Finding Lane Lines on the Road

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file. But feel free to use some other method and submit a pdf if you prefer.

Finding Lane Lines on the Road

The goals / steps of this project are the following:

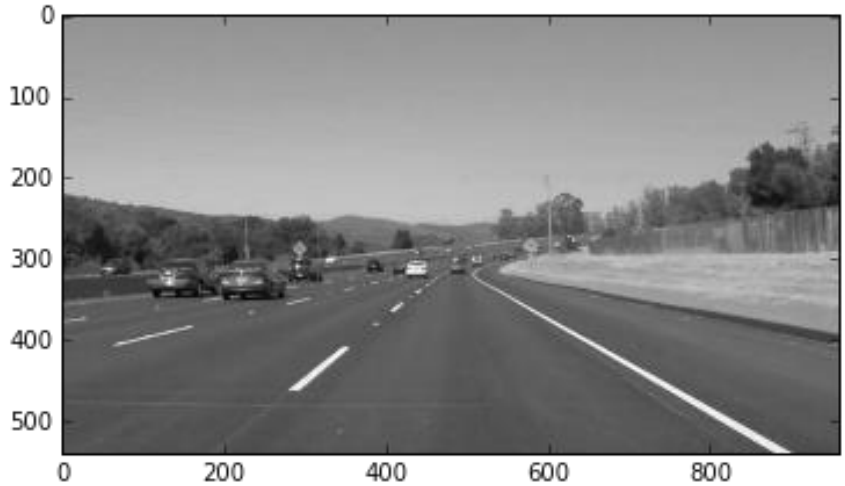
- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

Reflection

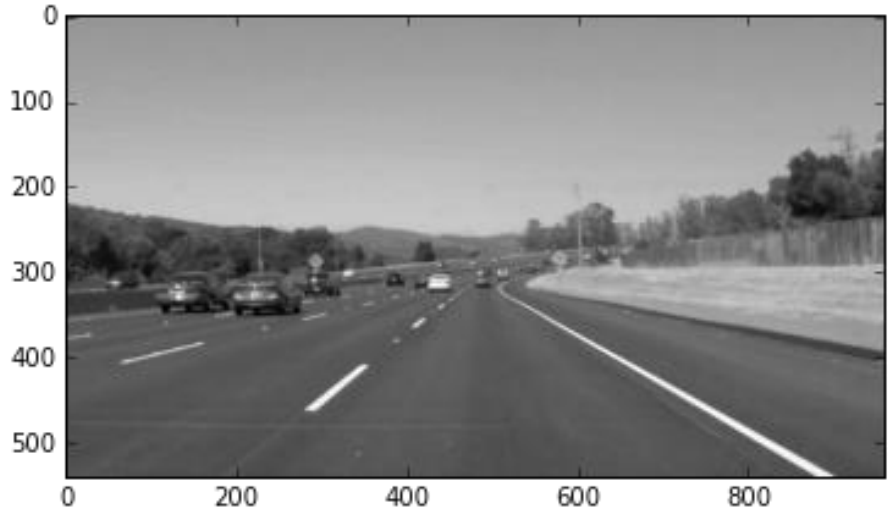
1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consisted of 5 steps.

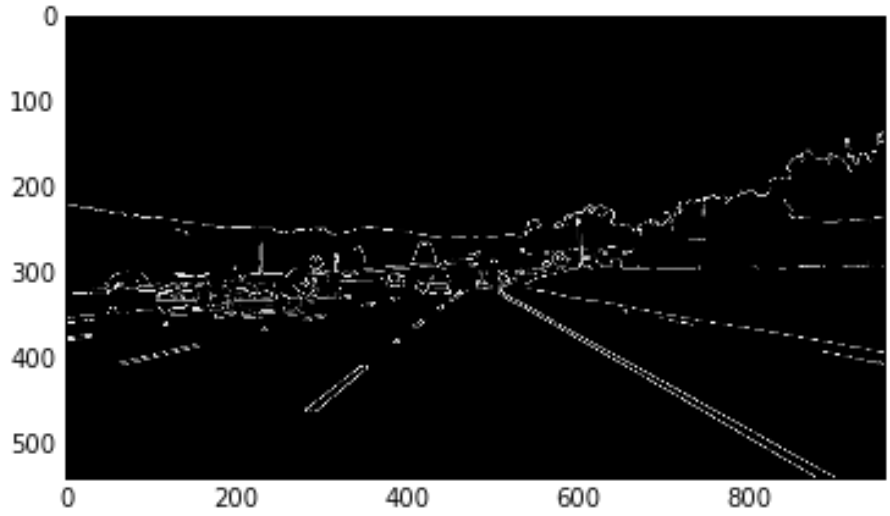
1. First, I converted the images to grayscale to reduce the number of channels



