Advanced Lane Finding Project

The goals / steps of this project are the following:

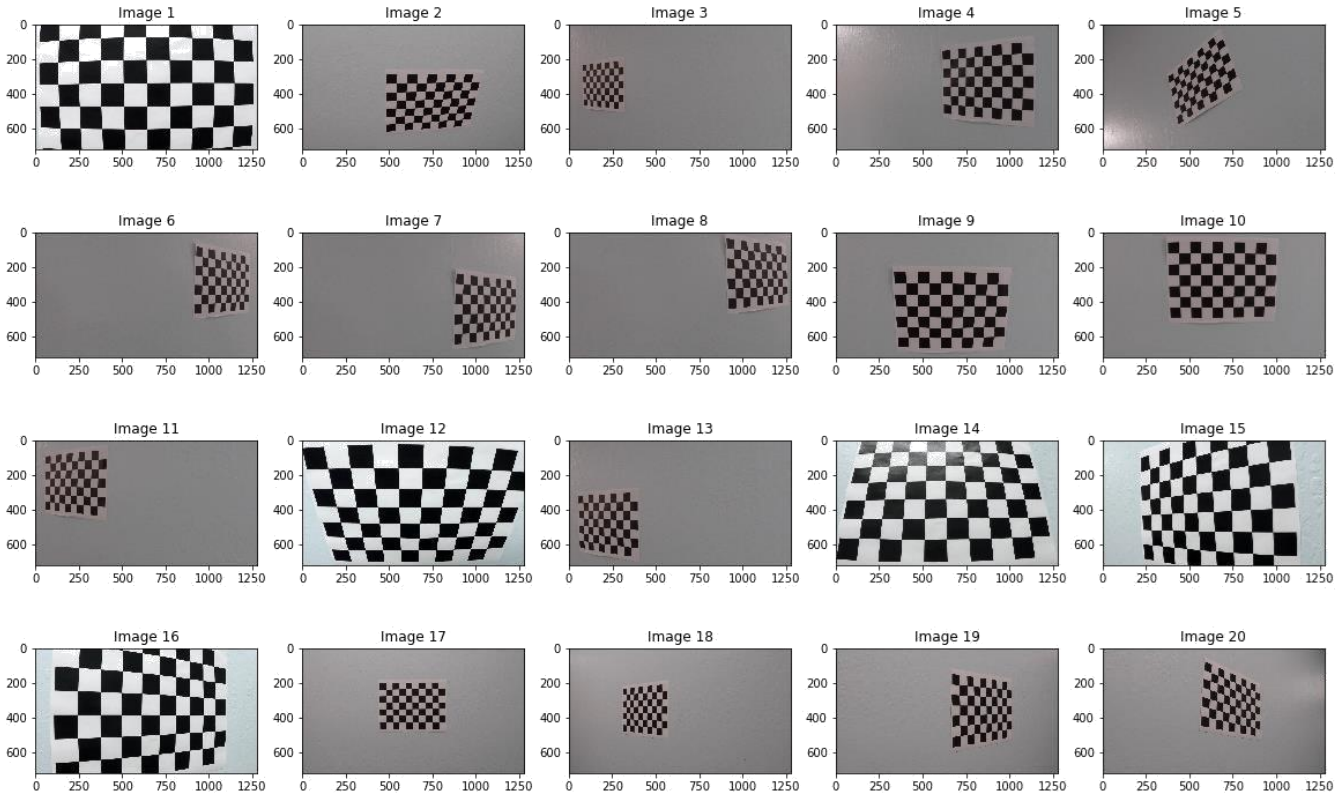
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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.

