SAEINDIA

“Advance Lane Detection”

Submitted by:
(Team Genesis)
Anurag Mahapatra

Contents:

1. Introduction
2. Theoretical contents/ Procedure
3. Code with explanation
4. Inference
5. Result
6. Reference (links)

1. INTRODUCTION:

- Computer Vision algorithm to compute road curvature and lane vehicle offset using OpenCV Image Processing, Camera Calibration, Perspective Transform, Color Masks, Sobels and Polynomial Fit.
-

- The Advanced Lane Finding project is a step further from Lane Lines Detection in identifying the geometry and curvature of the road ahead.
- Using a video recording of highway driving, this project's goal is to compute the radius of the curvature of the road. Curved roads are a more challenging task than straight ones.
- To correctly compute the curvature, the lane lines need to be identified but on top of that, the images need to be undistorted. Image transformation is necessary for camera calibration and for perspective transform to obtain a bird's eye view of the road.
- This project is implemented in Python and uses OpenCV image processing library.

2. Theoretical contents/ Procedure:

❖ Camera Calibration:

- Optic distortion is a physical phenomenon that occurs in image recording, in which straight lines are projected as slightly curved ones when perceived through camera lenses. The highway driving video is recorded using the front facing camera on the car and the images are distorted. The distortion coefficients are specific to each camera and can be calculated using known geometrical forms.
- Chessboard images captured with the embedded camera are provided in the camera_cal folder. The advantage of these images is that they have high contrast and known geometry. The images provided present 9 * 6 corners to work with.
- Object points are set based on the common understanding that in a chess board pattern, all squares are equal. This implies that object points will have x and y coordinates generated from grid indexes, and z is always 0. The image points represent the corresponding object points found in the image using OpenCV's function findChessboardCorners.
- After scanning through all the images, the image point list has enough data to compare against the object points in order to compute camera matrix and distortion coefficients. This leads to an accurate camera matrix and distortion coefficient identification using the 'calibrateCamera' function.
- OpenCV undistort function is used to transform the images using the camera matrix and distortion coefficients.

- The result of the camera calibration technique is visible when comparing these pictures. While on the chessboard picture the distortion is more obvious, on the road picture it's more subtle. Nevertheless, an undistorted picture would lead to an incorrect road curvature calculation.

❖ Perspective Transform from Camera Angle to Bird's Eye View:

- To calculate curvature, the ideal perspective is a bird's eye view. This means that the road is perceived from above, instead of at an angle through the vehicle's windshield.
- This perspective transform is computed using a straight lane scenario and prior common knowledge that the lane lines are in fact parallel.
- Source and destination points are identified directly from the image for the perspective transform.

```
(209, 720, 616, 432, 666, 432, 1121, 720)
```

```
(209, 720, 616, 432, 666, 432, 1121, 720)
```

- OpenCV provides perspective transform functions to calculate the transformation matrix for the images given the source and destination points. Using the warpPerspective function, the bird's eye view perspective transform is performed.

❖ Process Binary Thresholded Images:

- The objective is to process the image in such a way that the lane line pixels are preserved and easily differentiated from the road. Four transformations are applied and then combined.
    - Transform image to grayscale
    - Detect pixels that are white in the grayscale image
    - Convert image to HLS
    - Detect pixels that are yellow using the hue component

- Transformations:
    - The first transformation takes the x sobel on the gray-scaled image. This represents the derivative in the x direction and helps detect lines that tend to be vertical. Only the values above a minimum threshold are kept.
    - The second transformation selects the white pixels in the gray scaled image. White is defined by values between 200 and 255 which were picked using trial and error on the given pictures.
    - The third transformation is on the saturation component using the HLS colorspace. This is particularly important to detect yellow lines on light concrete roads.
    - The fourth transformation is on the hue component with values from 10 to 25, which were identified as corresponding to yellow.

❖ Lane Line Detection Using Histogram:

- The lane line detection is performed on binary thresholded images that have already been undistorted and warped. Initially a histogram is computed on the image. This means that the pixel values are summed on each column to detect the most probable x position of left and right lane lines.
- Starting with these base positions on the bottom of the image, the sliding window method is applied going upwards searching for line pixels. Lane pixels are considered when the x and y coordinates are within the area defined by the window. When enough pixels are detected to be confident they are part of a line, their average position is computed and kept as the starting point for the next upward window.
- All these pixels are put together in a list of their x and y coordinates. This is done symmetrically on both lane lines. leftx, lefty, rightx, righty pixel positions are returned from the function and afterwards, a second-degree polynomial is fitted on each left and right side to find the best line fit of the selected pixels.
- Here, the identified left and right line pixels are marked in red and blue respectively. The second degree polynomial is traced on the resulting image.

❖ Detection of Lane Lines Based on Previous Cycle:

- To speed up the lane line search from one video frame to the next, information from the previous cycle is used. It is more likely that the next image will have lane lines in proximity to the previous lane lines. This is where the polynomial fit for the left line and right line of the previous image are used to define the searching area.
- The sliding window method is still used, but instead of starting with the histogram's peak points, the search is conducted along the previous lines with a given margin for the window's width.
- The search returns leftx, lefty, rightx, righty pixel coordinates that are fitted with a second degree polynomial function for each left and right side.

- The transformation applied on the previous cycle speeds up the complete process of lane line detection.

❖ Calculate Vehicle Position and Curve Radius:

- To calculate the radius and the vehicle's position on the road in meters, scaling factors are needed to convert from pixels. The corresponding scaling values are 30 meters to 720 pixels in the y direction and 3.7 meters to 700 pixels in the x dimension.
- A polynomial fit is used to make the conversion. Using the x coordinates of the aligned pixels from the fitted line of each right and left lane line, the conversion factors are applied and polynomial fit is performed on each.
- The radius of the curvature is calculated using the y point at the bottom of the image. To calculate the vehicle's position, the polynomial fit in pixels is used to determine the x position of the left and right lane corresponding to the y at the bottom of the image.
- The average of these two values gives the position of the center of the lane in the image. If the lane's center is shifted to the right by nbp amount of pixels that means that the car is shifted to the left by nbp * xm_per_pix meters. This is based on the assumption that the camera is mounted on the central axis of the vehicle.
- EXAMPLE:
    - left curve radius in meters = 340.37514695323364
    - right curve radius in meters = 1164.9469563433286
    - vehicle position relative to center = 0.04757142857142857

❖ Project Lane Delimitations Back on Image Plane:

- Create an image to draw the lines.
- Recast the x and y points into a usable format for cv2.fillPoly().
- Draw the lane onto the warped blank image.
- Warp the blank back to the original image space using the inverse perspective matrix (Minv).
- Combine the result with the original image.
- Plot the result and apply the Lane Finding Pipeline on Video:

3. Codes with explanation:

- <u>COMPLETE CODE:</u>

#import libraries

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
import glob
import os
from moviepy.editor import VideoFileClip
%matplotlib inline
```

STEP 1: Camera Calibration

```
def distortion_factors():
   # Prepare object points
   # From the provided calibration images, 9*6 corners are identified
   nx = 9
   ny = 6
   objpoints = []
   imgpoints = []
   # Object points are real world points, here a 3D coordinates matrix is
generated
   # z coordinates are 0 and x, y are equidistant as it is known that the
chessboard is made of identical squares
   objp = np.zeros((6*9,3), np.float32)
   objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
   # Make a list of calibration images
   camera_cal = "OneDrive/Desktop/Study/SAE/Advance lane
detection/Advanced-Lane-Detection-master/Advanced-Lane-Detection-
master/camera_cal/"
   os.listdir(camera_cal)
   cal_img_list = os.listdir(camera_cal)
   # Imagepoints are the correspondent object points with their
coordinates in the distorted image
   # They are found in the image using the OpenCV
'findChessboardCorners' function
   for image_name in cal_img_list:
```

