# Finding Lane Lines on the Road

By Ricardo Picatoste

## 1 Introduction

In this document the steps followed to find lane lines on the road are explained.

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

## 2 Reflection

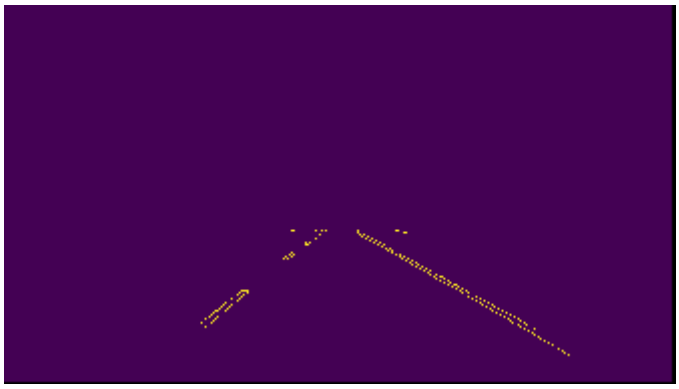
### 2.1 Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

I will explain the different steps with the application to an example image. These steps are also shown in the Jupyter noteboox, where the code can be executed.

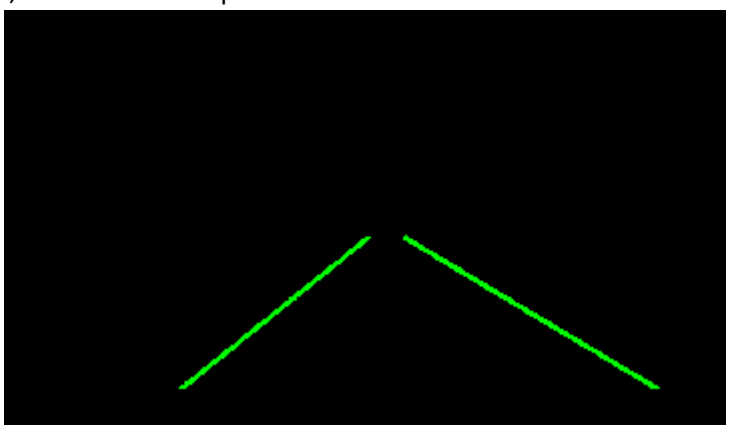
1. An image is loaded to start the example:

2. The image is converted to gray scale and blurred, following the steps given in the course.

3. Now the Canny transform is applied to obtain the gradient of the image. The result is cropped to have only the region of interest, that where the lane lines will mostly be. The region of interest has been limited to a trapezoid, with the base covering most of the bottom in the image, the top a bit below half the image, and much narrowed than the base.



4. Now the Hough transform is applied to the previous image. I played with the parameters until I found some that would be satisfying with all the images for testing and with the videos.
   The draw_lines function takes care of finding, among the lines found by the Hough transform, for each side (left and right), the one that is bigger, assuming that as the one coinciding with the actual lane.
   Then it will get the values of that line for the extremes of the region of interest before plotting them, in order to have plotted lines with a consistent size.



5. Finally, the obtained lines are superimposed on top of the original image:

## 2.2 Identify potential shortcomings with your current pipeline

I assume that this way of finding lanes will not be the one used for an actual self-driving car. It is very rigid in the sense that the parameters obtained for a few examples will hardly be used in every circumstance.

Also, given my limited ability with Python (I started learning it the same week that the course started), I was not able to do a running average of the lines over time, which would improve a lot the final result. But, since the process is done with:

yellow_clip = clip2.fl_image(process_image)

Where process_image is the function doing the job, I do not have an obvious way to return the lines from the previous iteration in order to apply the said temporal filtering.

Another shortcoming, which can be appreciated in the challenge result, is for the time when the line will not be straing. With a mild curvature, the Hough transform will find lines with more noise.

And finally, we are looking only in our region of interest to straight long lines. The situation will be different in crossovers, roundabouts, when entering the road from outside it, and a long etcetera.

## 2.3 Suggest possible improvements to your pipeline

With some more knowledge of Python (so maybe for the next project), the mentioned running average.

Another possible improvement would be to include curved lines in the search, but this is not very likely doable with the Hough transform. Plus, the more comparisons of this type are done, the harder will be to apply them in real time when they are finally applied on a real car.

## 2.4 Comments on the results

The videos look quite ok, with the line detected always on top of the real lanes for the 2 first videos. The challenge has a more noisy result, since the Hough transform will find several points looking like a line due to the curved lane.