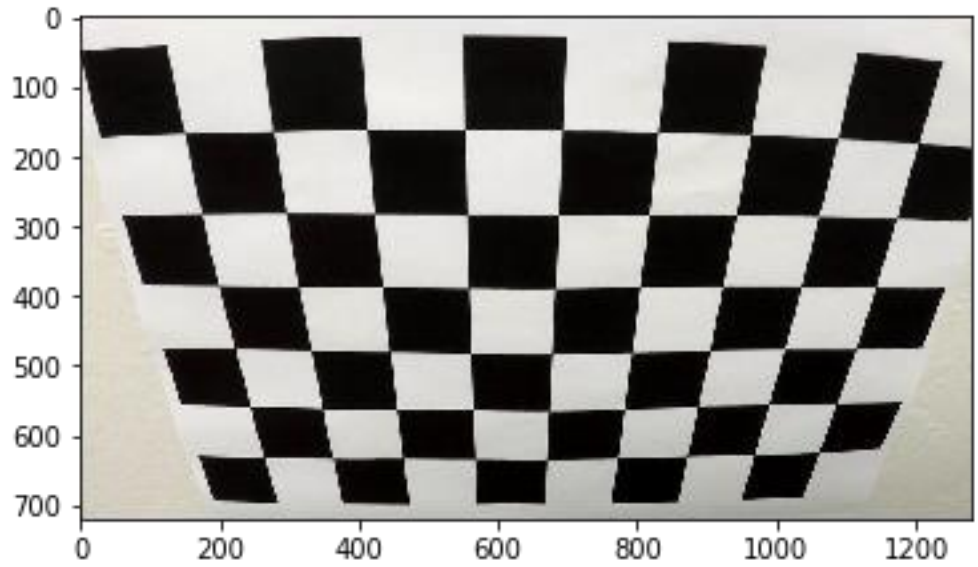
Write Up

In this write up I will describe all the steps that I've made to pass this project.

Firstly, I've started with distortion. As we know, distortion is huge problem, but it can be solved easily using some techniques.

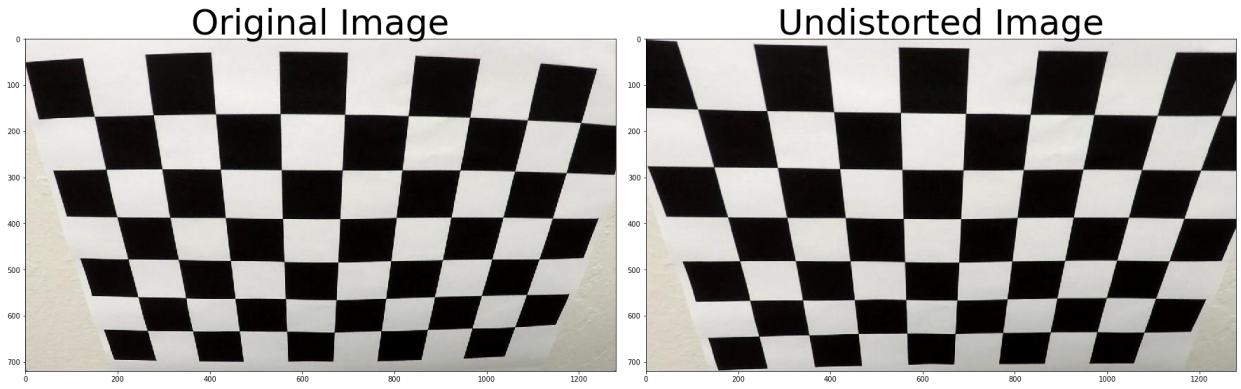
First, I load the image:



The corners of the images are distorted. Then I use the images in camera_cal folder to calculate matrix for undistorting the image.