```
        import_from = camera_cal + image_name
        img = cv2.imread(import_from)
        # Convert to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # Find the chessboard corners
        ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)
        # If found, draw corners
        if ret == True:
            # Draw and display the corners
            #cv2.drawChessboardCorners(img, (nx, ny), corners, ret)
            imgpoints.append(corners)
            objpoints.append(objp)
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
    #undist = cv2.undistort(img, mtx, dist, None, mtx)
    #export_to = 'camera_cal_undistorted/' + image_name
    #save the image in the destination folder
    #plt.imsave(export_to, undist)
    return mtx, dist
```

STEP 2: Perspective Transform from Car Camera to Bird's Eye View

```
mtx, dist = distortion_factors()
def warp(img):
    undist = cv2.undistort(img, mtx, dist, None, mtx)
    img_size = (img.shape[1], img.shape[0])
    offset = 300
    # Source points taken from images with straight lane lines, these are
to become parallel after the warp transform
    src = np.float32([
        (190, 720), # bottom-left corner
        (596, 447), # top-left corner
        (685, 447), # top-right corner
        (1125, 720) # bottom-right corner
    ])
    # Destination points are to be parallel, taken into account the image
size
    dst = np.float32([
        [offset, img_size[1]],          # bottom-left corner
        [offset, 0],                    # top-left corner
```

```
    [img_size[0]-offset, 0],          # top-right corner
    [img_size[0]-offset, img_size[1]]  # bottom-right corner
  ])
  # Calculate the transformation matrix and it's inverse transformation
  M = cv2.getPerspectiveTransform(src, dst)
  M_inv = cv2.getPerspectiveTransform(dst, src)
  warped = cv2.warpPerspective(undist, M, img_size)
  return warped, M_inv
```

STEP 3: Process Binary Thresholded Images

```
#img = cv2.imread('test_images/straight_lines1.jpg')
#img = cv2.imread('test_images/straight_lines2.jpg')
#img = cv2.imread('test_images/test1.jpg')
#img = cv2.imread('test_images/test2.jpg')
#img = cv2.imread('test_images/test3.jpg')
img = cv2.imread('OneDrive/Desktop/Study/SAE/Advance lane
detection/Advanced-Lane-Detection-master/Advanced-Lane-Detection-
master/test_images/test4.jpg')
#img = cv2.imread('test_images/test5.jpg')
#img = cv2.imread('test_images/test6.jpg')

def binary_thresholded(img):
  # Transform image to gray scale
  gray_img =cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # Apply sobel (derivative) in x direction, this is usefull to detect lines
that tend to be vertical
  sobelx = cv2.Sobel(gray_img, cv2.CV_64F, 1, 0)
  abs_sobelx = np.absolute(sobelx)
  # Scale result to 0-255
  scaled_sobel = np.uint8(255*abs_sobelx/np.max(abs_sobelx))
  sx_binary = np.zeros_like(scaled_sobel)
  # Keep only derivative values that are in the margin of interest
  sx_binary[(scaled_sobel >= 30) & (scaled_sobel <= 255)] = 1

  # Detect pixels that are white in the grayscale image
  white_binary = np.zeros_like(gray_img)
  white_binary[(gray_img > 200) & (gray_img <= 255)] = 1

  # Convert image to HLS
  hls = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
```

```
    H = hls[:,:,0]
    S = hls[:,:,2]
    sat_binary = np.zeros_like(S)
    # Detect pixels that have a high saturation value
    sat_binary[(S > 90) & (S <= 255)] = 1

    hue_binary =  np.zeros_like(H)
    # Detect pixels that are yellow using the hue component
    hue_binary[(H > 10) & (H <= 25)] = 1

    # Combine all pixels detected above
    binary_1 = cv2.bitwise_or(sx_binary, white_binary)
    binary_2 = cv2.bitwise_or(hue_binary, sat_binary)
    binary = cv2.bitwise_or(binary_1, binary_2)
    plt.imshow(binary, cmap='gray')

    #img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Draw figure for binary images
    #f, axarr = plt.subplots(1,6)
    #f.set_size_inches(25, 8)
    #axarr[0].imshow(img)
    #axarr[1].imshow(sx_binary, cmap='gray')
    #axarr[2].imshow(white_binary, cmap='gray')
    #axarr[3].imshow(sat_binary, cmap='gray')
    #axarr[4].imshow(hue_binary, cmap='gray')
    #axarr[5].imshow(binary, cmap='gray')
    #axarr[0].set_title("Undistorted Image")
    #axarr[1].set_title("x Sobel Derivative")
    #axarr[2].set_title("White Threshold")
    #axarr[3].set_title("Saturation Threshold")
    #axarr[4].set_title("Hue Threshold")
    #axarr[5].set_title("Combined")
    #axarr[0].axis('off')
    #axarr[1].axis('off')
    #axarr[2].axis('off')
    #axarr[3].axis('off')
    #axarr[4].axis('off')
    #axarr[5].axis('off')

    return binary
```

```
out_img = binary_thresholded(img)
```

```
#img = cv2.imread('test_images/straight_lines1.jpg')
#img = cv2.imread('test_images/straight_lines2.jpg')
#img = cv2.imread('test_images/test1.jpg')
#img = cv2.imread('test_images/test2.jpg')
#img = cv2.imread('test_images/test3.jpg')
#img = cv2.imread('test_images/test4.jpg')
#img = cv2.imread('test_images/test5.jpg')
img = cv2.imread('OneDrive/Desktop/Study/SAE/Advance lane
detection/Advanced-Lane-Detection-master/Advanced-Lane-Detection-
master/test_images/test6.jpg')

binary_thresh = binary_thresholded(img)
out_img = np.dstack((binary_thresh, binary_thresh, binary_thresh))*255
binary_warped, M_inv = warp(binary_thresh)
plt.imshow(out_img, cmap='gray')
```

STEP 4: Detection of Lane Lines Using Histogram

```
def find_lane_pixels_using_histogram(binary_warped):
    # Take a histogram of the bottom half of the image
    histogram = np.sum(binary_warped[binary_warped.shape[0]//2:,:],
axis=0)

    # Find the peak of the left and right halves of the histogram
    # These will be the starting point for the left and right lines
    midpoint = np.int(histogram.shape[0]//2)
    leftx_base = np.argmax(histogram[:midpoint])
    rightx_base = np.argmax(histogram[midpoint:]) + midpoint

    # Choose the number of sliding windows
    nwindows = 9
    # Set the width of the windows +/- margin
    margin = 100
    # Set minimum number of pixels found to recenter window
    minpix = 50

    # Set height of windows - based on nwindows above and image shape
    window_height = np.int(binary_warped.shape[0]//nwindows)
    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    # Current positions to be updated later for each window in nwindows
    leftx_current = leftx_base
    rightx_current = rightx_base

    # Create empty lists to receive left and right lane pixel indices
    left_lane_inds = []
    right_lane_inds = []

    # Step through the windows one by one
    for window in range(nwindows):
        # Identify window boundaries in x and y (and right and left)
        win_y_low = binary_warped.shape[0] - (window+1)*window_height
        win_y_high = binary_warped.shape[0] - window*window_height
        win_xleft_low = leftx_current - margin
        win_xleft_high = leftx_current + margin
        win_xright_low = rightx_current - margin
```

```
        win_xright_high = rightx_current + margin

        # Identify the nonzero pixels in x and y within the window #
        good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy <
win_y_high) &
        (nonzerox >= win_xleft_low) &  (nonzerox <
win_xleft_high)).nonzero()[0]
        good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy <
win_y_high) &
        (nonzerox >= win_xright_low) &  (nonzerox <
win_xright_high)).nonzero()[0]

        # Append these indices to the lists
        left_lane_inds.append(good_left_inds)
        right_lane_inds.append(good_right_inds)

        # If you found > minpix pixels, recenter next window on their mean
position
        if len(good_left_inds) > minpix:
            leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
        if len(good_right_inds) > minpix:
            rightx_current = np.int(np.mean(nonzerox[good_right_inds]))

    # Concatenate the arrays of indices (previously was a list of lists of
pixels)
    try:
        left_lane_inds = np.concatenate(left_lane_inds)
        right_lane_inds = np.concatenate(right_lane_inds)
    except ValueError:
        # Avoids an error if the above is not implemented fully
        pass

    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    return leftx, lefty, rightx, righty
```

```
def fit_poly(binary_warped,leftx, lefty, rightx, righty):
    ### Fit a second order polynomial to each with np.polyfit() ###
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)

    # Generate x and y values for plotting
    ploty = np.linspace(0, binary_warped.shape[0]-1,
binary_warped.shape[0] )
    try:
        left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
        right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
    except TypeError:
        # Avoids an error if `left` and `right_fit` are still none or incorrect
        print('The function failed to fit a line!')
        left_fitx = 1*ploty**2 + 1*ploty
        right_fitx = 1*ploty**2 + 1*ploty

    return left_fit, right_fit, left_fitx, right_fitx, ploty

def draw_poly_lines(binary_warped, left_fitx, right_fitx, ploty):
    # Create an image to draw on and an image to show the selection
window
    out_img = np.dstack((binary_warped, binary_warped,
binary_warped))*255
    window_img = np.zeros_like(out_img)

    margin = 100
    # Generate a polygon to illustrate the search window area
    # And recast the x and y points into usable format for cv2.fillPoly()
    left_line_window1 =
np.array([np.transpose(np.vstack([left_fitx-margin, ploty]))])
    left_line_window2 =
np.array([np.flipud(np.transpose(np.vstack([left_fitx+margin,
                              ploty])))])
    left_line_pts = np.hstack((left_line_window1, left_line_window2))
    right_line_window1 =
np.array([np.transpose(np.vstack([right_fitx-margin, ploty]))])
    right_line_window2 =
np.array([np.flipud(np.transpose(np.vstack([right_fitx+margin,
                              ploty])))])
    right_line_pts = np.hstack((right_line_window1, right_line_window2))
```

```
    # Draw the lane onto the warped blank image
    cv2.fillPoly(window_img, np.int_([left_line_pts]), (100, 100, 0))
    cv2.fillPoly(window_img, np.int_([right_line_pts]), (100, 100, 0))
    result = cv2.addWeighted(out_img, 1, window_img, 0.3, 0)

    # Plot the polynomial lines onto the image
    plt.plot(left_fitx, ploty, color='green')
    plt.plot(right_fitx, ploty, color='blue')
    ## End visualization steps ##
    return result

leftx, lefty, rightx, righty =
find_lane_pixels_using_histogram(binary_warped)
left_fit, right_fit, left_fitx, right_fitx, ploty = fit_poly(binary_warped,leftx,
lefty, rightx, righty)
print(left_fit)
out_img = draw_poly_lines(binary_warped, left_fitx, right_fitx, ploty)
plt.imshow(out_img)

prev_left_fit, prev_right_fit = left_fit, right_fit
print('prev left fit: ', prev_left_fit)
print('prev right fit: ', prev_right_fit)
```

```
[-4.37019313e-04  3.74146022e-02  5.00293260e+02]
prev left fit:  [-4.37019313e-04  3.74146022e-02  5.00293260e+02]
prev right fit:  [-1.80051966e-04 -1.72726757e-01  1.19010242e+03]
```

STEP 5: Detection of Lane Lines Based on Previous Step

```
#img = cv2.imread('test_images/test5.jpg')
#binary_thresh = binary_thresholded(img)
#binary_warped, M_inv = warp(binary_thresh)

