Digital Image Processing

Count Number of Empty Car Parking Lots

Team: **Sesh**

Smruti Biswal - 2020112011
Eshika Khandelwal - 2020114018
Srujana Vanka - 2020102005
Shreeya Singh - 2020102011

Github repo link

Problem statement

There are some reasons why counting the number of empty car parking lots is developed. The problems that have been identified are stated below:

1. The driver needs some relevant information before entering the parking lot such as the current available parking spaces in the parking lot.

2. There is a current system used in parking lots is based on installing a certain sensor on each division;
In the method with the sensor, the cost rises as the number of parking divisions increases.

3. Driver will have to drive through all parking divisions just to find an available parking space - this is time-taking.

Objective

Given an orthogonal , top-view of a car parking lots, detect and count the number of empty spots. The parking lots can have shadows in them.

Objectives of this project are to:

1. Capture and detect the existence of vehicles at the parking lot using image processing techniques.

2. Count, and display available parking spaces.

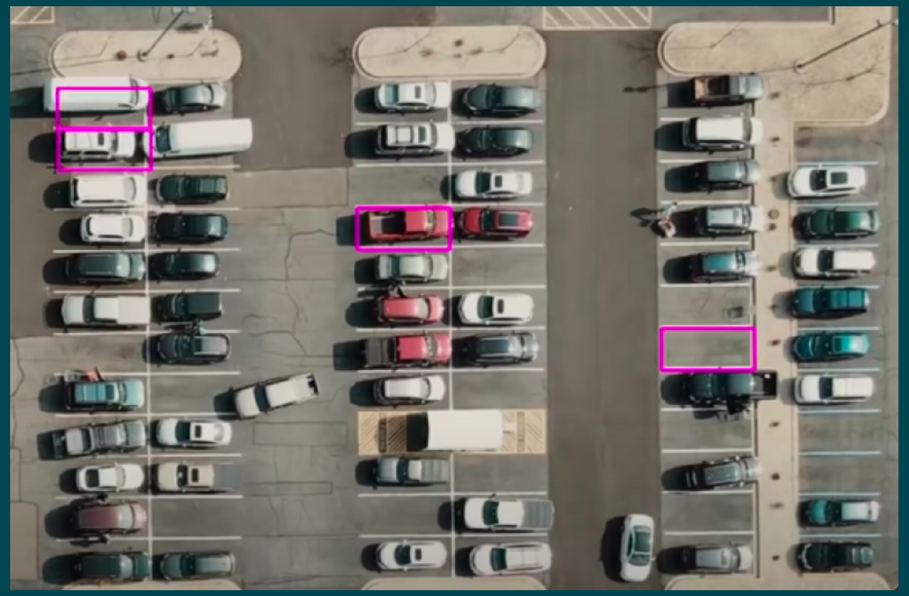
Approach

Method 1

- Tried starting by running a for loop from the top and marking our region of interest, that is, the parking space. But this causes errors because not all parking divisions might be of the same dimension, or there might be places that are not for parking (refer to the given input image).
- Hence, to avoid such errors, based on the input image we will **manually mark our region of interest.**

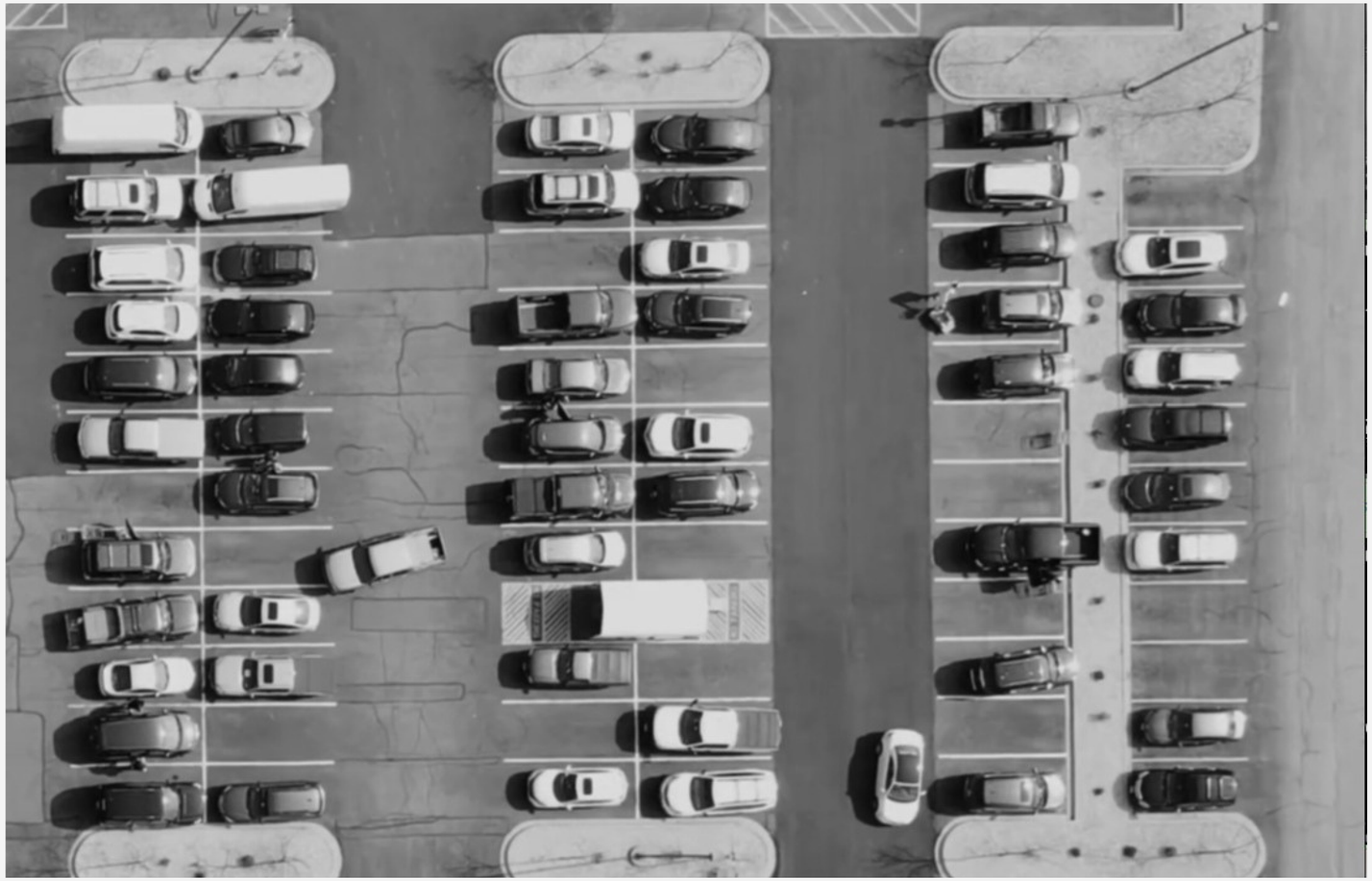
Input testcase 1

- Our region of interest, that is, parking spaces, are marked manually. This is done using a mouse-click function.

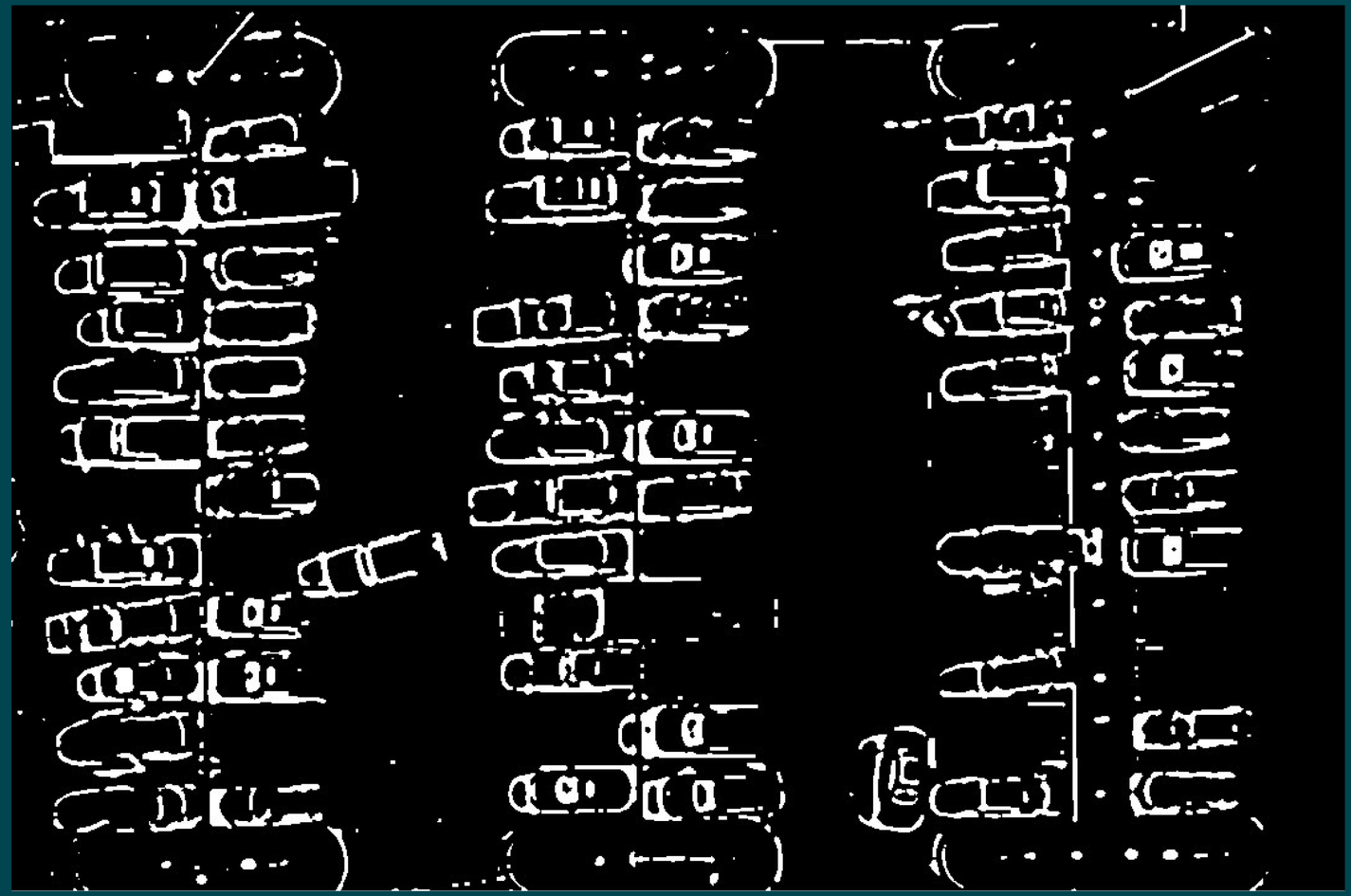


- The parking spaces are marked and these positions are stored as a list separately.

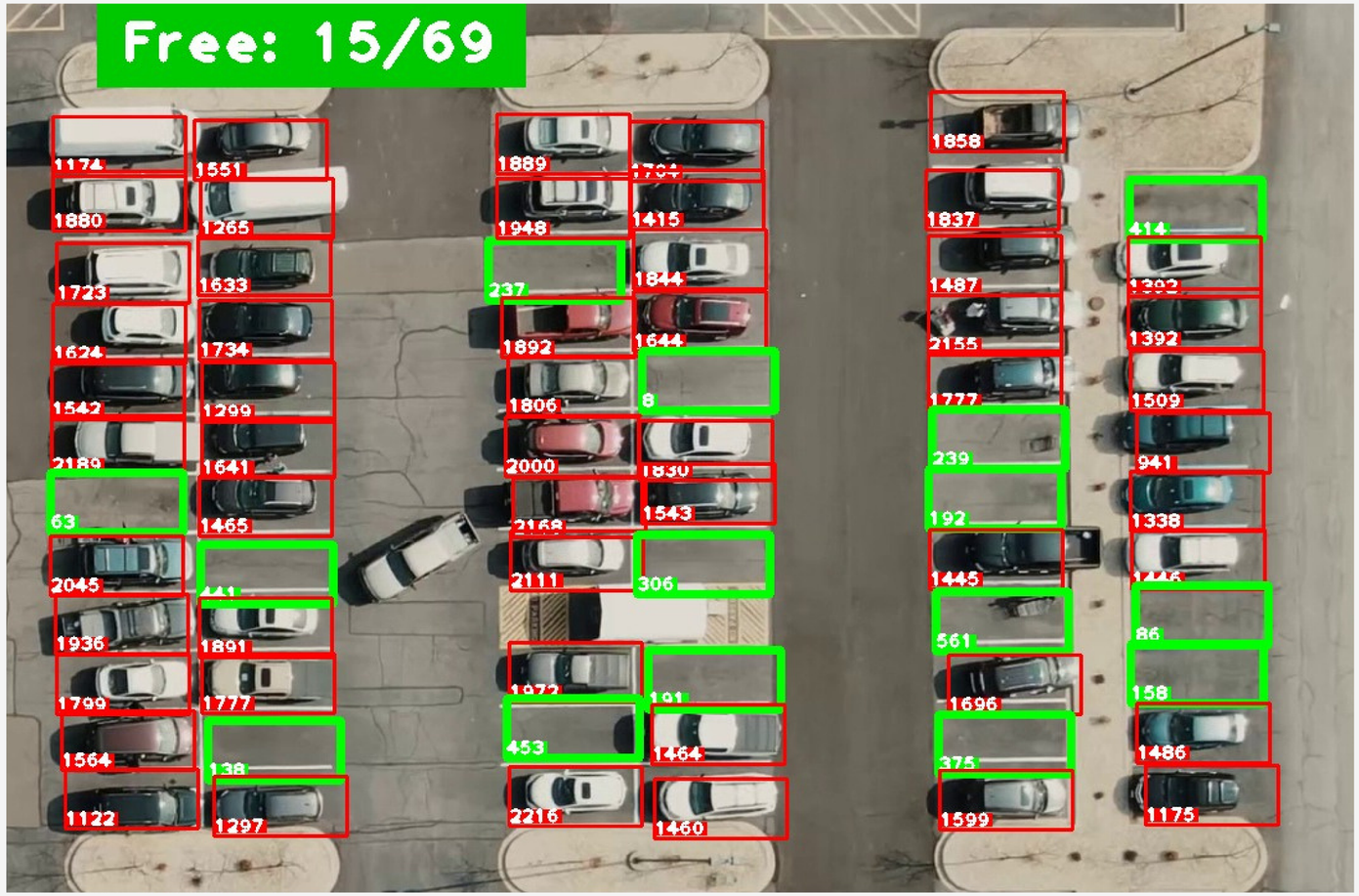
- Once we have our parking spaces marked, we check if each of these parking spaces is occupied or not. This is done by checking the **pixel count** of each parking space.
- The image is converted to **grayscale and subjected to binary blur**.



- **Adaptive thresholding** is performed over this blurred image to separate desirable foreground image objects from the background based on the difference in pixel intensities of each region.
- Salt and pepper noise is observed, hence **median filter is applied**.
- To make the edges and lines thicker and visible distinctly, the **filtered image is dilated**.



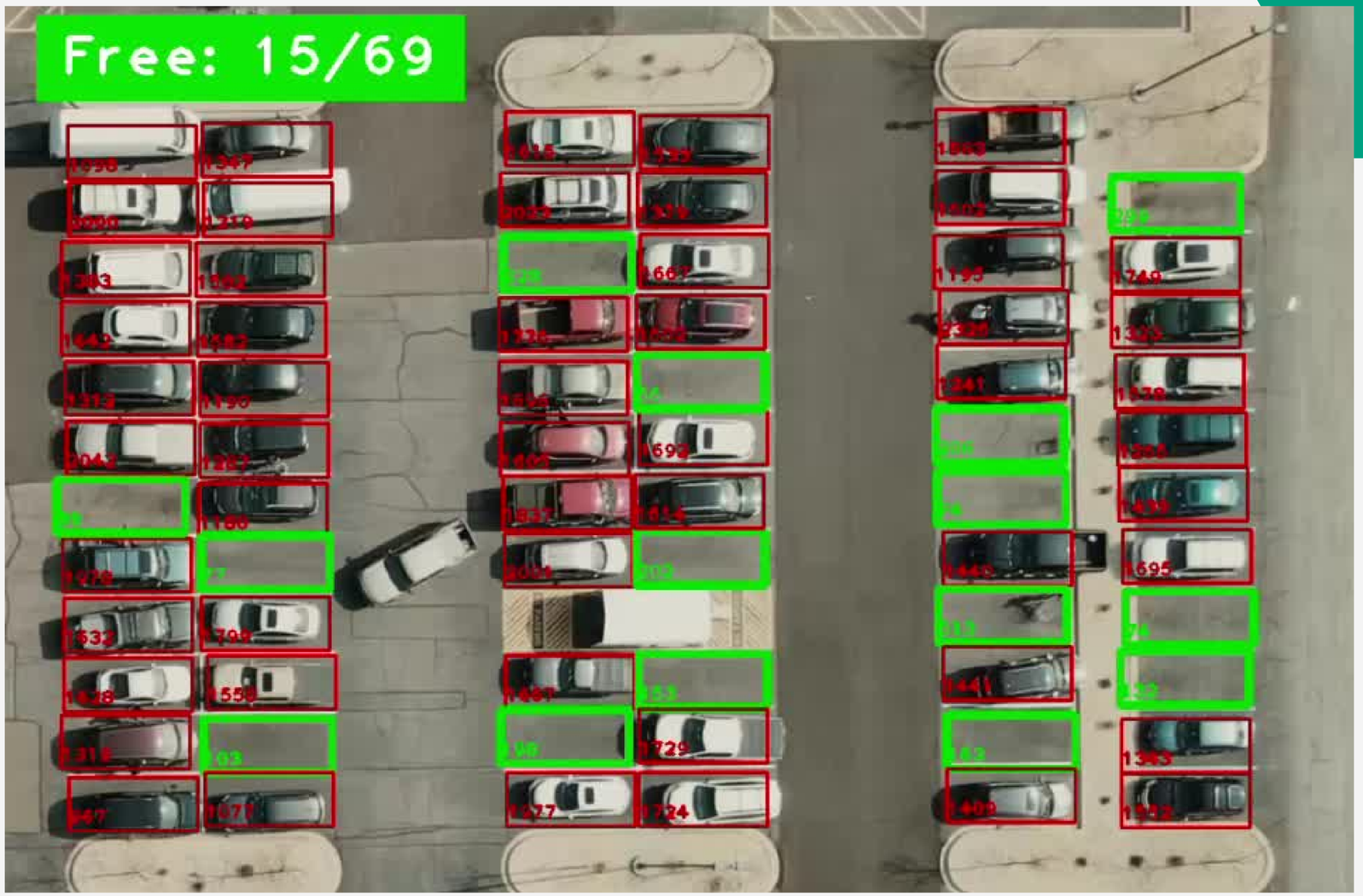
- The **pixel count** of each parking space is calculated from the dilated image.
- Parking spaces that have a low pixel count, ie, the number of curves is less, are observed to be empty.
- Spaces that have a pixel count between **200-900 are observed as empty parking spaces**. These are detected and marked as free parking spots.



This approach can also be extended to a video captured from a drone.

Video input for same parking lot:

Detecting and counting number of cars from video captured from drone:

