Stevens Institute of Technology

Department of Computer Science

CS558: Computer Vision

Homework Assignment 1

Submitted by:
Hadia Hameed
CWID: 10440803

Submitted to:
Prof. Xinchao Wang

April 6, 2019

1 Problem 1 - Pre-processing:

Download the image provided. You must implement two methods for detecting lines (Problems 2 and 3). Before extracting the lines, you need to detect potential points on them. Apply a Gaussian filter first and use the Sobel filters as derivative operators. Threshold the determinant of the Hessian and then apply non-maximum suppression in 3 x 3 neighborhoods. Ignore pixels for which any of the filters falls even partially out of the image boundaries.

Solution:

1.1 Loading and resizing image:

```
filename='road.png';
I = imread(filename);
im = im2double(I);
```

1.2 Gaussian Smoothing:

```
g = myFilter('gaussian',3,1);
im_g = myConv(im,g);
```

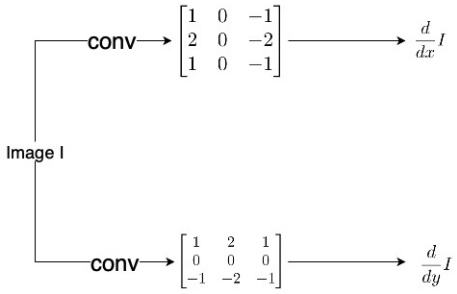
(a) Original Image (b) Smoothed image

Figure 1: Smoothing using Gaussian Filter with size = 3x3 and $\sigma = 1$

1.3 Sobel Filtering:

```
sobel = myFilter('sobel');
sobel_fx = sobel.x;
sobel_fy = sobel.y;

Ix = myConv(im_g,sobel_fx);
Iy = myConv(im_g,sobel_fy);
```



(a) Derivative w.r.t x

(b) Derivative w.r.t y

Figure 2: Sobel Filtering using 3x3 sobel filter

1.4 Thresholding Hessian Matrix (Harris Corner Detector)

```
thresh = 0.01; %threshold for selecting corner response values
[rows,cols,output] = myHarrisDetector(Ix,Iy,thresh);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [rows,cols,output ] = myHarrisDetector(Ix,Iy,thresh)
  g = myFilter('gaussian',5,1);
  Ixx = myConv(Ix.^2,g);
  Iyy = myConv(Iy.^2,g);
```



```
  Ixy = myConv(Ix.*Iy,g);

  im1 = zeros(size(Ixx,1), size(Ixx,2));
  for x=1:size(Ixx,1)
    for y=1:size(Ixx,2)
       % 1) Define at each pixel(x, y) the matrix H
       H = [Ixx(x, y) Ixy(x, y); Ixy(x, y) Iyy(x, y)];

       % 2) Compute the response of the detector at each pixel
       R = det(H);

       % 3) Threshold on value of R
       if (R > thresh)
          im1(x, y) = R;
       end
   end
  end
  filt = ones(3);
  filt(2,2) = 0;
  %non-maxima suppression
  output = im1 > myImageDilation(im1,filt);
  [rows,cols,output] = find(output);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ o ] = myImageDilation(I,filt)
  [p,q]=size(filt);
  [m,n]=size(I);
  kk =  zeros(m+2,n+2);
  kk(2:end-1,2:end-1) = I;
  I = kk;

  temp= zeros(m,n);
  for i=1:m
    for j=1:n
      subset_g = I(j:j+p-1,i:i+q-1).*filt;
        for k=1:p
            for l=1:q
                if(filt(k,l)==1&&subset_g(k,l)>0)
                    temp(j,i)=max(max(subset_g));
                end
            end
        end

    end
  end
  o = temp;
```



```
end
```

1.4.1 Results:

Two techniques were tested in Harris corner detection in order to reduce false positive lines.

