Advanced Lane Finding

Yuda Wang

Self-driving Car Nanodegree

Udacity

August 2017

Contents

Chapter 1

Introduction

This is the fourth project of self-driving car nanodegree, term 1.

In this project, several functions should be included accroding to the writeup template :

" 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."

The difficulty should be the after the 3rd step. Here is the main change I've made.

In the 3rd step, there are many hyper-parameters. The tradeoff between the selectivity and sensitivity is hard, for example, the sunlight may make the lane invisible, the tree shadow may make the lane invisible, the pitch between the road may make the lane invisible. But the reasons are different. So a multicase strategy is used in this project.

In the 4th step, the each source point from the 4 corners were taken from the mean point from each corners in the straight-line image. In case of false detection, two ways of warp strategy are used. When false detection, the second warp strategy will reduce the lane image height, thus the new lane image will be evaluated again in the next steps.

In the 5th step, if an exception happened(empty left and right inds), it will touch a value.

In the 6th step, if the curvature rad is higher than 5000, it will be 5000 and be recognized as straight line. The sanity check includes: parallel check, reasonable two lane line distance check, and curvature difference check.

In the 7th and 8th step, if the sanity check is failed, the image will return to the 2nd warp strategy in the 4th step, because the image height is reduced, the curvature is not accurate, the result will not be showed at the display.

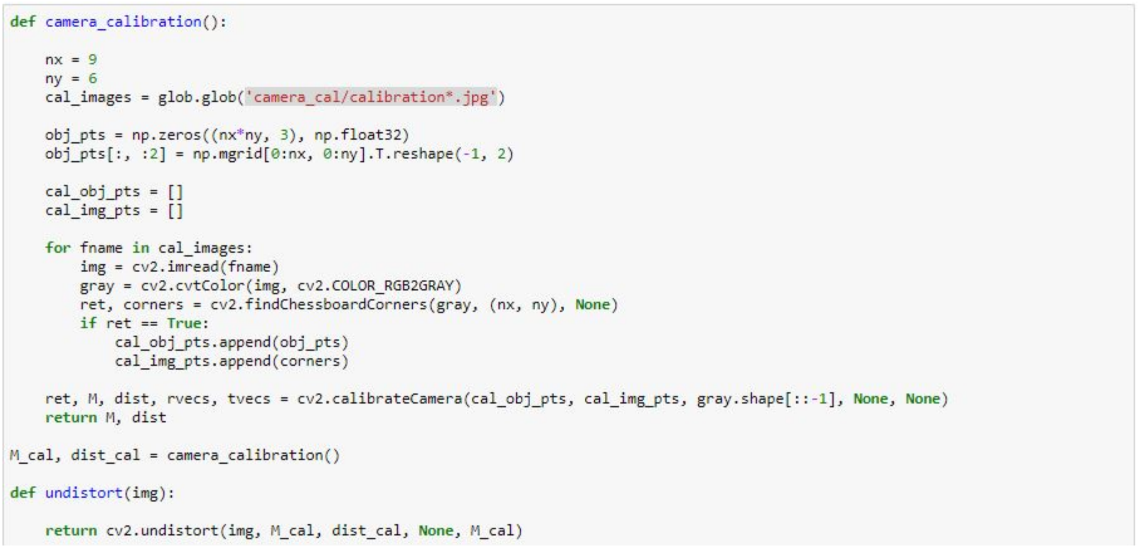
For the time issue, I didn't make improvement on the smooth and history strategy in the 'tips and tricks for the project'.

A detailed discussion will show up in the next chapters.

Chapter 2

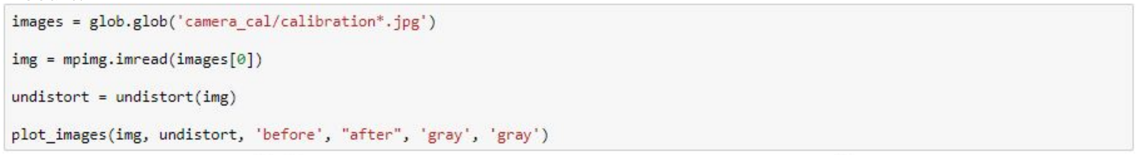
Camera Calibration

This is the beginning of the project. The whole process is similar with the example code.

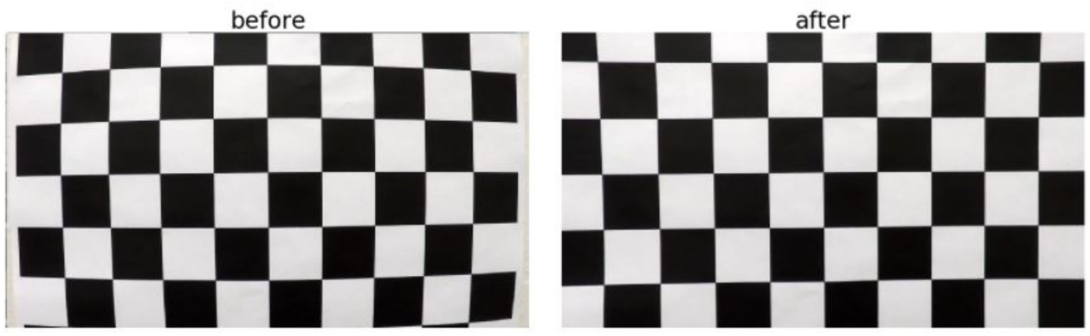


```
def camera_calibration():

    nx = 9
    ny = 6
    cal_images = glob.glob('camera_cal/calibration*.jpg')

    obj_pts = np.zeros((nx*ny, 3), np.float32)
    obj_pts[:, :2] = np.mgrid[0:nx, 0:ny].T.reshape(-1, 2)

    cal_obj_pts = []
    cal_img_pts = []

    for fname in cal_images:
        img = cv2.imread(fname)
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)
        if ret == True:
            cal_obj_pts.append(obj_pts)
            cal_img_pts.append(corners)

    ret, M, dist, rvecs, tvecs = cv2.calibrateCamera(cal_obj_pts, cal_img_pts, gray.shape[::-1], None, None)
    return M, dist

M_cal, dist_cal = camera_calibration()

def undistort(img):

    return cv2.undistort(img, M_cal, dist_cal, None, M_cal)
```

Result:



```
images = glob.glob('camera_cal/calibration*.jpg')

img = mpimg.imread(images[0])

undistort = undistort(img)

plot_images(img, undistort, 'before', "after", 'gray', 'gray')
```