2. Second, I blurred the image using Gaussian blur to remove noise. The width of the mask is chosen to be 5 and the variance is calculated according.

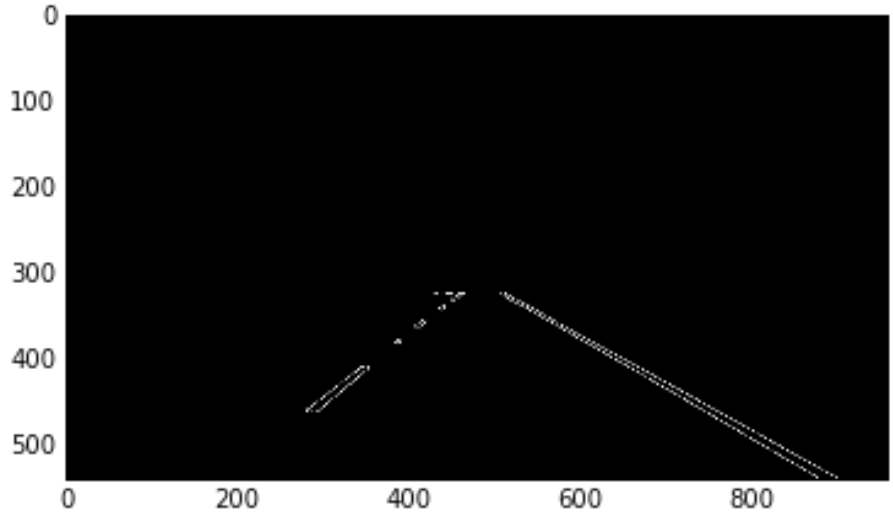


3. Third, I thinned the edges using Canny edge (threshold is determined using Otsu's method)
    a. Otsu's method first creates a histogram of the pixel intensities. Then it searches for a threshold that would minimize the variance between the 2 classes
    $$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$
    b. w represents the class probability (the summation of the probabilities of all pixels in the class) and sigma represent the class variance.
    c. Reference: http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

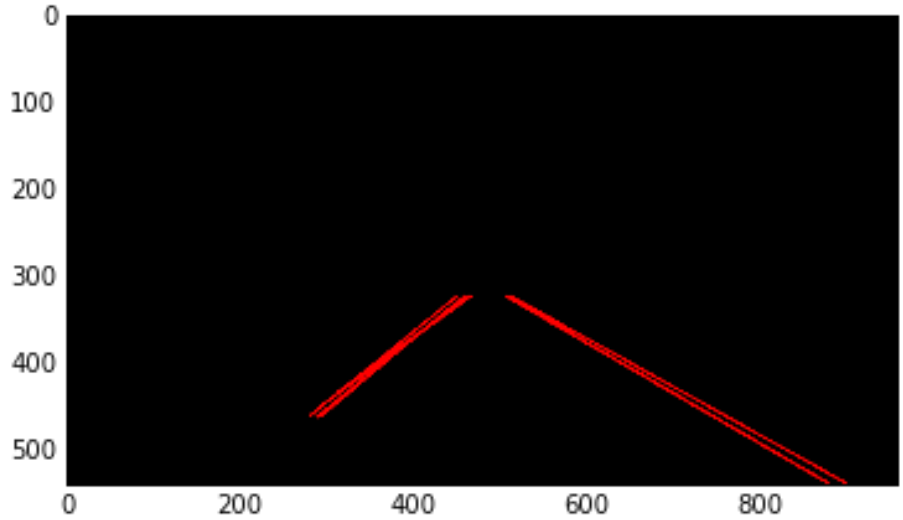


4. Fourth, I identified the Region of Interest (RoI) and later operations are performed only within the RoI to conserve computation power. After examining the test images, it seems like a centered trapezoid would identify RoI the best.
    a. The bottom of the trapezoid accounts for 90% of the width of the picture.
    b. The height of the trapezoid accounts for 40% of the height of the picture.
    c. The top of the trapezoid accounts for 10% of the width of the picture.