Camera Calibration

1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

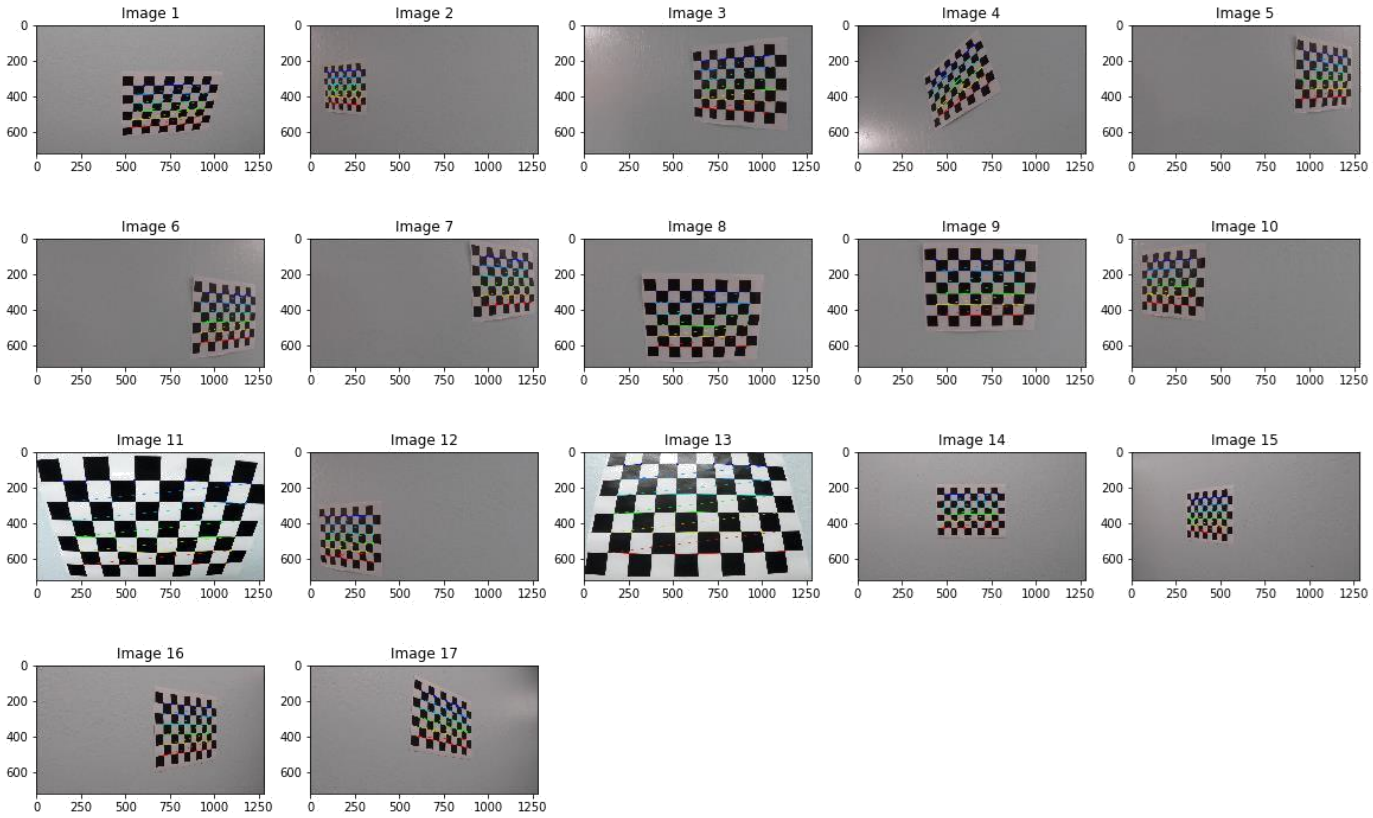
The code for this step is contained in the third code cell of the IPython notebook.

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. These are all the images present in the dataset for calibration.



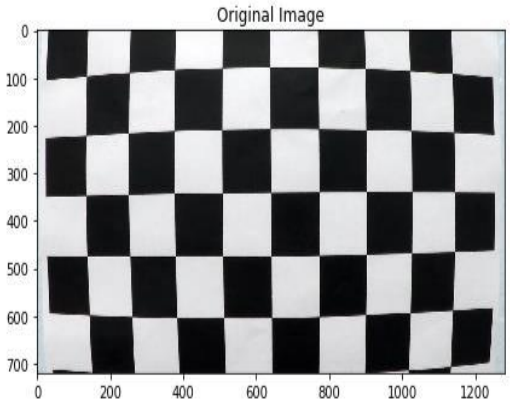
Images from calibration dataset

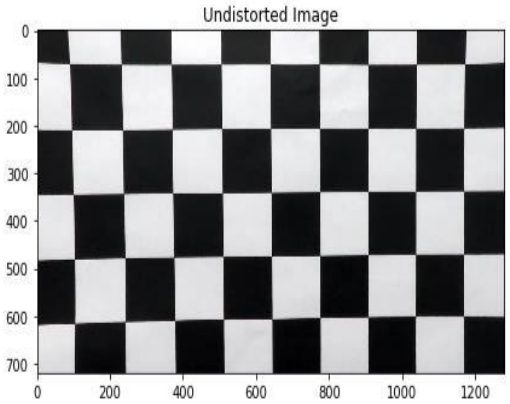
Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection. I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. The output after detecting corner points:



Images after finding corners

I applied the distortion correction to the test image using the cv2.undistort() function and obtained this result:

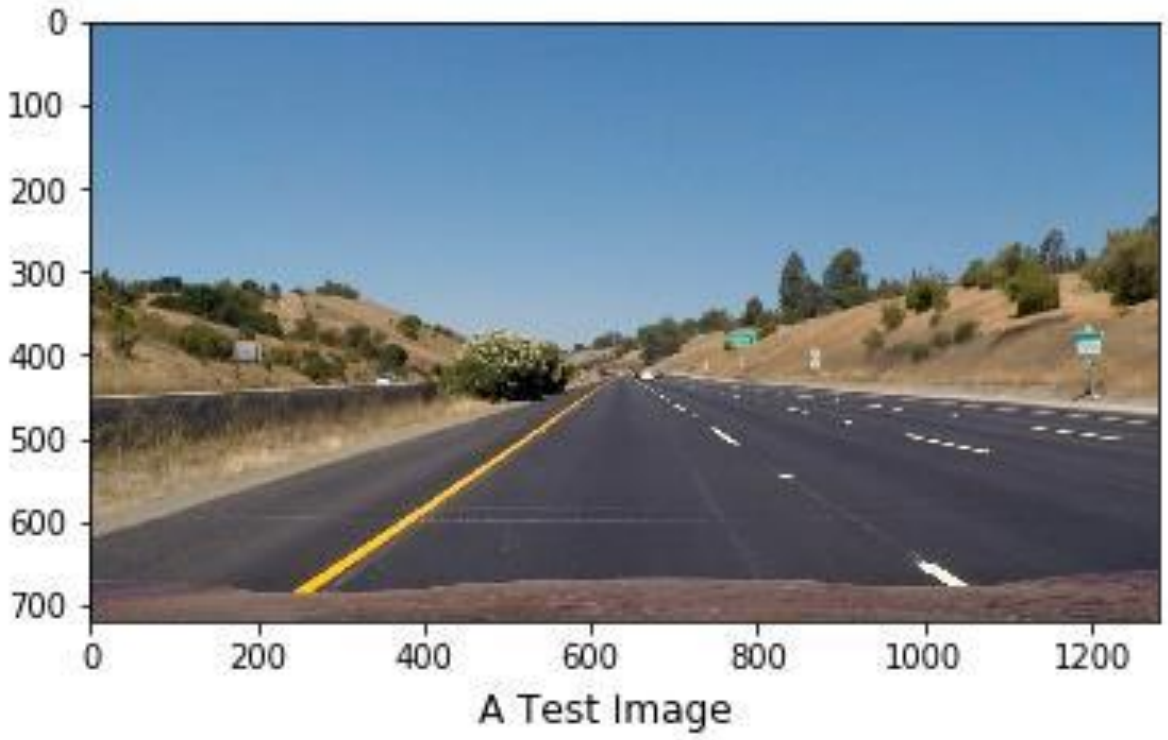




Pipeline (single images)

1. Has the distortion correction been correctly applied to each image?

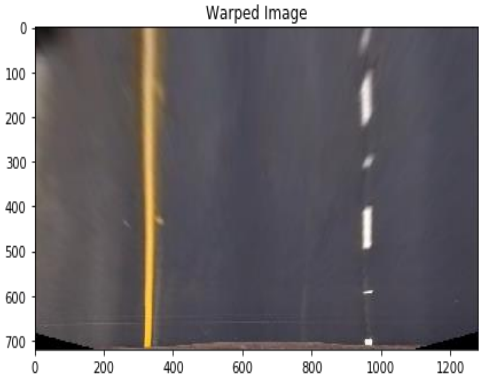
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:



