Finding Lane Lines on the Road

Report by Cristian Cucchiella

Objective

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

Pipeline

My pipeline consisted of the following steps:

- Convert Images to Grayscale
- Apply Gaussian Smoothing
- Apply Canny Transform
- Apply Region of interest
- Apply Hough Transform
- Separate left and right lanes
- Interpolate line gradients to create two smooth lines
- Merge Original Image with Lines

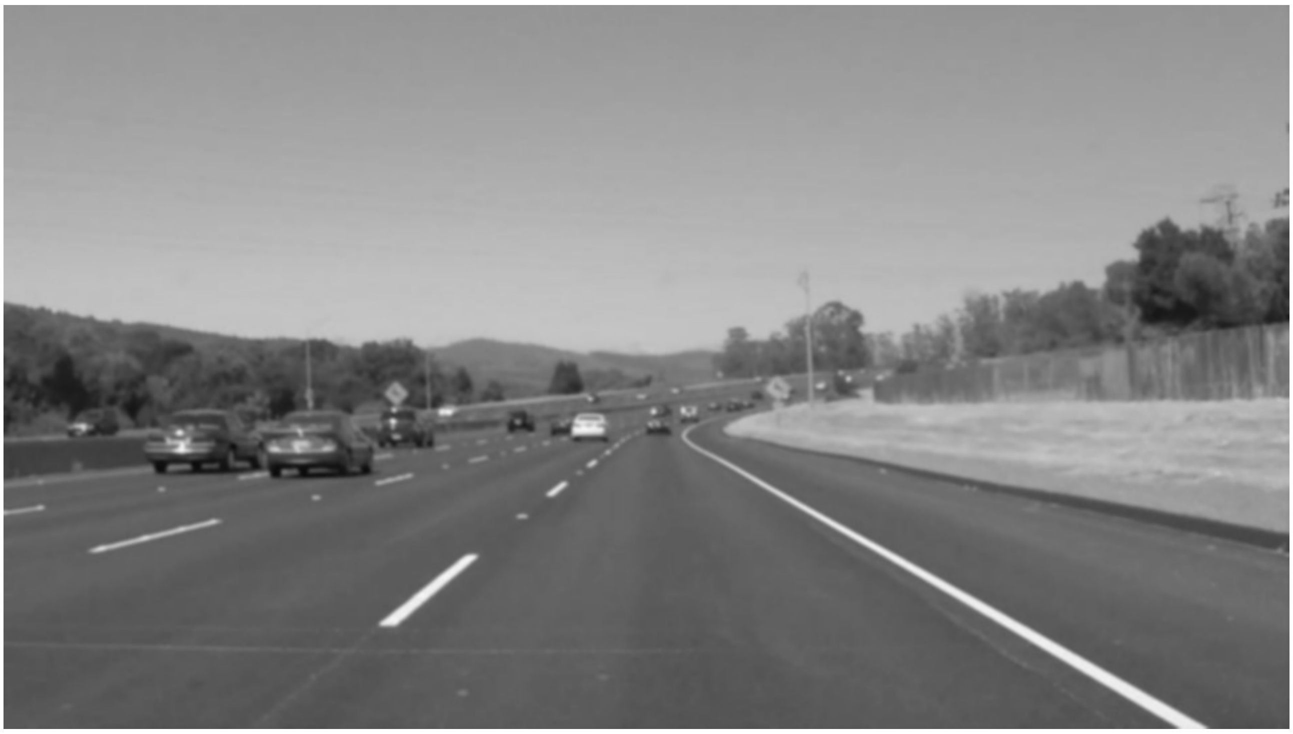
The input to each step is the output of the previous one. Now let's detail each step.

Convert Images to Grayscale

Our interest is in detecting white or yellow lines on images, which show a particularly high contrast when the image is in grayscale. The road is black, so anything that is much brighter on the road will come out with a high contrast in a grayscale image.
The conversion from RGB to a different space helps in reducing noise from the original three color channels. This is a necessary pre-processing steps before we can run more powerful algorithms to isolate lines.

