Finding Lanes

Arunava Nag

March 10, 2018

Abstract

Self driving cars are the now a necessity in this world, which makes it absolutely important to have a sound algorithmic system that would perform detection of various features on road such as sign boards, lane markers etc. In this report lanes will be detected using test images and also for stream video.

1 Introduction

This project defines a method to detect Lane lines in a road in order to safely guide and maintain the car on a specific lane. The goal is to detect the white lines as in fig 1.

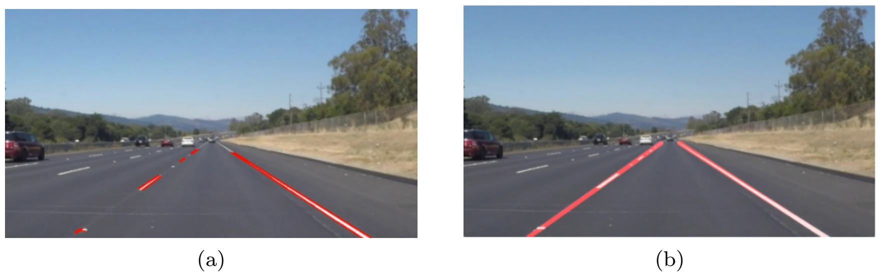


Figure 1: a) Detected Lines b) Detected Lines Joint

The project was completed following several steps. The pipeline will described in the sections below.

2 Pipeline Description

2.1 Pipeline Steps

1. Read in and grayscale the image.
2. Define a kernel size and apply Gaussian smoothing.
3. Define our parameters for Canny and apply to get edges image.
4. Mask edges image using cv2.fillPoly()
5. Define Hough transform parameters and run Hough transform on masked edge-detected image.
6. Draw line segments
7. Draw lines extrapolated from line segments
8. Combine line image with original image to see how accurate the line annotations are.

2.2 Drawing Lines from Line Segments through extrapolation

1. Divide the hough line segments into those with positive slopes (left lane line) and negative slopes (negative lane line). Take only those with length greater than 50 to remove noise.

   - For each Hough line segment (two points), if (1) the slope of the line is positive and (2) if the line segment length is greater than 50 pixels (length chosen by observation), add the slope and intercept to an array `positive_slope_intercept`. I also added the clause that if the array is empty, we should go through the Hough lines again and add at least one pair of points to the array `positive_slope_intercept`.
   - Do the same for Hough line segments with negative slope.
   - Line segments with positive slope belong to the negative

   itemize

2. Fit a line to the points belonging to the left lane line and to the points belonging to the right lane line separately using `find_line_fit`. Get the intercept and coefficient of the fitted lines.

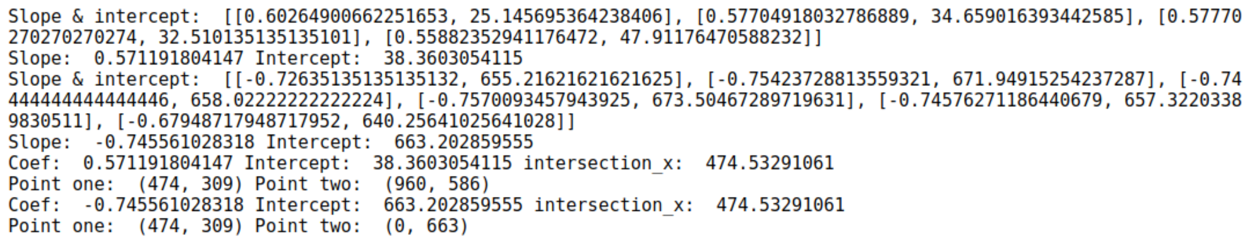
   - If there is only one slope and intercept, return those.
   - **Remove noise**: Remove points with slope not within 1.5 standard deviations of the mean. (1.5 chosen arbitrarily) This may fail if there is only one datapoint (it failed on one frame of the yellow lane line video), which is why the above clause is added.
   - Take the estimate of the slope and intercept to be that of the remaining values.

