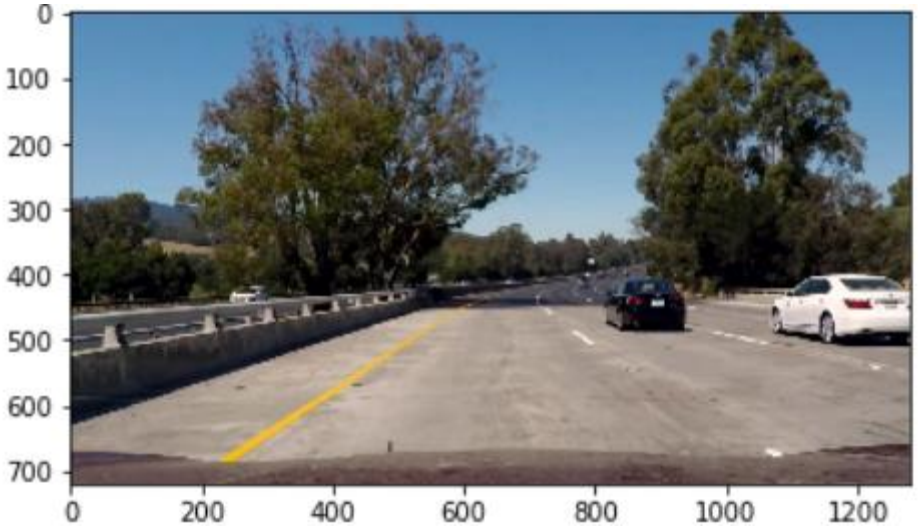
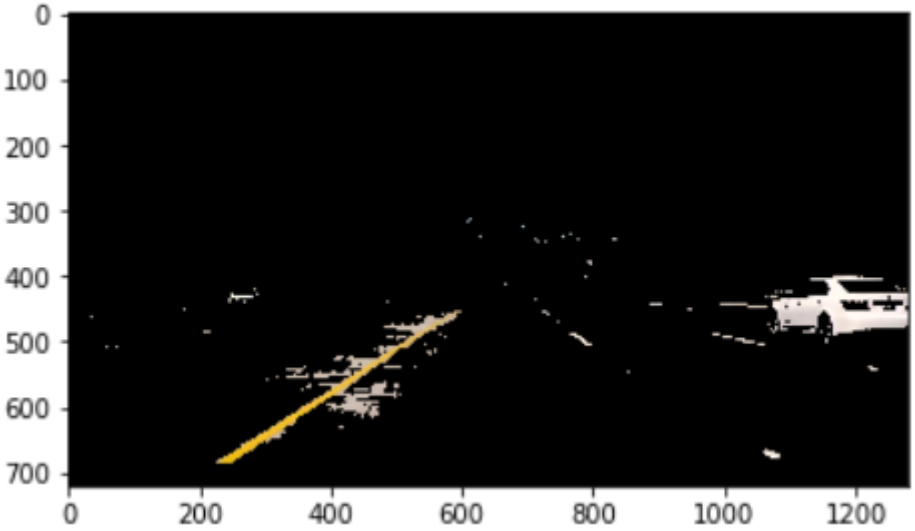
Finding Lane Lines on the Road

1. Pipeline description:

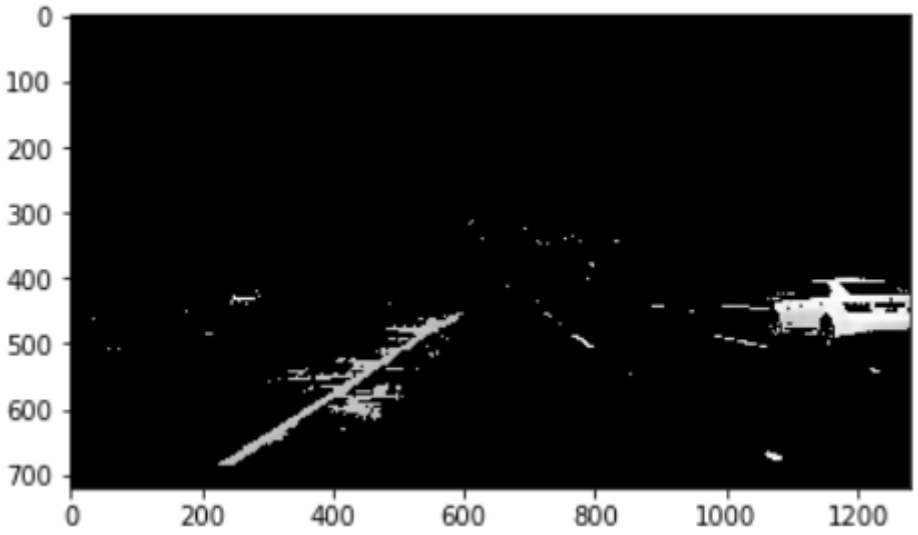
The pipeline consists of 12 steps. We start out with an initial image, such as the one seen below:



