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.

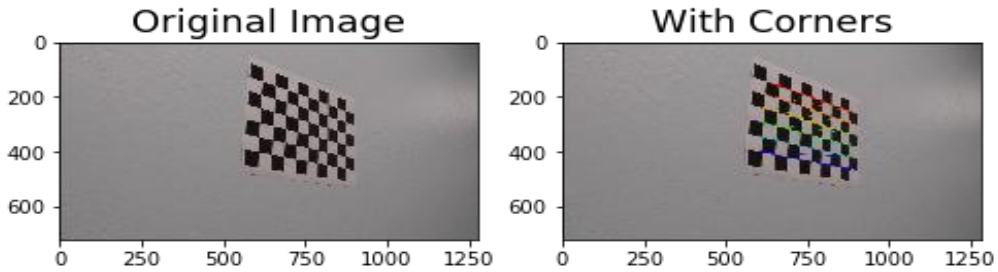
Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration

1. **Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.**

The first step of this project is to do the camera calibration. I correct this distortion with chessboard. By calibrating 20 images in test_image file, we use opencv function cv2.calibrateCamera() to find the arrays that describe the distortion of the lenses.

Then, we can apply cv2.undistort() function.



1. Provide an example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

3 Birds eye view

The code for my perspective transform includes a function called birds_eye(), which can get a view from top to below. By this view of the lanes, it is more sense to find a curve. Then unwarped to return to the original view.

The opencv function warp needs 4 origins and destinations points. The originas are like a trapezium in the lane. The destination is a rectangle.

Source to destination

```
src = np.float32([[490, 482],[810, 482], [1250, 720],[40, 720]])
```

```
dst = np.float32([[0, 0], [1280, 0], [1250, 720],[40, 720]])
```

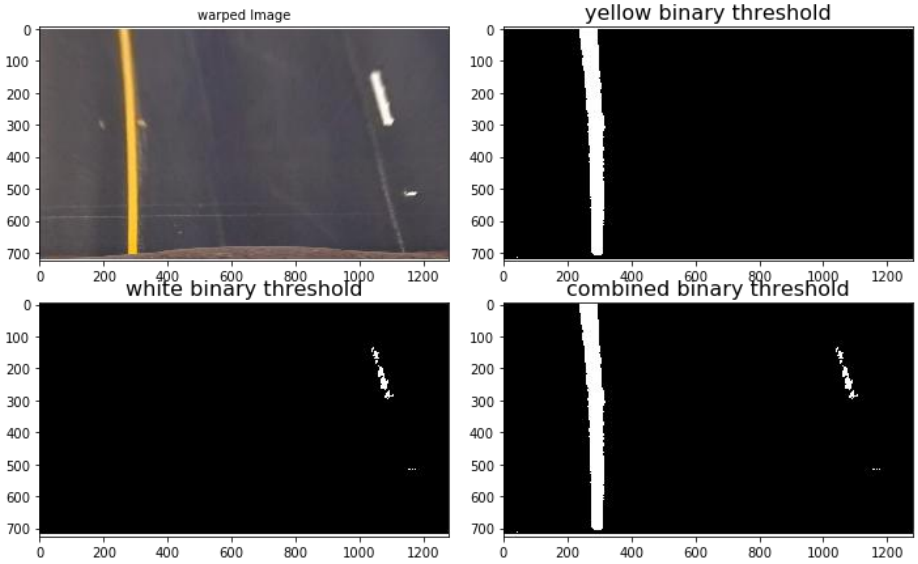
the effect of this transformation is:

3. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

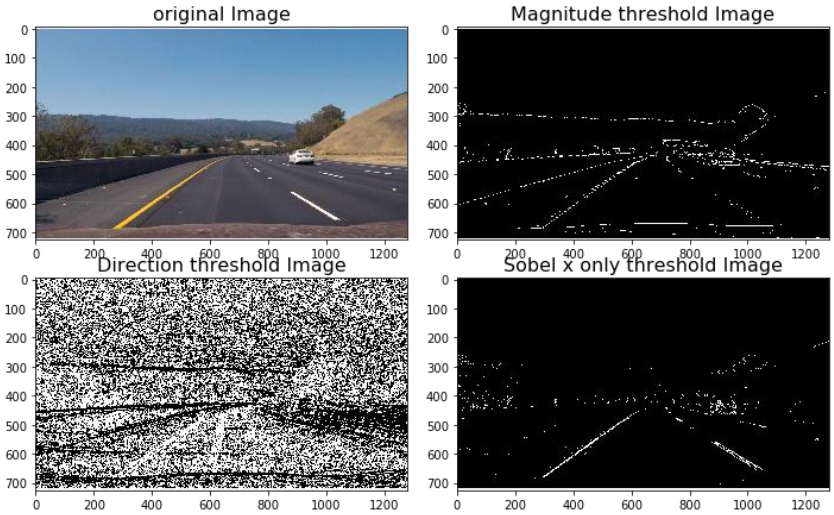
I used a combination of color and gradient thresholds to generate a binary image. Here's an example of my output for this step. The color threshold filter is used to pick only yellow and white elements, using opencv convert color to HSV space.

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

```
white_low=np.array([18,0,180])      white_high = np.array([255,80,255])
```



To find the contrast, we use the sobel operator. By tuning the threshold parameters, we can have an estimate of the lanes.



4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this: x = ay**2+by+c

In order to better estimate where the lane is, a histogram function is used on the bottom half of the image.

Then we divide the image in windows, and for each left and right window we find the mean of it, re/ centering the window.

5 Curvature of lanes and vehicle position with respect to center

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)

We assume the camera is mounted exactly in the center of the car.
Thus, the difference between the center of the image (1280 /2 = 640) and the middle point of beginning of lines if the offset (in pixels). This value times conversion factor is the estimate of offset.

((xL(720) + xR(720))/2−1280/2)* xm_per_pix

Input: curves of the lanes
Output: radius of curvature and offset from center

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

Here is an example of my result on a test image: