CPE 428-01 Computer Vision

Final Project: Detecting Open Spaces in a Parking Lot

By Nathan Orloff, Tushar Sharma, Conrad Kinsey, Micah Jeffries

Dr. Jane Zhang

18 March 2022

1) Introduction

In today's high traffic world, a trip to the grocery store or to the office can be prolonged by our search for an open parking space. Especially in high population density urban areas, people often spend much time trying to find a parking space and end up parking far from where they desired. With millions of applications available for download in our mobile age, one has to ask why there is no mainstream app to solve this epidemic. In this paper we propose a solution to and document our experimental work on this problem.

To approach this problem, we based our attempts off two different methods proposed by a group of students from Midwestern State University. Their paper [1] briefly documents their work and their two proposed solutions. The first method involves a color analysis comparison of an open parking space against an occupied parking space. By extracting a histogram of the RGB color content in each histogram, one can reliably determine whether a parking space is open or occupied. The second method involves the use of the canny edge detector to find the amount of edges in a parking space. An occupied parking space will presumably have much more edges than an open parking space. Thus, this also allows for a reliable detection method for determining whether a parking space is open or occupied.

For each image in our database, there is a corresponding mask which highlights the boundaries of each parking space in that image. Since the boundaries of these parking spaces could not reliably be extracted using an algorithm, the boundaries were self-drawn using the built-in MATLAB function roipoly(). This function provided us with a user-friendly interface to draw the rectangular regions around each parking space while saving them in a mask. Not all parking spaces in an image were necessarily drawn, just the ones we thought most important and visible from the camera angle.

The images of parking lots in our database were taken from the web and range from various locations. These images in our database provided great variability with certain features that affected our results such as the time of day and the camera orientation. As for the time of day, we assume that no image was taken at night. The time for each image ranges from sunrise to sunset. The camera orientation ranges from a horizontal capture all the way to an aerial capture. In the analysis of results for each method, we document how each of these factors affected the accuracy of our results.

2) Parking Space Detection

2.1) Parking Space Detection - ROI

This method of parking space detection was performed by manually extracting the parking spaces as regions of interest, and saving them as bit masks to be used for analysis later. Using Matlab code to enable multiple ROIs to be drawn for a single image, boxes representing each parking space were manually drawn in for each image, and saved for later. This method is simple, reliable, and very useful as long as the camera angle on a parking lot does not change. Even though this may seem tedious to do, as long as the camera angle does not change this step only needs to be performed once. After the bit masks are initially obtained, they are able to continually be used for analysis of the same camera position.

For this method, 5 test images were used for the bit mask extraction. The resultant bit masks obtained from this approach are displayed below. They show exact removal of the parking spaces which can then easily be used in both methods of analysis later on. This method avoids many of the problems that will be caused by automation of this step. Shadows, white spaces, low image quality, and many more outside factors that would disrupt an automated parking space detector are avoided using this manual method. By manually drawing all of the bit masks, an accurate way of detecting all of the parking spaces within an image is applied. The main downfall of this method is the tedious nature of drawing all of the regions of interest. As the number of parking spaces in an image increases, the amount of work required to detect all of the spaces also increases.

