Example Writeup

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to submit a pdf if you prefer.

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.

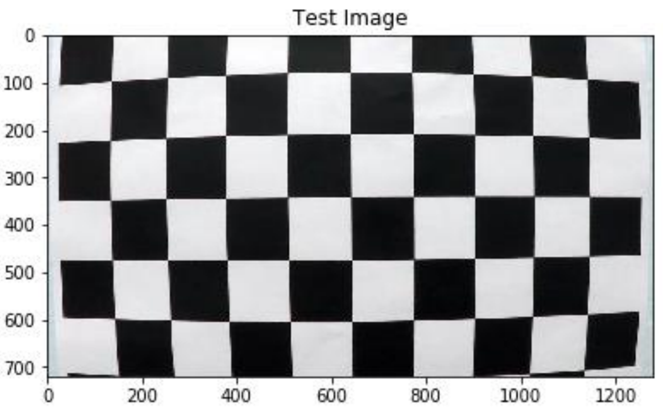
Rubric Points

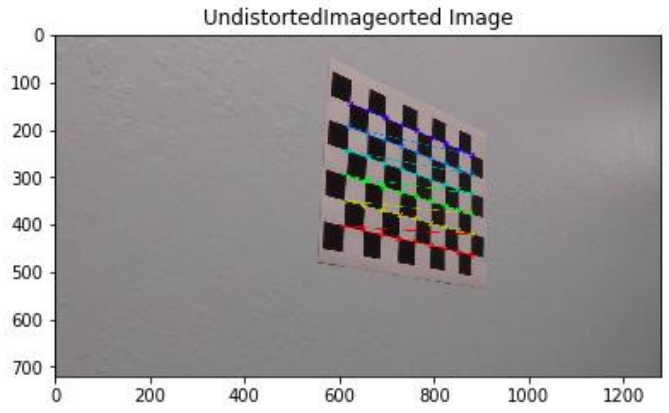
Camera Calibration

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 second code cell In [2]: of the IPython notebook called AdvanceCarLaneFinding.ipynb I used the code from the lesson that start by preparing "object points", which will be the (6, 9, 3) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (6, 9) 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 (6, 9) 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:

