

Advanced Lane Finding

Write up

1st submit: July 18th, 2nd submit: July 23rd 2021 Kenta Kumazaki

1. Purpose

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road and calculate curvature and position.
- Reflect on my work in a written report.

2. Submission

(1) GitHub

https://github.com/kkumazaki/Self-Drivig-Car_Project2_Advanced-Lane-Finding.git

(2) Directory

<folder: main>

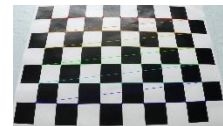
- **Writeup_of_Lesson8.pdf**: This file
- **P2.jpynb**: Pipeline
- Image files video files are saved as following:

<folder: output_images>

<folder: 1_Calibration >

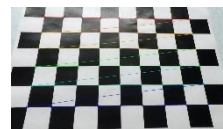
<folder: before_camera_cal>

Chessboard image before camera calibration.



<folder: after_camera_cal>

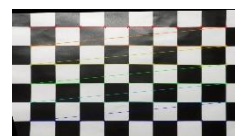
Chessboard image after camera calibration.



<folder: 2_Undistortion>

<folder: Chessboard>

Undistort and perspective transform of chessboard.



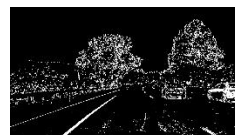
<folder: Camera>

Undistort camera images.



<folder: 3_Thresholded>

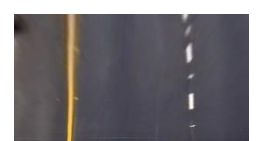
Color and Gradient threshold images.



<folder: 4_Transform>

<folder: Create>

Straight line images to get transform parameters



<folder: Adapt>

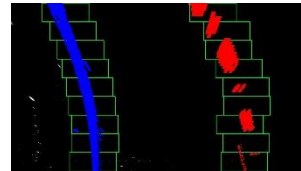
Adapt parameters and transform curve lines.



<folder: 5_Finding>

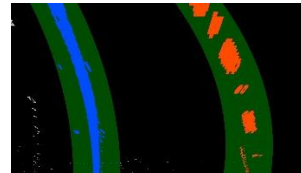
<Histogram>

Calculate by histogram.



<Prior>

Calculate by prior line info.



<folder: 6_Result>

Show the lane line area, curvature and position onto test images.



<folder: test videos output>

<folder: before_countermeasure>

Output of my pipeline on July 18th. (1st submit)

<folder: after_countermeasure>

Output of my pipeline on July 23rd. (2nd submit)



3. Reflection

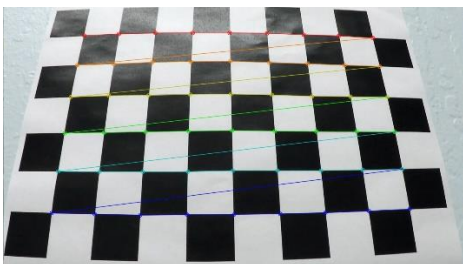
(1)Description of my pipeline

My pipeline consisted of 8 steps.

First, I calibrated the camera by using 20 chessboard images.

```
37 #1. Camera Calibration
38
39 #(1)prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....., (6,5,0)
40 objp = np.zeros((8*9,3), np.float32)
41 objp[:, :2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
42
43 #(2)Arrays to store object points and image points from all the images.
44 objpoints = [] # 3d points in real world space
45 imgpoints = [] # 2d points in image plane.
46
47 #(3)Make a list of calibration images
48 images = glob.glob('camera_cal/calibration*.jpg')
49
50 #(4)Step through the list and search for chessboard corners
51 for fname in images:
52     img = mpimg.imread(fname)
53     gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
54
55     # Find the chessboard corners
56     ret, corners = cv2.findChessboardCorners(gray, (9,6),None)
57
58     # If found, add object points, image points
59     if ret == True:
60         objpoints.append(objp)
61         imgpoints.append(corners)
62
63     # Draw the Chessboard Corners
64     img = cv2.drawChessboardCorners(img, (9,6), corners, ret)
65     #plt.imshow(img)
66
67     # Get the parameters of Camera Calibration
68     ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
69
70     # Undistorting test images
71     undist = cv2.undistort(img, mtx, dist, None, mtx)
72
73     logger1.debug(fname)
74     logger1.debug(mtx)
75     logger1.debug(dist)
76
77
78     # Plot original image and undistorted image
79     f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))
80     f.tight_layout()
81     ax1.imshow(img)
82     ax1.set_title('Original Image'+fname, fontsize=20)
83     ax2.imshow(undist)
84     ax2.set_title('Undistorted Image'+fname, fontsize=20)
85     plt.subplots_adjust(left=0., right=1, top=0.9, bottom=0.)
86
87     cv2.imwrite('before_'+fname+'.jpg', img)
88     cv2.imwrite('after_'+fname+'.jpg', undist)
```

I show one example of distortion correction below. Data folder is “output_images¥1_Calibration”.



