Writeup

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Advanced Lane Finding Project

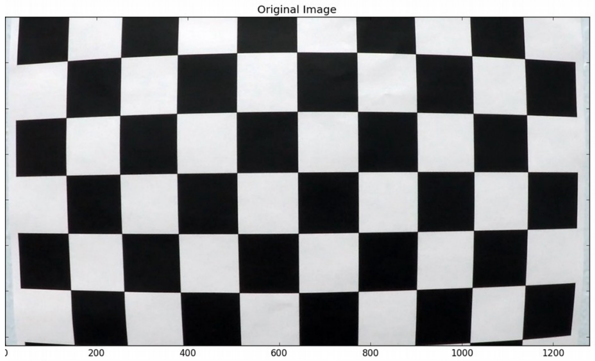
The goals / steps of this project are the following:

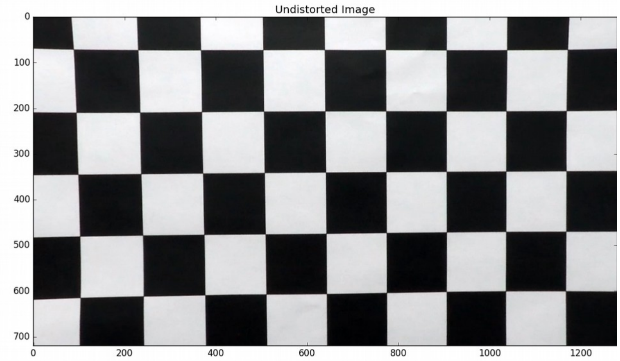
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").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

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 using the `cv2.undistort()` function and obtained this result:

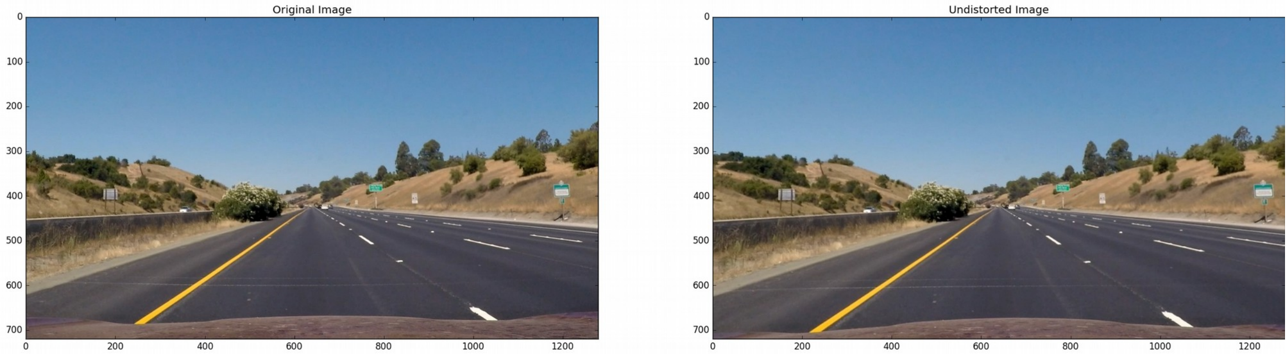




2. Apply a distortion correction to raw images

read in images in test_images folder, using the function cv2.undistort(img, mtx, dist, None, mtx) to undistort each image, then write to output_images folder.

```
# undistort test image and write to output_images folder
if not os.path.exists('output_images'):
    output_path = 'output_images'
    for img_name in glob.glob('test_images/*.jpg'):
        img = cv2.imread(img_name)
        undist = cv2.undistort(img, mtx, dist, None, mtx)
        # print(output_path+'/'+img_name.split('/')[1])
        cv2.imwrite(output_path+'/'+img_name.split('/')[1],undist)
```

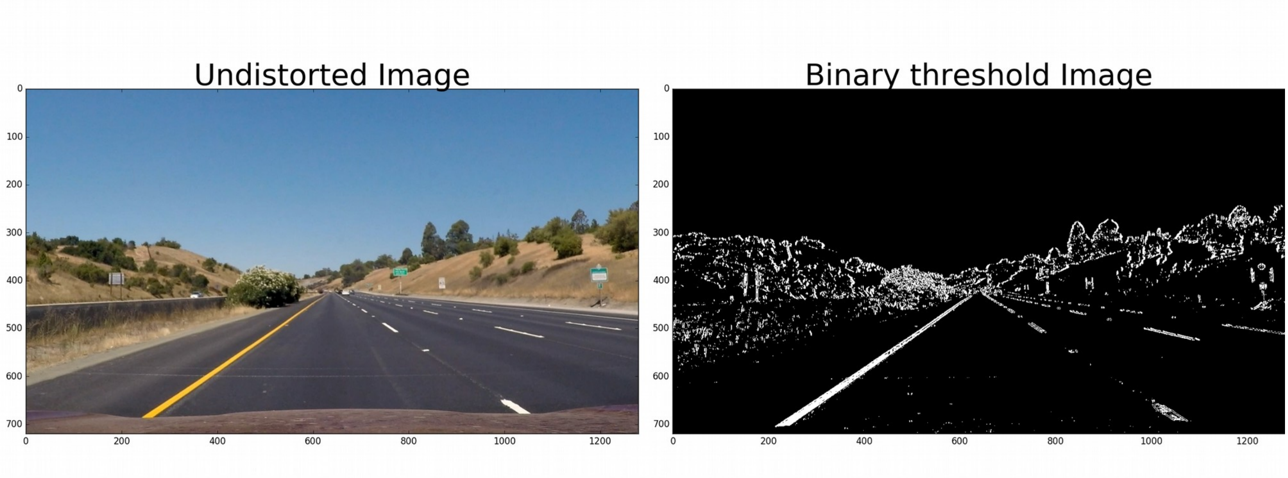


3. Use color transforms, gradients, etc., to create a thresholded binary image.

```
hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
h_channel = hls[:,:,0]
l_channel = hls[:,:,1]
s_channel = hls[:,:,2]
# Threshold color channel
s_thresh=(180, 255)
s_binary = np.zeros_like(s_channel)
s_binary[s_channel >= s_thresh[0]) & (s_channel <= s_thresh[1]] = 1

# Sobel x
sobelx = cv2.Sobel(l_channel, cv2.CV_64F, 1, 0) # Take the derivative in x
abs_sobelx = np.absolute(sobelx) # Absolute x derivative to accentuate lines away from horizontal
scaled_sobel = np.uint8(255*abs_sobelx/np.max(abs_sobelx))
# Threshold x gradient
sxbinary = np.zeros_like(scaled_sobel)
sxbinary[(scaled_sobel >= sx_thresh[0]) & (scaled_sobel <= sx_thresh[1])] = 1

# Stack each channel, gradient output in green, color output in blue channel
color_binary = np.dstack(( np.zeros_like(sxbinary), sxbinary, s_binary)) * 255
```