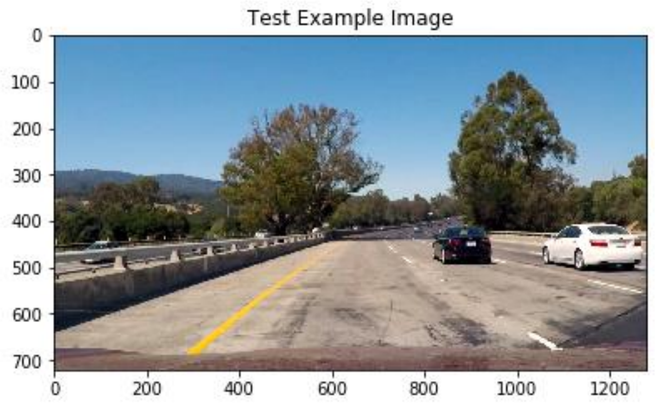


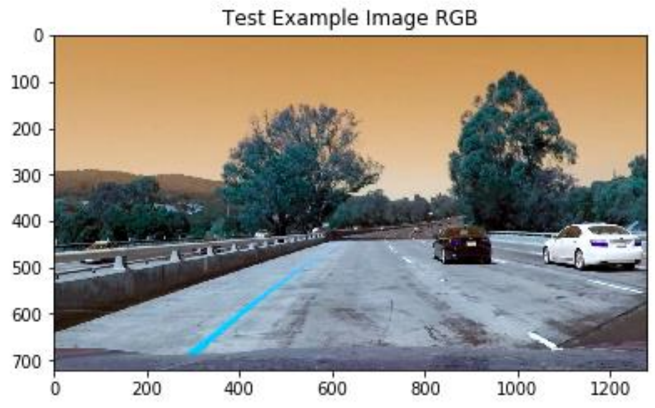


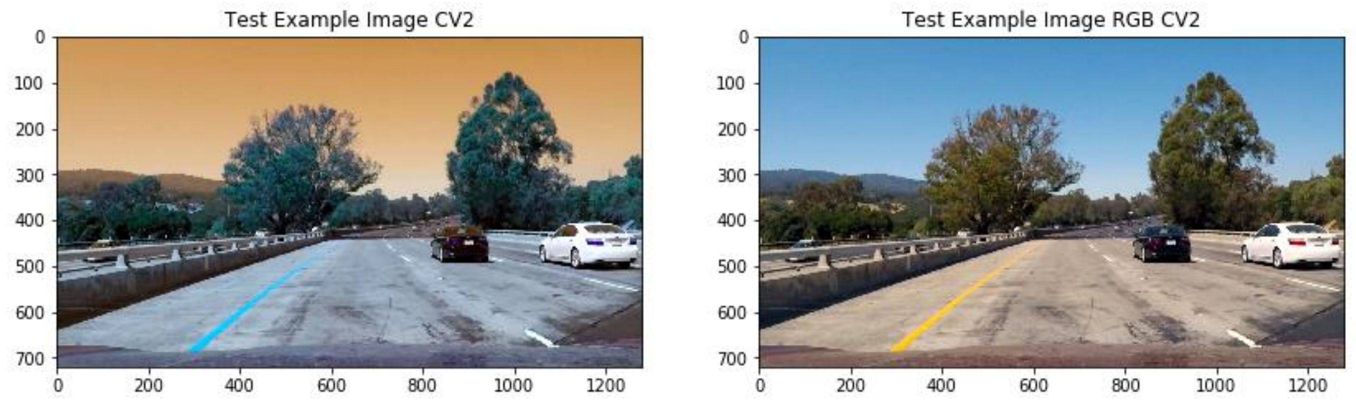
Pipeline (single images)

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

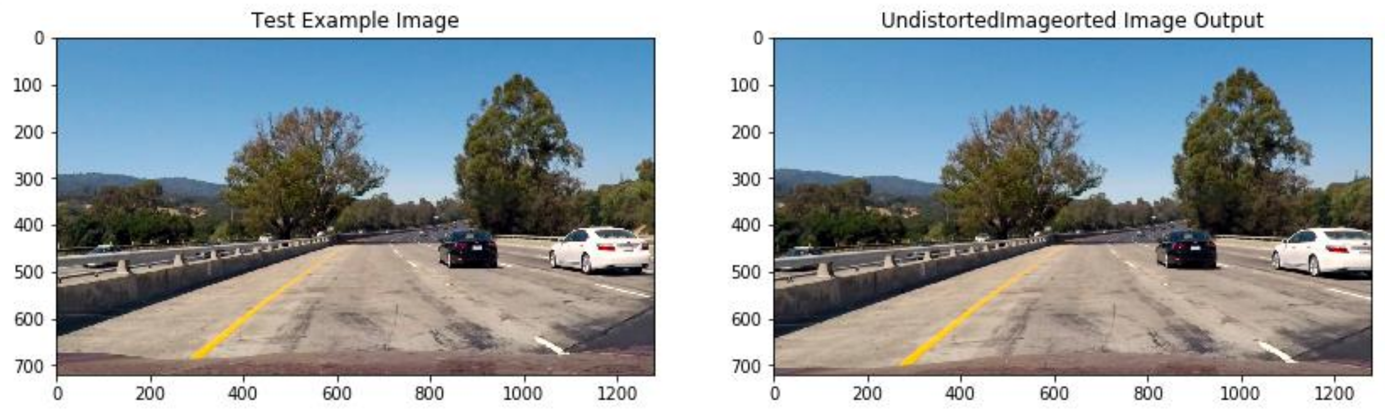
To demonstrate this step, following is the description of how I apply the distortion correction to one of the test images like below. Test Image is taken from the give set of test images, the RGB filter is applied below. First time the Image is read with `matplotlib.image` and the second time using the `cv2.imread` function the final output in RGB filter is achieved by using the `cv2.cvtColor`