d. The trapezoid is centered with the center of the picture.
e. Note: the coordinate system's origin is on top left

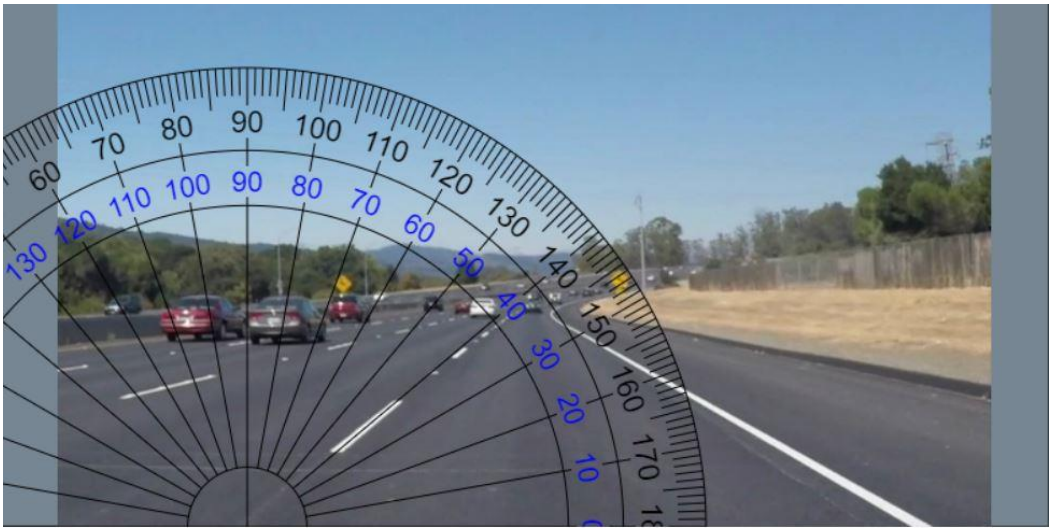


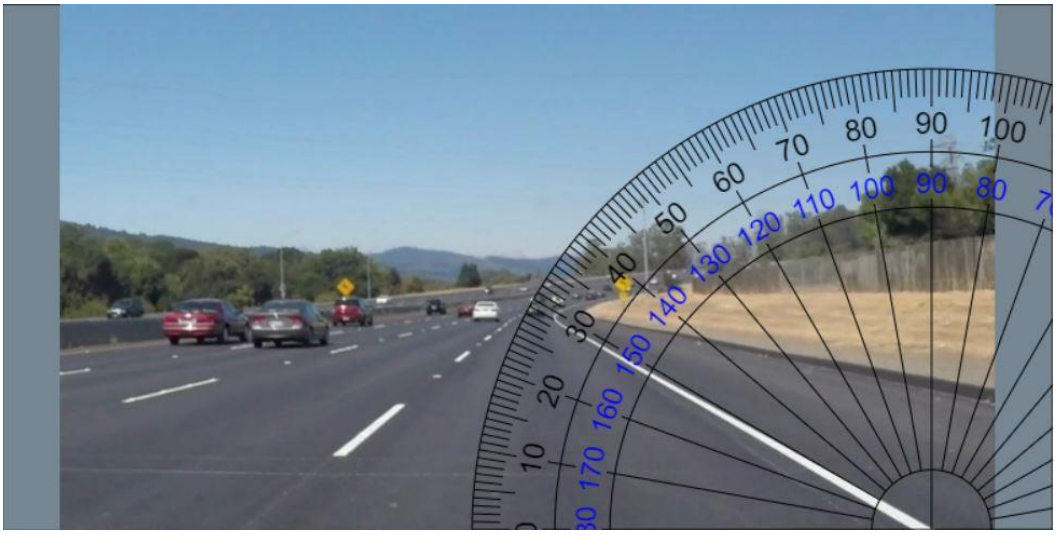
5. Fifth, I found the lines using Hough transform. After examining the example images and some trials:
    a. The resolution of r is set to 1 pixel
    b. The resolution of theta is set to 1 degree (pi/180)
    c. The threshold is set to 20 counts
    d. The minimum length to be considered as a line should be 80 pixels
    e. The max line gap should be set to 200 pixels. Because the max line gap is set so high, the dashed lane is appeared as a single lane, which is helpful in the following steps.