Original image

before_camera_cal /calibration3.jpg

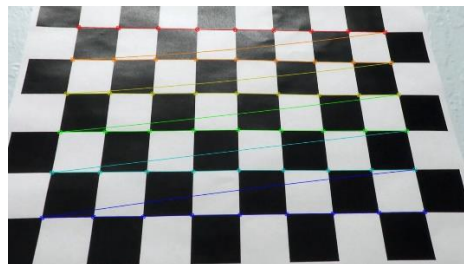
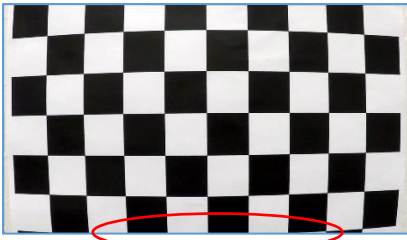


Image after calibration

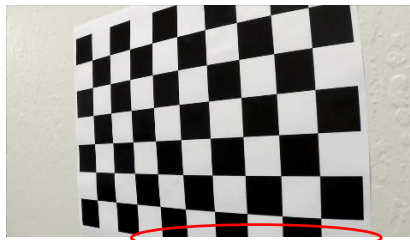
after_camera_cal/ calibration3.jpg

I couldn't use some images for calibration because there are **not enough cross points** as following.



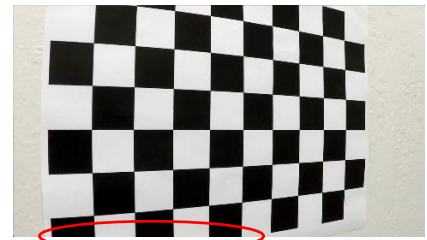
Original image (not for calibration)

camera_cal/ calibration1.jpg



Original image (not for calibration)

camera_cal/ calibration4.jpg



Original image (not for calibration)

camera_cal/ calibration5.jpg

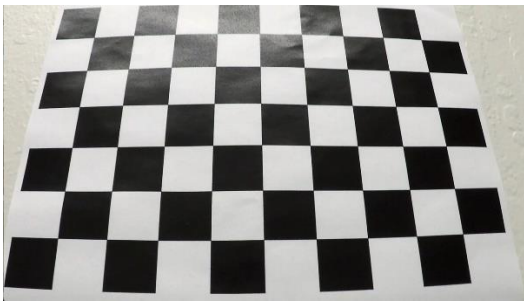
Second, I applied distortion correction and perspective transform of Chessboard.

```
1  #2. Distortion Correction
2  #Apply a distortion correction to raw images.
3  #refer Lesson5, section 18
4
5  #1) Distortion Correction and Perspective Transform of Chessboard
6  #Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
7  # Read Chessboard in an image
8  img = cv2.imread('camera_cal/calibration3.jpg')
9  nx = 9 # the number of inside corners in x
10 ny = 6 # the number of inside corners in y
11
12 # Calculate "M" for Perspective Transform
13 def corners_unwarp(img, nx, ny, mtx, dist):
14     img_size = (img.shape[1], img.shape[0])
15
16     # 1) Undistort using mtx and dist, which were calculated before.
17     # Calibration is not needed, because mtx and dist are already exist.
18     undist = cv2.undistort(img, mtx, dist, None, mtx)
19     img = undist
20
21     # 2) Convert to grayscale
22     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
23
24     # 3) Find the chessboard corners
25     ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)
26
27     # 4) If corners found:
28     if ret == True:
29
30         # a) draw corners
31         img = cv2.drawChessboardCorners(img, (nx, ny), corners, ret)
32
33         # b) define 4 source points src = np.float32([[x1, y1], [x2, y2], [x3, y3], [x4, y4]])
34         src = np.float32([corners[0], corners[nx-1], corners[-1], corners[-nx]]) # Model Answer
35
36         # c) define 4 destination points dst = np.float32([[x1, y1], [x2, y2], [x3, y3], [x4, y4]])
37         offset = 100 # Model Answer
38         dst = np.float32([[offset, offset], [img_size[0]-offset, offset],
39                           [img_size[0]-offset, img_size[1]-offset],
40                           [offset, img_size[1]-offset]])
41
42         # d) use cv2.getPerspectiveTransform() to get M, the transform matrix
43         M = cv2.getPerspectiveTransform(src, dst)
44
45         # e) use cv2.warpPerspective() to warp my image to a top-down view
46         warped = cv2.warpPerspective(img, M, img_size, flags=cv2.INTER_LINEAR)
47
48     return warped, M
49
50 # Show the result of Perspective Transform with Chessboard
51 top_down, perspective_M = corners_unwarp(img, nx, ny, mtx, dist)
52 f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))
53 f.tight_layout()
54 ax1.imshow(img)
55 ax1.set_title('Original Image of Chessboard', fontsize=30)
56 ax2.imshow(top_down)
57 ax2.set_title('Undistorted and Warped Image of Chessboard', fontsize=30)
58 plt.subplots_adjust(left=0., right=1, top=0.9, bottom=0.)
59
```

The result of distortion correction and perspective transform of Chessboard is following.

It looks distortion correction and perspective transform works well.

Data folder is "output_images¥2_Undistortion".



