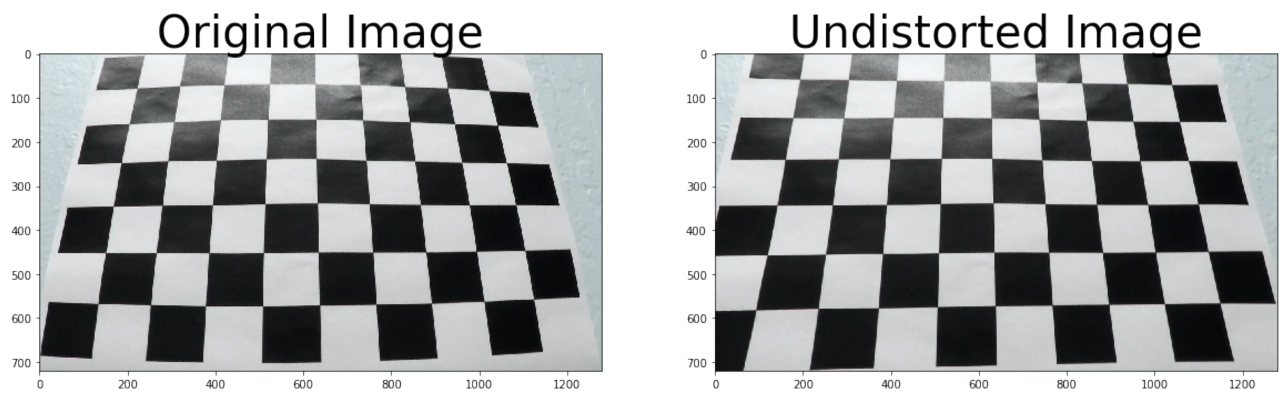
Advanced Lane Finding Project

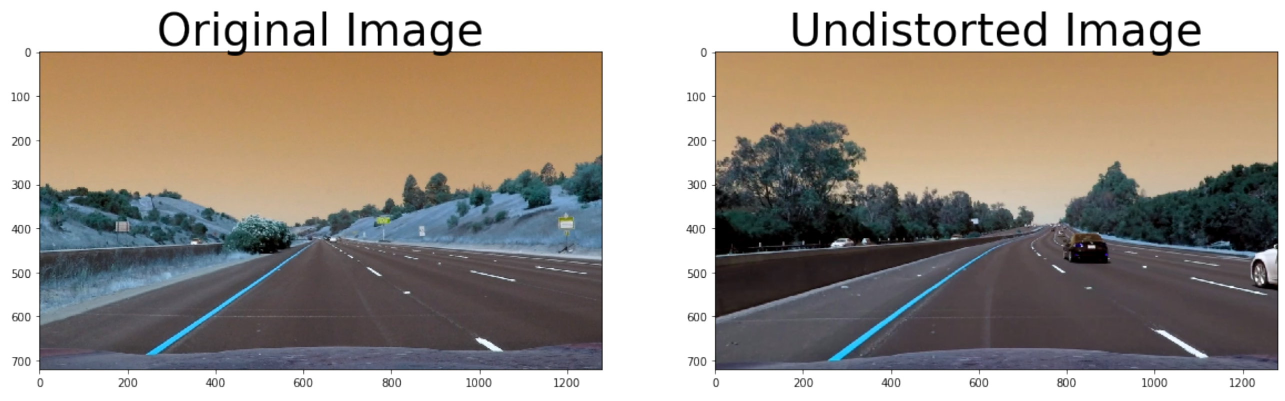
1. Camera Calibration

Code for camera calibration is found in the first cell of the advance_lane.ipynb
Chessboard images taken at different angles are used for camera calibration. Object points are used to represent the chessboard corners in an undistorted image and image points are used to represent chessboard corners found in the image taken from the camera(ie.distorted image).
Cv2.calibrateCamera() is used to find the camera matrix , distortion coefficients. Cv2.undistort() is used to undistort the image.
In the picture image on the left is the original image and that on the right is the undistort image after applying cv2.undistort() function.



2. Undistortion Example

Here in this picture you can see the undistorted image on the right.



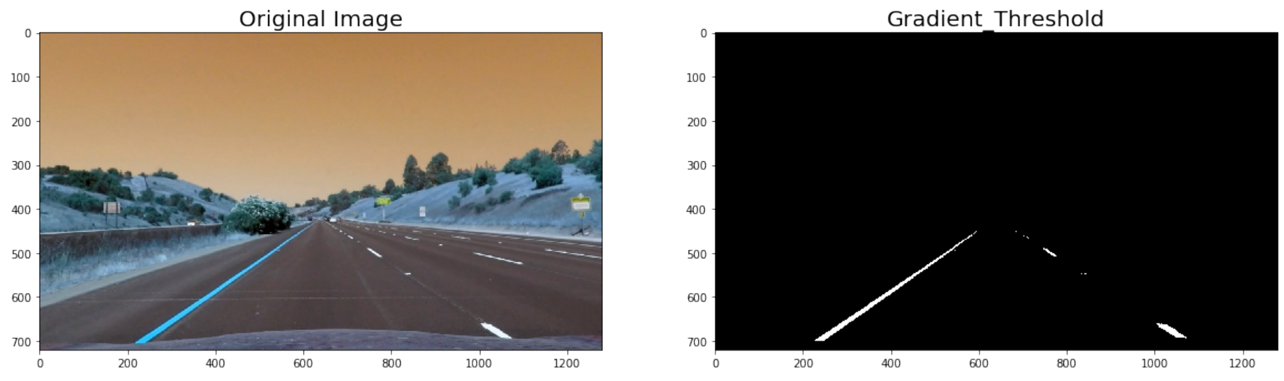
3. Lane Detection.

1. First red channel and green channel images are thresholded to detect the yellow and white lane on the road. But this can't detect lanes when the road color is other than black.
2. Then the image is converted into gray scale and sobelx and direction threshold is applied to detect the lane lines.But this can't detect lanes in the shadowed region.
3. The image is converted to HLS colorspace and Light and Saturate thresholds are used to detect the lanes in the shadowed region.

Then red,green and Light are combined together (red & green & light) ; sobel,direction and saturate are combined together (sobelx & direction | saturate).

Finallly (red & green & light) & (sobelx & direction | saturate) will detect lane lines smoothly.

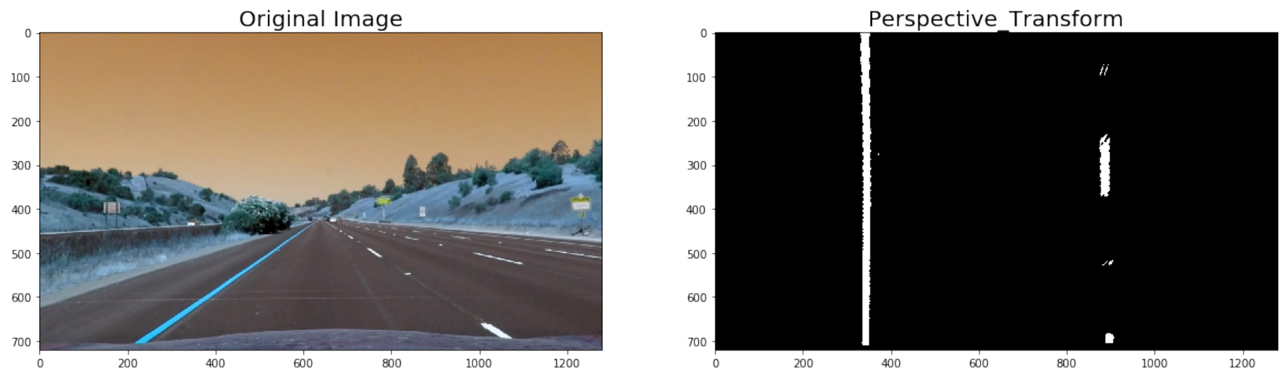
The code for this is in cell 3 of advance_lane.ipynb.



4. Perspective Transform.

To get a bird eye’s perspective cv2.getPerspectiveTransform() is used. Source points are choosen from the image where the lane lines are straight and destination points are choosen such that the left and right lane lines are parallel in the bird eye’s perspective.

Source points	Destination points
220,720	350,720
570,470	350,1
722,470	900,1
1110,720	900,720

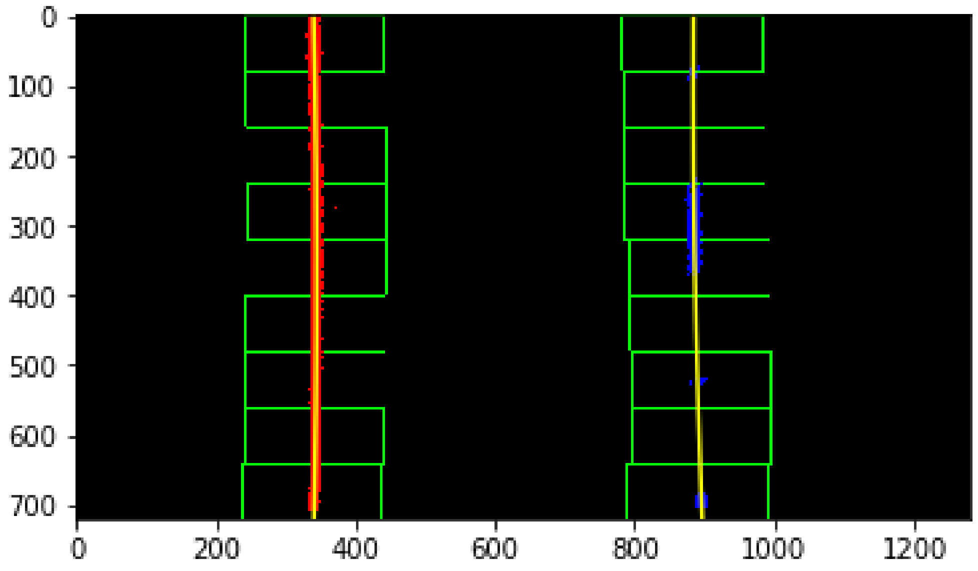


code for this is in cell 4 of advance_lane.ipynb.