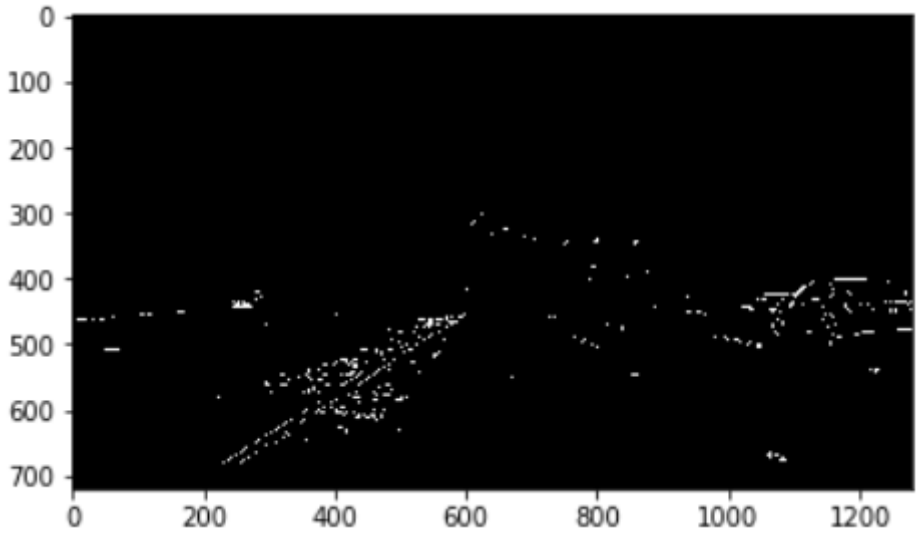
First, objects with only yellow and white colors in the image are selected. The rest are replaced with dark pixels. This has the advantage of highlighting only the required objects, i.e. lane lines which are either yellow or white. This already eliminates many unwanted edges for canny edge detection algorithm used later.



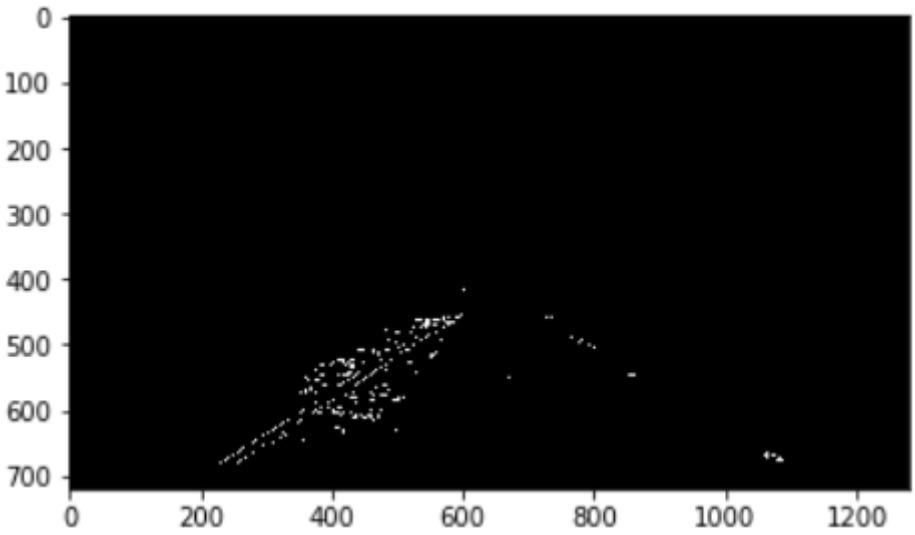
Then this image is converted to grayscale and then gaussian smoothing is applied.



