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.

The complete code for this project along with comments is contained in **project.py** file of this submission.

Camera Calibration

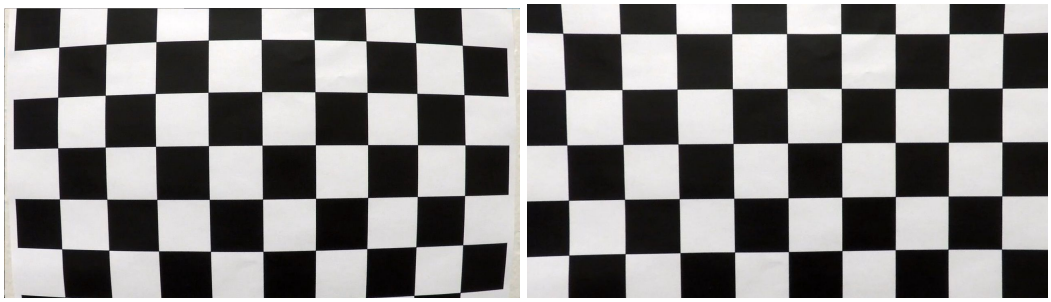
Compute the camera matrix and distortion coefficients

The code for this step is contained in the file project.py from **lines 9-35** where calibration parameters are calculated from the given camera_cal images. Most of the code is taken from the udacity lecture on camera calibration.

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:

Original Image Undistorted Image



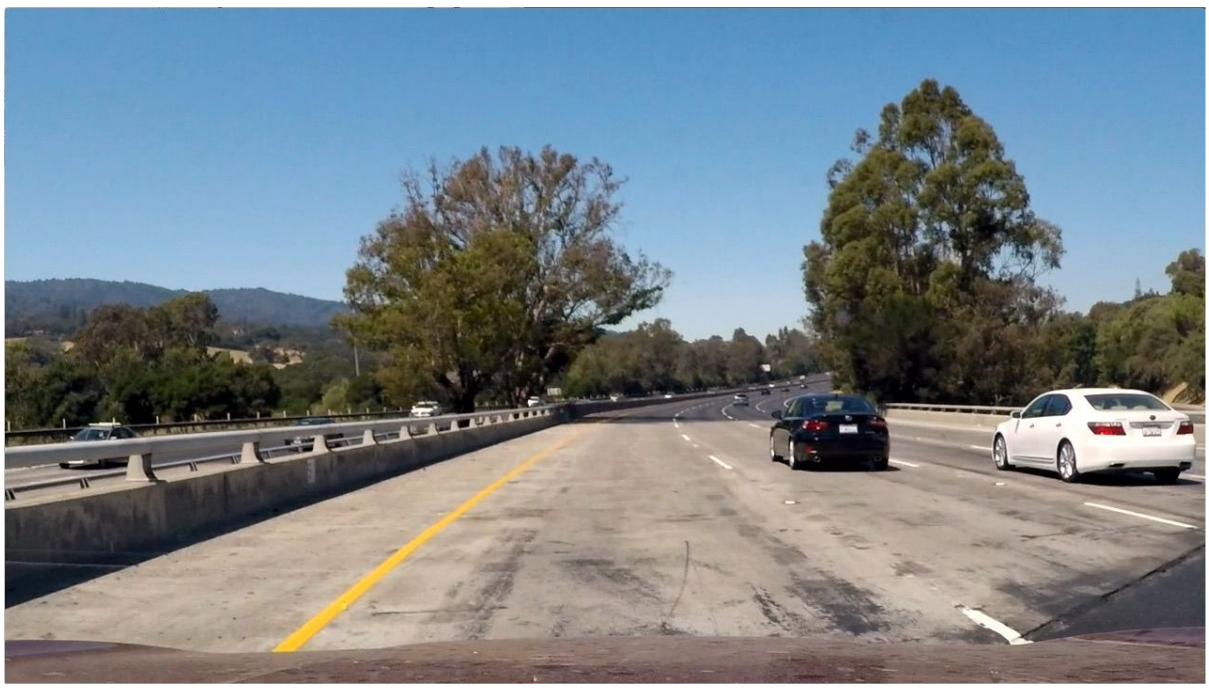
Above undistorted image is named as **undistorted_1.jpg** in **output_images** folder

