:
	with open('CarParkPos3', 'rb') as f:
	posList = pickle.load(f)
except:
	posList = []

def mouse_click(events, x, y, flags, params):
	if events == cv2.EVENT_LBUTTONDOWN:
		posList.append((x, y))
	if events == cv2.EVENT_RBUTTONDOWN:
		for i, pos in enumerate(posList):
			x1, y1 = pos
			if x1 < x < x1 + width and y1 < y < y1 + height:
				posList.pop(i)
```

```
        with open('CarParkPos3', 'wb') as f:
        pickle.dump(posList, f)

while True:
        img = cv2.imread('carParkImg3.png')


        for pos in posList:
                cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), (255, 0,
        255), 2)

cv2.imshow("Image", img)
cv2.setMouseCallback("Image", mouse_click)
cv2.waitKey(1)
```

main.py

```
import cv2
import pickle
import cvzone
import numpy as np
```

```
cap = cv2.VideoCapture('carPark3.mp4')

with open('CarParkPos2', 'rb') as f:
        posList = pickle.load(f)

width, height = 28,14



def check_parking_space(imgPro):

        spacecounter = 0

        for pos in posList:
                x, y = pos

                imgCrop = imgPro[y:y + height, x:x + width]
                count = cv2.countNonZero(imgCrop)
                cvzone.putTextRect(img, str(count), (x, y+height-3), scale= 0.6,
                thickness= 1, offset= 0)

                if count < 64:
                        color = (0, 255, 0)
                        thickness = 1
                        spaceCounter+=1
                else:
                        color = (0, 0, 255)
```

```
                thickness = 1
            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color,
            thickness)
        cvzone.putTextRect(img, f'Vacant Spaces: {spaceCounter}/{len(posList)}',
        (25, 20 + height - 3), scale=3, thickness=4, offset=5, colorR=(0, 200, 0))

while True:
        if cap.get(cv2.CAP_PROP_POS_FRAMES) ==
        cap.get(cv2.CAP_PROP_FRAME_COUNT):
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
        success, img = cap.read()
        imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgBlur = cv2.GaussianBlur(imgGray, (1, 1), 0)
        imgThreshold = cv2.adaptiveThreshold(imgBlur, 255,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 37, 28)
        imgMedian = cv2.medianBlur(imgThreshold, 1)
        kernel = np.ones((1, 1), np.uint8)
         imgDilate = cv2.dilate(imgMedian, kernel, iterations=0)

        check_parking_space(imgDilate)

        cv2.imshow("Image", img)
        cv2.waitKey(10)
```