Advanced Lane Finding

Udacity CarND

Writeup - Daniel Alejandro Reyna Torres

In this project, goal is to write a software pipeline to identify the lane boundaries in a video.

The Project

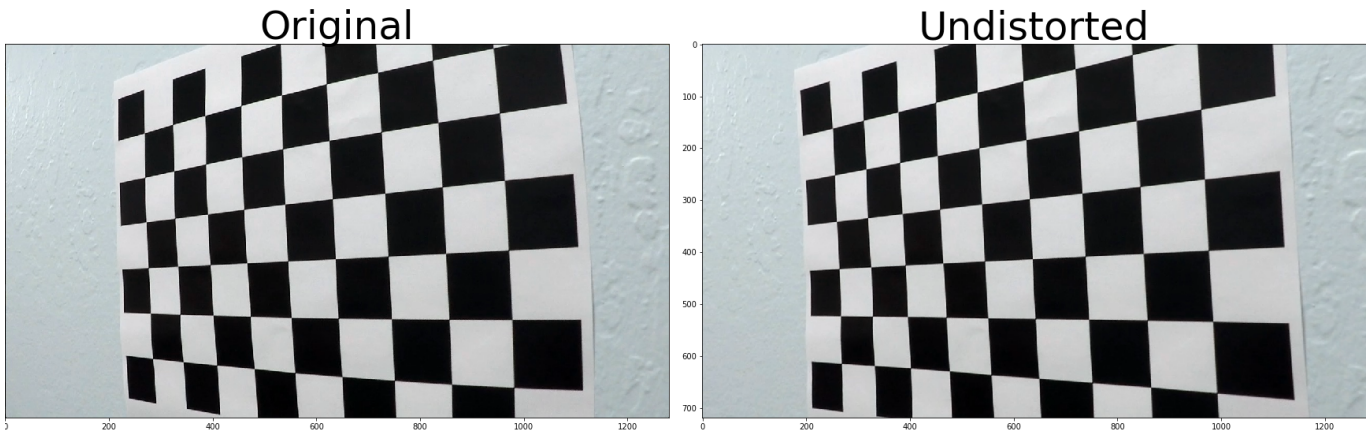
The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration

The very fist step in this project is camera calibration. We have to be sure that we have under certain control our image conditions. In order to deal with distortion caused by cameras we use OpenCV functions to perform the

camera calibration. During the calibration process we calculate the distortion matrix and distortion coefficients. These values are used in every image for ***distortion correction***. The following image is one of the images used to perform camera calibration, it can be seen both the original image and one ot the same image but undistorted (edges look straight):



Once we have performed the camera calibration, let's deep dive in the pipeline for lane detection.