Pipeline (single images)

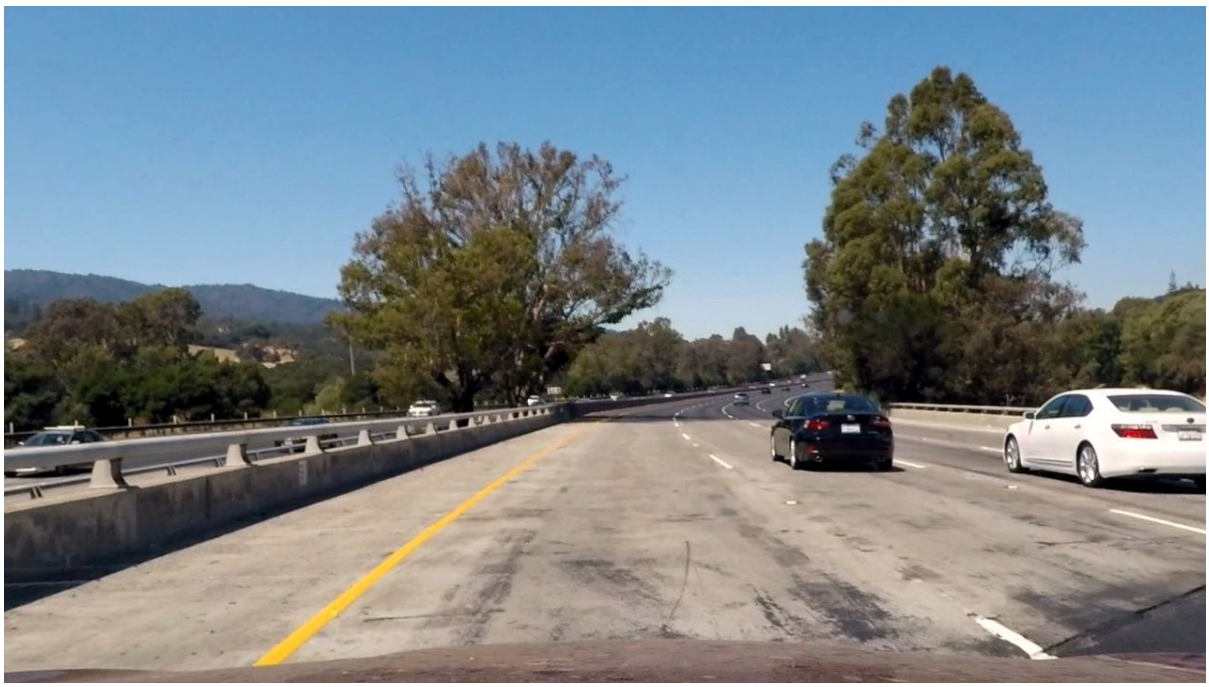
1. Example of distortion-corrected image

Distortion correction is applied on each image read from the video at **line 290.**
A sample of pipeline flow is shown for test1.jpg image

Original Image:



Distortion-Corrected Image:



Above image is named as **test1_undistorted.jpg** in **output_images** folder.

2. Color transforms and sobel gradients to get a thresholded binary Image.

I used a combination of color and gradient thresholds to generate a binary image. The code to this is in the lines **59-96.**
I convert the undistorted image into HLS space. I get a thresholded image by applying a fixed lower and upper threshold (**line 67)** on the S channel image (call it image1).
I get another binary image(call it image2) by applying threshold on the Sobel x gradient image applied on the S channel above. Code is in lines **80-83 and 92.**
I get the last binary image (call it image 3) by applying threshold on the Sobel x gradient image applied on the grayscale image of undistorted image. Code is in lines **75-78 and 89.**

Final binary image is obtained by combining all the three binary images (image1,image2 and image3) from above. Code is in lines **94 & 95.**

The S channel is used for image 2 and image 3 processing as it fared superior to other color spaces in identifying lanes as shown with examples in color thresholding udacity lecture and also through personal experimentation.