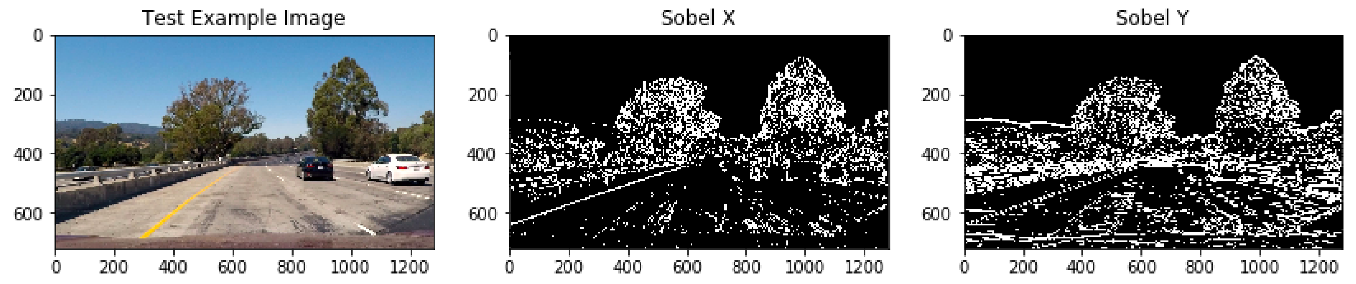


The undistorted image is created using the `cv2.calibrate` Camera and `cv2.undistort` function.