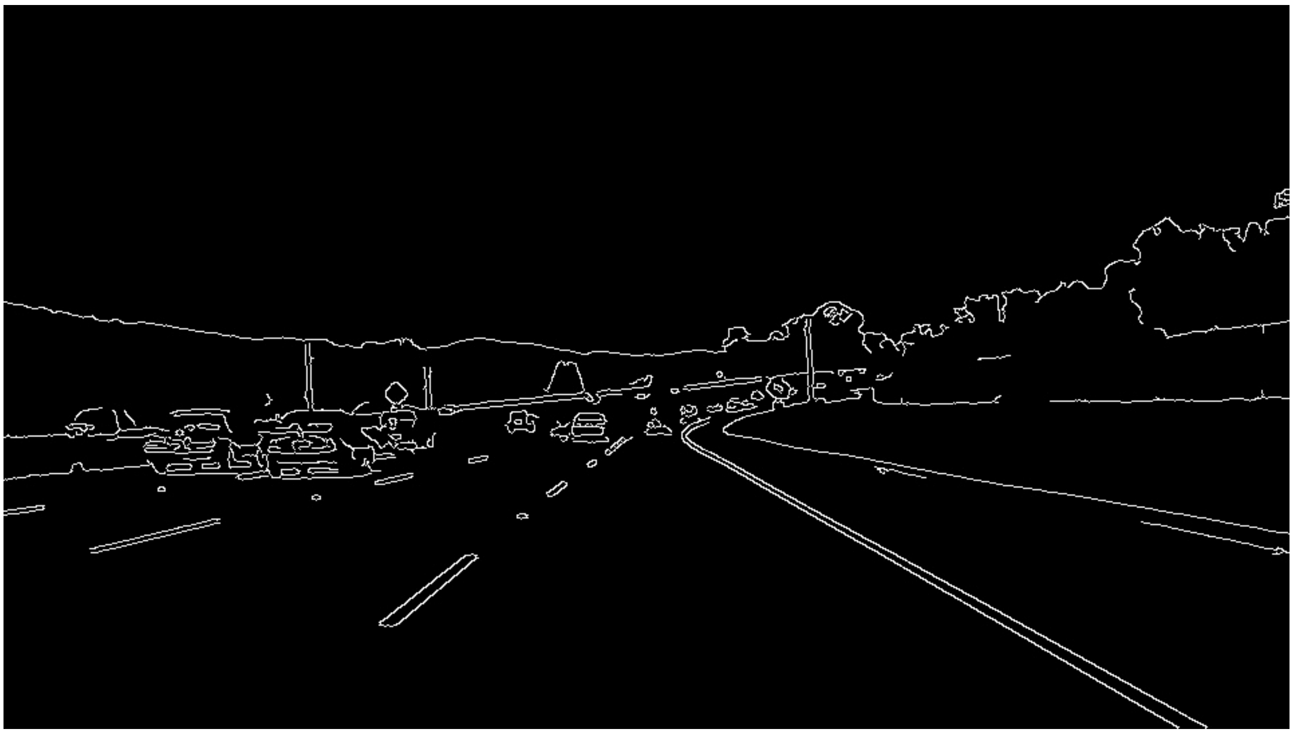
Apply Gaussian Smoothing

Gaussian smoothing (also referred to as Gaussian blur) is a pre-processing technique used to smoothen the edges of an image to reduce noise. We counter-intuitively take this step to reduce the number of lines we detect, as we only want to focus on the most significant lines (the lane ones), not those on every object. We must be careful as to not blur the images too much otherwise it will become hard to make up a line.
The OpenCV implementation of Gaussian Blur takes an integer kernel parameter which indicates the intensity of the smoothing.