Original image

Chessboard/ Original_chess.jpg

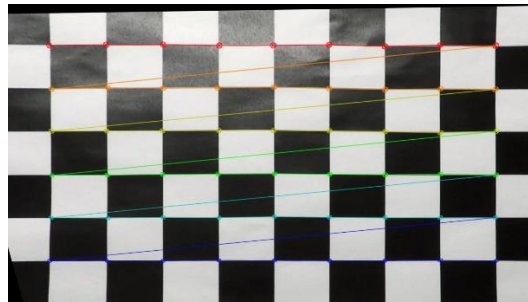


Image after distortion correction and perspective transform

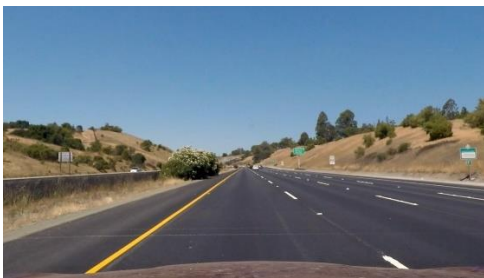
Chessboard/ Warped_chess.jpg

And also, I adapted distortion correction to the test images.

```
60 #(2)Apply a distortion correction to raw images
61 # Using List doesn't work well, so read each image each time.
62 straight_lines1 = mpimg.imread('test_images/straight_lines1.jpg')
63 straight_lines2 = mpimg.imread('test_images/straight_lines2.jpg')
64 test1 = mpimg.imread('test_images/test1.jpg')
65 test2 = mpimg.imread('test_images/test2.jpg')
66 test3 = mpimg.imread('test_images/test3.jpg')
67 test4 = mpimg.imread('test_images/test4.jpg')
68 test5 = mpimg.imread('test_images/test5.jpg')
69 test6 = mpimg.imread('test_images/test6.jpg')
70
71 # Undistort each image
72 undist_straight_lines1 = cv2.undistort(straight_lines1, mtx, dist, None, mtx)
73 undist_straight_lines2 = cv2.undistort(straight_lines2, mtx, dist, None, mtx)
74 undist_test1 = cv2.undistort(test1, mtx, dist, None, mtx)
75 undist_test2 = cv2.undistort(test2, mtx, dist, None, mtx)
76 undist_test3 = cv2.undistort(test3, mtx, dist, None, mtx)
77 undist_test4 = cv2.undistort(test4, mtx, dist, None, mtx)
78 undist_test5 = cv2.undistort(test5, mtx, dist, None, mtx)
79 undist_test6 = cv2.undistort(test6, mtx, dist, None, mtx)
80
```

I checked images after distortion correction and they looked as good as the examples of Lesson.

Data folder is "output_images¥2_Undistortion¥Camera".



Distortion Correction image (straight)

undist_straight_lines1.jpg



Distortion Correction image (curve)

undist_test3.jpg

Third, I applied Color Threshold and Gradient Threshold to detect the edges of lane lines.

I used HLS color space and set threshold on S channel because it has better result than other channels in Lesson.

```
1  #3. Color and Gradient Threshold
2  #Use color transforms, gradients, etc., to create a thresholded binary image.
3
4  #Define a function that thresholds the S-channel of HLS and Sobel gradient
5  def pipeline(img, s_thresh=(170, 255), sx_thresh=(20, 100)):
6      img = np.copy(img)
7      # Convert to HLS color space and separate the V channel
8      hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
9      l_channel = hls[:, :, 1]
10     s_channel = hls[:, :, 2]
11     # Sobel x
12     sobelx = cv2.Sobel(l_channel, cv2.CV_64F, 1, 0) # Take the derivative in x
13     abs_sobelx = np.absolute(sobelx) # Absolute x derivative to accentuate lines away from horizontal
14     scaled_sobel = np.uint8(255*abs_sobelx/np.max(abs_sobelx))
15
16     # Threshold x gradient
17     sxbinary = np.zeros_like(scaled_sobel)
18     sxbinary[(scaled_sobel >= sx_thresh[0]) & (scaled_sobel <= sx_thresh[1])] = 1
19
20     # Threshold color channel
21     s_binary = np.zeros_like(s_channel)
22     s_binary[(s_channel >= s_thresh[0]) & (s_channel <= s_thresh[1])] = 1
23     # Stack each channel
24     #color_binary = np.dstack(( np.zeros_like(sxbinary), sxbinary, s_binary)) * 255
25
26     color_binary = np.dstack((sxbinary|s_binary, sxbinary|s_binary, sxbinary|s_binary)) * 255 #Try showing white color
27     return color_binary
28
29 # Tune the threshold to try to match the images
30 hls_straight_lines1 = pipeline(undist_straight_lines1)
31 hls_straight_lines2 = pipeline(undist_straight_lines2)
32 hls_test1 = pipeline(undist_test1)
33 hls_test2 = pipeline(undist_test2)
34 hls_test3 = pipeline(undist_test3)
35 hls_test4 = pipeline(undist_test4)
36 hls_test5 = pipeline(undist_test5)
37 hls_test6 = pipeline(undist_test6)
38
```

I checked images after threshold and they looked as good as the examples of Lesson.

Data folder is "output_images¥3_Thresholded".