Below are the examples of images described above :

Image 1 (Threshold on S channel):

Image 2 (Threshold on Sobel x grad applied to S channel):

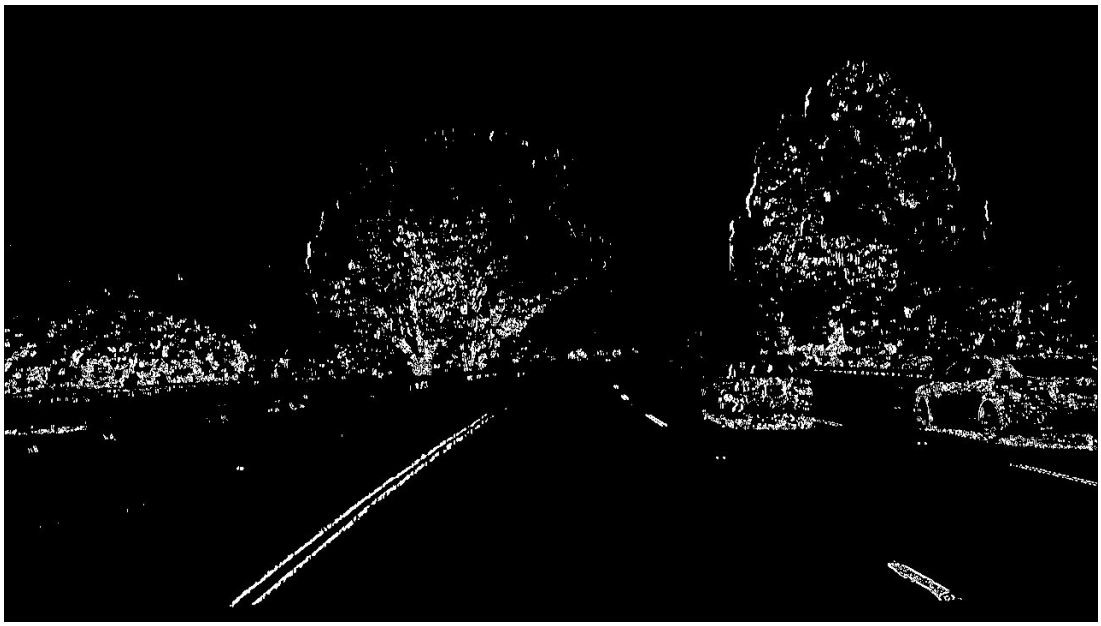
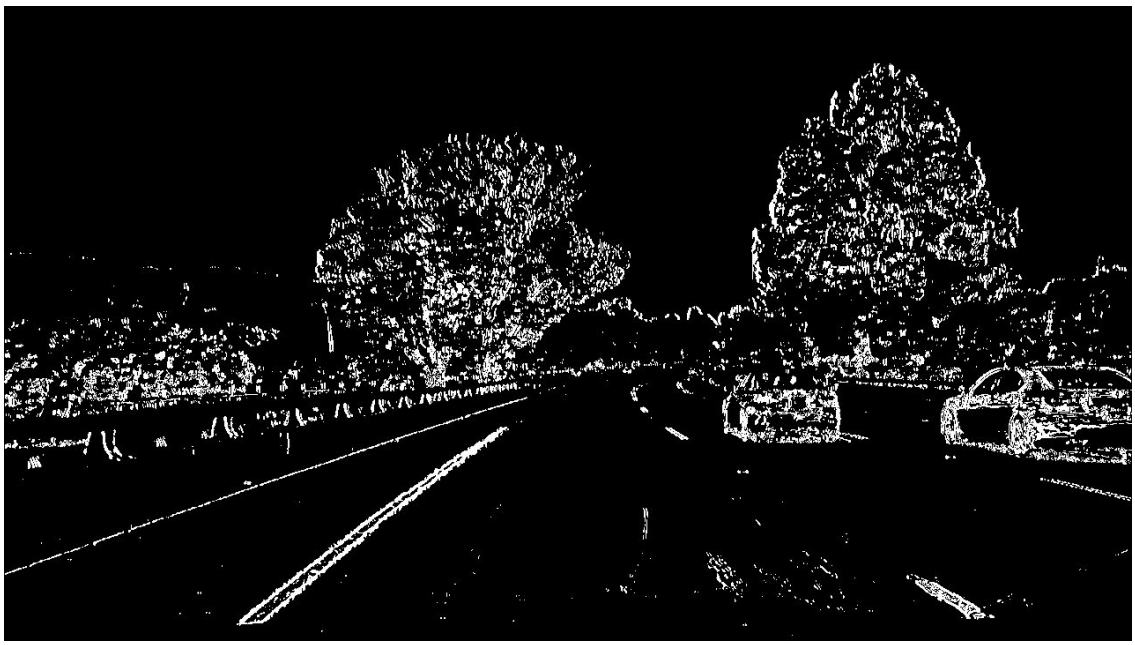


Image 3 (Threshold on Sobel x grad applied to Grayscale)

Combined image (Combining all three binary images above)



Above images are named as **S_channel_threshold.jpg ,** **Sobelx_Schannel_threshold.jpg, Sobelx_grayscale_threshold.jpg** and **Combined.jpg** in **output_images** folder.

3. Perspective transform and warping.

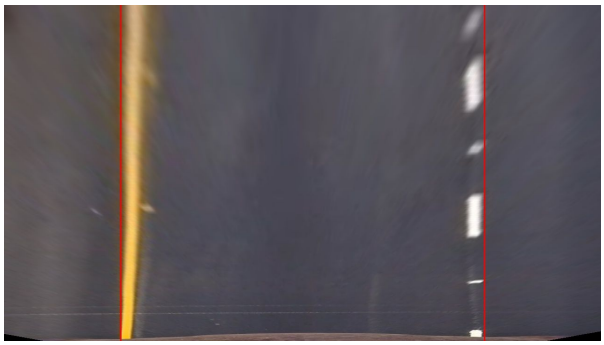
The code for perspective transform and warping is in the lines **37-52.**
For perspective transform I used the following hard coded source and destination points.
The source points are chosen by examining straight_line images given in the project folder.

Source	Destination
193, 720	250, 720
586, 454	250,0
695,454	1030,0
1125,720	1030,720

I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

Undistorted image with source points drawn

Warped results with dest points drawn



Above images are named as **unwarped.jpg** and **warped.jpg** in **output_images** folder

