Self Driving Car Program

Advanced Lane line Finding Project

Advanced Lane Finding Project

Writeup / README

Camera Calibration

In the Camera Calibration part I took images from 'camera_cal' folder containing chessboard images, using those images i have calibrated my camera and found out the camera calibration matrix and the distortion matrix for the camera.

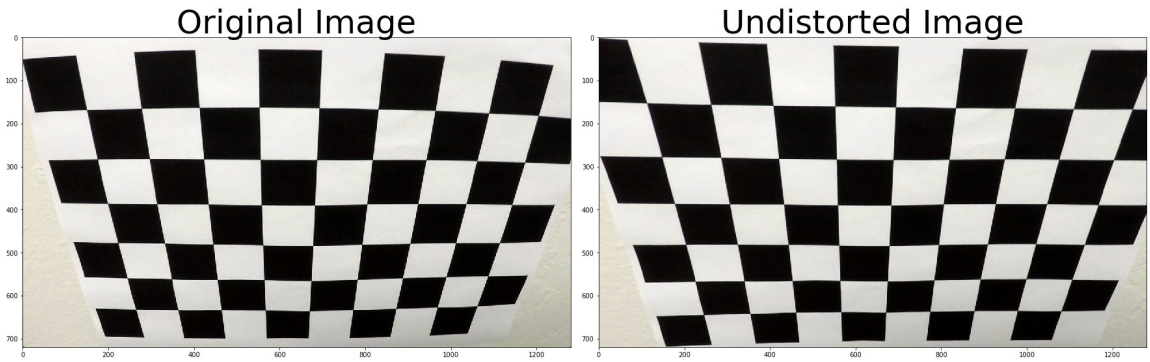
i started with calculating objpoints(which will be the (x, y, z) coordinates of the chessboard) and the imagepoints. Assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. For the lane lines which are curved we need images which are undistorted .In a distorted image the curve or the distortion in the images is undistorted once we have calculate the camera calibration matrix and the distortion matrix.

Distorted images can be easily corrected using the test images. For calculating camera calibration matrix and the distortion matrix i have first used cv2.findChessboardCorners() to find the corners in the chess board image. Using this i have calculated imgpoints and the objpoints for the chessboard images and then by using cv2.calibrateCamera() I calculated the calibration matrix 'mtx' and distortion matrix 'dist' and then applied cv2.undistort() to get the undistorted images.

 The same camera is used to record video from which the samples of chess board images were captured hence using the same calibration values for the test images.

The detail for camera calibration is in the 'Advance_lane_line_project4_carnd-term1.ipynb' jupyter notebook file.

Here is an example of output for one of the undisorted chessboard image :



Pipeline (single images)

Distortion correction on the test images:

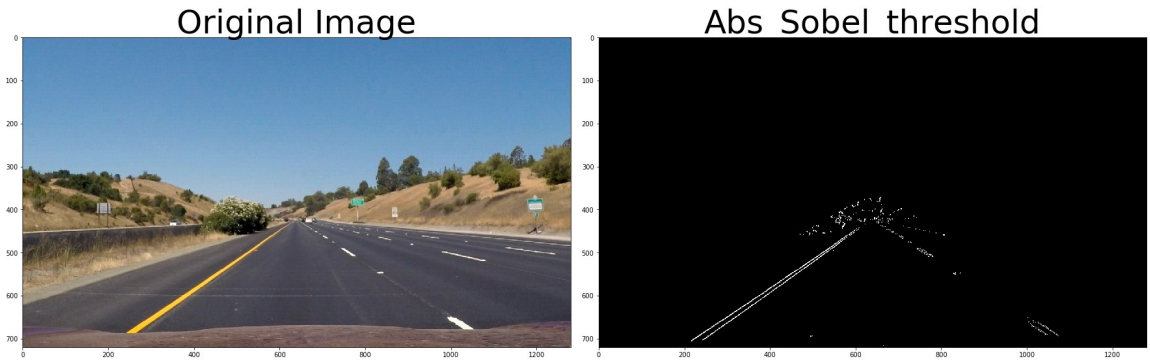
First i used the values used in chessboard images to obtain image points and object points, and then use the OpenCV functions cv2.calibrateCamera() and cv2.undistort() to compute the camera calibration and distortion matrix. The locations of the chessboard corners were used as input to the OpenCV function calibrateCamera to compute the camera calibration matrix and distortion coefficients. This camera matrix and distortion coefficients were used to remove distortion in the images.