Apply Canny Transform

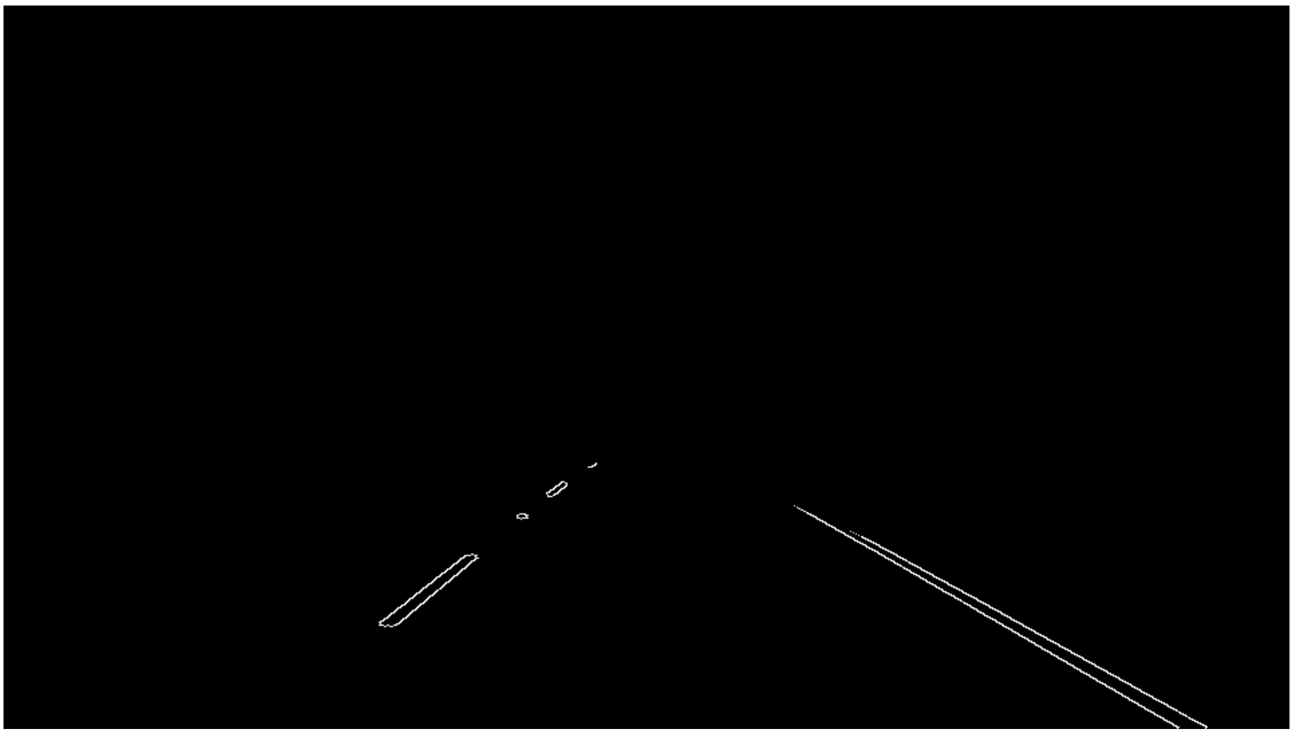
Canny Edge Detector identifies lines in an image and discard all other data.



The resulting image ends up being wiry, which enables us to focus on lane detection even more, since we are concerned with lines.
The OpenCV implementation requires passing in two parameters in addition to our blurred image, a low and high threshold which determine whether to include a given edge or not. A threshold captures the intensity of change of a given point (like a gradient). Any point beyond the high threshold will be included in our resulting image, while points between the threshold values will only be included if they are next to edges beyond our high threshold. Edges that are below our low threshold are discarded. Recommended low:high threshold ratios are 1:3 or 1:2.

Apply Region of interest

Next step is to determine a region of interest and discard any lines outside of this polygon. The crucial assumption in this task is that the camera remains in the sample place across all these image, and lanes are flat, therefore we can identify the critical region we are interested in. Looking at the above images, we "guess" what that region may be by following the contours of the lanes the car is in and define a polygon which will act as our region of interest below.