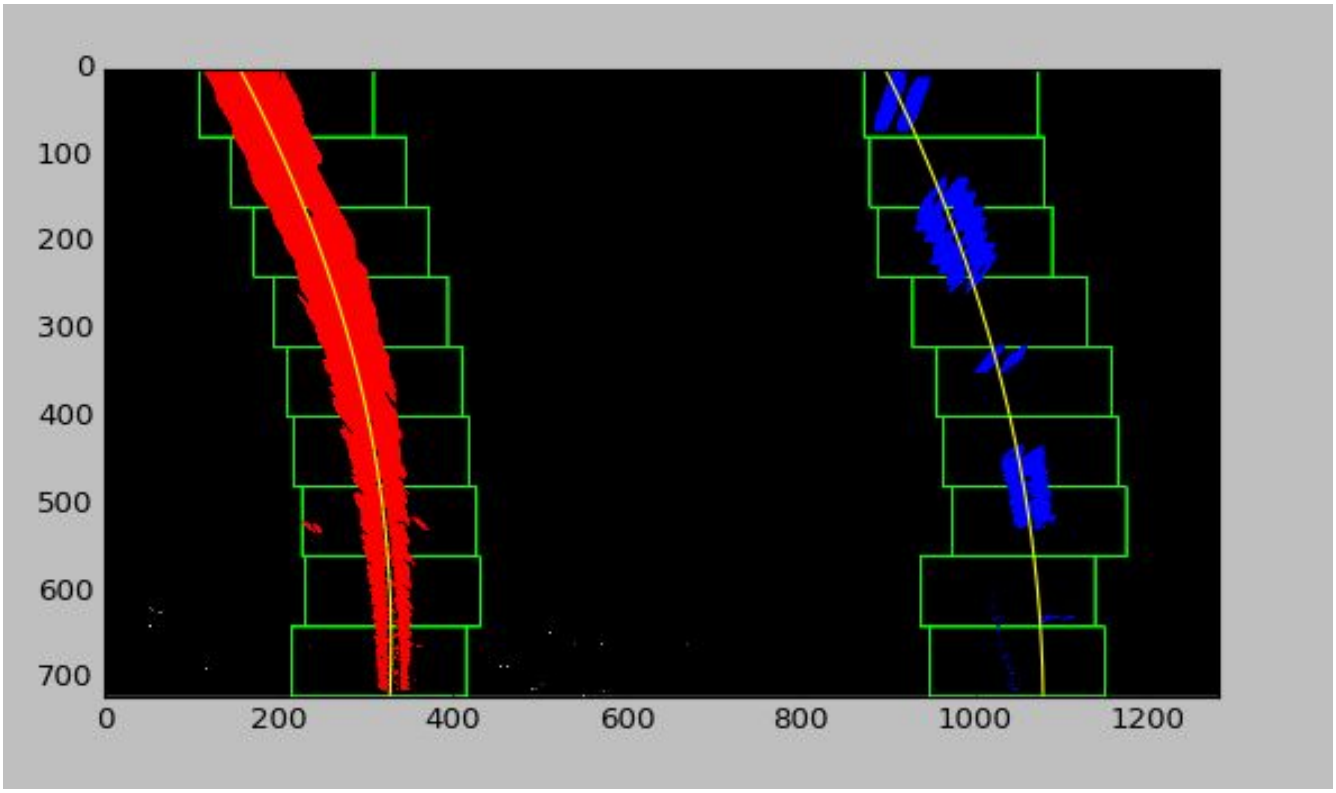
4. Identifying lane-pixels and fit their positions with a polynomial

Once we get the binary warped image, the procedure to find the lane lines and fitting the polynomial is taken directly from the Udacity lecture “ Finding the Lines”.

First a histogram is taken along the columns in the lower half of the image. The two peaks in the histogram are used to find the starting positions of left and right lane pixels. Then sliding windows placed around the line centers, follow the lines up to the top of the frame. Then a two second order polynomials are fitted for the pixels identified in left and right lanes (lines 189 and 190).

The code for this is present between the lines **112 and 216.**

Lane lines plotted on the warped image for test2.jpg in the given project folder



Above image is saved as **Lane_lines_with_curves.png** in **output_images** folder

