Self Driving Car Engineer Nanodegree
Udacity

Manikandan R
vasumaniram@gmail.com

Finding Advanced Lane Lines - Project 2

Objective

The objective of the project is to do the following in order to design a Computer Vision pipeline to detect curved lane lines and radius of the curve:

- 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 the 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

Assumptions

- The right and left lanes will be approximately within the same region in every image as its taken from the fixed front camera of the Car.
- Every image is a color image and will have three color channels (RGB).
- Both right and left markings are visible on every image.

1.Camera Calibration:

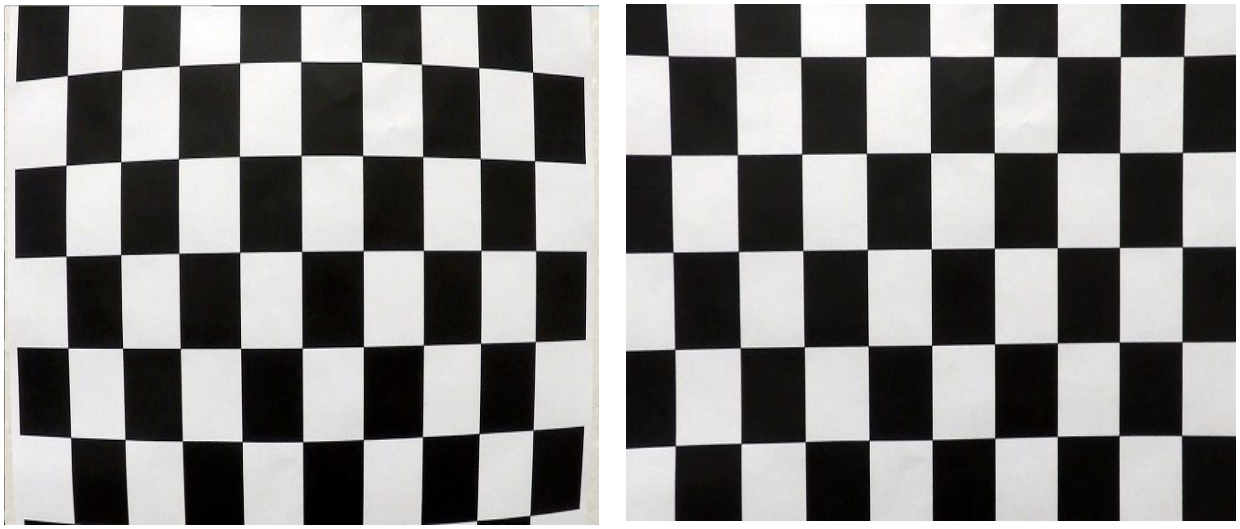
1.1 Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion-corrected calibration image.

Image distortion happens when the camera transforms the 3D objects into the 2D image and it changes the size and shape of the objects. We need to analyze the camera images to undistort the images and it requires the following,
1. Distorted image samples - 10 to 20 samples to have a better approximation. Chessboard images taken from different angle and position have been taken as sample images to calibrate camera as the corners are very clear and even.
2. Object Points describe the (x,y,z) coordinates of the real world Object. Considering the Chessboard are fixed in x and y with z set to 0, we can take Object points as Chess board positions with z appended as 0(ie. [(0,0,0),(0,1,0)..].Hence the Object points are going to be the same for every image.
3. Image Points describe the (x,y) coordinates of the Image. OpenCV has a function *detectChessboardCorners()* which could help us build the image points for every image.

For every distorted Chessboard image, the Objects points will be the same and the Image points will be collected by calling *cv2.detectChessboardCorners()* function. Object points and Image points will be used to call the function *cv2.calibrateCamera()* which will return the Camera Matrix and Distortion coefficients to undistort other images. The following pictures

show the distorted image and the undistorted image after the camera calibration.

Distorted Image *Undistorted Image*



Picture - 1.1 - Undistorted vs Distorted Images

Open CV functions used	*ret, corners = cv2.findChessboardCorners(gray,(9,6),None)* *ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,imgpoints,gray.shape[::-1],None,None)*
Undistorted image saved under	/ouput_images/camera_cal
Code reference - Filename	
Code reference - Lined	

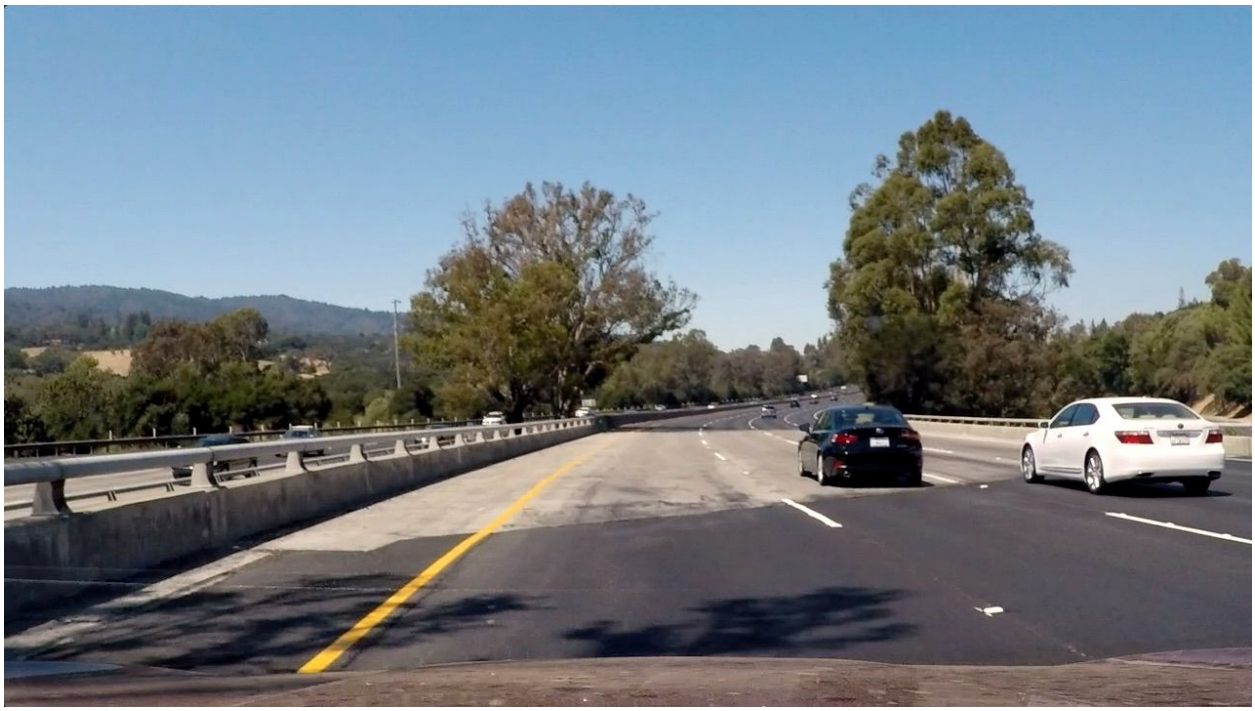
2. Pipeline (test images)

2.1 Provide an example of a distortion-corrected image.

Calculated Camera Matrix and Distortion coefficients have been applied to the test images and the images have been undistorted. The following is a example image before and after undistortion.