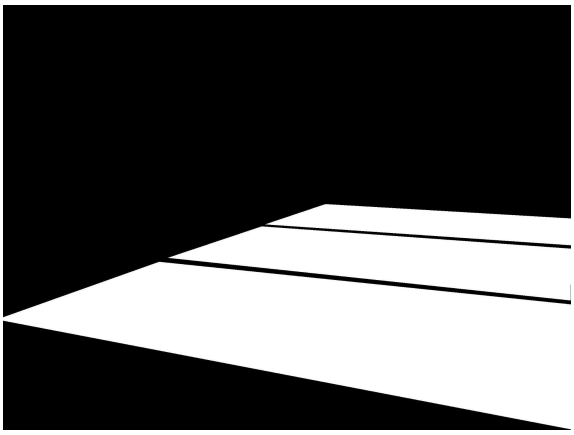

Figure 2.1.1: Three parking spaces ROI

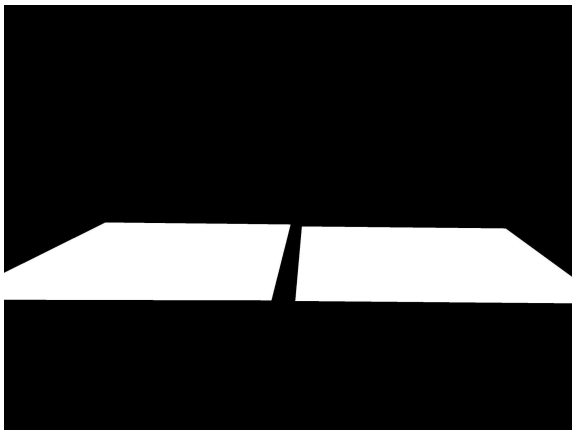

Figure 2.1.2: Two parking spaces ROI

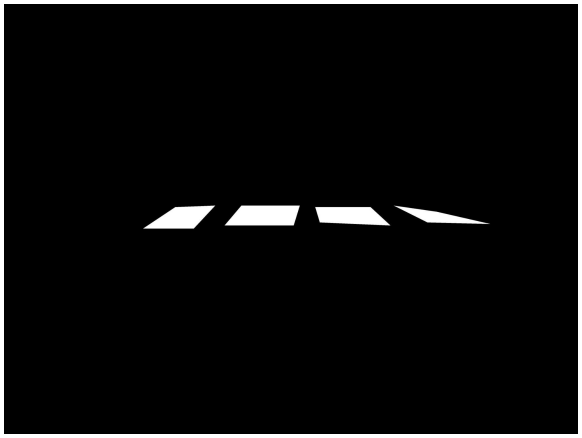

Figure 2.1.3: Four parking spaces ROI

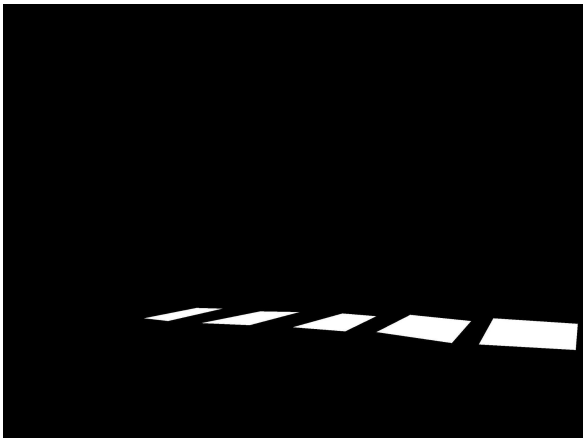

Figure 2.1.4: Five parking spaces ROI

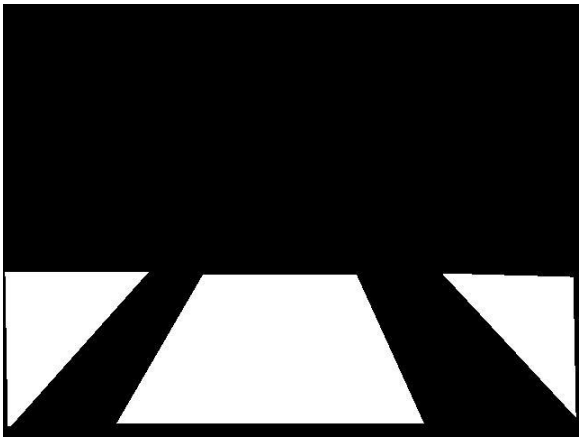

Figure 2.1.5: Three parking spaces ROI

2.2) Parking Space Detection - Hough Transform

This method of parking space detection uses image processing techniques including the hough transform to identify parking spaces and attempts to create bit masks, which can then be used in the parking space analysis to find open spaces. The objective of this method was to use image processing techniques to isolate just the parking lines of the parking space, then using the hough transform to obtain points/lines representing that parking space. From there these hough-lines would be used to generate a bit mask of each parking space in the image.

The first images seen below show the first couple steps taken for isolating the parking lines. The image on the left is the grayscale parking spaces after two smoothing filters were applied. First a 5x5 median filter was applied to remove any spikes in pixel intensity and any salt and pepper noise present in the image. Then a 9x9 gaussian filter was applied to further smooth the image removing any noise left over. Then the right image shows the parking spaces after a

threshold is applied to the image. Due to the parking lines being white and having high intensity pixels, a high intensity threshold is used to obtain only the pixels belonging to the parking lines.

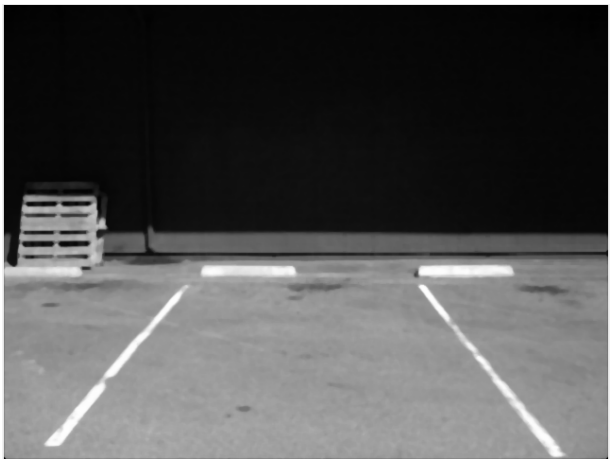

Figure 2.2.1: Grayscale parking spaces

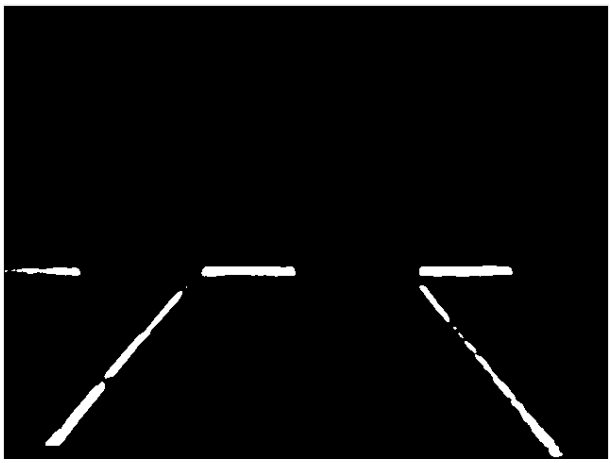

Figure 2.2.2: Thresholded parking spaces

```
i_gray_filter = medfilt2(i_gray,[5 5]);
i_gray_filter = imfilter(i_gray_filter, fspecial('Gaussian', [9 9]));

figure;
imshow(i_gray_filter)

threshold = 210;
i_thresh = i_gray_filter > threshold;
```

From there, morphological operations were performed on the thresholded image to strengthen the parking lines while reducing/removing any lines that are not useful parking lines. First, show in the left image, the lines are dilated using a disk structure. This emboldens all of the lines in the image preparing for erosion which will reduce them. The initial dilation prevents the parking lines from being eroded out of the image. After the initial dilation, erosion is performed twice using first a disk structure, then a line structure. This reduced the image down leaving only the useful parking lines and a couple other small leftover white spaces in the image. Dilation is performed once more using a line structure to embolden the parking lines without restoring the previously removed white spaces. The result of these operations is the image shown below on the right.

Figure 2.2.3: Dilated parking spaces

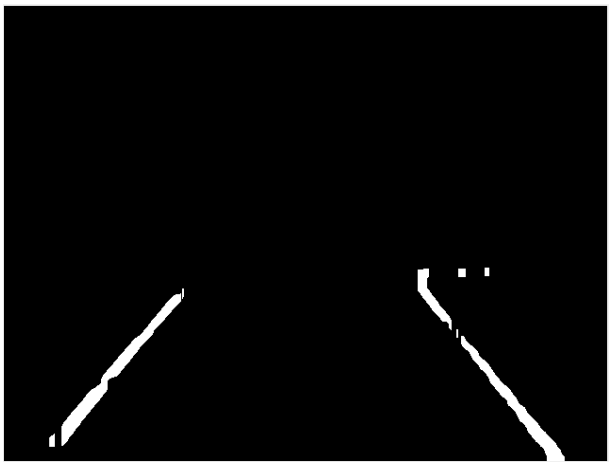

Figure 2.2.4: Eroded parking spaces

```
%dilate parking lines
se = strel('disk', 6);
dilatedBW = imdilate(i_thresh, se);

figure;
imshow(dilatedBW, [])

%remove anything non-parking lines
se = strel("line", 18, 90);
erodeBW = imerode(dilatedBW, se);

se = strel("disk", 2);
erodeBW = imerode(erodeBW, se);

%strengthen parking lines again
se = strel("line", 8, 90);
erodeBW = imdilate(erodeBW, se);

figure;
imshow(erodeBW, [])
```

Next, the hough transform is taken using the resultant image from the morphological operations. Since only the parking lines remain in the image, the hough transform only draws hough-lines on the parking lines, giving the image coordinates of the parking lines. These hough-lines are plotted on the grayscale image below.