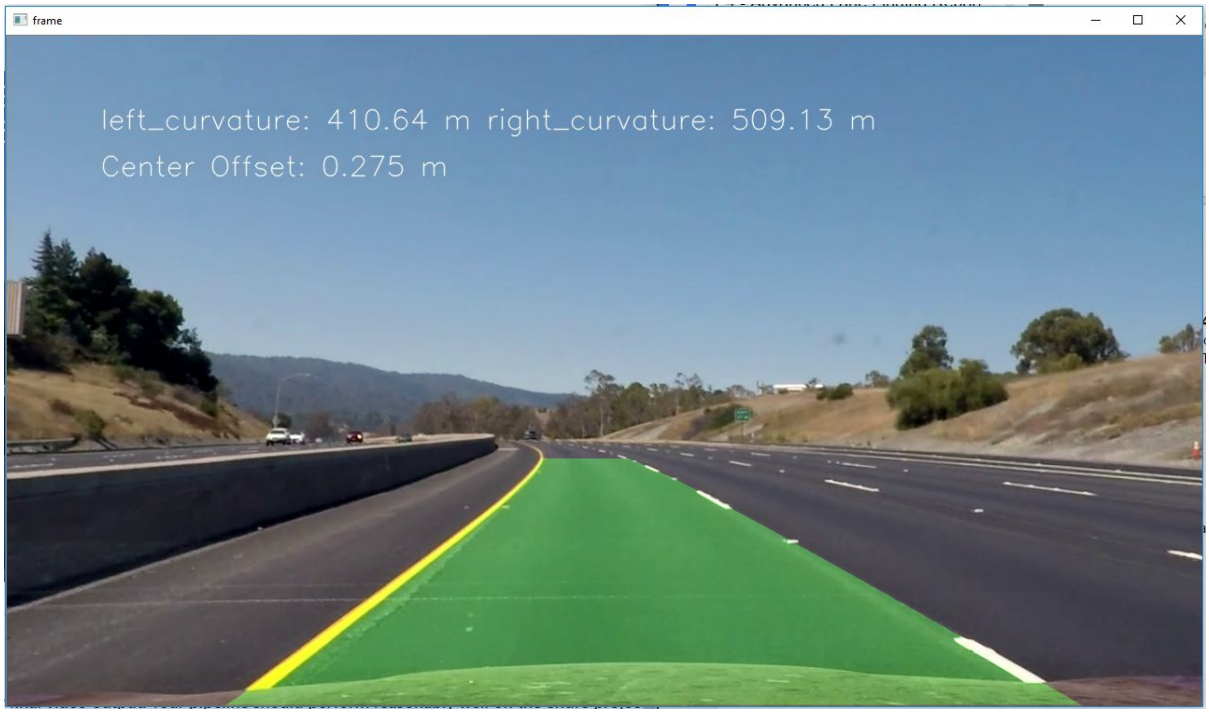
5. Radius of the curvature of the lane and the position of the vehicle with respect to center

The code to find curvature is presented in **lines 255-259.**
The radius of curvature is calculated from the equation
$R_{curve} = \frac{|2A|}{(1+(2Ay+B)^2)^{3/2}}$ where the polynomial equation is
$f(y) = Ay^2 + By + C$. The code and algorithm are used from "Measuring Curvature" udacity lecture.

The code to find the offset from center is present in the lines **236-243.**
We calculate the minimum and maximum x coordinate of the projected lane on the original image. Using this we find the center x coordinate as (min+max)/2. This is subtracted from the image center x coordinate to find the offset.

6. Result with the lane plotted on the image

The result image is created in the code at lines **234-246.** An example image with lane plotted is shown below.



Pipeline(Video)

A video named "**output.mp4**" is presented with this submission that has the lane identified with curvature and offset values plotted.

Discussion:

First task was to identify lane lines in the test images given. After a lot of experimentation with the images I came up with thresholds that generated a good binary image identifying the lines.

Once lane lanes has been identified from the binary image and curves have been plotted on the video, the images were very wobbly. So I introduced smoothing by creating a line class and storing the last few line detections to average with the current detection. I used a queue of fixed length to store the moving average (**line 104 and 108**). This mitigated the jerkiness in the images and yielded a good result. The code corresponding to this is at lines **197 to 212** where average lines are calculated.

Further improvements:
More robust pipeline to get the binary lane image need to be generated as the current pipeline doesn't perform well on challenge videos.
The pipeline is dependent on traditional computer vision techniques and few hardcoded values specific to the resolution of the image. More work needs to be done exploring deep

learning / machine learning techniques and adapting the hyperparameters to be more generic.