1. **Adjusting the Hessian threshold**: The first one was the base case in which the corner detector was used as it is and different values of threshold were tried to get the best set of corners as shown in Fig.3

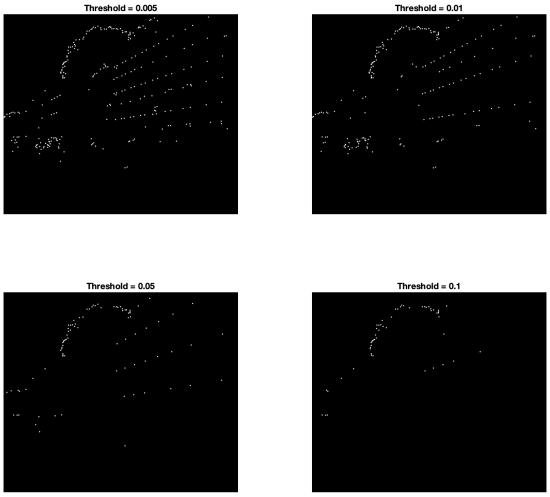


Figure 3: Different thresholds tried for Harris corner detector. The best one was $threshold_{harris} = 0.005$

2. **Removing crowded points**: As shown in the left subfigure in Fig.4, the corner points detected around the tree result in a lot of false positive lines being detected during line-fitting using RANSAC. The subfigure on the right in Fig.4 shows how a subset of the most relevant points were selected based on a new threshold i.e. $threshold_{cluster} < 10$, removing densely crowded corner points.

Figure 4: Harris corner detection with $threshold_{harris} = 0.03$, followed by dense cluster removal. All points less than $threshold_{cluster} < 10$ units apart were removed.

2 Problem 2 - RANSAC:

Apply RANSAC on the points detected above to find four lines with strong support. Since it is intractable to guarantee that you found the four lines with the most support, use your best judgement to determine that the outputs are correct. One possible implementation is to run RANSAC until it finds a line with a sufficient number of inliers. Make sure that the same inliers are not used again. Plot the line segments in the image by connecting the two extreme inliers of each line. Also plot the inliers as 3 x 3 squares.

2.1 Algorithm:

Input arguments:

- $[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$ = Set of data points
- d = minimum number of inliers for the algorithm to terminate.
- t = Threshold for selecting inliers based on their distance from the line.

The initial number of points for fitting the line in each iteration is fixed i.e. $s = 2$ because it's value is typically equal to the minimum number needed to fit the model, which in this case is a straight line which needs at least 2 points [1].

Steps for the algorithm:

1. **Random Sampling**: Randomly choose $n = 2$ number of samples i.e. $[(x_1, y_1), (x_2, y_2)]$ from the set of corner points detected in Problem I.
2. **Line Fitting**: Fit a line through $[(x_1, y_1), (x_2, y_2)]$ using least squares method.
3. **Error Calculation**: Calculate the distance of the rest of the points from the line obtained in Step II.
4. **Inlier Selection**: Select the points as inliers for which the distance is less than the threshold t.
5. **Termination Condition:** Terminate the procedure if the number of inliers is $>= d$

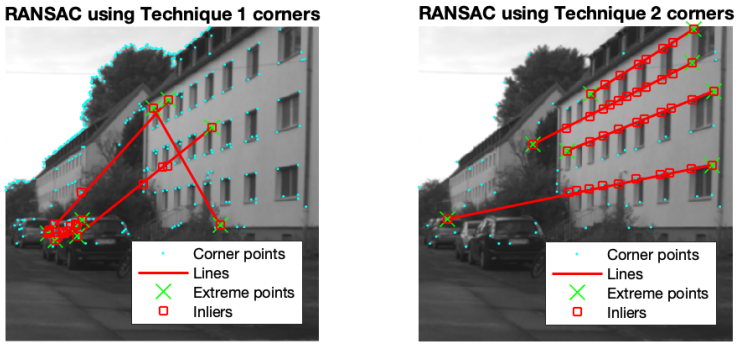
The steps are described in Algorithm 1. The function **linearRegression** on line 8 takes two vectors and calculates $B = X\backslash Y$ where B contains the model parameters which in case of a line are $m = slope$ and $y = intercept$ [2].

