Advanced Lane finding

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.

1. Describe image pre-processing and pipeline:

- The first step is to compute the camera calibration matrix and distortion coefficients given a set of chessboard images, and apply a distortion correction to raw images. The example below shows an image before and after distortion correction:

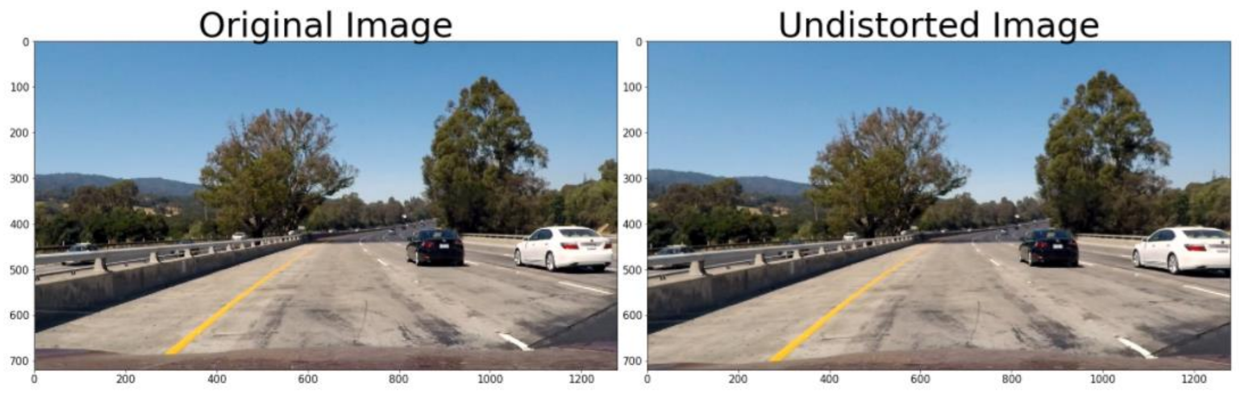

Figure 1. Left: Original image, right: after distortion correction.

- Warp the undistorted image based on the source and destination for getPerspectiveTransform function as:

Source	Destination
[565,457]	[0,0]
[729,457]	[1280,0]
[1380,720]	[1280,720]
[75,720]	[0,720]

Figure below shows an example of warped image:

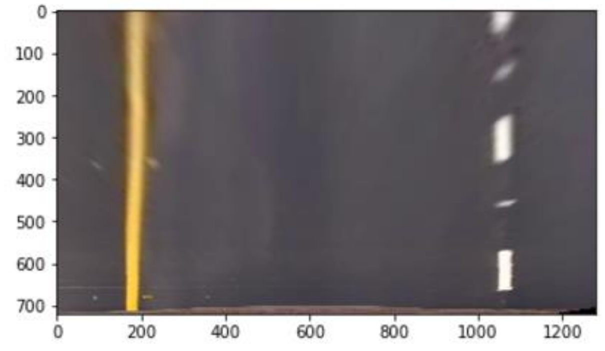

Figure 2. Warped image after distortion correction.

- Make a pipeline either by combining the thresholds or HLS color space and thresholds, to change the image RGB to binary and select specific channels or thresholds. It turned out combining the HLS color space and thresholds can give better result in the video processing.

