Udactiy Self driving Car Nanodegree Program
Project 4 : Advanced Lane Lines

Advanced Lane Finding Project

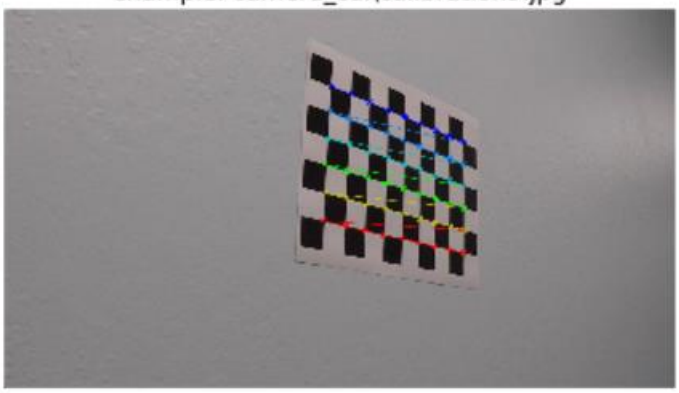
The goals of this project are the following:

- 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.

1- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.¶ <code cell 1 & 3>

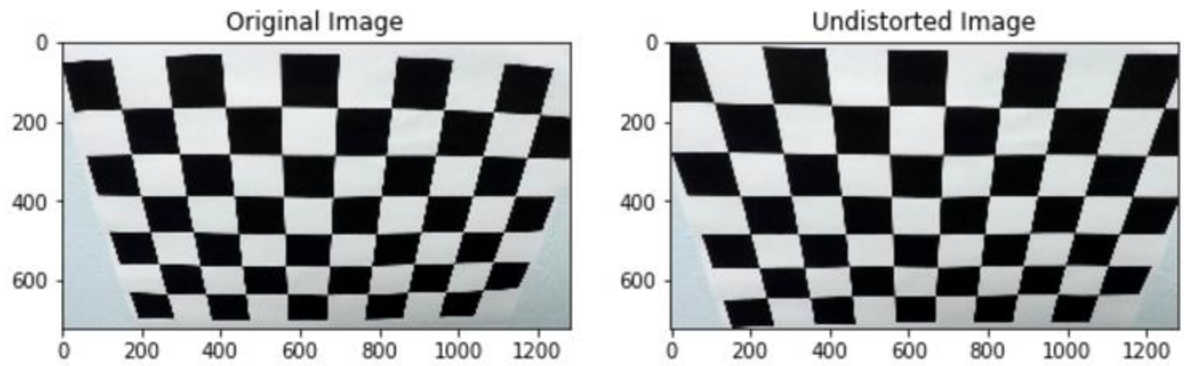
camera calibration. because the lenses of the camera distort the image of the real world. The calibration corrects this distortion.by find corners

