PROJECT2
ADVANCED LANE FINDING

The goal of this assignment is to find efficient lane lines present on the roads through camera images.

- Firstly to calibrate the Camera matrix to get the undistorted camera images from Chessboard images.
- The next step is to create a pipeline to find the lanes on an undistorted image through various computer vision techniques.
- The steps include, applying a distortion correction to every image.
- Thresholding the image through color transforms and edge detection techniques.
- Applying a perspective transform to get binary images of only the lanes
- Searching for the binary lines through hough transform and sliding window techniques.
- Determining the curvature and offset from the center of the road.
- Warp the detected boundaries on the original image.
- The final image should display the lane area and the radius of curvature and the offset.
- A similar pipeline has to be then applied to the videos.
- NOTE: The images are stored in the output_images folder. (Readme saved in the folder.)

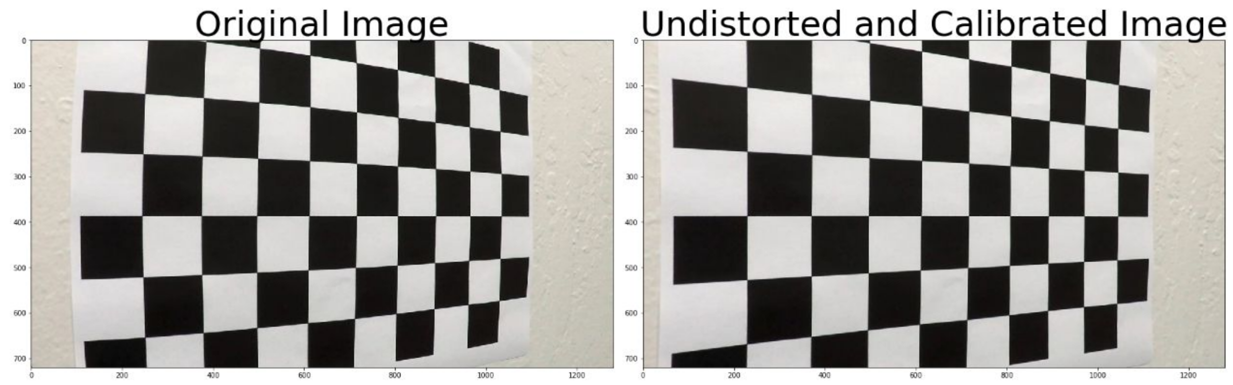
STEP 1 : CAMERA CALIBRATION

Images taken from cameras are often distorted in the process of converting it from 3 D real world plane to the 2 D image plane. It means the object might look small or big and shifted from its original size and place respectively. In case of detecting lanes on an undistorted camera image, we might predict the incorrect steering angle which will give wrong results. Thus, images need to be calibrated before lane line prediction. Camera calibration is a process of estimating the camera parameters. Here, we use multiple Chessboard images to get the camera matrix, rotation and translation vectors and the distortion coefficients.

Function: Calibrate_fun()

The function requires test images which convert them into grayscale and finds the corners of each image in real world space and images space. These points when fed to **cv2.calibrateCamera** return the camera matrix and distortion coefficients.

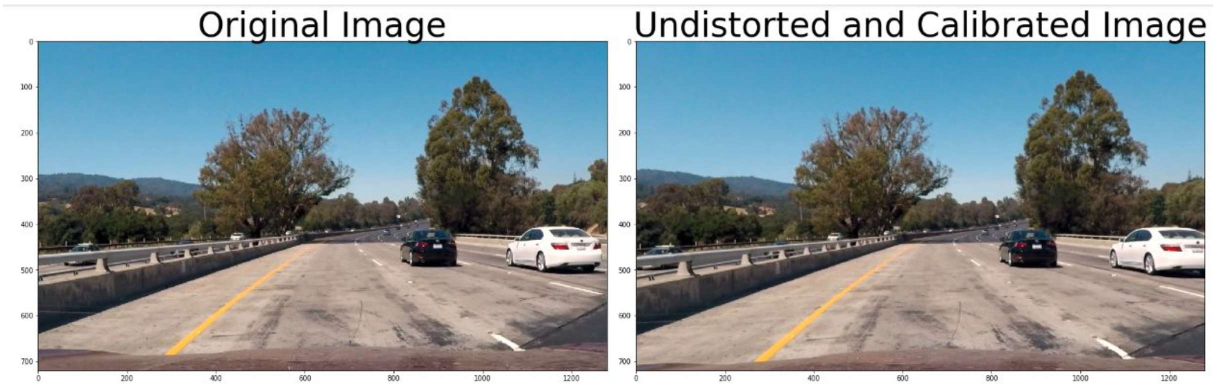
Result of Camera Calibration:



Note :Other undistorted camera images are stored in the folder ouptut_images→Calibration_result.