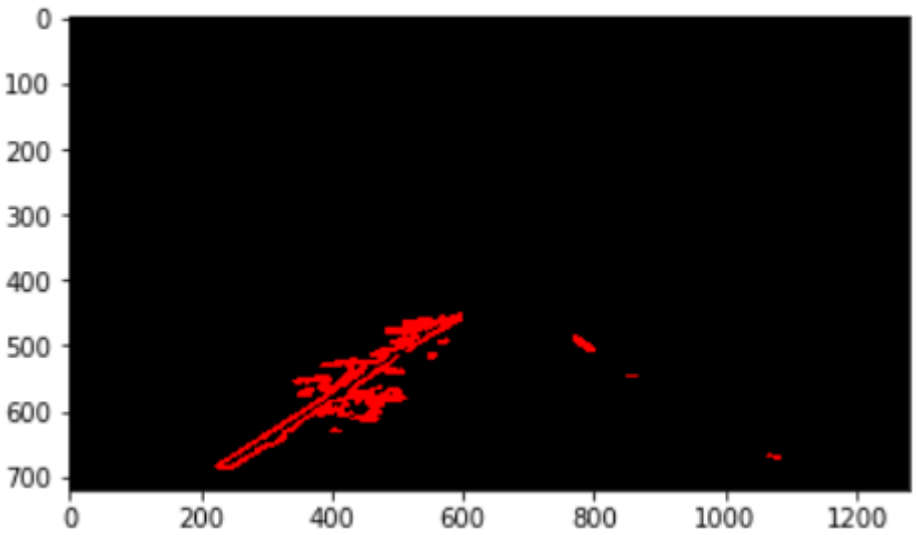
After that canny edge detection is applied. The result is seen below:



Then the resulting image is masked such that only the objects located within the desired area are left and others are replaced by dark pixels.



After that Hough transform is applied. The result is a list of detected lines corresponding to both lanes and some other irrelevant objects.



Now the goal is to filter out erratic lines, if any, and classify the remaining lines into two sets corresponding to left and right lanes. For this purpose, first slopes of all lines are calculated and inserted as a fifth element for each line in the list.

Then these lines are separated into two lists depending on whether their slopes are positive or negative. At the same time, lines with slopes close to zero (depending on a chosen threshold) are discarded. This step is specially very helpful for the third video since there a lot of horizontal lines are detected by the canny algorithm due to car's hood being visible in the images.

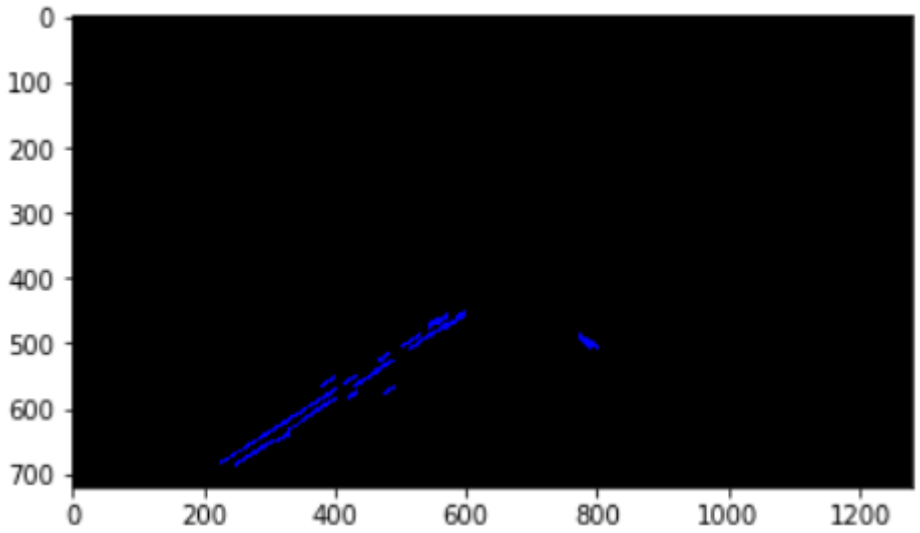
After that yet another step for further elimination of lines with deviant slopes was applied. Here I had to decide upon a criterion according to which I could chose which lines to retain for computation of final two lines.

