Advanced Lane Finding Project

The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

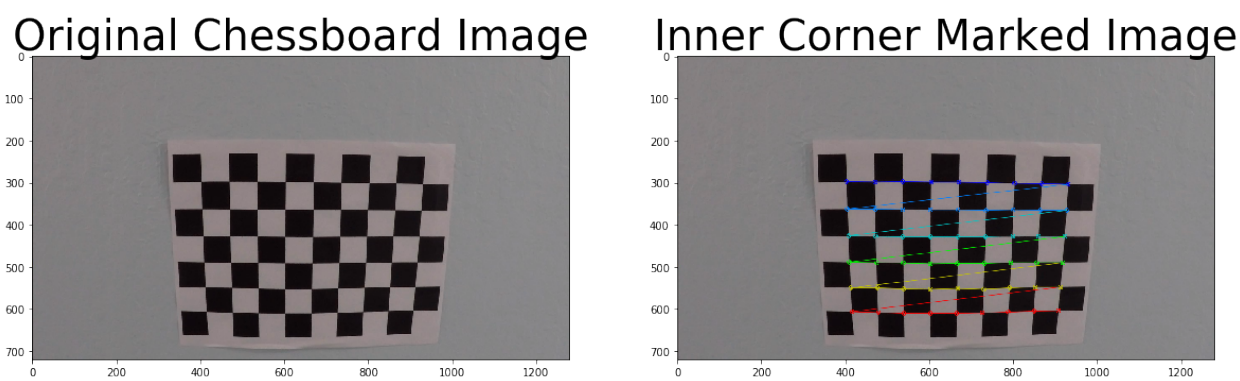
Rubric Points

###Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

###Camera Calibration

The calibration for the current camera is calculated and undistorted by using a 9x6 chessboard. Given 20 images of the chessboard in different angle and position. This part has been done in the first section of the Jupyter Notebook file ‘p4.ipynb’

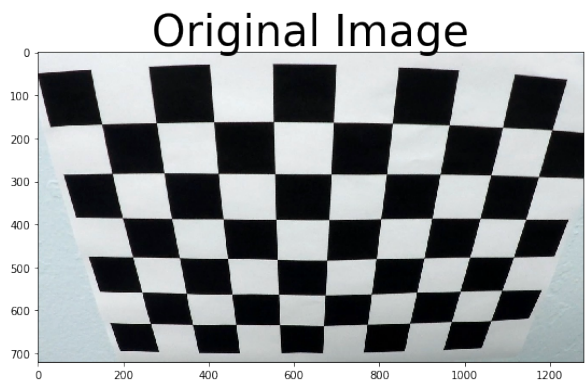
1. First we need to find out the inner conner of the chessboard image using openCV function ‘cv2.findChessboardCorners’

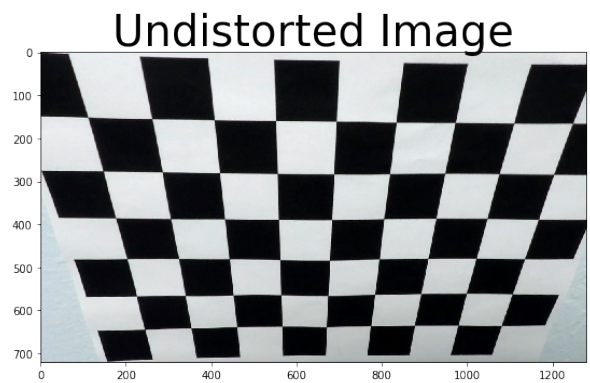


2.

2. With 20 images we basically covered the most distorted image cases, images are in the folder ‘camera_cal’. The second step is to create two points array, ‘objpoints’ and ‘imgpoints’. The object points give a 3D points array for all the inner conner points while the image points are the points that mapping those 3D points in to 2D plane. With theses points we can use ‘cv2.calibrateCamera’ to calculate the camera calibration matrix and distortion coefficients ‘mtx’ and ‘dist’, which will be used later.

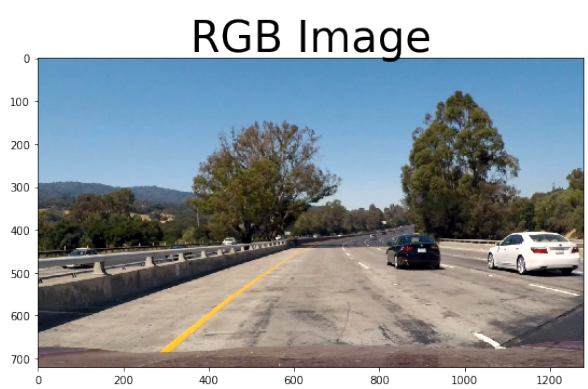
3. After we got ‘mtx’ and ‘dist’ we can use the function cv2.undistort to calibrate the distorted images like:

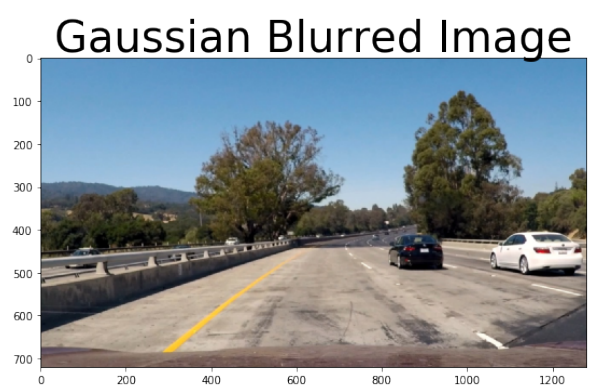




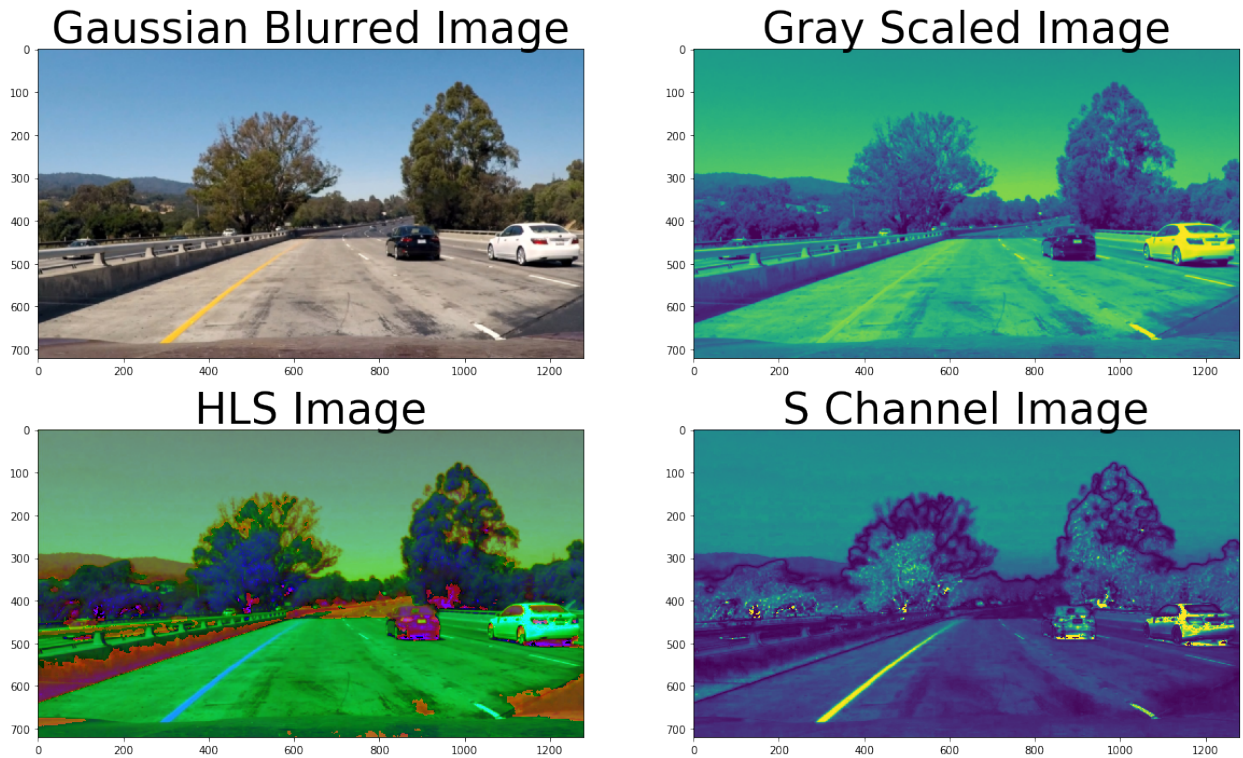
###Pipeline Construction
The pipeline construction only includes the generation of the final binary image, the calibration and warp will be explained in the next section. This section is mapping the 2nd section in the Jupyter Notebook file ‘p4’.

1. First I applied Gaussian Blur onto the original image to smooth the image

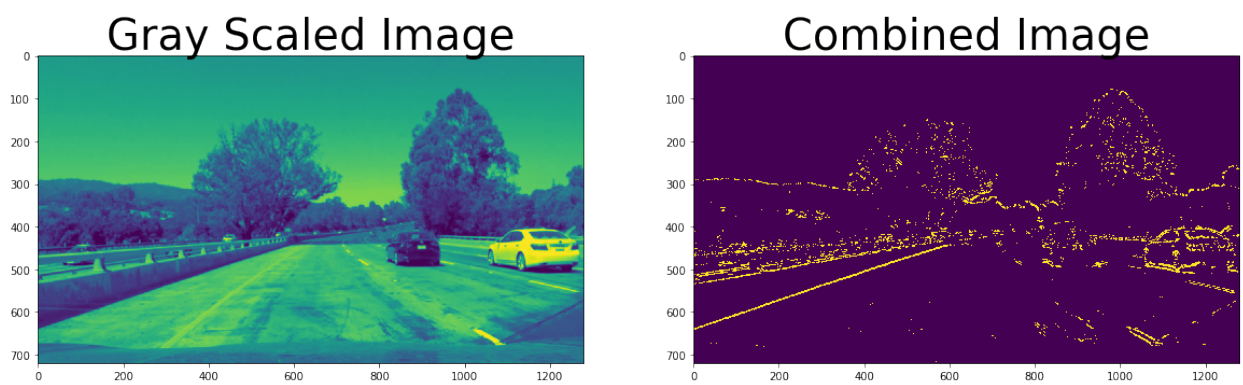




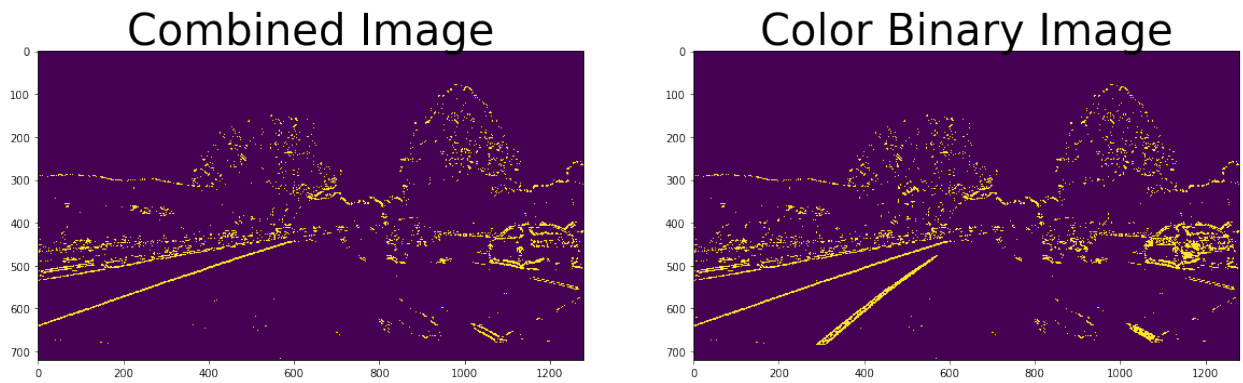
2. Second I used Gaussian Blurred Image generate one gray scale image and one HLS colorspace image and get the S channel. Gray scaled image will be applied with gradient thresholding and S channel image with be applied color thresholding.



3. Next I applied multiple gradient thresholding onto the gray scaled image using Sobel transfer, and got the combined image

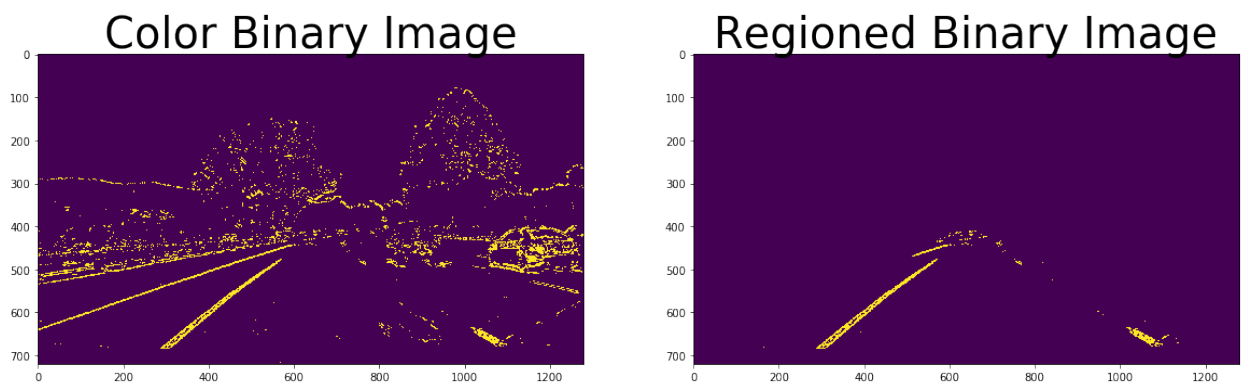


4. Next I applied thresholding onto the S channel Image and generated the binary image



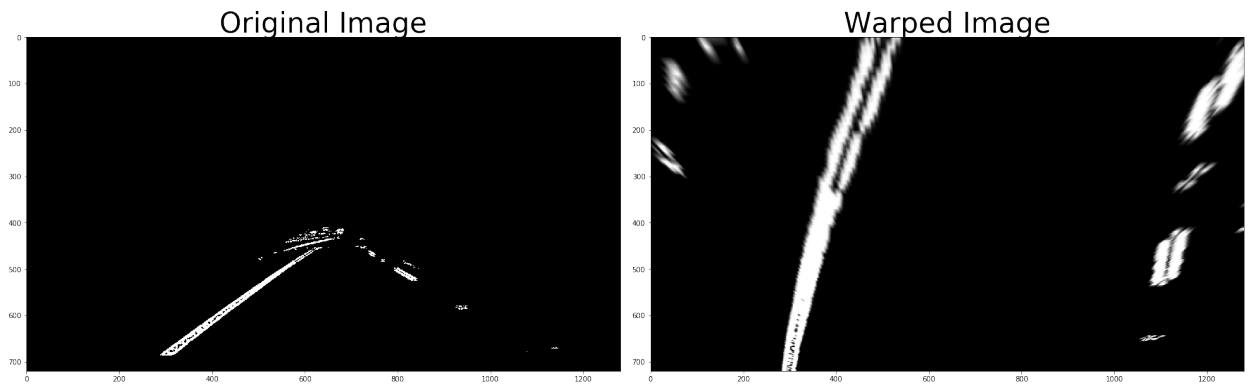
5.

