Advanced Lane Finding:

The project aims at finding left and right lanes on road and find the curvature and the offset of lane center from the image center.

Below are the steps used in the pipeline for finding lanes. Following that is a detailed explanation of the pipeline.

1. Calibrate Camera

2. Undistort the input image

3. Warp the undistorted image

4. Transform warped image to grayscale

5. Apply Gradient threshold on the gray image and obtain a binary image

6. Apply HLS transform and get binary image by applying color thresholds

7. Combine the above two binaries

8. Find left and right lanes on the image and highlight them

9. Unwarp the image obtained in step 8

10. Place the highlighted area of the image from step 9 onto the original image.

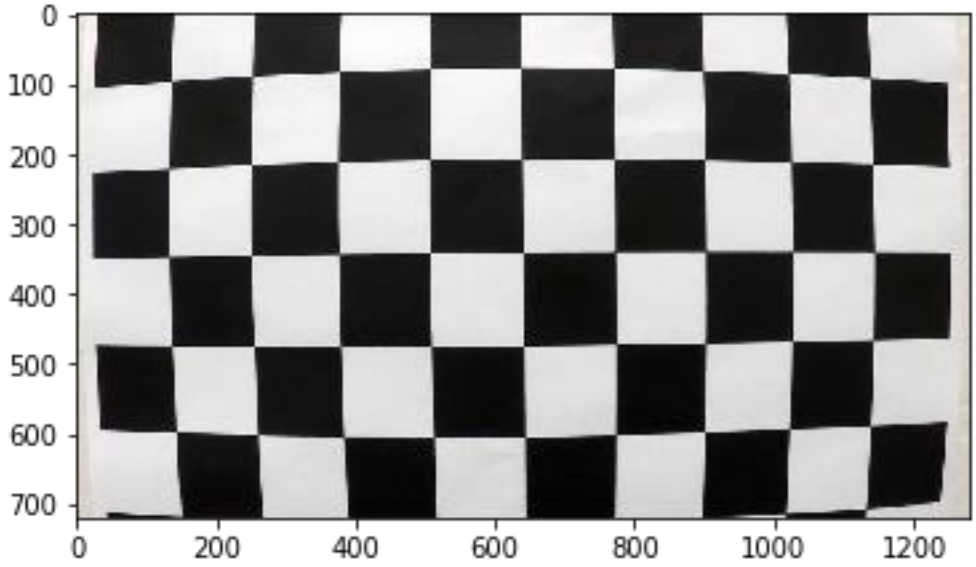
Camera Calibration:

The camera calibration is performed in the pipeline using the multiple 9X6 chessboard images. These images are in the folder “camera_cal/”.

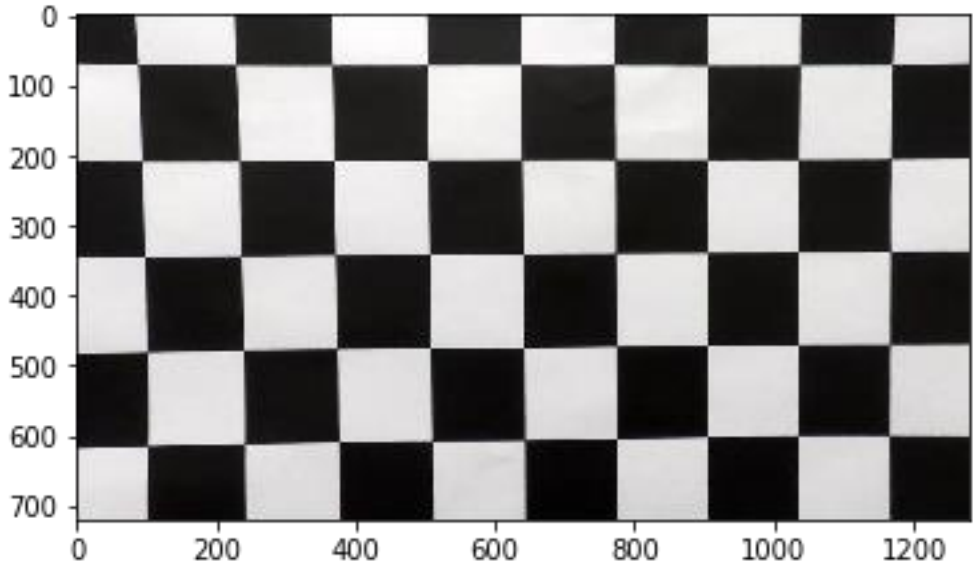
This camera caliberation is performed in the Jupyter notebook “P2.ipynb”. First input image is converted to Gray Scale and Object Points are generated in (x, y, z) coordinates with z = 0 always. Image points were generated using the OpenCV.

