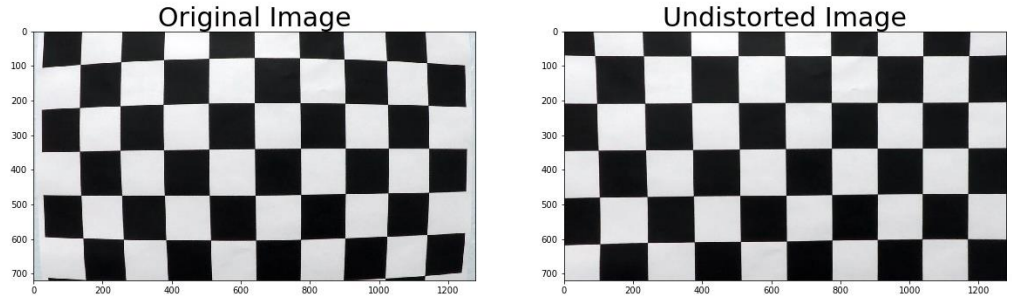
Illustrate

I submit two code file—'CarNdP4.py' and 'illustrate.ipynb'.
'illustrate.ipynb' are a jupyter notebook illustrating all my code.
'CarNdP4.py' contains a class that use those code to deal project video.

Camera calibration

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) 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 (x, y) 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 that didn't find all chessboard corners using the `cv2.undistort()` function and obtained this result:



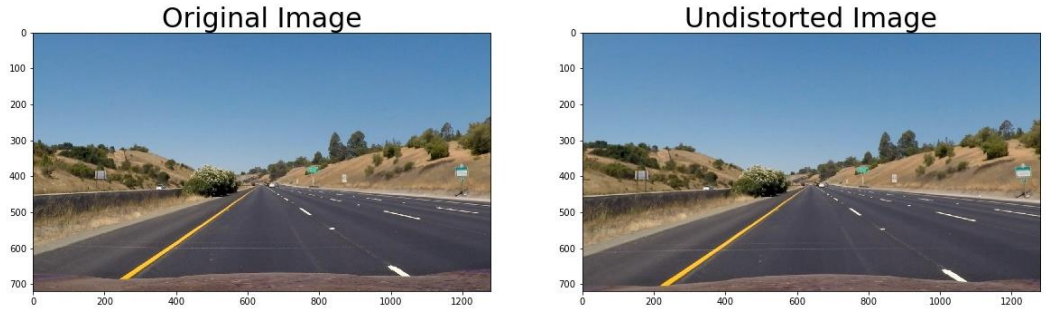
And I found it perform ok, so I save the coefficients as pickle file. So when I want undistort an image I just need to read the coefficients to memory.

Pipeline (single image)

In this section, I will illustrate how I find lane lines step by step. This is an image in 'test_images'.

distortion-corrected

I saved the coefficients as pickle file. So I just need read them in memory, and use 'cv2. Undistort()' function to get correct image.



Gradients to get lines

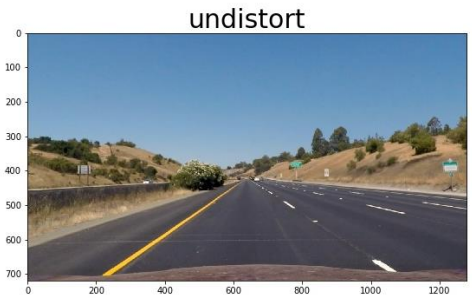
Sobel threshold

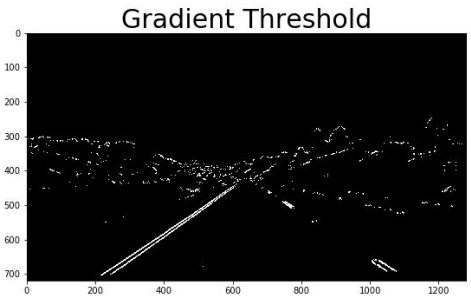
First, use cv2.Sobel() function on both horizontal and vertical direction and scale them to 0-255. If one pixel are both bigger than 50 in to sobel image, we think it might be lane lines.