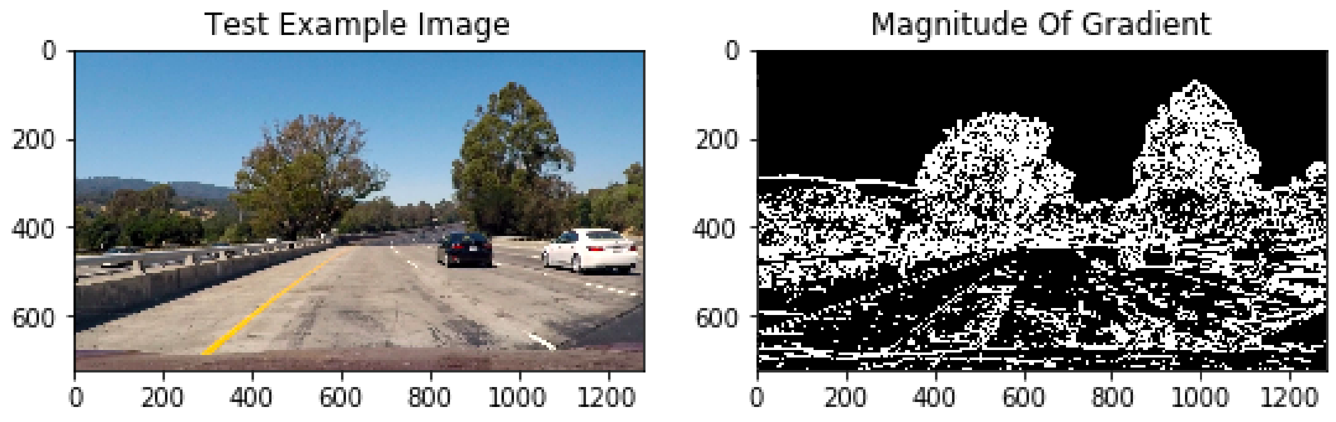


2. Has a binary image been created using color transforms, gradients or other methods?

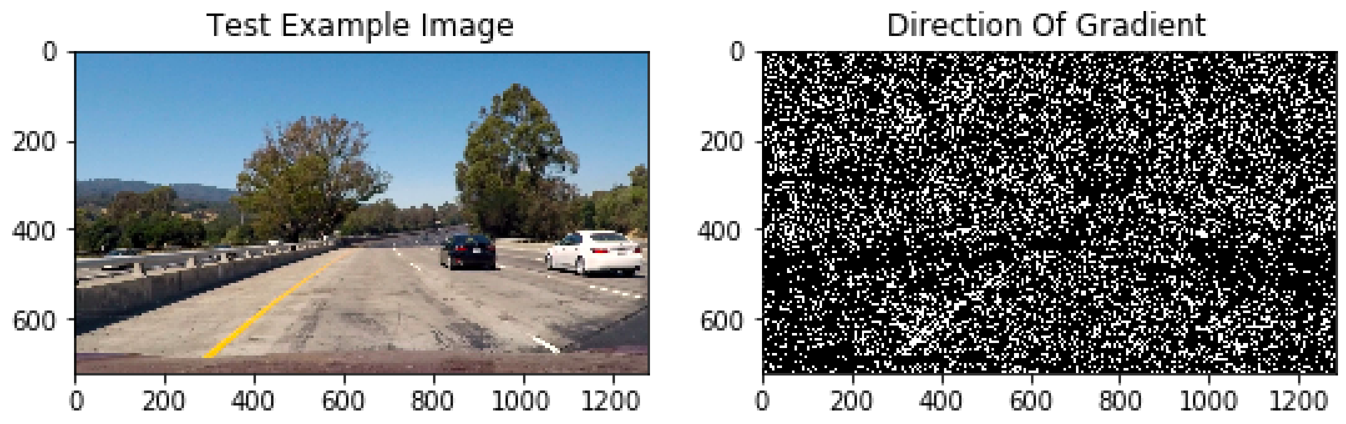
Test Image and the the images after applying the Absolute Sobel Filter. As the Sobel applys the gradient in X and Y Direction



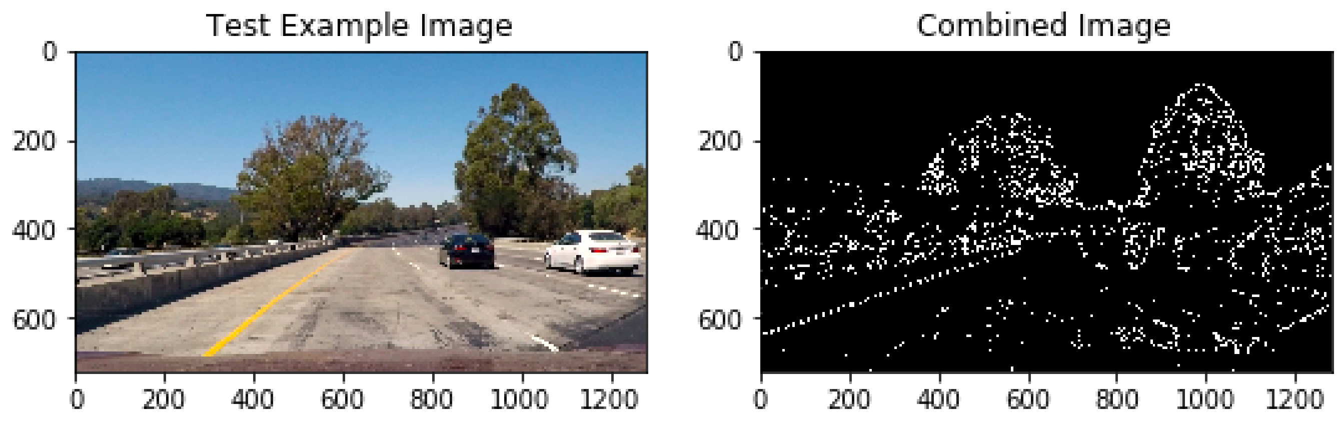
The mangnitued of gradient is applied to the image by converting the image to gray sacle using the `cv2.cvtColor` function and then using the `cv2.Sobel` function the gradient in X & Y are saperated using then the magnitude is calculated.



