Status Report

Will Poeppelmeyer

After setting out to create a program to count and identify the empty spaces in a parking lot photo, the plan was to start simple and see how robust of a solution could be created. The simplest way to begin was to find an image with vertically aligned cars and an overhead view. An initial test image was found and labeled as “simple_lot.jpg”.

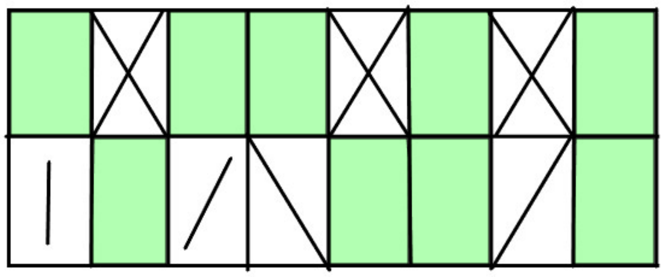
The first code to work with was to experiment with the edge detection functions in MATLAB. MATLAB was chosen to be the programming environment due to its convient image processing tools and fast development environment. Each of the algorithms available with the edge() function were tested on “simple_lot.jpg” with different threshold values until a suitable choice was determined. The Canny method was ideal in this situation with a threshold setting of 0.3 as it kept most of the desired edges while removing everything else.

Aside from edge detection, a first program was developed as a means to develop skills in segmentation and analysis for the simplest representation of a parking lot. Since regions of the dummy image were completely closed off, I found a strategy based on the bwlabel() function that labels all the closed regions of a binary image. I gained knowledge in targeting and analyzing unique parts of an image. The layout of the test image “fake_car_lot.jpg” affected the way the program was designed. A simple solution that offered little carryover to an actual parking photo was used. Afterwards, a more relevant method would be found for “simple_lot.jpg”. This first program works by determining if each enclosed region’s max length and width make the ratio of a parking space, and that the space is sufficiently empty from the binary image values.

```
%This program detects empty spaces in a simplified parking grid
%It works by segmenting closed regions
%Based on https://www.youtube.com/watch?v=XrC1r80-p-k
A = imread('fake_car_lot.jpg');
B = im2bw(A);
label = bwlabel(B);         %label closed regions
```

```
numOpen = max(max(label));  %start as all regions counted empty
imshow(A);   hold on;
s = regionprops(label, 'Extrema');
for i = 1 : max(max(label));         %examine each region
  [row, col] = find(label == i);
  len = max(row) - min(row) + 2;
  breadth = max(col) - min(col) + 2;
  target = uint8(zeros([len breadth]));
  sx = min(row) - 1;
  sy = min(col) - 1;
  for j = 1 : size(row, 1)         %assign data to target array
    x = row(j, 1) - sx;
    y = col(j, 1) - sy;
    target(x, y) = A(row(j, 1), col(j, 1));
  end
  ratio = len / breadth;
  if ratio < 1.4 | ratio > 1.9 %check if not correct possible rectangle
    numOpen = numOpen - 1;
  else
    emptiness = mean2(im2bw(target)); %check percentage of 1s
    if emptiness < 0.95
      numOpen = numOpen - 1;
    else
      patch(s(i).Extrema(:,1), s(i).Extrema(:,2), 'g', 'FaceAlpha', 0.3);
    end
  end
end
hold off;
M = [num2str(numOpen), ' empty spaces'];
disp(M);
```

```
9 empty spaces
```