Image after threshold
thresholded_test1.jpg



Image after threshold
thresholded_test2.jpg



Image after threshold
thresholded_test3.jpg



Image after threshold
thresholded_test4.jpg



Image after threshold
thresholded_test5.jpg

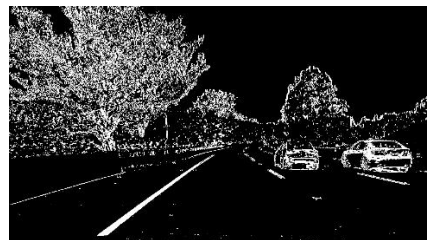


Image after threshold
thresholded_test6.jpg

Forth, I created bird view of the previous threshold mages.

```
1  #4. Perspective Transform to bird view
2  #Apply a perspective transform to rectify binary image ("birds-eye view").
3  # Should try with 2 straight line images
4
5  #(1) Create warp function which fits well with straight lines images.
6  # Save RGB color images to get good src points (if we use cv2.imwrite(), should be converted)
7  save_img1 = cv2.cvtColor(undist_straight_lines1, cv2.COLOR_RGB2BGR)
8  save_img2 = cv2.cvtColor(undist_straight_lines2, cv2.COLOR_RGB2BGR)
9  cv2.imwrite("undist_straight_lines1.jpg", save_img1)
10 cv2.imwrite("undist_straight_lines2.jpg", save_img2)
11
12 #Warp function
13 def warp(img):
14
15     # Define calibration box in source and destination
16     img_size = (img.shape[1], img.shape[0])
17
18     # Get src from undistorted images
19     src1 = np.float32([[708, 462], [1053, 688], [246, 688], [577, 462]]) # undist_straight_lines1
20     #src1 = np.float32([[662, 436], [1053, 688], [246, 688], [617, 436]]) # undist_straight_lines1
21     src2 = np.float32([[678, 441], [1040, 675], [280, 675], [607, 441]]) # undist_straight_lines2
22     src = (src1 + src2)/2 # Use average to get more robust src
23     offsetx = 300
24     offsety = 0
25     dst = np.float32([[img_size[0]-offsetx, offsety], [img_size[0]-offsetx, img_size[1]-offsety], [offsetx, img_size[1]], [offsetx, offsety]])
26
27
28
29
30     # Compute the perspective transform, M
31     M = cv2.getPerspectiveTransform(src, dst)
32
33     # Create warped image
34     warped = cv2.warpPerspective(img, M, img_size, flags = cv2.INTER_LINEAR)
35
36     return warped
37
38 # Generate warped images from straight lines images
39 warped_straight_lines1 = warp(undist_straight_lines1)
40 warped_straight_lines2 = warp(undist_straight_lines2)
```

To determine src parameters of Perspective Transform, I used 2 straight line images as below.

I get 2 sets of src (src1, src2) and determine the final src by calculating average of src1 and sec2.

Data folder is "output_images¥4_Transform¥Create".



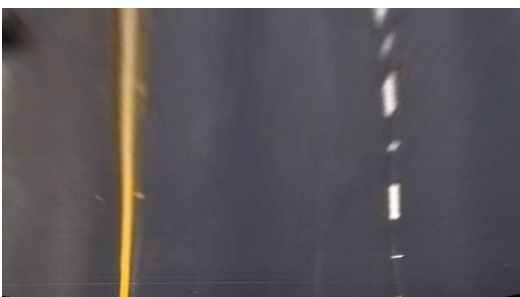
src1 points

undist_straight_lines1_get_src.jpg



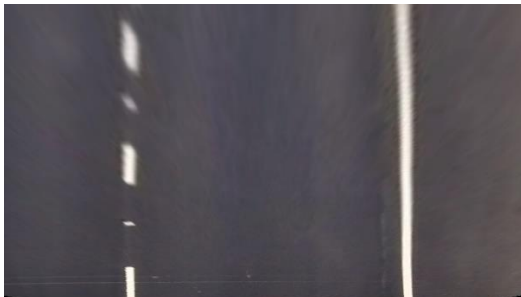
src2 points

undist_straight_lines2_get_src.jpg



Perspective Transform image

warped_straight_lines1.jpg



Perspective Transform image

warped_straight_lines2.jpg

By using those parameters, I calculated Perspective Transform of the Images after threshold and showed the bird view images as below.



Image after Transform

warped_test1.jpg



Image after Transform

warped_test2.jpg



Image after Transform

warped_test3.jpg

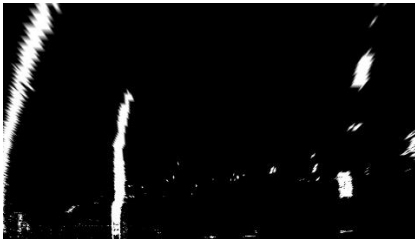


Image after Transform

warped_test4.jpg

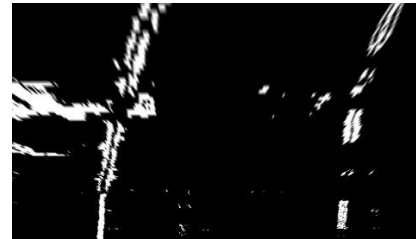


Image after Transform

warped_test5.jpg

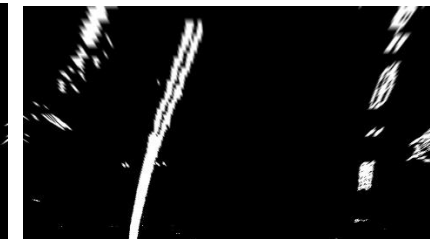


Image after Transform

warped_test6.jpg

Fifth, I found the lane lines in the bird view. There are 4 functions, so I explain step by step.

(1)find_lane_pixels(): Find pixels by using histogram.

```
1  #5. Finding the Lanes
2  #Detect lane pixels and fit to find the lane boundary.
3
4  binary_warped = warped_test6
5  file_name = "test6"
6  original_img = test6
7
8  Mode = 1 # Until Step 7: Plot Lane Lines area with green color
9  #Mode = 2 # Step 8: Plot Lane area with green color for movie file
10
11 # (1) Peaks in Histogram by Sliding Window
12 # Find pixels by using histogram
13 def find_lane_pixels(binary_warped):
14     # Take a histogram of the bottom half of the image
15     histogram = np.sum(binary_warped[binary_warped.shape[0]//2:,:], axis=0)
16     # Create an output image to draw on and visualize the result
17     out_img = np.dstack((binary_warped, binary_warped, binary_warped))
18     # Find the peak of the left and right halves of the histogram
19     # These will be the starting point for the left and right lines
20     #midpoint = np.int(histogram.shape[0]//2) #There's warning if I use np.int...
21     midpoint = int(histogram.shape[0]//2)
22     leftx_base = np.argmax(histogram[:midpoint])
23     rightx_base = np.argmax(histogram[midpoint:]) + midpoint
24
25 # HYPERPARAMETERS
26 # Choose the number of sliding windows
27 nwindows = 9
28 # Set the width of the windows +/- margin
29 margin = 100
30 # Set minimum number of pixels found to recenter window
31 #minpix = 50 # If it's 50, it's not good at test4.jpg...
32 minpix = 200
33
34 # Set height of windows - based on nwindows above and image shape
35 #window_height = np.int(binary_warped.shape[0]//nwindows) #There's warning if I use np.int...
36 window_height = int(binary_warped.shape[0]//nwindows)
37 # Identify the x and y positions of all nonzero pixels in the image
38 nonzero = binary_warped.nonzero()
39 nonzeroy = np.array(nonzero[0])
40 nonzerox = np.array(nonzero[1])
41 # Current positions to be updated later for each window in nwindows
42 leftx_current = leftx_base
43 rightx_current = rightx_base
```



```

45 # Create empty lists to receive left and right lane pixel indices
46 left_lane_inds = []
47 right_lane_inds = []
48
49 # Step through the windows one by one
50 for window in range(nwindows):
51     # Identify window boundaries in x and y (and right and left)
52     win_y_low = binary_warped.shape[0] - (window+1)*window_height
53     win_y_high = binary_warped.shape[0] - window*window_height
54     ### TO-DO: Find the four below boundaries of the window ###
55     win_xleft_low = leftx_current-margin # Update this
56     win_xleft_high = leftx_current+margin # Update this
57     win_xright_low = rightx_current-margin # Update this
58     win_xright_high = rightx_current+margin # Update this
59
60     # Draw the windows on the visualization image
61     cv2.rectangle(out_img,(win_xleft_low,win_y_low),
62                  (win_xleft_high,win_y_high),(0,255,0), 2)
63     cv2.rectangle(out_img,(win_xright_low,win_y_low),
64                  (win_xright_high,win_y_high),(0,255,0), 2)
65
66     ### TO-DO: Identify the nonzero pixels in x and y within the window ###
67     good_left_inds = ((nonzero >= win_y_low) & (nonzero < win_y_high) &
68                      (nonzero >= win_xleft_low) & (nonzero < win_xleft_high)).nonzero()[0]
69     good_right_inds = ((nonzero >= win_y_low) & (nonzero < win_y_high) &
70                      (nonzero >= win_xright_low) & (nonzero < win_xright_high)).nonzero()[0]
71
72     # Append these indices to the lists
73     left_lane_inds.append(good_left_inds)
74     right_lane_inds.append(good_right_inds)
75
76     ### TO-DO: If you found > minpix pixels, recenter next window ###
77     ### ('right' or 'leftx_current') on their mean position ###
78     #pass # Remove this when you add your function
79     if len(good_left_inds) > minpix:
80         #leftx_current = np.int(np.mean(nonzero[good_left_inds]))#There's warning if I use np.int...
81         leftx_current = int(np.mean(nonzero[good_left_inds]))
82     if len(good_right_inds) > minpix:
83         rightx_current = int(np.mean(nonzero[good_right_inds]))#There's warning if I use np.int...
84         #rightx_current = np.int(np.mean(nonzero[good_right_inds]))
85
86 # Concatenate the arrays of indices (previously was a list of lists of pixels)
87 try:
88     left_lane_inds = np.concatenate(left_lane_inds)
89     right_lane_inds = np.concatenate(right_lane_inds)
90 except ValueError:
91     # Avoids an error if the above is not implemented fully
92     pass

```

(2) fit_polynomial(): Calculate fitting polynomial by using find_lane_pixels().

```

102 #Calculate fitting polynomial
103 def fit_polynomial(binary_warped):
104     # Find our lane pixels first
105     leftx, lefty, rightx, righty, out_img = find_lane_pixels(binary_warped)
106
107     ### TO-DO: Fit a second order polynomial to each using 'np.polyfit' ###
108     left_fit = np.polyfit(lefty, leftx, 2)
109     right_fit = np.polyfit(righty, rightx, 2)
110
111     # Generate x and y values for plotting
112     ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )
113     try:
114         left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
115         right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
116     except TypeError:
117         # Avoids an error if 'left' and 'right_fit' are still none or incorrect
118         print('The function failed to fit a line!')
119         left_fitx = 1*ploty**2 + 1*ploty
120         right_fitx = 1*ploty**2 + 1*ploty
121
122     ## Visualization ##
123     # Colors in the left and right lane regions
124     out_img[lefty, leftx] = [255, 0, 0]
125     out_img[righty, rightx] = [0, 0, 255]

```

```

127 # Plots the left and right polynomials on the lane lines
128 #plt.plot(left_fitx, ploty, color='yellow')
129 #plt.plot(right_fitx, ploty, color='yellow')
130
131 #return out_img
132 return out_img, left_fit, right_fit # Add left_fit, right_fit for the next step
133
134 #Get result of fitting polynomial
135 out_img, left_fit, right_fit = fit_polynomial(binary_warped)

```

The result of histogram is shown below.

Data folder is “¥ output_images¥5_Finding¥Histogram”.

I used the same parameters as Lesson at first, but the parameter: **min_pix = 50** didn't work with “test4” image. It calculated the average of the bottom 4th area (A), then bottom 5th area moved to left and the right lane was not detected (B).

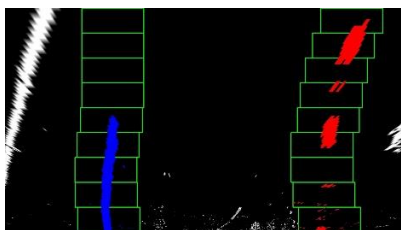


Image after Histogram

Find_from_Histogram_test1_
minpix_50.jpg

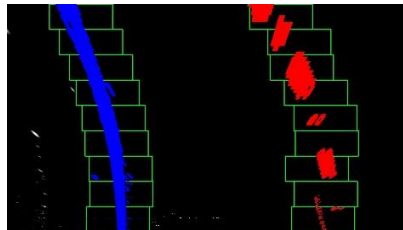


Image after Histogram

Find_from_Histogram_test2_
minpix_50.jpg

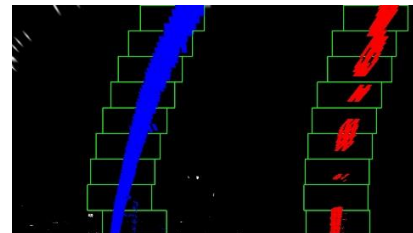


Image after Histogram

Find_from_Histogram_test3_
minpix_50.jpg

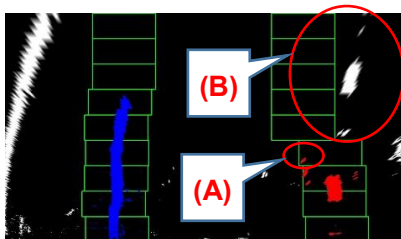


Image after Histogram

Find_from_Histogram_test4_
minpix_50_bad.jpg

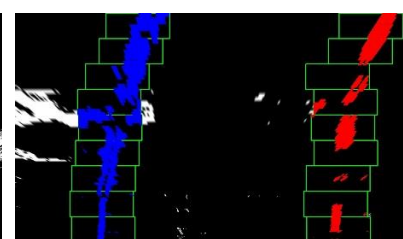


Image after Histogram

Find_from_Histogram_test5_
minpix_50.jpg

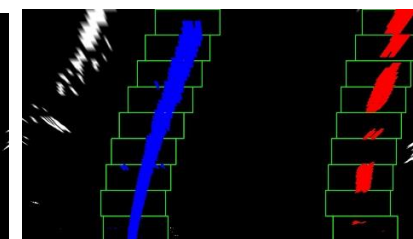


Image after Histogram

Find_from_Histogram_test6_
minpix_50.jpg

To avoid it, I changed parameter: **minpix = 200**, then the problem was solved (C).

I checked other images and there are no degradation.

