Writeup

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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Camera Calibration

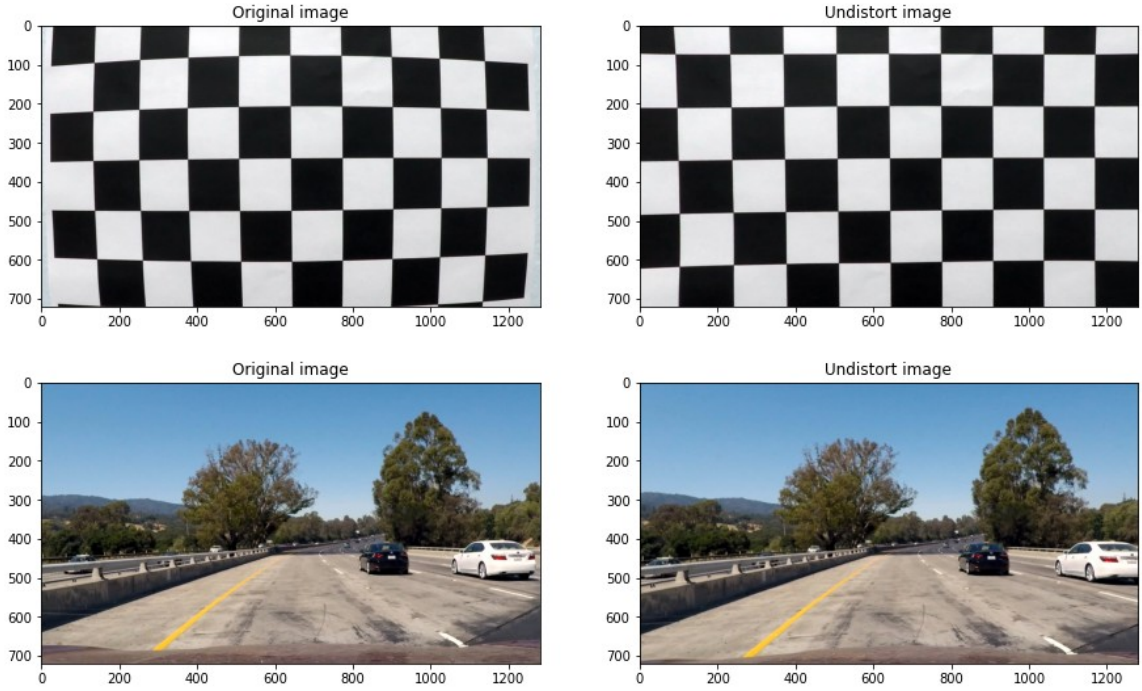
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the first code cell of the IPython notebook located in ".project2 v3.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 the test image using the `cv2.undistort()` function and obtained this result:

```
Out[28]: True
```

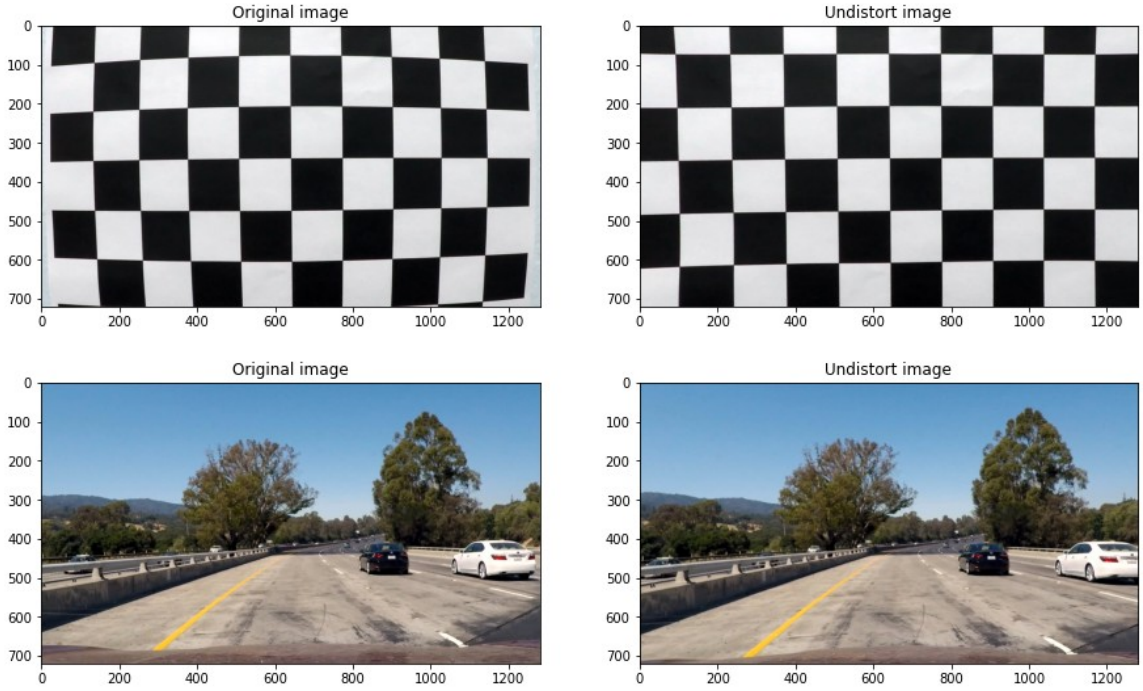


Pipeline (single images)

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:

```
Out[28]: True
```



2. 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.

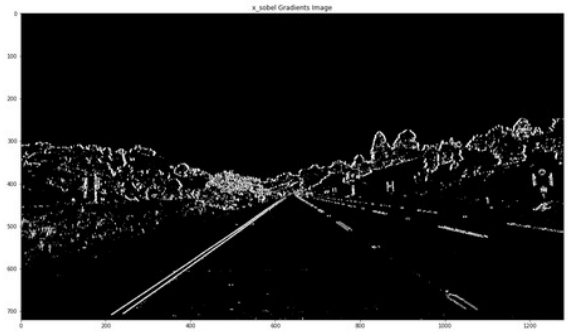
thresholding steps:

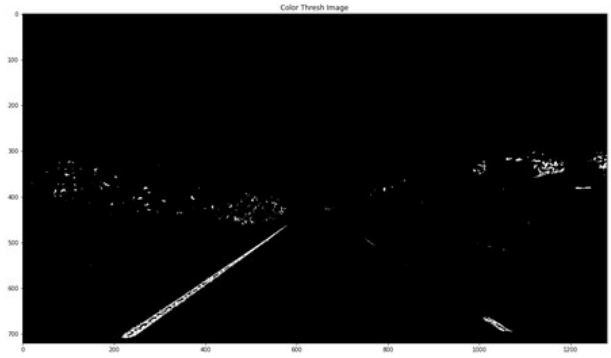
```
def dir_threshold(img, sobel_kernel=3, thresh=(0, np.pi/2)):
    # Grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Calculate the x and y gradients
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    # Take the absolute value of the gradient direction,
    # apply a threshold, and create a binary image result
    absgraddir = np.arctan2(np.absolute(sobely), np.absolute(sobelx))
    #print(absgraddir)
    binary_output =  np.zeros_like(absgraddir)
    binary_output[(absgraddir >= thresh[0]) & (absgraddir <= thresh[1])] = 1

    # Return the binary image
    return binary_output

def hls_select(img, thresh=(0, 255)):
    # 1) Convert to HLS color space
    hls = cv2.cvtColor(img,cv2.COLOR_RGB2HLS)
    # 2) Apply a threshold to the S channel
    s_channel = hls[:,:,2]
    # 3) Return a binary image of threshold result
    binary_output = np.zeros_like(s_channel)
    binary_output[(s_channel > thresh[0]) & (s_channel <thresh[1])] = 1
    return binary_output
```

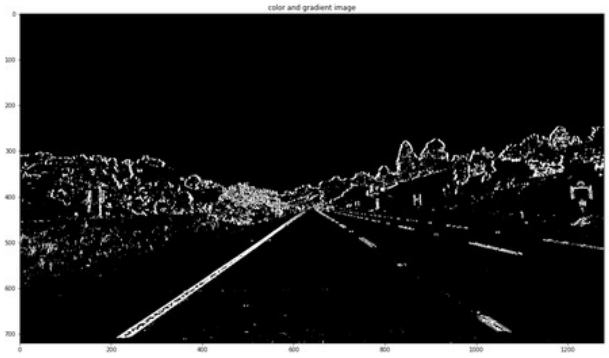
```
def abs_sobel_thresh(img,orient = 'x',sobel_kernel = 3,thresh = (0,255)):
    gray = grayscale(img)

    if orient == 'x':
        abs_sobel = np.absolute(cv2.Sobel(gray,cv2.CV_64F,1,0,ksize = sobel_kernel))
    if orient == 'y':
        abs_sobel = np.absolute(cv2.Sobel(gray,cv2.CV_64F,0,1,ksize = sobel_kernel))

    scaled_sobel = np.uint8(255 * abs_sobel / np.max(abs_sobel))
    binary_output = np.zeros_like(scaled_sobel)
    binary_output[(scaled_sobel >= thresh[0]) & (scaled_sobel <= thresh[1])] = 1
    return binary_output

def mag_thresh(img, sobel_kernel=3, thresh=(0, 255)):

    # Apply the following steps to img
    # 1) Convert to grayscale
    gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    # 2) Take the gradient in x and y separatel
    sobel_x = cv2.Sobel(gray,cv2.CV_64F,1,0,ksize = sobel_kernel)
    #print(sobel_x)
    sobel_y = cv2.Sobel(gray,cv2.CV_64F,0,1,ksize = sobel_kernel)
    # 3) Calculate the magnitude
    magnitude = np.sqrt(sobel_x ** 2 + sobel_y ** 2)
    # 4) Scale to 8-bit (0 - 255) and convert to type = np.uint8
    scale_factor = np.max(magnitude) / 255
    #print('scale_factor = ',scale_factor)
    magnitude = (magnitude / scale_factor).astype(np.uint8)
    # 5) Create a binary mask where mag thresholds are met
    binary_output = np.zeros_like(magnitude)
    # 6) Return this mask as your binary_output image
    binary_output[(magnitude >= thresh[0]) & (magnitude <= thresh[1])] = 1
    return binary_output
```

Here's an example of my output for this step.

```
Out[34]: True
```







3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

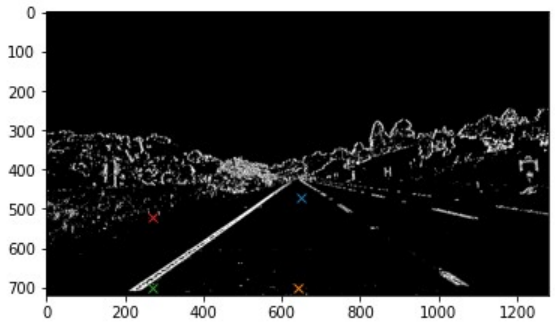
```
#apply perspective tansform and warp of the road lanes

def warp(img):
    img_size = (img.shape[1], img.shape[0])

    src = np.float32([[800,510],[1150,700],[270,700],[510,510]])
    dst = np.float32([[650,470],[640,700],[270,700],[270,540]])
    M = cv2.getPerspectiveTransform(src,dst)

    Minv = cv2.getPerspectiveTransform(dst,src)

    warped = cv2.warpPerspective(img,M,img_size,flags = cv2.INTER_LINEAR)

    unpersp = cv2.warpPerspective(warped, Minv, img_size, flags=cv2.INTER_LINEAR)

    return warped, unpersp, Minv


warped_img, unpersp, Minv = warp(color_x_sobel)
```

The code for my perspective transform includes a function called `warp()`. The warp() 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 in the following manner:

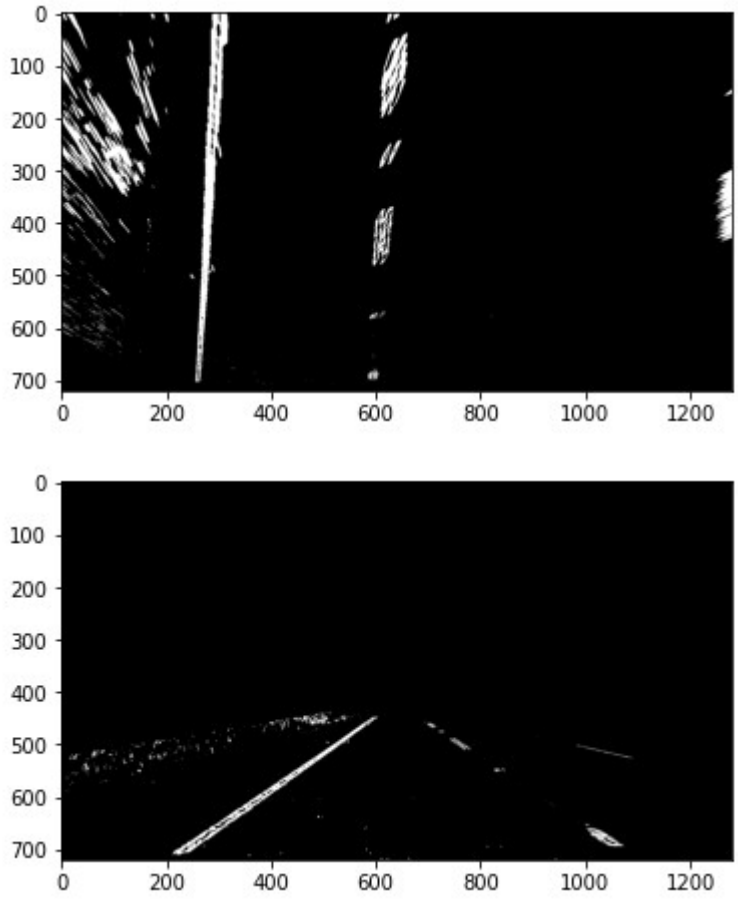
This resulted in the following source and destination points:

Source	Destination
850, 510	650, 470
1150, 700	640, 700
270, 700	270, 700
510, 510	270, 540

```
Out[32]: [<matplotlib.lines.Line2D at 0x7f51ed52e1f0>]
```

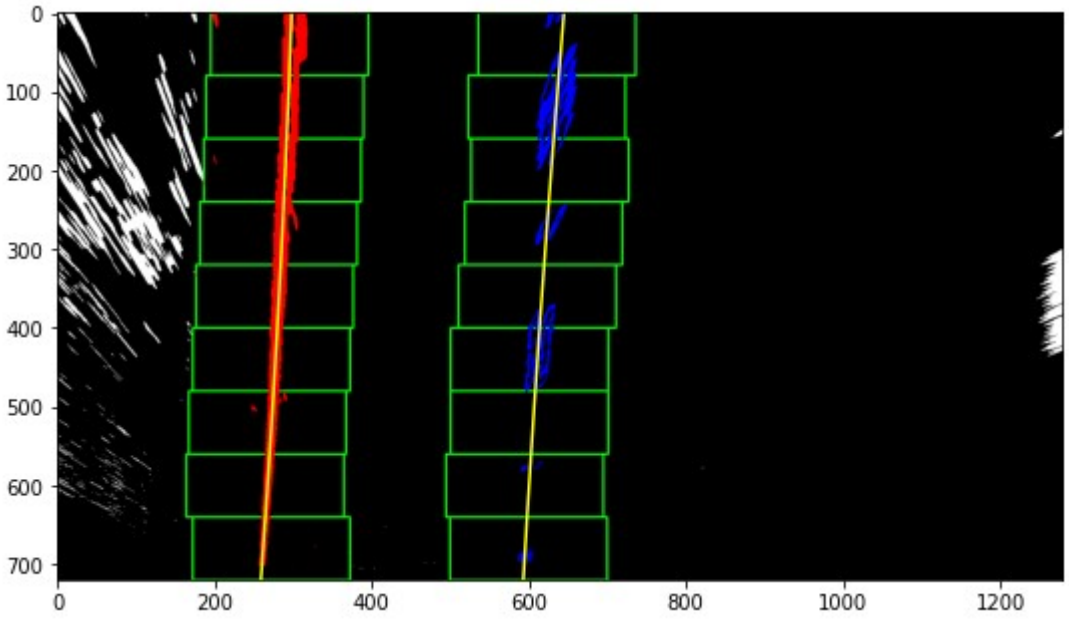


I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.



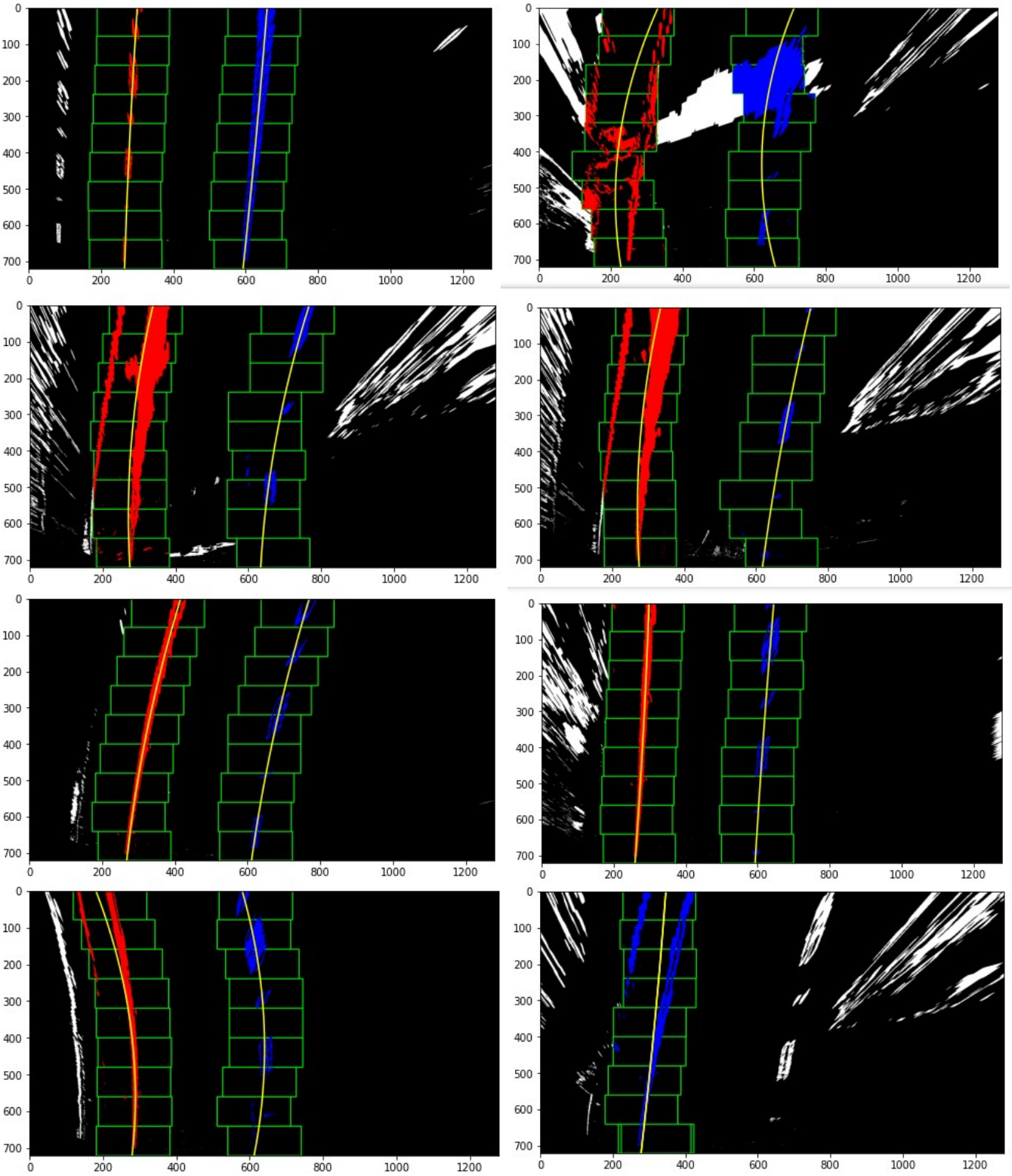
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Then I use find_lines() function to detect drive lanes pixels and find lane edges to fit my lane lines with a 2nd order polynomial kinda like this:



5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I use curvature() function to define the curvature of the drive lanes / car to 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 my code in the functions show_info() and draw_lines() to twist the edges detected into the original plot and draw lines.

Here are examples of my result on a test image:

Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

The video is included in my Github project folder as result_video_v2.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 found in the video that the outer side of the driver lane when it's turning can be detected wrongly by my codes. The possible reasons can be that when turning, the lighting environment changed , causing my thresholds not suitable for the clip pics for a sec.~~

By improving my find_lines() function into several different functions line_fit() & turn_fit() with detailed focusing functions I was able to improve the line finding and fitting.

```
tune_fit(binary_warped, left_fit, right_fit):
"""
Given a previously fit line, quickly try to find the line based on previous lines
"""
# Assume you now have a new warped binary image
# from the next frame of video (also called "binary_warped")
# It's now much easier to find line pixels!
nonzero = binary_warped.nonzero()
nonzeroy = np.array(nonzero[0])
nonzerox = np.array(nonzero[1])
margin = 100
left_lane_inds = ((nonzerox > (left_fit[0]*(nonzeroy**2) + left_fit[1]*nonzeroy + l
right_lane_inds = ((nonzerox > (right_fit[0]*(nonzeroy**2) + right_fit[1]*nonzeroy

# Again, extract left and right line pixel positions
leftx = nonzerox[left_lane_inds]
lefty = nonzeroy[left_lane_inds]
rightx = nonzerox[right_lane_inds]
righty = nonzeroy[right_lane_inds]

# If we don't find enough relevant points, return all None (this means error)
min_inds = 10
if lefty.shape[0] < min_inds or righty.shape[0] < min_inds:
    return None

# Fit a second order polynomial to each
left_fit = np.polyfit(lefty, leftx, 2)
right_fit = np.polyfit(righty, rightx, 2)
# Generate x and y values for plotting
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]

# Return a dict of relevant variables
ret = {}
ret['left_fit'] = left_fit
ret['right_fit'] = right_fit
ret['nonzerox'] = nonzerox
ret['nonzeroy'] = nonzeroy
ret['left_lane_inds'] = left_lane_inds
ret['right_lane_inds'] = right_lane_inds

return ret
```