Algorithm 1 RANSAC algorith for line fitting.

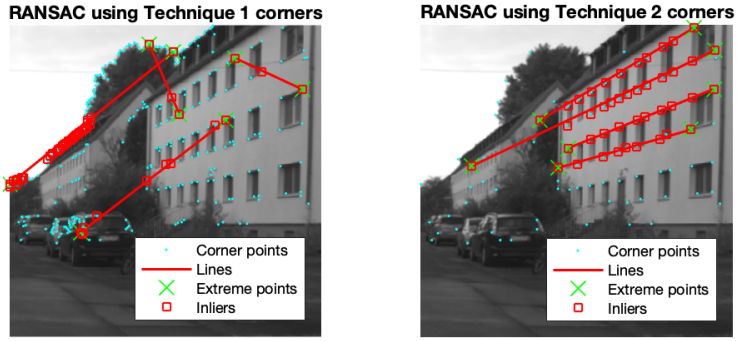
```
1:  procedure RANSAC
2:      [X,Y]  ← data points
3:      d ← minimum number of inliers for the algorithm to terminate
4:      t ← distance threshold for selecting inliers
5:      no_of_inliers ← 0
6:      while no_of_inliers is < d do
7:          [xx,yx]  ← Randomly select two data points from set (X,Y)
8:          [m,c] ← linearRegression(xx,yy)
9:          inliers ← 0
10:         for all samples in (X,Y) except [xx,yy] do
11:             y_p ← x * m + c
12:             Δy ← |y_p − y|
13:             if Δy < t then
14:                 inliers ← inliers + 1
15:         no_of_inliers ← inliers
        return [m, c]
```

2.2 Results:

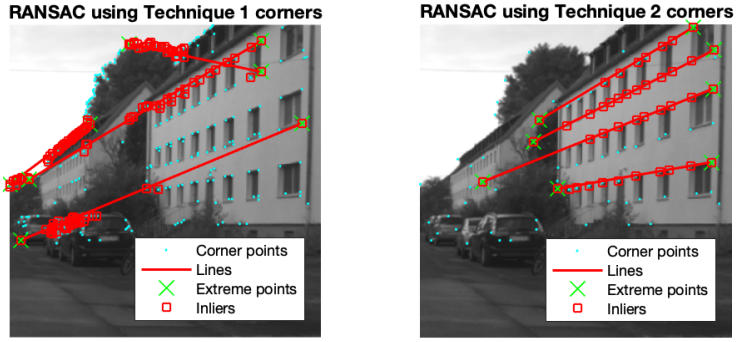
The overall results of applying RANSAC to the corner points obtained from the two Techniques mentioned in Section I, are shown in Fi



(a) $dist_{min} = 3$



(b) $dist_{min} = 5$



(c) $dist_{min} = 5$

Figure 5: The images in the left column show the lines obtained after RANSAC was applied to all the corners (without removing densely crowded clusters) for different values of $dist_{min}$ which is the minimum distance of a point from the line for it to be considered as an inlier. The images in the right column show better results obtained after removing clustered corner points around non-linear objects like trees, cars etc.

2.2.1 Results for the four passes

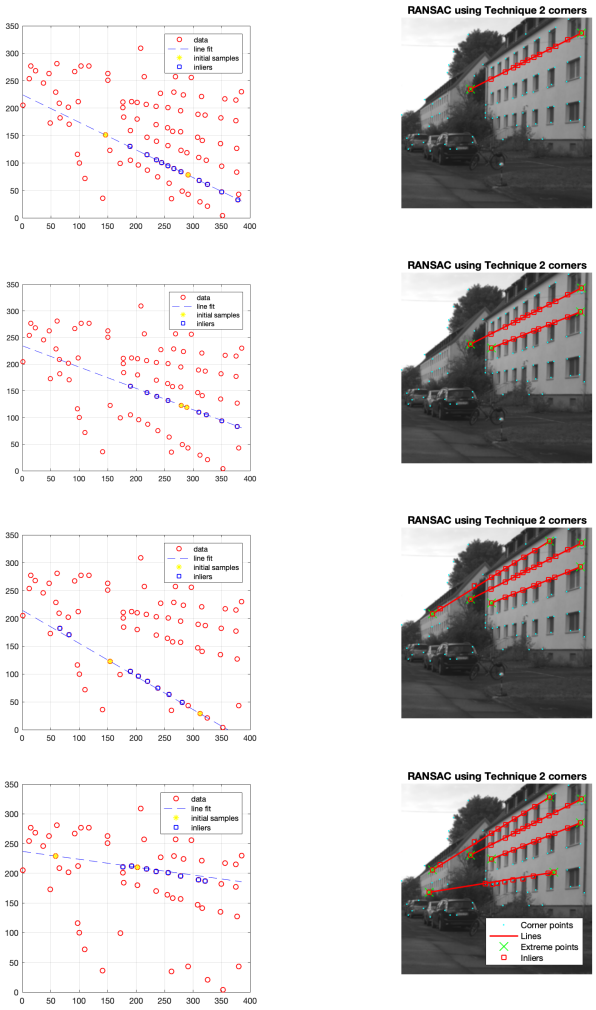
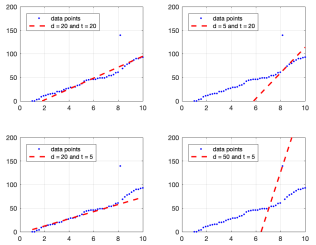
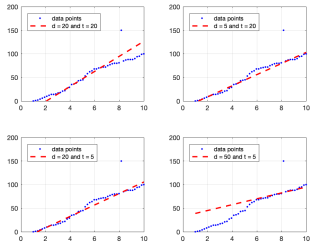


Figure 6: Results for RANSAC for three iterations with $d = 0.1 * length(data)$ and $dist_{min} = 8$. Parameter d is the minimum number of points that need to be inliers for a particular model to be selected and is adaptive as it signifies that for each iteration at least 10% of the data points must be inliers for the particular model to be selected. $dist_{min}$ is the minimum distance of a point from the line for it to be considered as an inlier and *data* is the remaining data points left after removing the inliers used for each of the three lines.

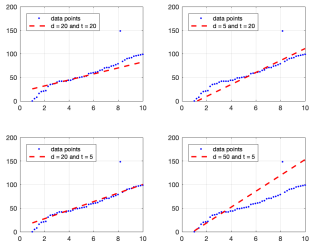
3 Choosing parameters:



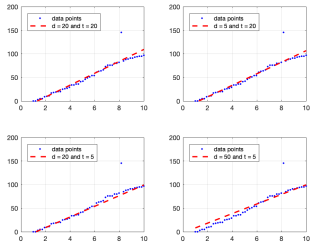
(a) Iteration = 1



(b) Iteration = 2



(c) Iteration = 3



(d) Iteration = 4

Figure 7: Results for four different configurations for $[d, dist_{min}]$ over four iterations for a toy example. The four configuration for $[d, dist_{min}]$ were [(20,20),(5,20),(20,5),(50,5)]. The variation in the results across several iterations show that the choice of initial query (randomly selected minimal subset) is crucial in RANSAC. Therefore, RANSAC does not always give the same results as the initial query may vary for different passes.