5. Finally I found our interesting region and limited the out put image



6. From the input and output images for the pipeline are

###Calibration and Warping
I integrated Calibration and Warping into one function called ‘undist_warp’, this function will take the binary image, ‘mtx', ‘dist’ as the input parameters, where binary image will be get from the last section, ‘mtx’ and ‘dist’ are the coefficient that we obtained from the chessboard images in the first section.
Using undist_warp, we will get the image processed as:

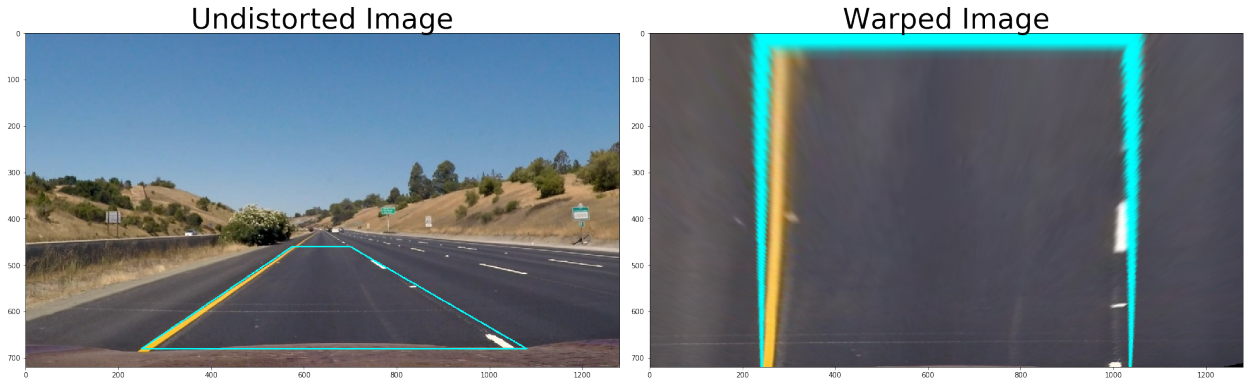


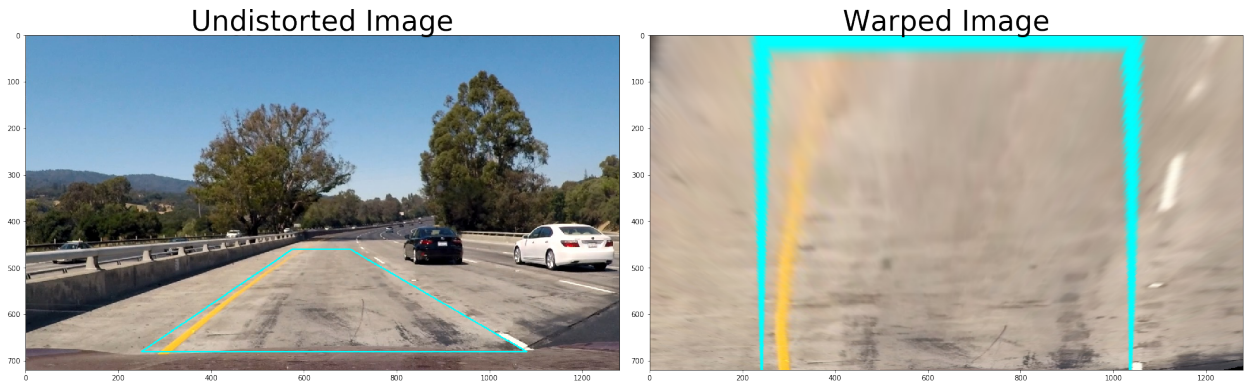
It is worth noting that we have also transferred the image from man-kind-eye view into the bird-eye view, because when I warped the image I applied perspective transform, so that we can generate the bird-eye view.

