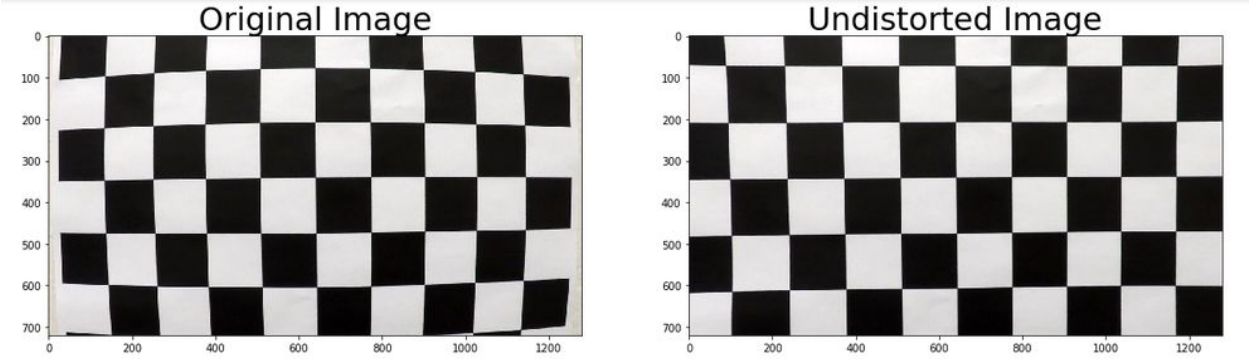
Camera Calibration

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

Procedure:

1. Prepare objectpoints array based on number of corners present in chessboard
2. Convert distorted input image to grayscale
3. Find corners using opencv function *cv2.findChessboardCorners*
4. Append corners and corresponding objectpoints to arrays
5. Find Camera Matrix and Distortion Coefficients to be used for undistortion using *cv2.calibrateCamera*
6. Undistort input image using *cv2.undistort* and values found previously

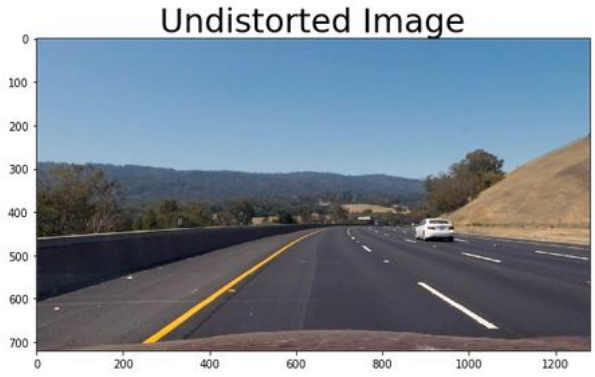
Result:



Pipeline (single images)

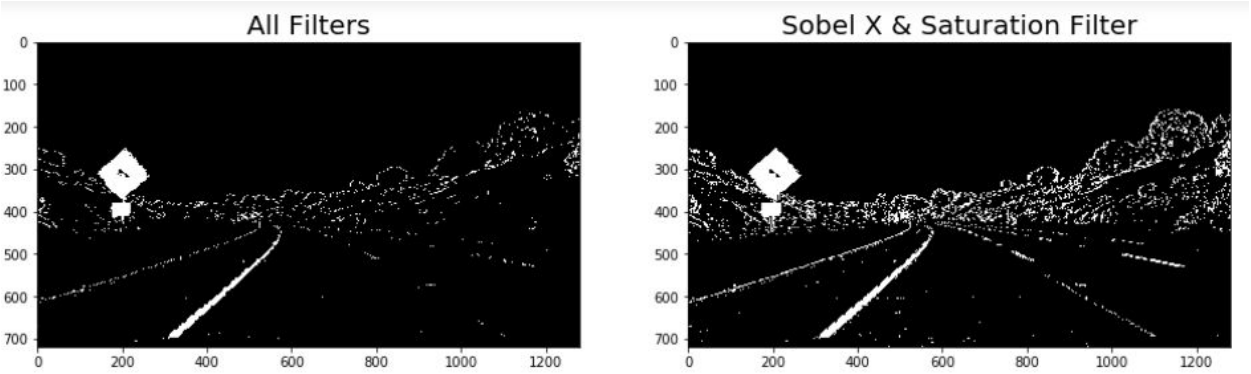
1. Provide an example of a distortion-corrected image.

Undistorted image was obtained using the camera calibration and distortion matrices found previously and applying the same function *cv2.undistort*



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.

1. The transforms applied were Sobel (in ‘x’ axis), Magnitude (using Sobel transform in both ‘x’ and ‘y’ axes and taking their magnitude) and saturation filter (where the image was converted to ‘hls’ space and the ‘s’ channel values that fit the threshold were passed)



2. The above comparison shows the advantage of including the magnitude filter. It helps in considerably reducing values of objects that are not lane lines.