Figure 2.2.5: Hough-lines drawn over parking lines

```
figure;
imshow(i_gray), hold on
%get hough transform
[H, theta, rho] = hough(erodeBW);
numPeaks = 5;
%get hough peaks
peaks = houghpeaks(H, numPeaks, 'Threshold', 30);
%get hough lines
lines = houghlines(erodeBW, theta, rho, peaks, 'FillGap', 5, 'minLen', 15);

%draw lines on image
for k=1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:,2), 'LineWidth', 1, 'Color', 'r');
end
hold off
```

This is another example of using this technique to retrieve the hough-lines of the parking spaces. The left image shows the image after all thresholding and morphological operations are applied. The right image shows after the hough-lines have been plotted on the grayscale image. This image shows some of the troubles faced with this method. There were many sections within this image that were unable to be removed using any of the image processing techniques. The roof, the arrow, and a few other parts of the image remained after all of the techniques were used. This did not stop the hough-lines from marking all of the parking lines, but additional hough-lines were drawn in areas where no parking lines were present.

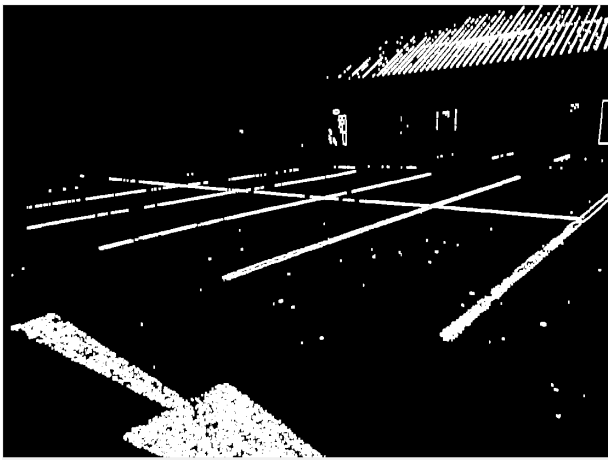

Figure 2.2.6: Dilated and Eroded parking spaces

Figure 2.2.7: Hough-lines drawn over parking lines

Using the hough-lines to obtain bit maps to be used for analysis caused many problems. Because the hough lines were unordered and could represent non-parking lines, getting accurate bit maps proved difficult and was not possible for this stage of the project. On top of that the current method exhibited problems where it was not widely applicable to different parking lot images. Many different features could cause problems that could interfere with this approach. This method makes the assumption that any high intensity pixels that are not parking lines will easily be able to be removed using different image processing methods. But as seen in the images above, this is not always the case, and these additional white spaces can cause many problems in the parking line detection.

3) Analysis Methods

3.1) Method 1 - Color Analysis

To detect open parking spaces, two different methods were utilized. The first method was to utilize RGB colors to detect open spaces. This method was utilized because it was determined that there would be a large difference between the RGB values of a vacant parking space and a filled parking space. The following two images below were used to conduct this analysis.

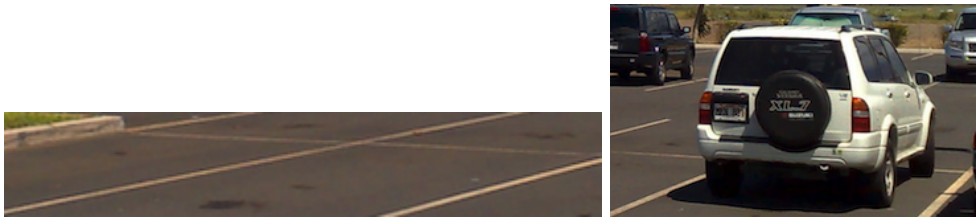

Figure 3.1.1: Parking Spots utilized to extract certain RGB values. Vacant(Left) vs Right(filled)

Once the images were determined, histograms were generated to analyze the spread of values for each parking space. The histogram takes these images and plots all the RGB values present in the image. As shown in Figure 3.1.2, the empty parking space had a smaller range of RGB color values whereas the RGB values present in an vacant parking space had a higher spread.

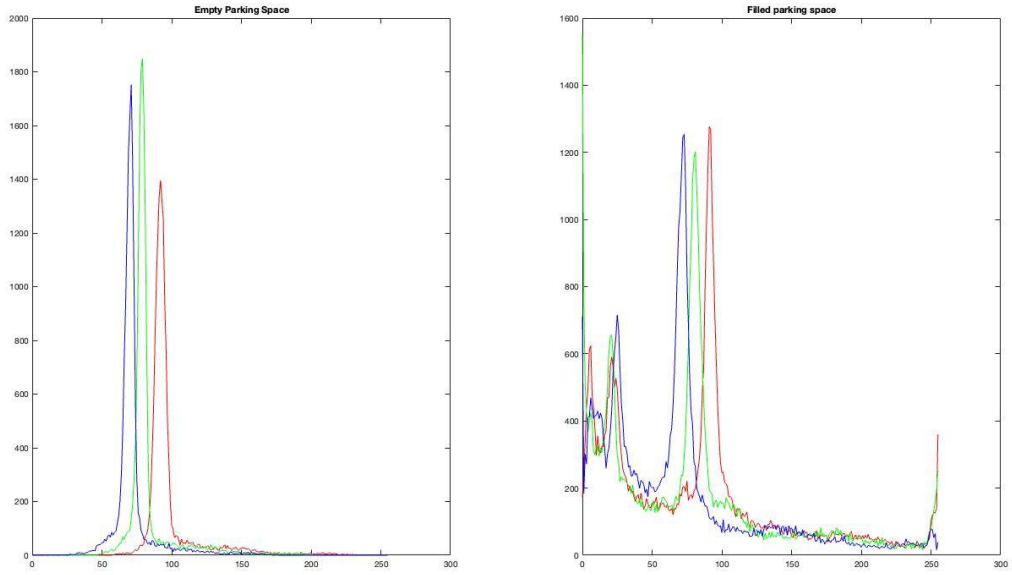

Figure 3.1.2: Histograms showcasing frequency of Red, Green, and Blue values in the image. Vacant parking spot(left) had high frequency of specific values whereas filled parking space(right) had a much higher range.

To identify individual parking spaces, filled or not filled, each space was identified with a self-drawn boundary around each. The example can be shown below.