Published with MATLAB® R2018b

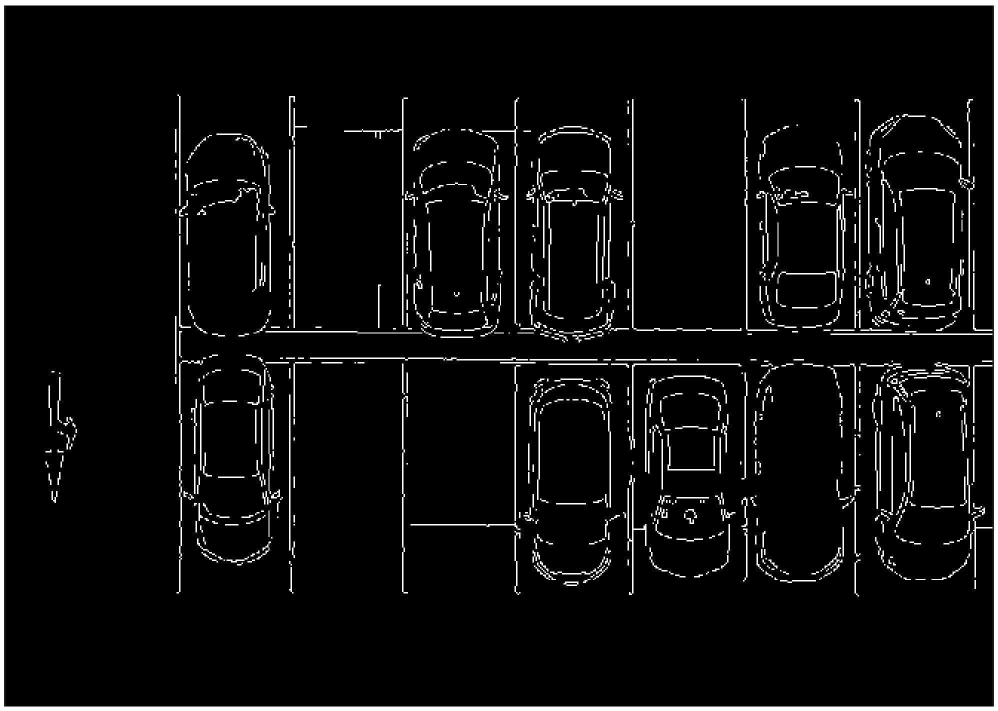
To begin building a program that would work with real photos of parking lots, “simple_lot.jpg” was studied in order to find a strategy . Since a good edge detection result was already found for this type of image, the idea was to use the binary edge image to segment the spaces based on the parking space lines. In this image, the vertical side edges of each space could be isolated using the function for the Hough transform with a near vertical range of theta values. This produced an array of structure for each of the line segments found for a given threshold. The desired number of lines running across the whole image defined by their rho – theta combination was needed as an argument in houghpeaks(), which extracted useful results from the initial Hough data array. The houghlines array was then sorted by horizontal value, rho, in order for the parking slots to be analyzed from left to right.