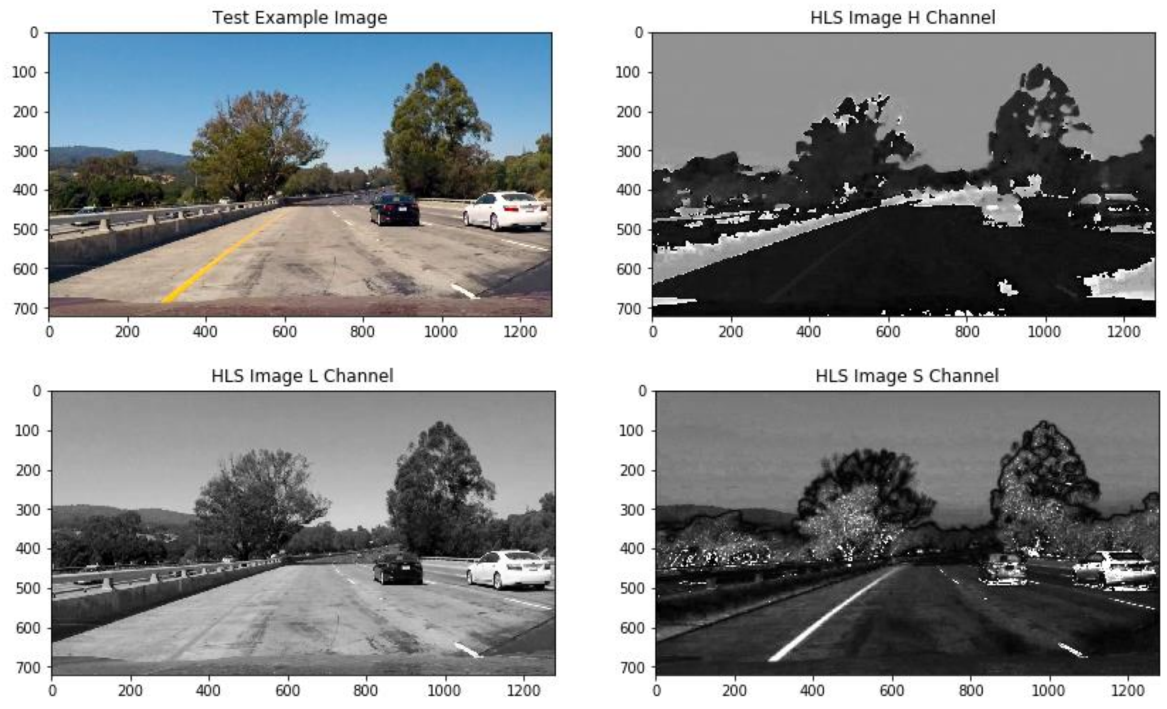
After Magnitude of Gradient calculated the Gradient Direction is calculated