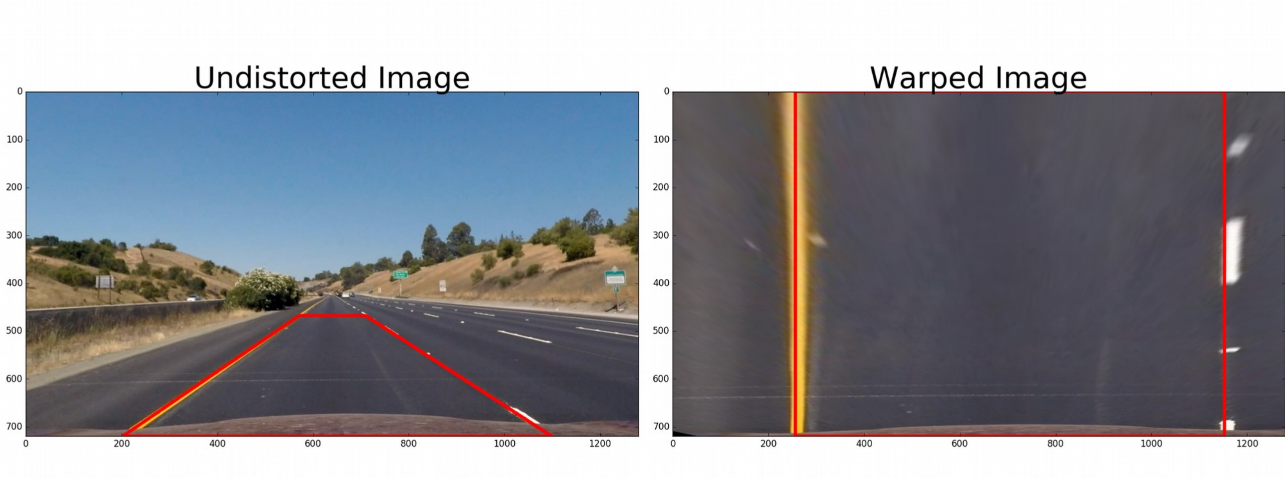


4. Apply a perspective transform to rectify binary image ("birds-eye view").

```
# select 4 source points
src = [[w*0.45+10,h*0.65],[w*0.55+30,h*0.65],[w*0.9, h],[w*0.2, h]]
# select 4 destination points
dst = [[w*0.2,0],[w*0.9,0],[w*0.9, h],[w*0.2, h]]

# use src, dst points to compute M
M = cv2.getPerspectiveTransform(src, dst)
# Warp an image using the perspective transform, M
warped = cv2.warpPerspective(img, M, (w,h), flags=cv2.INTER_LINEAR)
```

to illustrate the perspective transform effect. I draw source points in left image and draw destination points onto the warped image by contrast. The lane lines approximately appear parallel in warped view.



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

First get left line base, right line base by histogram, assuming given binary image.

```
# Take a histogram of the bottom half of the image
histogram = np.sum(binary_warped[binary_warped.shape[0]//2:,:], axis=0)
# Find the peak of the left and right halves of the histogram
midpoint = np.int(histogram.shape[0]//2)
leftx_base = np.argmax(histogram[:midpoint])
rightx_base = np.argmax(histogram[midpoint:]) + midpoint

# HYPERPARAMETERS
# Choose the number of sliding windows
```

```
nwindows = 9
# Set the width of the windows +/- margin
margin = 100
# Set minimum number of pixels found to recenter window
minpix = 50

# Set height of windows - based on nwindows above and image shape
window_height = np.int(binary_warped.shape[0]//nwindows)
# Identify the x and y positions of all nonzero pixels in the image
nonzero = binary_warped.nonzero()
nonzeroy = np.array(nonzero[0])
nonzerox = np.array(nonzero[1])

# Current positions to be updated later for each window in nwindows
leftx_current = leftx_base
rightx_current = rightx_base

# Create empty lists to receive left and right lane pixel indices
left_lane_inds = []
right_lane_inds = []
```

Loop through each window in `nwindows`

1.Find the boundaries of our current window. This is based on a combination of the current window's starting point (`leftx_current` and `rightx_current`), as well as the `margin` you set in the hyperparameters.

