ADVANCED LANE FINDING

- ***The goals of this project is to make a pipeline that finds lane lines on the road***

PROJECT OVERVIEW

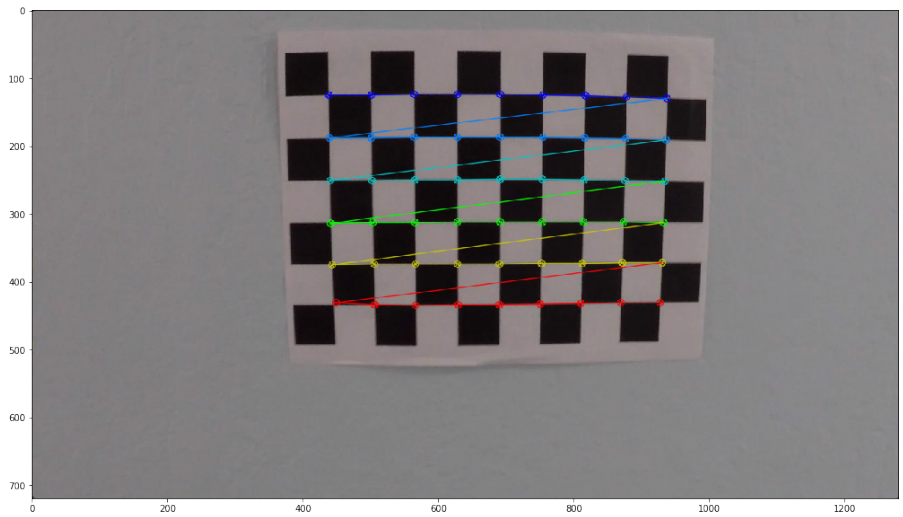
In this project, I have used computer vision techniques to identify lane boundaries and compute the lane metrics (radius of curvature, Offset to the center). I followed the steps cited bellow:

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

1. CAMERA CALIBRATION (LOOK FOR SECTION 1 IN THE NOTEBOOK FOR MORE DETAILS ABOUT THE CODE)

I used cv2.findChessboardCorners() for finding all the corners in the given image. I saved these corners from all the read images to imgpoints. Using cv2.calibrateCamera() with obtained imgpoints and real world co-ordinates, it outputs Camera Matrix, Distortion Coefficient, Rotational Vectors, Translation vectors with which we can undistort any image captured from the same camera.

One of the image with detected corners:



2. DISTORTION CORRECTION (LOOK FOR SECTION 1 IN THE NOTEBOOK FOR MORE DETAILS ABOUT THE CODE)

I used Camera Matrix and Distortion Coefficient from previous output to undistort the image using cv2.undistort(). The images may seem to be similar after undistortion.

The difference can be seem at the edges of the image for the following example. Other examples are saved in the folder "output_images" submitted with the project)

3. COLOR TRANSFORMS, GRADIENTS (LOOK FOR SECTION 2 IN THE NOTEBOOK FOR MORE DETAILS ABOUT THE CODE)

We will be using 2 types of gradient thresholds :

1. Along the X axis to detect horizontal gradient
2. Directional gradient threshold to detect edges closer to vertical

We will apply 3 color thresholds: R & G, L and S channel threshold,

1. S_channel performs good for detecting bright yellow and white lanes
2. Red & Green thresholds so that yellow lanes are fully detected
3. L_channel to keep clear shadowed pixels

Then combine all the thresholds :

[(R_G_Channels) AND (L_Channel)] AND [(S_Channel) OR (sobelx + directional gradient)]

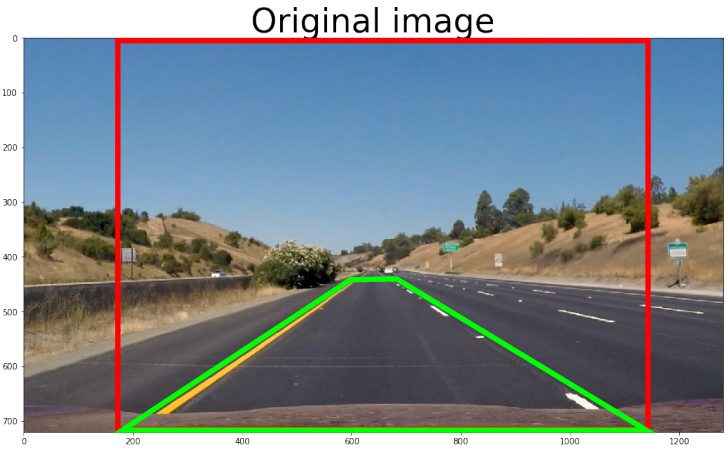
(other examples are saved in the folder “output_images” submitted with the project)

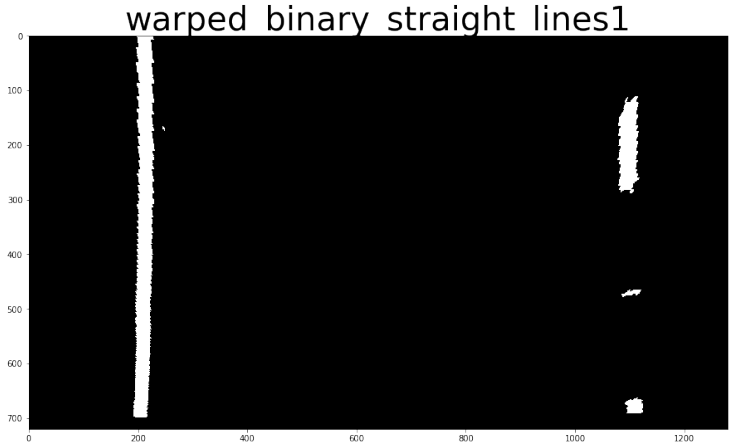
4. PERSPECTIVE TRANSFORM (LOOK FOR SECTION 3 IN THE NOTEBOOK FOR MORE DETAILS ABOUT THE CODE)

Now that we can undistort images and performing thresholds in order to detect useful informations, we need to focus our information selection only at the portion of the image we are interested in -> the road.

To focus on that portion of the image we will then transform the binary image to show the birds eye view.

Source_points = 4 vertices of the Red rectangle
Destination_points = 4 vertices of the Green rectangle





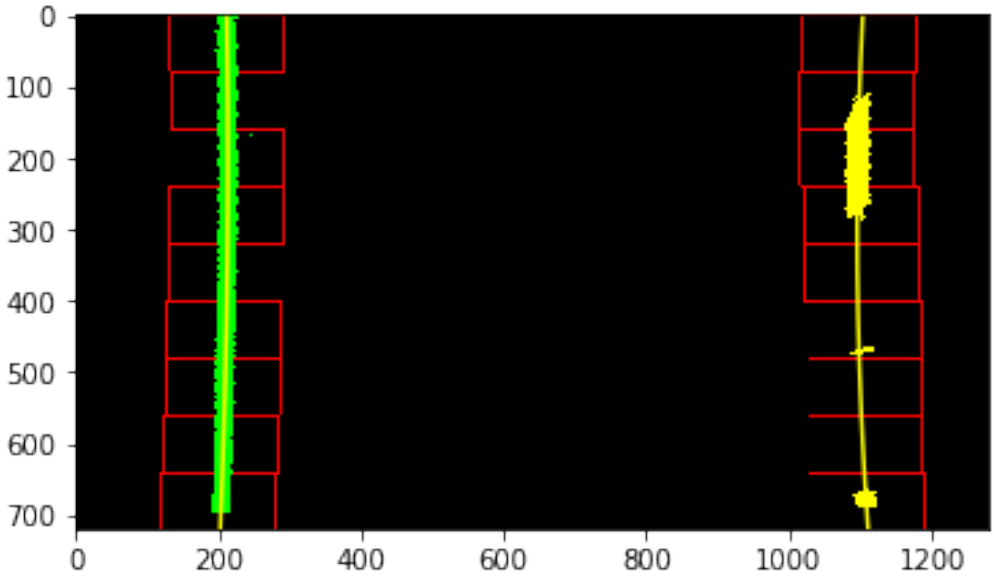
5. LINE FINDING METHOD (LOOK FOR SECTION 4 IN THE NOTEBOOK FOR MORE DETAILS ABOUT THE CODE)

After applying calibration, thresholding, and a perspective transform to the image, we have a binary image where the lane lines stand out clearly. However, you still need to decide explicitly which pixels are part of the lines and which belong to the left line and which belong to the right line. Plotting peaks in a histogram of where the binary activations occur across the image is one potential solution for this.

With this histogram we are adding up the pixel values along each column in the image. In our thresholded binary image, pixels are either 0 or 1, so the two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. We can use that as a starting point for where to search for the lines. From that point, we can use a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame.

I used the output from perspective for fitting the lane. Using histogram of the perspective image, I detected the left & right base using np.argmax(). Starting from that initial points left and right points are collected by deciding specific number of windows. Each window has left and right points correspoding to a lane. Then, polynomial corresponding to each lane is obtained by np.poly-fit(), it's is used for fitting the lane.

Example of lane fitting:



6. LANE METRICS(CURVATURES, OFFSET) (LOOK FOR SECTION 5 IN THE NOTEBOOK FOR MORE DETAILS ABOUT THE CODE)

After defining conversions in x and y from pixels space to meters, I used the formula explained in the course to compute the radius of curvature of each lane and the position of the vehicle with respect to center.(loot at section 5.1 & 5.2 in the notebook)

7. ESTIMATING LANE CURVATURE AND VEHICLE POSITION (LOOK FOR SECTION 5 IN THE NOTBOOK FOR MORE DETAILS ABOUT THE CODE)

Now that we have a warped binary image, and fitted lines with a polynomial and arrays called ploty, left_fitx and right_fitx, which represent the x and y pixel values of the lines. We can then project those lines onto the original image using the Draw function(section 5.3)

Below an example output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

I tested the complete pipeline using a class (section 7) on the project_video : the output is saved in : output_videos

DISCUSSION

Issues

+ I had to try many gradient and color channnel thresholding to detect a good quality edges with minimal noises.
+ The lanes lines in the challenge and harder challenge videos are difficult to identify. They are either too bright or too dark.
+ The challenge video has a section where the car goes underneath a tunnel and no lanes are founded
+ The lanes in the challenge video change in color, shape and direction.

+ On the challenge video, I was facing this error : TypeError("expected non-empty vector for x"). One of the frame was not having detected left line in the binary warped lane, so np.polyfit() couldn't fit the left lane line. I had to modifiy the blind_search, quality_lane functions and also the main pipeline to avoid this error (empty arrays that contains pixel lines positions)

Failures

The pipeline seems to fail for the harder challenge video and perform poorly on the challenge video.

& Areas of Improvement

As an improvements I think that we should :

+ Take a better perspective transform for the harder_challenge --> choose a smaller section to take the transform since the lenght of a lane is shorter than the project_video and then average over a smaller number of frames because the shape and direction of lanes changes fast.

+To improve the lane detection on the challenge video(in general too), I see that we should carefully find an optimal color and channel tresholds to overcome the changes in gradients, lights and colors