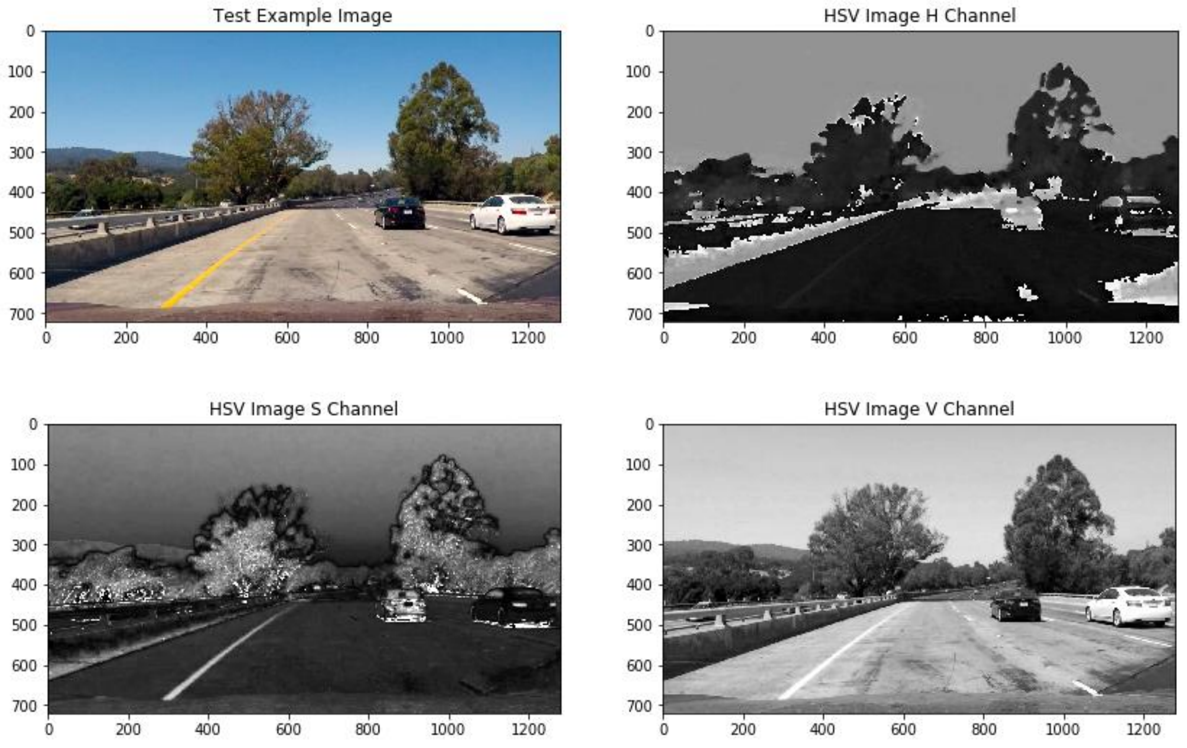


The final output image is as follows after applying all three above filters on the test image.

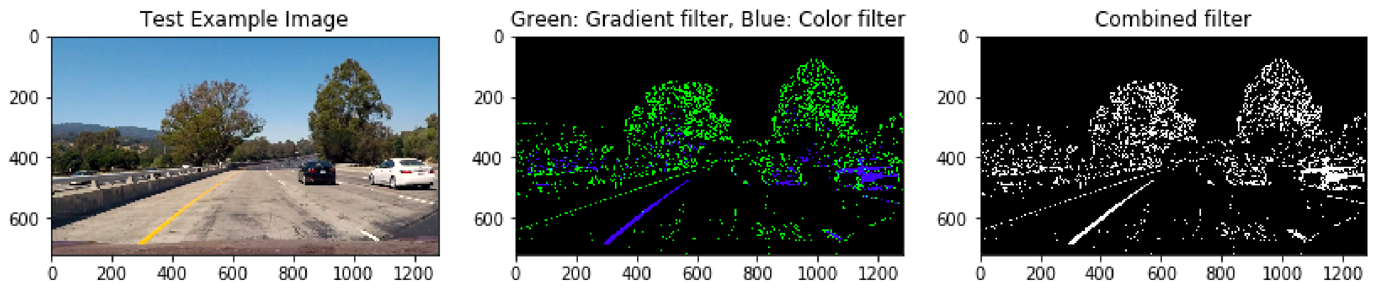


Then the HLS and HSV filters are applied using the `cv2.cvtColor(<IMAGE>, cv2.COLOR_RGB2HLS)` function.



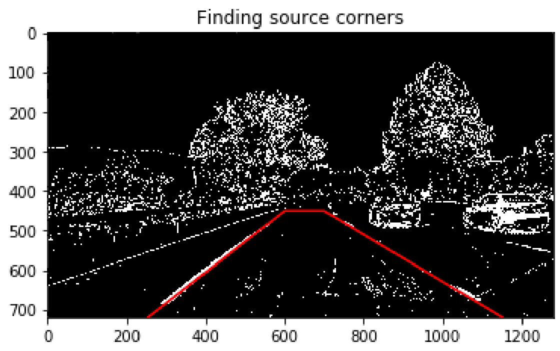


The final output with the Gradient filter and the combine filter.



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

The code for my perspective transform is includes a function called `CornersUnwarp`, which appears in code block In [18] in the IPython notebook called AdvanceCarLaneFinding.ipynb