Pipeline

Distortion Correction

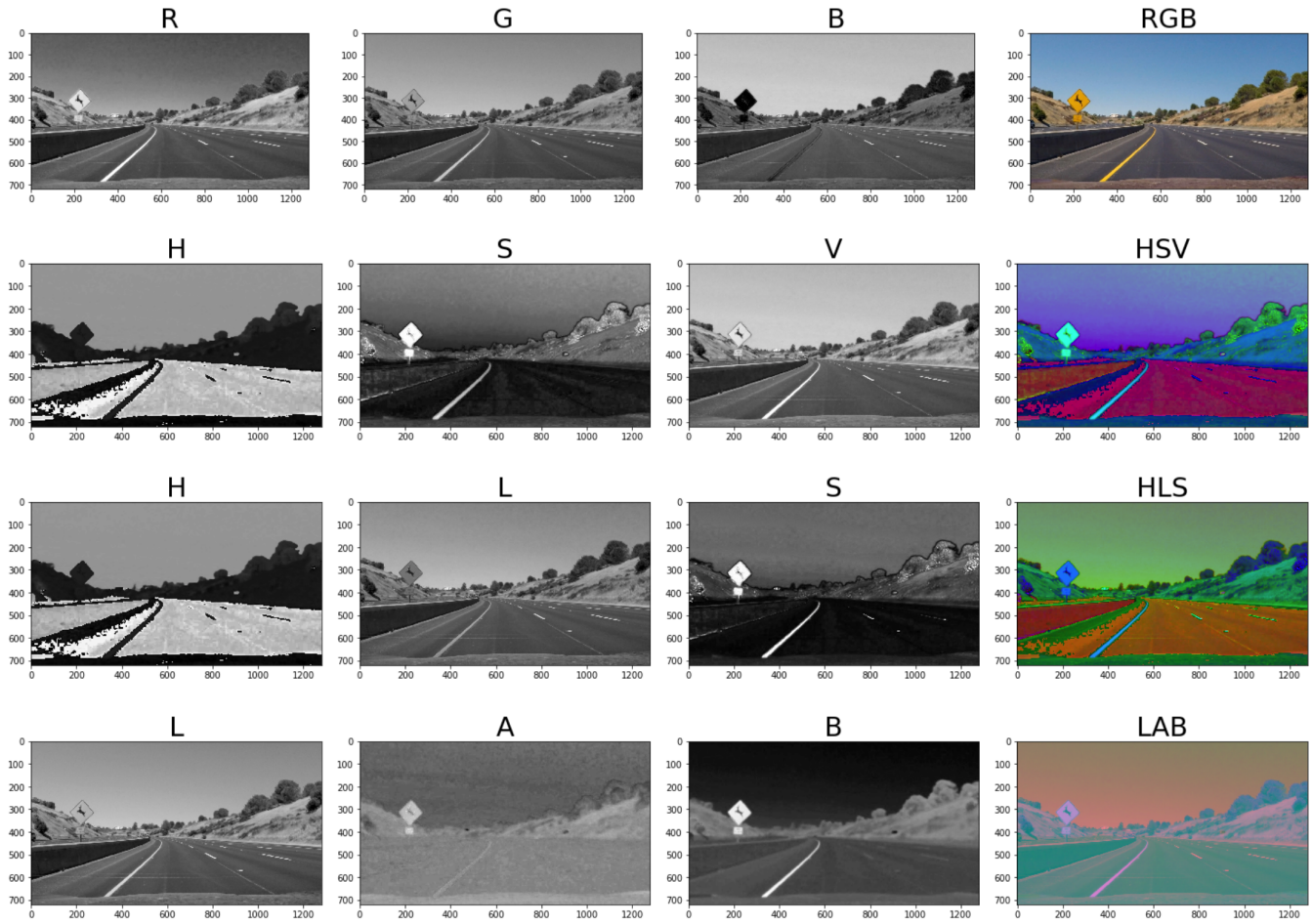
We apply the distortion correction to one of the the test_images:

This undistorted image is used as reference for lane detection within the pipeline.

Threshold Image - Filtering

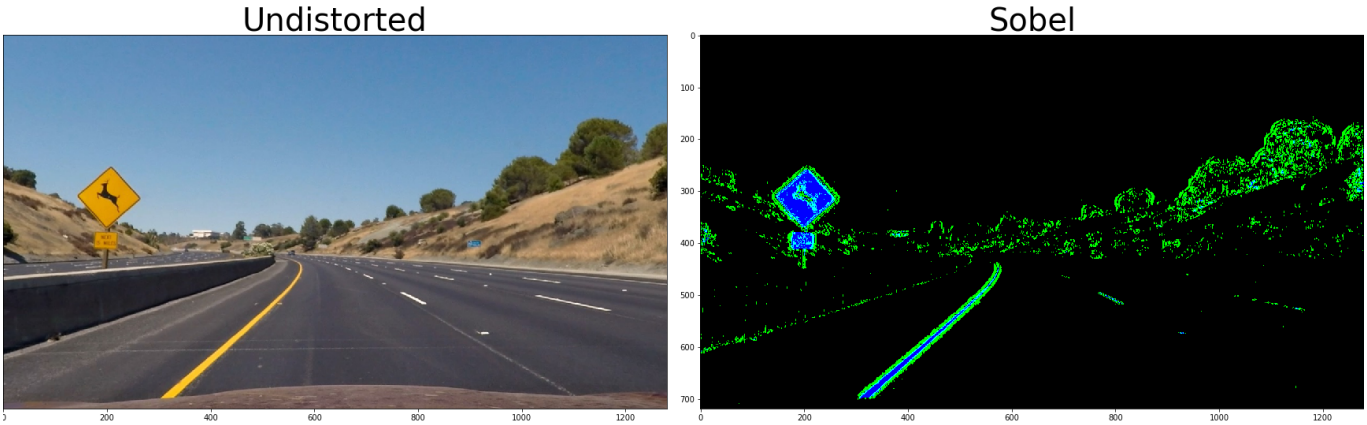
We apply color and gradient thresholding, i.e., filtering, in order to better detect lane lines that will be used for fitting the lane. This is a key step in the pipeline. There are several colorspaces and for this project we explore RGB, HSV, HLS, LAB and the Sobel operator to detect edges (or gradients).

Results of these colorspaces are shown here:



On the RGB components, we see that the red channel captures both yellow and white lines. For HSV and HLS, the S channel (saturation) gives the best result on detecting the lane lines. Finally, the LAB colorspace through the B channel seems to detect well only the yellow lines.

The Sobel operator is useful to identify changes in color intensities that allow us to filter line components going in either the vertical or horizontal direction. For this project was used the HLS colorspace for lane lines detection. Results of sobel operator using the S and L channels:



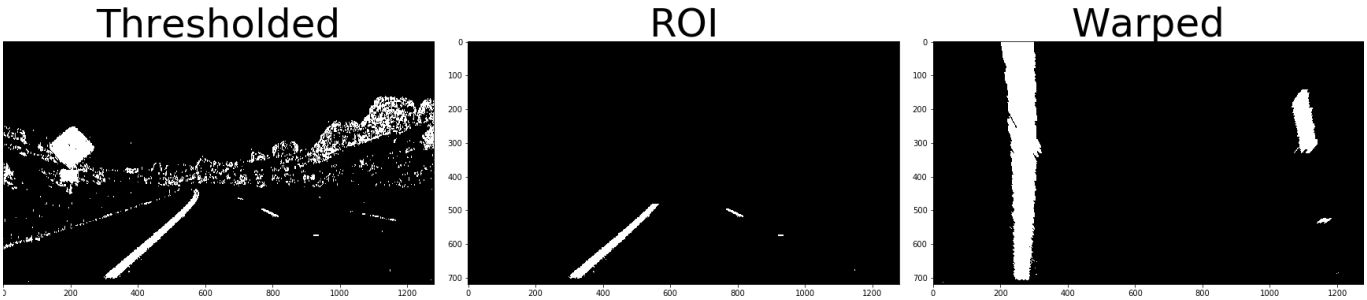
The goal in this section is to find right thresholds for detecting both yellow and white lines of the lane, we should consider that images were taken under different lighting conditions, shades, etc. We have to deal with this in order to perform a stable lane lines detection.

Finally, the thresholded image that we came up with at the end of this process is the following:

Perspective Transformation - "Birds-eye-view"

Now that we have our thresholded image we have to select a region of interest (ROI) that will go through a perspective transformation that will turn the image view into a birds-eye-view! This helps when fitting lines for the lane lines.

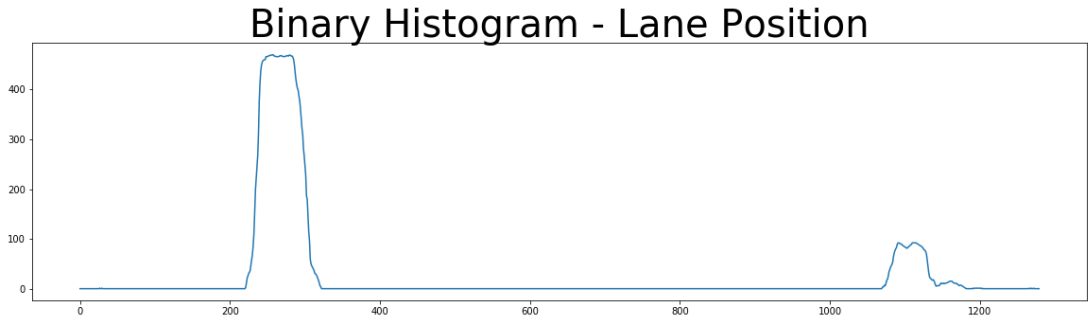
Since the camera position throughout all the images is constant, we manually defined the 4 ROI points that makes possible the perspective transformation. Lines should appear parallel in the warped image, which corresponds to a top view of the image. The perspective transformation produces the following results:



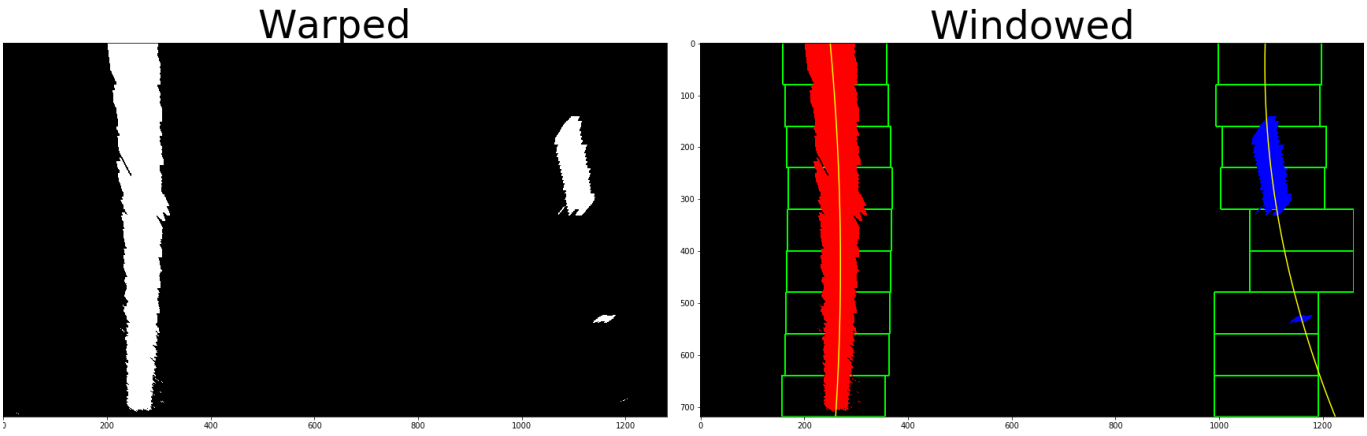
Lane Fit

Now, let's try to fit the lane.

We compute a ***histogram*** of the warped image in the y direction. We identify the positions where pixel intensities are highest. The peaks on the histogram represents an approximation of the lane lines position!



Next is to run a ***sliding window*** search over the warped image, using the peaks of the histogram as starting points, in order to estimate both left and right lane lines. The sliding window runs from bottom to top searching for pixels that belongs to a line. If find a certain numer of pixels, the window is re-centered, this will ensure that lane lines are well estimated. At the end, we fit a 2nd order polynomial to each group so we end up with an estimation of both lane lines.



Given the polynomial fit for the left and right lane lines obtained from the sliding window process, was calculated the radius of curvature for each line according to the formula:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

where 'y' are values going from top to bottom of the image use for lane line estimation and 'A' and 'B' values are the coefficients from the 2nd order polynomial fitted por each lane line.