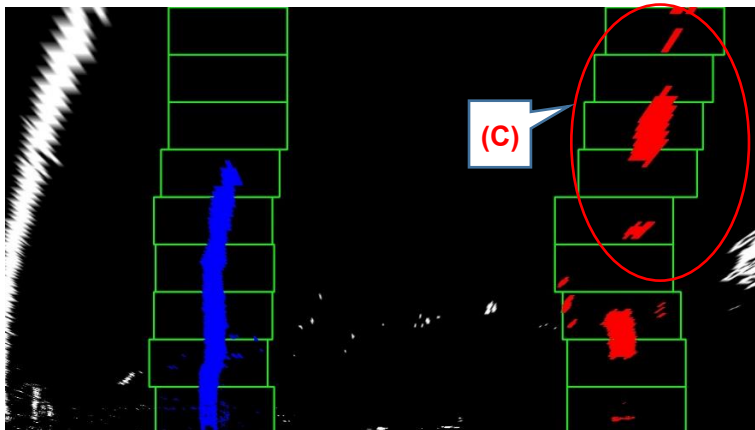


Image after Histogram with parameter tuning

Find_from_Histogram_test4_minpix_200.jpg

(3) fit_poly(): Calculate polynomial.

```
139 ##(2)Search lines from Polynomial fit values from the previous frame
140 #def fit_poly(img_shape, leftx, lefty, rightx, righty):
141 def fit_poly(img_shape, leftx, lefty, rightx, righty):
142     ### TO-DO: Fit a second order polynomial to each with np.polyfit() ###
143     left_fit = np.polyfit(lefty, leftx, 2)
144     right_fit = np.polyfit(righty, rightx, 2)
145     # Generate x and y values for plotting
146     ploty = np.linspace(0, img_shape[0]-1, img_shape[0])
147     ### TO-DO: Calc both polynomials using ploty, left_fit and right_fit ###
148     left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
149     right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
150
151     return left_fitx, right_fitx, ploty
```

(4) search_around_poly(): Search lines from Polynomial fit values from the previous frame.

```
153 def search_around_poly(binary_warped, xm_per_pix, ym_per_pix):
154     # HYPERPARAMETER
155     # Choose the width of the margin around the previous polynomial to search
156     margin = 100
157
158     # Grab activated pixels
159     nonzero = binary_warped.nonzero()
160     nonzeroy = np.array(nonzero[0])
161     nonzerox = np.array(nonzero[1])
162
163     ### TO-DO: Set the area of search based on activated x-values ###
164     ### within the +/- margin of our polynomial function ###
165     ### Hint: consider the window areas for the similarly named variables ###
166     left_lane_inds = ((nonzerox > (left_fit[0]*(nonzeroy**2) + left_fit[1]*nonzeroy +
167         left_fit[2] - margin)) & (nonzerox < (left_fit[0]*(nonzeroy**2) +
168         left_fit[1]*nonzeroy + left_fit[2] + margin)))
169     right_lane_inds = ((nonzerox > (right_fit[0]*(nonzeroy**2) + right_fit[1]*nonzeroy +
170         right_fit[2] - margin)) & (nonzerox < (right_fit[0]*(nonzeroy**2) +
171         right_fit[1]*nonzeroy + right_fit[2] + margin)))
172
173     # Again, extract left and right line pixel positions
174     leftx = nonzerox[left_lane_inds]
175     lefty = nonzeroy[left_lane_inds]
176     rightx = nonzerox[right_lane_inds]
177     righty = nonzeroy[right_lane_inds]
178
179     # Fit new polynomials
180     left_fitx, right_fitx, ploty = fit_poly(binary_warped.shape, leftx, lefty, rightx, righty)
181
182     #Calculate left_fit, right_fit for Curvature measuring
183     left_fit_pix = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
184     right_fit_pix = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)
185
186     left_fitx_pix = left_fitx*xm_per_pix
187     right_fitx_pix = right_fitx*xm_per_pix
188
189     ## Visualization ##
190     # Create an image to draw on and an image to show the selection window
191     out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255
192     window_img = np.zeros_like(out_img)
193     # Color in left and right line pixels
194     out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
195     out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255]
196
197     # Generate a polygon to illustrate the search window area
198     # And recast the x and y points into usable format for cv2.fillPoly()
199     left_line_window1 = np.array([np.transpose(np.vstack([left_fitx-margin, ploty]))])
200     left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx+margin,
201         ploty]))))]
202     left_line_pts = np.hstack((left_line_window1, left_line_window2))
203     right_line_window1 = np.array([np.transpose(np.vstack([right_fitx-margin, ploty]))])
204     right_line_window2 = np.array([np.flipud(np.transpose(np.vstack([right_fitx+margin,
205         ploty]))))]
206     right_line_pts = np.hstack((right_line_window1, right_line_window2))
```

```

208 if Mode == 1:
209     # Draw the lane onto the warped blank image
210     cv2.fillPoly(window_img, np.int_([left_line_pts]), (0,255, 0))
211     cv2.fillPoly(window_img, np.int_([right_line_pts]), (0,255, 0))
212     result = cv2.addWeighted(out_img, 1, window_img, 0.3, 0)
213
214     # Plot the polynomial lines onto the image
215     #plt.plot(left_fitx, ploty, color='yellow')
216     #plt.plot(right_fitx, ploty, color='yellow')
217     ## End visualization steps ##
218
219 elif Mode == 2:
220     # Draw the lane onto the warped blank image
221     final_line_window1 = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
222     final_line_window2 = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
223     final_line_pts = np.hstack((final_line_window1, final_line_window2))
224     cv2.fillPoly(window_img, np.int_([final_line_pts]), (0,255, 0))
225     result = cv2.addWeighted(out_img, 1, window_img, 0.3, 0)
226
227     #return result
228     return result, left_fit_pix, right_fit_pix, left_fitx_pix, right_fitx_pix # Add values to calculate curvature

