Advanced Lane Finding Project

The goals / steps of this project are the following:

1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.

2. Apply a distortion correction to raw images.

3. Use color transforms, gradients, etc., to create a thresholded binary image.

4. Apply a perspective transform to rectify binary image ("birds-eye view").

5. Detect lane pixels and fit to find the lane boundary.

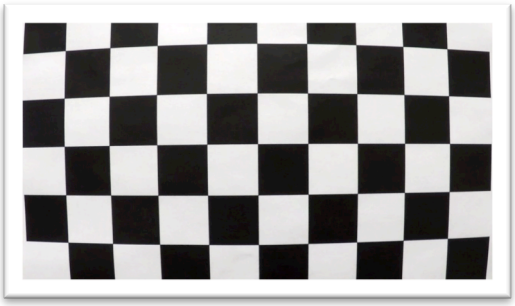
6. Determine the curvature of the lane and vehicle position with respect to center.

7. Warp the detected lane boundaries back onto the original image.

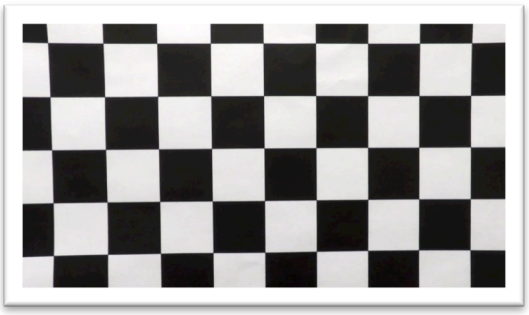
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.

In [1]-In[2] - I prepared "object points", which will be the 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. The 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 using the ***cv2.undistort()*** function and obtained this result:



ORIGINAL IMAGE

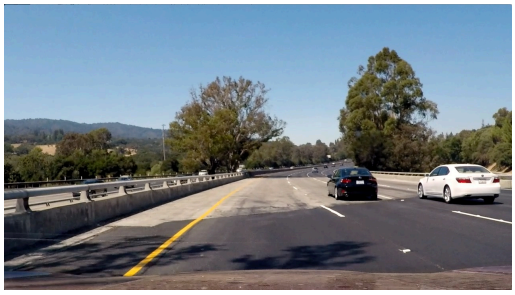
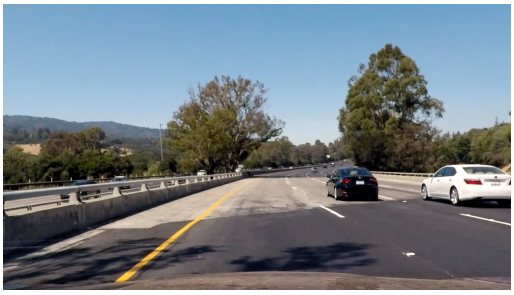


UNDISTORTED IMAGE

2. Apply a distortion correction to raw images.

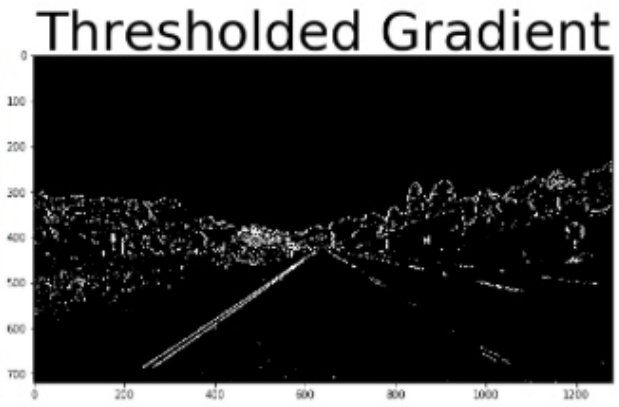
I have Tested undistortion on raw image from **test_images** folder.
test_raw_img = mpimg.imread('test_images/test4.jpg')
img_size = (img.shape[1], img.shape[0])

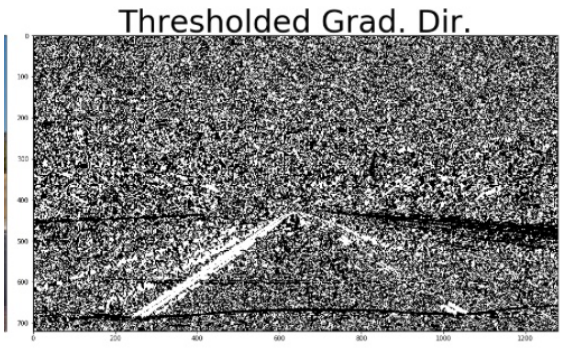
And I used a ***cal_undistort(test_raw_img, objpoints, imgpoints):*** to get following results:

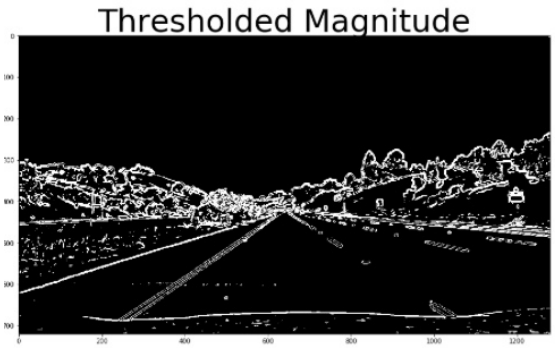



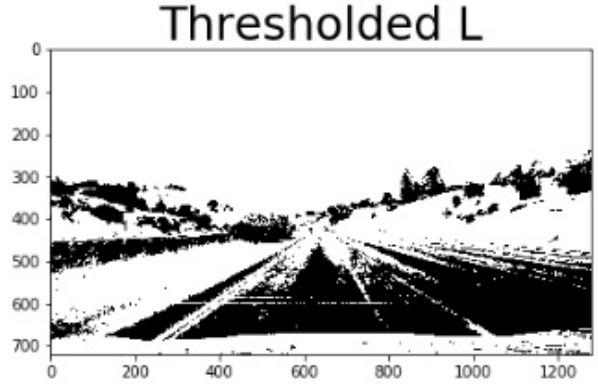
3. Use color transforms, gradients, etc., to create a thresholded binary image.

I used lection materials to make experiments with original images:





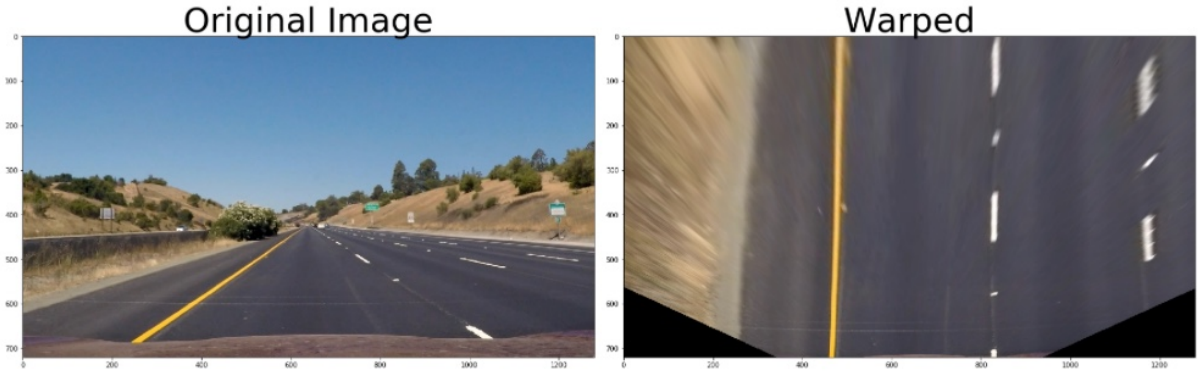




4. Apply a perspective transform to rectify binary image ("birds-eye view").

The ***unwarper()*** function takes as inputs an image (img), as well as source
(src) and destination (dst) points. I chose the hardcode the source and destination points