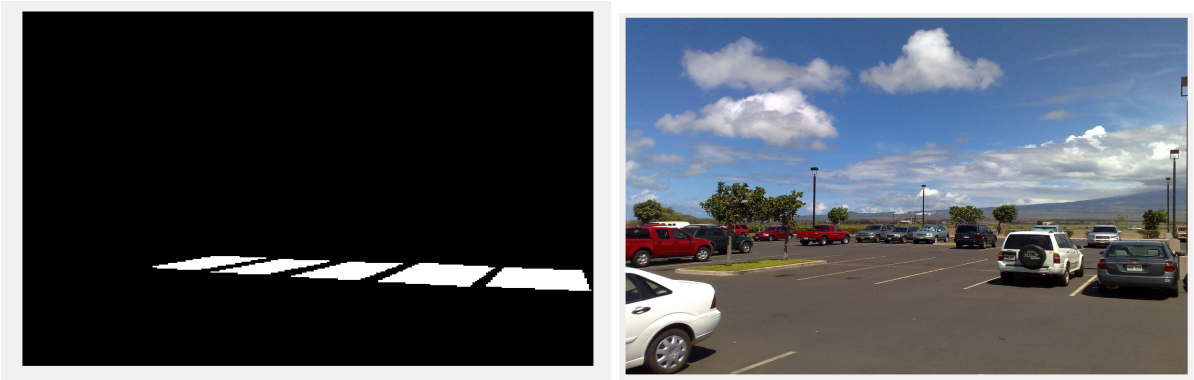

Figure 3.1.3: Drawing Multiple Region of Interest around all parking spots

From the histograms, preselected values were selected for a vacant parking space. These values were based on the peak values for each channel. For Red, the value was 80. For Green, the value was 92. For blue, the value selected was 71. These values were stored in a 3-tuple in the format {R, G, B}. From there, the average color for each parking space was calculated. This operation was done by summing up all the pixel values for each RGB channel and then dividing by the total number of pixels present in the channel. These values were stored in a 3-tuple. A predetermined threshold was determined to compare the average parking space value to the gray parking space value. The difference is compared to the threshold to determine if the parking space is vacant or not.

$$\text{Difference} = \left| \begin{pmatrix} B_{avg} \\ G_{avg} \\ R_{avg} \end{pmatrix} - \begin{pmatrix} B_{gray} \\ G_{gray} \\ R_{gray} \end{pmatrix} \right|$$

Figure 3.1.4: Equation to compute the difference between a parking space value and the predetermine red, green, and blue values

3.2) Method 1 - Results

Though we have many test images in our database, we will present the test results for three parking lot images to keep the analysis of the results succinct. These three test images provide enough variability between camera orientation and lighting to perform a thorough analysis of the impacts of each of these factors for both methods.

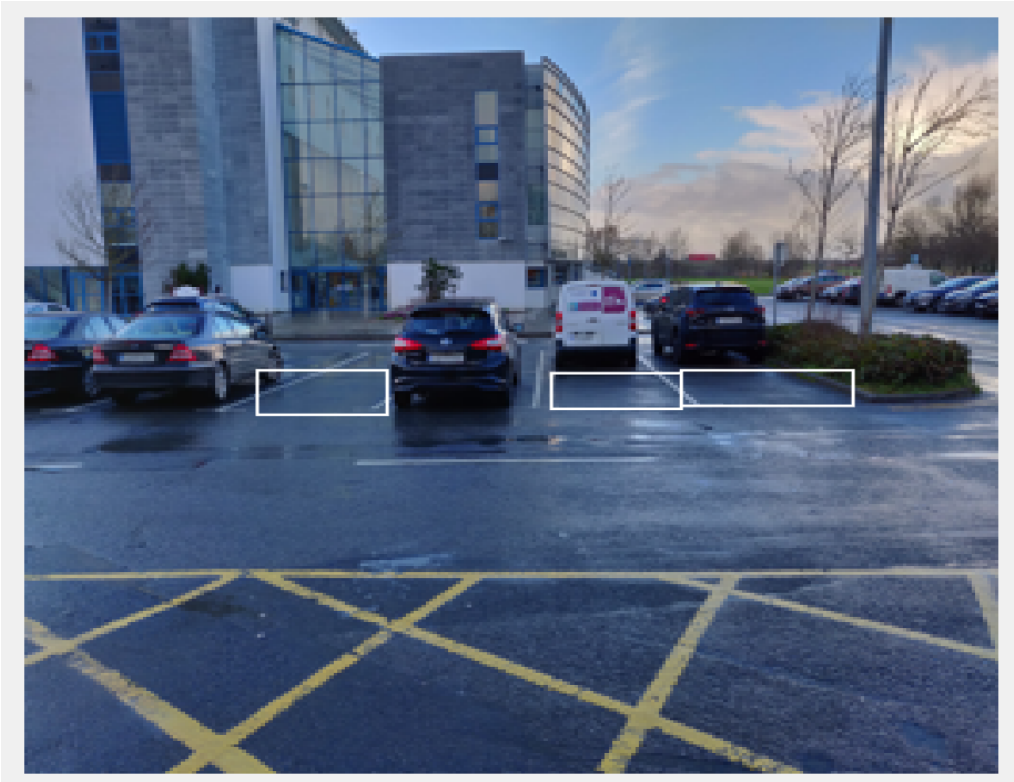

Figure 3.2.1**:** Color Analysis Results for First Test Image

Figure 1 shows the results of the color analysis method on our first test image. For this image, we are just considering the four front parking spaces. The algorithm correctly identified three of the parking spaces as vacant and the parking space with the black car as closed. The gray colors of this parking space are darker than usual due to the relatively low lighting of this image. The gray RGB, as explained in the previous section, had to be adjusted accordingly. This presents a problem as exemplified in the next test result.

Figure 3.2.2: Color Analysis Results for Second Test Image

As can be seen for the second test case, the results are completely inaccurate. For this test case, we are just considering the five front parking spaces. The algorithm incorrectly identifies the two parking spaces with cars as vacant and the three vacant parking spaces as closed. This is due to the problem mentioned earlier. The gray RGB vector was adjusted for the first test image to account for the low lighting. Because this test image is much lighter, the gray values for the vacant parking spaces were higher than was accounted for.

Figure 3.2.3: Color Analysis Results for Second Test Image with Adjusted Gray Vector

As shown in figure 3, this problem can easily be accounted for by adjusting the values of the gray vector to account for the lighting of the image. However, this process is tedious and requires the user to manually adjust the gray vector according to the time of day when the image was taken. If we had more time to work on this method, we could possibly devise a method for determining the lighting of the photo as a preprocessing step. Then we would use this information to adjust the gray vector accordingly.

