
Advanced Lane Finding Project

The goals / steps of this project are the following:

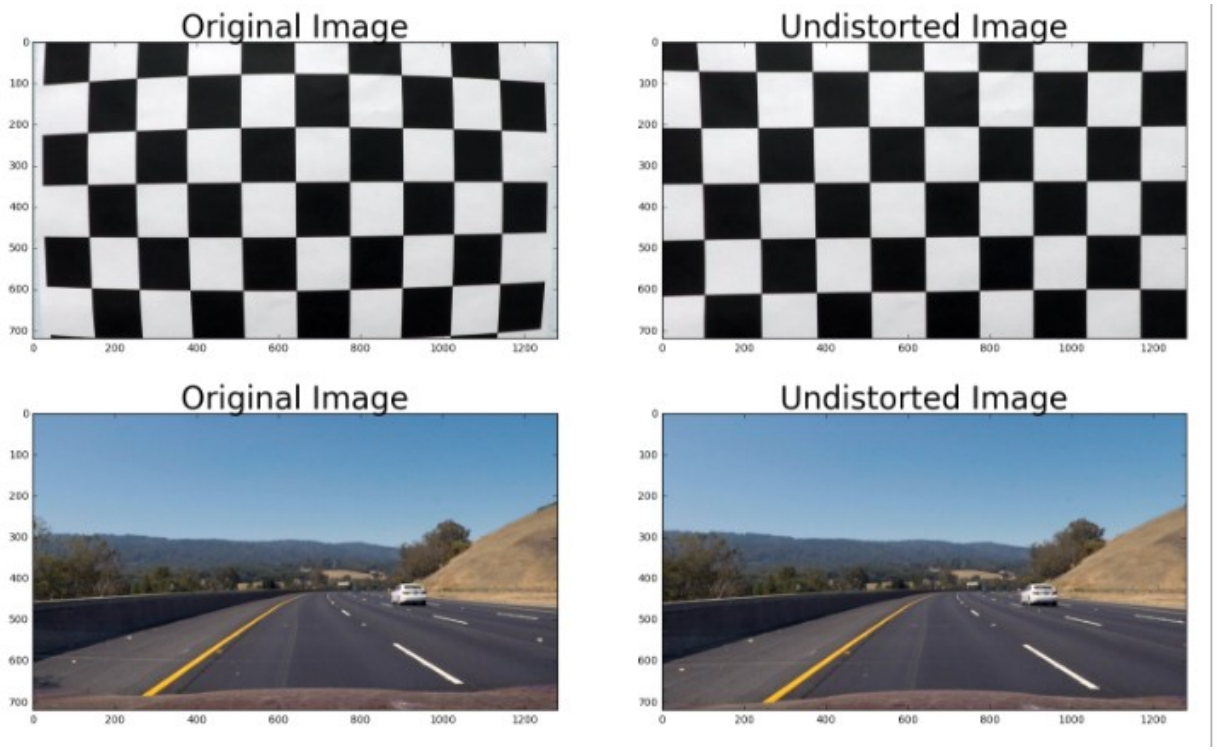
- •Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- •Apply a distortion correction to raw images.
- •Use color transforms, gradients, etc., to create a thresholded binary image.
- •Apply a perspective transform to rectify binary image ("birds-eye view").
- •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.

---

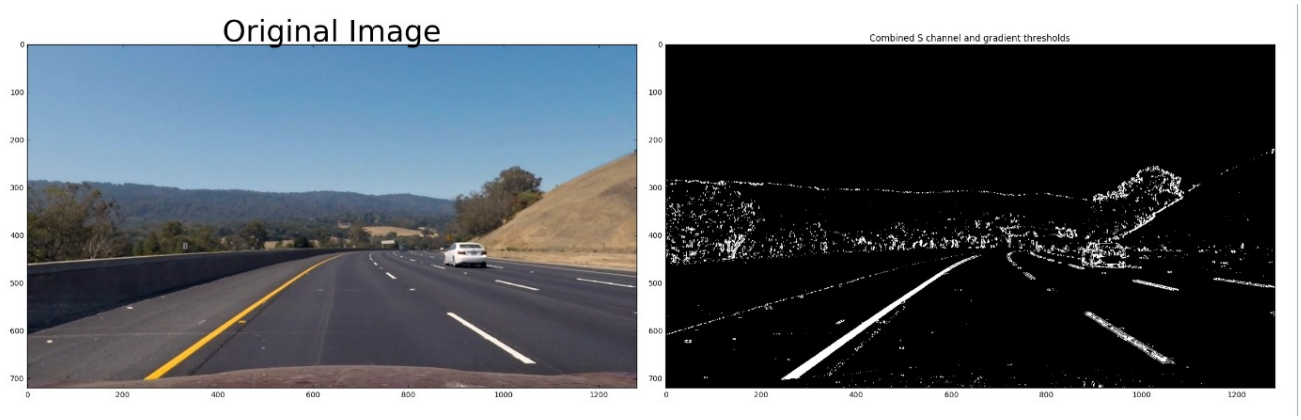
1. Camera Calibration

The code for this step is contained in the first code cell of the IPython notebook located in "./examples/example.ipynb"
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 following test images using the cv2.undistort() function and obtained this result:



2. Create a thresholded binary image. This can be found in cell 2 of my jupyter notebook I used a combination of color and gradient thresholds to generate a binary image. The pipeline function take in the undistorted image and return the thresholded binary image. Here's an example of my output for this step.