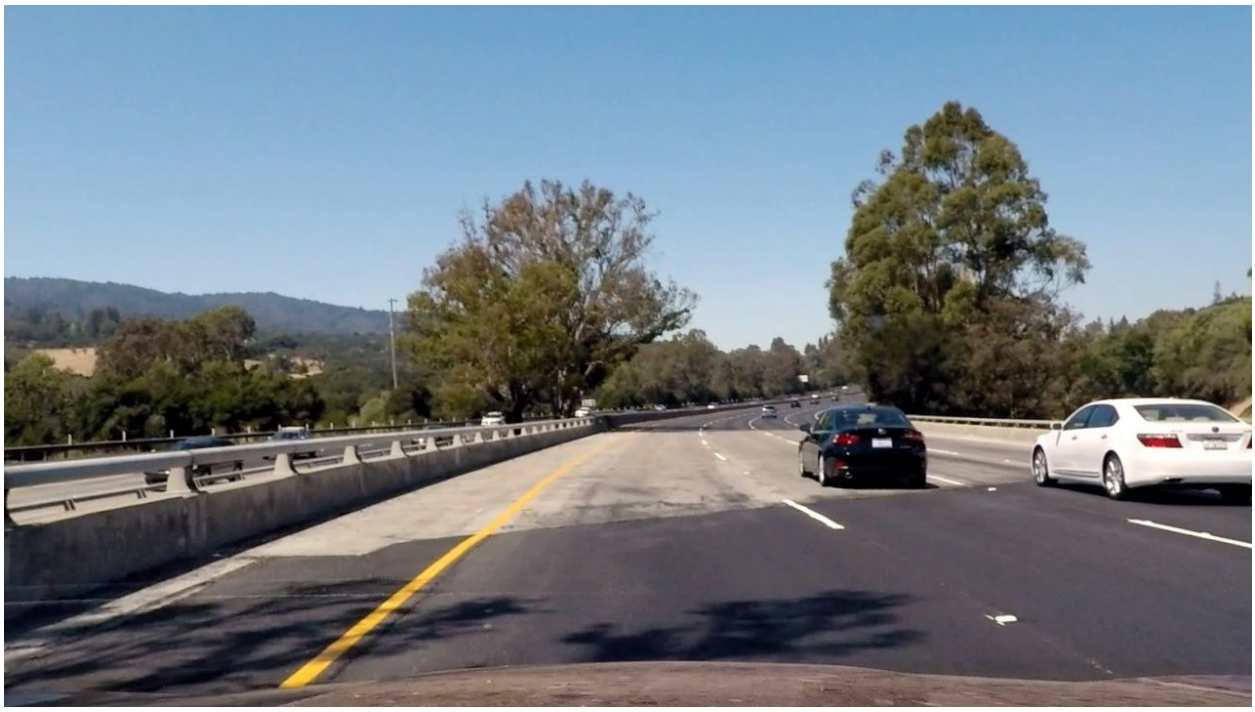
Open CV functions used	*cv2.undistort(img,mtx,dist,None,mtx)*
Undistorted image saved under	/ouput_images/test_images
Code reference - Filename	advanced_lane_finding.py
Code reference - Function Names	calibtrate_camera() pipeline()

Distorted Image:



Picture - 2.1.0 - Distorted Test Image

Undistorted Image:



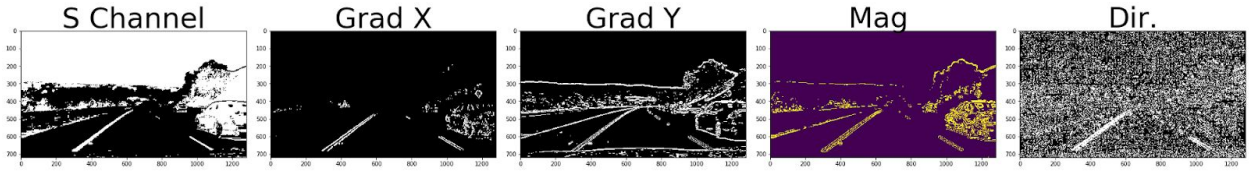
Picture - 2.1.1 - Undistorted Test Image

2.2 Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result

The objective here is to find the right mix of the following color and gradients which identifies the left and right lanes correctly.

1. Color space(RGB, HLV, HLS)
2. Gradients (Sobel X and Sobel Y direction / Gradient Magnitude / Gradient Direction

The following picture shows the result of each one of the Color and Gradient listed above.