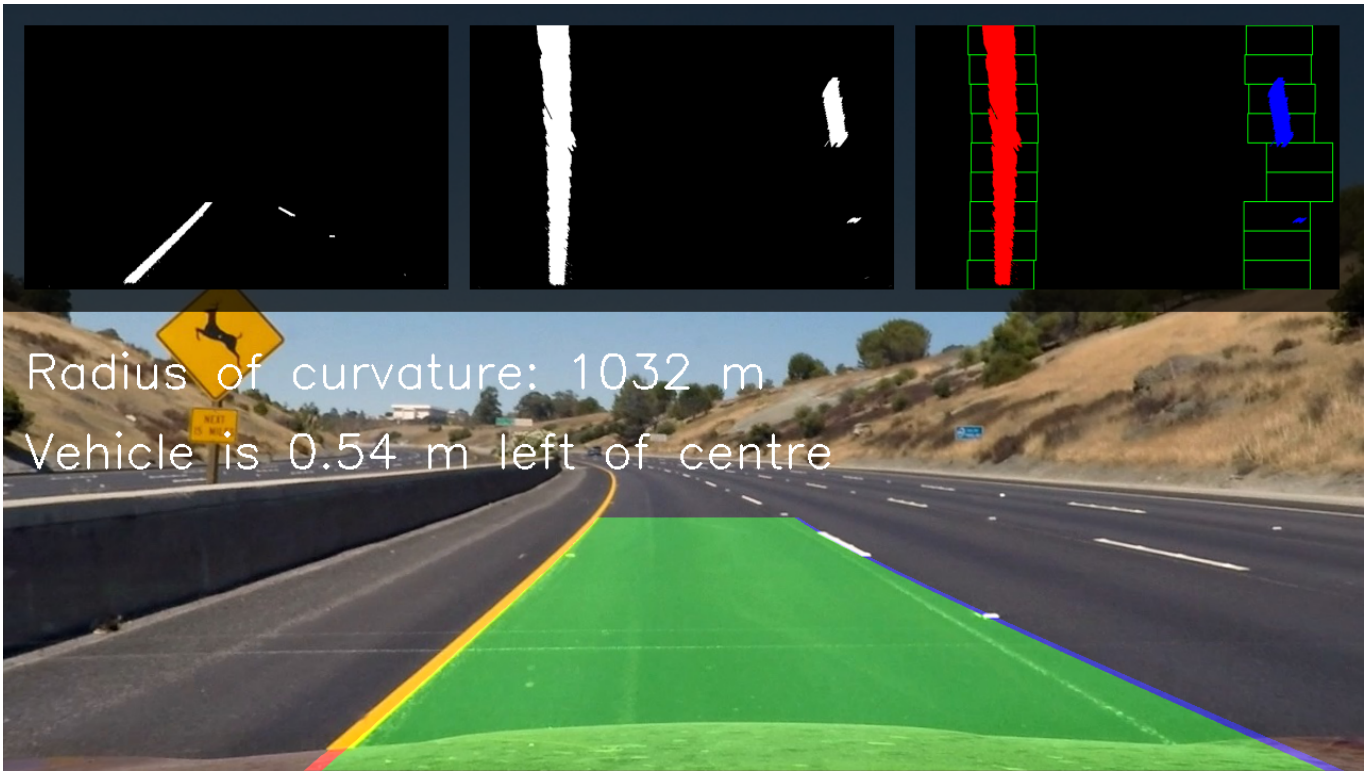
Let's compute the car position with respect to the centre of the lane. It is assummed that the lane centre is the centre of the image. Car's position would be the difference between the lane centre and the mean value of x positions at the bottom for both left and right lane lines.

The results are given in pixels, so they are converted from pixels to meters for more realistic interpretation. It was considered a resolution of 30m/720pixels in y dimension and 3.7m/700pixels in x dimension.

Results are display in the final results shown below.

Final Result

At the end of the process we end up with this:



Here's a link to my video result. The folder `output_images` contains examples of the output from each stage of the advanced lane detection pipeline, including videos.

Discussion

This project has been challenging and of course very interesting. Some exploration and update can be performed in the thresholdins process, thinking on combining different filters. For this project I selected the HLS to work with, but a combination might yield in better results during lane estimation.

Also, the warping stage could be dynamic instead of fixed tuned, we have to somehow ensure straight lines! Moreover, under some scenarios (on challenging videos) the current version lane detection is not 100% accurate, there are bigger shadows, cars, irregular street conditions, etc. to be highly considered. These conditions can be tackled by storing information related to previous video frames, thus in cases where lanes jump drastically or get lost, they can be validated by using past references, mean values, etc., making the detection cleaner.

Thank you for reading this report.

Daniel