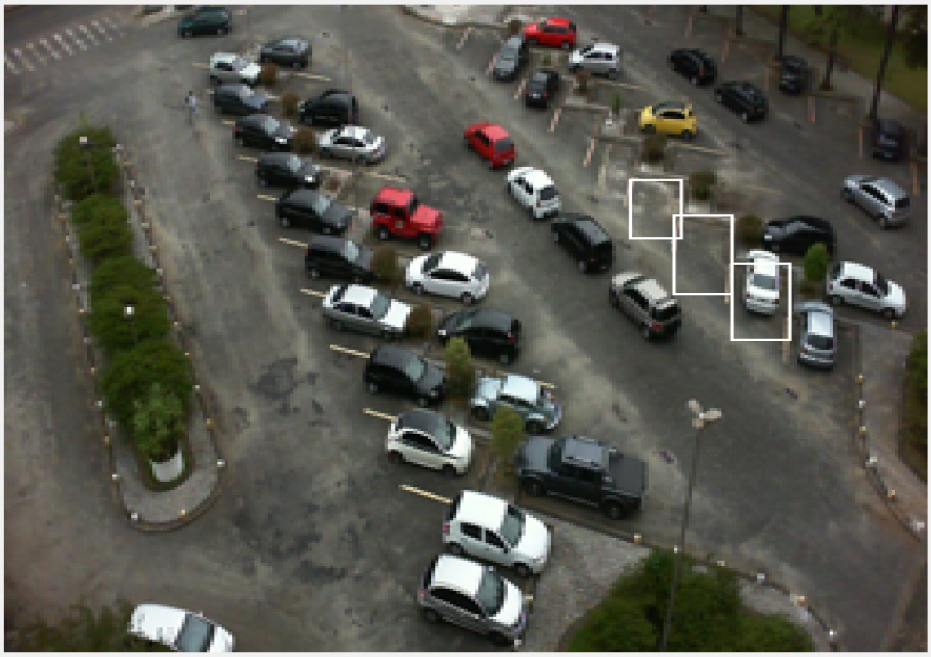

Figure 3.2.4: Color Analysis Results for Third Test Image

Figure 4 displays the results for the final test case which emphasizes an aerial capture of the parking lot as opposed to a horizontal capture. For this test image, we are just considering the four parking spaces on the right hand side of the parking lot. The algorithm correctly identifies the two open parking spaces as vacant. However, it also misidentifies one closed parking space as vacant and correctly identifies the other parking space as closed. An aerial capture does help improve the results because there are less shadows which interfere with the results. In the horizontal captures of the previous two test cases, the car's black shadow threw off the color analysis of each parking space. The shadows from each car are just barely visible from an aerial capture. Overall, this method is fairly decent with some adjustments to the algorithm that could be made. However, we believe the next method has better performance than this method.

3.3) Method 2 - Canny Edge Detector

To combat the problem found with adjusting the gray parking space values, a new parking space detection method was proposed utilizing edge detection. For the edge detection method, canny edge detector was utilized to detect edges in the car space. Canny was selected over other edge detection algorithms since it applies a gaussian filter mask to smooth the image. In several of the parking space images, noise was present which interfered with proper parking space detection. Similar to the RGB color analysis method, each parking space was identified by manually drawing a ROI around the space. From there, the canny edge detector was run on each space to detect suspected edges in the photo. In this method the amount of white pixels and black pixels were extracted to determine how many edges are present in a particular space. A predetermined threshold was created to classify spaces into vacant or filled. The following equation was used.

$$\frac{White\ Pixels}{White\ Pixels + Black\ Pixels} > \varepsilon$$

Figure 3.3.1: Equation to detect open parking space

The reasoning behind this method was that a parking space which had more edges is more likely to be a parking space than not one. Figure 3.3.2 illustrates the difference in detected lines for a vacant parking space and filled parking space.

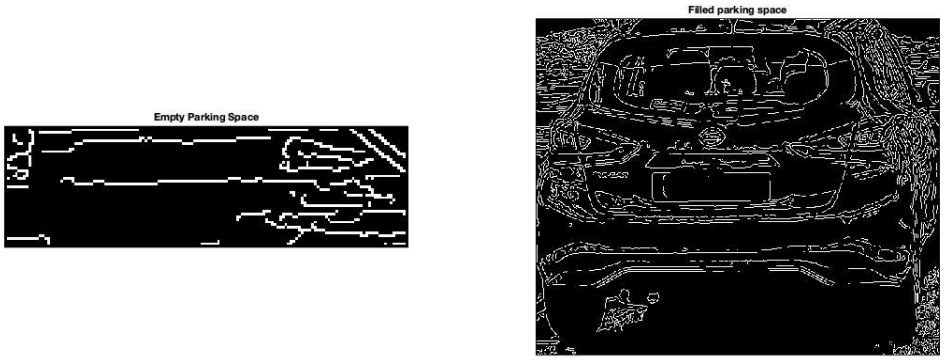

Figure 3.3.2: Vacant Parking Space(Left) vs Filled (Right).

Significantly more edges are detected in the filled parking space image than the vacant. Although there are some miscellaneous lines present in the image, the threshold filters out the effect of those lines. Another example is shown in Figure 3.3.3.

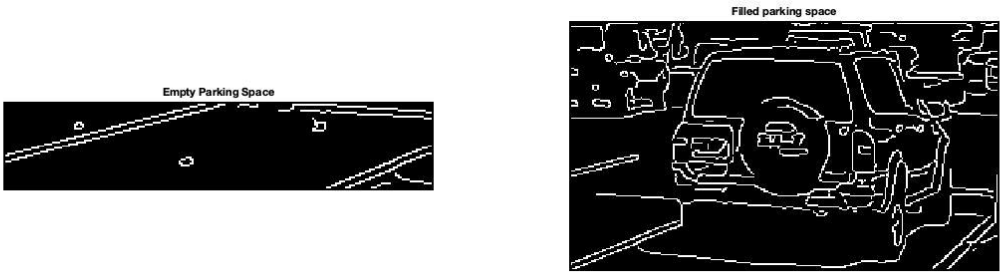

Figure 3.3.3: Vacant(Left) vs Filled(Right) parking space. Significantly more lines are detected on the right image than the left.

From the results shown above, a conclusion was reached that canny edge detector would be an improved solution over RGB analysis. Even in different lighting conditions, the lines would still be detected since cars add more edges to a parking space. Also, this removes the issue of having difficulty to identify lightly colored cars since cars are detected by edges, not color.

3.4) Method 2 - Results

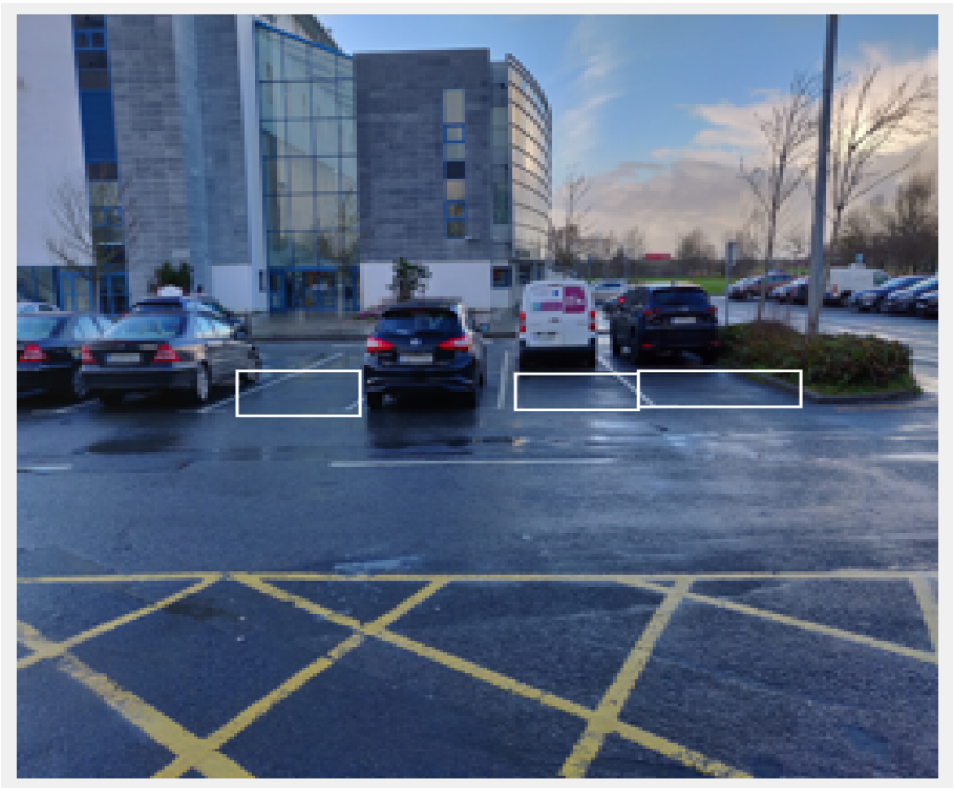

Figure 3.4.1: Canny Edge Detector Results for First Test Image

Figure 5 shows the results of the canny edge detector method for the first test image. For this image, we are just considering the four front parking spaces. The algorithm correctly identified three of the parking spaces as vacant and the parking space with the black car as closed. The results for the above image were obtained with a white pixel threshold of 12% which was sufficient for this test image. The canny edge detector was able to pick up enough edge pixels for the black car to exceed the predefined threshold. However, the results are not always so clean as evidenced by the next test case.

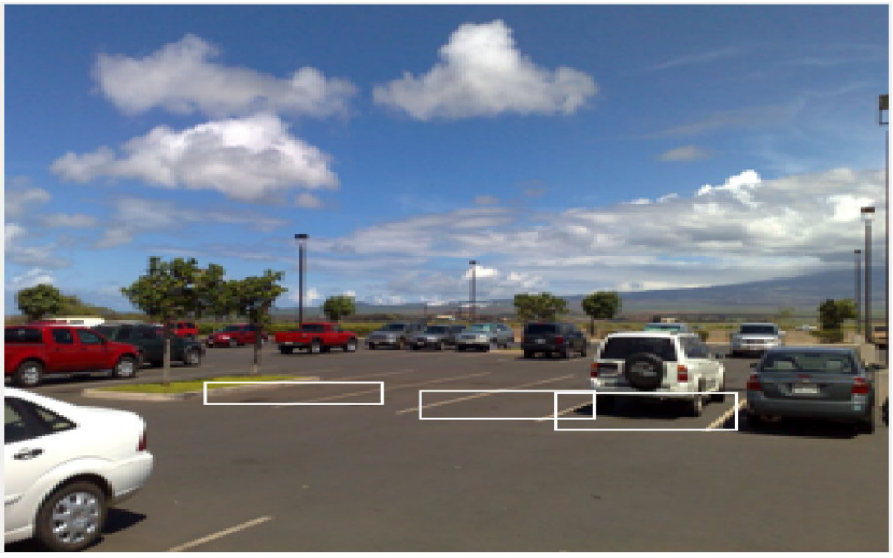

Figure 3.4.2: Canny Edge Detector Results for Second Test Image with Threshold = 12%

In this second example, the results are more problematic. The accuracy of identification depends on the threshold of edge pixels that we choose to apply. Based on these results, we conclude that there are more edge pixels in the parking spot occupied by the white car than in the open parking spot which is not supposed to be the case. Because of the geometry of the masks, it is the shadow of the white car that is most overlapping with its respective parking space. This dark lighting obscures the edges of the car and largely goes unnoticed in the edge detection algorithm. Thus, the algorithm is led to believe there are too little edges here for it to be occupied by a car. We can fix this problem by adjusting the threshold parameter slightly.

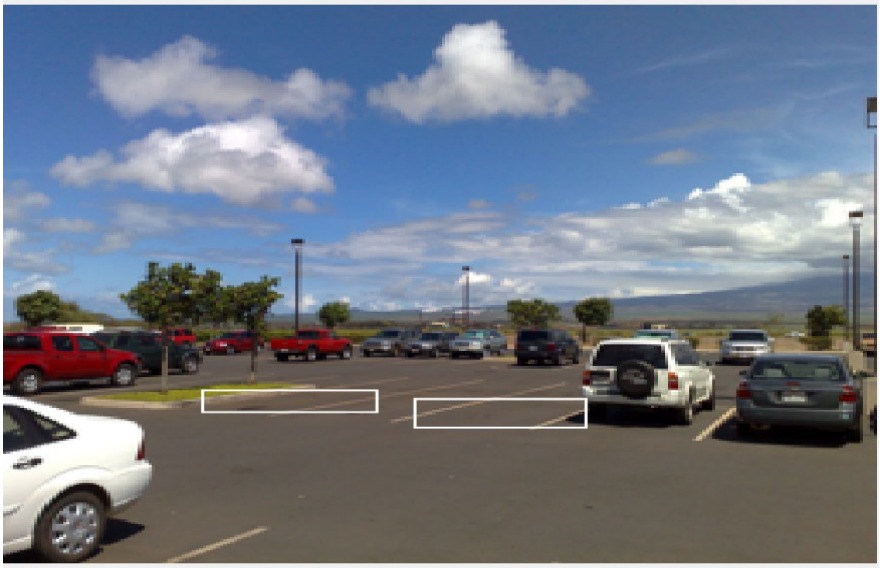

Figure 3.4.3: Canny Edge Detector Results for Second Test Image with Threshold = 10%

Adjusting the threshold does work in this case, but it does not serve as a general solution to this problem. We do not want to require the user to manually readjust the threshold parameter. Perhaps if we had more time to work on this problem, we could devise an algorithm that could detect the shadow pixels of a car and remove those from the parking spot while performing the canny edge detection. Of course, all this could be avoided by simply adjusting the camera orientation.

