Writeup Template

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 use some other method and 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

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

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.

You're reading it!

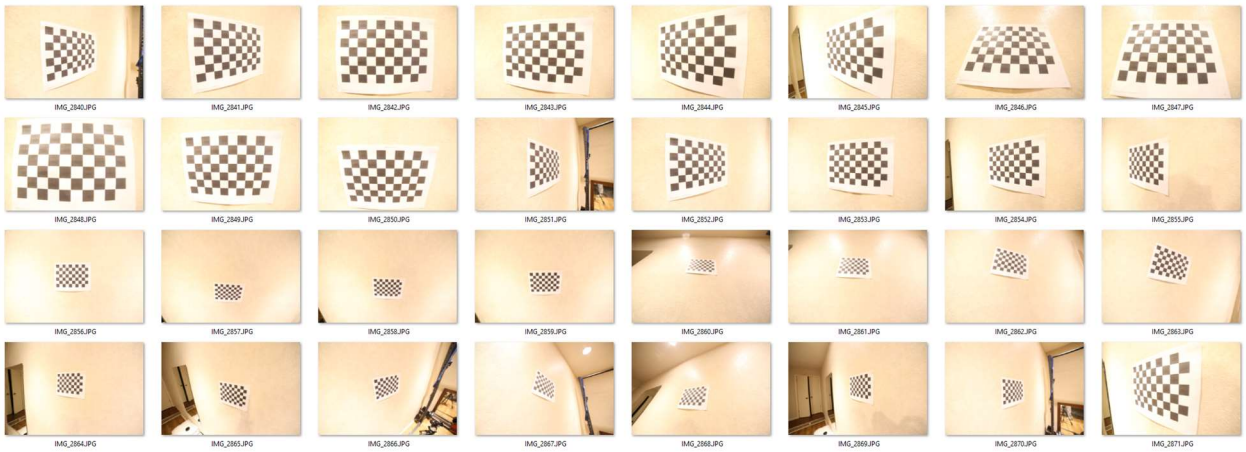
Camera Calibration

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

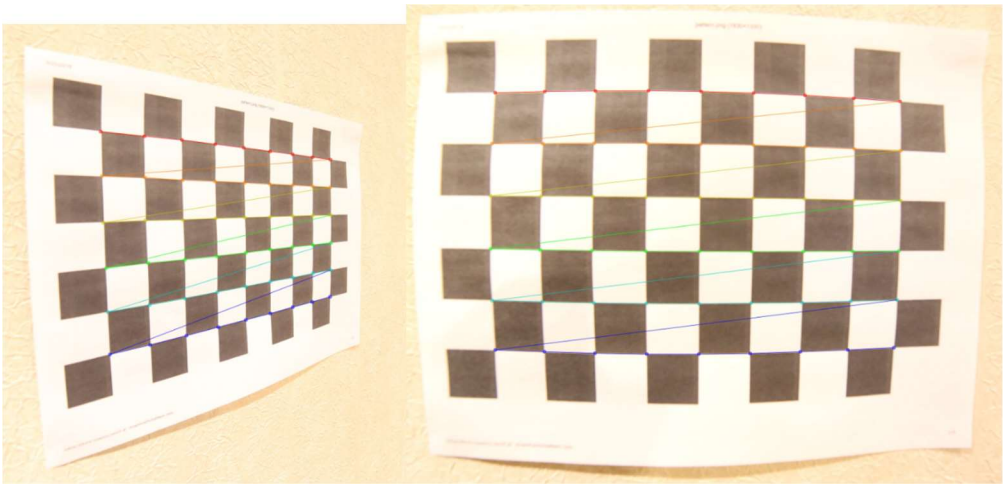
I was in Sweden for a work project first two weeks of March'18, driving our self driving car on snow!! But more related to this camera calibration: In Sweden I used a 14 mm lens to capture the northern lights. This lens has very pronounced distortion so I was working on