Magnitude threshold

Second, We calculate the square root and direction of soble_x and soble_y obtained in the last step.
If one pixel's the square root are in [70,255] and its' direction are in [0.7,1.3], we think it might be lane lines' pixels.

Finally, if one pixel pass through sobel_threshold or magnitude threshold, we think it's a lane lines' pixel.





Color space threshold

I turn the image to hls color space using cv2.cvtColor()

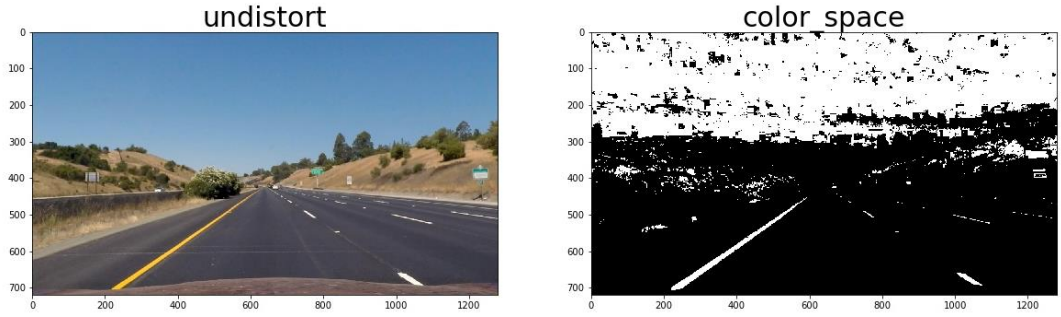
Saturation channel get lane lines

See the picture above, the lane lines pixels in s channel are obvious. So I use s channel to get lane lines pixels. Of course, if car runs on a cloudy day or on an old road, lane lines may be not that obvious. I do Histogram equalization for saturation channel. If one pixel's value is bigger than 165, it might be lane lines pixel.

Lightness channel Remove the shadow

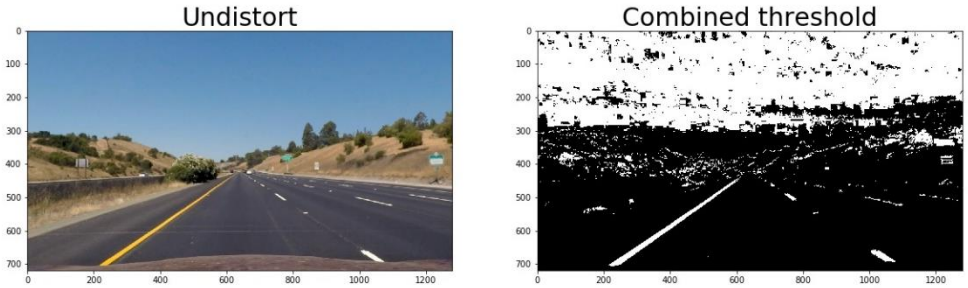
The lane lines from s channel has got an important problem. The shade of the tree is considered as lane line. But the shade is clear in lightness channel and easy to get. So if one pixel in l channel is smaller than 30, we think it's just the shade of the tree instead of lane.

Hear is the color space result of color space



Combined gradient and color space

Here is the result that combined gradient function and color space function



Perspective Transformation

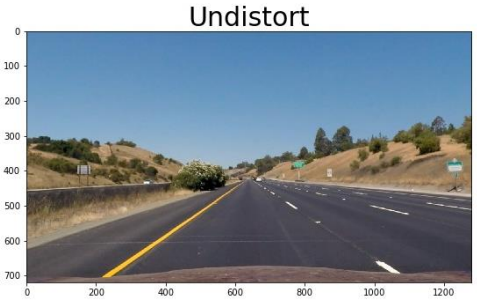
Through several experiments, I determine perspective transform:

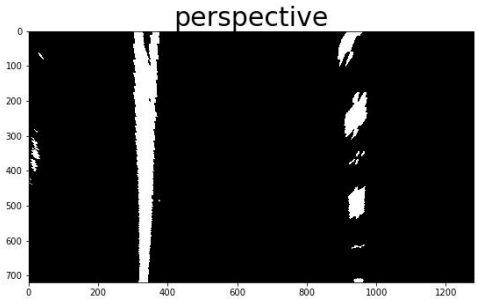
```
perspective_src = np.float32(
        [[603,445],
         [679,445],
         [1058, 688],
         [254, 688]])
perspective_des = np.float32(
        [[330, 0],
         [950, 0],
         [950, 720],
         [330, 720]]
)
```

Then, I get perspective parameters by function cv2.getPerspectiveTransform().
I use cv2.warpPerspective() function when I need to get a overlook images.

```
names = glob.glob('..\\test_images\\*.jpg')
```

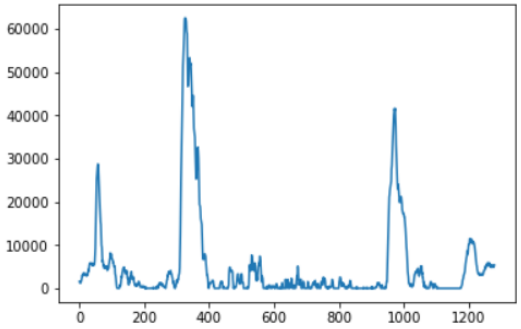
```
img = mpimg.imread(names[0])
img_size = (img.shape[1], img.shape[0])
M = cv2.getPerspectiveTransform(perspective_src, perspective_des)
res_M = cv2.getPerspectiveTransform(perspective_des,perspective_src)
```



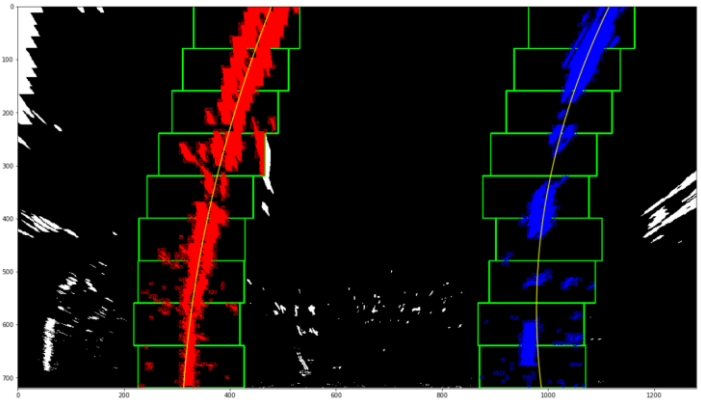


Identify lane lines

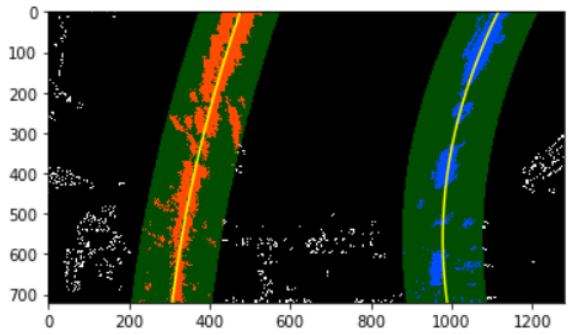
Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:
1. I use np.sum() to get histogram to get the liens start position.



2. Base on these two start position, I split the image into 9 window in vertical direction. I count the number of pixels in that windows. If it's bigger enough, I think I get the lane lines. This way, I get all lane lines pixels.
Then, I use np.ployfit() function to fit those two lines with a 2nd order polynomial.



3. Once I've got the lines, I will use 2nd order polynomial to find lanes in a easy way：



Calculate curvature and bias\ Reliability detection

In CarNdp4.py line 512 and 548, I write 2 functions to Calculate curvature and bias and detect the lines reliability.
Reliability detection contains 3 points: lines distance/car bias/radius

Draw lines on source image

I wrote this function at line 474 in CarNdP4.py.
Once I've got a warped image with lines, I use cv2.warpPerspective() function to transform this image back to source shape, and just add it to source image.