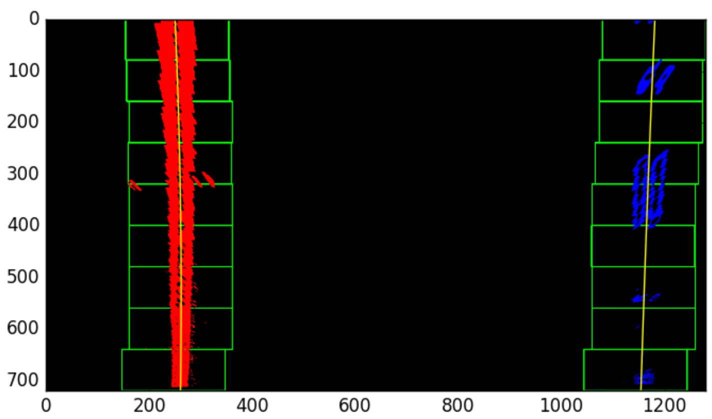
2.Use `cv2.rectangle` to draw these window boundaries onto our visualization image `out_img`. This is required for the quiz, but you can skip this step in practice if you don't need to visualize where the windows are.

3.Now that we know the boundaries of our window, find out which activated pixels from `nonzeroy`and `nonzerox` above actually fall into the window.

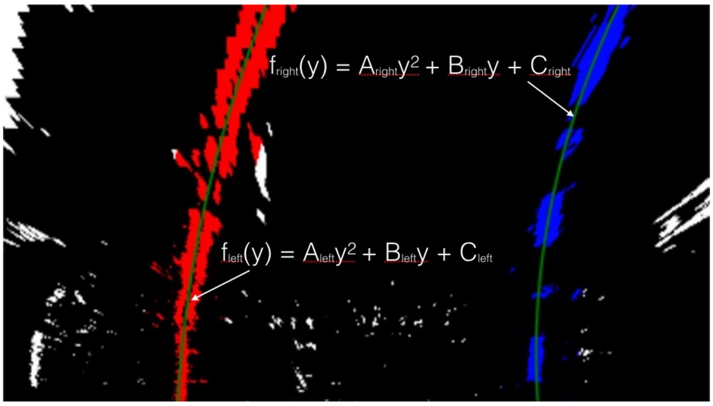
4.Append these to our lists `left_lane_inds` and `right_lane_inds`.

5.If the number of pixels you found in Step 4 are greater than your hyperparameter `minpix`, re-center our window (i.e. `leftx_current` or `rightx_current`) based on the mean position of these pixels.

The result as below:



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

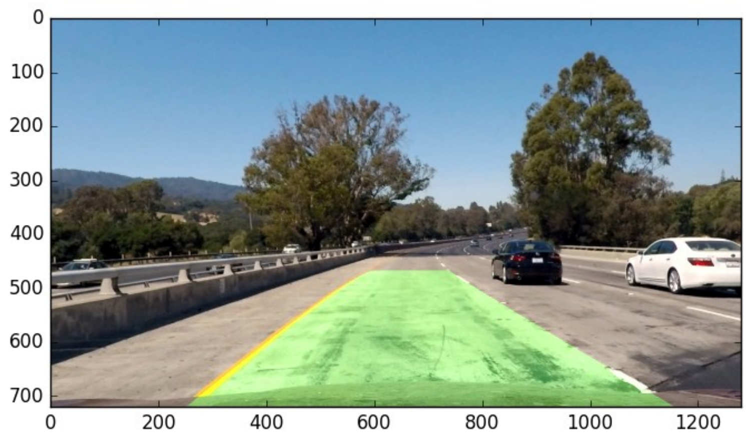


f(y) = A * (y ** 2) + B * y + C
f'(y) = 2 * A + B
f''(y) = 2 * A
R_curve = ((1 + (2 * A * y) ** 2) ** (3/2))/(abs(2A))

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

```
# Recast the x and y points into usable format for cv2.fillPoly()
pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
pts = np.hstack((pts_left, pts_right))

# Draw the lane onto the warped blank image
cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))
# compute inverse M transformation martix
Minv = cv2.getPerspectiveTransform(dst, src)
# Warp the blank back to original image space using inverse perspective matrix (Minv)
newwarp = cv2.warpPerspective(color_warp, Minv, (img.shape[1], img.shape[0]))
```



Pipeline:

Disscussion: