Advanced Lane Finding 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:

Code for camera calibration is located in code cell #2. To calibrate the camera, I read the chessboard images provided in camera_cal folder and converted them to gray scale to find out the corners of the image. To find the corners I used cv2. findChessboardCorners() function which returns the coordinates of the found corners from chessboard. Using these image points and a 3D object points to calibrate camera. calibrate_camera() function returns the camera matrix and distortion coefficients which will be used in later steps.

Distortion Correction:

Code cell #3 shows the code to correct image distortion using undistort() function which takes camera matrix and distortion coefficients as inputs and un-distort the image. In code cell used correct_distortion function to undistort images in camera_cal folder. Un-distorted images are save in **camera_cal_undistorted** folder of this zip file

Perspective Transform:

In code cell #5, I performed perspective transformation, this is done first by identifying 4 rectangle points from the test image(src) and also 4 rectangle points in the birds-eye view image. Using these two points to get the perspective transformation. getPerspectiveTransform() is used for calculating perspective transformation matrix. Also inverse of this matrix calculated to un-warp the image.

Here is the example of warped test image

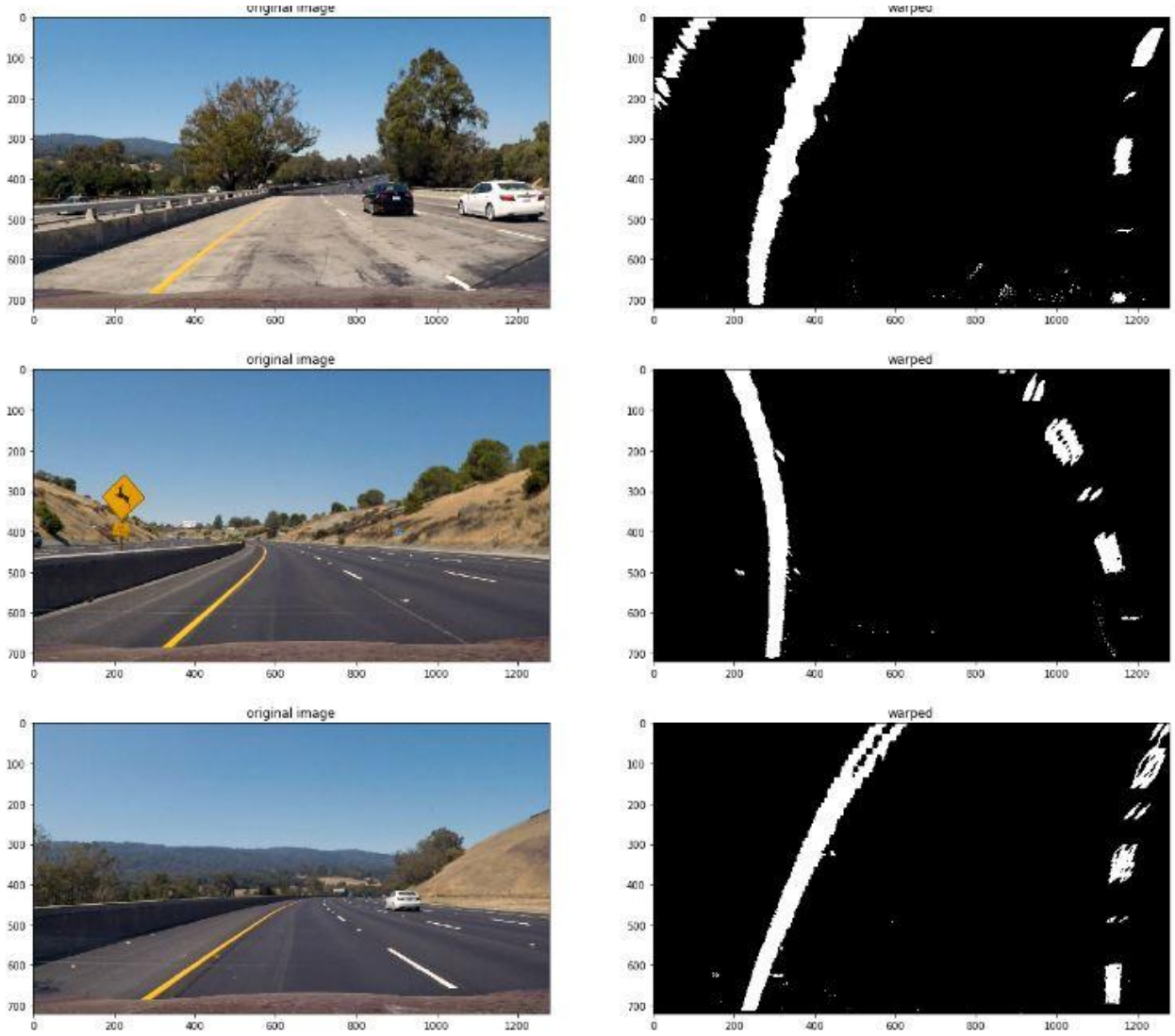
Crating Thresholded binary image:

Code cell #6 shows different functions that can be used to find the line pixels, based on sobel threshold, magnitude threshold, direction threshold, hls threshold.

Code cell #7 defines a function get_binary_warped_img which uses threshold functions defined in code cell#6, and creates an image where lane pixels can be easily found. Also this function returns warped image of lane pixels found.

Code cell #8 is to test the get_binary_warped_img function, also it displays the original test image and the image that is returned from get_binary_warped_img function.

Here are the example test images after applying threshold



Lane Curvature and vehicle position detection:

Code cell #9 contains two functions to calculate the lane curvature and vehicle position in the line. get_roc() will calculate the curvature radius, it does that by fitting a curve into the given polynomial and uses the coefficients in the formula given in lecture notes. get_veh_pos function is responsible for calculating vehicle position from the given fits (left lane and right lane fits), I calculate the offset be calculating frame mid-point, two lanes mid-point and subtracting it to get the actual offset.

Detect lane pixels and fit to find the lane boundary:

Code cell #10 has the functionality to find the lane boundaries. First we have to process the image to get the thresholded image, and them use histogram, sliding window method mentioned in the class notes to calculate the lane pixel points and then using those points to fit the curve for both left and right lanes. Once the left and right fits are found, radius or curvature and vehicle positon is calculated and overlaid it on the original image. Also I created a polynomial using points from the left and right fits to create an overlay image, this image is un-warped and drawn on top of original image to show the lane boundaries.

Code cell #11 has the code to test the pipeline (from code cell #10). This code reads all images from test_images folder and apply pipeline and displays the images after pipeline (lane boundaries detection), also these images are written to **output_images** folder in this zip.

Code cell #12 contains the code to read image from the video and apply pipeline to finding the lane boundaries, radius or curvature, and vehicle position. After all the frames are calculated, images are written to output.mp4 which can be found in this zip

Visual display:

Code cell #10 in here I'm displaying the output of the test video where each frame is shown with the overlaid lane boundaries and numerical estimation of lane curvature and vehicle position.

My implementation is failing in the challenge video; I assume it is because of the road appearance. Even though the lane marking is clear, my pipeline is trying to pick the middle of the lanes because it looks like it as a lane even though it is an intersection of old and new road. In my implementation, I'm calculating the lane line pixels and fit for each and every frame of a video, to make lane finding better I need to keep track of previous frames lane location and make adjustments in the new frame to overcome the outliers (or jumping lanes)