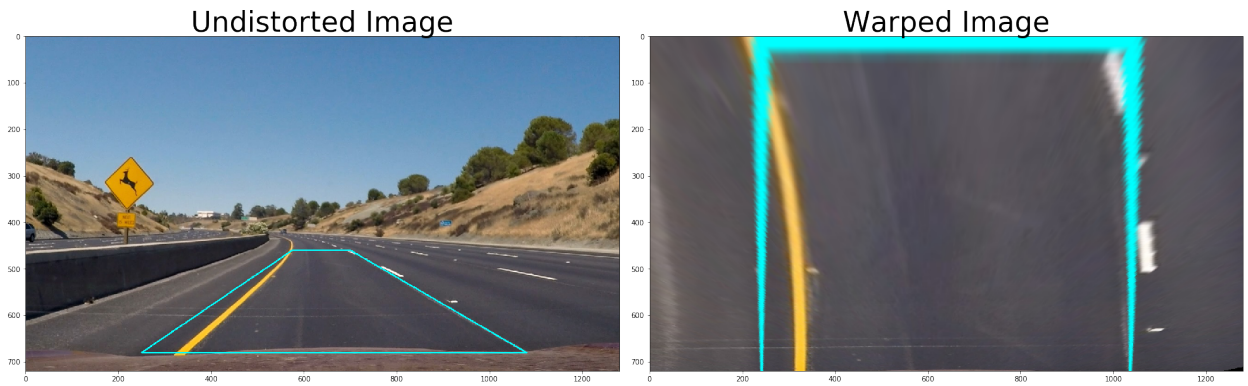
The mapping points are showing in the following table

Source Points	Destination Points
[575, 460]	[240, 0]
[700, 460]	[1040, 0]
[1080, 680]	[1040, 720]
[250, 680]	[240, 720]

Therefore the calibration and warp precess of few test images with visualization of the interesting region will be like:

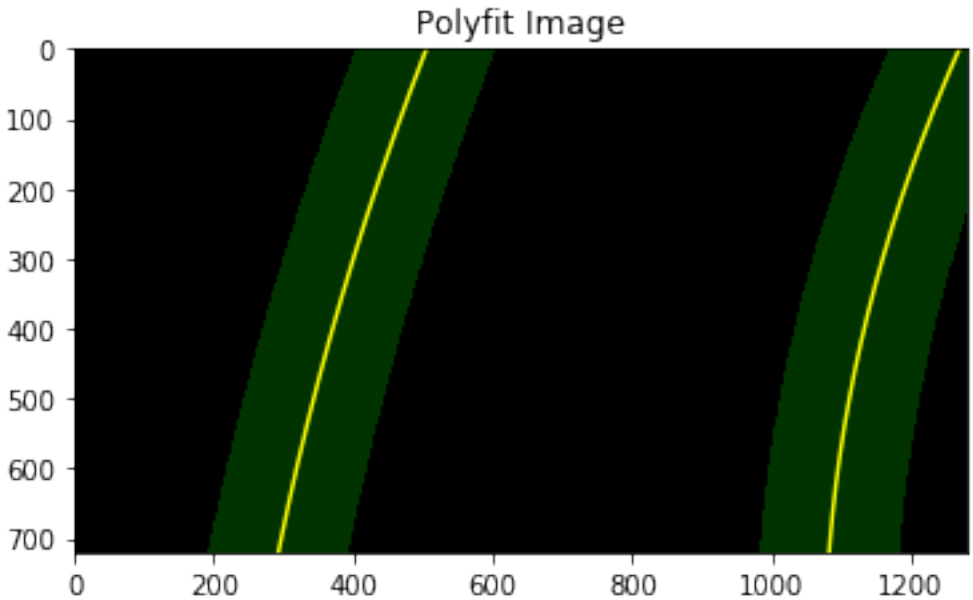






###Find Lanes and Polyfit
After we got the bird-eye view binary image of the lane lines, I used the few functions given by Udacity to find the polynomial fitting function of both lane lines, so that we can use the found equations to draw the found lane area as well as calculating the curvature of the lane lines.