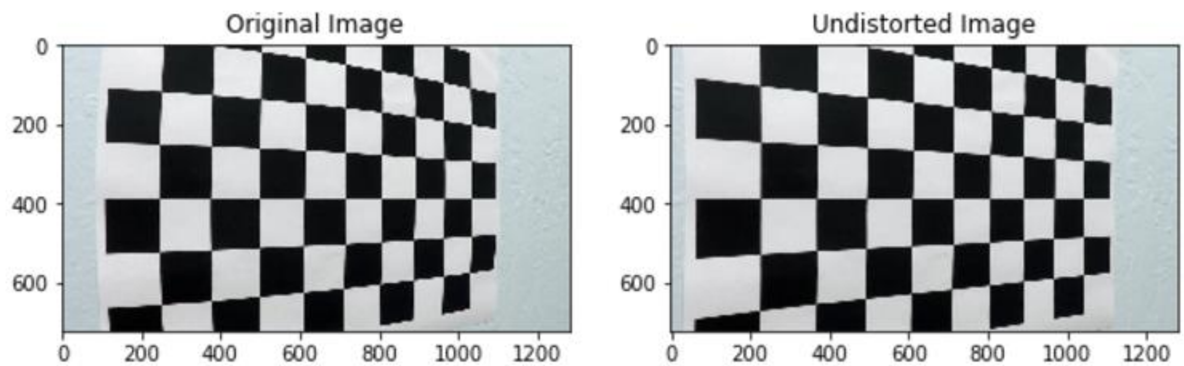
example: camera_cal\calibration9.jpg

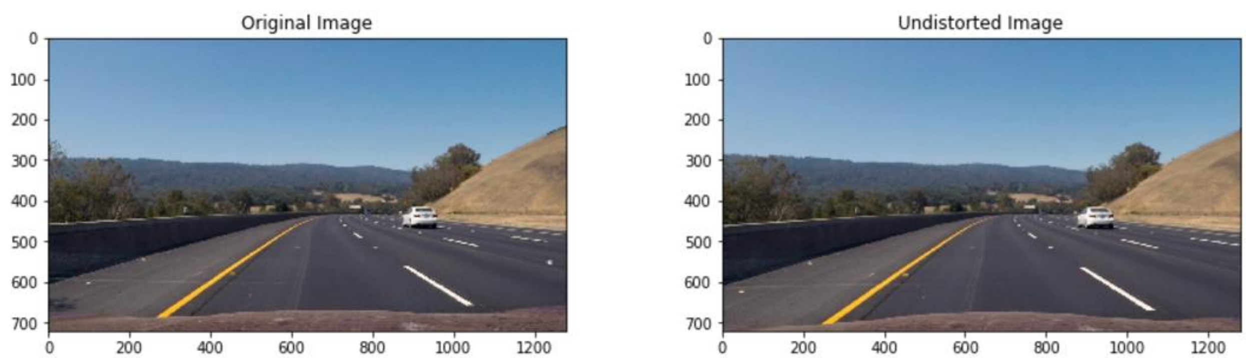


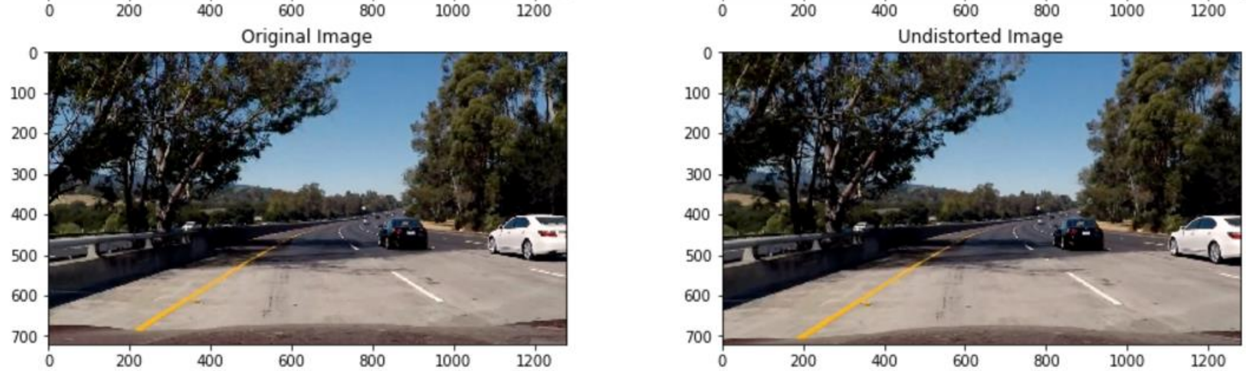
2- undistort images¶ <code cell 4>

Using `cv2.undistort` from opencv module to undistort image









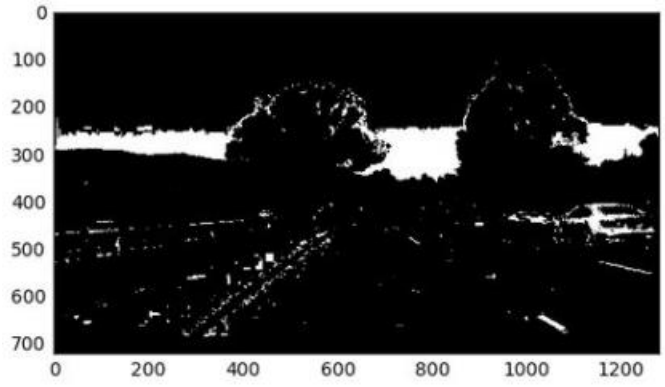
3- Color and gradient threshold¶<code cell 6 & 7>

a color threshold filter to identify lanes only (yellow and white) , using opencv convert color to HSV space (Hue, Saturation and Value). The HSV dimension is better to do this, because it isolates color (hue), amount of color (saturation) and brightness (value).