OpenCV functions “cv2.findChessboardCorners()”, “cv2.calibrateCamera()” are used to generate Calibration Matrix and Distortion Coefficients. “cv2.undistort()” function is used to generate an undistorted image.

The below is the actual image.



The below is the undistorted image of the above image.



Pipeline (Test Images):

The pipeline is demonstrated below using one of the test images. The test images are in the folder "test_images/". The image used is "straight_lines1.jpg". All the images in the test folder are processed using the pipeline and written into the folder **"output_images/"**.

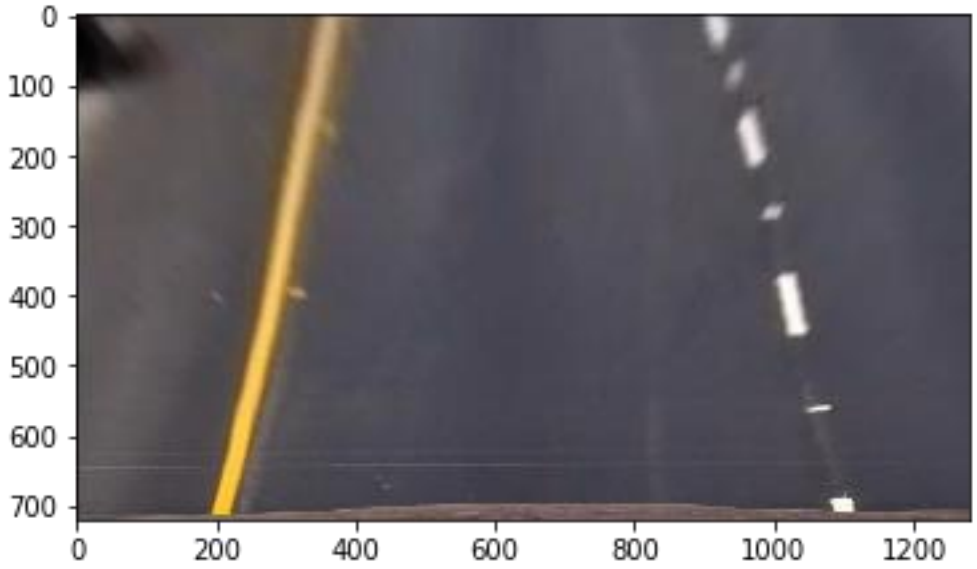
1. **Caliberate Camera:** The above section explains the camera calibration procedure.

2. **Undistorted Image:** The below is the actual image.

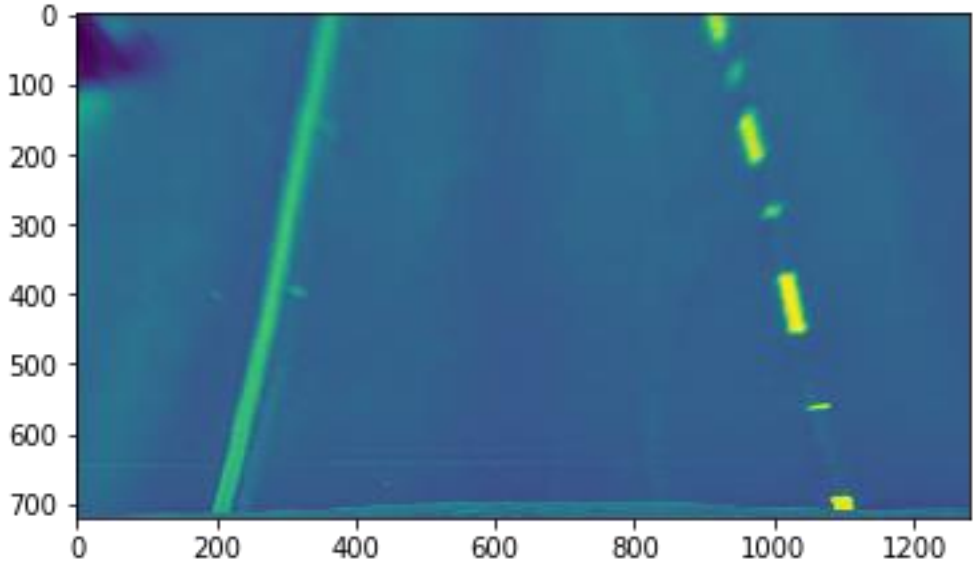
The below is the undistorted image using the calibration matrix distortion coefficients obtained by calibrating the camera.