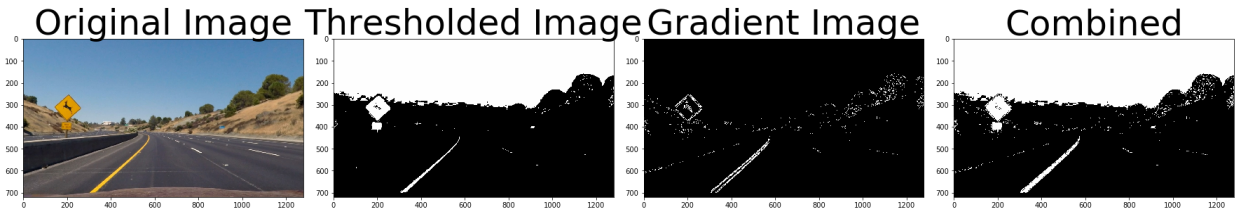
As we see, there is 9 edges in the row and 6 edges in the column. So, I set nx = 9, ny = 6.

After calculating the matrix to undistort the image we get the following:



Now, after undistorting image, I apply threshold techniques to find my lanes. Firstly, I define finding the

lanes using gradient. I use absolute (horizontal and vertical), magnitude and direction Sobel thresholds. For absolute I use min and max threshold = (20,100), for magnitude threshold = (30,100), and for direction threshold = (0, np.pi/2). Then I combine all of them. After experimenting these values give satisfactory output. I also define 2 color spaces, HLS and LAB color space. I take S channel from HLS and after trying B channel from LAB seems the best. I define threshold (160, 255) and (150, 255) for S and B respectively and I combine them together. After that I combined both gradient threshold and color space threshold together using function Combined(). In the end, I have the following result:



After getting good threshold, now it's time unwarp image to get the bird-eye view.

bottom_L = ([190, 720], [320, 720])

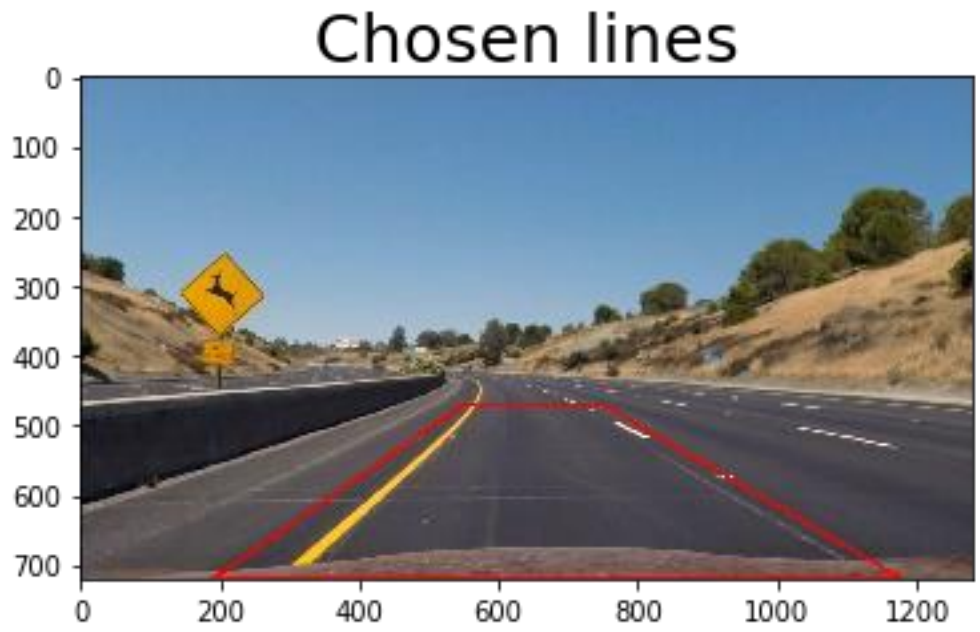
bottom_R = ([1180, 720],[920, 720])

top_L = ([550, 470], [320, 1])

top_R = ([750, 470], [920, 1])