Distortion correction applied on one of the test images after calibrating the camera is shown as :

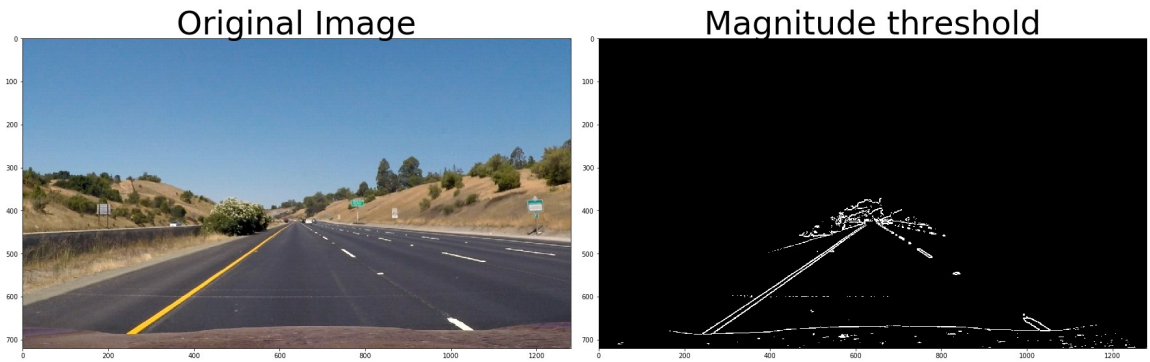
This is the image obtained after using the function cal_undistort() which is used to undistort an image an it returns an undistorted image.

Then I followed it up with different combination of color/gradient threshold calculations

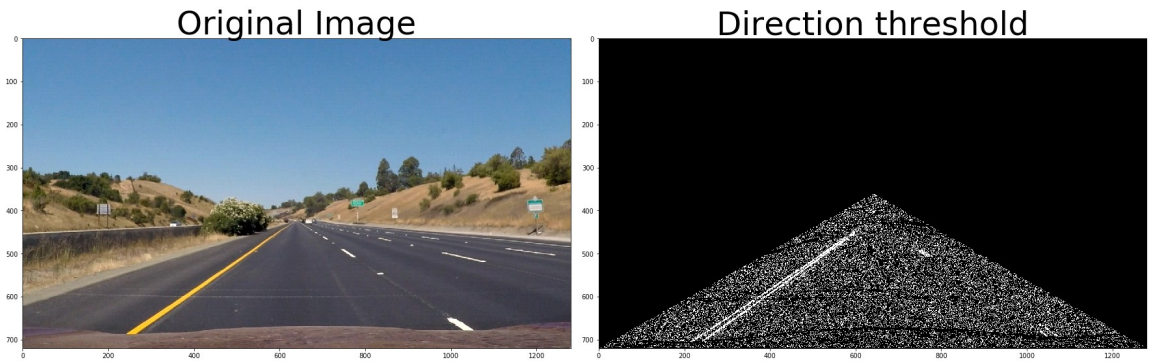
Calculating abs_sobel_threshold for one of the images provided and getting the binary output of the undistorted image calculated above:



Magnitude threshold output:



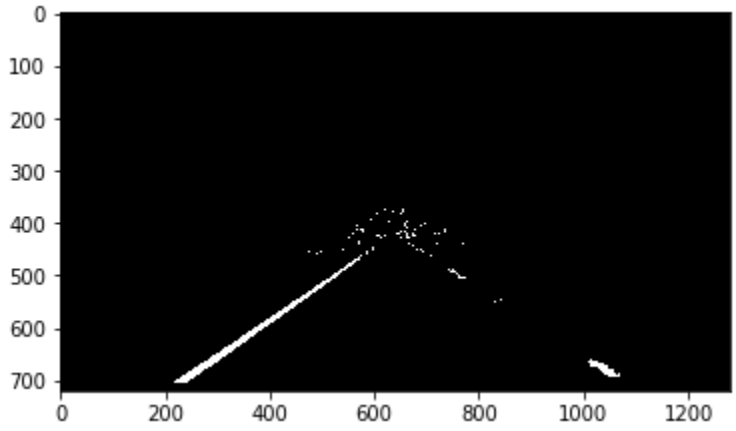
Direction threshold:



HLS threshold:

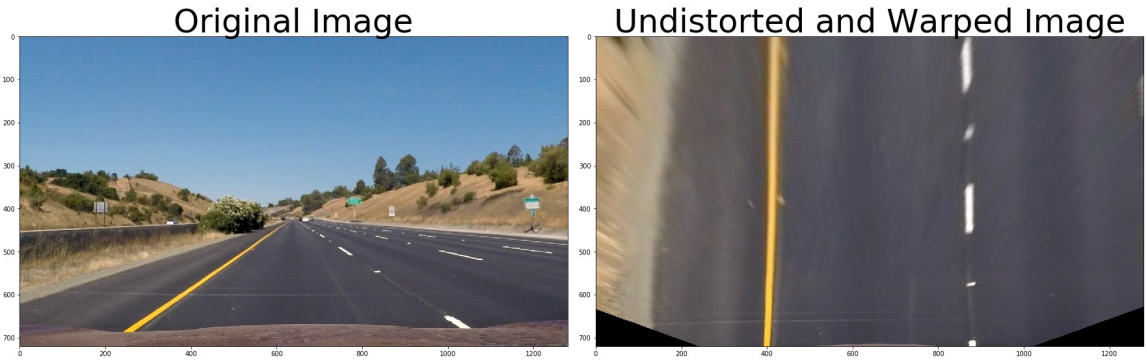
Then I tried with different combination of for Sobelx, sobely, HSL values and ultimately got the best outcome using the comination of sobel_x along with hsl binary values.

This is calculated in the function thresholded() which takes an undistorted image as input.The output thresholded image calculated :



Then I calculated the perspective transform of the thresholded image to get the 'Bird's Eye View' of the road in which only lane lines are focused and appear relatively parallel to each other using cv2.warpPerspective().

Below is the output obtained after taking wrapPerspctive of the undistorted image.



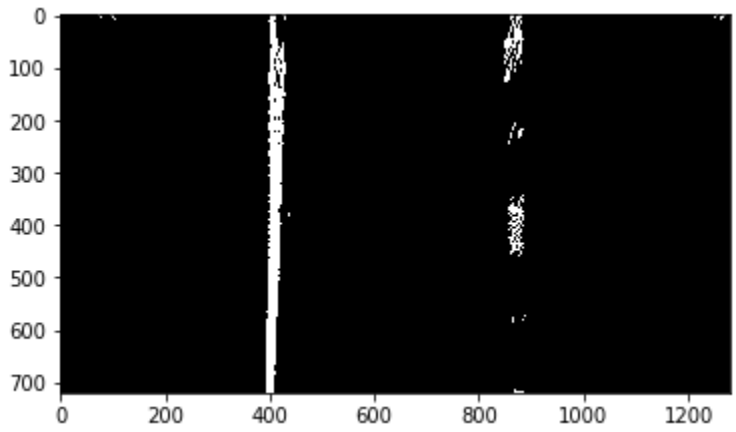
To calculate the perspective transform is under title "Warping the image" i have made function unwarp_img() which takes the undistorted image as an input.