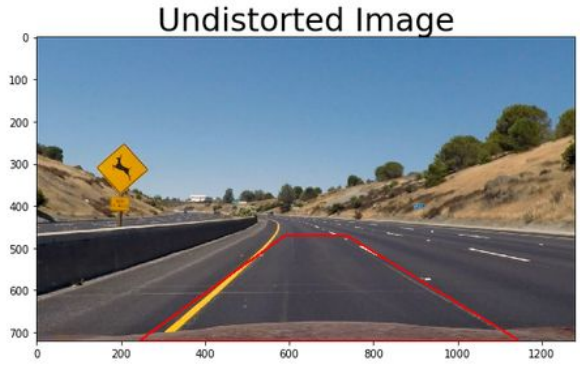
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

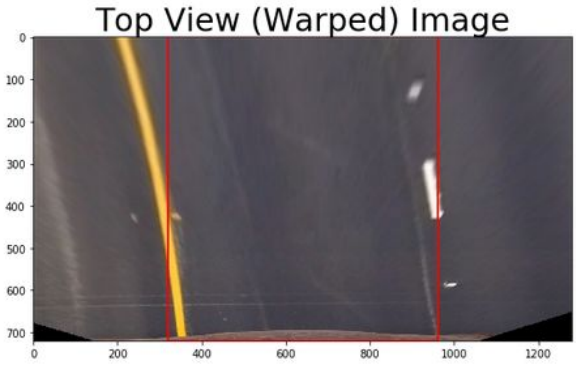
1. Source points from input image were chosen and destination points were selected to map those source points at destination points in warped output image

2. Perspective conversion matrix was obtained from function *cv2.getPerspectiveTransform* based on source and destination points

3. *cv2.warpPerspective* was used to map the source points to destination points and get the required image

Source	Destination
590, 470	320, 0
245, 719	320, 720
1145, 719	960, 720
735, 470	960, 0





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

1. The input image was passed through the pipeline previously described to get a warped image (bird's eye view)

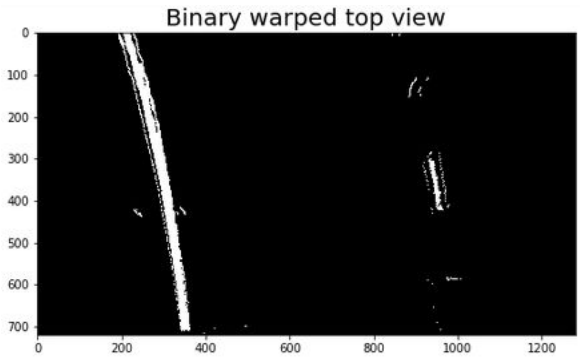
2. Nonzero points for the bottom half of the warped image and for the x-axis were identified considering values and plotting them using a histogram

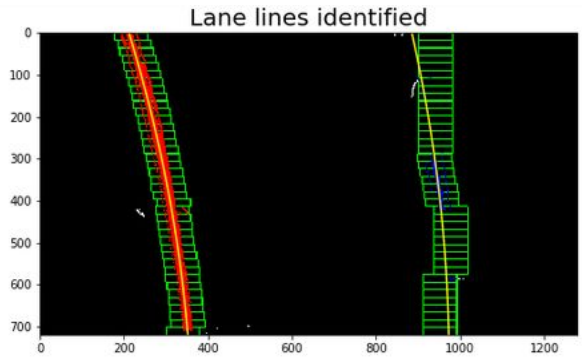
3. The histogram had peaks on either side of the midpoint of the image. These peaks corresponded to left and right lane point clusters.

4. These x-axis values were considered as starting points for windows (indicated by green rectangle) which were used to find out which of the nonzero points were actually the lane lines.

5. The windows successively moved over the image based on a decided margin identifying left and right lanes

6. All the points that were classified as lane points were curve fitted to determine the lane profile (yellow curve)





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 center. **and** 6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

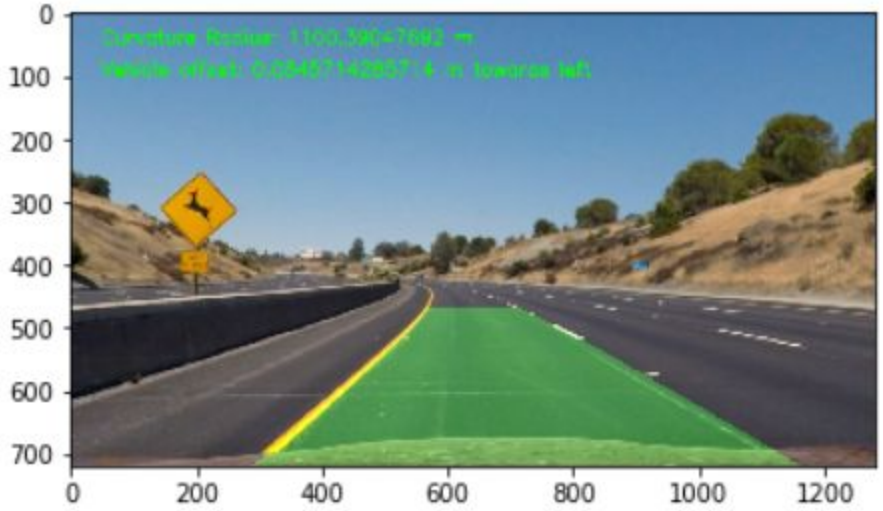
1. Pixel to metre conversion values were used to curve fit the identified lane line values to metres.

2. The curve fit values were passed through the formula provided in Udacity's lectures to calculate the radius of the curve in metres

3. For the vehicle offset with respect to the lane, it was given that the camera was mounted in the center of the vehicle. Hence the half image length gave the camera position and the position of the centre of the car

4. Through the curve fitting procedure the location of left and right lanes on the images was known.

5. The value middle of the lane was compared with 640 (half the image size). Based on that the offset direction and value was found out.



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!).

Present in submitted Jupyter Notebook directory

Also uploaded to github-
https://github.com/saurabh-sk/Udacity-SDC-ND/blob/master/02%20Advanced%20Lane%20Finding/Writeup.pdf

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?

Problems faced:

1. Tuning values for filters and src

identified failures:

1. Sharp corners
2. Multiple and subtle gradients
3. In presence of other cars (as current video shows)

Steps to make it more robust:

1. Better lane identification by implementing more filters