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.

My submission files include:

pipline.ipynb : image process pipline, main file

_output.mp4 : output video from project video by applying pipline

fit_polynomial.py: a python script to fit polynomial

sliding_window_search.py : a python script to do sliding window search operation

output_images: folder, saved images were listed here

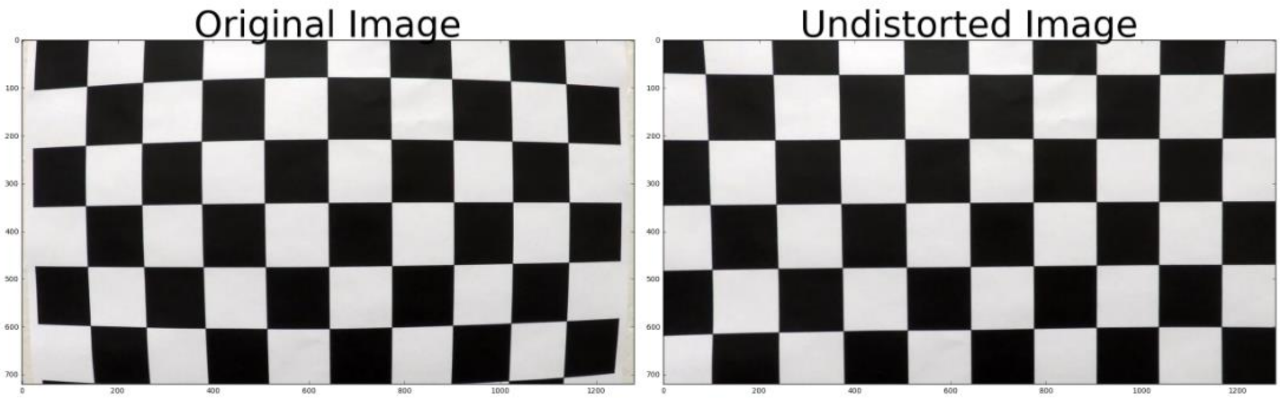
Camera Calibration

The code for this step is contained in the first code cell of the IPython notebook located in pipline.ipynb.

I use camera_cal/calibration*.jpg to get the mtx and dist

I use objpoints to store 3D points and imgpoints to store 2D points.

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.undist()` function and obtained this result:

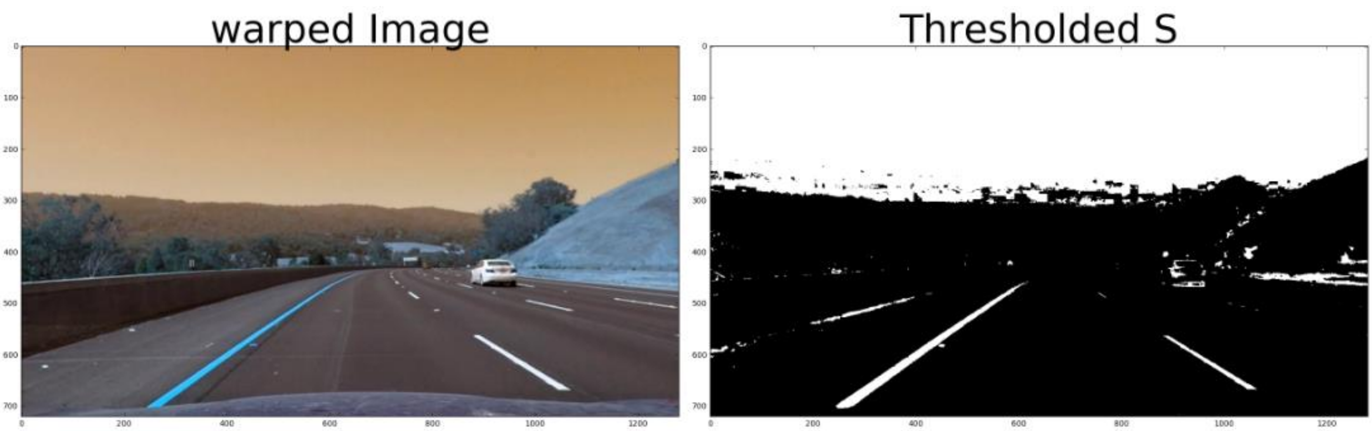


And below is an undistorted image example:

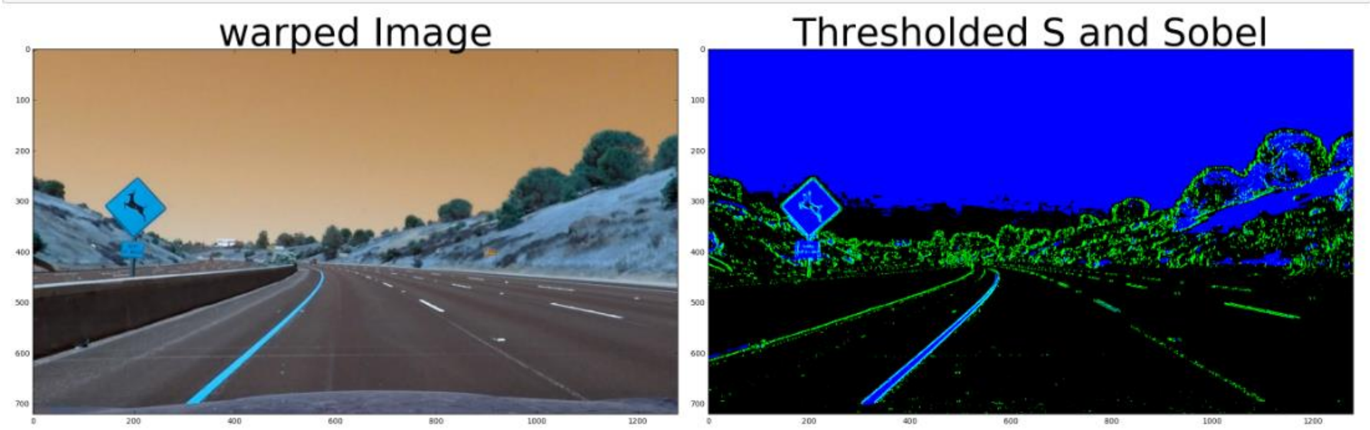
COLOR AND GRADIENT THRESHOLD

I build three thresholds in my pipline [cell 4]. But I only used color threshold and abs_sobel_thresh in this project

And then I applied color threshold to the image. I got this result.



To get a better quality, I applied a combined threshold to warped image:



Perspective Transform

At first, I defined a function to do the perspective transform.

```
# def image_unwarp(undisto,img_size):
#     left_top_src = [550, 580]
#     left_down_src =[800, 580]
#     right_top_src = [400, 680]
#     right_down_src = [1050, 680]
#     src = np.array([ left_top_src,left_down_src,right_down_src,right_top_src], dtype=np.float32)
#     left_top_dst = [300, 310]
#     left_down_dst = [700, 370]
#     right_top_dst = [300, 870]
#     right_down_dst = [700, 800]
#     dst = np.array([left_top_dst,left_down_dst,right_down_dst,right_top_dst], dtype=np.float32)
```

