1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

Here it is :)

Camera Calibration

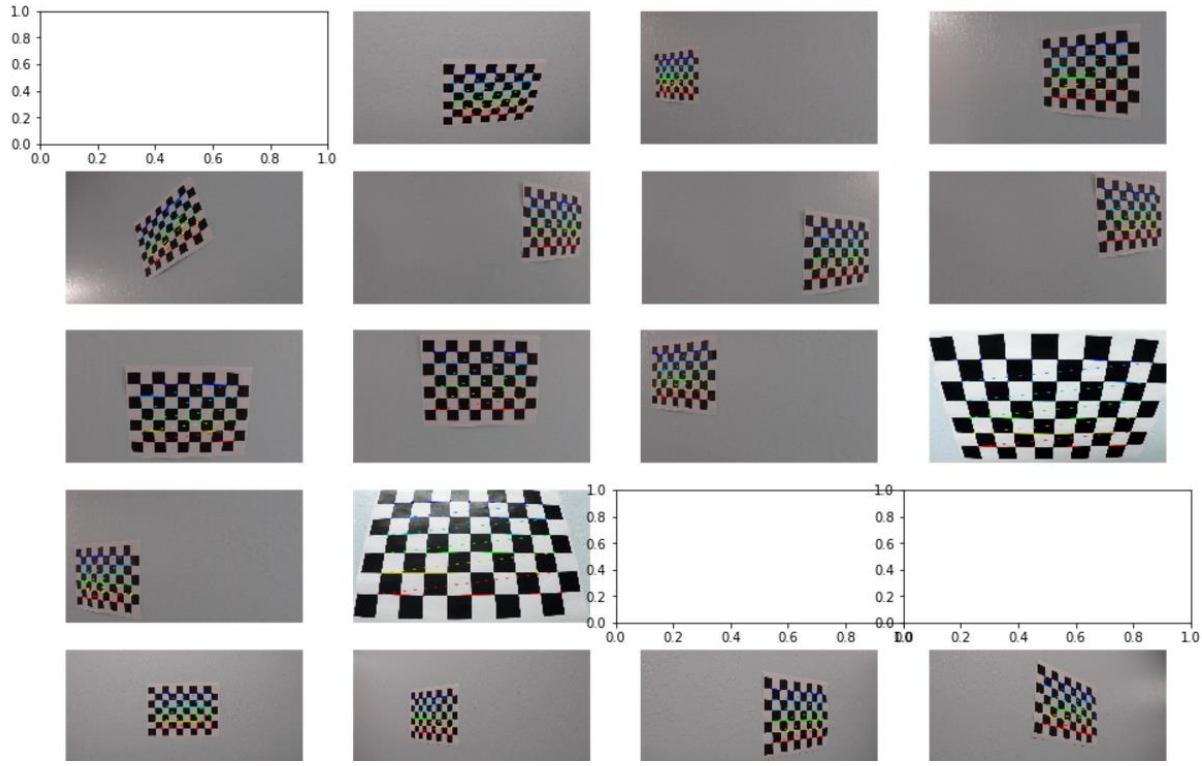
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

I created to variables. One for the 3D coordinates points `objp` of the a chessboard that our camera took pictures of it in different angle views. And the other variable is the 2D coordinates of the corners of the chessboard in the image `imgpoints,` then I assumed that the object points are the corners of the chessboard when the z coordinates = 0. The x and y coordinates can be known from the real dimensions of the chessboard. However the image points will be estimated by the OpenCV command `cv2.findChessboardCorners`