This resulted in the following source and destination points:

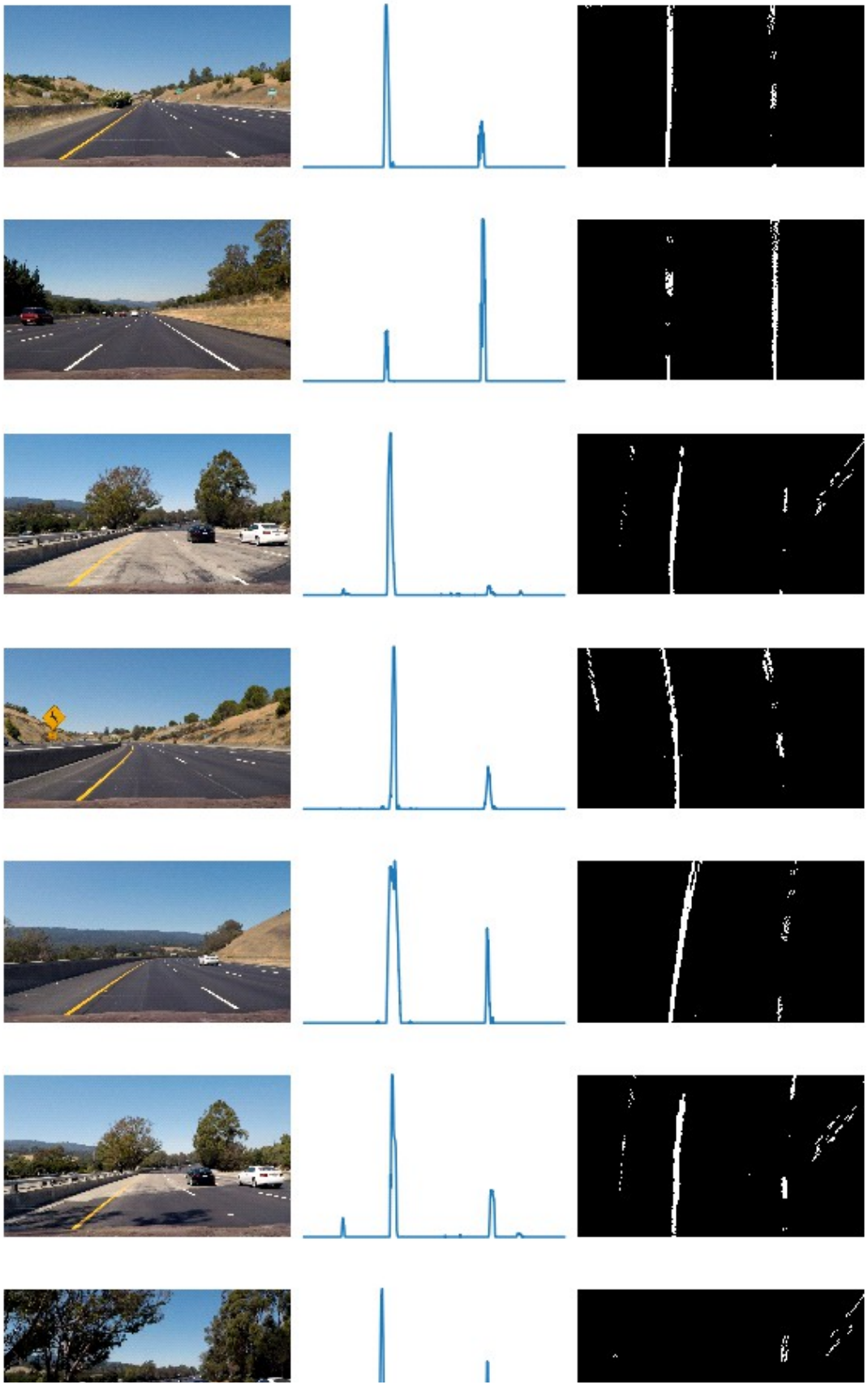
Source points	Destination points
(577, 460)	(dimsize,0)
(705, 460)	(w-dimsize,0)
(260, 684)	(dimsize,h)
(1050, 684)	(w-dimsize,h)