5. Polynomial Fitting:

Left and Right lane indices in the images are identified by using sliding window algorithm, where for each image small rectangular windows are iterated throughout the image height .
After finding the lane indices then np.polyfit() is used to fit a polynomial of degree 2.

Here in this picture the yellow lines represents the fitted polynomial and the green boxes indicate the rectangular windows.

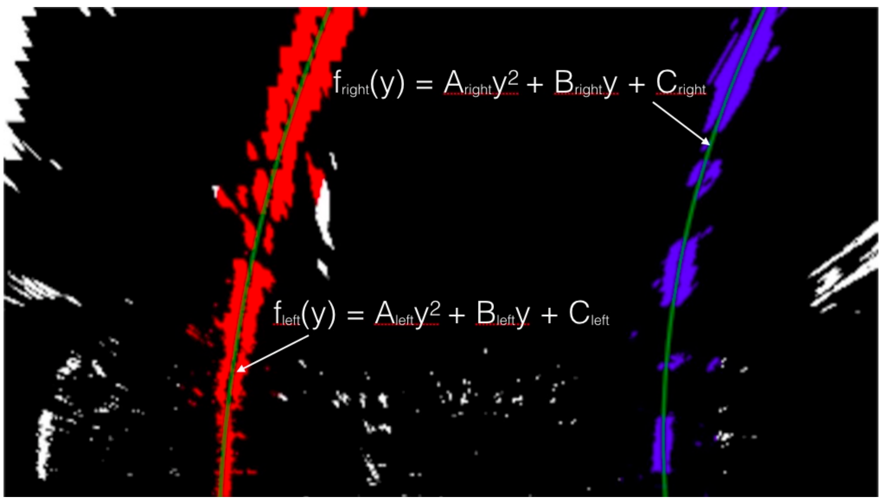


6. Radius of curvature and vehicle offset:

For radius of curvature first pixel to meter conversion of the lane indices is done and then a second degree polnomial is fitted.

ym_per_pix = 30/720 (meters per pixel in y dimension)
xm_per_pix = 3.7/700 (meters per pixel in x dimension)

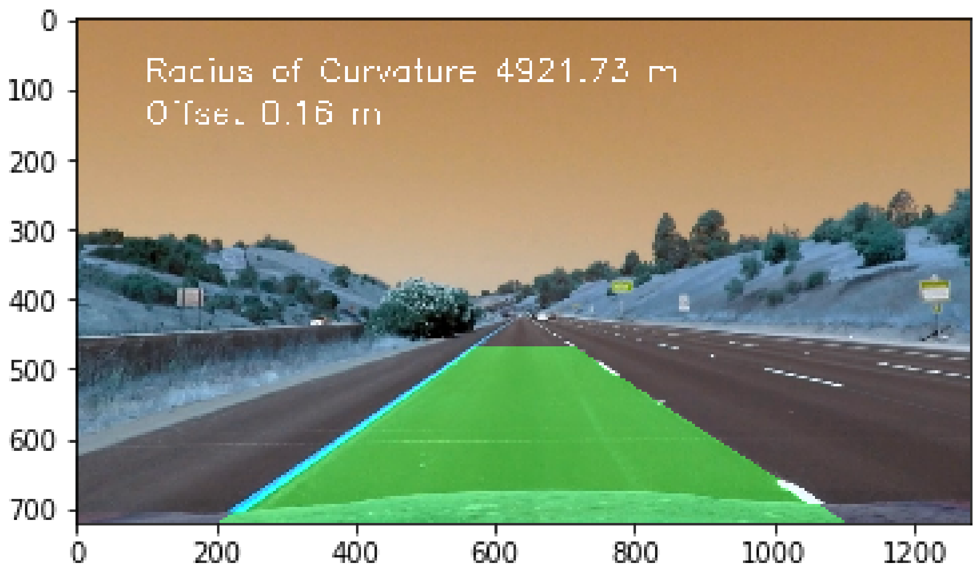
Rcurve= ((1+(2Ay+B)**2)**3/2) /|2A|


$f_{right}(y) = A_{right}y^2 + B_{right}y + C_{right}$
$f_{left}(y) = A_{left}y^2 + B_{left}y + C_{left}$

vehicle Offset is calculated by (image_center – lane_center)*(pixel to meter conversion)

7. Unwarp Image.

The area inside the lane lines are colored green using cv2.fillpoly() and the image is transformed to the original perspective using mxt_inv (matrix calculated with source and destination points swaped).



8. Video testing.

The above code tested on the video and the video file is also attached.

9. Challenges Faced.

1. In the detection of lane lines using gradient thresholds and colorspaces i felt difficult to find the correct combination of various thresholds.
2. Source and destination points for the perspective transform works only when the road is flat and the camera's perspective doesn't changes. So it would be better if the source and destination points are calculated dynamically.
3. It would be difficult if the lane lines are faded and discontinuous for a longer distance.