Advanced Lane Finding

Self-Driving Car Nano Degree - Term1: Project 4

Objective:

To apply Computer Vision Concepts to accurately find the lanes in a video under different scenarios.

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.

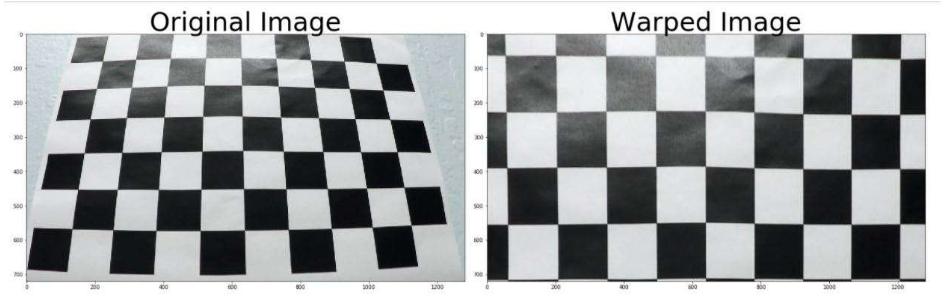
Overview:

For this project the solution to create a pipeline to meet the goals is divided into 3 topics:

1. **Camera Calibration**
2. **Image Pre-Processing (Lane Identification) & Perspective Transform**
3. **Lane Marking**

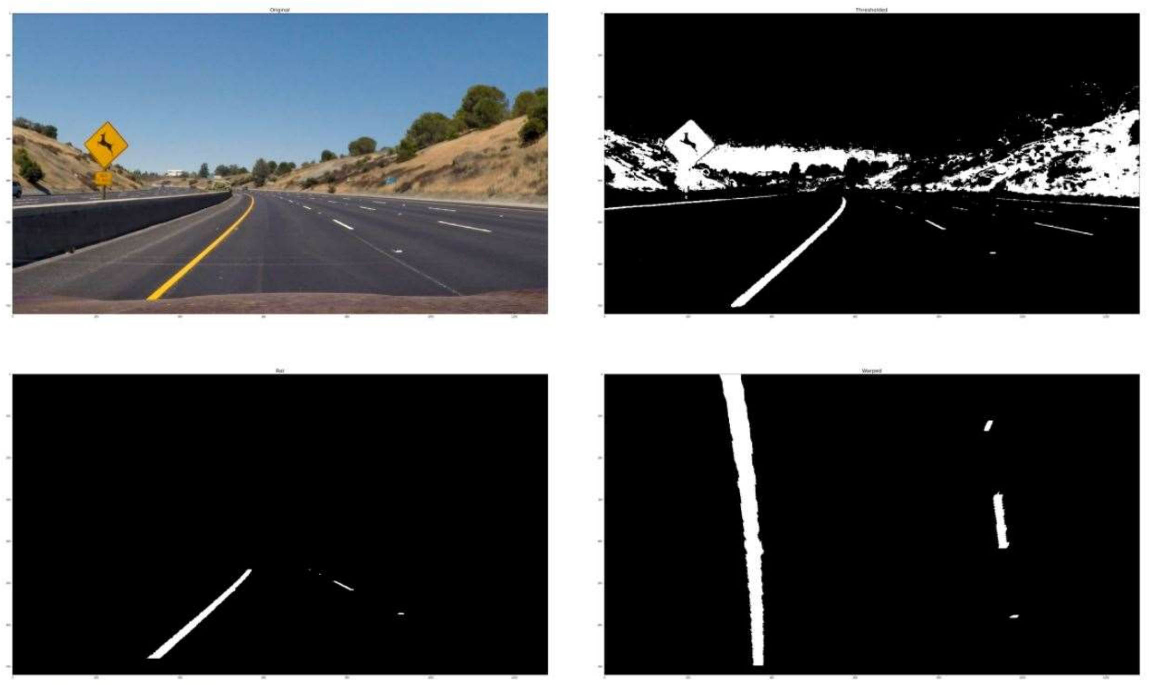
1. Camera Calibration

- First step of the pipeline is to obtain the Camera Matrix and Distortion Coefficients of the Camera to convert 3D Images to 2D and to correct the distortions in the images that is read through the lens camera.
- We do so by taking a good sample (20 images) of Chessboard Images and finding the corners in them.
- Use these corners to calibrate the Camera and obtain the Camera Matrix and Distortion Coefficients in them.
- Use this Camera Matrix and Distortion Coefficients to undistort the images for further use in the pipeline.



2. Image Pre-Processing (Lane Identification) & Perspective Transform

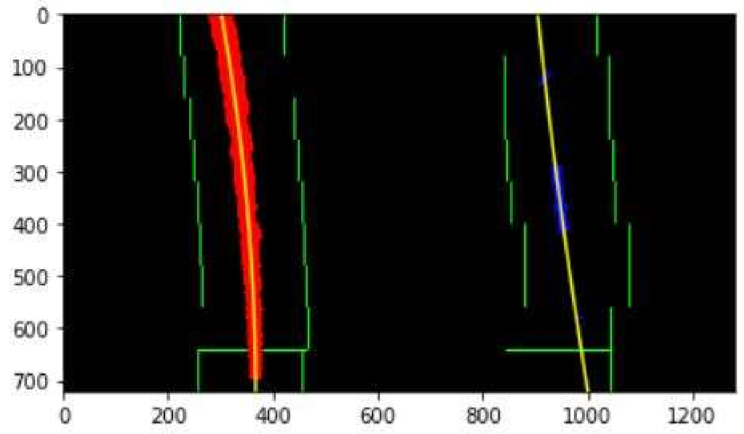
- Now that we have an undistorted image, we can proceed to find the lane lines, the first step would be to find the Lane Pixels (Yellow and White) under different lighting conditions.
- Initially tried a lot of combinations of Sobel, Magnitude, Direction, H, S, C & L Thresholding on the test images + a few more challenge video images to find the best way to identify the yellow and white pixels.
- This was not very fruitful, but then after reading through some articles on the images found a simple and effective way to find the yellow and white lane pixels under different lighting conditions.
- Courtesy: http://www.learnopencv.com/color-spaces-in-opencv-cpp-python/
- Used this info and identified the thresholds for shades of Yellow using H,S & V channels in the HSV format and using the L channel in the HSL format.
- Next we only need to focus on Lane Lines so use a Trapezoidal Mask and remove every other detail from the image. Now we have a binary image with only lane line pixels triggered high.
- A Bird's Eye View is considered very helpful, so used the manually tuned Source and Destination Co-ordinates to perform Perspective Transform on the Binary Images to get a top down view of the direction and gradient of the lane line pixels.



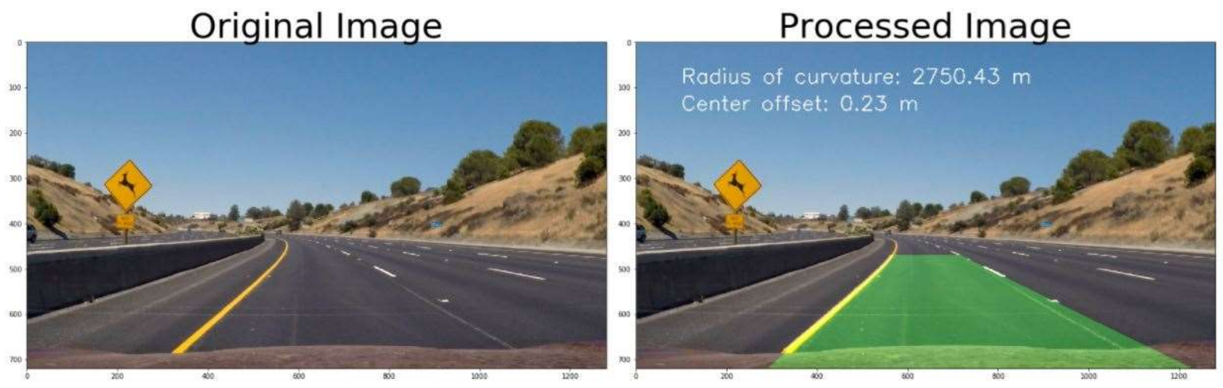
3. Lane Detection & Marking

- Now that we have suitable information of the lane pixels, we need to find the starting points (usually to the left & right of the image center), the trajectory/direction of the lane pixels and finally the 2D polynomial that can fit the left and the right pixels.
- First, I used the code from the lesson to find & track the left and right lane pixels using the sliding window search on a histogram of the vertical axis values of the bottom half of the image.
- This helps use to find the positions left and the right lane pixels in the bottom of the image, with which we can use a margin of +/- 100 pixels and find the positions of left and right lane pixels in the top half.

- Once found, we use a 2D polynomial to fit a line on the identified left and right pixels.



- For further frames in the video, we perform extrapolation and check if we the newly predicted left and right lane positions are not deviating significantly from the running average previous values we obtained.
- For paucity of time, I referenced Subodh Malgonde's project (https://github.com/subodh-malgonde/advanced-lane-finding/blob/master/Advanced_Lane_Lines.ipynb) for calculating the Lane Positions of further frames, instead of writing on my own from scratch.
- If we did not find a suitable lane lines then we re-calculate from scratch using the sliding window method.
- After we have an average left and right lane line values, we use Perspective Transform to draw the calculated lines back on the original image.



Shortcomings & To Do:

- My primitive solution fails to work in Challenge Videos, the to do's below (especially thresholding) needs to be worked on for solving the challenge videos.
- The Thresholding used is a very primitive one, it needs to be fine tuned with Sobel Thresholds to handle all lighting (including night time) cases presented in the challenge videos.
- I want to experiment with the idea of using a dynamic adaptive thresholding based on the brightness OR darkness in the Region Of Interest in the images from the challenge videos.
- Better handle continuity of lane detection for a new frame based on historical data.

- Handle scenarios where only one lane is visible OR there is a vehicle on one of the lanes in front of us.
- Ensure that the white/yellow markings in the center of the lane does not interfere with the Sliding Window Calculations.
- Smooth transitions during lane switching (need a new input video for this)
- Optimum Perspective Transform logic to handle tight turns and steep ascent and descent.