Picture - 2.2.1 - Color & Gradient Thresholds

S channel of HLS performs good in picking up both yellow and white lane lines in various lighting condition(i.e shadows, very sunny etc). Sobel operator with X orientation is very good at picking the vertical lines compare to Sobel with Y orientation which picks the horizontal lines. After trying the combinations of color and gradient measures, decided to use ‘S’ channel of ‘HLS’ color space and Sobel with X orientation.

The following picture depicts the binary representation of the two lane lines clearly found after the perspective transform has been done.

Picture - 2.2.2 - Binary Image Of Warped Image

Open CV function used	*cv2.Sobel(gray,cv2.CV_64F,x,y,ksize=sobel_kernel)* *np.arctan2(y,x)*
Code reference - File Name	advanced_lane_finding.py
Code reference - function Names	get_hls_channels() abs_sobel_thresh() mag_thresh() dir_threshold() color_threshold() gradient_color_threshold()

2.3 Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

Perspective Transform is used to transform the Region Of Interest into a different perspective defined by us. Its very import for finding the lane lines because,

1. In real world, the parallel lines will never intersect but in the 2D image all the parallel lines will intersect in some point. So find isolate the lane lines exactly as they are in the real world, we need to transform them into a new perspective(Birds-Eye-View).
2. When the lanes are curved, its required to find the radius of the curvature because only if we know the radius of the curvature, the steering angle could be tuned accordingly and the speed could be lowered if its sharp curve.

Perspective transform in OpenCV is a three step process,

1. Need to mark the source points of Region Of Interest,its the lane lines in this case. Also need to find the destination points where the transformed image would be plotted back.
2. Using the source and destination points, need to compute the Transformation matrix and the Inverse Transformation matrix. Transformation matrix will be used to transform the image into perspective transformed image and Inverse Transformation matrix would later be used to transform back from perspective to original image.

3. Using the Transformation matrix, the perspective transformed warp image could be achieved

The following is a perspective transformed(Birds-Eye-View) Warped image

Picture - 2.3.1 - Perspective Transformed(Warped) Image

Open CV function used	*cv2.getPerspectiveTransform(src,dst)* *cv2.warpPerspective(undist, M, img_size,flags=cv2.INTER_LINEAR)*
Code reference - Filename	advanced_lane_finding.py
Code reference - Function Names	get_warp_matrices()

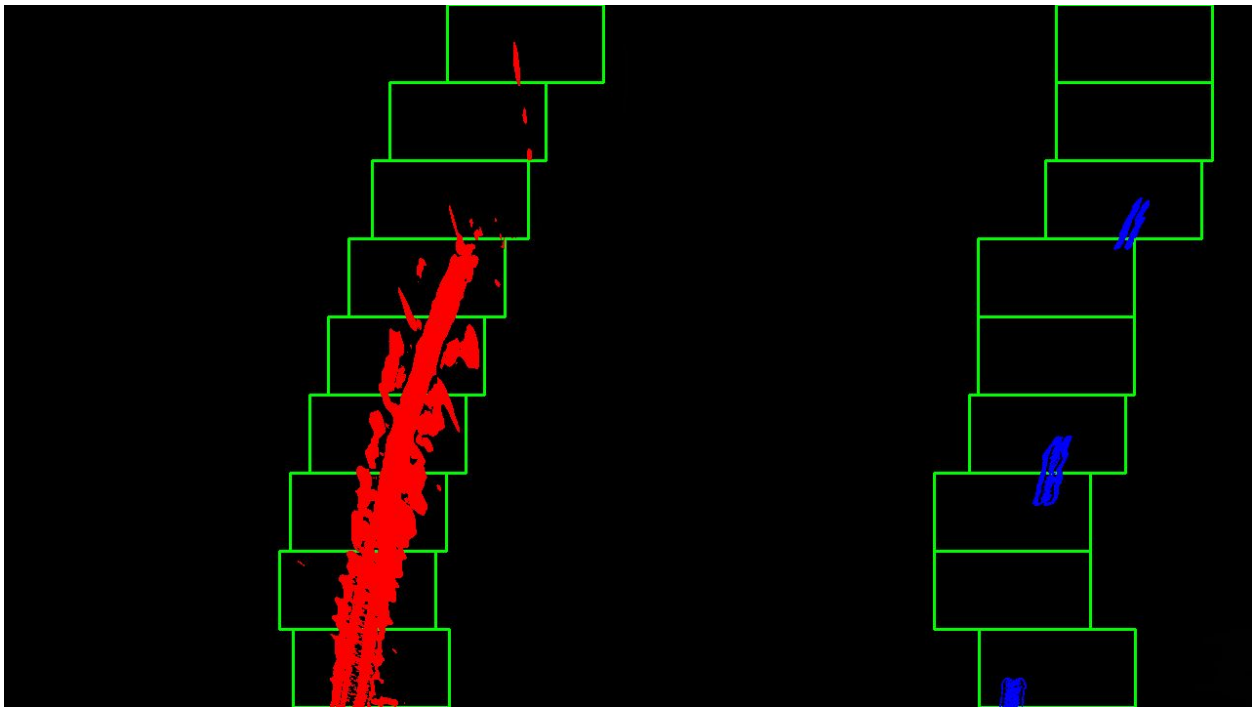
2.4 Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

