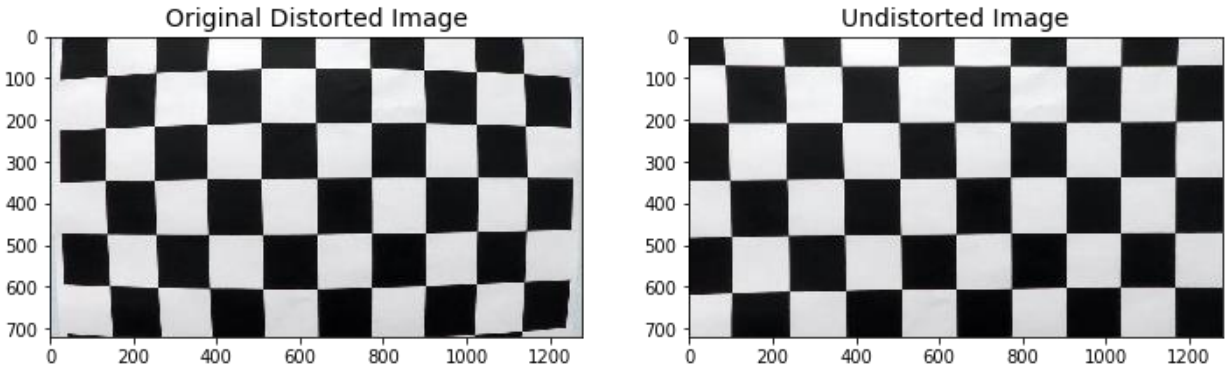
Project4 Advanced Lane Finding

Basic Summary

Goal of this project was to detect lanes more robustly as compared to the first project. This gave an insight in to how real world Lane Keep Assist feature might work. This project made me appreciate machine learning much more since this was really hard to calibrate manually.

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

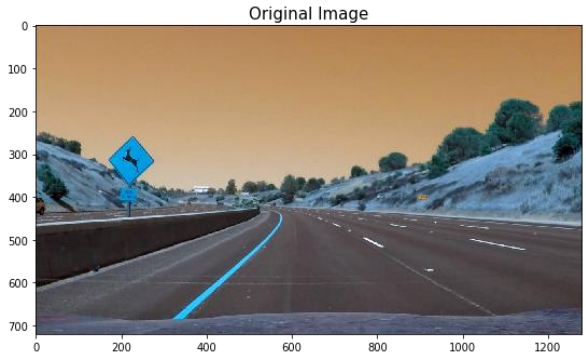
Computation was done using opencv functions findChessboardCorners and calibrateCamera. Also, undistort is used to remove any distortions from the image.

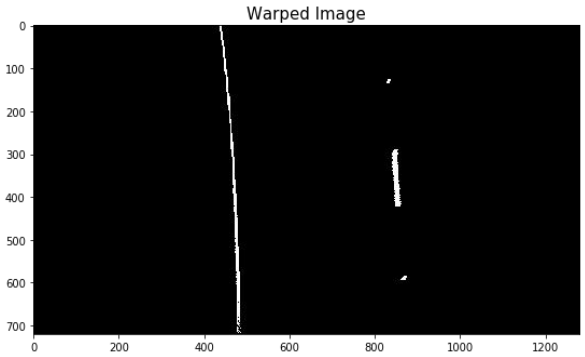


Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result. Where did you perform the perspective transform?

I initially played around with generic techniques like Gaussian Blur and Canny Edge. But, for a better solution I ended up creating a HLS_THRES function using the LAB colorspace after trying HLS and LUV. This function was working with the main pipeline but I was missing lane detection during the shadow areas. Adding CLAHE helped to make it more robust.