3.1 Variation in Results:

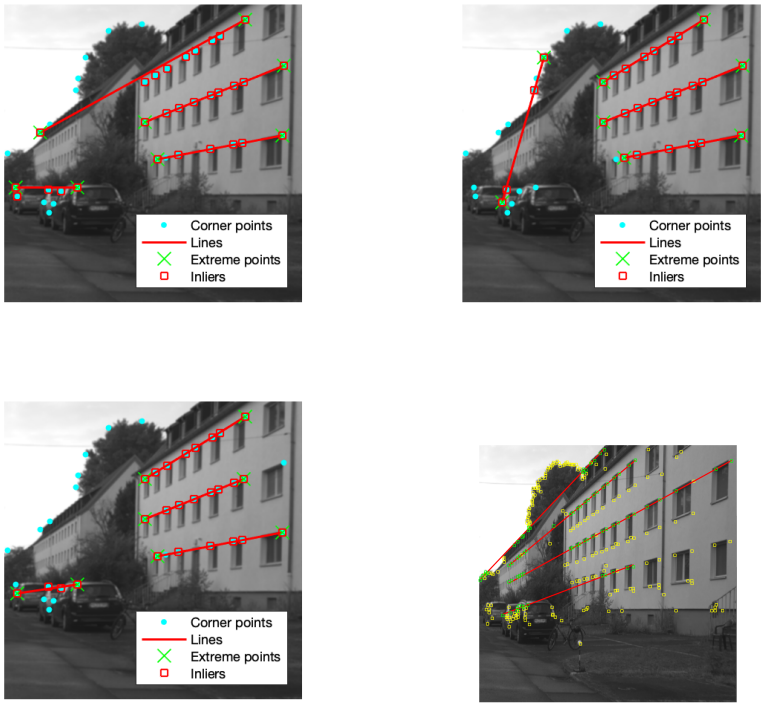


Figure 8: Results for RANSAC for different passes. The results show that the final output varies depending on the initial queries selected.

3.2 Modifications to RANSAC:

The following modifications were made to the parameters used in RANSAC in order to improve accuracy:

1. **Adaptive d:** As RANSAC is being used to fit four models and after each of the iterations the inliers selected are removed from the dataset, the value of d is kept dynamic instead of static. In each of the four iterations, $d = 10\%$ of the total number of the points in the dataset during the current iteration.

2. **Error Thresholding:** As show in Fig.9, in order to avoid having false positive inliers, their distance from the initial query points (yellow dots) is thresholded. Only those points are considered as inliers for which

$|y_{predicted} - y_{actual}| < dist_{min}$ and they are in the neighborhood of the initial query points.

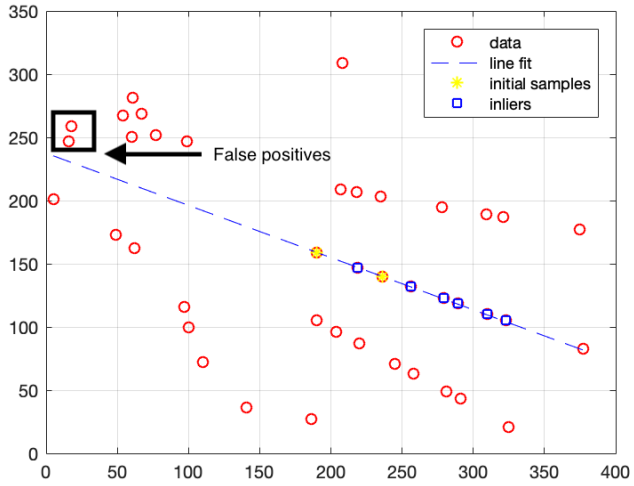


Figure 9: Points that are close to both the line and the initial query points (yellow dots) are considered as inliers (blue dots). Therefore the points in the black bounding box are false positives and are not selected as inliers.

4 Problem 3 - Hough Transform:

Apply a Hough transform to detect the four lines with the strongest support using the polar parameterization of the line (not the point coordinates). Plot the results as above.

4.1 Canny Edge Detection:

Performing canny edge detection before extracting lines through Hough Transform can improve accuracy [3]. The steps for Canny Edge detection are described below:

4.1.1 Algorithm:

Input arguments:

- I = Input image
- $lowThreshold$
- $highThreshold$

Steps for the algorithm [4]:

1. **Smoothing**: Smooth image with Gaussian Filter.
2. **Derivative**: Compute derivative of filtered image.
3. **Magnitude and Gradient:** Find magnitude and gradient of filtered image.
4. **Non-maximum Suppression:** Thin wide "ridges" down to single pixel width.
5. **Hysteresis Thresholding:** Define two thresholds: $thresh_{high}$ and $thresh_{low}$. Use the high threshold to start edge curves and the low threshold to continue them.

4.1.2 Discretization of directions:

In order to do non-maxima suppression, the gradient directions need to be discretized into four bins i.e. $[0°, 45°, 90°, 135°]$ corresponding to the eight neighbors of every pixel in the image. This is illustrated in Fig.10

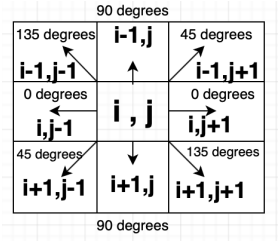


Figure 10: Four discrete gradient directions for non-maxima suppression in Canny Edge Detection.

4.1.3 Results for Edge Detection:

(a) $edge(image, `canny', 0.1)$ (b) $myCannyEdge(image, thresh_{high}, 0.4 * thresh_{high})$

Figure 11: Results for Canny Edge Detection using $thresh_{high} = 0.1$ and $thresh_{low} = 0.4 * thresh_{high}$. Fig. 9(a) shows results from a standard MATLAB function $edge(image, `canny', 0.1)$ and Fig. 9(b) shows the results for the edge detection implemented from the scratch.

4.2 Hough Transform:

Input arguments:

- I = Binary image

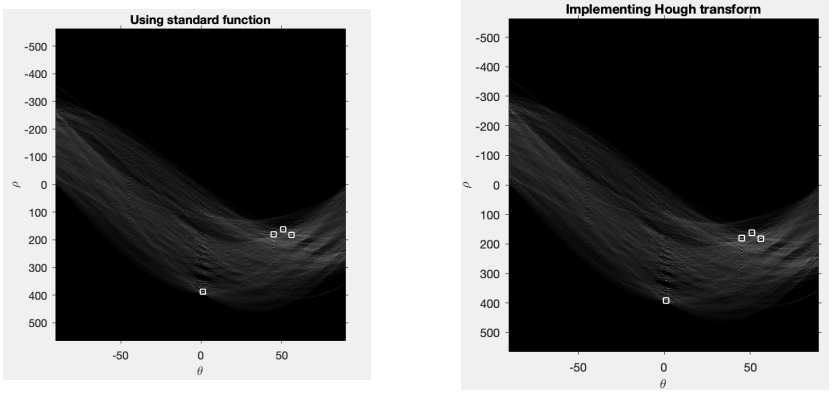
Steps for the algorithm:

1. **Discretization of θ and ρ**: Create parameter vectors $theta = [-90, 90)$ and $\rho = [-\rho_{max}, \rho_{max}]$ where ρ_{max} is the norm of x and y dimension of the image. This is because 'for ρ, the maximum distance possible is the diagonal length of the image' [5].
2. **Accumulator Array**: Initialize the accumulator array H which will hold the votes for each parameter configuration of θ and ρ.
3. **Voting:** For each point (x_i, y_i) in the image I, calculate ρ for each value of θ using Equation(1) and increment the corresponding entry in the accumulator array H.
$$\rho = x_i cos\theta + y_i sin\theta \tag{1}$$
4. **Local Maxima**: Find the value(s) of $\theta, \rho)$ where $H(\theta, \rho)$ is a local maximum. The detected line in the image is given by
$$y = x_i \frac{-cos\theta'}{sin\theta'} + \frac{\rho'}{sin\theta'}$$

Algorithm 2 Hough Transform for extracting lines.

procedure MYHOUGHTRANSFORM
  $\theta \leftarrow$ -90:1:89
  $\rho_{max} \leftarrow$ norm of image dimensions
  $\rho \leftarrow -\rho_{max} : 1 : \rho_{max}$
  $H \leftarrow$ empty accumulator array the size of $[length(\theta), length(\rho)]$
  for *every point* (x_i, y_i) *in image* I **do**
    for *every point* (θ_i) *in theta* **do**
      $\rho = x_i cos\theta + y_i sin\theta$
      $H(\theta, \rho) = H(\theta, \rho) + 1$
  $[\theta', \rho'] \leftarrow$ **localMaxima(H)**
  return $[\theta', \rho']$

4.3 Results:



(a) $hough(image)$ (b) $myHoughTransform(image)$

Figure 12: Results for Hough Transform with $numberHoughPeaks = 5$. The peaks are marked in white squares.Fig. 10(a) shows results from a standard MATLAB function $hough(image)$ and Fig. 9(b) shows the results for the edge detection implemented from the scratch $myHoughTransform(image)$.