3. Calculate where the lane lines intersect.

4. Draw the lines from the intersection point to the bottom of the image.

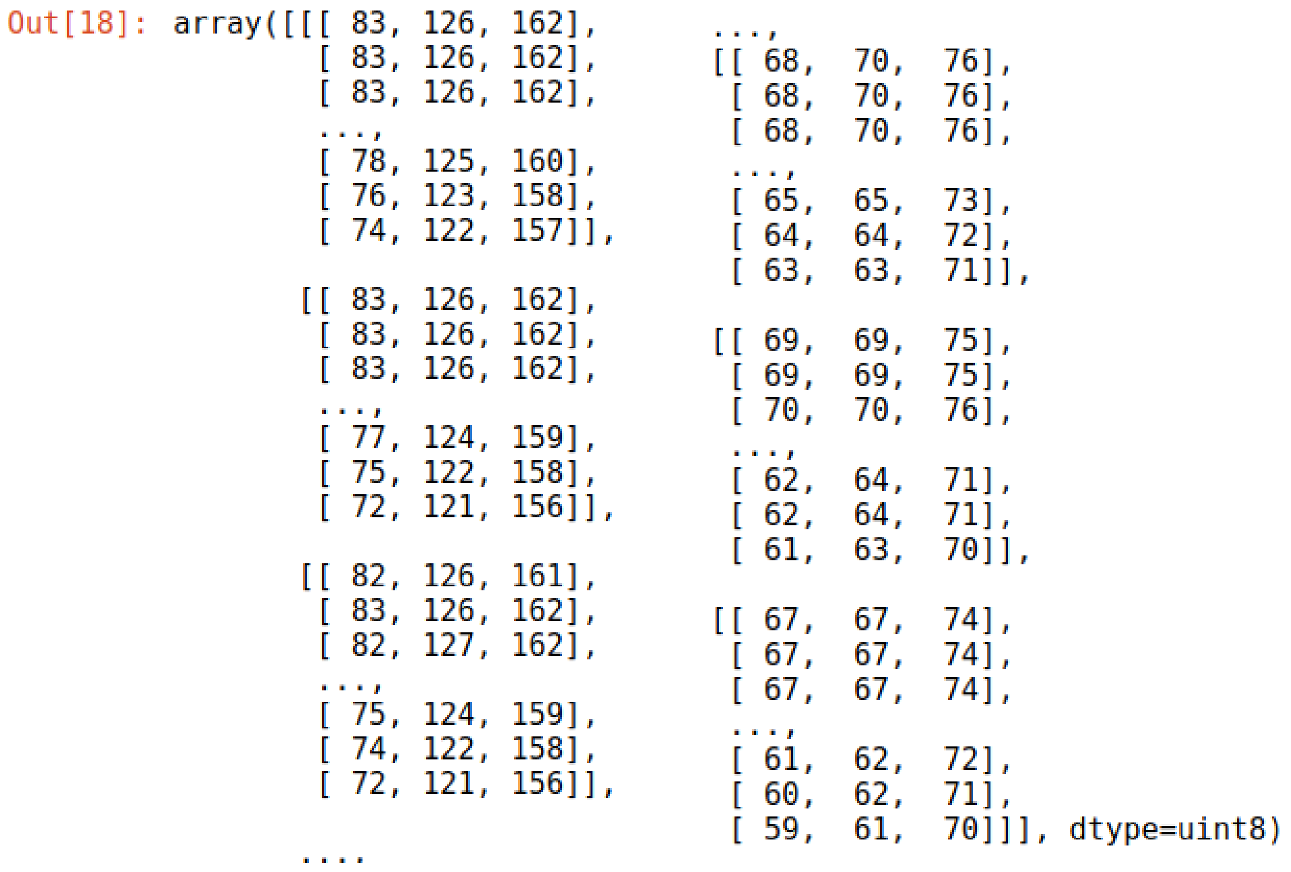
3 Result

Only one images result could be presented here. Rest has could be seen in the py script. We will take image **solidYellowCurve.jpg** to present the results.



```
Slope & intercept:  [[0.60264900662251653, 25.145695364238406], [0.57704918032786889, 34.659016393442585], [0.57770
270270270274, 32.510135135135101], [0.55882352941176472, 47.91176470588232]]
Slope:  0.571191804147 Intercept:  38.3603054115
Slope & intercept:  [[-0.72635135135135132, 655.21621621621625], [-0.75423728813559321, 671.94915254237287], [-0.74
444444444444446, 658.02222222222224], [-0.7570093457943925, 673.50467289719631], [-0.74576271186440679, 657.3220338
9830511], [-0.67948717948717952, 640.25641025641028]]
Slope:  -0.745561028318 Intercept:  663.202859555
Coef:  0.571191804147 Intercept:  38.3603054115 intersection_x:  474.53291061
Point one:  (474, 309) Point two:  (960, 586)
Coef:  -0.745561028318 Intercept:  663.202859555 intersection_x:  474.53291061
Point one:  (474, 309) Point two:  (0, 663)
```

Figure 2: Slope and Intercept output



```
Out[18]: array([[[ 83, 126, 162],
                 [ 83, 126, 162],
                 [ 83, 126, 162],
                 ...,
                 [ 78, 125, 160],
                 [ 76, 123, 158],
                 [ 74, 122, 157]],

                [[ 83, 126, 162],
                 [ 83, 126, 162],
                 [ 83, 126, 162],
                 ...,
                 [ 77, 124, 159],
                 [ 75, 122, 158],
                 [ 72, 121, 156]],

                [[ 82, 126, 161],
                 [ 83, 126, 162],
                 [ 82, 127, 162],
                 ...,
                 [ 75, 124, 159],
                 [ 74, 122, 158],
                 [ 72, 121, 156]],

                ....
```

(a)

```
 ...,
[[ 68,  70,  76],
 [ 68,  70,  76],
 [ 68,  70,  76],
 ...,
 [ 65,  65,  73],
 [ 64,  64,  72],
 [ 63,  63,  71]],

[[ 69,  69,  75],
 [ 69,  69,  75],
 [ 70,  70,  76],
 ...,
 [ 62,  64,  71],
 [ 62,  64,  71],
 [ 61,  63,  70]],

[[ 67,  67,  74],
 [ 67,  67,  74],
 [ 67,  67,  74],
 ...,
 [ 61,  62,  72],
 [ 60,  62,  71],
 [ 59,  61,  70]]], dtype=uint8)
```

(b)

Figure 3: Processed image matrix

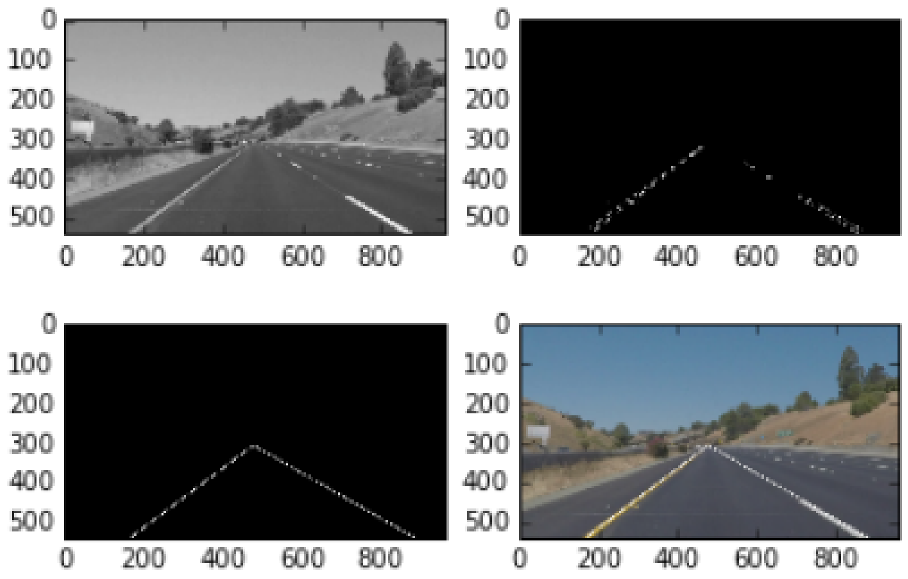


Figure 4: Result on Processed Image

4 Conclusion

4.1 Drawbacks

During the project, there were certain shortcomings, which could be listed as below.

1. There are a few instances in the yellow lane line video where the lane lines are marked incorrectly. The error is always in the lane line that is not solid.
    - This was improved by adjusting the Hough parameters further, however it resulted in no hough lines being created in at least one frame in yellow lane line video.
2. Image dimensions were pre-programmed in one case.
3. If the lane lines are not straight lines but have high curvature, this algorithm may give weird results.
4. The algorithm was trained on a specific type of lane and type of day (daytime, moderately bright with no snow, rain or hail), so it may not generalise well.

4.2 Future Work

The hough transform parameters need more tuning with more test images instead of trial and error. Also, currently the lane lines are drawn in solid white. It would be nice if they were drawn in thicker, semi-transparent red lines.