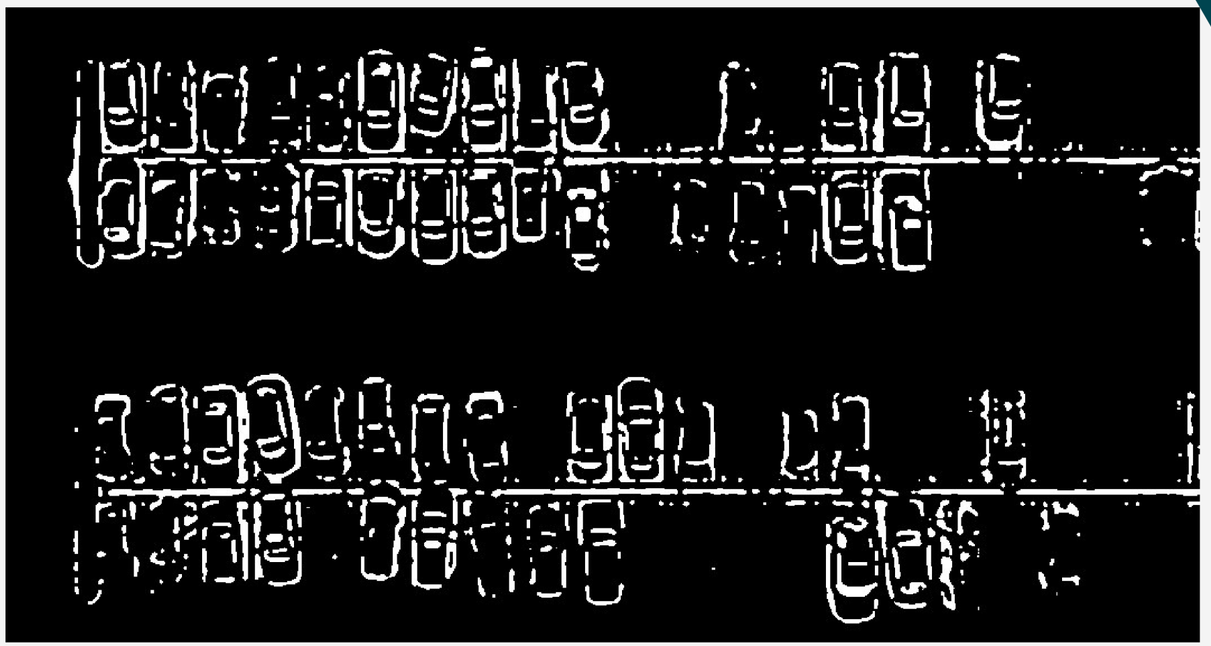


Testcase 2

Input testcase 2

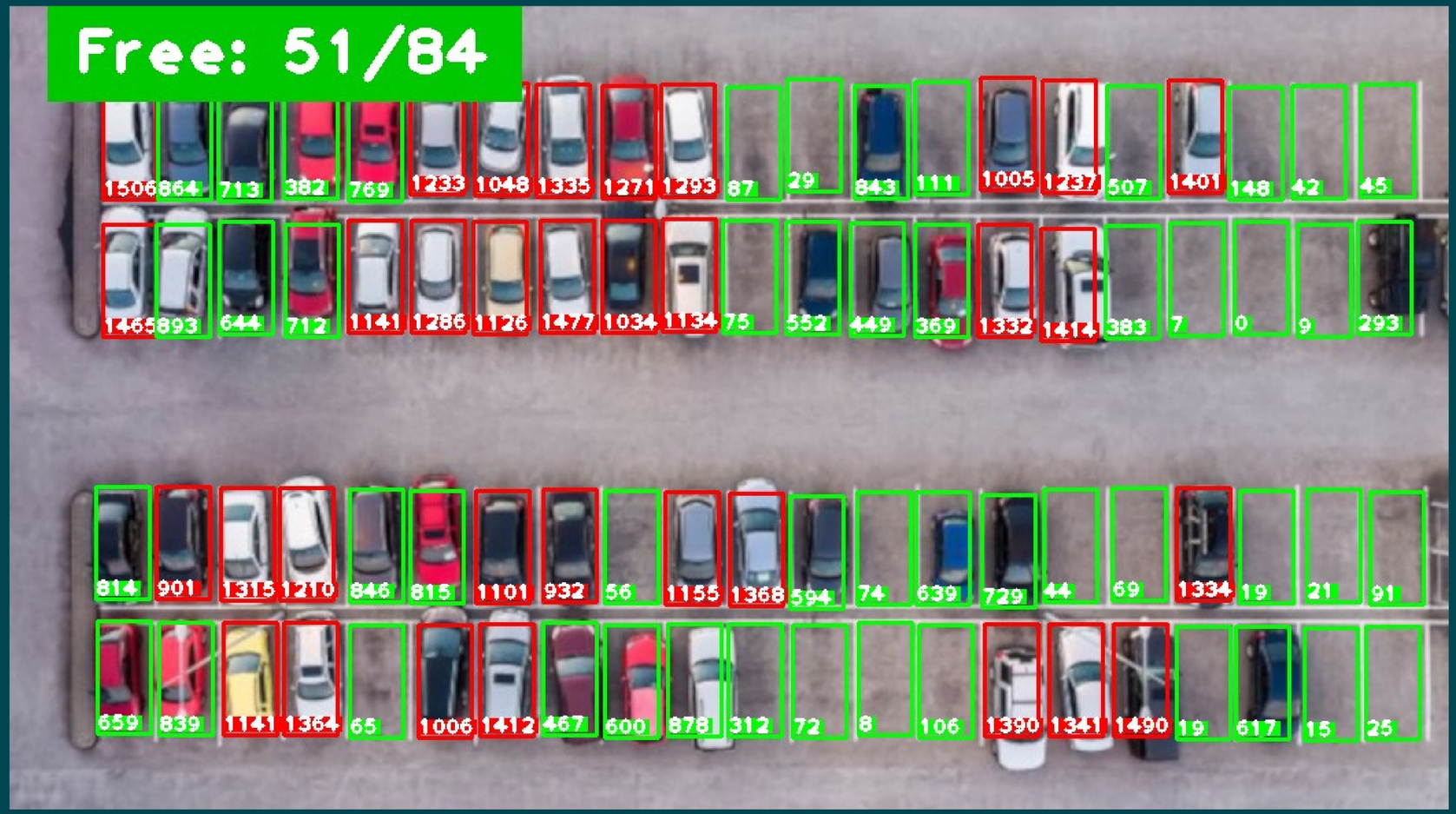
After marking the parking spaces manually

Grayscale image subjected to gaussian blur



Adaptive thresholding and median filter followed by dilation

Counting and detecting empty parking spaces



Final output after calculating pixel count

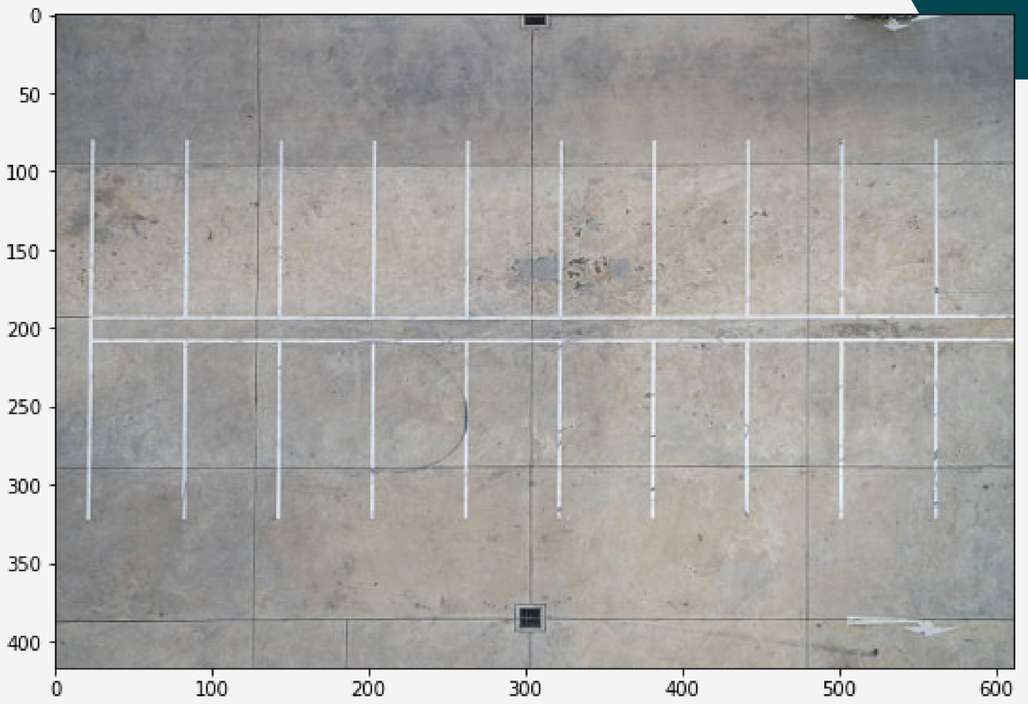
Drawbacks of this approach

- In this method, one needs to manually mark our region of interest, that is, the parking lot from each input image. This is time-taking and not the most efficient approach.
- The dimension of the grid changes for each input image, hence if the dimensions are not given it cannot be expented to more testcases.
- This approach may also give an error in the total count if the parking lots are marked incorrectly initially.

Approach

Method 2

- Making use of occupied and unoccupied parking lot images

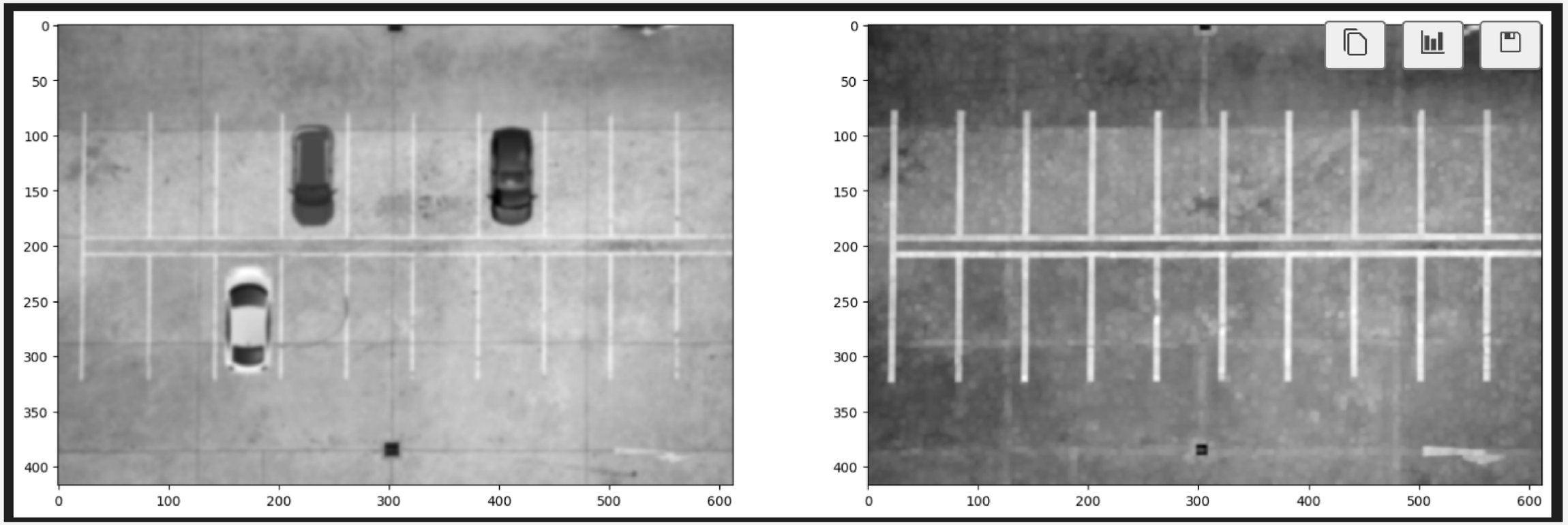


Step 1

- Load, convert BGR to RGB and resize the occupied image to match the size of the unoccupied image

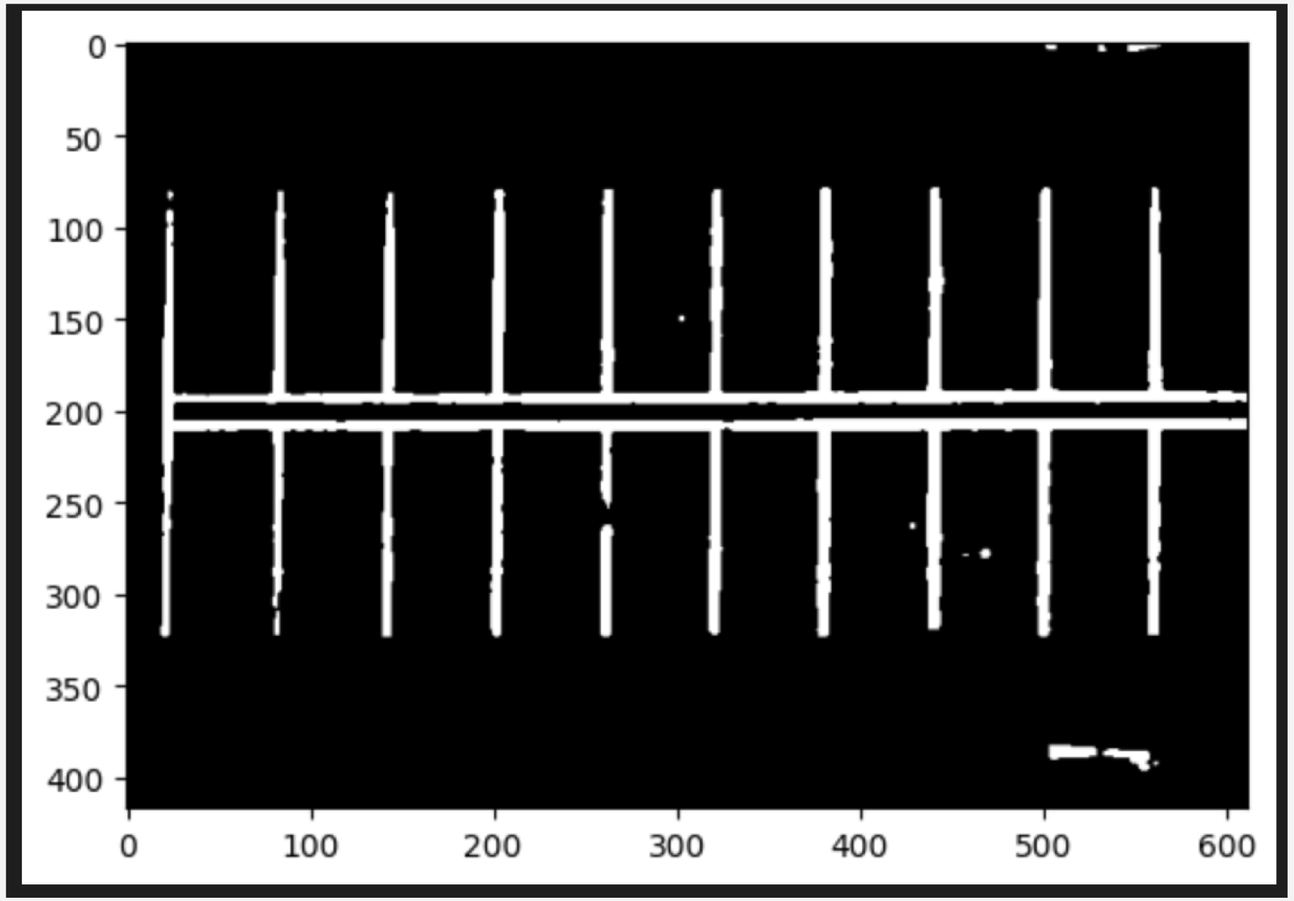
Step 2

- Convert to grayscale
- Dilate the unoccupied parking lot to emphasise the parking lot boundaries
- Apply Gaussian Blur to both the images to reduce noise and smoothen the images