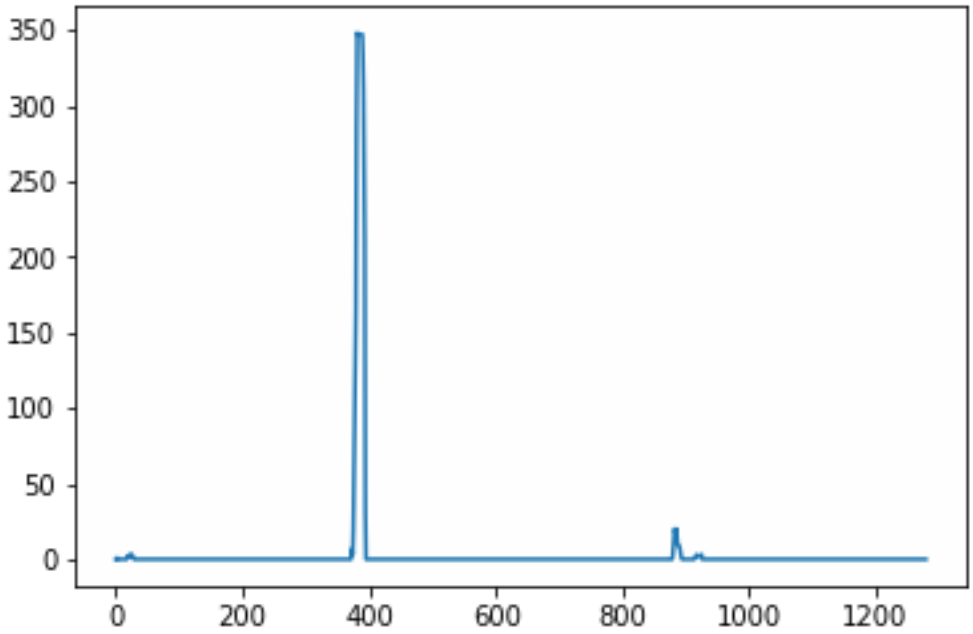
I chose this points, [0] members of array describe source points and [1] destination points.

Here are how source points look on my image

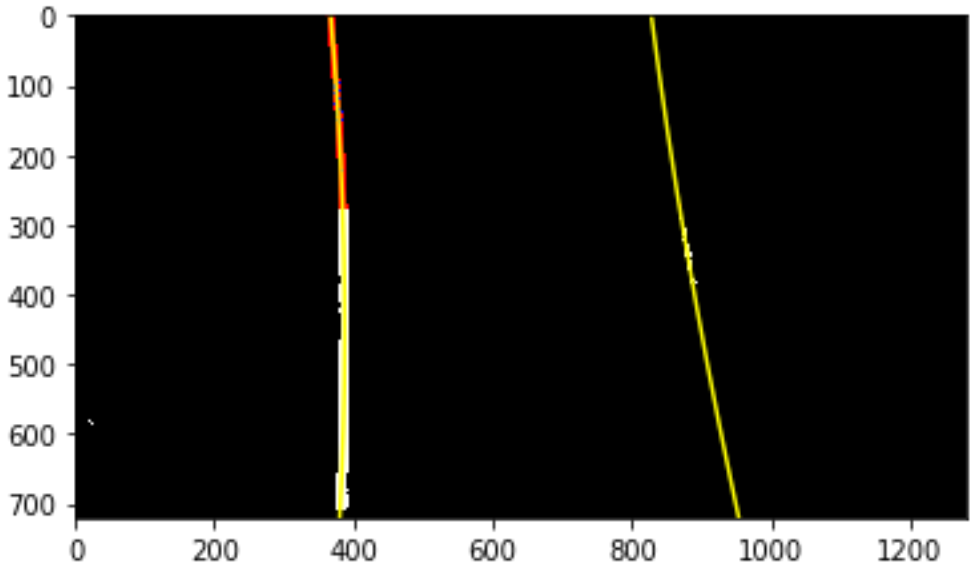


After applying this transformation and combined thresholding we get the following result

After that we can calculate histogram to detect peaks and find lines.