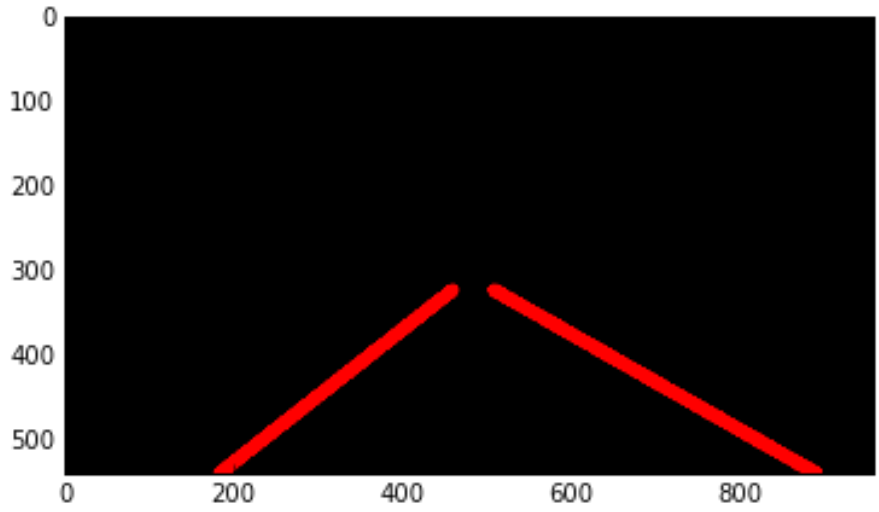
6. The last step is to take the many lines in the above image, divide them into two groups (left and right lane) and represent each group with a single line. When dividing many lines to 2 lines (left and right lane):
    a. Note: the coordinate system's origin is on top left
    b. If slope < 0, the line is part of left lane, otherwise right lane

c. Count the line iif 20 degree < abs(arctan(slope)) < 60 degree, discard the line otherwise (the line is not part of left or right lane). The boundary angles are selected from inspections on some example picture, like below.





d. Find the averages of left slopes and right slopes, find the averages of the left points and right points. (I initially used medians but realized the lines are very jumpy from image to image in the video, changing to means stabilizes the lanes)
e. The left lane is the mean of the left slopes that passes through the median of the left points. The same things for the right lane



7. Finally, overlay the marked lanes on top of the original image.

2. Identify potential shortcomings with your current pipeline

1. The current pipeline only identifies lanes that are straight; as a result, it does not perform well when the lane is curvy (like the optional challenge).

2. Another shortcoming could be a bright (white, yellow, orange…) car is driving in front of our car. Because of the sharp contract of the bright color with the pavement color, it could mess up the pipeline.

3. Most extensively, the RoI is hard coded. The current pipeline only takes one scenario (when the camera is in the middle of the lane). When the car is changing lanes, the RoI might only contain one lane, causing the operating system to be confused.

3. Suggest possible improvements to your pipeline

1. A possible improvement would be to filter out lines that clearly don't represent lanes. For example, if there is white car in front, our pipeline may recognize the edges of it as lanes. An approach has already been done by checking the angle of the line and set boundary values, for which a line is discarded if the angle is outside of the boundary values. This has already been implemented in the draw_lines() function.

2. Another potential improvement could be to remove the hard code for RoI. This may be done by scanning the whole picture for RoI, using a filter and cross-correlation. This would allow the RoI detection to be a lot more robust.

3. The low/high threshold of the Canny edge detection should be adjusted automatically based on the background, lighting condition, weather condition… This has already been tried and implemented by Otsu's method.