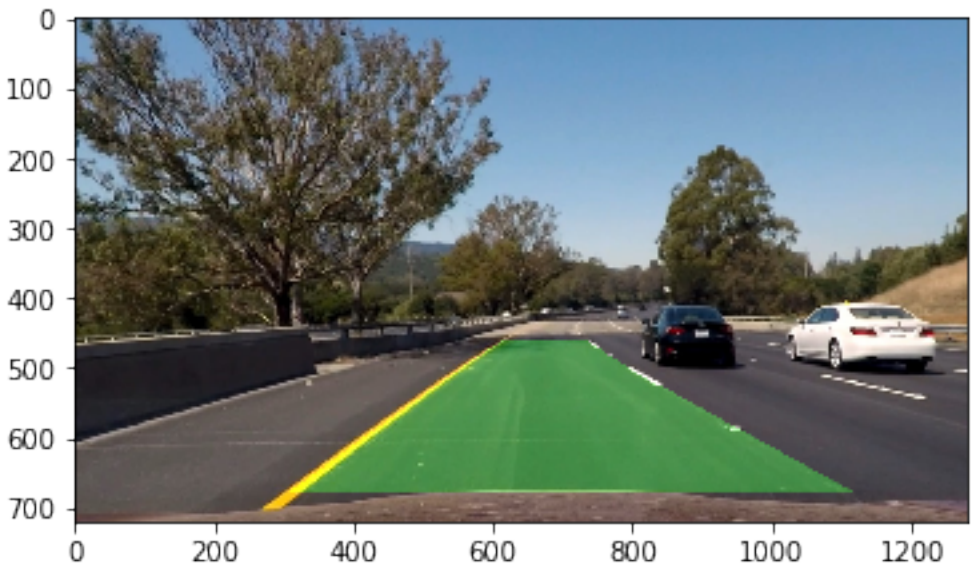
This parting is mapping the section in Section 4 of Find Lanes in the Jupyter Notebook file ‘p4’ in the function ‘find_equations’.



The yellow lines are indication the polyfit equations from the lane lines and the green color there is just paddings.

It is worth noting that the process to find the lane lines actually includes 1. Divide the image in the vertical direction into a number of windows, in the code, I divided it into 9 window. 2. Loop through each window, we will be able to find the max summed y values using histogram. 3. Because we have separated the image into left and right two parts, so the left peak with be where the left lane located and same for the right side. 4. After we found all the lane line points, we can use Numpy function ‘polyfit’ to find the lane lanes and we will later use this points to calculate the curvature.

###Draw the Lines / Lane Area back into the Calibrated Image
Here I applied the perspective warp again but in a reverse way to draw the area back into the calibrated image mainly using two functions, first 'cv2.fillPoly', which will do the drawing work, second, 'cv2.warpPerspective', which will do the perspective transform. The result is like:



###Find Curvature of the Lane Lines and the Camera Position
These are covered in the 7 and 8 Section in the Jupyter Notebook file 'p4', in which I created two functions that takes the polynomial equations info that we obtained from last section to calculate the curvature and position.

To calculate all the distance related values, we have to establish an relation between pixels and meters. I used the following constant given by Udacity:

ym_per_pix = 30/720 # meters per pixel in y dimension
xm_per_pix = 3.7/700 # meteres per pixel in x dimension