# Polynomial fit values from the previous frame
# Make sure to grab the actual values from the previous step in your
project!
#left_fit = np.array([ 2.13935315e-04, -3.77507980e-01,
4.76902175e+02])
#right_fit = np.array([4.17622148e-04, -4.93848953e-01,
1.11806170e+03])

def find_lane_pixels_using_prev_poly(binary_warped):
    global prev_left_fit
    global prev_right_fit
    # width of the margin around the previous polynomial to search
    margin = 100
    # Grab activated pixels
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    ### Set the area of search based on activated x-values ###
    ### within the +/- margin of our polynomial function ###
    left_lane_inds = ((nonzerox > (prev_left_fit[0]*(nonzeroy**2) +
prev_left_fit[1]*nonzeroy +
                    prev_left_fit[2] - margin)) & (nonzerox <
(prev_left_fit[0]*(nonzeroy**2) +
                    prev_left_fit[1]*nonzeroy + prev_left_fit[2] +
margin))).nonzero()[0]
    right_lane_inds = ((nonzerox > (prev_right_fit[0]*(nonzeroy**2) +
prev_right_fit[1]*nonzeroy +
                    prev_right_fit[2] - margin)) & (nonzerox <
(prev_right_fit[0]*(nonzeroy**2) +
                    prev_right_fit[1]*nonzeroy + prev_right_fit[2] +
margin))).nonzero()[0]
    # Again, extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]
```

```
    return leftx, lefty, rightx, righty

leftx, lefty, rightx, righty =
find_lane_pixels_using_prev_poly(binary_warped)
left_fit, right_fit, left_fitx, right_fitx, ploty = fit_poly(binary_warped,leftx,
lefty, rightx, righty)
out_img = draw_poly_lines(binary_warped, left_fitx, right_fitx, ploty)
plt.imshow(out_img)
```

<matplotlib.image.AxesImage at 0x2d765250f10>

STEP 6: Calculate Vehicle Position and Curve Radius

```
def measure_curvature_meters(binary_warped, left_fitx, right_fitx,
ploty):
    # Define conversions in x and y from pixels space to meters
    ym_per_pix = 30/720 # meters per pixel in y dimension
    xm_per_pix = 3.7/700 # meters per pixel in x dimension

    left_fit_cr = np.polyfit(ploty*ym_per_pix, left_fitx*xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty*ym_per_pix, right_fitx*xm_per_pix, 2)
    # Define y-value where we want radius of curvature
    # We'll choose the maximum y-value, corresponding to the bottom of
the image
    y_eval = np.max(ploty)

    # Calculation of R_curve (radius of curvature)
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix +
left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
```

```
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])

    return left_curverad, right_curverad

def measure_position_meters(binary_warped, left_fit, right_fit):
    # Define conversion in x from pixels space to meters
    xm_per_pix = 3.7/700 # meters per pixel in x dimension
    # Choose the y value corresponding to the bottom of the image
    y_max = binary_warped.shape[0]
    # Calculate left and right line positions at the bottom of the image
    left_x_pos = left_fit[0]*y_max**2 + left_fit[1]*y_max + left_fit[2]
    right_x_pos = right_fit[0]*y_max**2 + right_fit[1]*y_max + right_fit[2]
    # Calculate the x position of the center of the lane
    center_lanes_x_pos = (left_x_pos + right_x_pos)//2
    # Calculate the deviation between the center of the lane and the center of the picture
    # The car is assumed to be placed in the center of the picture
    # If the deviation is negative, the car is on the felt hand side of the center of the lane
    veh_pos = ((binary_warped.shape[1]//2) - center_lanes_x_pos) * xm_per_pix
    return veh_pos

left_curverad, right_curverad = measure_curvature_meters(binary_warped, left_fitx, right_fitx, ploty)
print('left curve radius in meters  = ', left_curverad)
print('right curve radius in meters = ', right_curverad)
veh_pos = measure_position_meters(binary_warped, left_fit, right_fit)
print('vehicle position relative to center  = ', veh_pos)
```

```
left curve radius in meters  =  340.37514695323364
right curve radius in meters =  1164.9469563433286
vehicle position relative to center  =  0.04757142857142857
```

STEP 7: Project Lane Delimitations Back on Image Plane and Add Text for Lane Info

```
def project_lane_info(img, binary_warped, ploty, left_fitx, right_fitx, M_inv, left_curverad, right_curverad, veh_pos):
    # Create an image to draw the lines on
    warp_zero = np.zeros_like(binary_warped).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

    # Recast the x and y points into usable format for cv2.fillPoly()
    pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
    pts = np.hstack((pts_left, pts_right))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))

    # Warp the blank back to original image space using inverse perspective matrix (Minv)
    newwarp = cv2.warpPerspective(color_warp, M_inv, (img.shape[1], img.shape[0]))

    # Combine the result with the original image
    out_img = cv2.addWeighted(img, 1, newwarp, 0.3, 0)

    cv2.putText(out_img,'Curve Radius [m]: '+str((left_curverad+right_curverad)/2)[:7],(40,70), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.6, (255,255,255),2,cv2.LINE_AA)
    cv2.putText(out_img,'Center Offset [m]: '+str(veh_pos)[:7],(40,150), cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.6,(255,255,255),2,cv2.LINE_AA)

    return out_img

new_img = project_lane_info(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), binary_warped, ploty, left_fitx, right_fitx, M_inv, left_curverad, right_curverad, veh_pos)
```

```
# Plot the result
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))
f.tight_layout()
ax1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
ax1.set_title('Original Image', fontsize=20)
ax2.imshow(new_img, cmap='gray')
ax2.set_title('Image With Lane Marked', fontsize=20)
```

```
global left_fit_hist
left_fit_hist = np.array([])
#print(len(left_fit_hist))
global right_fit_hist
right_fit_hist = np.array([])
#left_fit = [1.3, 4.5, 6.7]
#left_fit_hist = np.array(left_fit)
#print(len(left_fit_hist))
#left_fit = [2.3, 5.5, 9.7]
#new_fit = np.array(left_fit)
#left_fit_hist = np.vstack([left_fit_hist, new_fit])
#left_fit = [8.3, 3.5, 21.7]
#new_fit = np.array(left_fit)
#left_fit_hist = np.vstack([left_fit_hist, new_fit])
#print(left_fit_hist)
#print(len(left_fit_hist))
#prev_left_fit = [np.mean(left_fit_hist[:,0]), np.mean(left_fit_hist[:,1]),
np.mean(left_fit_hist[:,2])]
#left_fit_hist = np.delete(left_fit_hist, 0,0)

#print(left_fit_hist)
#print(prev_left_fit)
```

STEP 8: Lane Finding Pipeline on Video

```
def lane_finding_pipeline(img):
    global left_fit_hist
    global right_fit_hist
    global prev_left_fit
    global prev_right_fit
    binary_thresh = binary_thresholded(img)
    binary_warped, M_inv = warp(binary_thresh)
    #out_img = np.dstack((binary_thresh, binary_thresh,
binary_thresh))*255
    if (len(left_fit_hist) == 0):
        leftx, lefty, rightx, righty =
find_lane_pixels_using_histogram(binary_warped)
        left_fit, right_fit, left_fitx, right_fitx, ploty =
fit_poly(binary_warped,leftx, lefty, rightx, righty)
        # Store fit in history
        left_fit_hist = np.array(left_fit)
        new_left_fit = np.array(left_fit)
        left_fit_hist = np.vstack([left_fit_hist, new_left_fit])
        right_fit_hist = np.array(right_fit)
        new_right_fit = np.array(right_fit)
        right_fit_hist = np.vstack([right_fit_hist, new_right_fit])
    else:
        prev_left_fit = [np.mean(left_fit_hist[:,0]), np.mean(left_fit_hist[:,1]),
np.mean(left_fit_hist[:,2])]
        prev_right_fit = [np.mean(right_fit_hist[:,0]),
np.mean(right_fit_hist[:,1]), np.mean(right_fit_hist[:,2])]
        leftx, lefty, rightx, righty =
find_lane_pixels_using_prev_poly(binary_warped)
        if (len(lefty) == 0 or len(righty) == 0):
            leftx, lefty, rightx, righty =
find_lane_pixels_using_histogram(binary_warped)
        left_fit, right_fit, left_fitx, right_fitx, ploty =
fit_poly(binary_warped,leftx, lefty, rightx, righty)