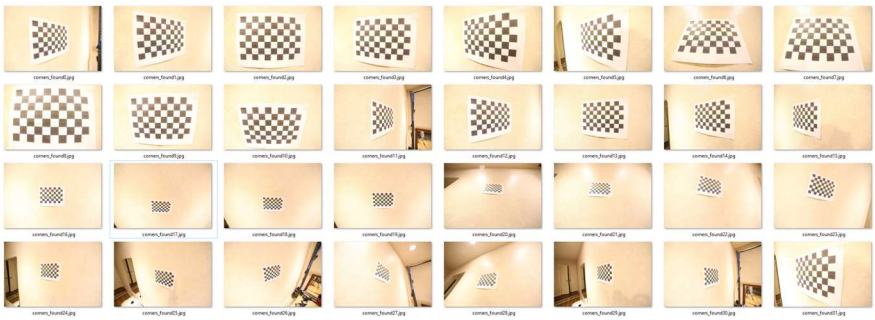
how to correct that using Lightroom. And lo and behold, as you might have noticed, I am also doing a Udacity course that just so happened to be teaching camera calibration.

The camera I use is a Canon 6D with Rokinon 14 mm lens. So after I returned to the US, I took some images with my camera of the chess board pattern.

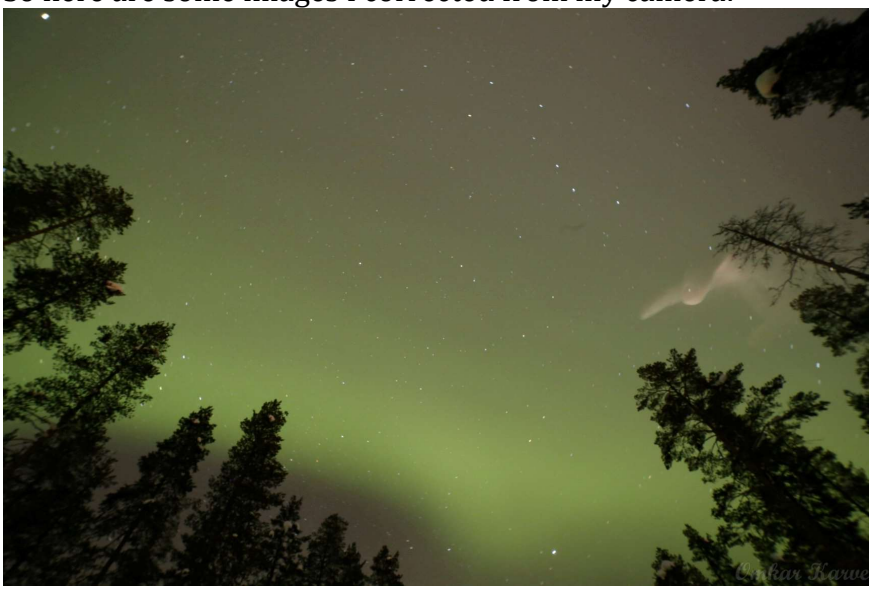


Then I applied the calibration steps as outlined in the “Canon6D_camera_calibration.pdf” Jupyter notebook. Here are the results of corner finds:





So here are some images I corrected from my camera:

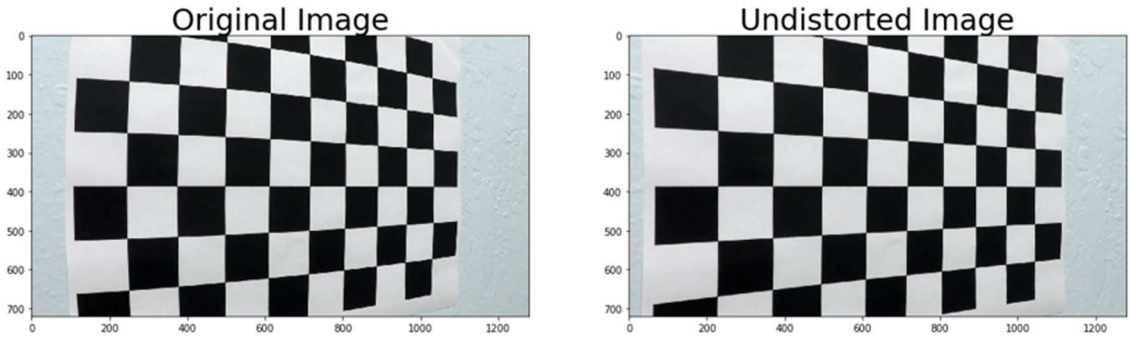


Original Image

Undistorted Image

More related to the project. I followed similar path in doing camera calibration for the images given in the project database for camera calibration. This can be seen in the jupyter notebook titled “01. Camera calibration.html”

```
Out[29]: <matplotlib.text.Text at 0x213dcf42198>
```



alt text

Pipeline (single images)