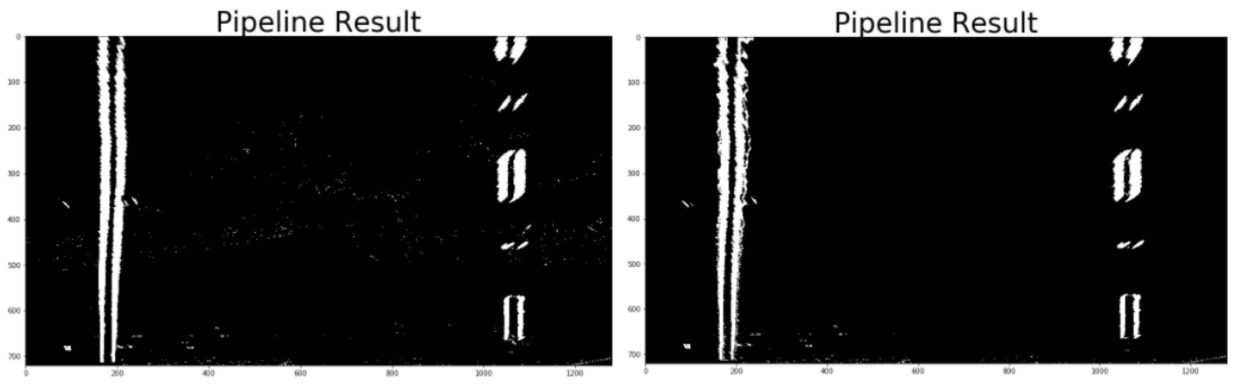

Figure 3. Left: result after combining the absolute_Sobel, magnitude and direction thresholds, right: result after combining S channel threshold with Sobel threshold of L channel.

- Apply sliding windows to find the lines:

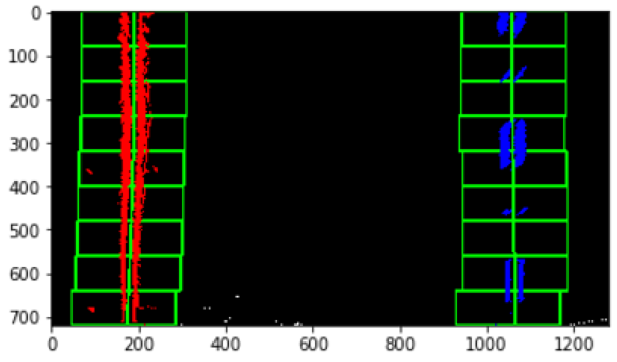

Figure 4. Found left and right line using sliding windows algorithm.

- Draw the finding lines on the original undistorted RGB image by adding polygon to the image:

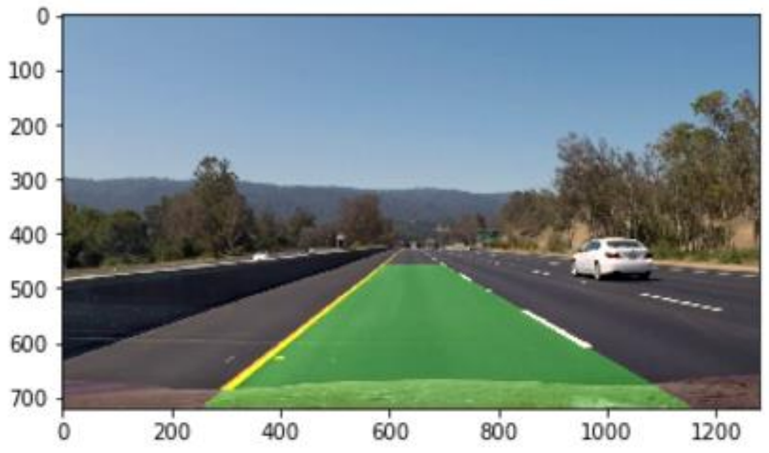

Figure 5. Area between the found lines in the original image.

2. Process the video file:

Each from the video can be process as describe above and the final output can be saved in the output file. To make the process little bit faster, instead of using sliding windows to find lines in each frame, can just search around the area the lines found in the previous frame.

Also after processing the each frame before feeding to the line detection algorithms, I have added black rectangles close to the bottom of image between the expected area of the lines, this can remove the extra features can come from the processing the image and lead to more accurate line detection. To reduce the jumps and vibration in the lines between the frames and make smoother line and polygon detection, I always kept the last 20 detected line in a repo and averaged them to get the position of the lines in the current frame.

For the detected left and right lines in each frame the curvatures are calculated and added to the frame image on the top left corner (left number shows the left line curvature and right number shows the right line curvature in meter). The calculated curvatures are not precise when the lines are straight as in this regions, it gets very sensitive to the fitted line parameters. Also for each video frame the offset of vehicle from center of lane is calculated, calibrated to meter scale and showed on the output image.

As shown in the notebook the time to process the video is pretty long, which come from distortion correction of the initial frame from video. It is possible to process the video without the distortion corrections, the lines can still be detected with the same quality, but the calculated curvatures won't be precise.