        # Add new values to history
        new_left_fit = np.array(left_fit)
        left_fit_hist = np.vstack([left_fit_hist, new_left_fit])
        new_right_fit = np.array(right_fit)
        right_fit_hist = np.vstack([right_fit_hist, new_right_fit])
```

```
    # Remove old values from history
    if (len(left_fit_hist) > 10):
        left_fit_hist = np.delete(left_fit_hist, 0,0)
        right_fit_hist = np.delete(right_fit_hist, 0,0)

    left_curverad, right_curverad =
measure_curvature_meters(binary_warped, left_fitx, right_fitx, ploty)
                                #measure_curvature_meters(binary_warped, left_fitx,
right_fitx, ploty)
    veh_pos = measure_position_meters(binary_warped, left_fit, right_fit)
    out_img = project_lane_info(img, binary_warped, ploty, left_fitx,
right_fitx, M_inv, left_curverad, right_curverad, veh_pos)
    return out_img

video_output = 'OneDrive/Desktop/Study/SAE/Advance lane
detection/Advanced-Lane-Detection-master/Advanced-Lane-Detection-
master/project_video_output.mp4'
clip1 = VideoFileClip("OneDrive/Desktop/Study/SAE/Advance lane
detection/Advanced-Lane-Detection-master/Advanced-Lane-Detection-
master/project_video.mp4")
output_clip = clip1.fl_image(lane_finding_pipeline)
%time output_clip.write_videofile(video_output, audio=False)
output_clip.ipython_display(width=960)
```

- STEPS:

1. First import all the required modules.
2. Camera Calibration.
3. Perspective Transform from Car Camera to Bird's Eye View.
4. Process Binary Thresholded Images
5. Detection of Lane Lines Using Histogram.
6. Detection of Lane Lines Based on Previous Step
7. Calculate Vehicle Position and Curve Radius.
8. Project Lane Delimitations Back on Image Plane and Add Text for Lane Info.
9. Lane Finding Pipeline on Video on the project_video.
10. Lane Finding Pipeline on Video on the challenge_video.
11. Display the output_clip videos and images.

4. Inference:

- In this project, I have used Python and OpenCV image processing libraries. Computer Vision algorithm to compute road curvature and lane vehicle offset using OpenCV Image Processing, Camera Calibration, Perspective Transform, Color Masks, Sobels and Polynomial Fit.
- In addition, to correctly compute the curvature, the lane lines need to be identified but on top of that, the images need to be undistorted. Image transformation is necessary for camera calibration and for perspective transform to obtain a bird's eye view of the road.
- For more accurate Advance lane detection we need more and more images to process the camera calibration and bird eye view to detect the exact curvature for the video frame.

5. Result:

- Resulting video has been produced which is available in the Jupyter Advance Lane Detection File.

6. Reference:

- https://github.com/OanaGaskey/Advanced-Lane-Detection
- https://towardsdatascience.com/advanced-lane-detection-for-autonomous-vehicles-using-computer-vision-techniques-f229e4245e41
- https://kushalbkusram.medium.com/advanced-lane-detection-fd39572cfe91
- https://medium.com/@NickHortovanyi/advanced-lane-detection-8b98b79b9cac
- https://www.linkedin.com/pulse/chapter-2-self-driving-car-advanced-lane-finding-theory-snoussi/
- https://github.com/mohamedameen93/Advanced-Lane-Finding-Using-OpenCV