After a few iterations, I decided that the slope belonging to the longest line in each of the two groups will be considered as the optimum slope and if the slope of a line deviated from it for more than a chosen error tolerance, it will be discarded.

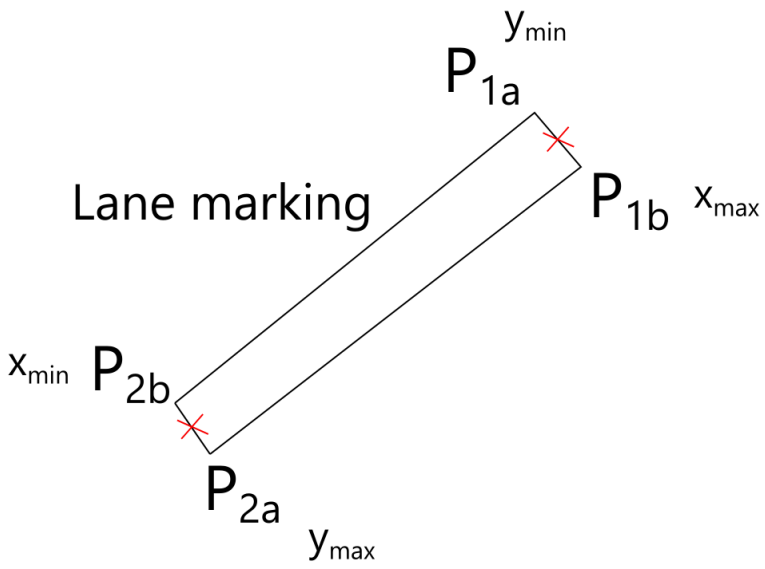
Of course this criterion is not perfect, but it worked mostly well after careful tuning of parameters for initial color selection and canny detection steps. In spite of that, we still see some glitches in some frames in the challenge video, which are likely caused by this. I feel that further tuning of parameters might have decreased these number of frames.

Another criterion that I tried for this step was to find out the lines with median slopes in both lists, and then remove the outliers. But this fails if there are many unwanted small lines detected, which results in elimination of lines actually corresponding to the lanes. After some experimentation, the former criterion was chosen.

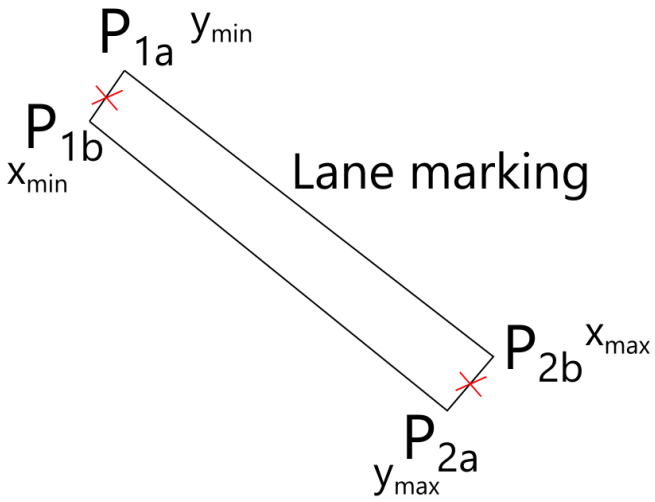
The picture below shows the remaining lists of lines after all the erratic lines have been removed by these two steps.