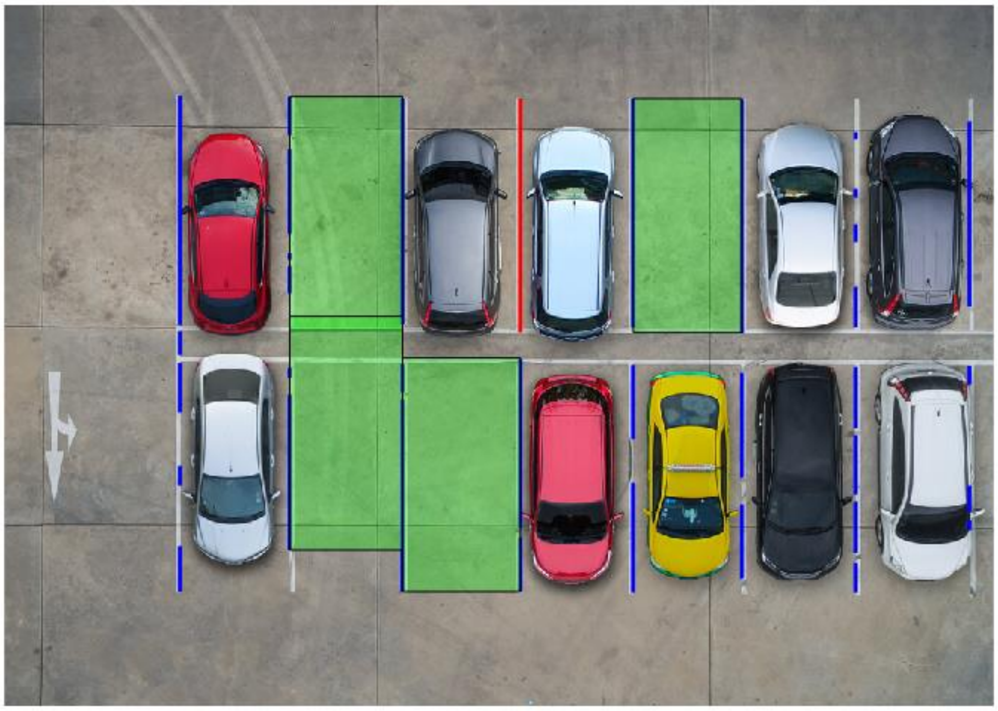
The segments and analysis were performed by finding the longest edge segment, then starting at the top of the leftmost Hough line and using imcrop() of a rectangle with the max length and the horizontal distance between Hough lines. The bottom point of the Hough line was then used as the bottom-left point in a rectangle to analyze the parking space below the previous one. Again, the percentage of 1’s in the binary was used to determine if the space was empty. The program loops through this process and the Hough lines array to continue analyzing until the rightmost Hough line is reached.

```
%This program finds parking lines with the Hough transform.
%Parking spots are found by moving over from line segment points.
%The main loop is designed for 2 rows of spaces with vertical cars.
%Spaces without both lines in the image will be ignored.
I = imread('simple_lot.jpg');
G = rgb2gray(I);
BW = edge(G, 'Canny', 0.3);
[H, T, R] = hough(BW, 'Theta', -1:0.5:1);
P = houghpeaks(H, 8, 'threshold', ceil(0.3 * max(H(:))));
lines = houghlines(BW, T, R, P, 'FillGap', 5, 'MinLength', 7);
T = struct2table(lines);
sortedT = sortrows(T, 'rho');
sortedLines = table2struct(sortedT);
figure, imshow(BW),  figure, imshow(I),   hold on;
max_len = 0;  numOpen = 0;  percent1 = 0.05;
for k = 1 : length(lines)
  xy = [lines(k).point1; lines(k).point2];
  plot(xy(:, 1), xy(:, 2), 'LineWidth', 2, 'Color', 'blue');
  len = norm(lines(k).point1 - lines(k).point2);
  if len > max_len  %Check for longest segment
    max_len = len;
    xy_long = xy;
  end
end
plot(xy_long(:, 1), xy_long(:, 2), 'LineWidth', 2, 'Color', 'red');
rho = 0;
for i = 1 : length(sortedLines) %run through entire Hough lines array
  if rho ~= sortedLines(i).rho  %if new rho segment, else move on
    rho = sortedLines(i).rho;
    u = i + 1;
    while u <= length(sortedLines) %Look for next rho in array
      if rho ~= sortedLines(u).rho %if found, now segment up then down
        nextRho = sortedLines(u).rho;
        deltaX = nextRho - rho;
        Seg = imcrop(BW, [sortedLines(i).point1(1) ...
                          sortedLines(i).point1(2) deltaX max_len]);
        check = mean2(Seg);
        if check < percent1 %percentage of 1s
          numOpen = numOpen + 1;
          X = [sortedLines(i).point1(1) ...
               sortedLines(i).point1(1) ...
               sortedLines(i).point1(1)+deltaX ...
               sortedLines(i).point1(1)+deltaX];
          Y = [sortedLines(i).point1(2) ...
               sortedLines(i).point1(2)+max_len ...
               sortedLines(i).point1(2)+max_len ...
```

```
                sortedLines(i).point1(2)];
          patch(X, Y, 'g', 'FaceAlpha', 0.3);
        end
        %Find lowest point for this rho and segment low space
        w = i;
        while rho == sortedLines(w).rho
          lowestPoint = sortedLines(w).point2;
          w = w + 1;
        end
        Seg = imcrop(BW, [lowestPoint(1) lowestPoint(2)-max_len ...
              deltaX   max_len]);
        check = mean2(Seg);
        if check < percent1 %percentage of 1s
          numOpen = numOpen + 1;
          X = [lowestPoint(1)  lowestPoint(1) ...
                lowestPoint(1)+deltaX  lowestPoint(1)+deltaX];
          Y = [lowestPoint(2)  lowestPoint(2)-max_len ...
                lowestPoint(2)-max_len  lowestPoint(2)];
          patch(X, Y, 'g', 'FaceAlpha', 0.3);
        end
        break %Segmenting finished for this rho, now exit while loop
      end
      u = u + 1;
    end
  end
end
hold off;
if numOpen == 1
  text = ' empty parking space has been detected.';
else
  text = ' empty parking spaces have been detected.';
end
disp([num2str(numOpen), text]);
```

```
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
4 empty parking spaces have been detected.
```