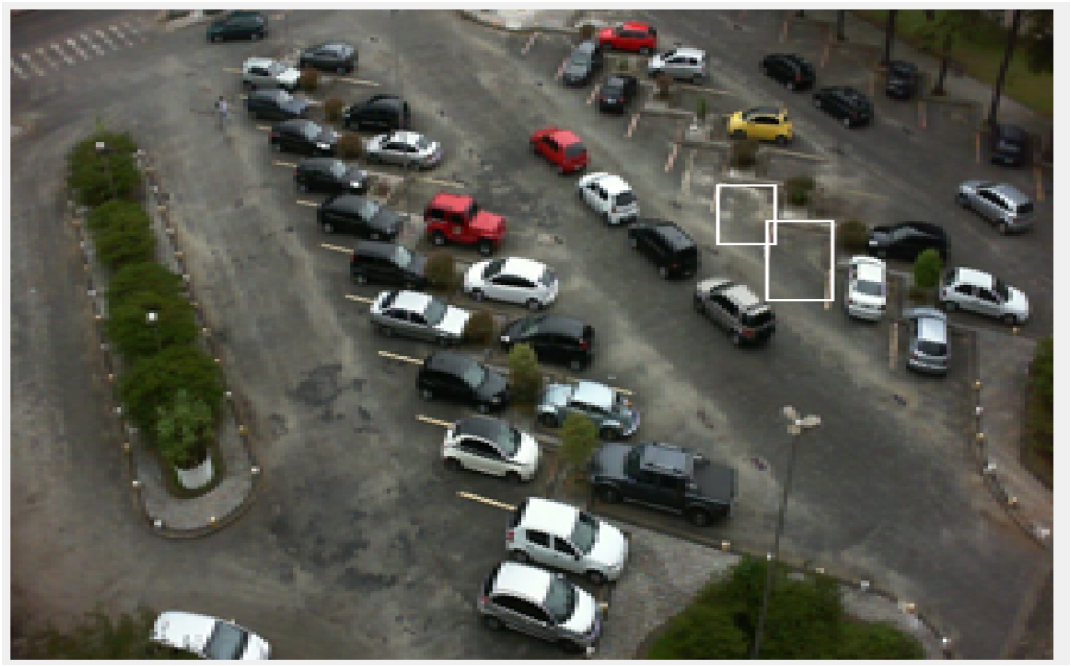

Figure 3.4.4: Canny Edge Detector Results for Third Test Image

The problem of shadows is eliminated by simply taking an aerial capture of the parking lot. As can be seen in figure 8, the algorithm correctly identified all four parking spaces as either open or closed respectively. There are enough detected edges in the white and gray car for the white pixels in each parking spot to exceed the threshold of 12%.

Overall, both methods work decently in identifying open parking spaces. However, we believe that the canny edge detector method is more robust to outside factors such as camera orientation and image lighting. Both methods are susceptible to the car shadows due to a horizontal image capture. However, the color analysis method is far more susceptible to the time of day the image was taken. The gray vector has to be adjusted constantly throughout the day to account for this change whereas the results for the canny edge detector method are less affected by this change.

4) Future Work

While the program is in a semi-working state, there are several tasks that still need to be done to fully complete it. One of the largest issues with our current implementation is the need to manually segment the image. While this method works for the purposes of the project, the system makes for a relatively tedious and poor user experience. We attempted to use the hough transform to automate finding lines relating to the parking spots, but were unable to fully develop those results into segmented parking spaces due to false positives. If we had more time, we could have likely developed a way to reject these false positives through euclidian distance in the feature space or some other form of rejection. This would have given us more reliable parking space data.

Another large problem that we noticed is that our current implementation marks a space as being full if any non-uniform object blocks the parking spot that the program is trying to decide on. This is due to the canny edge detector solely detecting the edges of an object. We ultimately decided to leave this part of the program since a gaussian takes care of most false edges and an object besides a car can still cause a parking space to be rendered unusable. If we had more time, we could have found and implemented a search for feature discriptors that specifically finds cars. If the program were to detect a full spot with no car in it, it could then notify an operator or staff member that there is an object blocking the spot. This would become fairly complicated as we would need a descriptor for any acceptible vehicle type, possibly ranging from vans to motorcycles. While doable, it is outside of the range of what we could reasonable accomplish in this time frame.

One of the other issues we encountered with this project were the various complications due to shadows being cast from the cars and surroundings. While these shadows mainly caused problems for the color based detection method, the shadows also posed a significant problem to the canny edge detection method as well. While the shadows from cars posed minimal errors to the canny method, shadows cast by trees and environmental objects resulted in significant error. These errors were caused by the shadow edges being captured by the

canny edge detector, resulting in false negatives for open spaces. Luckily, our thresholding was able to withstand smaller shadows, but color normalization methods may be necessary for the next stage of the project. Another solution would be to use the images we currently have to construct a more robust car detection feature set using methods such as Adaboost. As our image set contains over a thousand images, there should be enough parking space samples to use as training data..

5) Conclusions

Through the process of working on this project, we were able to put into practice topics learned in lecture such as the Hough transform, color spaces analysis, and segmentation. The portion of the project that stood out the most was how difficult segmentation was. More specifically, isolating individual parking spaces. The first method we thought of was to use a color space segmentation method, such as k-means, to find the parking spaces automatically. However, we quickly realized that such a method would not be very effective for in isolating a parking space. We then moved to segmenting the parking spaces using MatLab's roipoly function. While tedious, the roipoly did yield a space that was easy to work with. After that, it was a matter of finding an effective way to detect whether the segment had a car in it. While the canny edge detection method is not very robust, it does provide relatively consistent results in a controlled environment. A controlled invironment in this case is an area without background clutter and without shadows that partially cover parking spots.