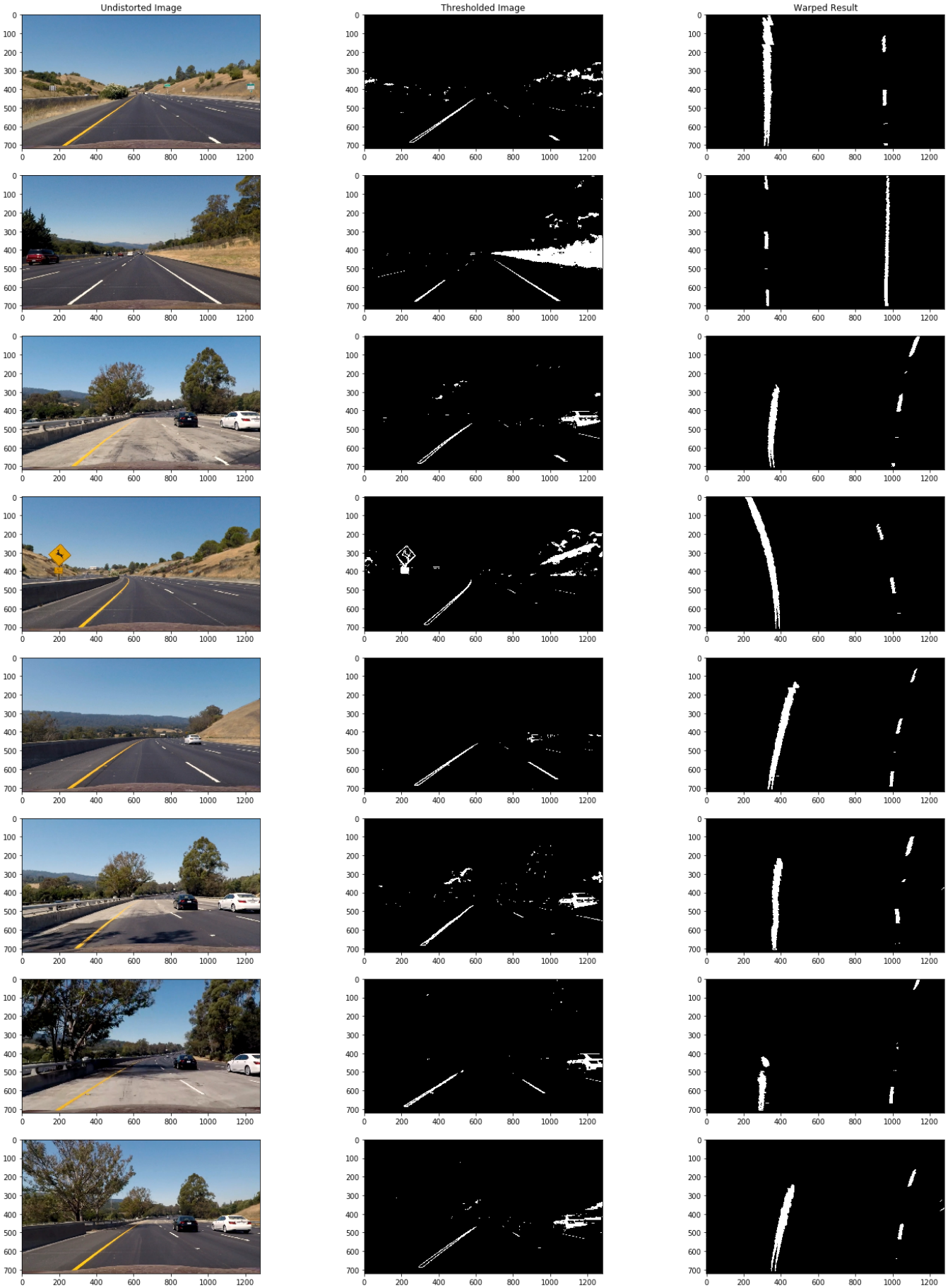
I have created a function to undistort the Sample Road images and used it inside and outside pipeline to undistort the images. Here's the output after undistorting the test image.