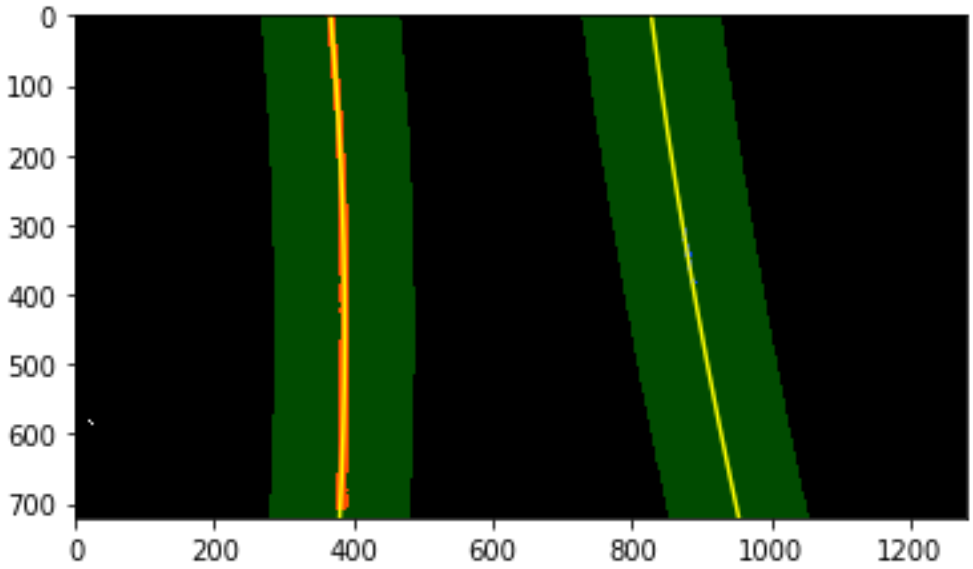


After that we can calculate fit of our lanes and detect them. Here is how they look like in our example



I also define next_step_lines() functions to find lanes after they were found once. This helps to use time efficiently.

Here are how the results of this function looks like



But not only we have to calculate lanes but find curvature and distance from the center.

I do all of this in detect_lanes() function and in next_step_lines() function, after calculating the left_fit and right_fit, I use

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


y_eval = np.max(lefty)
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)
left_curverad=((1+(2*left_fit_cr[0]*y_eval+left_fit_cr[1])**2)**1.5)\
                    /np.absolute(2*left_fit_cr[0])
right_curverad = ((1 + (2*right_fit_cr[0]*y_eval + right_fit_cr[1])**2)**1.5) \
                     /np.absolute(2*right_fit_cr[0])


radius = (left_curverad + right_curverad)//2
```

This code to calculate radius and

```
left_fit_x = left_fit[0]*height**2 + left_fit[1]*height + left_fit[2]
righ_fit_x = right_fit[0]*height**2 + right_fit[1]*height+ right_fit[2]
center_position = (left_fit_x + righ_fit_x) / 2
center_dist = (car_position - center_position) * xm_per_pix
```

This code to calculate center distance.

After that I am ready to fill my lanes using left and right fits. I also write my curvature and center distance.

Here is the result

In the end, I define class for having values for every situation. I define best fit and I use it to calculate my best fit in several scenarios. I also check if best fit and normal fit are different by too much or not. I use detected parameter to know if I have previously detected lanes or not, that allows me to check for margins and not running my detection all over again. Also, I have current_fit parameter, I use that not only to have current fit, but I also to get rid of the fit if I have more than 5 into array, that helps me to have more efficient algorithm. I define two classes for left_lanes and for right_lanes to have more robust result. Then everything happens in process_img function where I combine every step described above together and apply it on video. You can see my results in uploaded file.

Discussion

One of the difficulties of this project was to have a good unwarping function. It was hard to find optimal points for unwarping my image. I had to experiment a lot to get good unwrapped image and I am sure there can be better ways and better results, which will give better output. The reason why I am saying this is that my lane is too big for challenge and harder challenge video. On image it successfully identifies the positions of lanes, but it has hard time at fitting the green area between the detected lanes. Using smaller area for unwarping the image might be good idea.
Also, at first, I was trying to find lines only by using gradients and it was very unsuccessful (well, it could detect in some scenarios, but it failed in different situations, meaning it was not that robust). I was running through parameters a lot, but it was not improving, that cost me a lot of hours. HLS color space does a good job at detecting lanes but for more robustness I added LAB color space. I found this color space on https://en.wikipedia.org/wiki/CIELAB_color_space.

Overall, This project was interesting, I had several difficulties but I managed to complete it. Hopefully described everything well.