One of the things we realized while working on this project is that it would have been better to have limited the image conditions early in the project. This is because most of the issues we encountered while working on this project were due to the multitude of perspectives we were trying to get the program to work for. This introduced various error sources such as the car shadows and background objects that would not be present in a top down view. As we were too focused on finding something that would work for every possible angle and case, the scope of the project led to us having overgeneralized solutions to each problem. As such, the solutions can easily be influenced by lighting differences and stray objects. They are not robust. The solution came closer to calculating if an object was in a pre-marked spot of an image rather than detecting whether a parking spot was available provided an image of the parking spot.

Overall, the project worked to provide us a view into just how difficult image segmentation and object recognition can be. While our lab projects typically focused around a specific use case for the algorithms we learned in class, this project allowed us to see what it is like to try and classify an object with wildly varying conditions. A large amount of the issues we experienced may have been offput if we were able to find feature descriptors using a training method, as we did have a large amount of training data at our disposal. Unfortunately, we did not have the time to identify a feature set that could be used to identify parking spaces and cars from such a wide array of perspectives. Therefore, while the method implemented in this project is feasible for parking space detection in controled environments, it is not currently feasable for real world use.

6) Appendix

6.1) Work Division

Nathan Orloff:

- Parking Space Detection - ROI
- Parking Space Detection - Hough Transform

Tushar Sharma:

- Method 1 - Color Analysis
- Method 2 - Canny Edge Detector

Conrad Kinsey:

- Future Work
- Conclusion

Micah Jeffries:

- Introduction
- Method 1 - Results
- Method 2 - Results

6.2) References

[1] M. Lopez, T. Griffin, K. Ellis, A. Enem, and C. Duhan, “Parking lot occupancy tracking through Image Processing,” *EPiC Series in Computing*, vol. 58, pp. 265–270, 2019.

6.3) Code

Method 1 - Color Analysis

(This code was all written by us)

```
%Read in input images
input = imread('p2.jpg');
mask = imread('s2_roi.jpg');

%Resize Input Images
input = imresize(input,[255 255]);
mask = imresize(mask, [255 255]);

% Break the image up into red, green and blue planes
R = input(:,:,1);
G = input(:,:,2);
B = input(:,:,3);
threshold = 15;

[park_spots, num_park] = bwlabel(mask);
%Find Box Coordinates for each part
stats = regionprops('table', logical(mask), 'BoundingBox');
stats_table = table2array(stats);
park_count = 0;
hold off
imshow(input)

% For each parking space...
for i=1:height(stats_table)

    % Get area and location of parking space
    loc = stats_table(i,:);
    area = loc(3) * loc(4);
    r = uint8(loc(1)); c = uint8(loc(2)); w = uint8(loc(3)); h =
uint8(loc(4));

    if (area > 40) %ignore noise pixels

        % Extract RGB planes of parking space
        extract_R = R(r:r+w, c:c+h);
        extract_G = G(r:r+w, c:c+h);
```

```
        extract_B = B(r:r+w, c:c+h);

        % Find average value of each plane
        average_R = sum(extract_R, 'all')/area;
        average_G = sum(extract_G, 'all')/area;
        average_B = sum(extract_B, 'all')/area;

        % Form two vectors for comparison
        average_color = [average_R, average_G, average_B];
        free_space = [80, 92, 71];

        % If the absolute difference between the two vectors <
threshold,
        % then the parking space is most likely open
        if ~(abs(average_color(1) - free_space(1)) > threshold &&
abs(average_color(2) - free_space(2)) > threshold &&
abs(average_color(3) - free_space(3)) > threshold)
            rectangle('Position', loc, 'EdgeColor','w',
'LineWidth',1);
            park_count = park_count + 1;
        end
    end
end
```

Method 2 - Canny Edge Detector
(This code was all written by us)

```
%Read in input images
input = imread('p5.jpg');
mask = imread("s5_roi.JPG");

%Resize Input Images
input = imresize(input,[255 255]);
mask = imresize(mask, [255 255]);

% Break the image up into red, green and blue planes
R = input(:,:,1);
G = input(:,:,2);
B = input(:,:,3);
threshold = .12;

[park_spots, num_park] = bwlabel(mask);
%Find Box Coordinates for each part
stats = regionprops('table', logical(mask), 'BoundingBox');
stats_table = table2array(stats);
park_count = 0;
hold off
imshow(input)

% For each parking space...
for i=1:height(stats_table)

    % Get area and location of parking space
    loc = stats_table(i,:);
    area = loc(3) * loc(4);
    r = uint8(loc(1)); c = uint8(loc(2)); w = uint8(loc(3)); h =
uint8(loc(4));

    if (area > 40) %ignore noise pixels
        box = input(r:r+w, c:c+h);

        % perform edge detection on parking space
        bw = edge(box, 'canny');
```

```
        % Find ratio of white pixels in parking space
        whitePixels = sum(input(bw == 1));
        blackPixels = sum(input(bw == 0));
        morePixels = whitePixels/(whitePixels + blackPixels);

        % If ratio of white pixels < threshold, it is most likely an
open
        % parking space
        if morePixels < threshold
            rectangle('Position', loc, 'EdgeColor','w',
'LineWidth',1);
            park_count = park_count + 1;
        end
    end
end
```

get_spaces

```
(This code was all written by us)

image = imread("spaces/parking1.jpg");
i_gray = rgb2gray(image);

i_gray_filter = medfilt2(i_gray,[5 5]);
i_gray_filter = imfilter(i_gray_filter, fspecial('Gaussian', [9 9]));

figure;
imshow(i_gray_filter)

threshold = 210;
i_thresh = i_gray_filter > threshold;

figure;
imshow(i_thresh)


%dilate parking lines
se = strel('disk', 6);
dilatedBW = imdilate(i_thresh, se);

figure;
imshow(dilatedBW, [])

%remove anything non-parking lines
se = strel("line", 18, 90);
erodeBW = imerode(dilatedBW, se);

se = strel("disk", 2);
erodeBW = imerode(erodeBW, se);

%strengthen parking lines again
se = strel("line", 8, 90);
erodeBW = imdilate(erodeBW, se);

figure;
```

```
imshow(erodeBW, [])

figure;
imshow(i_gray), hold on
%get hough transform
[H, theta, rho] = hough(erodeBW);
numPeaks = 5;
%get hough peaks
peaks = houghpeaks(H, numPeaks, 'Threshold', 30);
%get hough lines
lines = houghlines(erodeBW, theta, rho, peaks, 'FillGap', 5,
'minLen', 15);

%draw lines on image
for k=1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:,2), 'LineWidth', 1, 'Color', 'r');
end
hold off
```