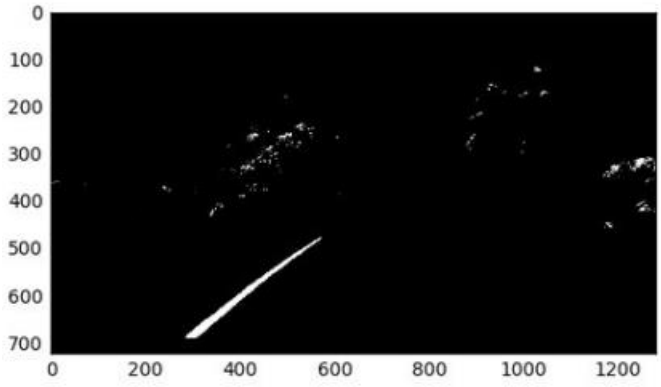
used HSV color space to identify yellow and white colors. We applied yellow mask as

```
yellow_low = np.array([0,100,100])
yellow_high = np.array([50,255,255])
```

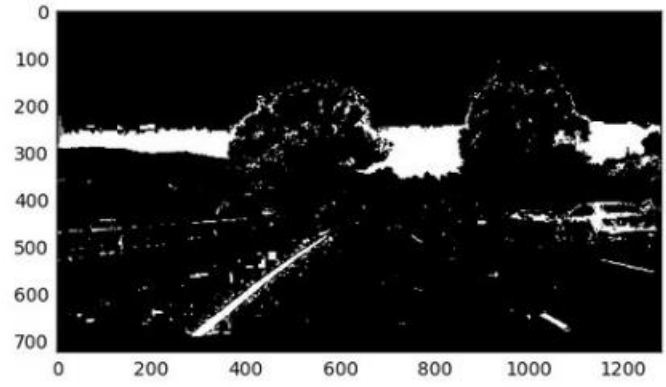
where apply_color_mask returns pixels with intensities specified between low and high values



White color filter

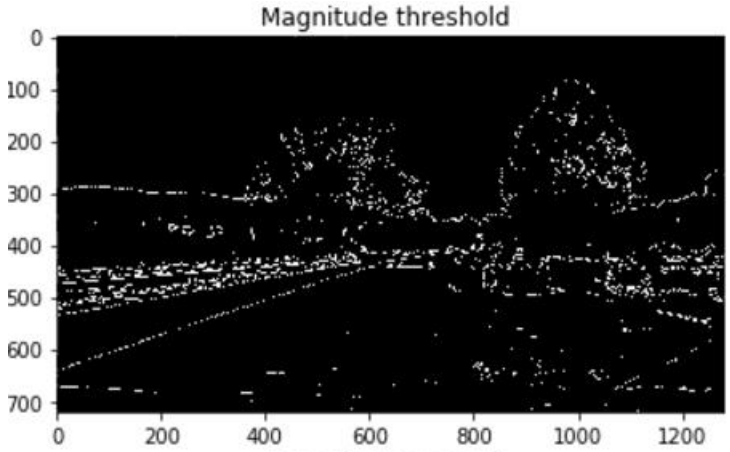


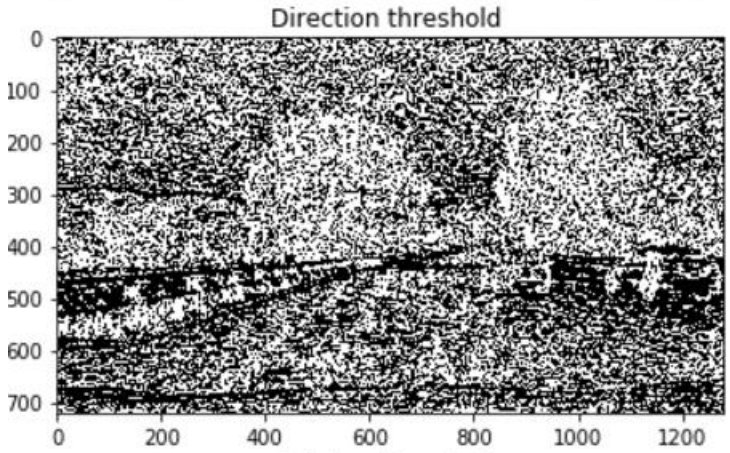
Yellow color filter

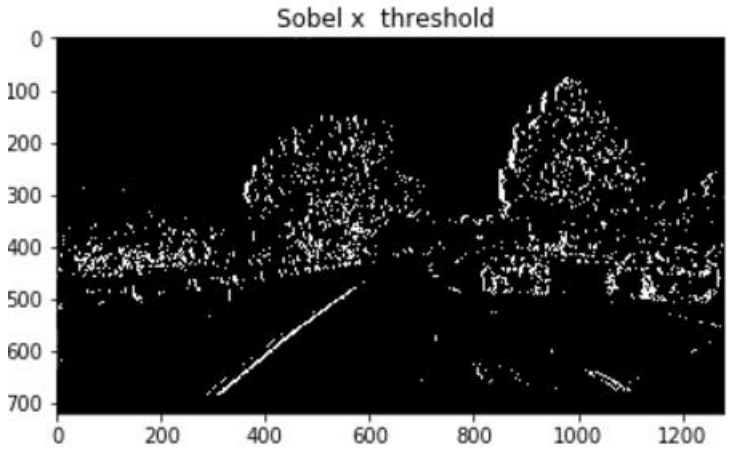


Yellow and white filters

To find the contrast, we use the Sobel operator. difference in color between two points is very high, the derivative will be high.
But we can compare any two neighbor points to do this derivative.







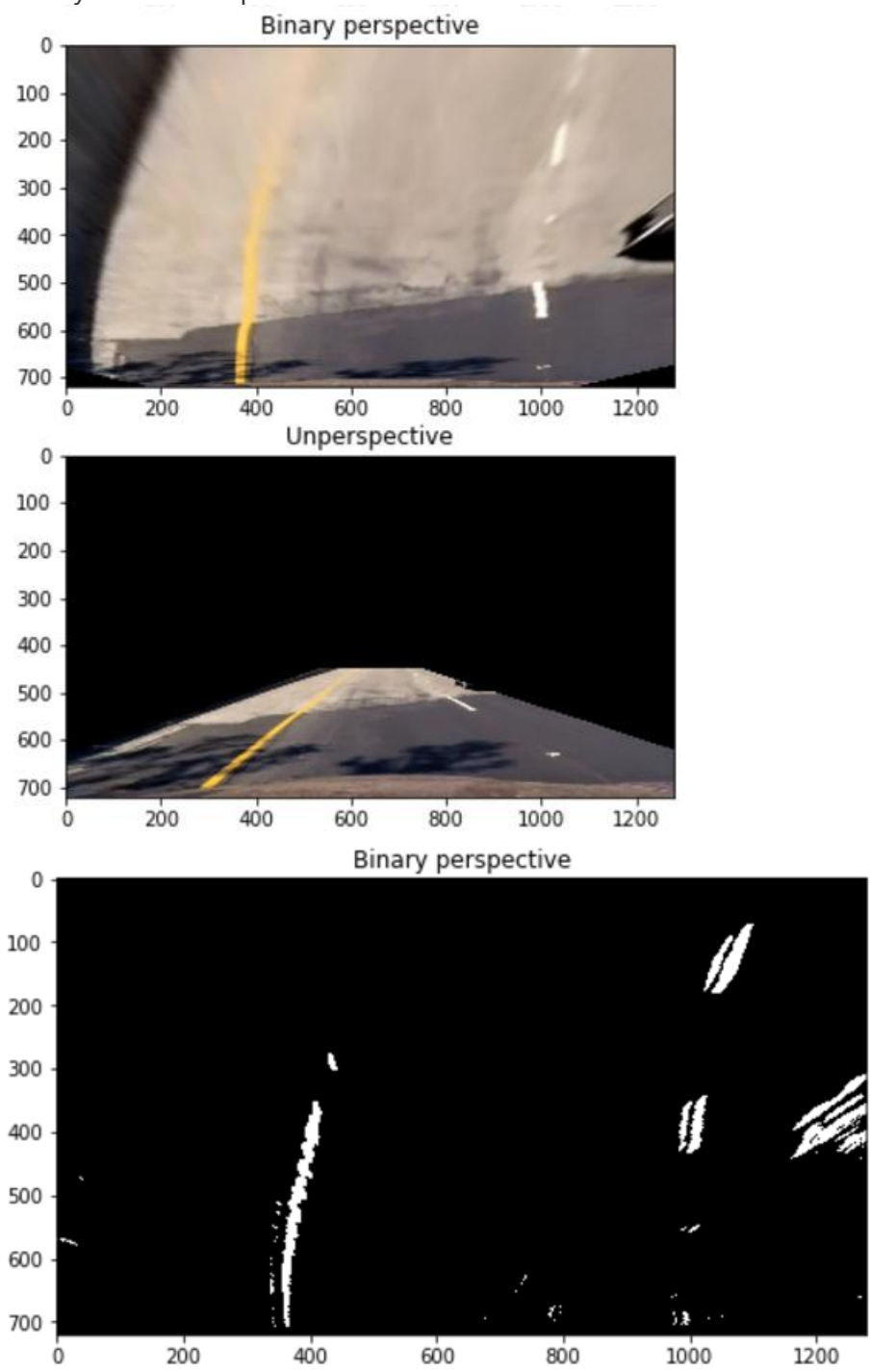
4- Birds eye view (Transform perception)¶<code cell 10 & 14 & 15>