From the binary warped image where the lane lines are visible, the next step to be identifying the lane lines and fit the polynomial function of the lane lines.

1. Finding the peaks of the pixel distributions from the binary image helps finding the region to start finding the lane lines. Its achieved by summing up the histogram values of the pixels(1/0).
2. Defined a sliding window of size(80,160) to start searching for the pixel activations. Based on the pixel activations against the threshold value(50), the sliding window would be recentered to follow and capture the line. If there is no or less change, it would just stack up the windows.
3. Once the lane line x, y pixels are found for both left and right lanes, they will be used to find the 2nd order polynomial coefficients using np.polyfit() function and would further the coefficients be applied on 'y' values to form a line using np.poly1d() function.

    Second Order Polynomial = $f(y) = Ay2 + By + C$

The following picture depicts how the sliding window was stacked up to find the active pixels from the binary warped image.



Picture - 2.4.1 - Sliding Window Search On Binary Warped Image

Open CV function used	np.polyfit() np.poly1d()
Code reference - Filename	advanced_lane_finding.py
Code reference - Function Names	find_lane_pixels() fit_polynomial()

2.5 Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to the center.

Finding the radius of the curvature of the lines are very important for changing the steering angle and speed of the vehicle and the following is the formula for calculating the same.

$$Rcurve = [1+(dy/dx)2]3/2 \;/\; |dy2/d2x| /$$

The following is a R Curve formula after taking the first and second derivative of the above R curve formula,

$$Rcurve = (1 + (2Ay + B)2)3/2 \;/\; |2A|$$

As we are very interested in knowing the curvature radius values as close to the vehicle, we will consider taking the maximum of the ‘y’ value and the left and right line fits returned by the previous polynomial to calculate the curvature radius values for both right and left.

Code reference - Filename	advanced_lane_finding.py
Code reference - Function Names	measure_curvature_pixels()

2.6 Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

The following image is an result of running the Advanced Lane Finding Computer Vision Pipeline applied on one of the test images under the folder */test_images.*

Picture - 2.6.1 - Final Result With Lane Lines Marked With Curvature Radius

Code reference - Filename	advanced_lane_finding.py
Code reference - Function Names	pipeline()

3. Pipeline (video)

3.1 Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!)

Project Folder: output_videos/project_video.mp4

Youtube Link: - Processed Project Video

4. Discussion

4.1 Briefly discuss any problems/issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

1. Around 25 to 27 seconds of the project video, one of the perspective transformed frame had a small object with high pixel density found on the right bottom. It influenced the sliding window to fall off the right end and detect a wrong lane line. Had to try 'error and trial' to find a source and destination points for the perspective transform to get thru this issue.
2. While trying to use the smart search to leverage the previously detected lane lines to identify the line pixels, at half of the project video, the right lane started wrongly fit into left side and merged. Initiate a full sliding window search,when the found lines are not parallel or very distant would fix this issue.