STEP 2: PIPELINE (TEST IMAGES)

- Distortion Correction to test images



The distortion correction can be observed at the lower left corner of the image. Undistorted test images are stored in output_images → Distortion_Correction

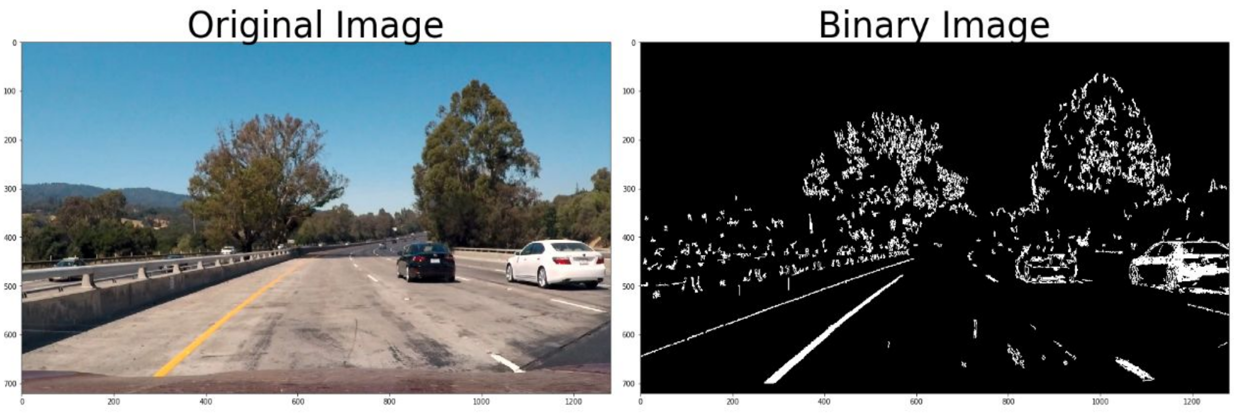
- Generating a threshold image

Function: image_transform()

The undistorted image is converted into HLS color space and the S channel is separated, which clearly distincts the yellow lane, and added with the filtered output for vertical edge detection.

A thresholding applied on both the images results in the binary image.

The output contains some noise in form of small blobs. Finding these blobs using connected components and removing blobs lesser than a min_size results in reduction of noise.

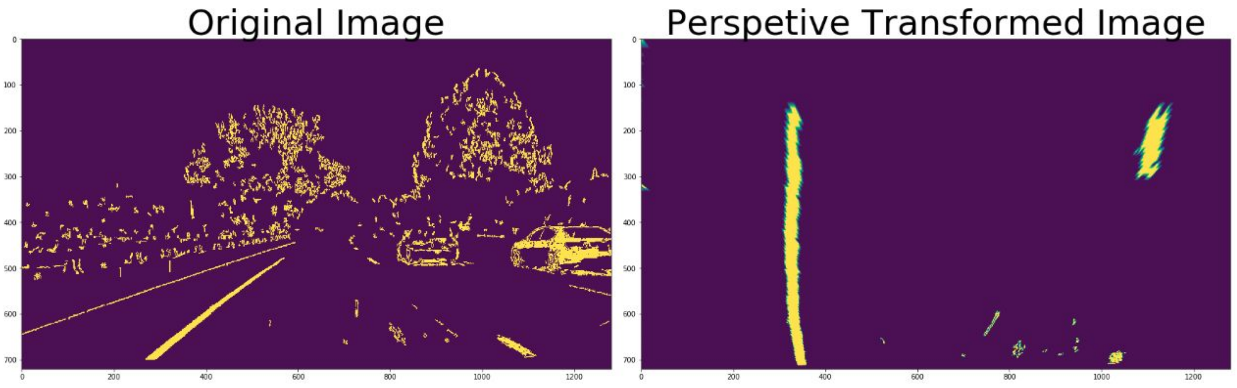


Threshold test images are stored in output_images → Lane_lines_Images

● Perspective transform

To calculate the lane curvature from a bird's eye view point a perspective transform is needed. The lane might look converging farther away in non-perspective image.

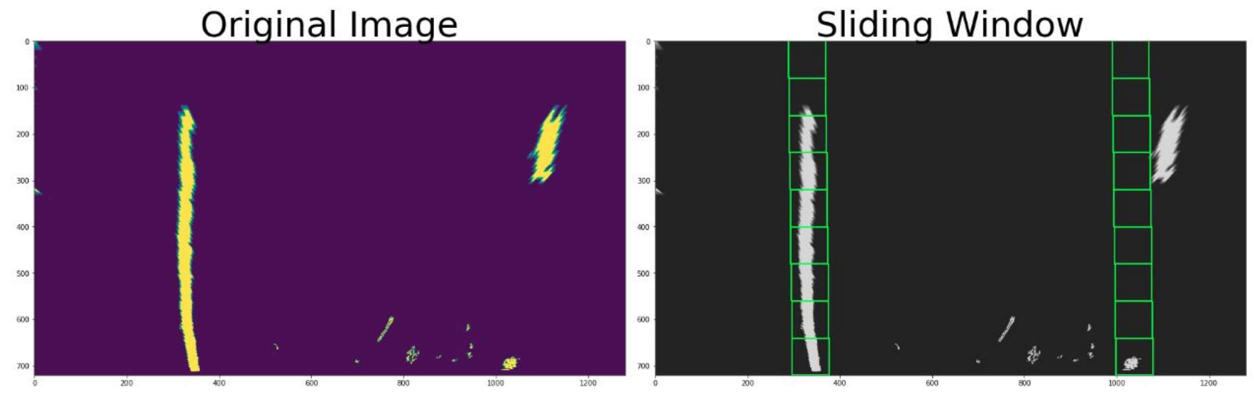
Function : perspective_func()