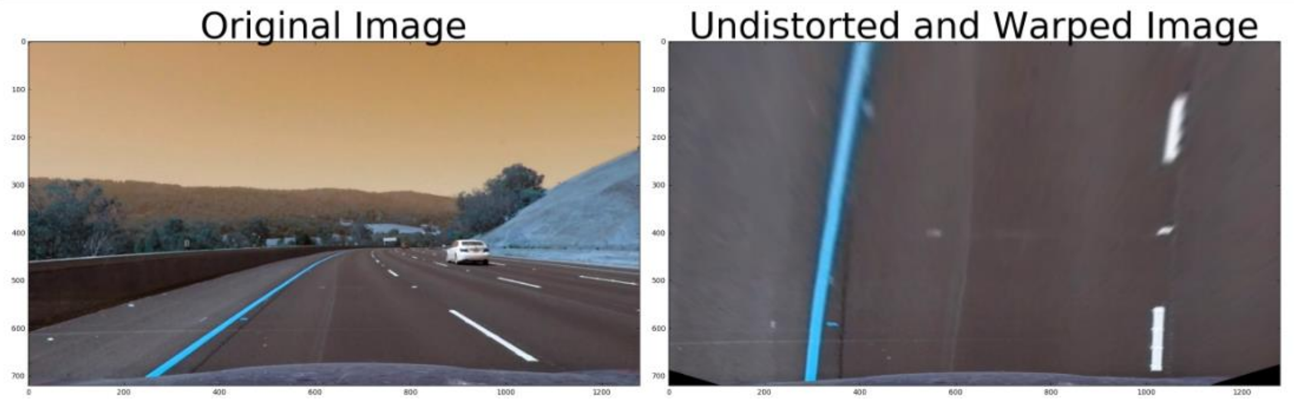
But I can only get a top_down result if I applied this method to my image. After searching the forum and googled for a while. I decide to use this function:

```
def image_unwarp(img, bird_view = True):
    xsize = img.shape[1]
    ysize = img.shape[0]

    p1_src = [175, ysize]
    p2_src = [560, 470]
    p3_src = [730, 470]
    p4_src = [1155, ysize]
```

```
src = np.array([p1_src, p2_src, p3_src, p4_src], dtype=np.float32)

p1_dst = [250, ysize]
p2_dst = [250, 0]
p3_dst = [1030, 0]
p4_dst = [1030, ysize]
dst = np.array([p1_dst, p2_dst, p3_dst, p4_dst], dtype=np.float32)
```

The difference between them is their coordinates. The src and dst coordinates here are essential.

By finishing this function, I can finish my perspective transform. And then I got this result:

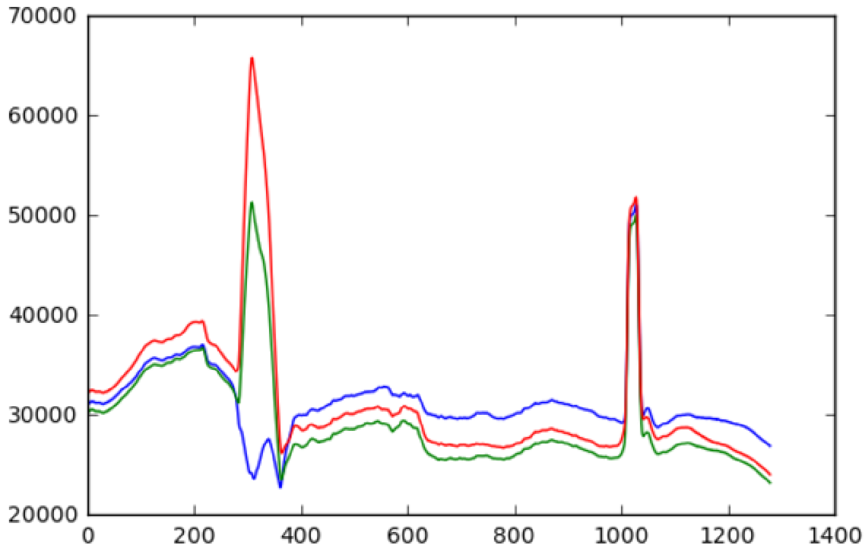


The undistorted and Warped Image has a good quality. It can be obviously found that lane lines are parallel.

Histogram

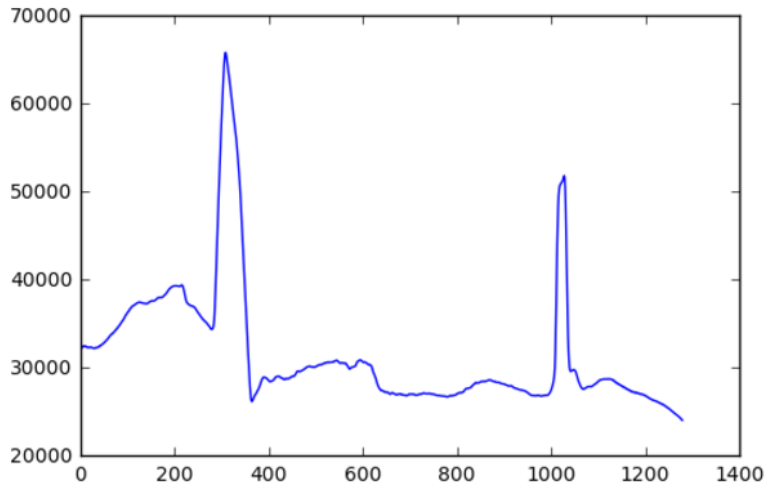
To have a better understanding of the undistorted and Warped Image, I did a histogram operation to the image.