Figure 13: Plotting hough lines

5 List of functions implemented

Function Name	Input	Output	Definition
myFilter.m	1) Filter name = “gaussian” , “sobel” 2) Filter size (int) 3) Value of sigma (optional)	1) filter = Filter matrix	Gives sobel or gaussian filter
myConv.m	1) g = input image 2) F = filter matrix	1) h = result of convolving g and f	Convolution of two matrices without zero-padding. (‘valid’ convolution)
myHarrisDetector.m	1) Ix = derivative of input image in x-direction 2) Iy = derivative of input image in y-direction 3) thresh = threshold for corner response R	1) [rows ,cols] = coordinates of corner points in an image	Detecting corner points in an image using Harris corner detction
myImageDilation.m	1) I = Input image 2) filt = filter for image dilation	1) o = dilated image	Non-maxima suppression in Harris corner detection
RANSAC.m	1) x = x-coordinates of points 2) y = y-coordinates of points 3) d = minimum no. of points that need to be inliers for a particular model to be selected. 4) thresh = minimum distance of a point from the line for it to be considered as an inlier.	1) inliers = Points satisfying the final model 2) b = selected parameter configuration	RANSAC Algorithm for line-fitting
myLinearRegression.m	1) x = x-coordinates of points 2) y = y-coordinates of points	1) X = x-coordinates concatenated with a vector of 1s. 2) B = selected parameter configuration i.e. slope and intercept.	Linear Regression for line-fitting
myCannyEdge.m	1) img = input image 2) T_High = High threshold for Hysteresis Thresholding 3) T_Low = Low threshold for Hysteresis Thresholding	1) T_res = Binary image with edges marked	Canny Edge Detection
myHoughTransform.m	1) img = input image 2) resolution = resolution for discretizing theta and rho	1) H = Hough transform matrix 2) theta = discretized theta 3) rho = discretized rho	Hough Transform method for multiple model fitting
myHoughPeaks.m	1) img = input image 2) Numpeak = number of peaks to be detected	1) peaks = Hough Transform buckets with highest votes	Finding local maxima in Hough Transform
myHoughLines.m	1) img = input image 2) Theta = optimal theta 3) Rho = optimal rho 4) Peaks = Hough peaks	Plot with hough lines overlayed on the image	Plotting hough lines
removeClusters.m	1) x = x-coordinates of corner points 2) y = y-coordinates of corner points 3) thresh = corner points less than ‘thresh' number of pixels apart are removed	1) new_data = new set of corner points	Removes corner points around non-linear objects like cars and trees to avoid having false positive lines later on during RANSAC

References

[1] R. Raguram, J.-M. Frahm, and M. Pollefeys, “A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus,” in *European Conference on Computer Vision*, pp. 500–513, Springer, 2008.

[2] “MATLAB Simulink linear regression.” `https://www.mathworks.com/help/matlab/data_analysis/linear-regression.html`. Accessed: 2019-03-31.

[3] R. C. Gonzalez and P. Wintz, "Digital image processing(book)," *Reading, Mass., Addison-Wesley Publishing Co., Inc.(Applied Mathematics and Computation*, no. 13, p. 451, 1977.

[4] M. Shah, "Cap 5415 computer vision fall 2012," *University of Central Florids*, 2012.

[5] "Hough Line Transform opencv." `https://docs.opencv.org/3.1.0/d6/d10/tutorial_py_houghlines.html`. Accessed: 2019-03-31.