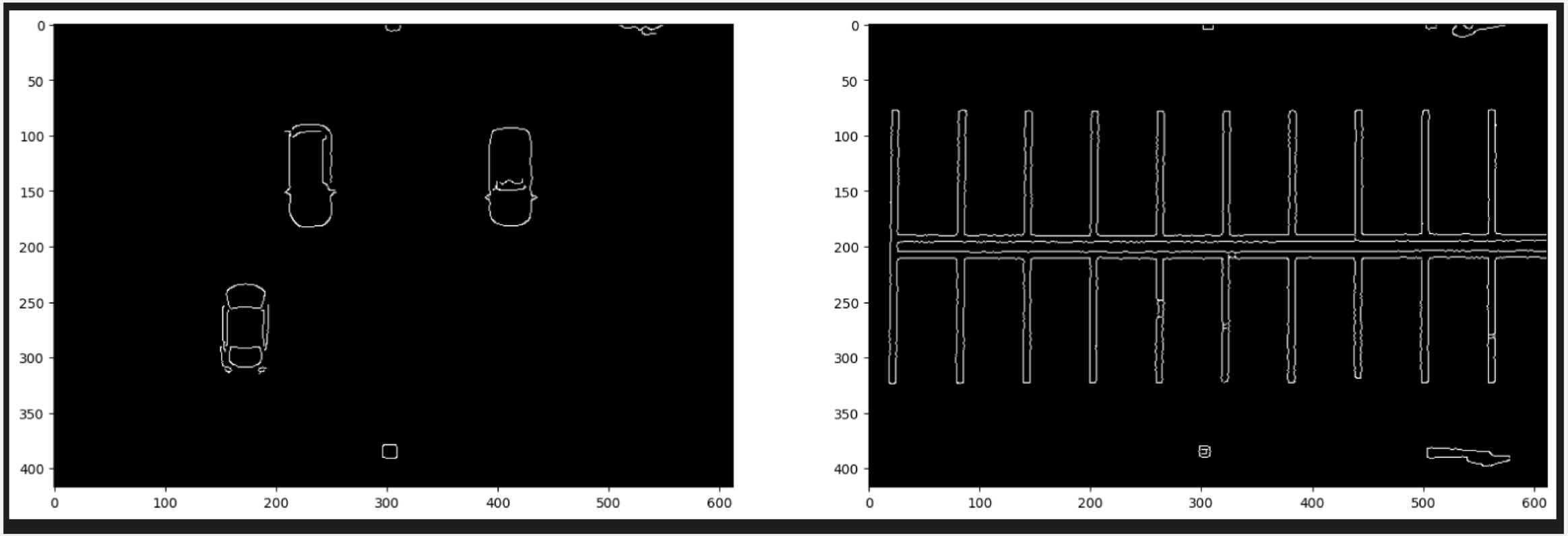
Step3

- Convert the unoccupied image to Binary using appropriate threshold



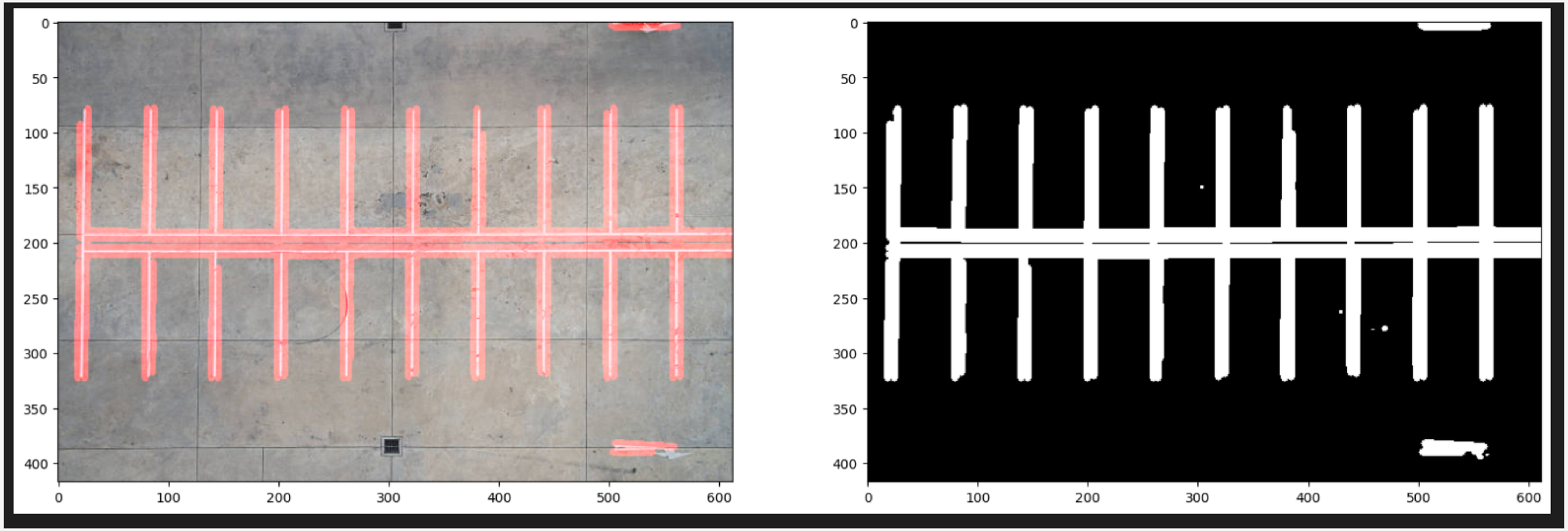
Step 4

- Apply canny edge detection to both the images



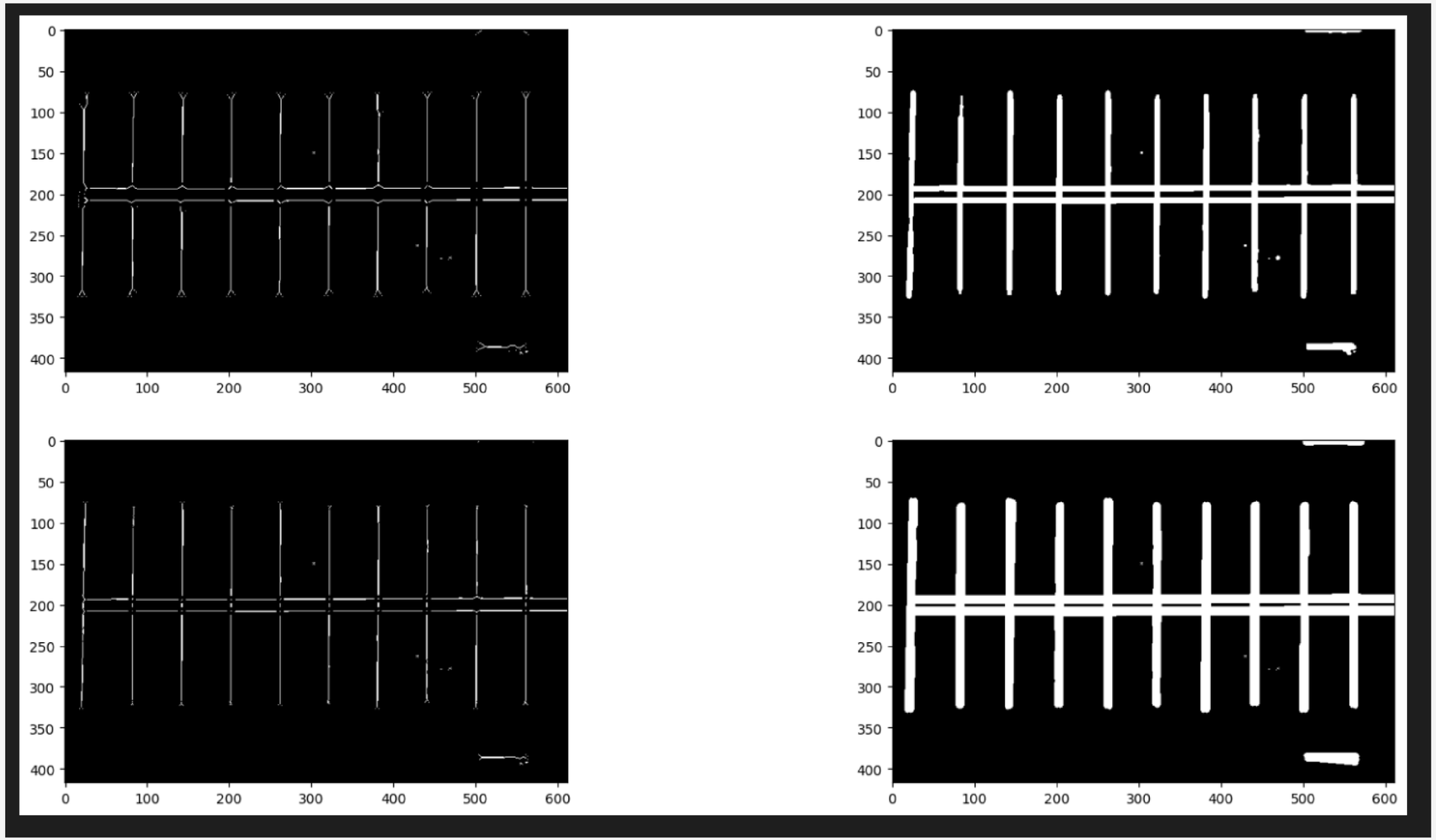
Step 5

- Use Houghline transform on the edges detected in the previous step for the unoccupied image



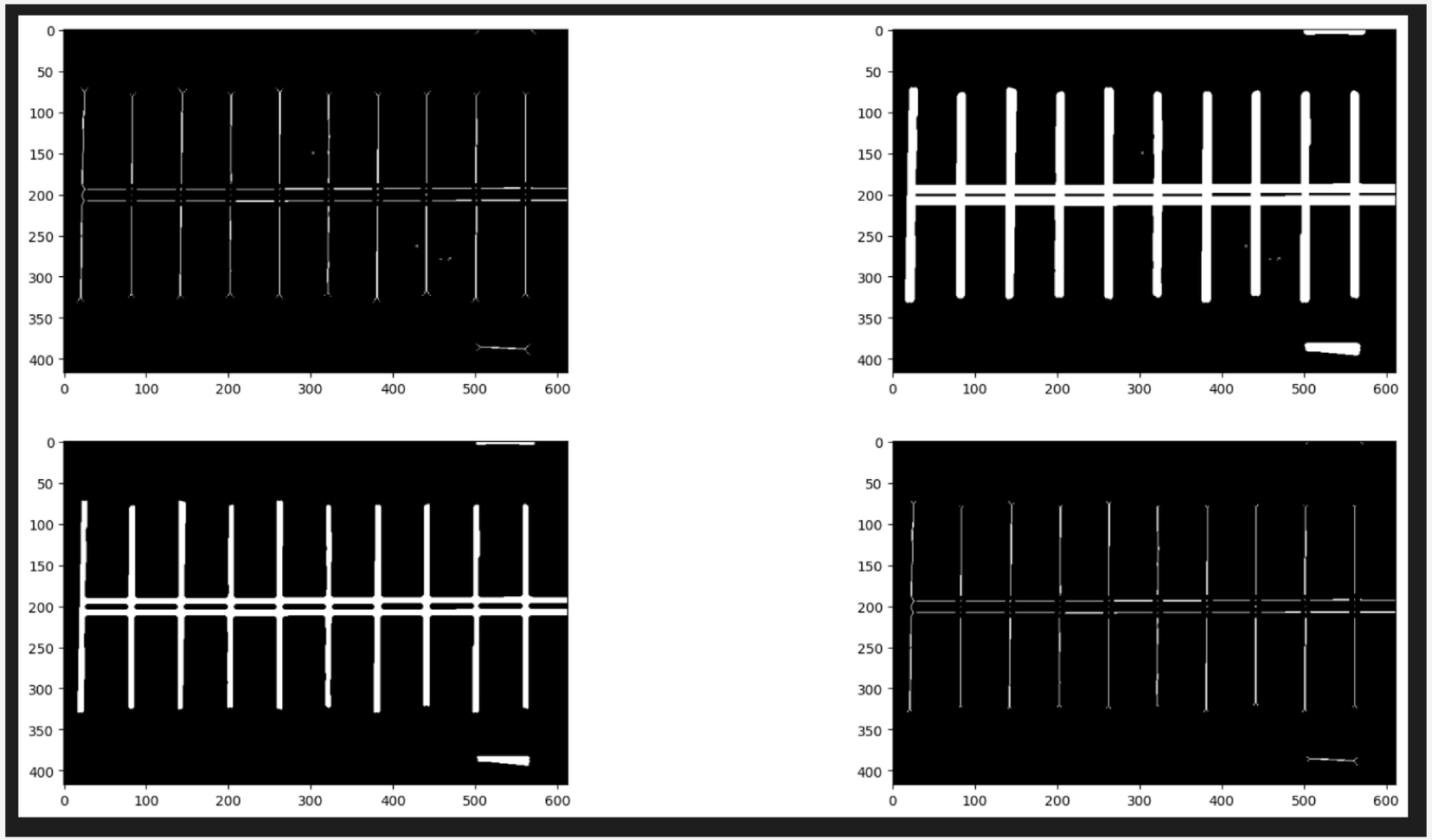
Step 6

- To accurately retrieve the straight lines, use skeletonization and detect HoughLines repeatedly



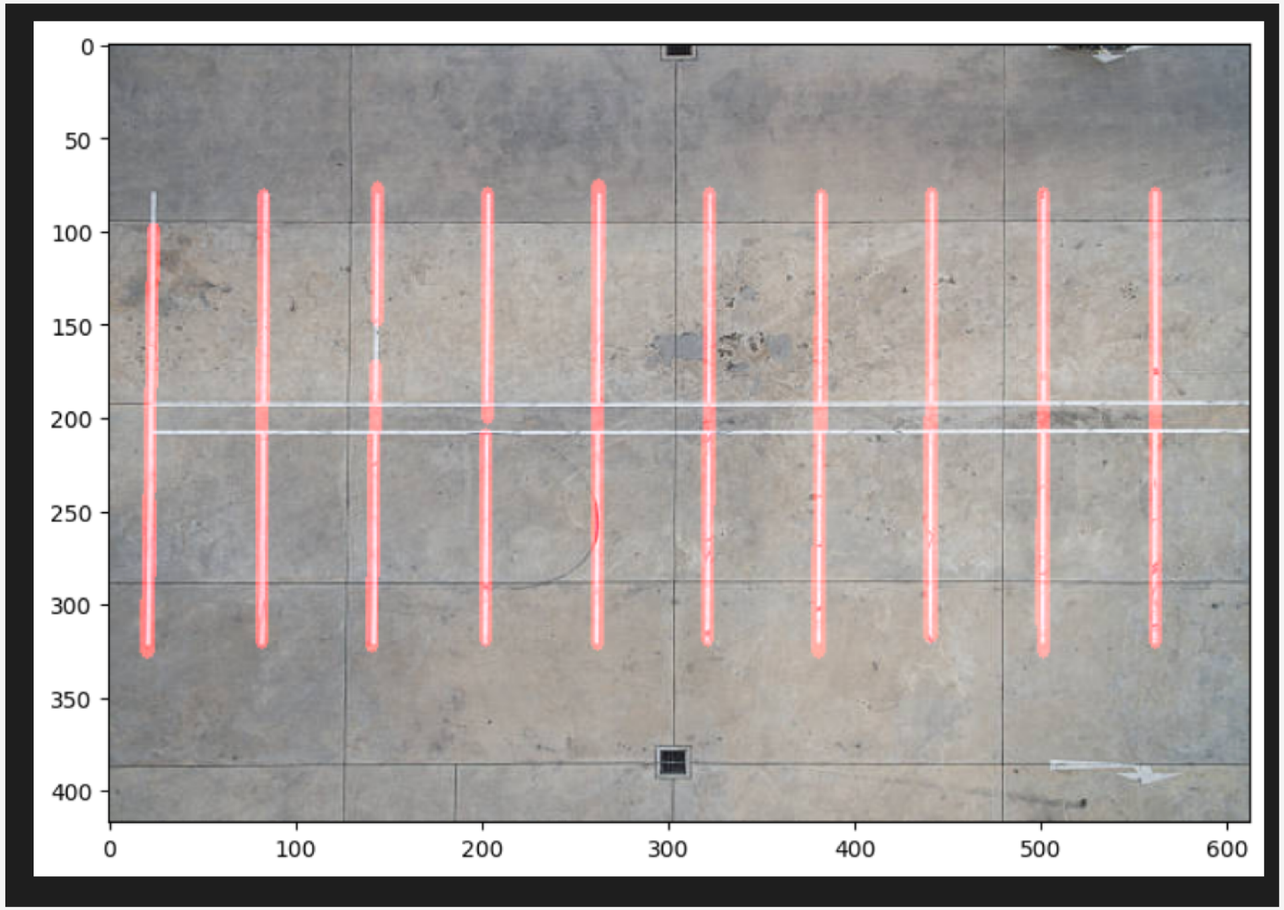
Step 7

- Erode the final output obtained from previous step to detect minimum width straight lines of the parking lot



Step 8

- Draw only the vertical straight lines of the parking lot in order to detect the centre of every parking space
- To do this we hide all the lines with slope close to 0. Using this approach even the slanting parking lots are accommodated.



Step 9

- Find the centre coordinate of every parking space using the x1, x2, y1, y2 coordinates of vertical lines obtained

```
findPKLots(skel, src_unoccupied)

✓ 0.2s                                Python

18

[[51.0, 147.0],
 [51.0, 270.0],
 [111.5, 147.0],
 [111.5, 270.0],
 [171.5, 147.0],
 [171.5, 270.0],
 [232.0, 147.0],
 [232.0, 270.0],
 [291.5, 147.0],
 [291.5, 270.0],
 [350.5, 147.0],
 [350.5, 270.0],
 [410.0, 147.0],
 [410.0, 270.0],
 [470.0, 147.0],
 [470.0, 270.0],
 [530.5, 147.0],
 [530.5, 270.0]]
```

center = img.shape[1]/2

h = max height of houghlines

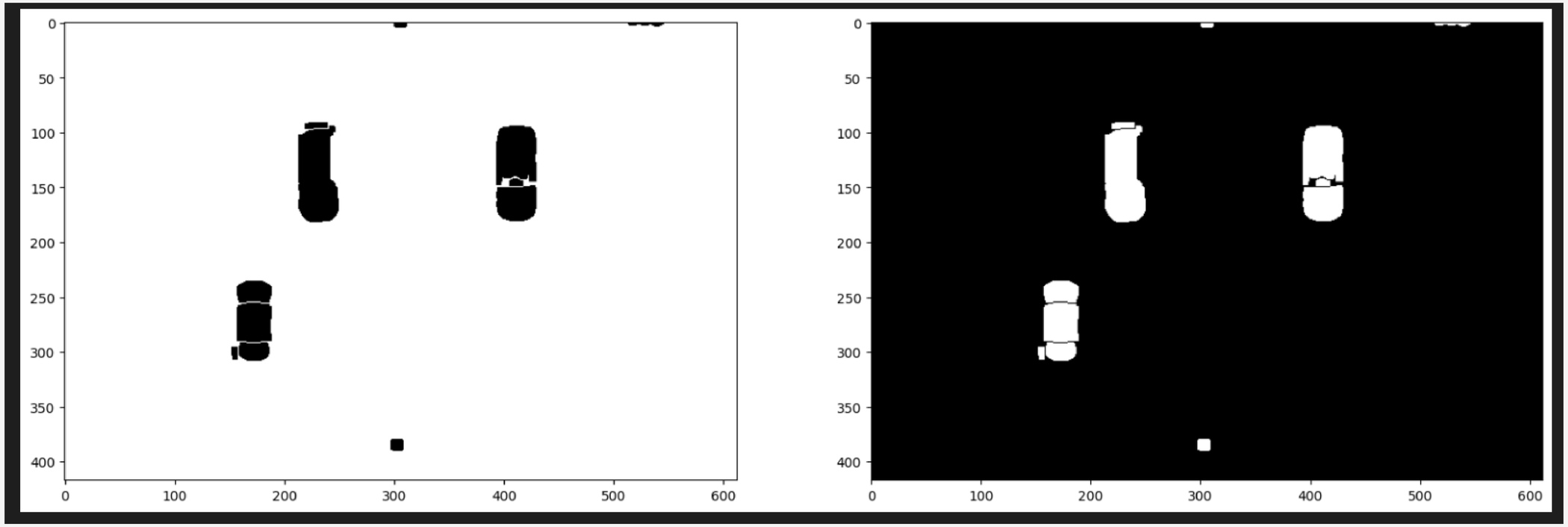
y1 = c + h

y2 = c - h

x coordinates = x coordinates of all the hough lines where, if abs(x1-x2)<6 then its considered to be a straight line

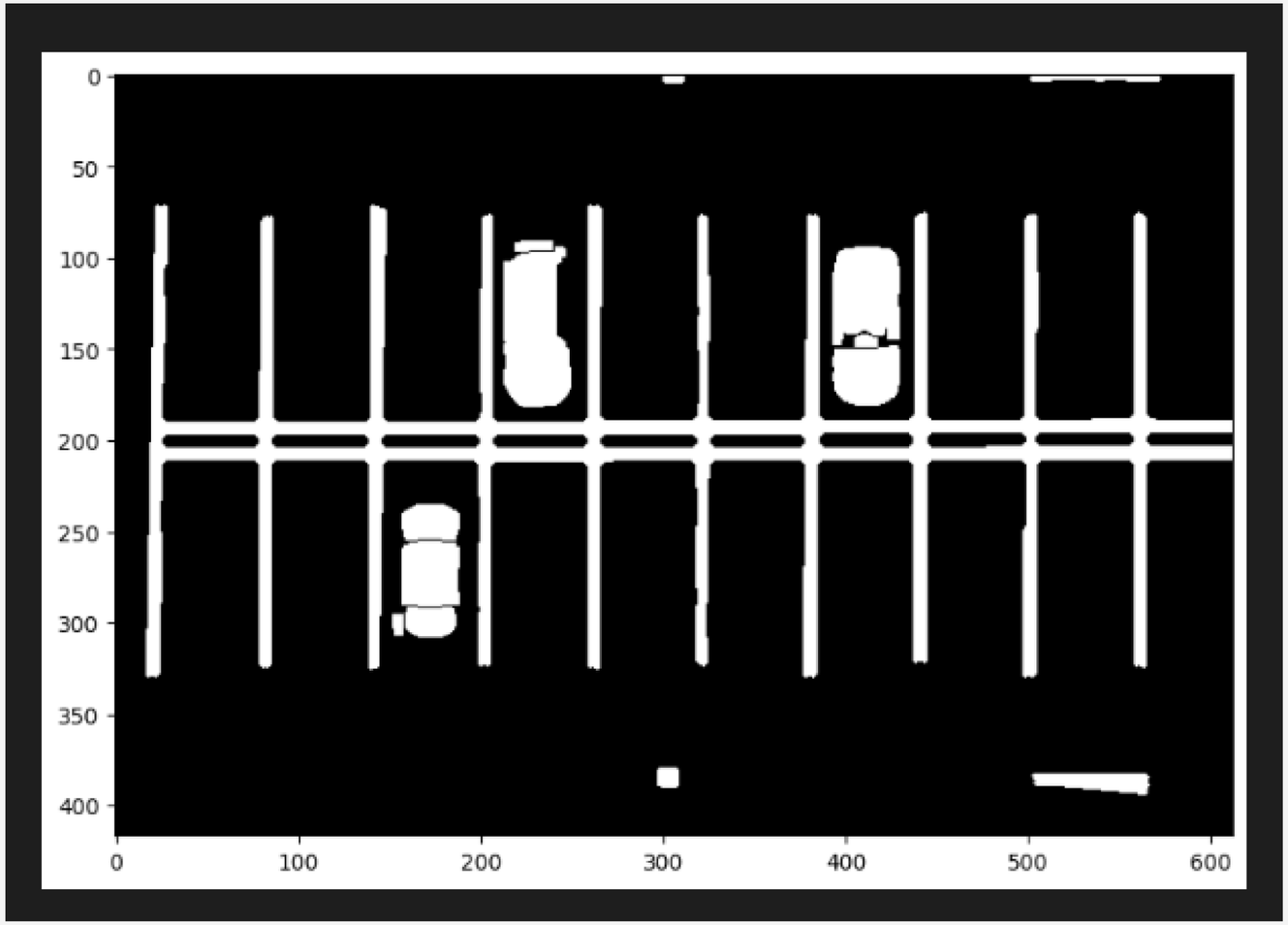
Step 10

- Perform closing on the output from step 4 (canny) for the occupied parking lot to fill the holes of the car edges detected
- Apply thresholding
- Apply floodFill to fill the gaps and have a solid colored component in place of the car
- Invert the image for car to be represented in white color



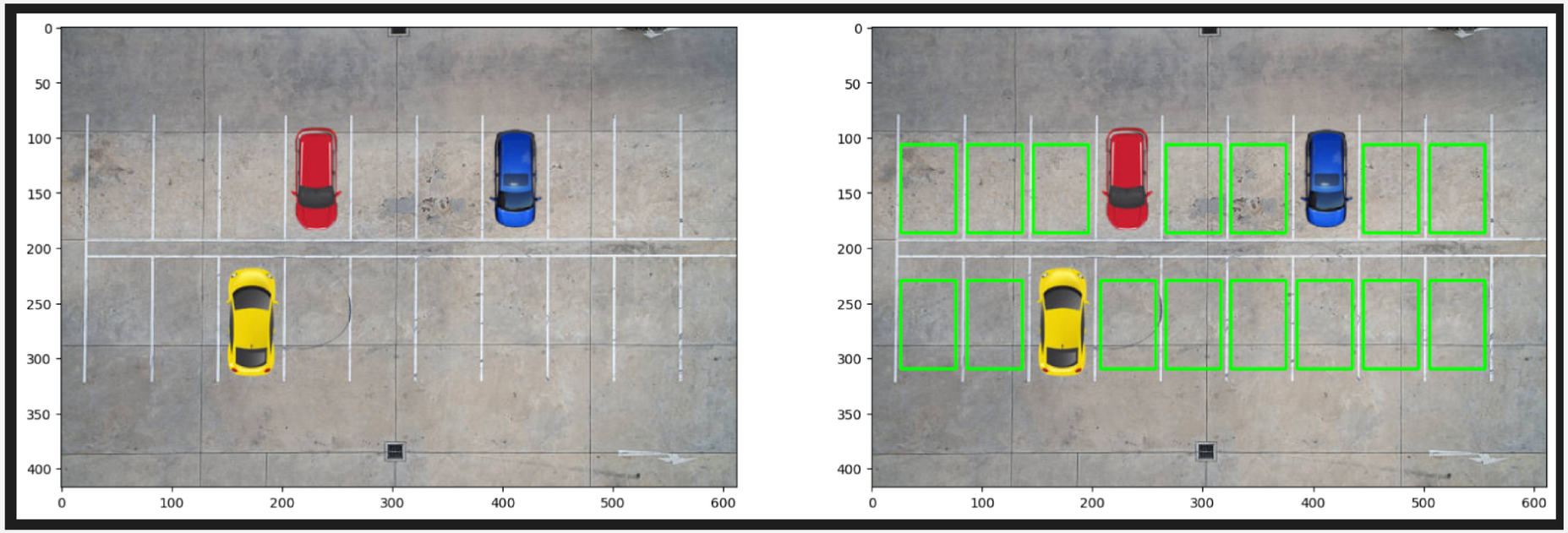
Step 11

- Perform bitwise OR between the outputs of the previous step and the parking lot lines detected from the unoccupied image



Step 12

- Using the centres previously generated, count all the pixels given then centre using BFS algorithm and connected components
- If the number of pixels crosses the given threshold, a car is detected in that parking space
- Check for the presence of the car on every centre coordinate
- Store all the empty parking lot coordinates in a list and lastly mark them using cv2.rectangle on the original occupied image and print the number of unoccupied spots