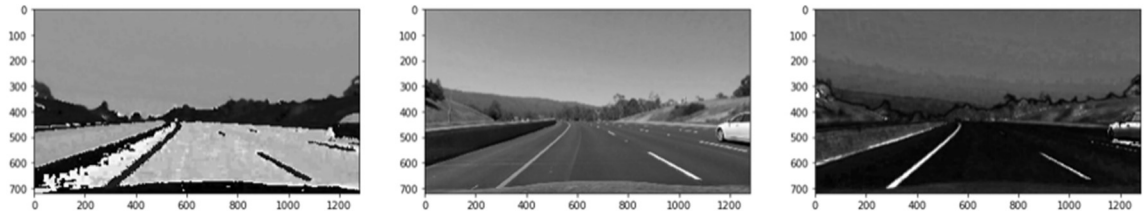
1. Provide an example of a distortion-corrected image.

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

Here the undistorted image is shown in its three different channels.

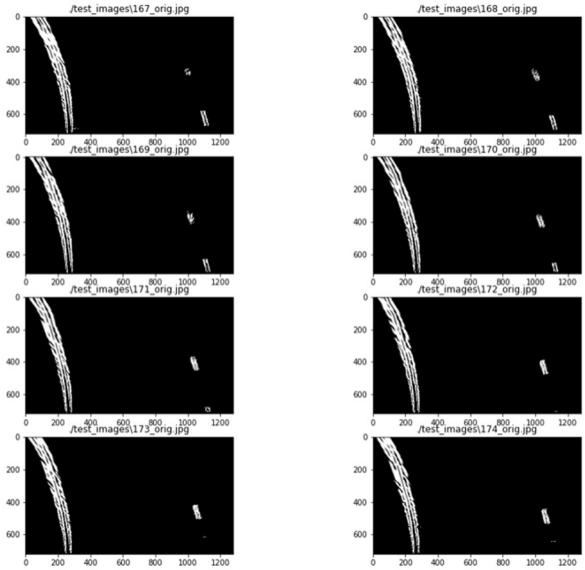
```
In [7]:
# Undistort and change color space to HLS
def undistandHLS(img, mtx=mtx,dist=dist):
    undist=cv2.undistort(img,mtx,dist,None,mtx)
    return cv2.cvtColor(undist,cv2.COLOR_RGB2HLS)

hls_test_img=undistandHLS(test_images[1])
fig, axes = plt.subplots(ncols=3, figsize=(20,10))
for index, a in enumerate(axes):
    a.imshow(hls_test_img[:,:,index], cmap='gray')
```



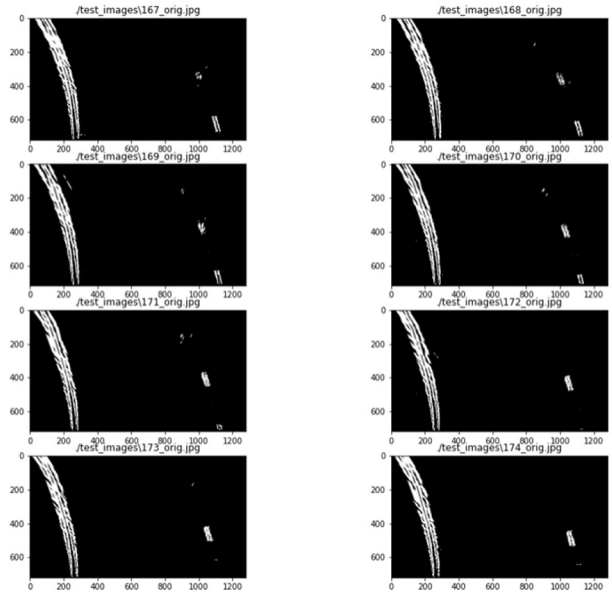
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

Here I started off by only using the L channel of the image (after converting the image from RGB to HLS format using CV2). The eventual result of this was a processed video that resulted in the right lane not being recognized properly. It is seen in the video titled "project_video_processed_9windows.mp4". I worked out that this was because just using the L-channel was working only on about 90% of the images in the video:

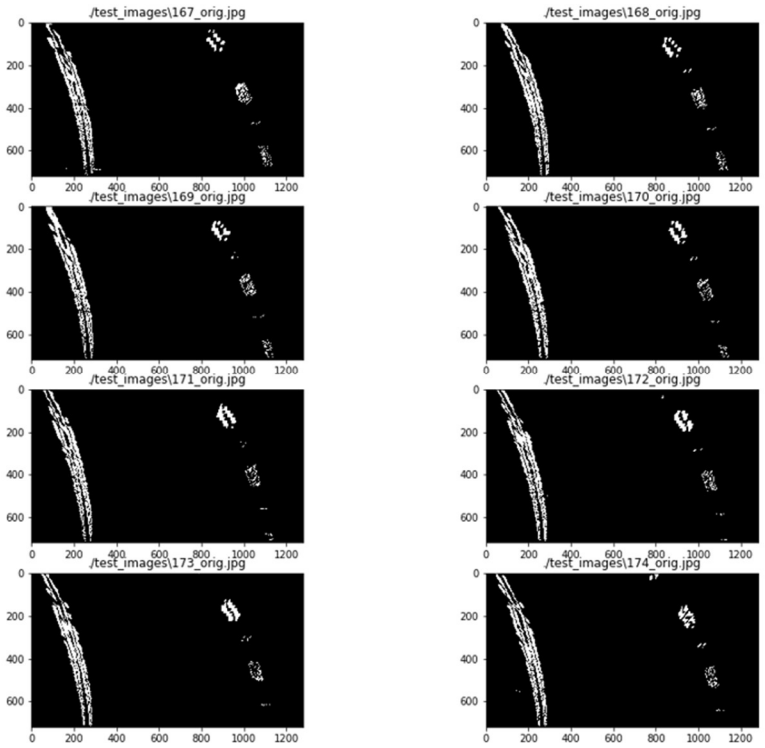


The images shown here are the frames that I thought were more challenging than the test images that were given as a part of database. So the image number is the frame number from the video. Here you can see the right side lane is not being picked up that will give a reliable polyfit result.

So I experimented with using the S channel only. Here are the results:



As can be seen there is a slight improvement. But I the video output using just the S channel results in missing the lane markings where the pavement is of lighter shade (such as on the bridge). So eventually, after playing around with the thresholds for S and L channel, I settled on combining the output from thresholded S and L channel combined. The output is shown below:



This has much improved right lane recognition.

In the Jupyter notebook, the functions gradients, sobelimg, thresholdimg do this job.

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

I tested my perspective transform function in a Jupyter notebook called “03. Perspective transform”. Before passing the image to perspective transformation function, it was undistorted using the parameters worked out in step 2 above.

The source and destination grid grid used:

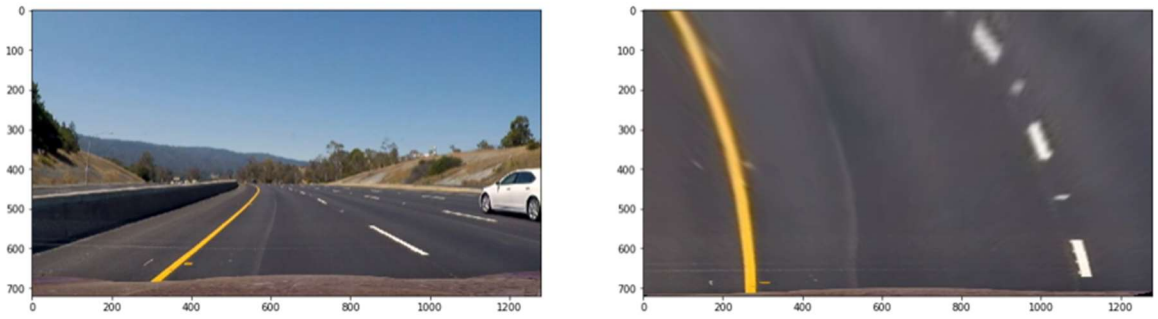
```
[[ 585.  455.]          [[ 200.    0.]
 [ 705.  455.]           [1080.    0.]
 [1130.  720.]           [1080.  720.]
 [ 190.  720.]]          [ 200.  720.]]
```

After applying the warpPerspective CV2 function:

```
M = cv2.getPerspectiveTransform(src, dst)
Minv = cv2.getPerspectiveTransform(dst, src)
warped = cv2.warpPerspective(undist, M, img_size)
```

Following are the results of the perspective transform:

```
Out[6]: <matplotlib.image.AxesImage at 0x1e00866ea90>
```

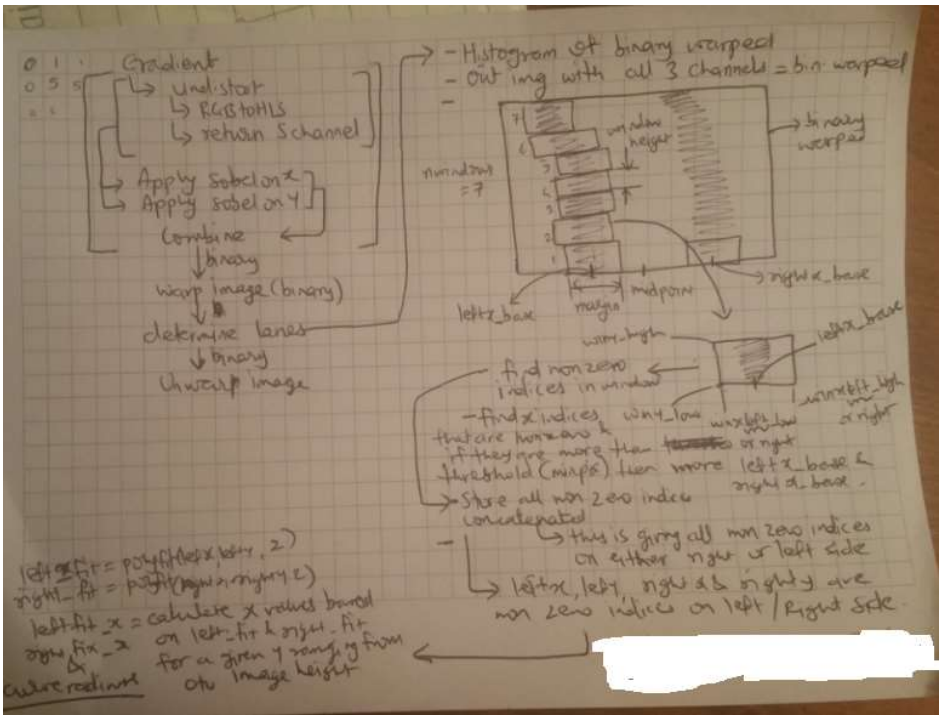


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

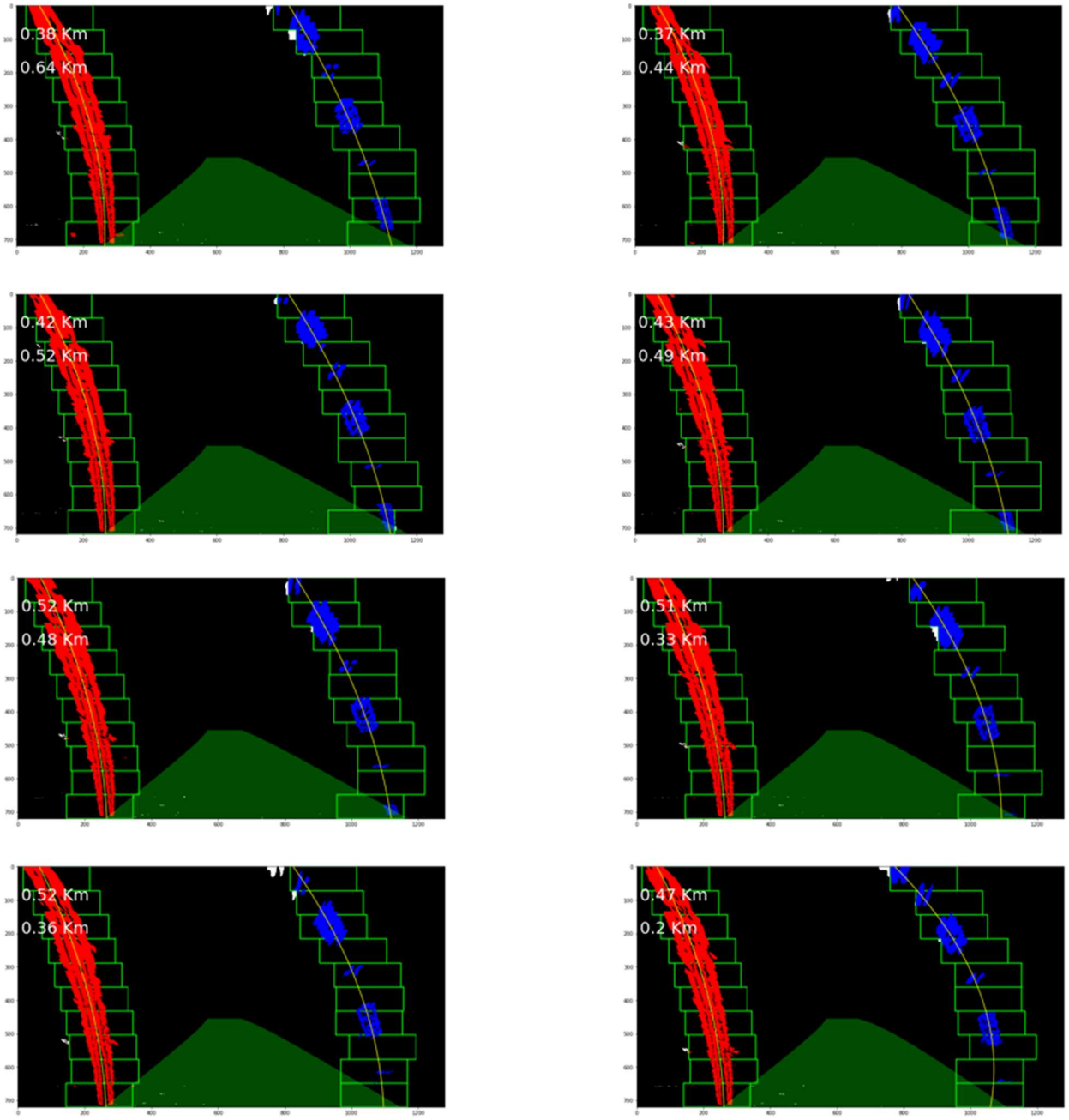
This was where most of my time was spent. Most of the functions used in this are copied from the course. The modifications I made are related to smoothing (averaging) between frames.

To test all the functions, I prepared another Jupyter notebook called "04. Lane detection and curvature estimation". The function determinelanes takes the input of warped binary image. In order to work out the tool chain explained in the courseware, I prepared a cheat sheet. This sheet was so that I understood how the various functions eventually reach to the point of working out the left_fitx and right_fitx – these are the coordinates for left and right lanes that are plotted against Y axis (yvals = np.linspace(0, yMax - 1, yMax)).

Here is my cheatsheet (if it doesn't make sense completely to the reader, its ok, I only want to show the background work I did to help me understand):



The output of my lane detection function is as follows:



These are not the test images given by the courseware. These are the 8 images I selected as being more challenging from the video frames.

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.

Following lines show the code that detects the curvature from the left_fit and right_fit polynomial coefficients obtained in the previous step:

```
# Fit a second order polynomial to each
left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx, 2)

# Generate x and y values for plotting
ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )
left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]

# Fit new polynomials to x,y in world space
left_fit_cr = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
right_fit_cr = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)

# Calculate the new radii of curvature
y_eval = np.max(ploty)
left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0]
```

Main points to note here are the ym_per_pix and xm_per_pix variables. These variables are used to convert the image pixel coordinates to meters. The steps to work out left_fit and left_fit_cr are the same except for the conversion from pixels to meters. The left_curverad and right_curverad are basically the following equation:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

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

In the Jupyter notebook "04. Lane detection and curvature estimation", the code below outputs the curvature for the left and right lanes onto images and displays them as shown:

```
1  cols=4
2  rows=2
3  fig, axes = plt.subplots(cols, rows, figsize=(40,40))
4  indexes = range(cols * rows)
5  for ax, index in zip(axes.flat, indexes):
6      test_image = test_images[index]
7      img=gradients(test_image ,10,160)
8      img_size = (img.shape[1], img.shape[0])
9      img = cv2.warpPerspective(img, M, img_size)
10     out_img, left_fitx, right_fitx, ploty, left_curverad, right_curverad, \
11     left_fit, right_fit = determinelanes(img)
12     result = unwarpimg(test_image, left_fit, right_fit)
13     #result = unwarpimg(out_img, left_fit, right_fit)  # Uncomment and comment line 12
14     ax.imshow(result)
15     #ax.plot(left_fitx, ploty, color='yellow')         # Uncomment and comment line 12
16     #ax.plot(right_fitx, ploty, color='yellow')        # Uncomment and comment line 12
17     ax.text(10,100,str(round(left_curverad/1000,2))+' Km',fontsize=32, color='white')
18     ax.text(10,200,str(round(right_curverad/1000,2))+' Km',fontsize=32, color='white')
19
```

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!).

Link to my final video:

https://youtu.be/5g1Yeavub38

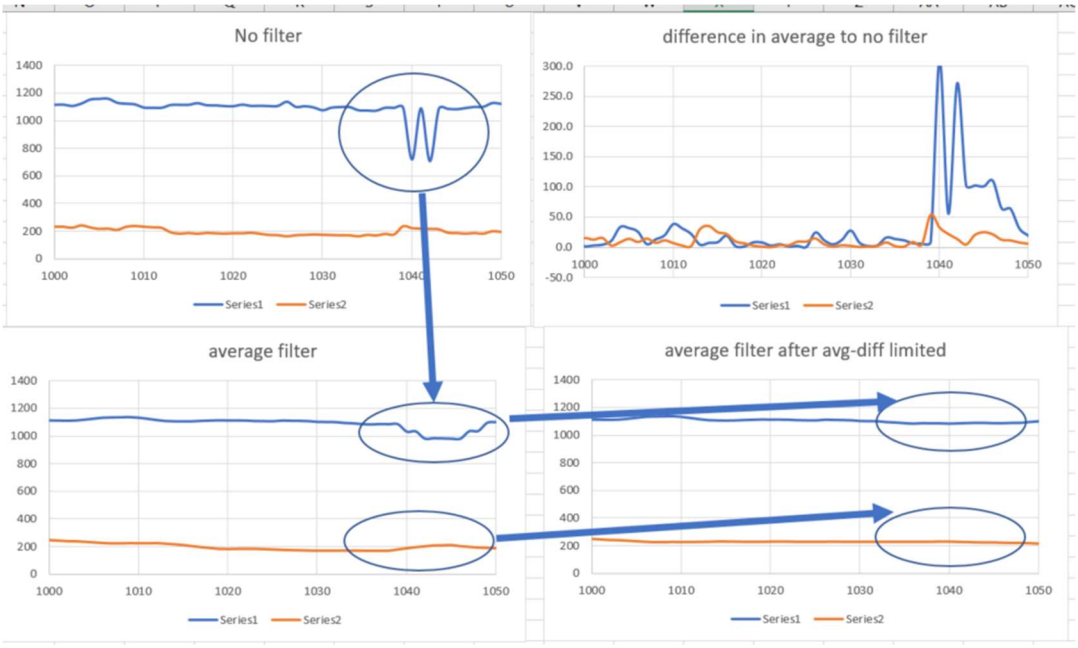
Discussion

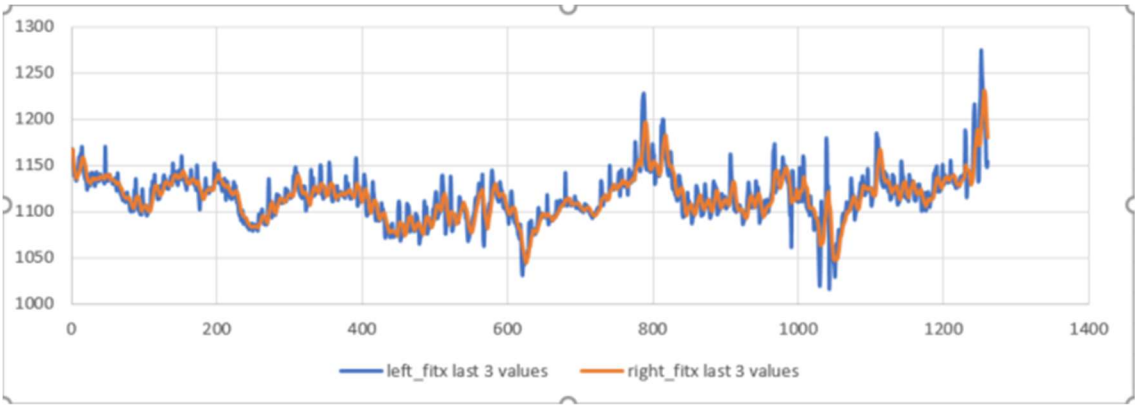
1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

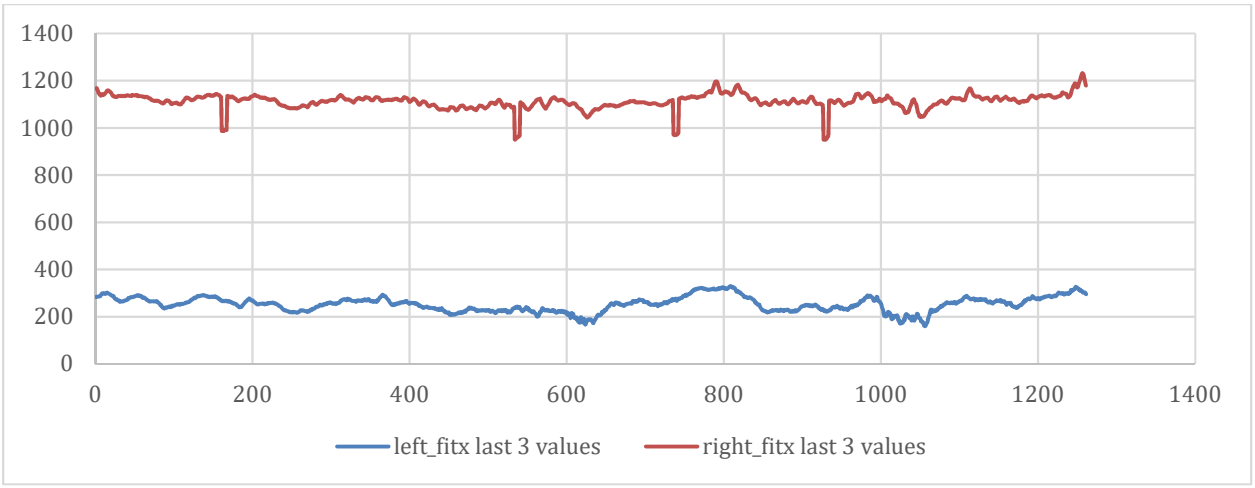
Not to make any excuses (but here is one), my 2 weeks in Sweden for work, really robbed me of any extra time to convert my test code to a more efficient format. I am submitting this about 4-5 days late from suggested date. I would have loved to use classes for storing current and past lane information such as the fits and curvature and use the class methods to smooth out the variations and peaks between the frames. But I have to push on to next stages in the interest of time. I will come back to this project anyway as soon I will be doing something similar for work.

But for what I did finish doing, following were my challenges:

Biggest challenge was to get the lane base starting point between the frames to smooth out. I eventually used averaging based on some excel work I did. It is shown here. I exported the base fitx values to excel to analyse where the function needed smoothening and by how much. I directly used this information to write the smoothing functions (code lines 95 to 118 in the Jupyter notebook "05. Video pipeline and implementation").







These images show before and after smoothening.

Current issues with the video pipeline:

1. The smoothening of the lane lines needs to be more robust. As indicated, Kalman filtering will be able to achieve that.
    a. I am planning on studying and using the techniques demonstrated in this link: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/README.md
2. More tuning to include the lighter sections of the track to enable better lane recognition
    a. I have only used S and L channels from the HLS image. More could be done to include HSV related isolation of channels.
    b. Yellow and white lines could be recognized separately and then combined to improve robustness.
3. Current pipeline will falter when the image being obtained is obscured by sun reflections, rain or weather related events.