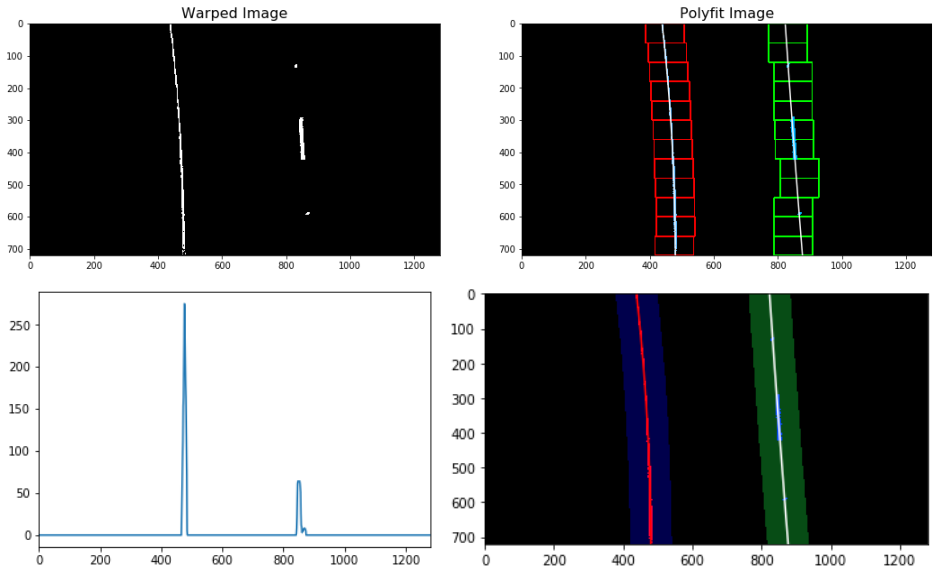
After this I created a master function called WARP which used undistort along wit HLS_THRES function above for simplicity of usage. I also added perspective transform to this WARP function with source and destination points.





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

I used a sliding window function which determines lane pixels from left and right lines. Histogram was created for lower half of the image to determine base left and right lane position. I got better results by halving this further to quarter size. Parameters for minimum pixels for each lane line and 12 widows to identify lane lines and hence do the polyfit 2nd order based on good pixels. Also, histogram plot for left and right line was created.Further poly_last_fit function uses the last good historical values to display the threshold range.



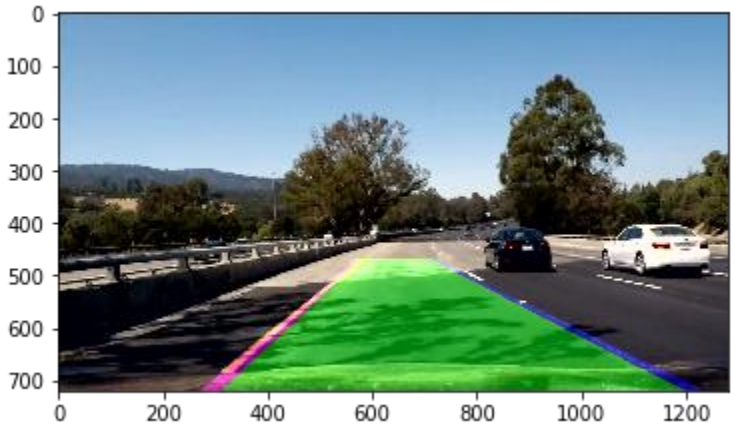
Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center?

This was done in the ROC function which calculated both radius of curvature and lane offset.

```
#Radius of curvature snippet

left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5)
/ np.absolute(2*left_fit_cr[0])

right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.
5) / np.absolute(2*right_fit_cr[0])

#Lane offset snippet

l_fit_x_int = l_fit[0]*j**2 + l_fit[1]*j + l_fit[2]

r_fit_x_int = r_fit[0]*j**2 + r_fit[1]*j + r_fit[2]

lane_center_position = (r_fit_x_int + l_fit_x_int) /2

center_dist =(car_position - lane_center_position) * xm_per_pix
```

Provide an example image of your result plotted on the road

I forgot to mention earlier but I had also converted video frames in to images for tricky shadow regions to test my algorithm rather than running the entire model. This can be seen below.



Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video

project_video_output.mp4

Reflections and problems faced

Calibration was the hardest part for this project. My algorithm was acting funky around shadows until I introduced CLAHE to my pipeline. This pipeline would fail around tighter radius of curvatures like in the mountains and will be sensitive to conditions like snow.