# Source points	#Destination points
(570,460),	(450,0)
(700,460),	(830,0)
(230,680),	(450,720)
(1050,680)	(830,720)



5. Detect lane pixels and fit to find the lane boundary.

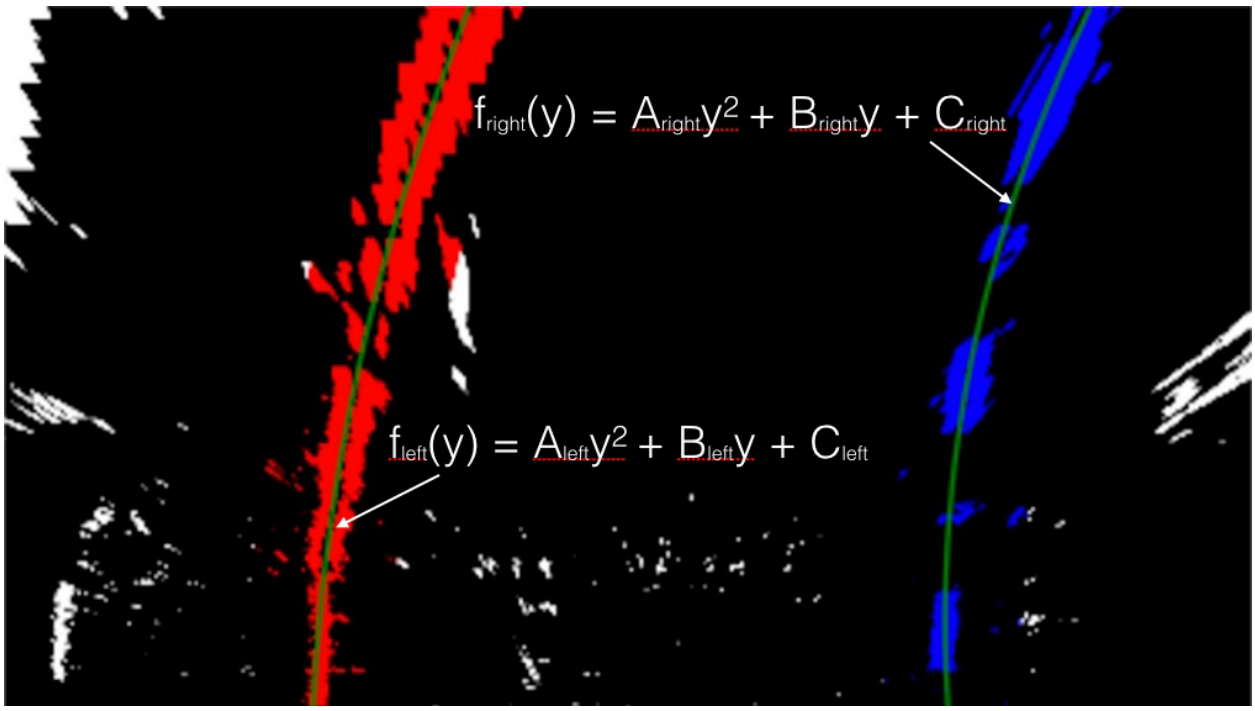
In order to decrease computed area I've implemented function
Which can read in an image and grayscale it. And than I cut the interested area:

```
img = mpimg.imread('test_images/straight_lines1.jpg')
def region_of_interest(img):
    ysize = img.shape[0]
    xsize = img.shape[1]

    vertices = np.array([[(200,ysize-40),
                          (630,420),
                          (650,420),
                          (xsize-170,ysize-40) ]] ,dtype = np.int32)
```

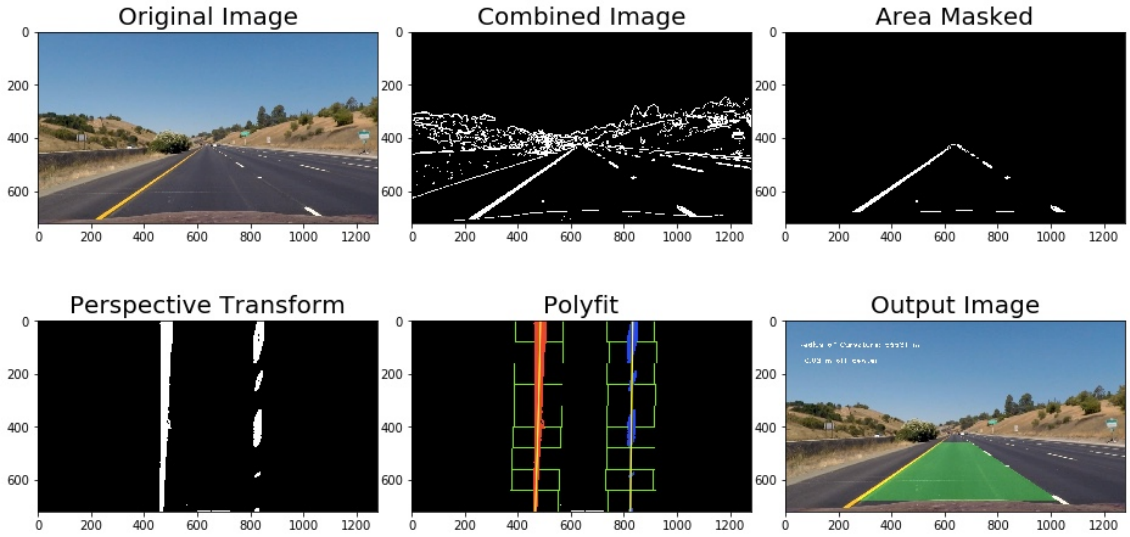
6. Determine the curvature of the lane and vehicle position with respect to center.

I used lection materials to find the curvature of the lane. To illustrate that function I'd put here the image from the lection:



7. Warp the detected lane boundaries back onto the original image.

My pipeline looks like that:



8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Here is the video (mp4).

DISCUSION

The pipeline to hold “green” area inside of lines should be improved. The shadows, different colors of asphalt and lane lines on intersections should not affect to the green lane stability. Perhaps, better result can be reached by using prediction algorythm.