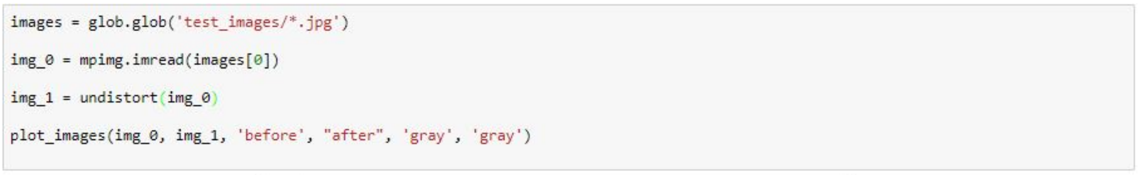


Chapter 3

Pipeline

3.1 Undistort

Undistort is the first step.



```
images = glob.glob('test_images/*.jpg')

img_0 = mpimg.imread(images[0])

img_1 = undistort(img_0)

plot_images(img_0, img_1, 'before', "after", 'gray', 'gray')
```

3.2 Combined Threshold

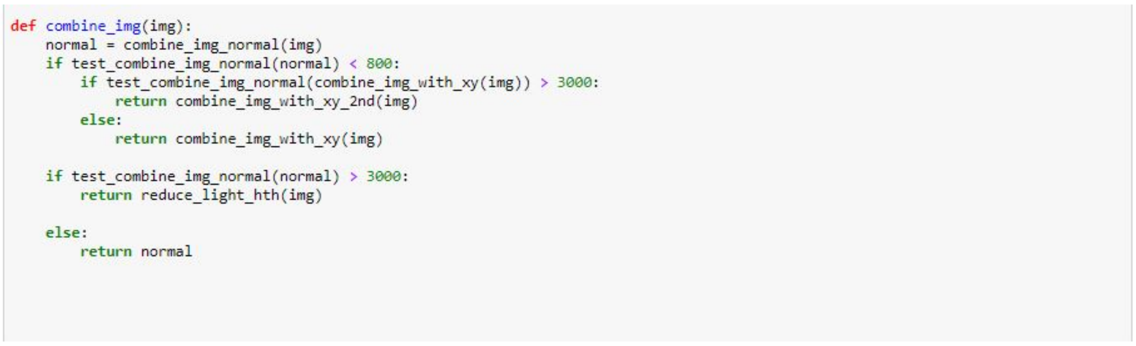
As mentioned in the introduction, multi-case threshold strategy could handle the difficulties in the tradeoffs between selectivity and sensitivity.

When our human brain process the image, we will concentrate on the color threshold first. As the light is too shining or too dark, we will reprocess the image and concentrate on the gradient threshold.

My multi-case threshold strategy is similar to this.

Check the the amount of pixels in the first strategy is in a normal range. If the amount is small, use gradient xy, if the gradient xy is not in a normal range, combine with dir threshold. If the amount is too large, because the problem is mainly caused by the sunlight, still use color threshold strategy with higher limitation.

I've tried to do the regression to make the function logic works better. However, the delay will be large. So I use several if-else to implement this logic.



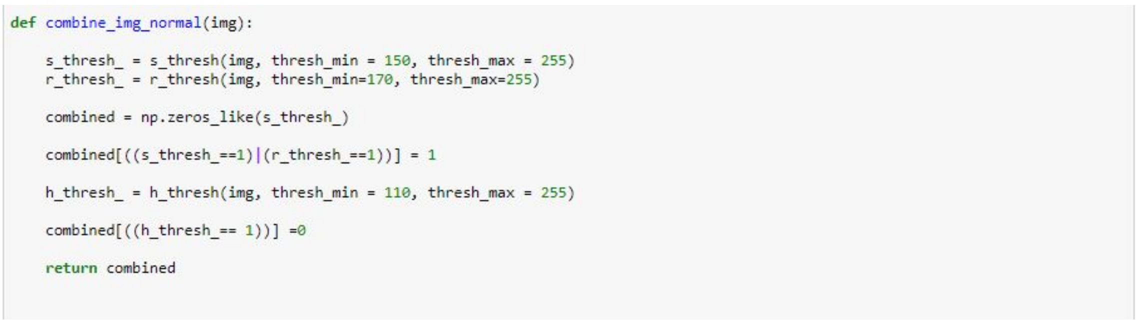
```
def combine_img(img):
    normal = combine_img_normal(img)
    if test_combine_img_normal(normal) < 800:
        if test_combine_img_normal(combine_img_with_xy(img)) > 3000:
            return combine_img_with_xy_2nd(img)
        else:
            return combine_img_with_xy(img)

    if test_combine_img_normal(normal) > 3000:
        return reduce_light_hth(img)

    else:
        return normal
```

3.2.1 Prepare Test Set

The augmented test set is from the screenshots from the ’project$_{v}ideo','challenge_{v}ideo','harder_{c}hallenge_{v}ideo'$.

Most of the cases maybe included in the test set.

3.2.2 Basic Color Threshold



```
def combine_img_normal(img):

    s_thresh_ = s_thresh(img, thresh_min = 150, thresh_max = 255)
    r_thresh_ = r_thresh(img, thresh_min=170, thresh_max=255)

    combined = np.zeros_like(s_thresh_)

    combined[((s_thresh_==1)|(r_thresh_==1))] = 1

    h_thresh_ = h_thresh(img, thresh_min = 110, thresh_max = 255)

    combined[((h_thresh_== 1))] =0

    return combined
```

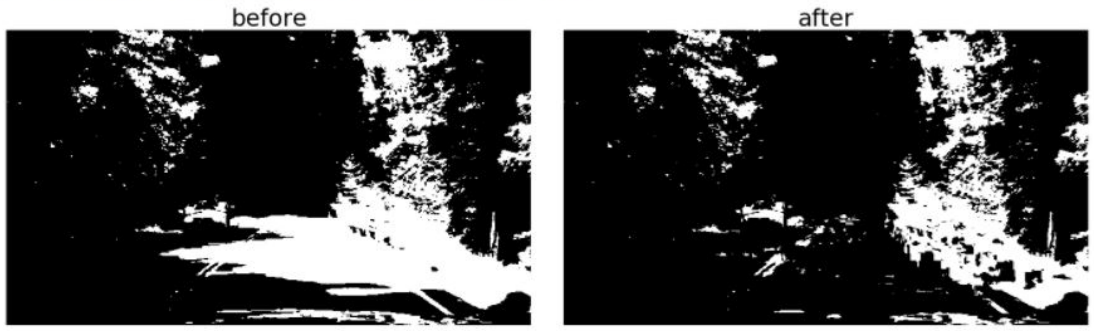
If the result is ’normal’, the combined threshold will return the basic R and S color threshold, most of the road is relevant to H threshold, so the Hue color threshold will be used to reduce the road background.

Hue threshold is necessary. For example,

```
img = undistort(mpimg.imread(images[11]))

plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0xbd09ef0>
```

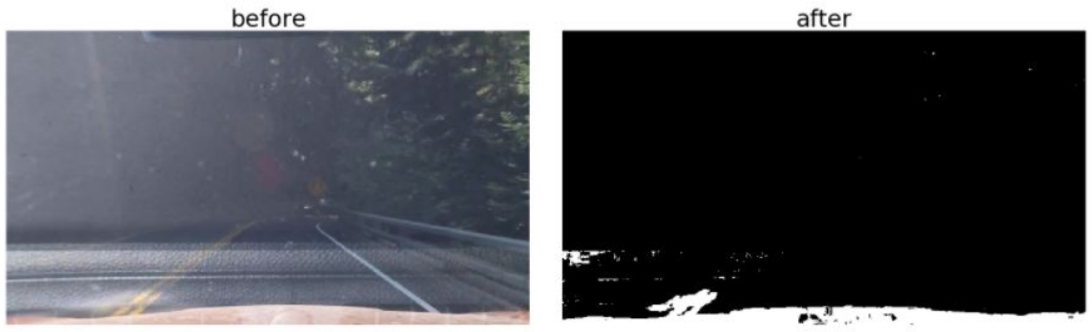
```
s_thresh_ = s_thresh(img, thresh_min = 150, thresh_max = 255)
r_thresh_ = r_thresh(img, thresh_min=170, thresh_max=255)

combined = np.zeros_like(s_thresh_)
combined_2nd = np.zeros_like(s_thresh_)

combined[((s_thresh_==1)|(r_thresh_==1))] = 1
combined_2nd[((s_thresh_==1)|(r_thresh_==1))] = 1


h_thresh_ = h_thresh(img, thresh_min = 110, thresh_max = 255)

combined_2nd[((h_thresh_== 1))] =0

plot_images(combined , combined_2nd, 'before', "after", 'gray', 'gray')
```



3.2.3 Gradient Threshold

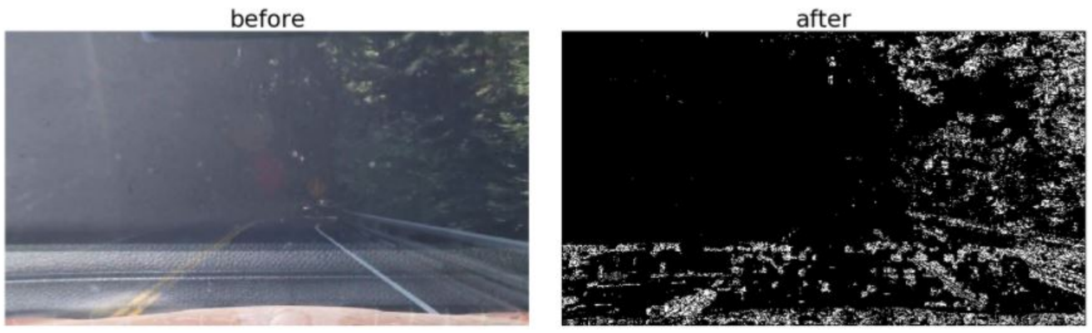
In some cases, basic color threshold can't handle the light reflection of the front car window.

```
img = undistort(mpimg.imread(images[3]))

after = combine_img_normal(img)

plot_images(img, after, 'before', "after", 'gray', 'gray')
```



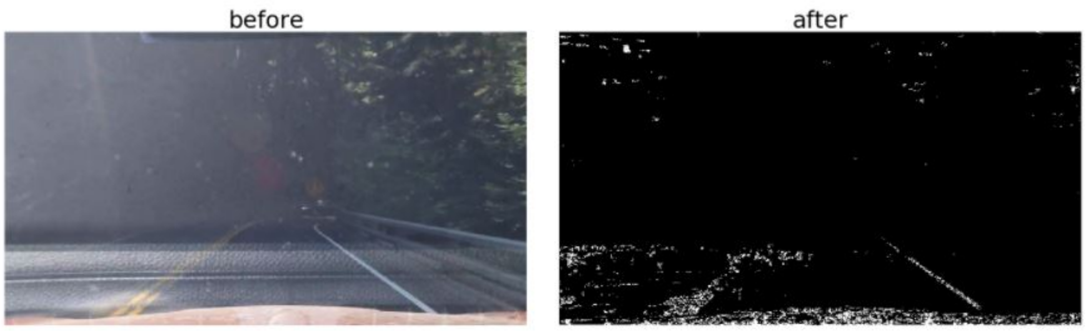
Gradient xy threshold with hue threshold can handle this situation.

```
img = undistort(mpimg.imread(images[3]))

after = combine_img_with_xy(img)

plot_images(img, after, 'before', "after", 'gray', 'gray')
```



However, it needs more threshold. So, direction threshold will be applied to this situation.

```
img = undistort(mpimg.imread(images[3]))
after = combine_img_with_xy_2nd(img)
plot_images(img, after, 'before', "after", 'gray', 'gray')
```



This time, the lane frame looks better.

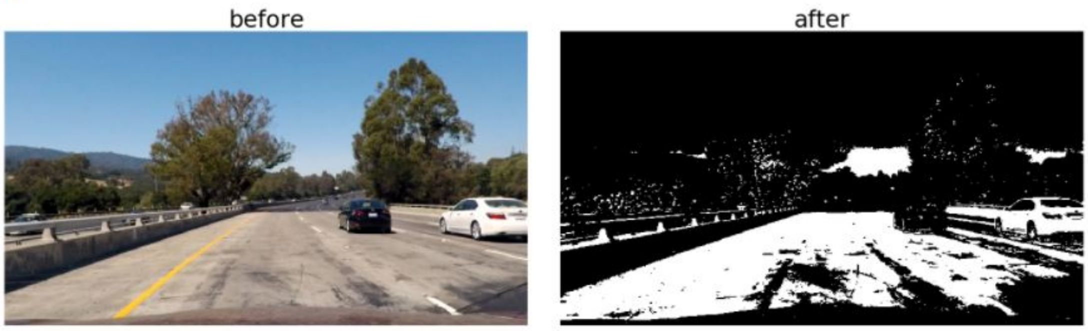
3.2.4 Higher Color Threshold

When the number of pixels are smaller than the lower bound, the reason is the light is too dark. When the number of pixels are higher than the upper bound, the reason is the light is too shinny. When the image is dark, gradient Threshold has a good impact.When the image is light, although gradient threshold also works, but color threshold with another standard can have a better quality.

For example, this is the first glance of the image.

```
img = undistort(mpimg.imread(images[2]))

warped, M, Minv = warp(combine_img_normal(img))
print(test_combine_img_normal(combine_img_normal(img)))
plot_images(img,  combine_img_normal(img), 'before', "after", 'gray', 'gray')
```

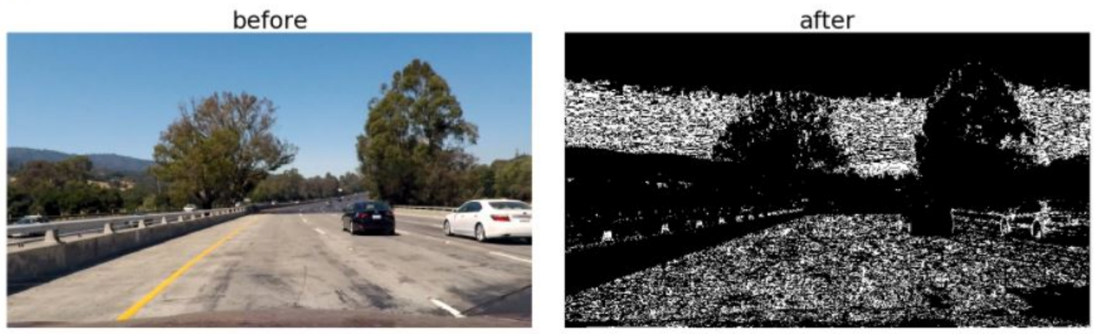
```
102981
```



If switch to gradient threshold

```
img = undistort(mpimg.imread(images[2]))

warped, M, Minv = warp(combine_img_with_xy_2nd(img))
print(test_combine_img_normal(combine_img_with_xy_2nd(img)))
plot_images(img,  combine_img_with_xy_2nd(img), 'before', "after", 'gray', 'gray')
```

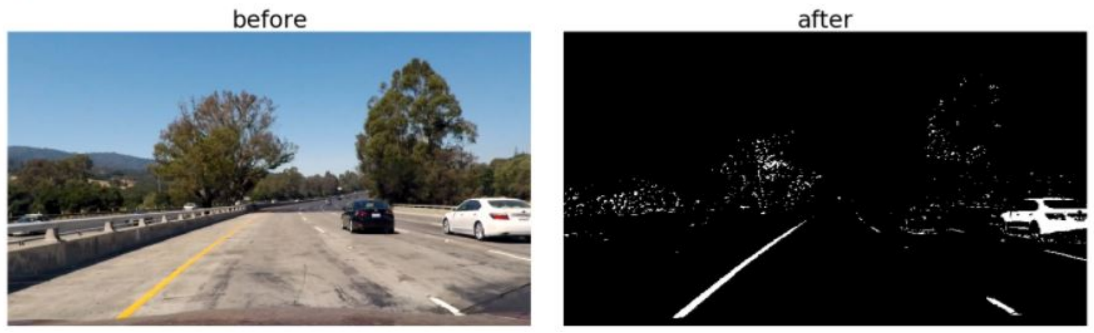
```
41425
```



It will also fail. When the higher color threshold applied,

```
img = undistort(mpimg.imread(images[2]))

warped, M, Minv = warp((img))
print(test_combine_img_normal(reduce_light_hth(img)))
plot_images(img,  reduce_light_hth(img), 'before', "after", 'gray', 'gray')
```

```
2667
```



This time, Light in HLS color domain applied.

3.2.5 Test Flow

To test the lane profile is normal. First, warp the image according to the straight-line profile, count the pixels in between height in 600 to 700 (the image height is 720, pixels in 700 to 720 maybe from the car head, pixels in 0 to 600 is not accurate).

3.3 Perspective Transform

To get the 4 points from the straight-line image, I take the average four points from the straight-line image.

So I implement functions for finding 4 corners.

When the detected lane is not reasonable in the sanity check. I give those images a second chance to check again: only warp the lane that is close to the car head, finding the curvatures and offset, check again. The second check won't display the curvature and the offset because the polyfit uncertainty calculation is related to the range.

First, find the corners that is nearest to the car and furthest to the car. Second, warp the image.

```python
def find_corners():

    straight_line = r_thresh(undistort(mpimg.imread(images[0])), thresh_min=150, thresh_max=255)
    left_upper=[]
    upper = 500
    lower = 690
    for i in range(500, 600, 1):
        if straight_line[upper][i]==1:
            left_upper.append(i)
    c_0 = (np.mean(left_upper), upper)

    right_upper=[]
    for i in range(650, 780,1):
        if straight_line[upper][i]==1:
            right_upper.append(i)
    c_1 = (np.mean(right_upper), upper)

    left_bottom=[]
    for i in range(100, 300,1):
        if straight_line[lower][i]==1:
            left_bottom.append(i)
    c_2 = (np.mean(left_bottom), lower)

    right_bottom=[]
    for i in range(900, 1200,1):
        if straight_line[lower][i]==1:
            right_bottom.append(i)
    c_3 = (np.mean(right_bottom), lower)

    return c_0, c_1, c_2, c_3
```

```python
def warp(img):

    image_height, image_width= img.shape[0:2]
    # Source coordinates
    src = np.float32([find_corners()])

    # Destination coordinates
    dst = np.float32([
            [image_width * 0.25, image_height * 0.75],
            [image_width * 0.75, image_height * 0.75],
            [image_width * 0.25, image_height * 1],
            [image_width * 0.75, image_height * 1],
        ])

    M = cv2.getPerspectiveTransform(src, dst)
    Minv = cv2.getPerspectiveTransform(dst, src)
    warped = cv2.warpPerspective(img, M, (image_width, image_height))

    return warped, M, Minv
```

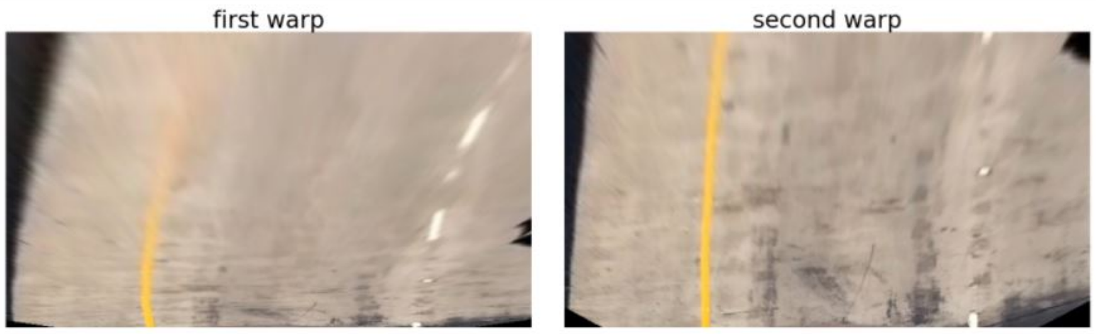
If the sanity check in the next steps failed, find the corners that is not too far away from the car. Second, warp the image.

```python
def find_corners_2nd():

    straight_line = r_thresh(undistort(mpimg.imread(images[1])), thresh_min=150, thresh_max=255)
    left_upper=[]
    upper = 590
    lower = 680
    for i in range(200, 400, 1):
        if straight_line[upper][i]==1:
            left_upper.append(i)
    c_0 = (np.mean(left_upper), upper)

    right_upper=[]
    for i in range(800, 1200,1):
        if straight_line[upper][i]==1:
            right_upper.append(i)
    c_1 = (np.mean(right_upper), upper)

    left_bottom=[]
    for i in range(200, 400,1):
        if straight_line[lower][i]==1:
            left_bottom.append(i)
    c_2 = (np.mean(left_bottom), lower)

    right_bottom=[]
    for i in range(800, 1200,1):
        if straight_line[lower][i]==1:
            right_bottom.append(i)
    c_3 = (np.mean(right_bottom), lower)

    return c_0, c_1, c_2, c_3
```

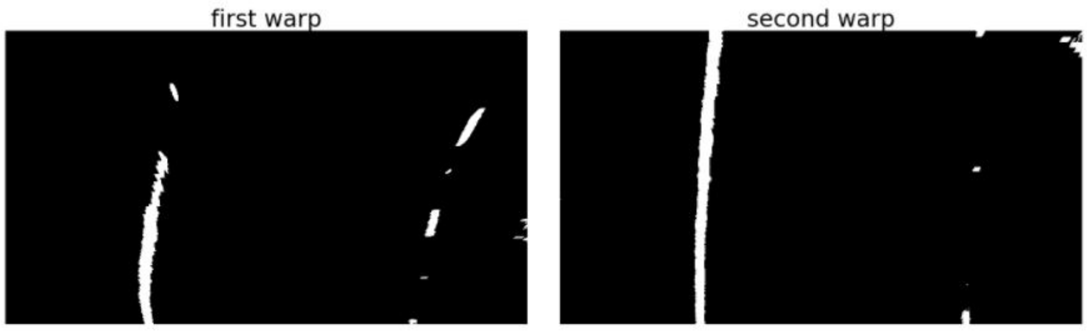
```python
def warp_2nd(img):

    image_height, image_width= img.shape[0:2]
    # Source coordinates
    src = np.float32([find_corners_2nd()])

    # Destination coordinates
    dst = np.float32([
            [image_width * 0.25, image_height * 0.9],
            [image_width * 0.75, image_height * 0.9],
            [image_width * 0.25, image_height * 1],
            [image_width * 0.75, image_height * 1],
        ])

    M = cv2.getPerspectiveTransform(src, dst)
    Minv = cv2.getPerspectiveTransform(dst, src)
    warped = cv2.warpPerspective(img, M, (image_width, image_height))

    return warped, M, Minv
```

Below is the comparison between the 2 warp strategies.

```python
img = undistort(mpimg.imread(images[2]))


warped, M, Minv = warp(img)
warped_2nd, M, Minv = warp_2nd((img))


plot_images(warped,  warped_2nd, 'first warp', "second warp", 'gray', 'gray')
```



```python
img = undistort(mpimg.imread(images[2]))

img = combine_img(img)

warped, M, Minv = warp(img)
warped_2nd, M, Minv = warp_2nd((img))


plot_images(warped,  warped_2nd, 'first warp', "second warp", 'gray', 'gray')
```



3.4 Find Lane Lines

I use the example code from the course note with several modifications:

1. Add a second check rule, if the left line or right line indices is zero, this time, the np sum will begin in a larger range.

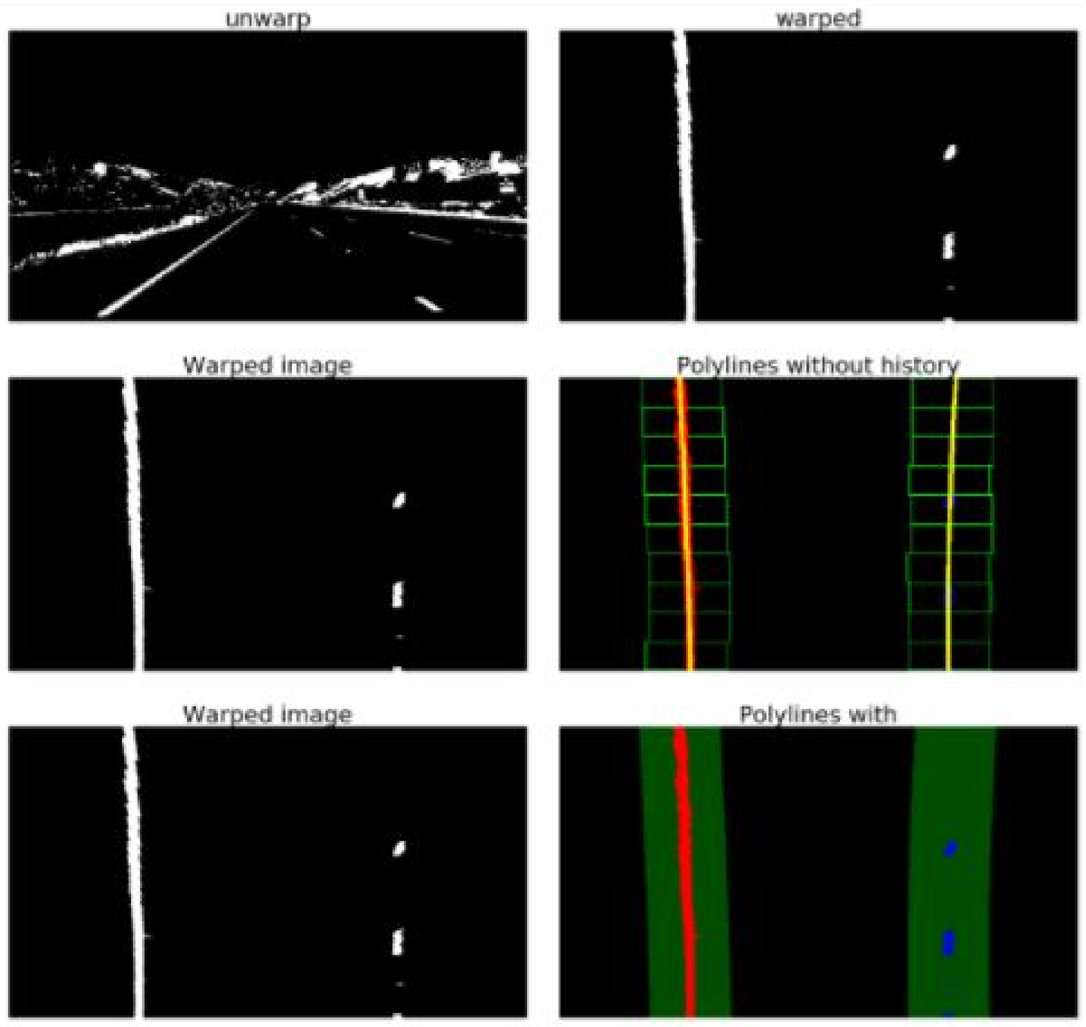
```
    if len(left_lane_inds) == 0 or len(right_lane_inds) == 0:
        # Take a histogram of the bottom half of the image
        histogram = np.sum(binary_warped[binary_warped.shape[0]*3//7:,:], axis=0)
```

2. Add a exception, is the left line or right line still empty, touch a value in it.

```
    try:
        # 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]

        # Fit a second order polynomial to each
        left_fit = np.polyfit(lefty, leftx, 2)
        right_fit = np.polyfit(righty, rightx, 2)
    except:
        left_fit = [0,0,0]
        right_fit = [0,0,0]
```

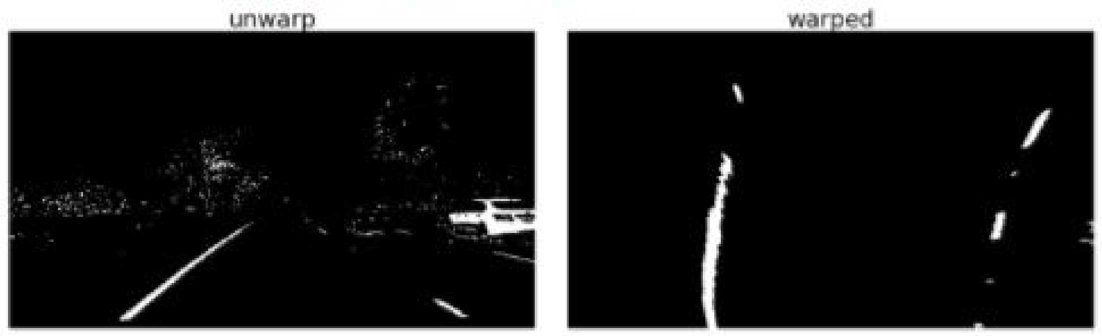
Here is the demo

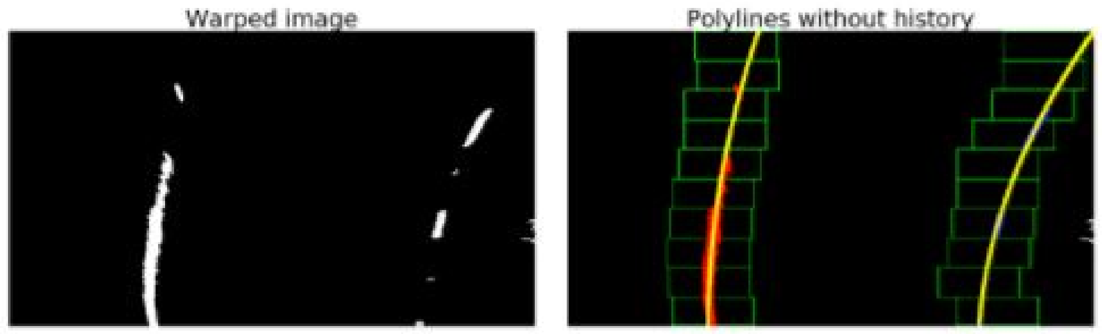
```
img = combine_img(undistort(mpimg.imread(images[0])))
warped, M, Minv = warp(img)
plot_images(img, warped, 'unwarp', "warped", 'gray', 'gray')


fit, out_img = detect_lane_lines_initial(warped)
plot_images(warped, out_img, 'Warped image', "Polylines without history", 'gray', 'jet')

fit, out_img, left_x, left_y, right_x, right_y= detect_lane_lines(warped, fit)
plot_images(warped, out_img, 'Warped image', "Polylines with", 'gray', 'jet')
```

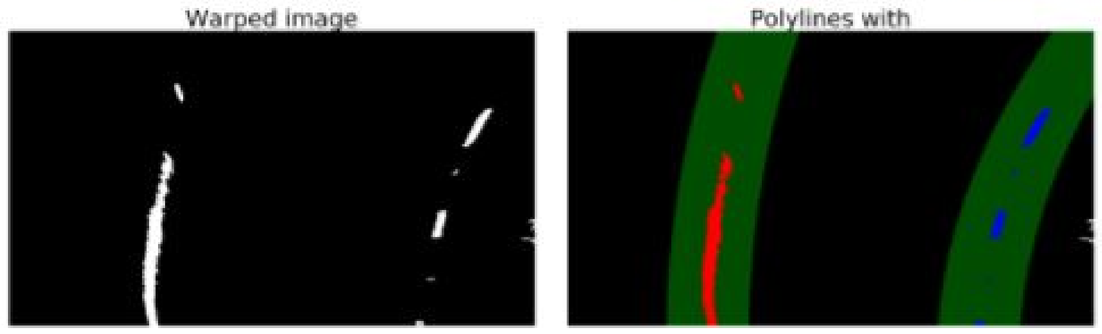


Another demo

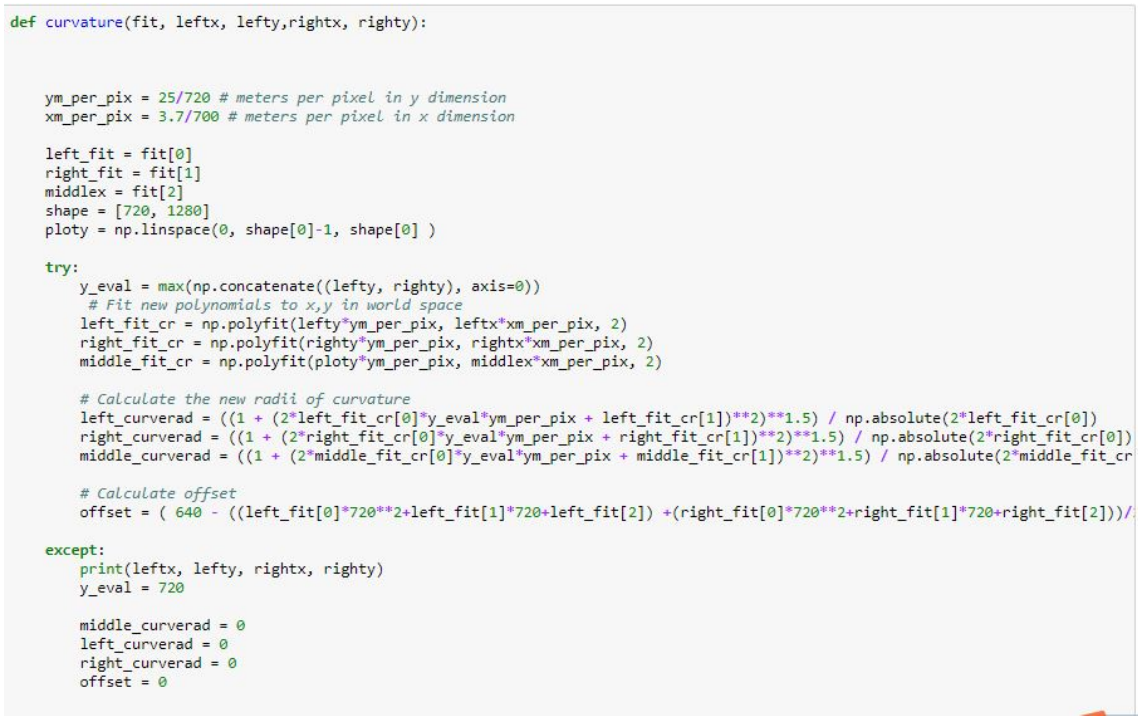
```
img = combine_img(undistort(mpimg.imread(images[2])))
warped, M, Minv = warp(img)
plot_images(img, warped, 'unwarp', "warped", 'gray', 'gray')


fit, out_img = detect_lane_lines_initial(warped)
plot_images(warped, out_img, 'Warped image', "Polylines without history", 'gray', 'jet')

fit, out_img, left_x, left_y, right_x, right_y= detect_lane_lines(warped, fit)
plot_images(warped, out_img, 'Warped image', "Polylines with", 'gray', 'jet')
```







3.5 Measure Curvatures and Offset



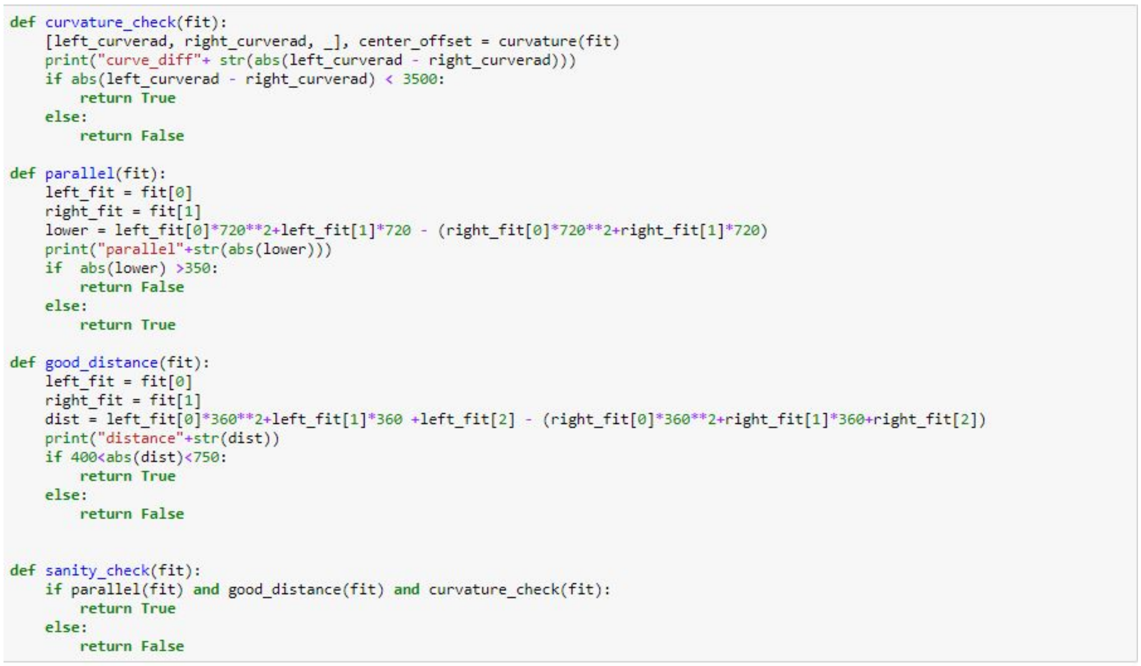
```
def curvature(fit, leftx, lefty,rightx, righty):



    ym_per_pix = 25/720 # meters per pixel in y dimension
    xm_per_pix = 3.7/700 # meters per pixel in x dimension

    left_fit = fit[0]
    right_fit = fit[1]
    middlex = fit[2]
    shape = [720, 1280]
    ploty = np.linspace(0, shape[0]-1, shape[0] )

    try:
        y_eval = max(np.concatenate((lefty, righty), axis=0))
         # Fit new polynomials to x,y in world space
        left_fit_cr = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
        right_fit_cr = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)
        middle_fit_cr = np.polyfit(ploty*ym_per_pix, middlex*xm_per_pix, 2)

        # Calculate the new radii of curvature
        left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
        right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
        middle_curverad = ((1 + (2*middle_fit_cr[0]*y_eval*ym_per_pix + middle_fit_cr[1])**2)**1.5) / np.absolute(2*middle_fit_cr

        # Calculate offset
        offset = ( 640 - ((left_fit[0]*720**2+left_fit[1]*720+left_fit[2]) +(right_fit[0]*720**2+right_fit[1]*720+right_fit[2]))/

    except:
        print(leftx, lefty, rightx, righty)
        y_eval = 720

        middle_curverad = 0
        left_curverad = 0
        right_curverad = 0
        offset = 0
```

The code is from the course note.

I reference the idea from the Internet, if the curvature is higher than 5000, then the curvature is counted to 5000 and displayed as straight line.

The curvature is measured by the mid-line of the two lane lines. The left and right curvatures are used to be sent to sanity check.

3.6 Sanity Check

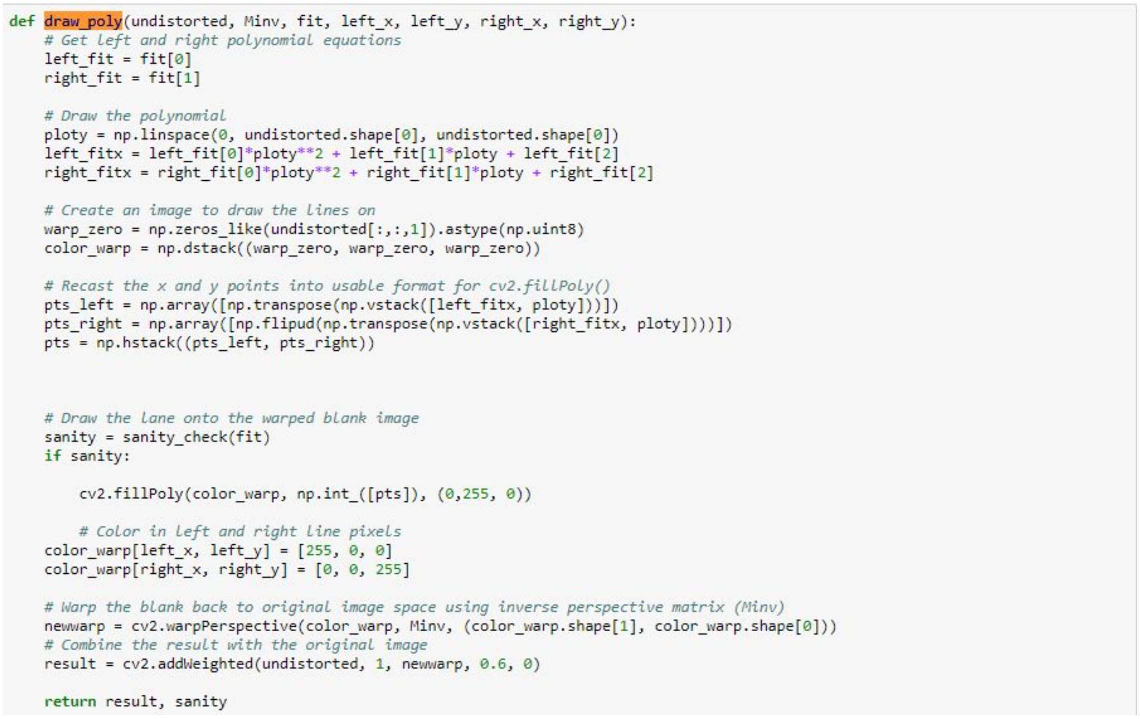


```
def curvature_check(fit):
    [left_curverad, right_curverad, _], center_offset = curvature(fit)
    print("curve_diff"+ str(abs(left_curverad - right_curverad)))
    if abs(left_curverad - right_curverad) < 3500:
        return True
    else:
        return False

def parallel(fit):
    left_fit = fit[0]
    right_fit = fit[1]
    lower = left_fit[0]*720**2+left_fit[1]*720 - (right_fit[0]*720**2+right_fit[1]*720)
    print("parallel"+str(abs(lower)))
    if  abs(lower) >350:
        return False
    else:
        return True

def good_distance(fit):
    left_fit = fit[0]
    right_fit = fit[1]
    dist = left_fit[0]*360**2+left_fit[1]*360 +left_fit[2] - (right_fit[0]*360**2+right_fit[1]*360+right_fit[2])
    print("distance"+str(dist))
    if 400<abs(dist)<750:
        return True
    else:
        return False


def sanity_check(fit):
    if parallel(fit) and good_distance(fit) and curvature_check(fit):
        return True
    else:
        return False
```

The sanity check includes parallel, distance and curvature checks. If the sanity check failed, it will use the second warp strategy and check again.

3.7 Combine Frame

3.7.1 Basic Frame



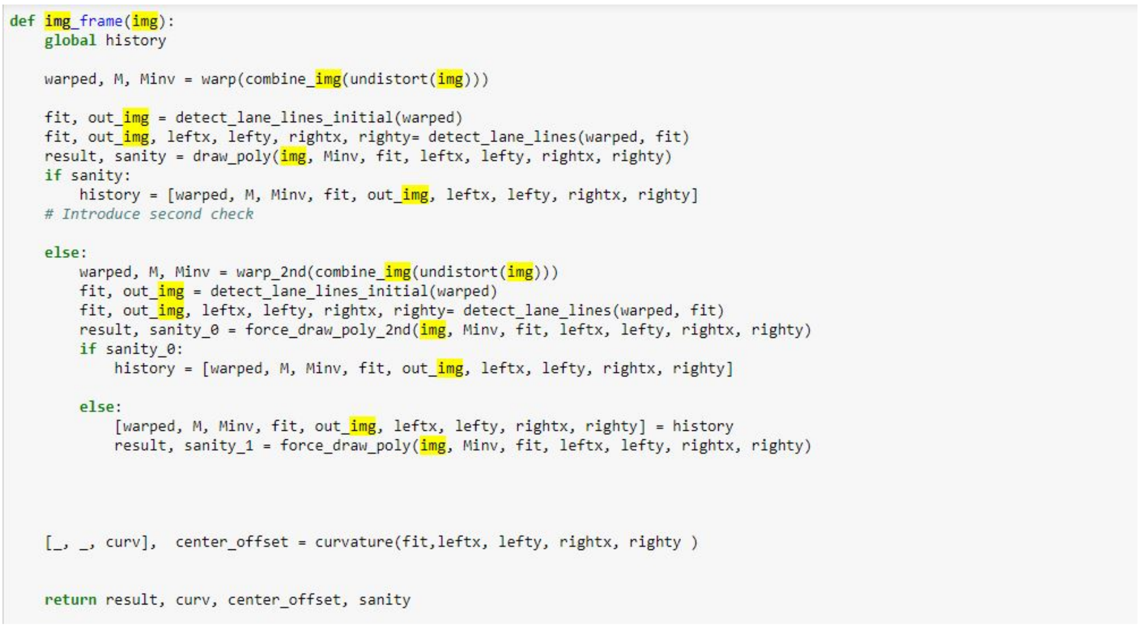
```python
def draw_poly(undistorted, Minv, fit, left_x, left_y, right_x, right_y):
    # Get left and right polynomial equations
    left_fit = fit[0]
    right_fit = fit[1]

    # Draw the polynomial
    ploty = np.linspace(0, undistorted.shape[0], undistorted.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]

    # Create an image to draw the lines on
    warp_zero = np.zeros_like(undistorted[:,:,1]).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

    # 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
    sanity = sanity_check(fit)
    if sanity:

        cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))

        # Color in left and right line pixels
    color_warp[left_x, left_y] = [255, 0, 0]
    color_warp[right_x, right_y] = [0, 0, 255]

    # Warp the blank back to original image space using inverse perspective matrix (Minv)
    newwarp = cv2.warpPerspective(color_warp, Minv, (color_warp.shape[1], color_warp.shape[0]))
    # Combine the result with the original image
    result = cv2.addWeighted(undistorted, 1, newwarp, 0.6, 0)

    return result, sanity
```

In addition to the example:

If sanity pass, then draw two lanes with different color and poly.

If sanity not pass, just draw two lanes.

3.7.2 Frame Pipeline



```python
def img_frame(img):
    global history

    warped, M, Minv = warp(combine_img(undistort(img)))

    fit, out_img = detect_lane_lines_initial(warped)
    fit, out_img, leftx, lefty, rightx, righty= detect_lane_lines(warped, fit)
    result, sanity = draw_poly(img, Minv, fit, leftx, lefty, rightx, righty)
    if sanity:
        history = [warped, M, Minv, fit, out_img, leftx, lefty, rightx, righty]
    # Introduce second check

    else:
        warped, M, Minv = warp_2nd(combine_img(undistort(img)))
        fit, out_img = detect_lane_lines_initial(warped)
        fit, out_img, leftx, lefty, rightx, righty= detect_lane_lines(warped, fit)
        result, sanity_0 = force_draw_poly_2nd(img, Minv, fit, leftx, lefty, rightx, righty)
        if sanity_0:
            history = [warped, M, Minv, fit, out_img, leftx, lefty, rightx, righty]

        else:
            [warped, M, Minv, fit, out_img, leftx, lefty, rightx, righty] = history
            result, sanity_1 = force_draw_poly(img, Minv, fit, leftx, lefty, rightx, righty)



    [_, _, curv],  center_offset = curvature(fit,leftx, lefty, rightx, righty )


    return result, curv, center_offset, sanity
```

If the sanity check not pass, the second warp strategy will work. If the sanity check still not pass, then return the last frame that pass the sanity check.

The first warp sanity check and the second warp sanity check are different. Because the image height of the second warp is smaller. The second warp sanity is much more strict.

3.7.3 Combined Frame

I also build a text frame for displaying curvatures and offset. Only the frames with sanity check in the first time will display the result.

Chapter 4

Discussion

4.1 Program Running Speed

4.1.1 Code Optimization

As mentioned in the previous chapter, I use complicated logic to maintain the correctness of detection. The actual processing speed per frame is low. Many same functions with the same inputs were called many times. This could be optimized by better coding.

4.2 Smoothing

History could be used to smooth the video. For the time issue, I didn't use history method.

4.3 Different hyper-parameters for different cases

In challenge video or harder challenge video, a whole new set of different hyper-parameters could be used for better effect. But I want to build up a general architecture that satisfy every cases.