2. Has a binary image been created using color transforms, gradients or other methods?

Yes, I have created binary image after applying undistortion. But before that I warped (perspective transform) the image just to check whether the souce and destinations points are correct. This is the output after warping the test image:



After that, I checked all the outputs of different thresholds techniques – Sobel X, Sobel Y, HSL – S threshold, LUV – L threshold, LAB – B threshold, Combined Thresholds. Then, I found that for this project combination of LUV-L and LAB-B will be enough. So I used thresholds for LUV-L and LAB-B channel only. This is output after thresholding and warping the image:


Undistorted Image
Thresholded Image
Warped Result

3. Has a perspective transform been applied to rectify the image?

Yes, perspective transform has been applied to rectify the image.

My source and destination points are as follows:

w,h = 1280, 720

x,y = 0.5*w, 0.8*h

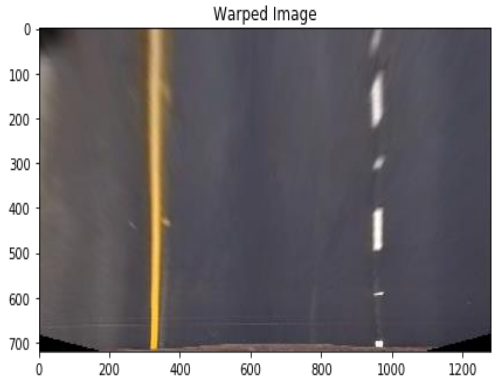
src = np.float32([[200./1280*w, 720./720*h], [453./1280*w, 547./720*h], [835./1280*w, 547./720*h], [1100./1280*w, 720./720*h]])

dst = np.float32([[(w-x)/2., h],[(w-x)/2., 0.82*h], [(w+x)/2., 0.82*h],[(w+x)/2., h]])

These values I found out in the discussion forum. I tried brute force method to find source and destination points but the result wasn't as expected. So my mentor sent me a link for discussion forum where I found out this technique. In this project, the camera is fixed on the car and car is moving in a single lane, a fixed source and destination window works better for the transformation. A dynamic window selection will be much better when we have lane crossings – but this adds an over-head in the computation. But the dynamic window selection will be good for the harder challenge video.

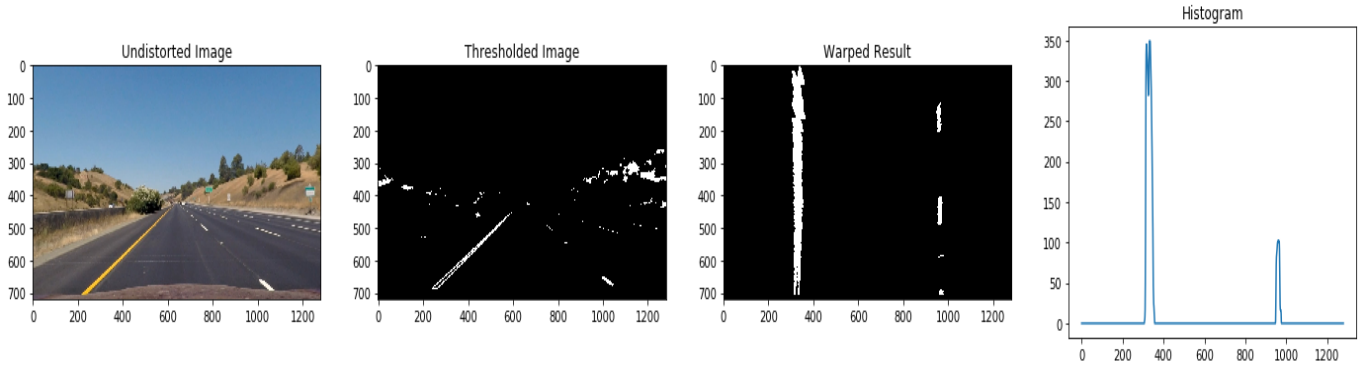
After finding source and destination points, a cv2 method (cv2.getPerspectiveTransform) is used to find the Perspective transformation matrix. This matrix is independent of Color Channels.