3. **<u>Warping the Image:</u>** First source and destination points defining rectangles are defined for warping the image. Using the OpenCV functions “cv2.getPerspectiveTransform()” on the source and destinations points the parameters for Warping and Unwarping the image are calculated.

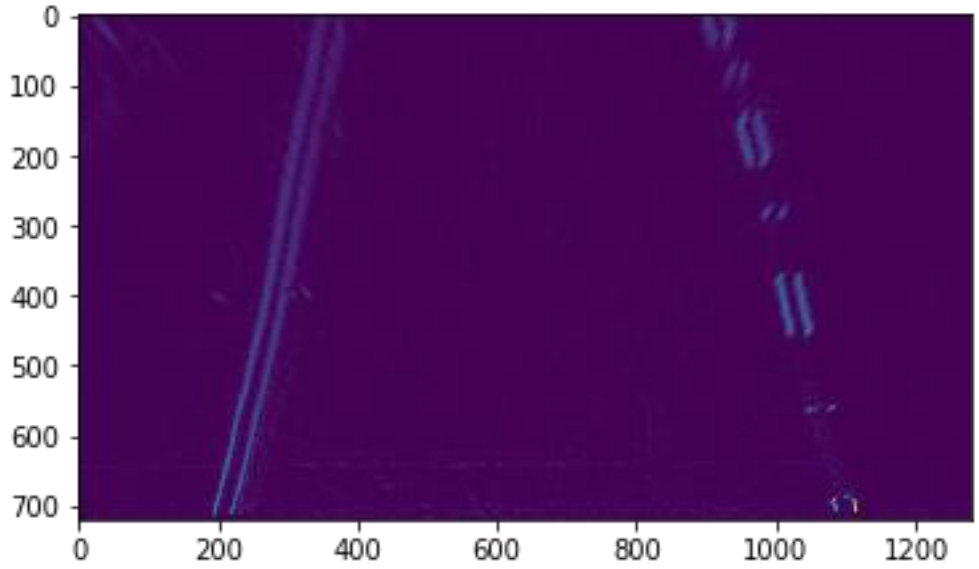
   Below is the warped image for the undistorted image shown above.



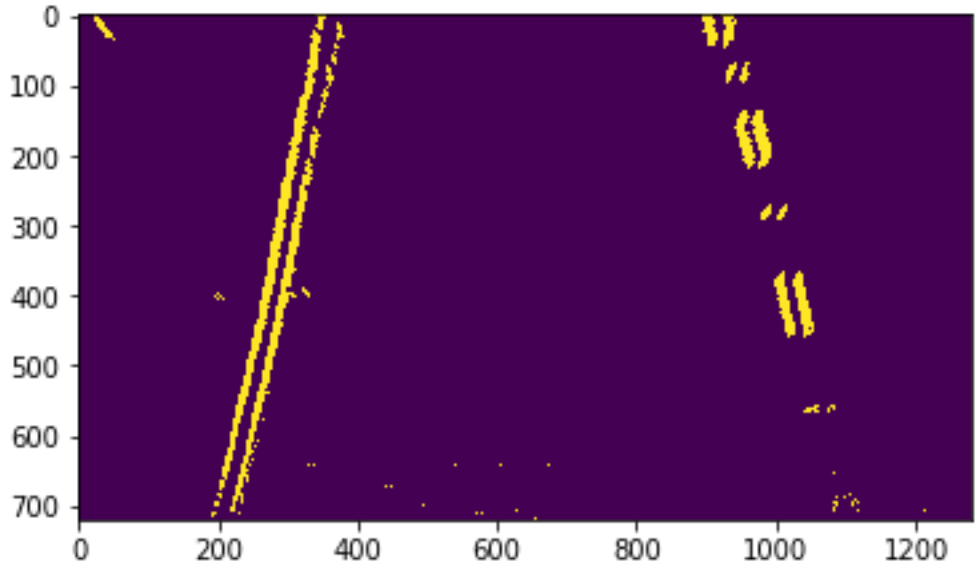
4. **Warped Image to Grayscale:** The above warped "BGR" image is transformed to grayscale using the OpenCV function "cv2.cvtColor()". Below is the grayscale image.