Since we only consider second order polynomial function, so that the equation of the lanes can be treated as

$$f(y) = Ay^2 + By + C$$

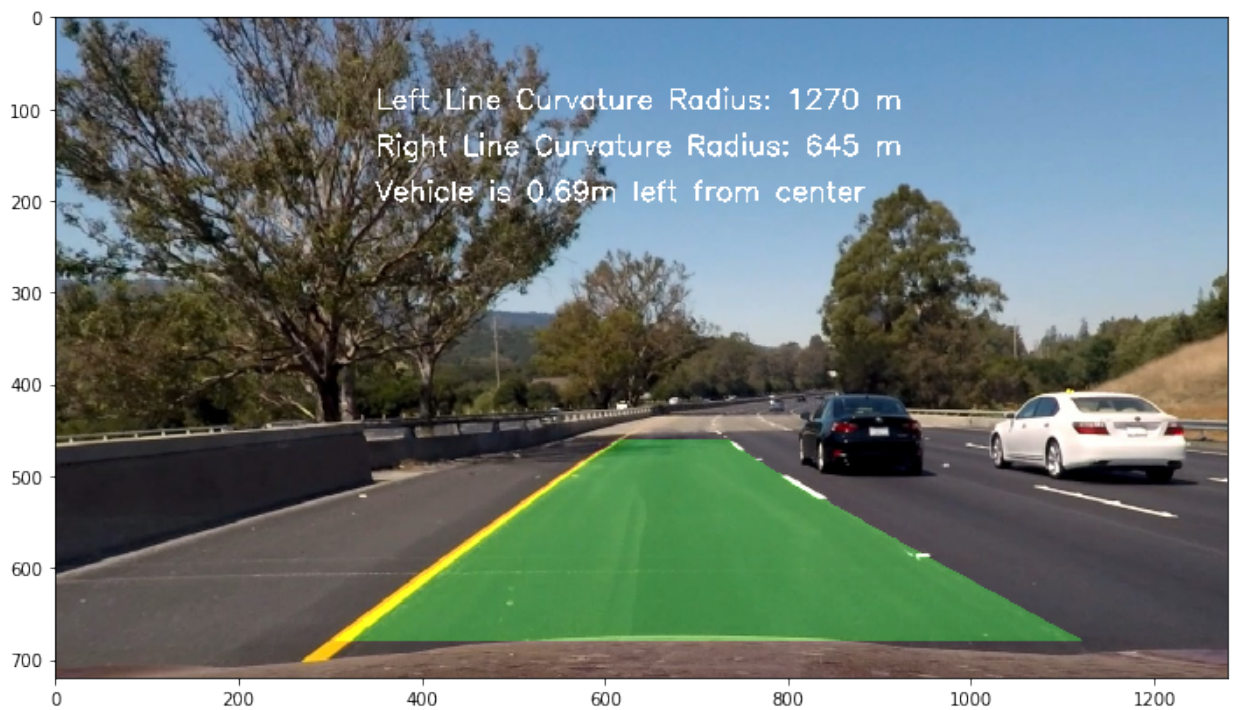
The radius of curvature at any point *x* of the function *x*=*f*(*y*) is given as follows:

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

So, our equation for radius of curvature becomes:

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

Which is what I used in my Jupyter Notebook file to do the calculation. After all the calculation, I used ‘cv2.putText’ to put all the text info onto the image, like



Here are few more examples with the comparison of the original images.

Finally everything including pipeline and all other precesses are put into one single function called ‘precess_image’, and in the end I used Moviepy framework to generat the final video.

###Final Project Video Link

I have uploaded the processed image video for the second submission here in Youtube.
https://youtu.be/BhhuXfa0P2s

###Discussion
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?
The image_precess can be better tuned with the thresholds in the pipeline function, also the region of interest are so much dependent on the car where the camera is mounted. This is the mainly reason that my model doesn't work well on the challenge videos.

In the second submission, I added lane line smoothing onto my existing function 'find_equation' in the Section 4.1 in the Jupyter notebook file 'p4'. The idea of lane line smoothing comes from the hint in the project section. I have created two empty arrays to store the previous 14 frames lane line information.
temp_left_fitx = []
temp_right_fitx = []
Every time I got a new frame of the image and found out the lane line left and right fit x values, I am calculating an average left_fitx and right_fitx instead of directly using whatever I got from the current frame.

This process and dramatically improve and eliminate some sudden failure of the current frame image.