The result I got have three curve, which means three channels.:



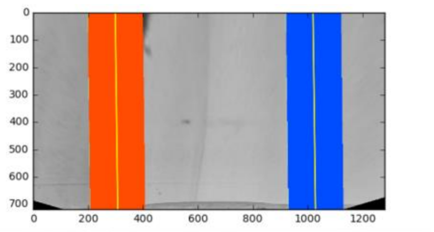
And then I extract the S channel [cell 11]

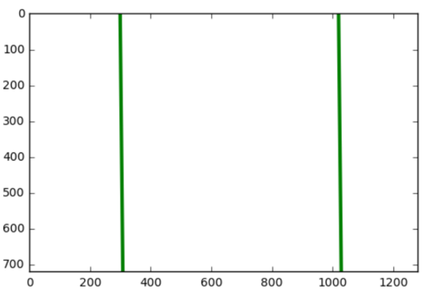
histogram = np.sum(s_img[s_img.shape[0]/2:,:], axis=0)



This shows that S channel is a good channel for the threshold because its peak value is obvious.

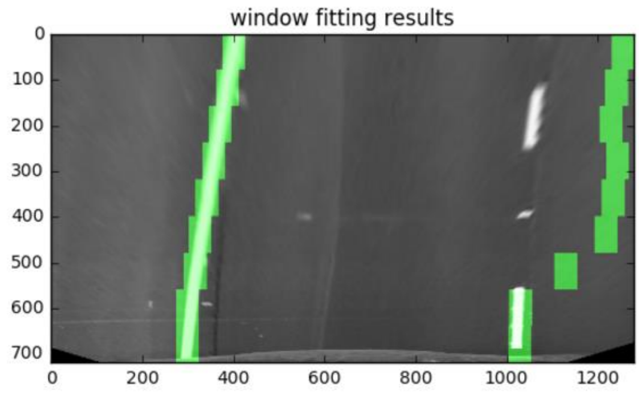
In the cell 12, I fit the polynomial. I built a python file called fit_polynomial.py, which was include in this forder. fit_polynomial.py has a function to get value of those two lane lines and fit them.





I build a python script called silding_window_search.py to apply sliding window search to the warped image. In the cell 14, I use silding_window_search to detect two lane lines and use sliding windows to detect them.

The result I got is good:

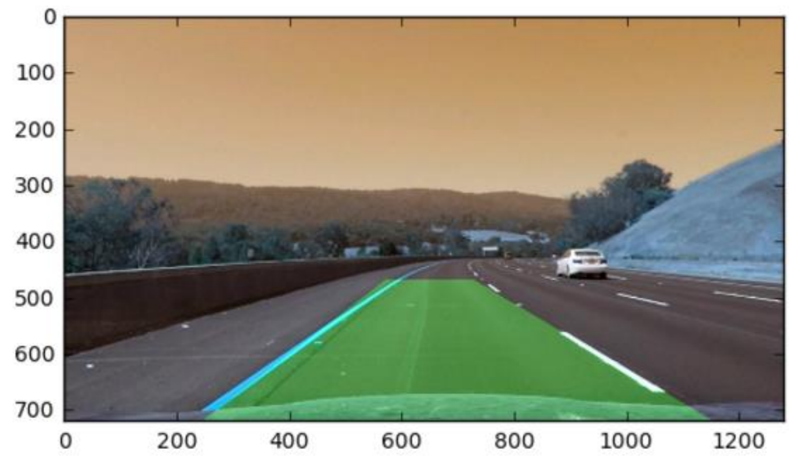


Draw lane lines back

The most essential part is draw lane lines back to the road.

In cell 15, this part is similar to the lecture's useful code. I only changed them slightly(the warped image part).

```
get lane line
```



By applying my method mentioned above to the origin image, I get the result above. It is pretty. And I processed all test images and saved them in output_images folder.

The video:

The video I processed is project video. It was nearly precisely draw the lane line out. It can be found in _output.mp4. More code details can be found in my pipline.ipynb. I explained my code thoroughly.