Warp the image, as if it is seem from above. That is because makes more sense to fit a curve on the lane from this point of view, then unwarp to return to the original view.
The opencv function warp needs 4 origins and destinations point:

```
src = np.float32([[585, 450], [203, 720], [1127, 720], [695, 450]])
dst = np.float32([[320, 0], [320, 720], [960,720], [960, 0]])
```

note: I tried many color space before this step to make sure that my choice is the better

Bird eye view example:



5- Lane detection and fit <code cell 16,17,18,19,20>

we use a histogram on the bottom half of image. that the lane has most probability to be where there are more vertical points.

then feed the numpy polyfit function to find the best second order polynomial to represent the lane

6- Curvature of lanes and vehicle position <code cell 21,22,23>

The radius of curvature is given by following formula.
Radius of curvature= (1 + (dy/dx)**2)**1.5 / abs(d2y /dx2)
We will calculate the radius for both lines, left and right, and the chosen point is the base of vehicle, the bottom of image.
x = ay2 + by + c
Taking derivatives, the formula is: radius = (1 + (2a y_eval+b)**2)**1.5 / abs(2a)

I convert it to real space using :

```
left_curverad = ((1 + (2*left_fit[0]*y_eval*ym_per_pix + left_fit[1])**2)**1.5) / np.absolute(2*left_fit[0])
right_curverad = ((1 + (2*right_fit[0]*y_eval*ym_per_pix + right_fit[1])**2)**1.5) / np.absolute(2*right_fit[0])
center = (((left_fit[0]*720**2+left_fit[1]*720+left_fit[2]) +(right_fit[0]*720**2+right_fit[1]*720+right_fit[2]) ) /2 - 640)*xm_per_pix
```

7- composition pipeline <code cell 24>

Add threshold and bird eye views to the composition

8- Create video file pipeline <code cell 28>

Used VideoFileClip and clip1.fl_image functions with my process_image function to produce and clip.write_videofile to save video

Future Improvement:
Need to know more mathmetics to make things more smoother with challenge videos and also have problem in case no lanes I think I need method can use just one lane to predict other one.
And also may CNN with good size of labeled data can lead us to more accurate result on challenge video

Files:

Main Problem/project video: out_test_video2.mp4
Challenge video: challenge_test_video.mp4
Writeup.pdf
P4+Advanced+lane+lines+.html: html version from notebook
P4 Advanced lane lines .ipynb: notebook