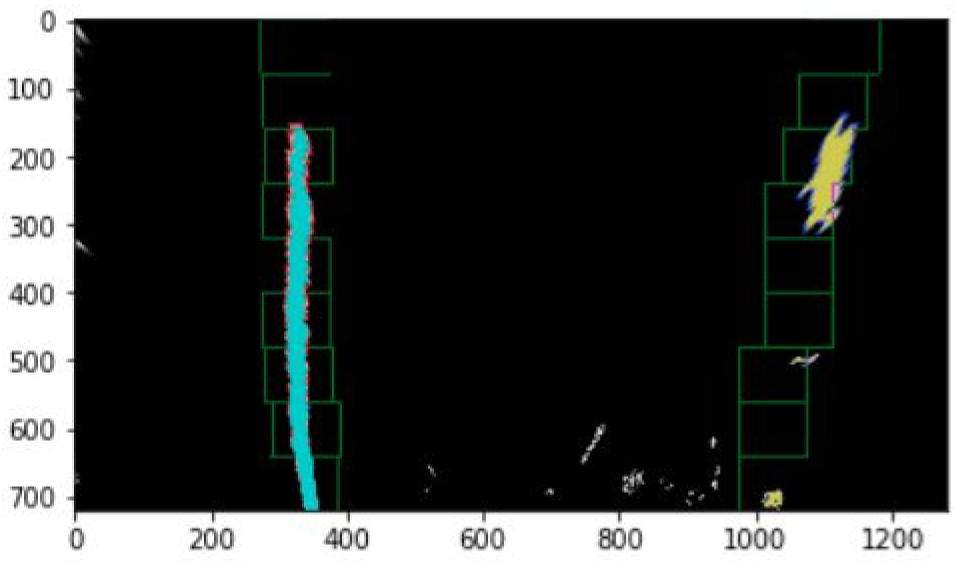
● Finding the Lane Line pixels

Function :find_lane_pixels()

Using hough transforms and sliding window technique to detect the lane line pixels. A problem arise when number of non zero pixels in a window is less than the min_size then the window position does not changes very significantly. (as shown in the diagram). Therefore, if the lane line breaks or their are no pixels in the sliding window, we search for pixels sliding in horizontal direction, keeping the sliding size= 25. In any case, if we find pixels value greater than the sliding window is then replaced.



The result of updating the algorithm :



Measuring the Radius of Curvature and the Offset:

$$R_{cur} = (1 + (2Ay + B)^2\,)^{3/2}/|2A|$$

The radius of curvature at any point gives the radius of the approximating circle. To get the precise result we computed the Radius of Curvature in metres by scaling the x and y coordinate as,

ym_per_pix = 30/720

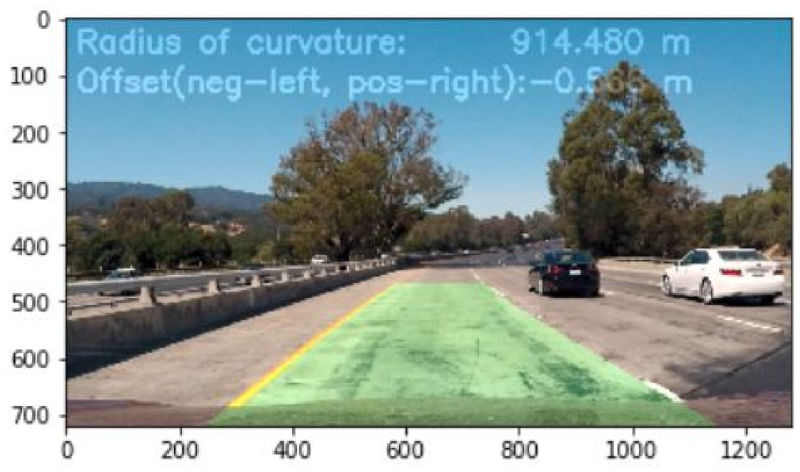
xm_per_pix = 3.7/700

Function: measure_curvature_pixels()

The offset, i.e the deviation from the center of the lane was calculated as the, difference in the average points on the lane lines (measured at the bottom of the image from left and right polynomial) and the center of the lane.

Function : measure_offset()

Result on Images: (Pipeline for test images)


Radius of curvature: 914.480 m
Offset(neg-left, pos-right):-0.5 m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

The image shows the result on test images.

Videos are stored in the output_images folder.

Function: func_videos()

- ## Challenges:

1. We have used the code for thresholding which detects the yellow line and the white line pavements. If the pavements are of any other colour, broken or not defined in the area it would be difficult to recognize.
2. The source points and the destination points used to calculate the warped images are hardcoded.

3. The algorithm might fail to work on night images or images on a rainy day, since the thresholding has been performed keeping daylight images.
4. The algorithm does not performs well on the challenge and harder challenge videos.