The output of perspective transformation would be as a bird-eye view of the road. The output is as follows:

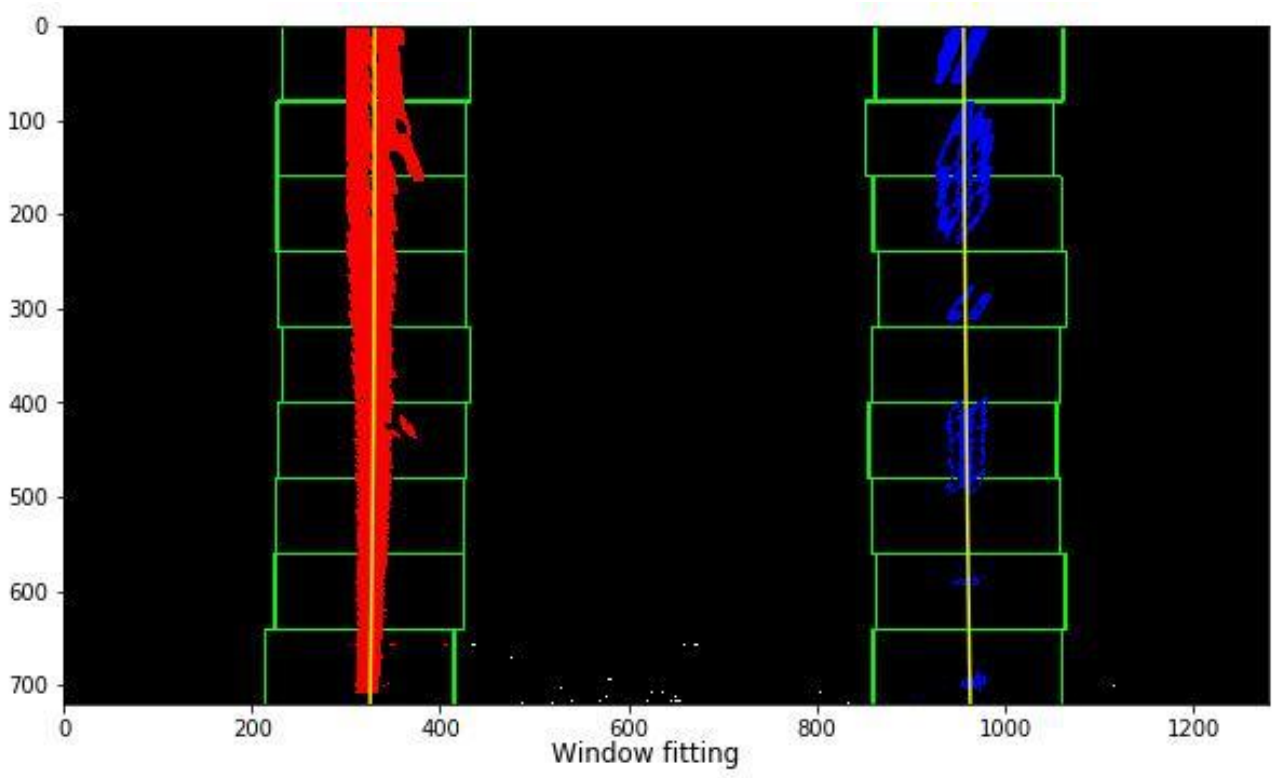


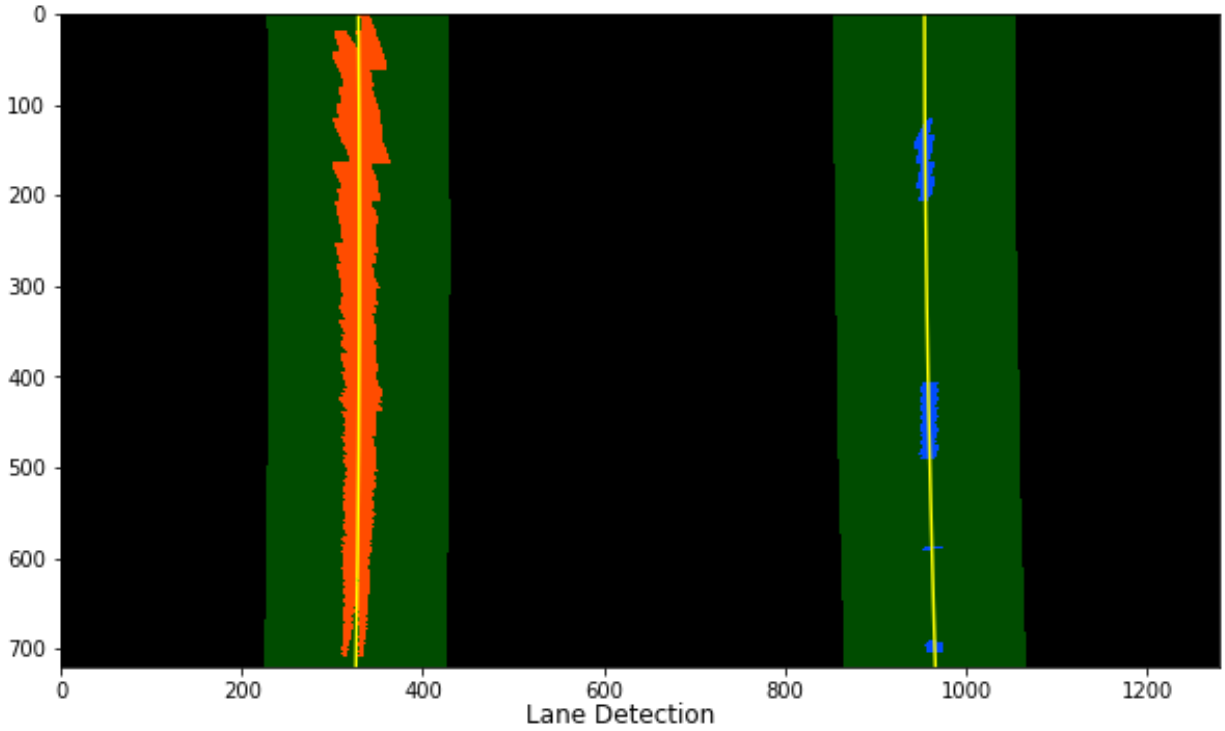
4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

For finding the lane line pixels and fitting it with a polynomial, I have used sliding window approach. I first take a histogram along all the columns in the lower half of the image like this:



With this histogram I am adding up the pixel values along each column in the image. In my thresholded binary image, pixels are either 0 or 1, so the two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. I can use that as a starting point for where to search for the lines. From that point, I can use a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame. Later, these points were used to fit in a 2^{nd} order polynomial for each lane (both Left and Right).





The green shaded area shows where we searched for the lines this time.

5. Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?

The formula for radius of curvature is:

$$f(y)=Ay2+By+C$$

$$R_{curve}= (1+(2Ay+B)_2)_{3/2} / |2A| \text{ at a particular y.}$$

But, we know that the polynomial fit operation done will be Pixel metric and not in general Metric (meters). So, to compensate this, I used this conversions:

ym_per_pix = 30/720 # meters per pixel in y dimension

xm_per_pix = 3.7/700 # meters per pixel in x dimension

Post conversion, the data points have been refitted with a cv2 Polynomial fit which gave a new set of A, B, C values and thus, the radius of curvature have been computed.

I calculated the radius of curvature at y_eval = 700 which gives a good approximation of radius and the viewer can relate to it logically during the video. The radius increases while the road becomes straight and decreases while the car is maneuvering across a turn.

Assuming the camera is fixed at the center of car, the center of the image's width will, approximately, be the center of car. The mid-point of the lanes in perspective view will be the center of road. The difference between these two metrics will let us know how far the car is moving away from center. Here's the final output after unwarping the image on plotting on it the radius of curvature, distance from center and region of interest (region between lanes).

Pipeline (video)

1. Does the pipeline established with the test images work to process the video?

Yes, it does. With the help of moviePy each frame of the input video was passed to the pipeline and then it is processed and a video is generated. The output video is really good.

Discussion

The pipeline worked well on the project video. But when car shakes sometimes and this will make the region little wobble as I have not used a dynamic window selection for perspective transformation. This may be the reasons for the wobbling of lines, sometimes. The solution could be the dynamic window selection.

If the car changes lanes or is maneuvering around a sharp turn, the perspective transformation of this pipeline will not be able to capture both the lanes as the transformation window is small. An augmented Perspective transformation by using Left, Center and Right cameras may help at that time which can be used for the harder challenge video.