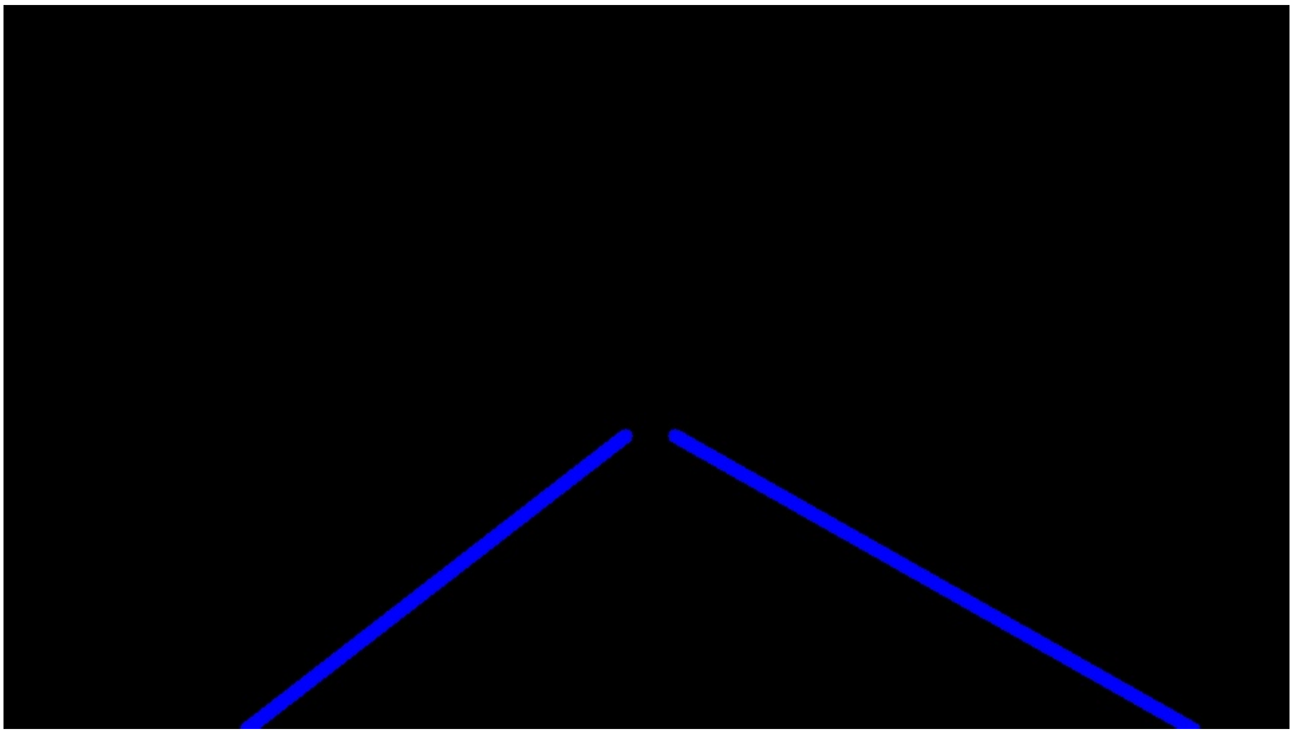


Apply Hough Transform

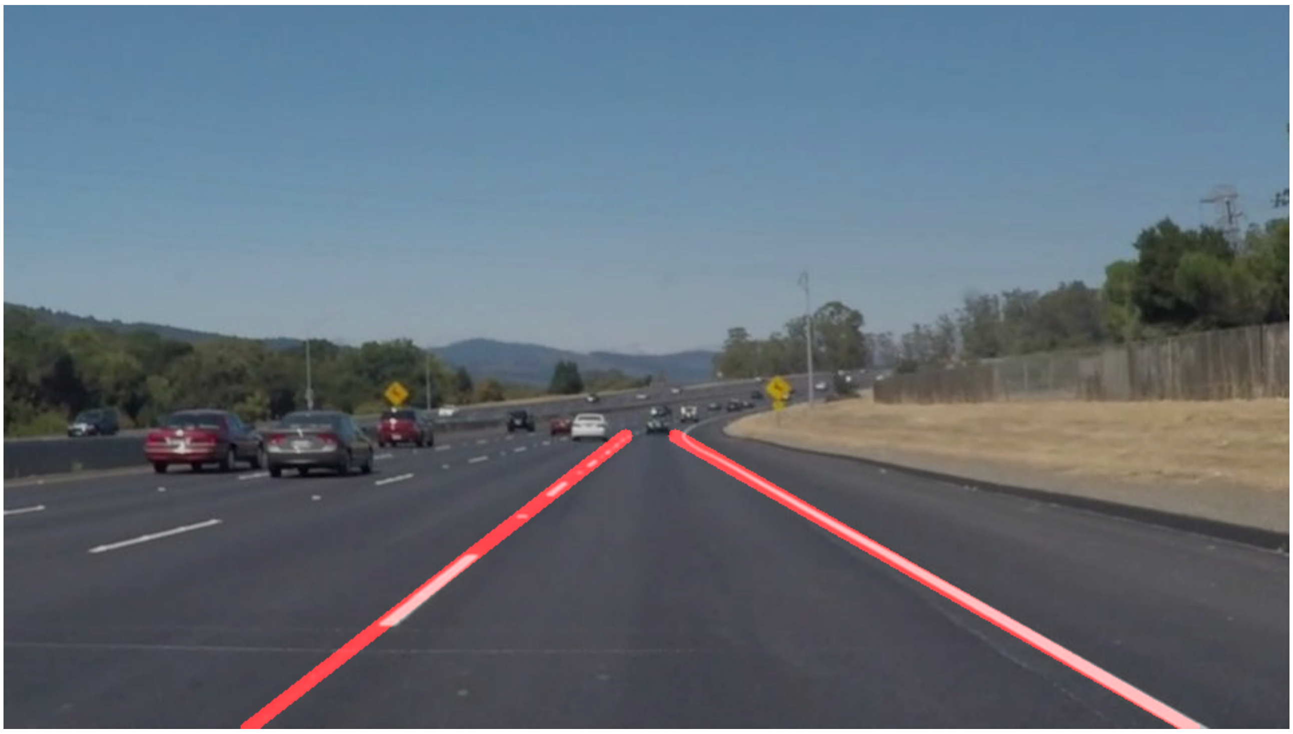
The next step is to apply the Hough Transform technique to extract lines and color them. The goal of Hough Transform is to find lines by identifying all points that lie on them. This is done by converting our current system denoted by axis (x,y) to a parametric one where axes are (m, b). In this plane:

- lines are represented as points
- points are presented as lines (since they can be on many lines in traditional coordinate system)
- intersecting lines means the same point is on multiple lines

Therefore, in such plane, we can more easily identify lines that go via the same point. We however need to move from the current system to a Hough Space which uses polar coordinates one as our original expression is not differentiable when m=0 (i.e. vertical lines).
In polar coordinates, a given line will now be expressed as (ρ, θ), where a line is reachable by going a distance ρ at angle θ from the origin, thus meeting the perpendicular to the line; that is $\rho = \underline{x \cos \theta + y \sin \theta}$. All straight lines going through a given point will correspond to a sinusoidal curve in the (ρ, θ) plane. Therefore, a set of points on the same straight line in Cartesian space will yield sinusoids that cross at the point (ρ, θ). This naturally means that the problem of detecting points on a line in cartesian space is reduced to finding intersecting sinusoids in Hough space.



Merge Original Image with Lines



At this point, the test required highlighting the lines with a continuous line (like in the video P1_example.mp4) and recognising both the lines on the right and the lines on the left.
To connect these segment lines, it was modified the helper function draw_line() to extrapolate the segments. For this it is possible to classify the segments to left or right by calculating their slopes using the formula m = (y2 - y1)/(x2 - x1). Then, it is drawn a line as the average of all segments. Finally the extrapolated line segments image is combined with the original image to show the lane lines.

Shortcomings

- Straight lines do not work well when there are curves on the road.
- Hough Transform params are very tricky to get right. I am not sure I got the best settings.

Future Improvements

- I have not explored the possibility of applying the HSL color filtering as another pre-processing step, converting colours from the RGB format to the HLS format. In RGB we cannot separate color information from luminance. Hue Saturation Value is used to separate image luminance from color information. It could be useful in presence of shadows.

- I plan to use deep learning to identify lanes and compare those results against what I obtained with a pure computer vision approach.

<u>You can find the previous images into the folder</u>:
Grayscale: ./test_images_output/gray.jpg
Gaussian smoothing: ./test_images_output/blur_gray.jpg
Canny Edge detection: ./test_images_output/canny.jpg
Region of Interest: ./test_images_output/roi.jpg
Hough Transform: ./test_images_output/hough_lines.jpg
Draw lines: ./test_images_output/solidWhiteCurve.jpg

Resources:

https://en.wikipedia.org/wiki/Hough_transform

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

https://medium.com/@c.ramprasad273/self-driving-car-nanodegree-udacity-project-1-finding-lane-lines-on-the-road-6b3c208aeada