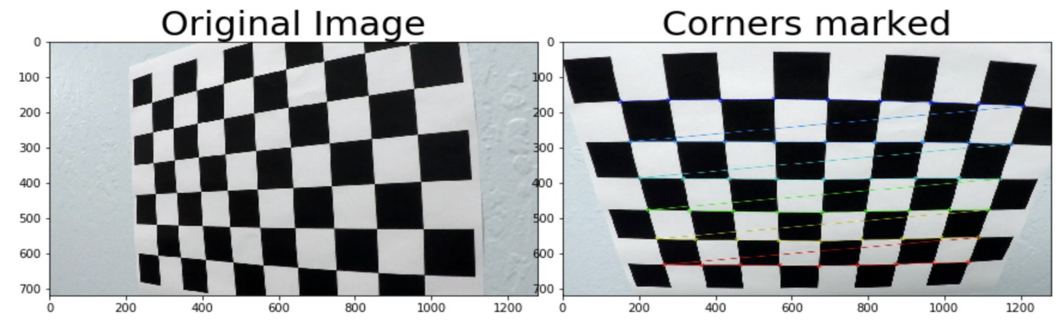
Advanced Lane Lines Project Write-Up

Camera Calibration:

- 20 Images were provided for as input for camera calibration and visually reviewing a sample of 3 or 4 images showed that the number of inside corners was 9 x 6.
- Glob API was used to read the images in as they had consistent names 'calibration(n).jpg'
- OpenCV was used to convert them to Gray scale and then to find chessboard corners as below, the images with corners found are also saved in the 'output_images/images_with_corners' directory.



- Then OpenCV calibratecamera function was to get the Camera's matrix with distortion coefficients and was saved for use with the image pipeline.
- Undistortion calibration was performed on one of the provided calibration images using the OpenCv undistort function as below:

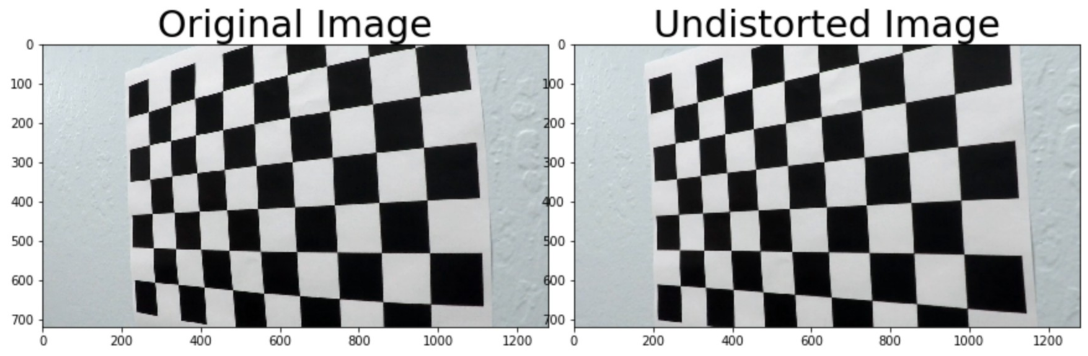
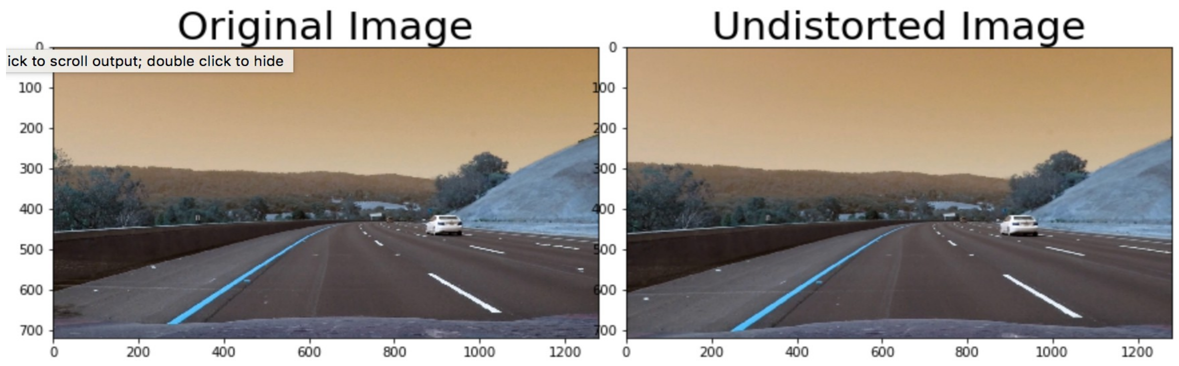


Image Pipeline :

I tested out the individual steps of the image pipeline before putting them together to create a complete Image pipeline.

Distortion Corrected Image:

Distortion correction calculated via Camera calibration was applied in the image pipeline, an example of the distortion corrected image is as below, rest of the images can be found in the 'output_images/undistorted_images' directory.

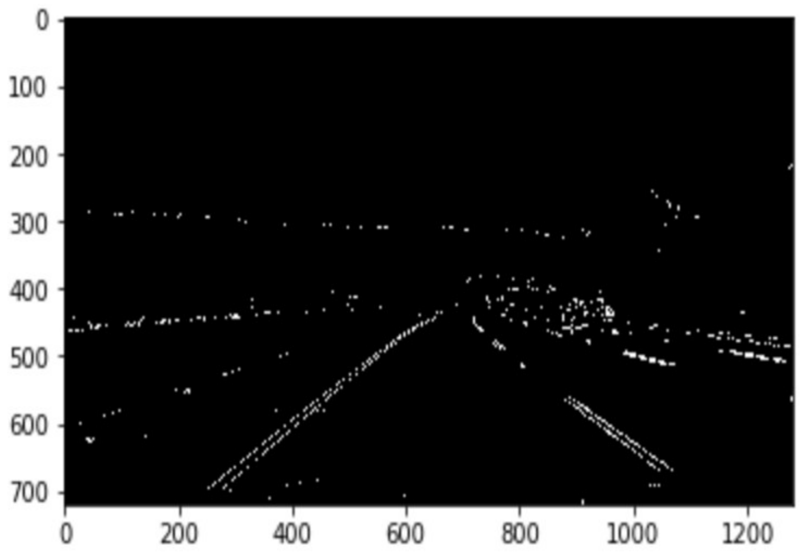


Thresholds, Color Transform

Based on the course lectures I applied the following thresholds (combined) to the test image:

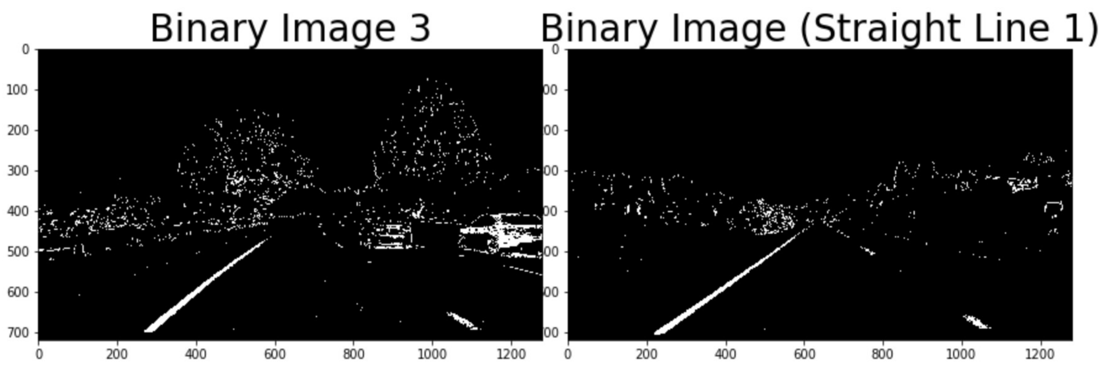
- Sobel Mask Threshold
- Magnitude of the gradient Threshold
- Direction of the Gradient Threshold

But as can be seen from the result of applying these threshold, the resultant binary image does not have good lane markings [helper function `apply_thresholds in helperfunctions.py]`



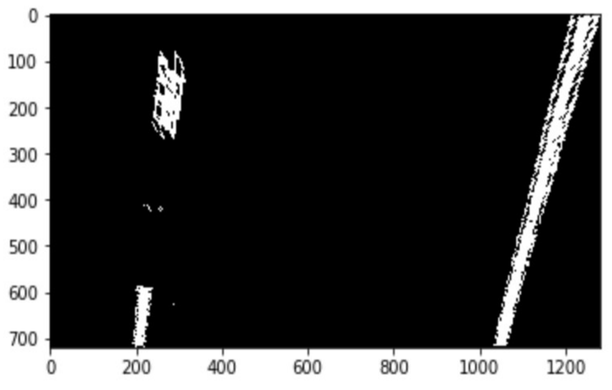
Now I learnt that converting image to Grayscale initially leads to loss of the lane color information, switching to a different color space gives us the color required color information therefore a revision was made to the thresholds as below [helper function `apply_threshold_rev in helperfunctions.py]`

- Taking the gradient in the x direction emphasizes edges close to the vertical, so I only applied Sobel in the X orientation.
- Image was converted to the HLS (Hue Lightness Saturation) color space and the Saturation channel was separated.
- Combing these thresholds gave me a better result as below



Perspective Transform:

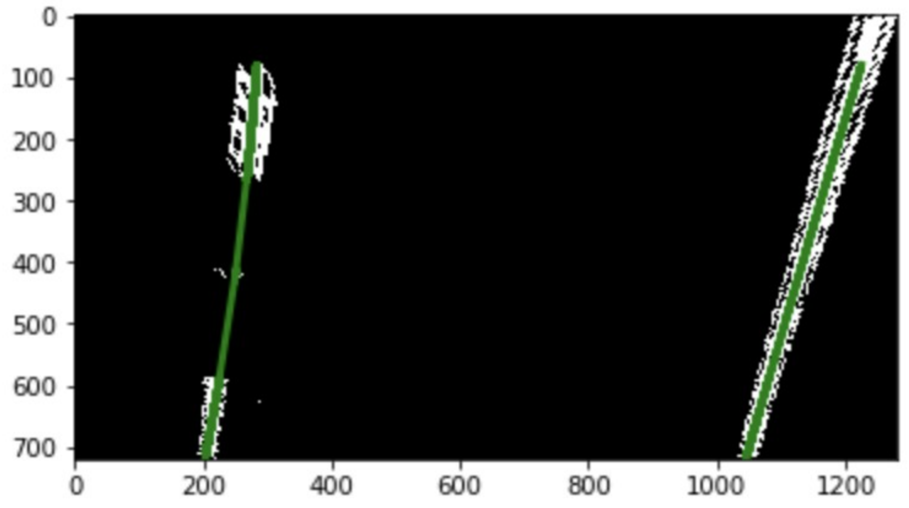
- I picked a straight line test image 'straight_lines2.jpg' to get the perspective transformation source and destination co-ordinates.
- OpenCV warpPerspective function (inside the 'image_pipeline' function) was used to warp images in the pipeline, the image in bird's eye view, undistorted and applied the combined thresholds looks as below:



Lane Line Pixels and Polynomial fit:

- 'histogram_pixels' helper function takes the warped perspective image and applies sliding window of radius 200.
- Vertical offset is used t (this is redundant as during the perspective transformation I had already picked the source co-ordinates such that the hood of the car is ignored.
- Horizontal offset is used as we want to ignore any smaller peaks which are left of the left lane peak and on the right side of the right lane peak.
- Histogram was taken of the bottom of the image
- Histogram was smoothed with a medianfilter of kernel size 51
- Left and right peaks were identified and then left and right window pixels were found.
- Polynomial fit was done using the 'fit_second_order_poly' function which uses the numpy 'polyfit' function.

Image with Lane pixels identified and polynomial fit is as below:

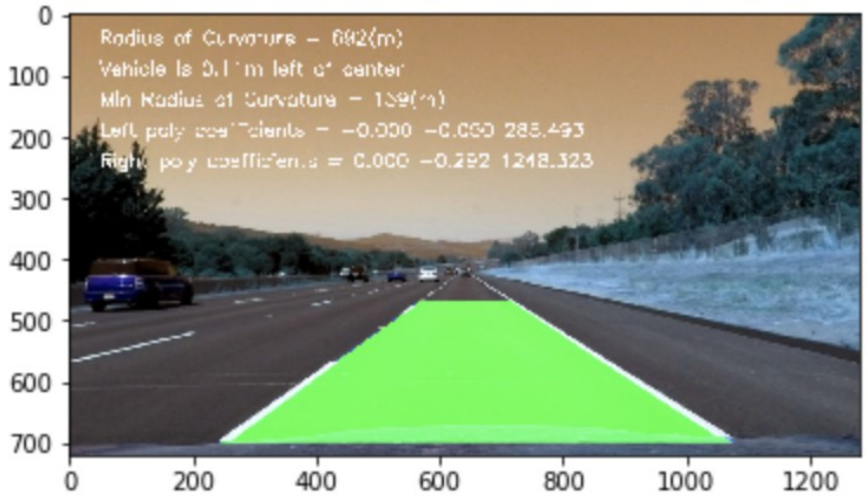


Radius of Curvature of the lane, vehicle position wrt Center:

- Y value for determining the Radius of curvature was chosen as 719, corresponding to the bottom of the image closest to the vehicle.
- Radius of curvature was calculated on the pixel values. Note that the left and right co-efficients were already determined from the second order polynomial fitting.
- Helper functions 'plausible_curvature' and 'plausible_continuation_of_traces' perform Sanity check on the calculated curvature and the co-efficents to weed out the outliers.
- Helper function 'center' position of the vehicle with respect to the center.
- Helper function 'add_figures_to_image' which will be described in the next section also converts pixels to meters.

Image Plotted back on the Road (Lane):

- Helper functions 'draw_poly' draws the left and right lane pixels.
- Image is then converted to color and the lane areas are highlighted with the 'highlight_lane_line_area' helper function.
- Image is warped back to the original configuration and the 'add_figures_to_image' which also does the pixels to meters conversion add the lane overlay and the following information
    - Radius of curvature
    - Min Radius of curvature
    - vehicle's position wrt center
    - Left Polynomial Co-efficients (3)
    - Right Polynomial Co-efficients (3)



```
Radius of Curvature = 692(m)
Vehicle is 0.1 m left of center
Min Radius of Curvature = 139(m)
Left poly coefficients = -0.000 -0.000 285.493
Right poly coefficients = 0.000 -0.292 1248.323
```

Test Image Pipeline on the Video:

Output video 'project_video_lane_lines.mp4' is included in the repository.

- Lanes are identified in every frame
- Radius of curvature and vehicle position outputs are generated
- Pipeline does not change with shadows, road color changes.

Discussion:

- As described in the threshold section, I had to revised the combined threshold function as the first version of the function lost color information due to the image being Grayscale. HLS color space was used to overcome to improve the quality.
- I had trouble with the histogram detecting the smaller peaks on the left of the left lane and right of the right lane which was solved by selecting a horizontal offset.
- I had trouble selecting the source and destinations for the perspective transformation as the interactive plot was not loading on my computer so after several revisions and accounting for the fact that the hood of the car is present at the bottom of the image, I was able to get the numbers right.
- I had trouble determining the vehicle position wrt center as I had picked a Y value which is not closest to the vehicle, I then set it as 719 (close to **image.shape[0]**.)
- On the test video, the pipeline struggled to detect the right lane when the color of the pavement changed, so while checking the absolute difference between the previous and current co-efficent, I relaxed the absolute difference and the performance then was nominal as can be seen in the output video.
- While the pipeline performed extremely well, I tested it out on the harder video and found issues below:
    - The hardcoded source and destination positions do not translate well in this new scenario.
    - Also the pipeline does not handle the rapid transitions between bright sunlight (oversaturated) frames and the dark frames.