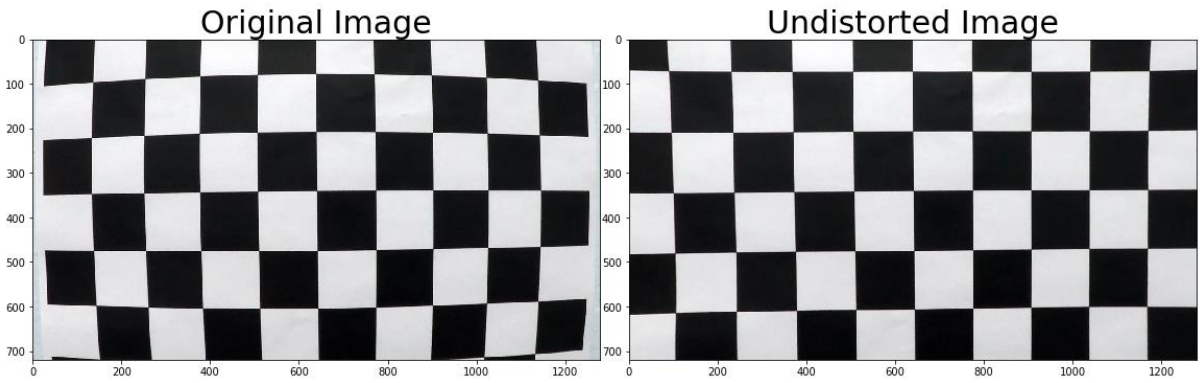
Then we have used these coordinates to compute the camera calibration using the OpenCV command `cv2.calibrateCamera()`

Then the cv2.undistort() function will apply image distortion correction on all our images before carrying any other processing to correct for the lens distortion of the camera.

Below is the images of the chessboard in different point of views and OpenCV is finding out their corners. Few of the images are not shwn since OpenCV failed to find their corners. Just a few images are enough to estimate the distortion matrix.



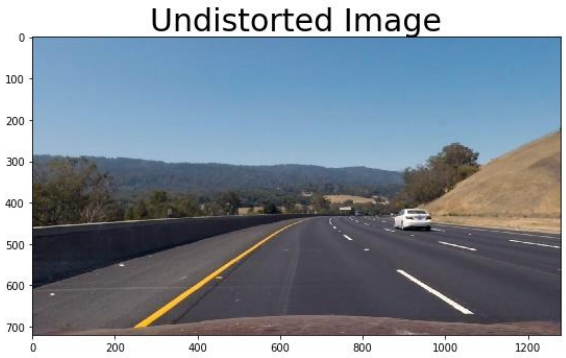
And here is an example to undistort one of the chessboard images.



Pipeline (single images)

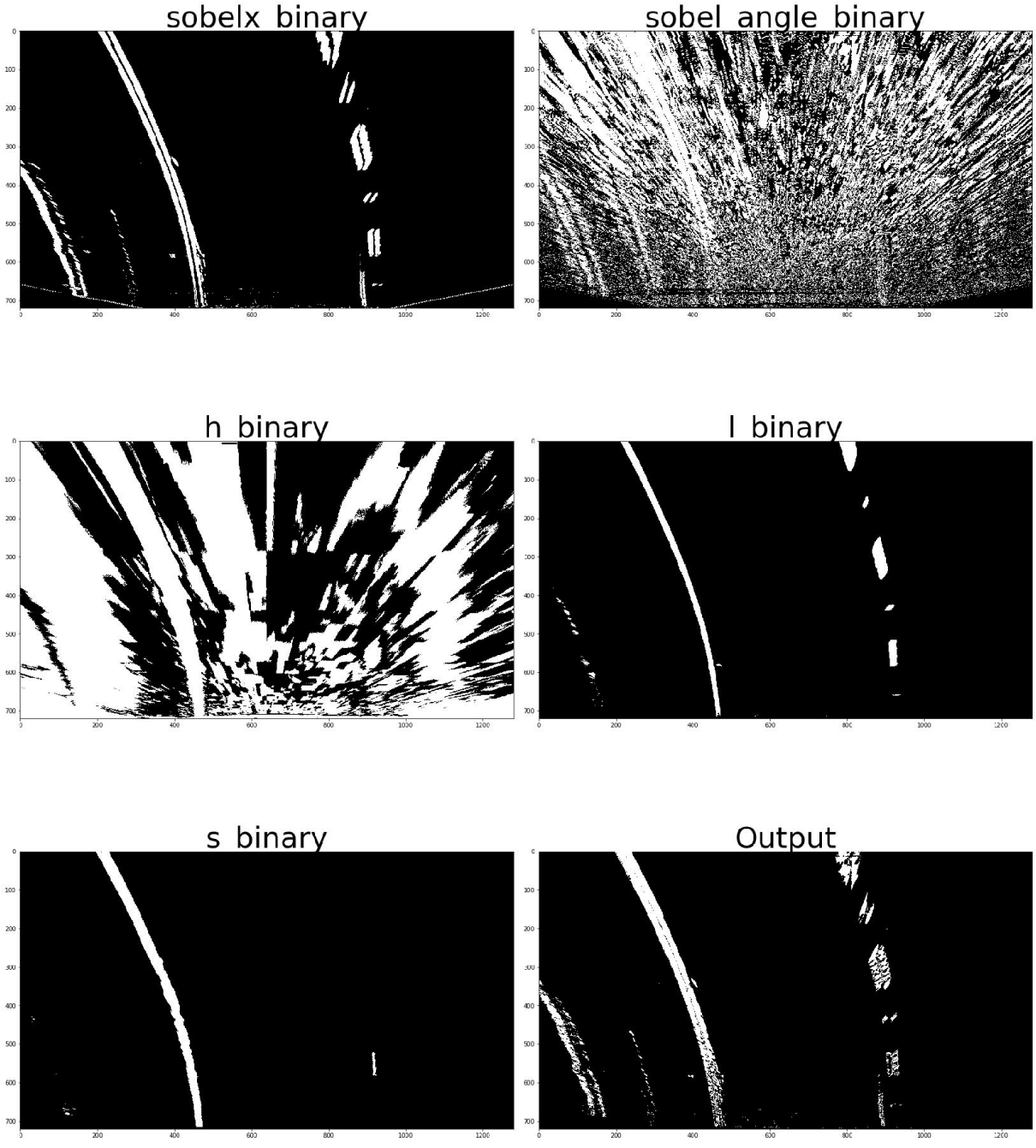
1. Provide an example of a distortion-corrected image.

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



2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image. HLS clor space was used and the x-axis sobel gradients. Here are examples of my output for this step.



3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

I used the OpenCV function called `perspective_transform` that takes an input image and the source `(src)` and destination points `(dst)` and performs the perspective transform. The source and the destination points for our project is as the following:

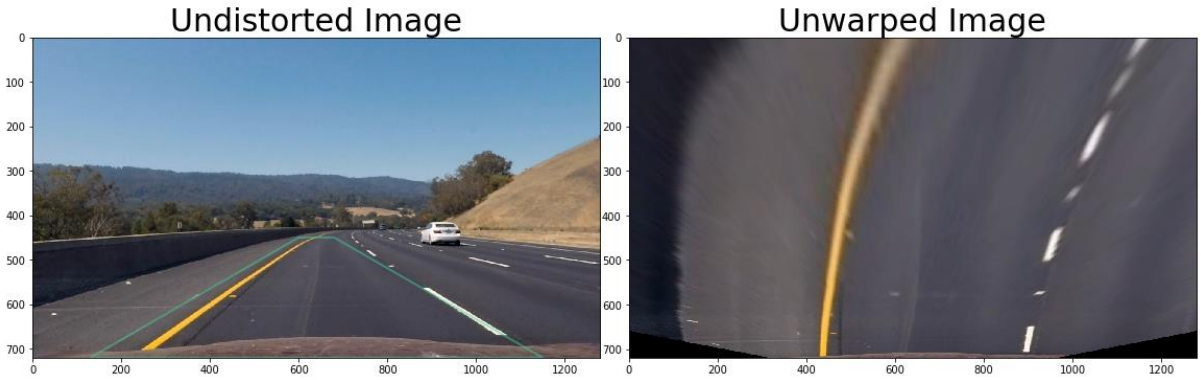
```
src = np.float32([(600,450),
                  (680,450),
                  (1150,720),
                  (130,720)])
dst = np.float32([(385,0),
                  (895,0),
                  (895,720),
                  (385,720)])
```

This resulted in the following source and destination points:

Source	Destination
600,450	385,0
680,450	895,0
1150,720	895,720
130,720	385,720

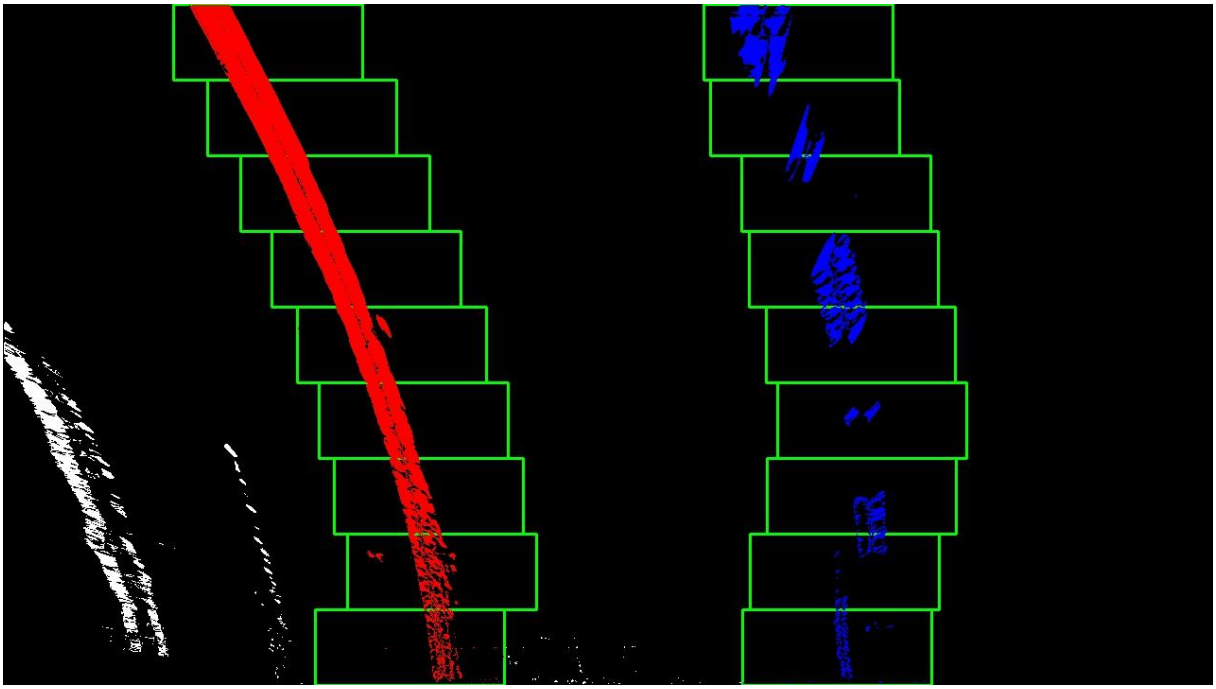
The code is in my `P4_Submission.ipynb` under the markdown comment **`'Perspective transform'`**

I verified that this transform is working by drawing the `src` and `dst` points onto the test image and warp it to check the results like below:

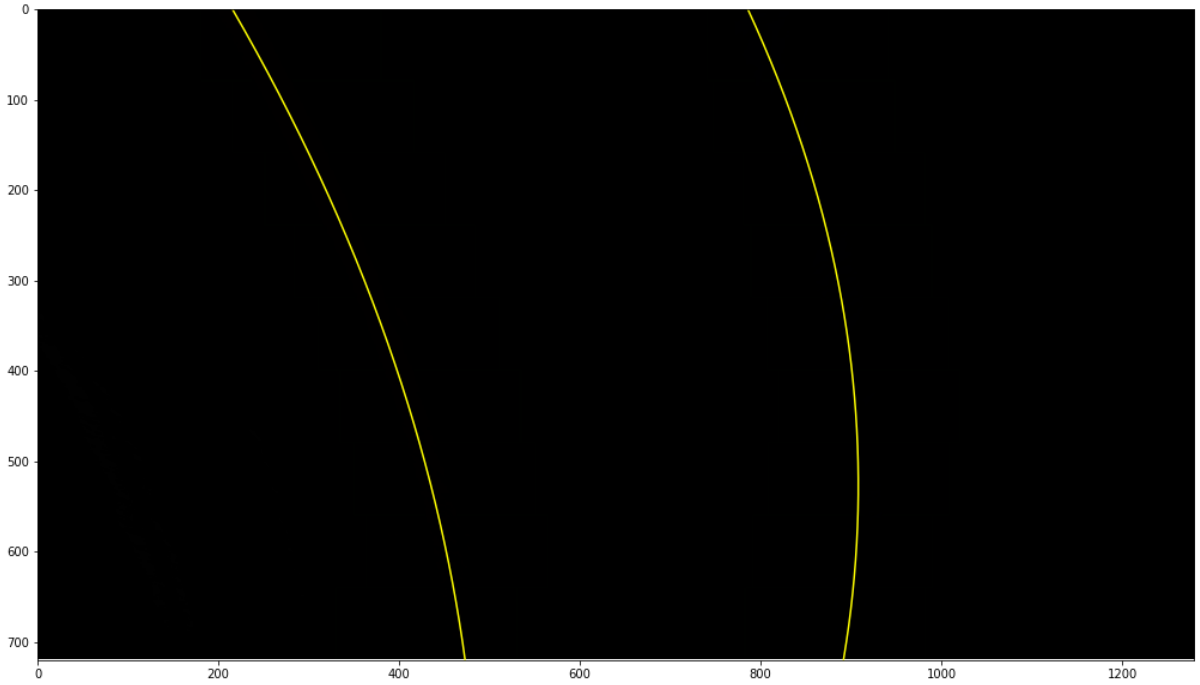


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

First, I checked the histogram of the lower part of the image and finding out the two peaks for the left and right lanes. Then I used the sliding window method to go upwards and finding the relevant points in the image which mark the lane. Then I used the `np.polyfit()` method to fit a second degree polynomial to these points. I did this in the `fit_polynomials()` function under the markdown comment '**Sliding Window and Polynomial Fit'.** Here is a picture showing this process on the test image:



And here is the estimation of the lanes using polygon fit:



5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

After determining the second degree polynomial that fits the estimated points, the radius of curvature is calculated of this polynomial. In addition, I had to convert from the pixel space to real space in meters. I did this in `get_curvature()`function.

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I implemented this step in the function `render_lane`

Below is an example of my result on a test image showing the radius of curvature of the lane and the vehicle offset from lane center at the top left of the image. And at the top right, I put 2 diagnostic images to show how the code is seeing the road which are interesting to see when the code estimation is going wrong on some difficult points on the road. I added also an arrow showing the width of the lane, which can be used as a sanity check whether the estimation of the lanes position is correct or not, by checking whether the width of the lane within acceptable range. Of course this is just one way of checkup and it is not immune to false positive result of sanity check even when the two lanes are equally deviated from the correct position,

however this is unlikely usually, and when something get wrong the distance between the two sides of the lane will be outside of the normal range.

:

Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

Here is my video result:

https://www.youtube.com/watch?v=-009iRWXidY

Discussion

1. 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?

This method is extremely sensitive to the tuning parameters of the gradient thresholds and the color thresholds. Now I really appreciate the deep learning method over the computer vision method.

Most of the problems of this method was due to the changes in the lighting of the scene, shadows, discoloration. In addition, any pedestrian or vehicle in front of the camera will likely make this method fail.

The HLS color space helped a lot in isolating the yellow and white colors of the lanes even when the lighting is changing. I think when the road is full of snow; this will be a real problem for this algorithm.

Checking out the width of the lane, I think, is an important step in sanity check. At least if the width estimation is unrealistic the algorithm is able to tell that there is something wrong, and perhaps switches to another method to direct the autonomous vehicle.