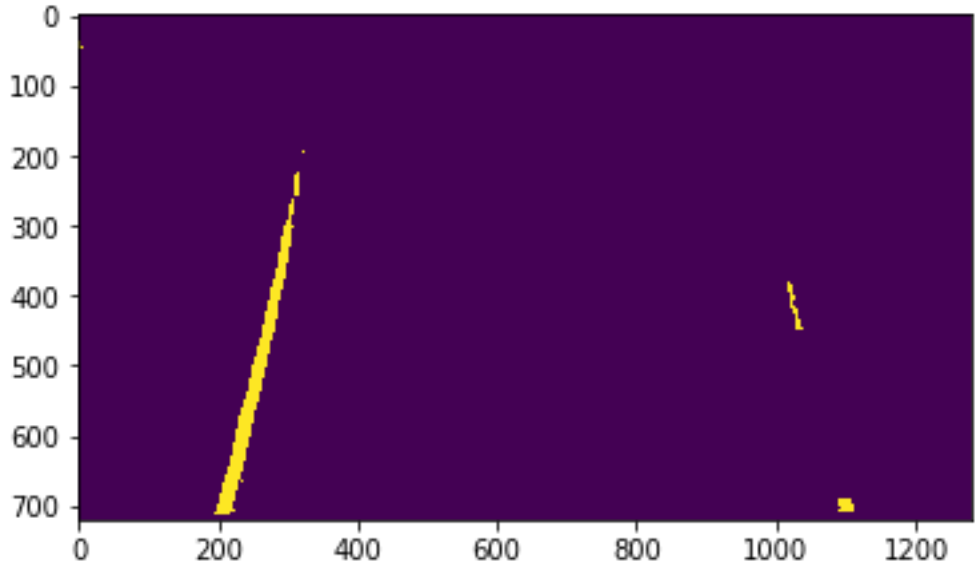
5. **Gradient Threshold on Gray Image:** On the above Grayscale image a Sobel x Operator of Kernel Size 31 is applied. Below is the result of applying Sobel Operator in x direction. (OpenCV function "cv2.Sobel()" is used here).



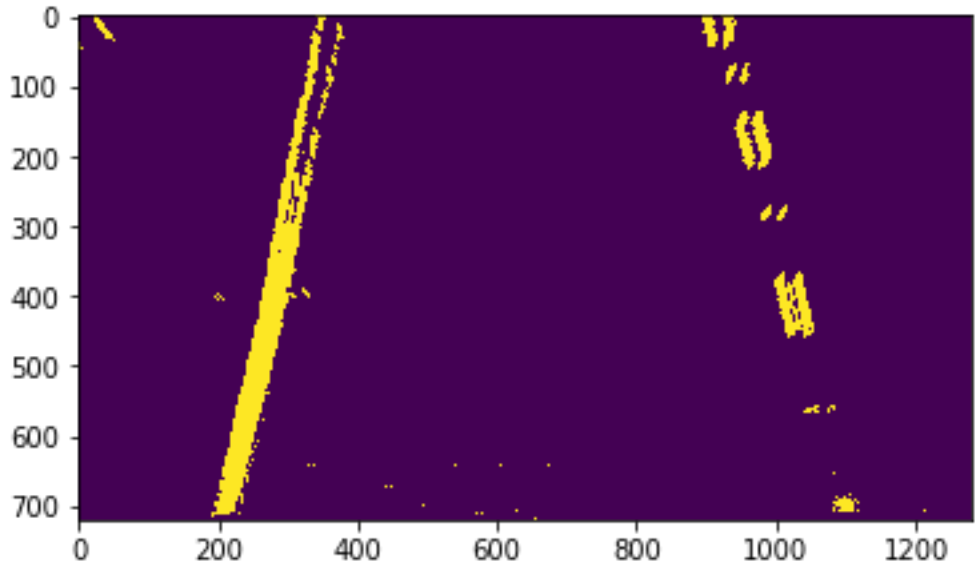
The above image is converted to a binary image by applying a threshold of (20, 100) to get the image to show only the lane lines. Below is the image.