Some other test cases

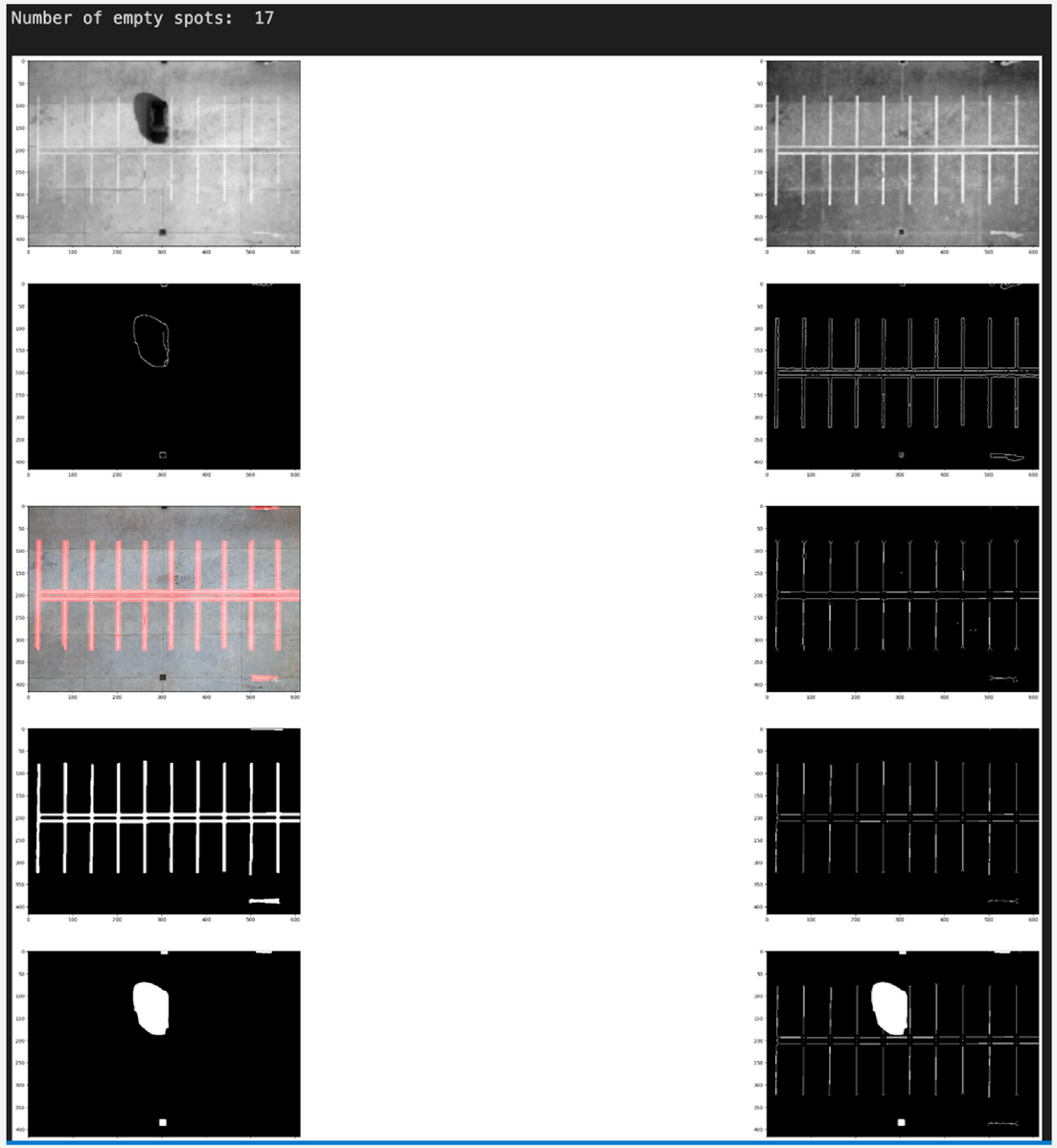
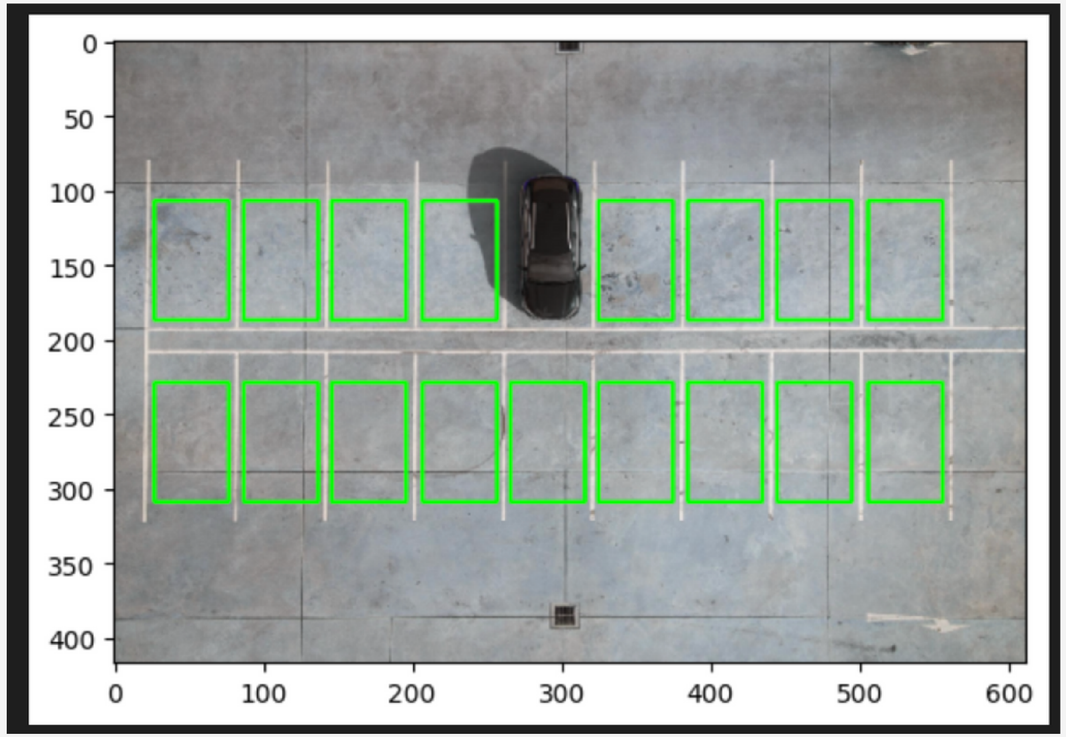
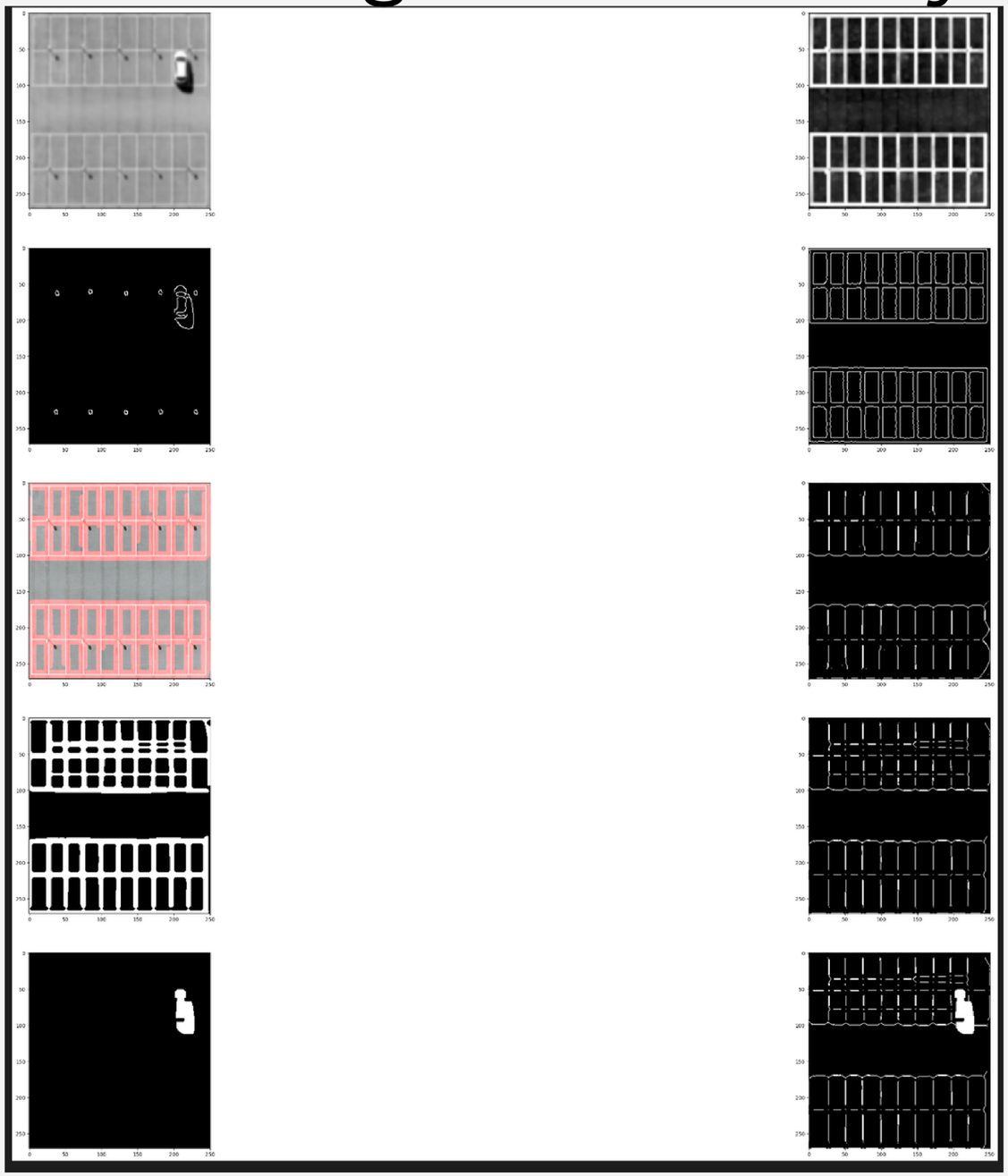
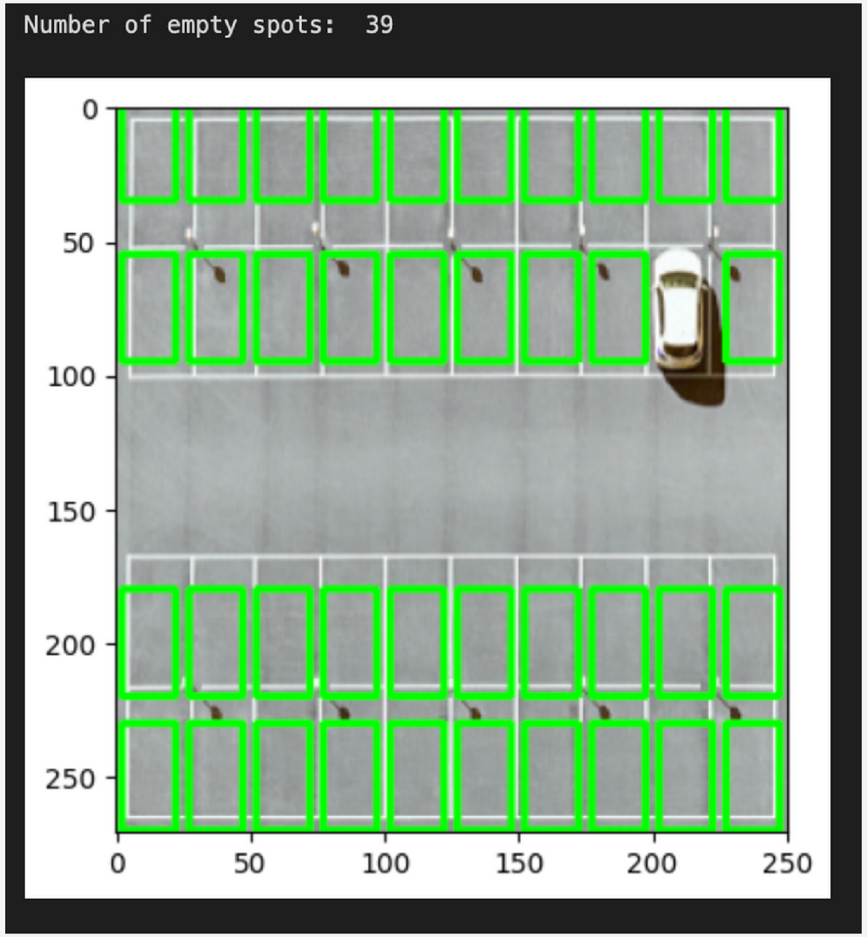


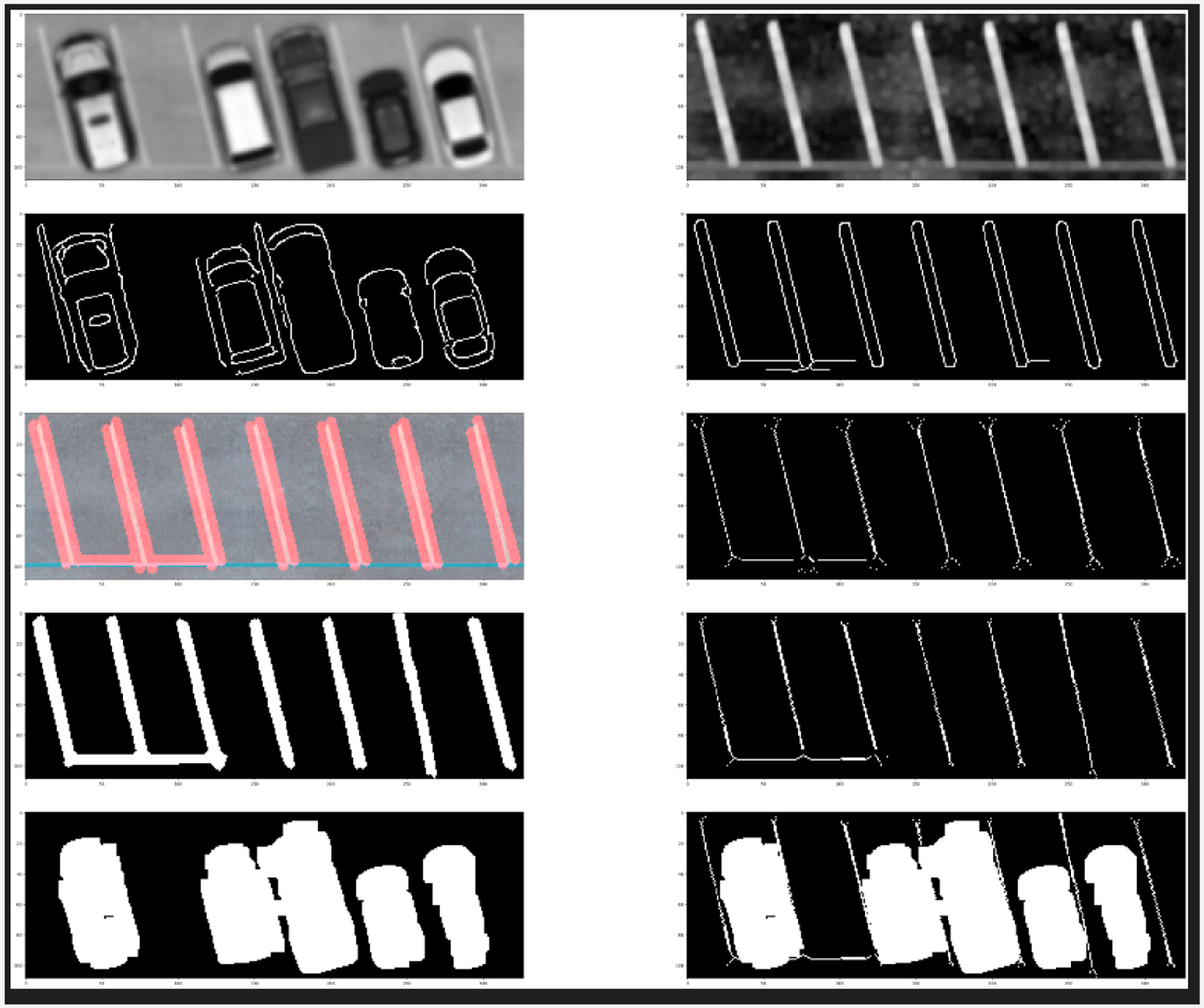
Image with shadow

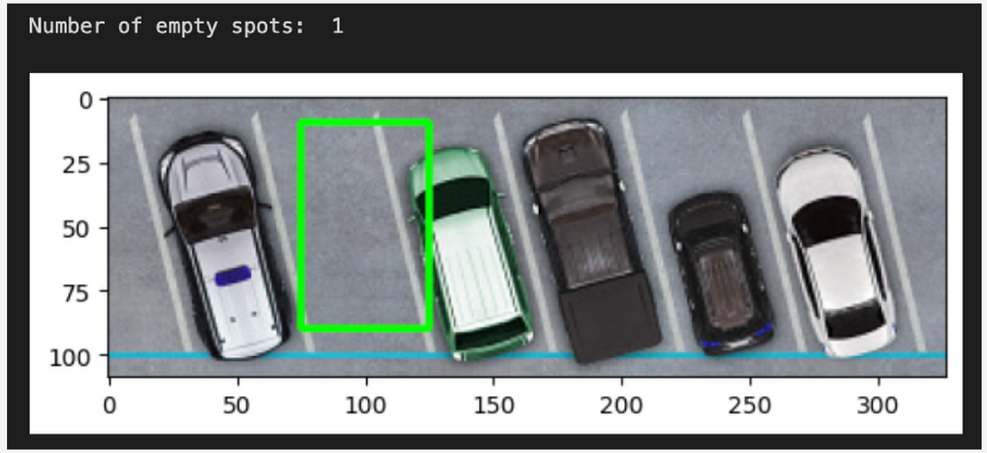


Parking lot with many spaces and shadow









Slanting parking lot

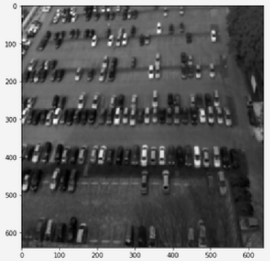
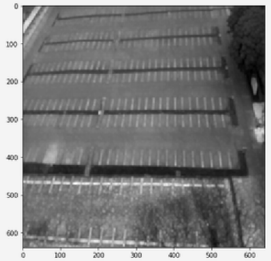
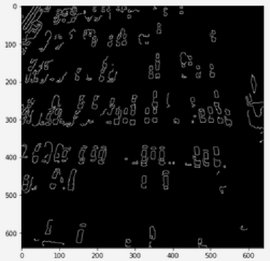
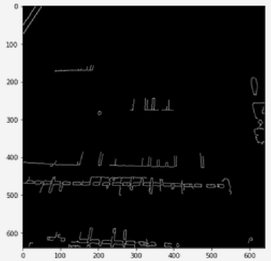
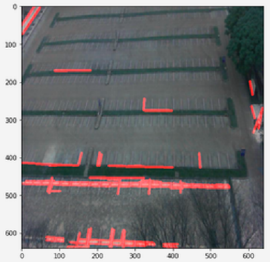
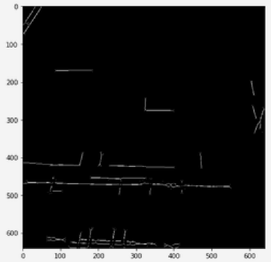
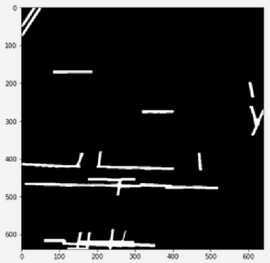
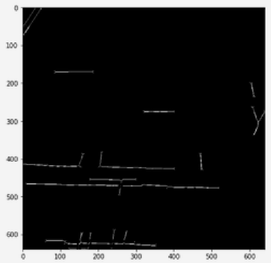
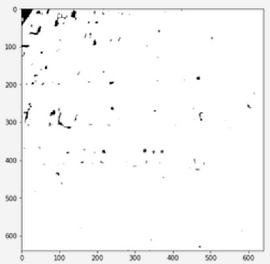
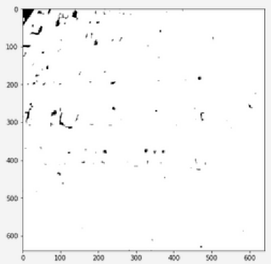











Image from Car Pklot dataset

- Not top view of the parking lot as mentioned in the problem statement
- the parking lot boundaries are very faint to be detected by though lines

Bibliography

1. Pickel documentation: Used to save positions of parking spaces in approach
1. [link]
2. Pixel count to detect if the parking space is empty or not. [link]
3. Capturing mouse click events with Python and OpenCV - SetMouseCallBack documentation [link]
4. Adaptive Thresholding for detecting curves and edges. [link]
5. Cvzone documentation: Used for displaying the pixel count over an image. [link]
6.