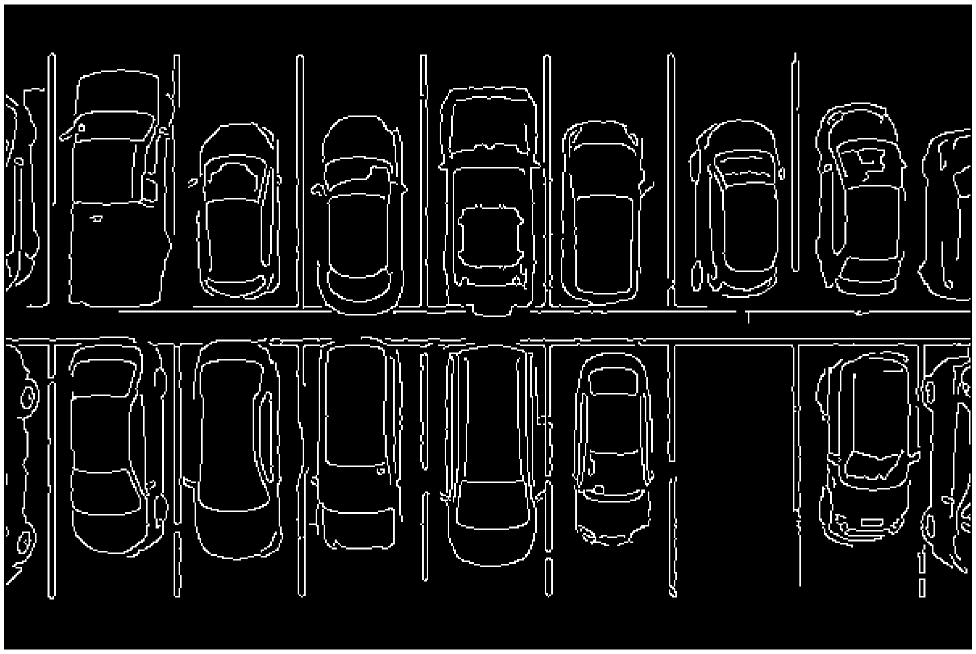


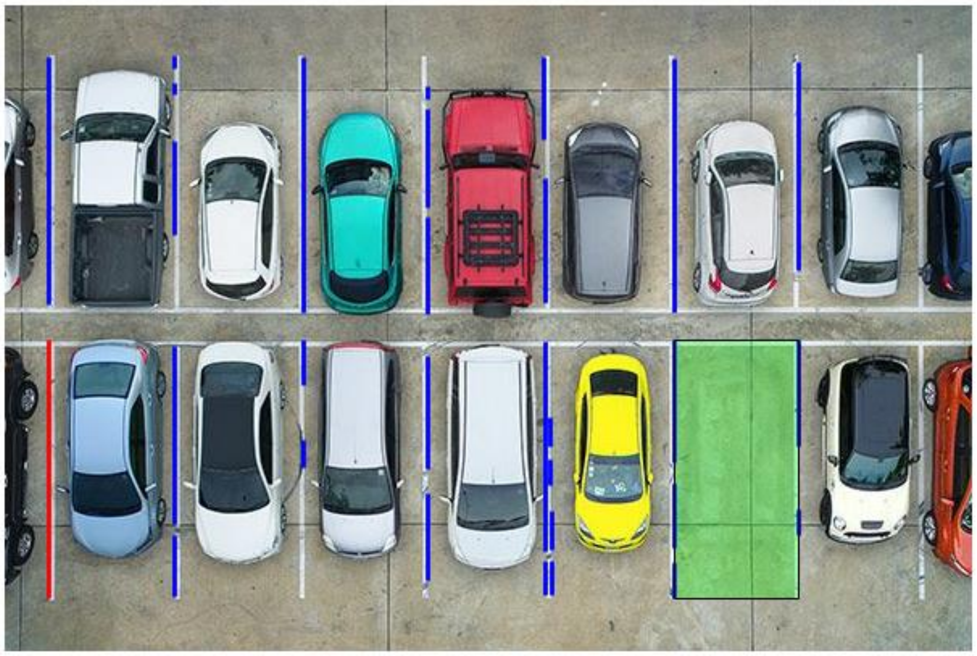


The longest edge segment is shown in red

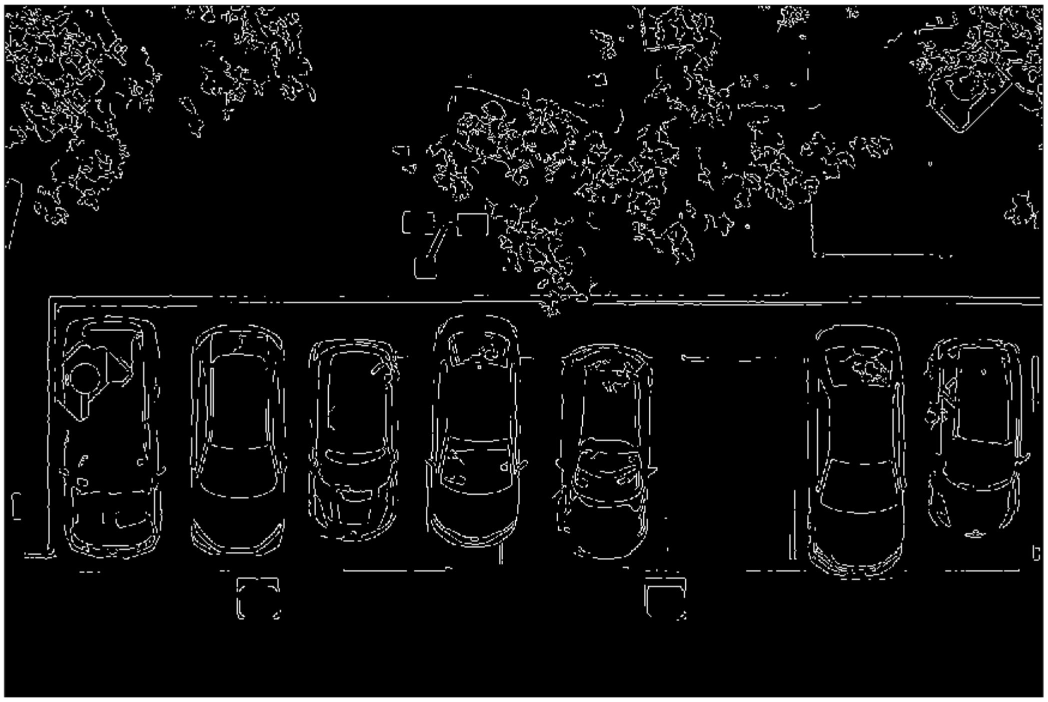
When testing with other pictures:

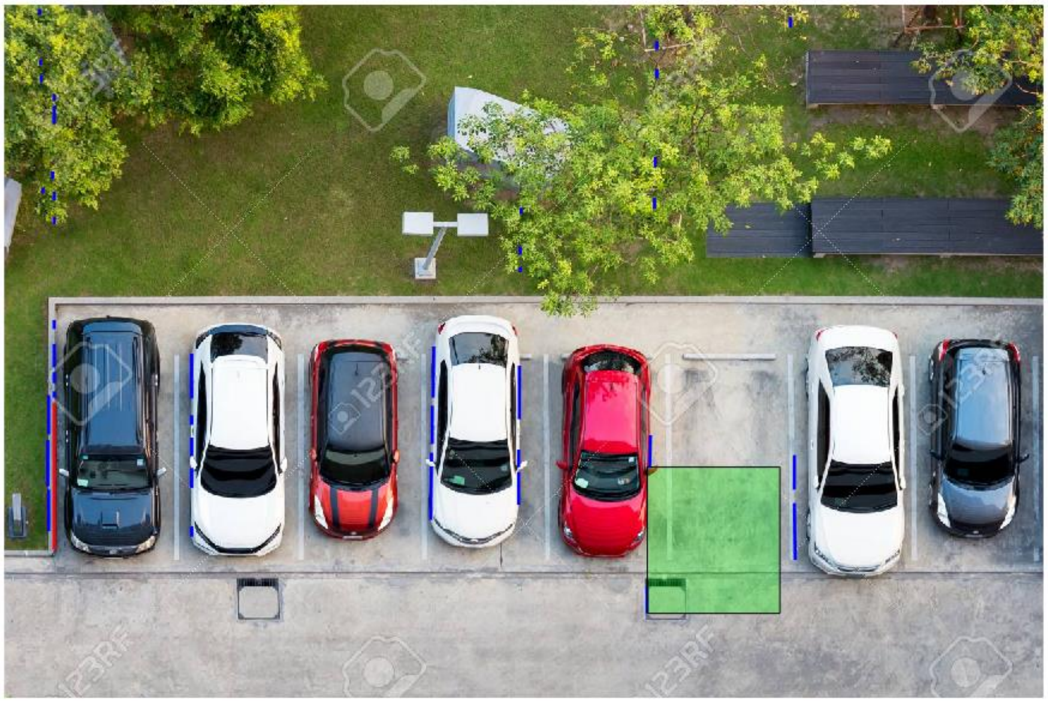
```
1 empty parking space has been detected.
```





```
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
1 empty parking space has been detected.
```





In the previous image it is apparent that edge detection did not eleminate the unwanted edges, and the desired parking line edges were below the threshold that worked well on the past two images. The longest segment was shorter than the parking lines, and the top eight Hough lines did not match well with the parking lines.

The next image shows more ability to have the Hough lines on track, but the edge detection threshold was too high. In order to improve on this code, it will be important to find a more versatile way of creating the binary image. The way that the final Hough lines are determined will also need improvement; a large number of lines could possibly be analyzed, with some being ruled out due to proximity. Also, it is necessary to improve the method for determining if a space is empty through color processing.

```
2 empty parking spaces have been detected.
```