6. **HLS Transform and Color Threshold:** On undistorted image show in one of the above images, HLS transform is applied using the OpenCV function “cv2.cvtColor()”. On the “S” channel of resultant image a color threshold of (170, 255) is applied and converted to a binary image. Below is the resultant image.

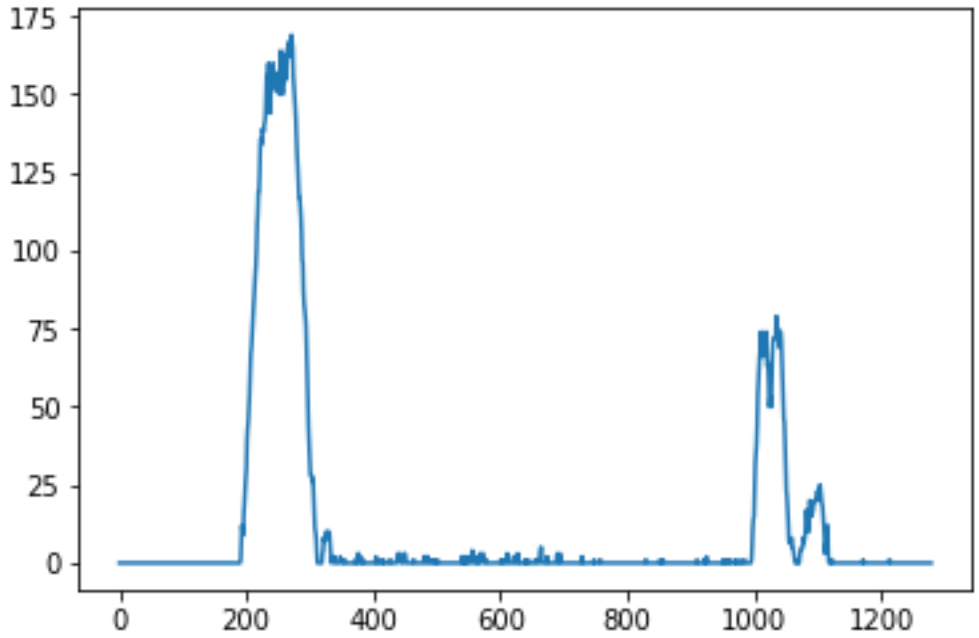


7. **<u>Combine the two binary images:</u>** The above two binary images are combined to have a better lane lines marked. The resultant image is as follows.



8. **<u>Find Left and Right Lanes and Highlight them:</u>** On the above combined binary images multiple processes are applied as explained below.

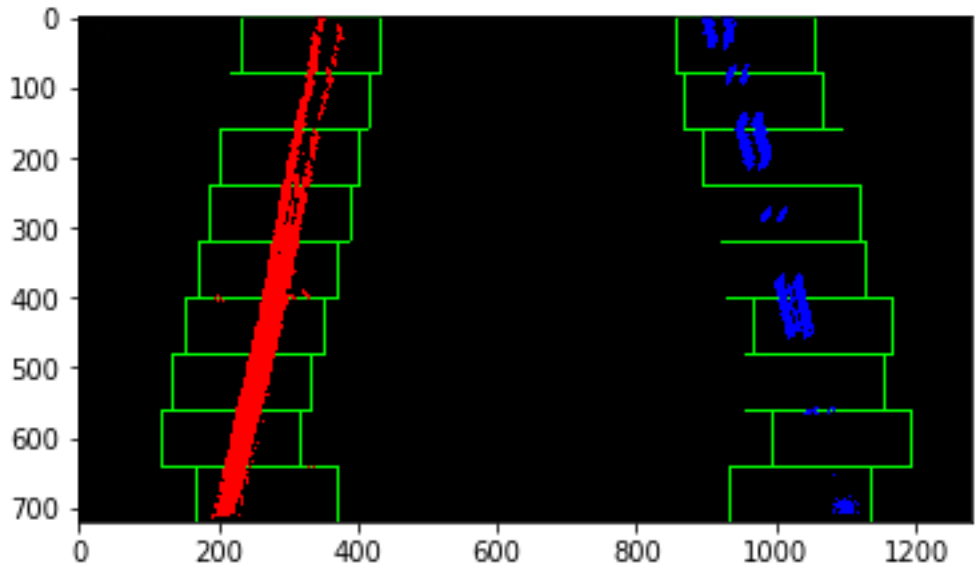
   Histogram is calculated on the bottom half of above binary image. The peaks on the histogram show that the lane lines are present as can be seen in the below image.



Sliding Window technique is used to obtain the regions where lane lines can be found. If the pipeline is applied on a single image, the entire sliding window techniques is performed.

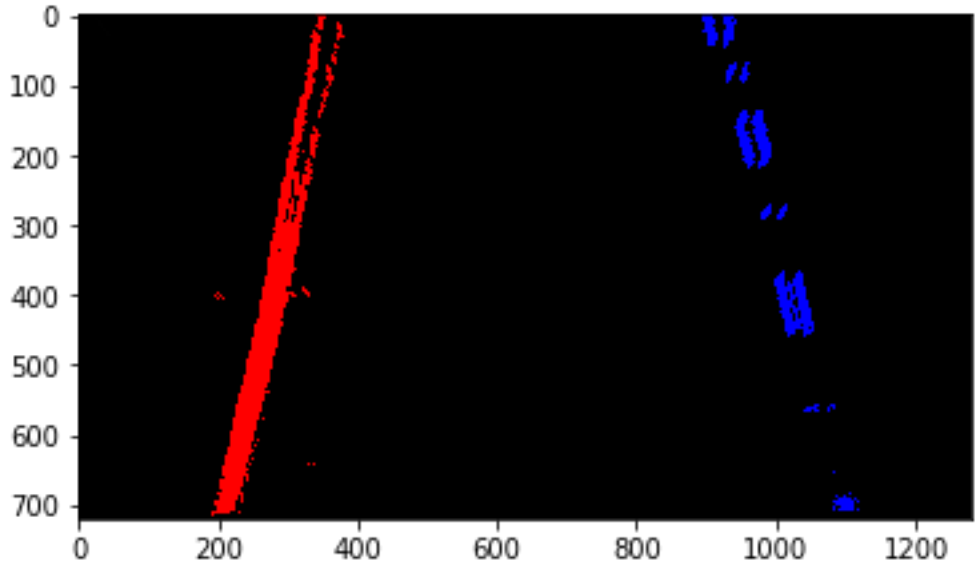
Otherwise if it is applied on a video, this technique is applied only on the region of interest that can be calculated from the previous frames.

The below image shows the windows highlighted.

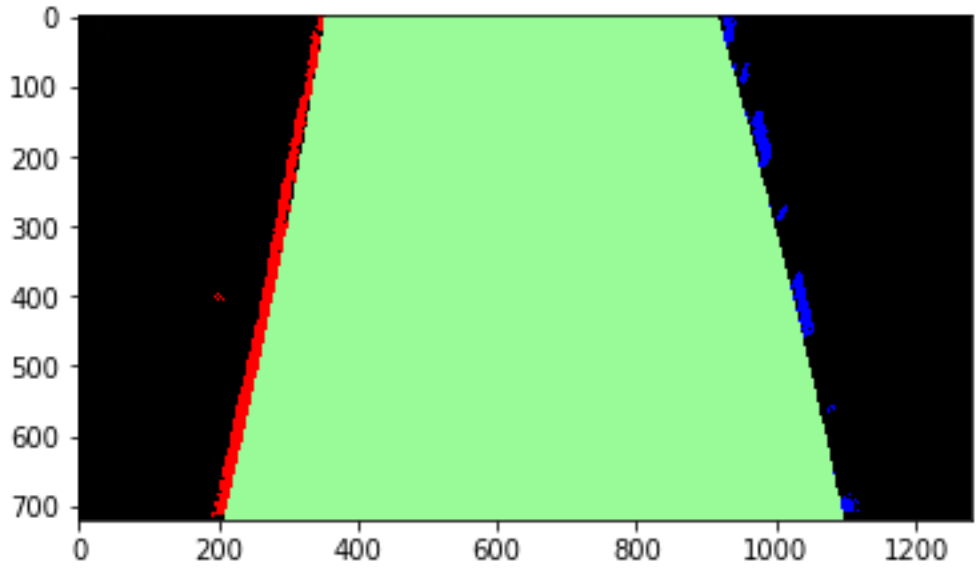


Thus, from the above image the pixels constituting the left and right lanes are taken. A second order polynomial coefficients are calculated to the points using the numpy function “np.polyfit()”. These coefficients are stored in a “collections.deque()” with a

maximum length of 10. Which means that the coefficients for the last 10 frames are stored. They are averaged to smoothen the output image highlighting the lanes and lane region. Below is the resultant image (left lane marked RED and right lane marked BLUE).



The region between the lanes is highlighted in Light Green Color using the OpenCV function “cv2.fillPoly()”. The resultant image is as follows.

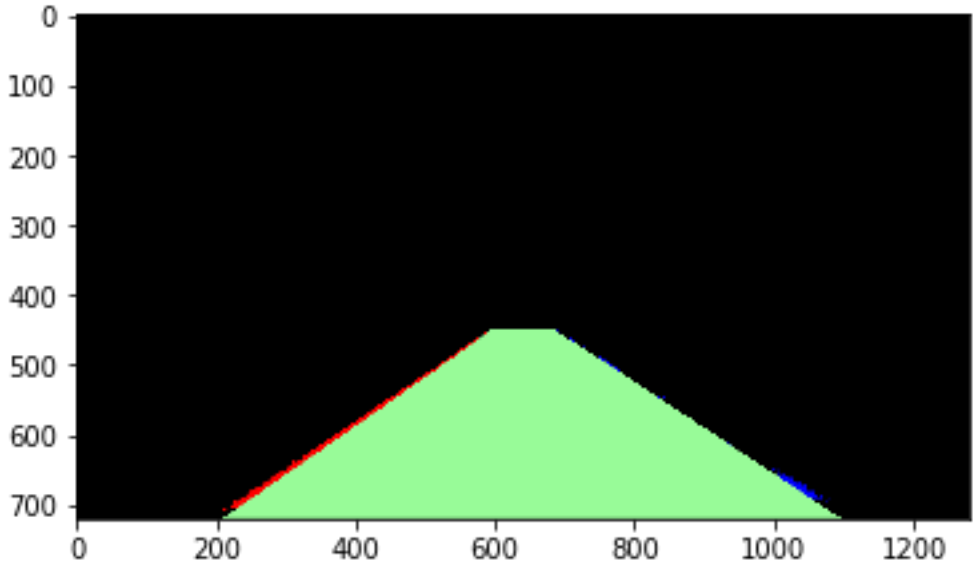


On the above image the Radius of curvature and the offset of the lane center from the center of the image are calculated. Since, we have the coefficients of the polynomials that fit the left and right lanes, we can easily calculate the Radius of curvature. If we assume that the second order polynomial is “**f(y) = Ay^2 + By + C”** then the radius of curvature is as follows.

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

The curvature for both left and right lanes is calculated. They are corrected to real world values of meters using the pixel to meter conversion parameters. Then an average of the two radii are taken and displayed on the final image.
Also, the offset of lane center from the image center is calculated as the difference between the image center and the actual lane center calculated from the polynomials fitted.

9. **Unwarp the above image:** The above image is unwrapped. Below is the resultant image.



10. **Place the above image on the actual image:** The actual image and the above image are combined using the OpenCV function “cv2.addWeighted()”. Also, the Radius of Curvature and offset of Lane center are displayed on the image. Below is the resultant image.

Pipeline (On Video):

The above-mentioned pipeline is also applied on the video "project_video.mp4" and its output is written into the folder **"output_vedios"** as **"output_vedios/project_video.mp4"**. Please watch the video from the folder.

Discussion:

1. My pipeline is failing on the challenge videos.
2. If there is any dark marking on the road, it is being considered as lane line. This can be removed by putting strict constraints on the pipeline to find lanes with certain distance apart.
3. When there are very sharp turns and very shady conditions the pipeline fails. This can be improved by applying different combinations of Gradient and Color Thresholds.