3. Perform a perspective transform

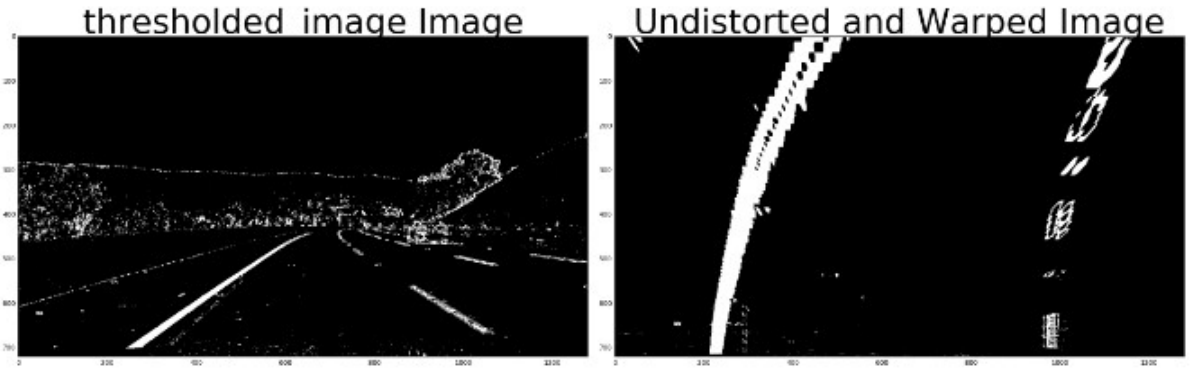
The code for my perspective transform can be found in cell 3 (ln 21).

The source point are taken to be
src_points = np.float32([[200,img_size[1]],[600,450],[700,450],[1130,img_size[1]]]) .

And the destination points are
dst_points = np.float32([[200,img_size[1]],[200,1],[970,1],[970,img_size[1]]])

The cv2.getPerspectiveTransform function takes as inputs as source (src) and destination (dst) points as mentioned above

I verified that my perspective transform was working as expected by drawing the src and dst points onto the thresholded image resulted fro step 2 and its warped counterpart to verify that the lines appear approximately parallel and of same same curvature in the warped image.



---

4. Identify lane-line pixels and fit their positions with a polynomial

After applying calibration, thresholding, and a perspective transform to a road image, I have a binary image where the lane lineym_per_pix = 30/720 # meters per pixel in y dimension

xm_per_pix = 3.7/700 # meters per pixel in x dimension

leftx = leftx*xm_per_pix

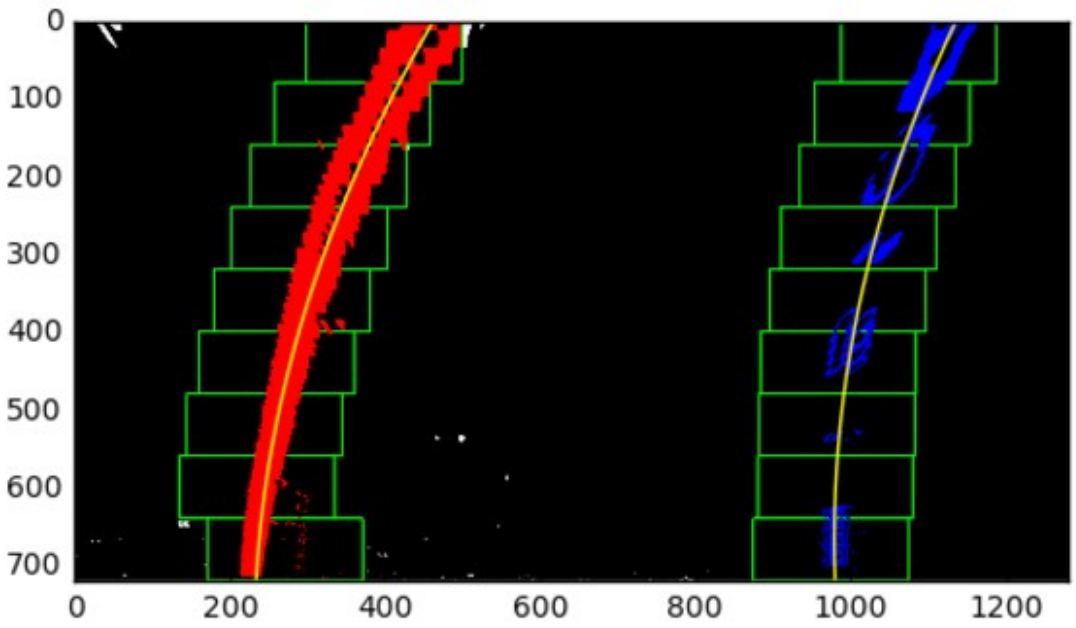
lefty = lefty*ym_per_pix

rightx = rightx*xm_per_pix

```
righty = righty*ym_per_pix



# Fit a second order polynomial to each

left_fit = np.polyfit(lefty, leftx, 2)

right_fit = np.polyfit(righty, rightx, 2)

y_eval = np.max(ploty)

left_curverad = ((1 + (2*left_fit[0]*y_eval + left_fit[1])**2)**1.5) /
np.absolute(2*left_fit[0])

right_curverad = ((1 + (2*right_fit[0]*y_eval + right_fit[1])**2)**1.5) /
np.absolute(2*right_fit[0])

#  Now our radius of curvature is in meters

print('Left Curvature in world space is:', left_curverad, 'm and Right Curvature in worlds
pace is:', right_curverad, 'm')

print('')

# ------------------------------------------------------------------------------------------

# Finding distnce of vehicle from the center

x_avg = (left_fitx[-1] + right_fitx[-1])/2

vehiclePosWRTCenter = xm_per_pix*(midpoint-x_avg)

if vehiclePosWRTCenter > 0:

    print('Vehicle is ',vehiclePosWRTCenter, 'm left from  the center')

else:

    print('Vehicle is ',vehiclePosWRTCenter, 'm right from  the center')s stand out clearly.
```
The code for dentify lane-line pixels and fit their positions with a polynomial is in cell 4. I first take a **histogram** along all the columns in the *lower half* of the image. With this histogram I am adding up the pixel values along each column in the image. In my thresholded binary image, pixels are either 0 or 1, so the two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. I can use that as a starting point for where to search for the lines. From that point, I can use a sliding window, placed around the line centers, to find and follow the lines up to

the top of the frame. Once I know position of all the widows, I can find the pixels positions in each of the windows (called leftx, lefty, rightx, righty in the code where leftx means x cordinate of pixels in left lane, lefty means y coordinate of pixels in left lane, and same thing for rightx and righty).

The following are my results:



5. Calculate the radius of curvature of the lane and the position of the vehicle with respect to center.

This is done in cell 4. I compute radius of curvature for both pixel space as well as real world space.

Once we have got the points leftx, lefty, rightx, righty in the previous step, these can be used to find a quadratic polynomial using np.polyfit as follows:

```
left_fit = np.polyfit(lefty, leftx, 2)

right_fit = np.polyfit(righty, rightx, 2)
```

To draw the curve, we find x values for all the y values in the image:

```
ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )

left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]

right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
```

To find the distance of the vehvile from the center, I assume the camera is mounted at the center of the car, such that the lane center is the midpoint at the bottom of the image between the two quadratic curves that detected. The offset of the lane center from the center of the image (converted from pixels to meters) is car's distance from the center of the lane. Below, *left_fitx[-1] and right_fitx[-1]* correspond to x values of theleft and right curve at the maximum possible y value (number of rows in the image)

```
x_avg = (left_fitx[-1] + right_fitx[-1])/2

vehiclePosWRTCenter = xm_per_pix*(midpoint-x_avg)

If vehiclePosWRTCenter > 0:

    cv2.putText(result,'Vehicle is ' + str(abs(vehiclePosWRTCenter)) + 'm on left of
        center',(0,200), cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2)

else:

    cv2.putText(result,'Vehicle is ' + str(abs(vehiclePosWRTCenter)) + 'm on right of
        center',(0,200),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2)
```

The results of my code from the image generated from setp 4 are as follows:

```
Left Curvature in pixel space is: 1385.3851702 and Right Curvature in pixel
space is: 1421.67223567


Left Curvature in world space is: 2706.48481727 m and Right Curvature in
worlds pace is: 2668.27028269 m

Vehicle is  0.175565927511 m from  the center
```

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

I implemented this step in cell 5. results are as follows:

NOTE THAT THE RESULT FOR OTHER TEST IMAGES ARE SAVED IN THE OUTPUT_IMAGES DIRECTORY

###Pipeline (video)

the video can be found in my jupyter notebook in the videos directory by the name project_video_output.mp4

###Discussion

####1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

I faced issues like finding the right thresholds and correct source and destination points to be used for warping.

The pipeline is will fail when the road is not flat or if there are bumps. It will also fail if the car departs continuously away from the center This is because right now the source and destination points are hardcoded for perspective transformation.

To improve the pipeline, I will have to maintain an estimate of current best fit and radius of curvature, and then use it whenever sanity check is not met, like if radius of cuvature is changing drastically, or if the lane lines are not parallel.