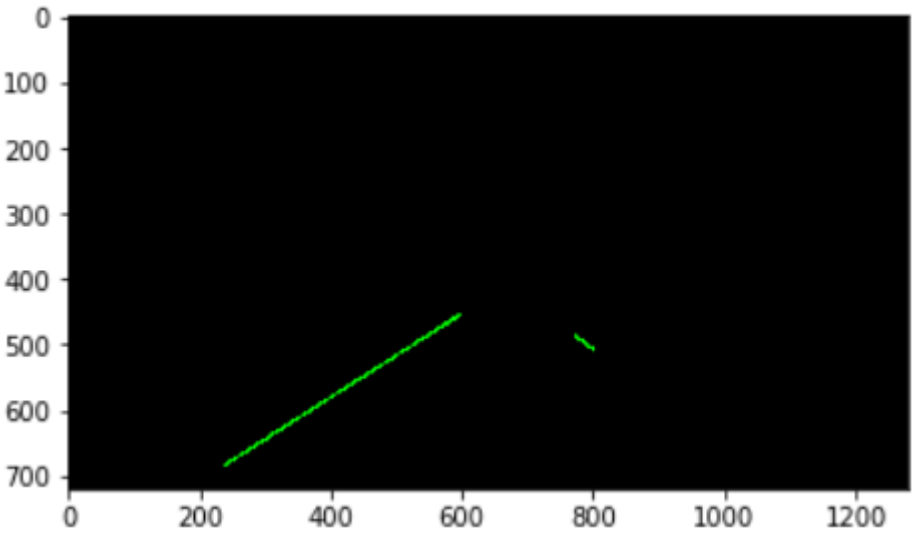
After this step, we now have two final lists of lines which can be averaged to find a line for each lane. For the lines corresponding to the left lane, I search in the list for the point with smallest y and the point with largest y. Similarly, I search for the point with maximum value of x and the point with minimum value of x. The idea is roughly illustrated in the figure below. Then the two points on each side of the lane marking are averaged to find the mid-points (the red crosses).



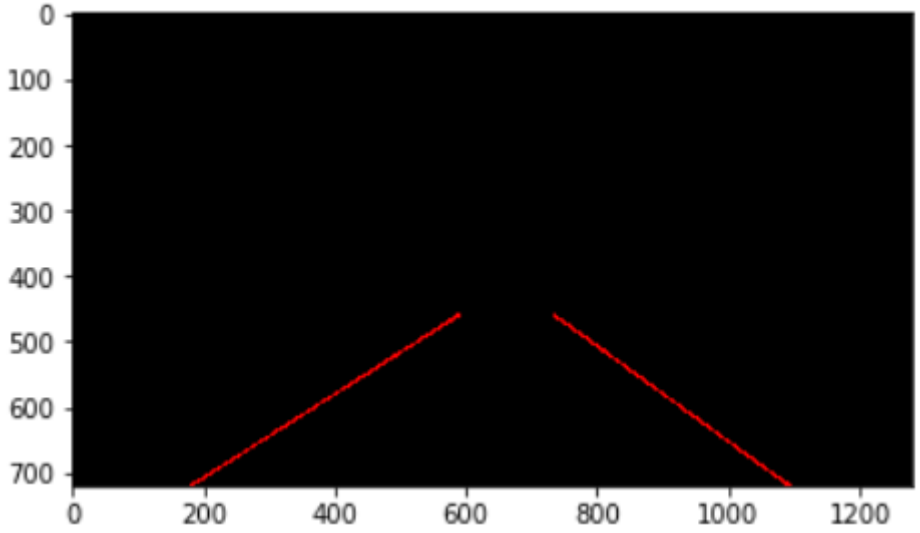
The calculation of average line for right side of lane is illustrated in the figure below.



Now we have two single lines each corresponding to a lane. They are combined again in a single list which can be drawn on an image for observation:



As a second last step, the two lines are extrapolated.



In the last step, the lines are displayed on top of the initial image.

2. Potential shortcomings with current pipeline

One shortcoming is already described in the previous section, i.e. the criterion chosen for elimination of lines with bad slopes. Currently I assume that the longest detected line has the correct slope and those differing too much from it are removed. However sometimes when there is too much clutter in the image or presence of another unwanted object, this assumption might fail. This results in completely wrong lane line drawing.

Another shortcoming would be the assumption for averaging of the lines as described in the two diagrams before. The points are not always arranged as assumed in the diagrams, for example it is possible that point P1a in the first diagram might have both

y_{min} and $x_{max.}$ However, in that case the average point will only slightly be shifted from the true average and the slope of the final extrapolated lines looks a bit off.

3. Possible improvements

Currently I am using fixed values for upper y coordinate for extrapolation of lines. The value works for the first two videos but is too large for the challenge video. One improvement could be to automatically compute the suitable value for each image by finding out the intersection of lane lines and then selection a value at some percentage below that point.

Other improvement would be to find a better way to detect lines not belonging to lanes.