(here dimsize i have taken as 400, w and h are the width and height of the image which are 1280x720)

Also I verified using a test images and they were looking fine. Below is one of the example of how warped image looks like:



At the end of the wrap_img function i got all the pixels where the binary value is 1 and plotted as in the image above, I took histogram of the thresholded image along all the columns and on the basis of that, I identified those pixels using euqation f(y) = A.x** + B*y + c. I have used this equation for left and right lane lines. These all calculations are done under the function pipeline(). Below are the histograms for test images along with their unwrap 'bird eye view' image':



I reffered to the classroom materials to calculate radius of curvature of the lane. Below is the formula which i used for this:

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

Then I also calculated the curve radius for both left and right lane lines and calculated average of them.

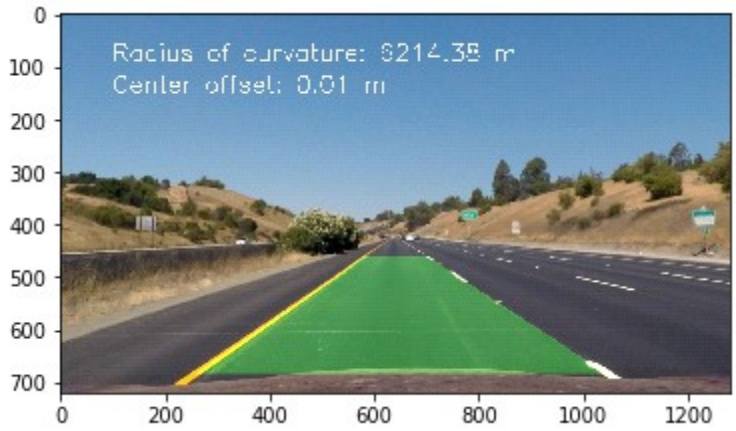
```
left_curve_rad = return_radius_of_curvature(left_fitx)

right_curve_rad = return_radius_of_curvature(right_fitx)

average_curve_rad = (left_curve_rad + right_curve_rad)/2
```

With the histogram I added 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. I got all non zero pixels around histogram peaks using the numpy function numpy.nonzero().

Fitting a polynomial to each lane i used numpy function numpy.polyfit().

After fitting the polynomials I was able to calculate the position of the vehicle with respect to center.

At the end of the the pipeline I printed the inverse perspective transform for the image resulting in the below image:



Pipeline (video)

The final output video '**project_video_out_test.mp4**' is in the '**output_video_final'** folder.

Discussion

I had to experiment a lot with gradient and color channnel thresholding specially, in the areas with brighter lightings campared to most part of the road. Earlier I was not getting the correct areas covered in the brighter patches of the road so, to overcome that i took hls image thresholding for light and saturation with threshold values of 120 and 150. then i combined it with sobel_x thesholding and also followed up with the regional masking to ommit all the unwanted pixels which i wanted to discard as they were of no importance .

I checked my pipeline with a number of possible cominations of threshold values to achive a overall working pipeline.

The video pipeline in this project did a fairly robust job of detecting the lane lines in the test video provided for the project, which shows a road in basically ideal conditions, with fairly distinct lane lines, and on a clear day.

It is relatively easy to finetune a software pipeline to work well for consistent road and weather conditions, but what is challenging is finding a single combination which produces the same quality result in any condition is difficult.So, in different condtions of roads with either no lane lines or missing lane lines for a longer duration may find a bit difficult in plotting the correct color fills.