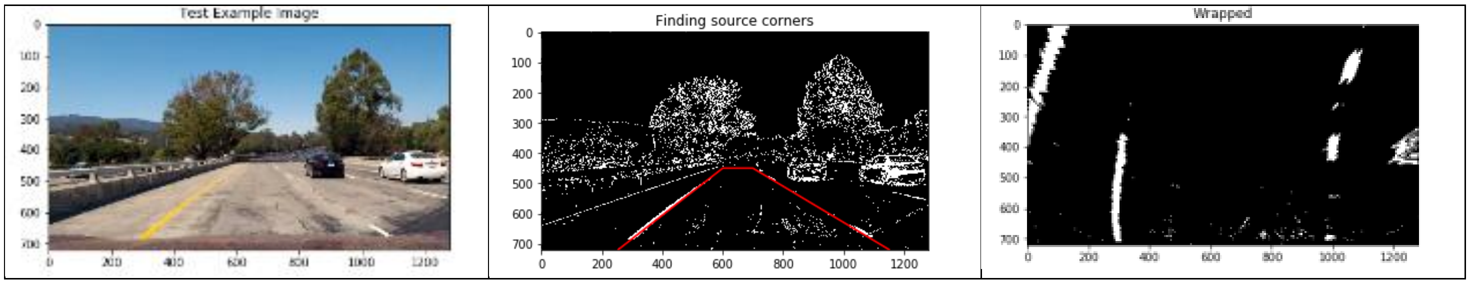
The "TransformPerspective" function uses the (`src`) and destination (`dst`) points to get the wrapped image. I chose the hardcode the source and destination points in the following manner:

```
 src = np.float32(
      [[600, 450],
       [700), 450],
       [1150, 719],
       [255, 719]])
# offset for dst points
 OffsetX = 300
 OffsetY = 0
 dst = np.float32([
        [OffsetX, OffsetY],  # top left
        [ImageSize[0] - OffsetX, OffsetY],  # top right
        [ImageSize[0] - OffsetX, ImageSize[1] - OffsetY], # bottom right
        [OffsetX, ImageSize[1] - OffsetY]  # bottom left
    ])
```

This resulted in the following source and destination points:

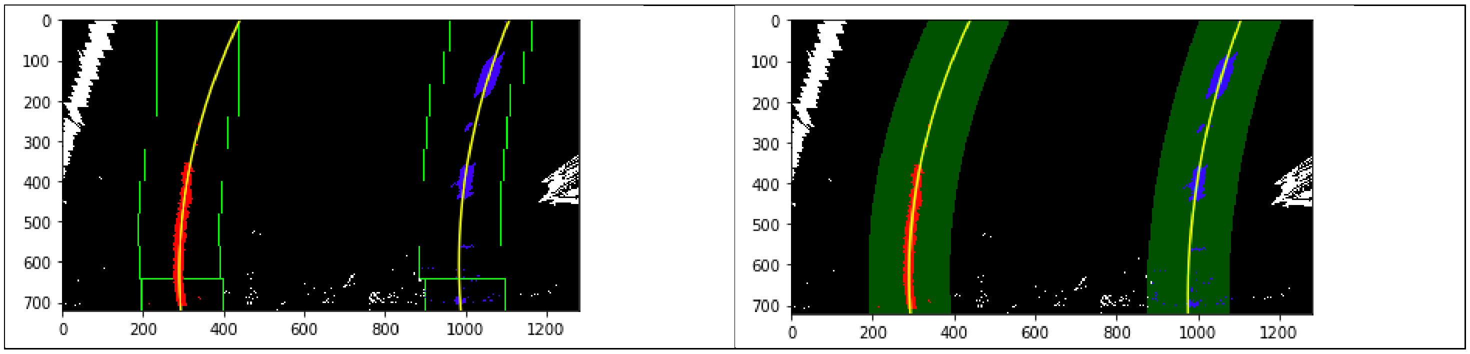
Source	Destination
600, 450	300, 0
700, 450	980, 0
1150, 719	980, 720
255, 719	300, 720

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.



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

Then the FindLanes and TheFindLanes function performs the fit my lane lines with a 2nd order polynomial sliding window technique.



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?

Yes have done that in the code In[41], In[45], In[48]

Pipeline (video)

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

The Video file is in the location ./TestOutputVideo.mp4

README

1. Has a README file been included that describes in detail the steps taken to construct the pipeline, techniques used, areas where improvements could be made?

This is the Readme file.

Discussion

Following are the key steps that are used to create the pipeline and get the right output.

First step is to get the Undistorted input image using the “`cv2.undistort`” function this task is accomplished. Then I apply the gradient and color filter using the custom function “`ApplyGradColorFilters`”. The next task is to apply the respective transform which is accomplished using the custom function “`TransformPerspective`”. The custom function “`FindLanes`” allows the lane detection using the sliding windows and the “`CalculateRadiusOfCurvature`” function calculates the curvature of the lane.

I think the functions can be further tweaked to get the more accurate results so it works better on the challenge video. I tried running the current function and based on the results I feel the source corners needs to be tweaked so that the trapezoid fits the lane correctly in challenge video.