```

In fifth step, finally get the result of lane finding in bird view.

```

230 # Define conversions in x and y from pixels space to meters
231 ym_per_pix = 30/720 # meters per pixel in y dimension #Define outside
232 xm_per_pix = 3.7/700 # meters per pixel in x dimension #Define outside
233
234 # Run image through the pipeline
235 # Note that in your project, you'll also want to feed in the previous fits
236 #result = search_around_poly(binary_warped)
237 result, left_fit_pix, right_fit_pix, left_fitx_pix, right_fitx_pix = search_around_poly(binary_warped, xm_per_pix, ym_per_pix)
238 cv2.imwrite('output_images/Search_from_Prior_'+file_name+'.jpg',result)

```

Data folder is “¥output_images¥5_Finding¥Prior”.

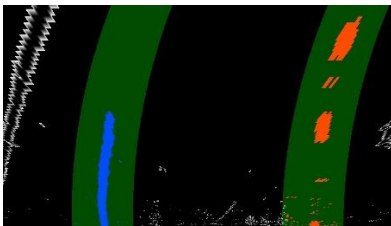


Image after Polynomial fit

Search_from_Prior_test1_
minpix_200.jpg

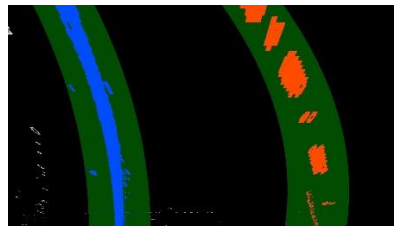


Image after Polynomial fit

Search_from_Prior_test2_
minpix_200.jpg

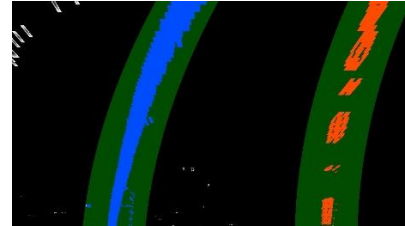


Image after Polynomial fit

Search_from_Prior_test3_
minpix_200.jpg

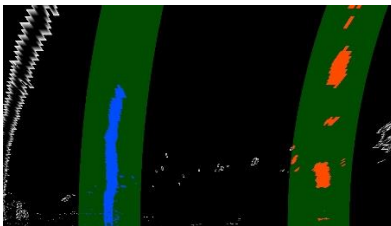


Image after Polynomial fit

Search_from_Prior_test4_
minpix_200.jpg

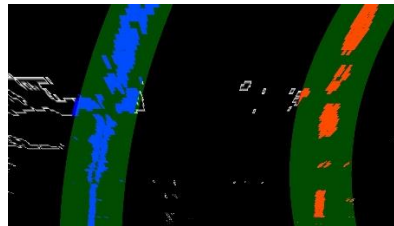


Image after Polynomial fit

Search_from_Prior_test5_
minpix_200.jpg

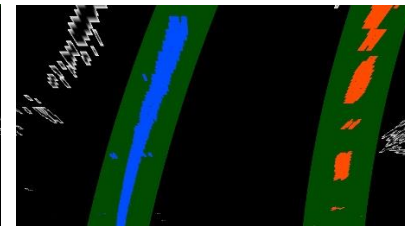


Image after Polynomial fit

Search_from_Prior_test6_
minpix_200.jpg

Sixth, I measured curvature and vehicle position.

```
249 #6. Measuring curvature and vehicle position
250 #Determine the curvature of the lane and vehicle position with respect to center.
251
252 #(1)Measuring curvature
253
254 def measure_curvature_real():
255     """
256     Calculates the curvature of polynomial functions in meters.
257     """
258     # Define conversions in x and y from pixels space to meters
259     #ym_per_pix = 30/720 # meters per pixel in y dimension # Define outside
260     #xm_per_pix = 3.7/700 # meters per pixel in x dimension # Define outside
261
262     # Use Meter-based left_fit, right_fit
263     left_fit_cr = left_fit_pix
264     right_fit_cr = right_fit_pix
265
266     # Define y-value where we want radius of curvature
267     # We'll choose the maximum y-value, corresponding to the bottom of the image
268     #y_eval = np.max(ploty)
269     ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )
270     y_eval = np.max(ploty)* ym_per_pix
271
272     # Calculation of R_curve (radius of curvature)
273     left_curverad = ((1 + (2*left_fit_cr[0]*y_eval + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
274     right_curverad = ((1 + (2*right_fit_cr[0]*y_eval + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
275
276     #print(y_eval)
277     #print(left_fit_cr[0], left_fit_cr[1], left_fit_cr[2])
278     #print(right_fit_cr[0], right_fit_cr[1], right_fit_cr[2])
279
280     return left_curverad, right_curverad
281
282
283 # Calculate the radius of curvature in meters for both lane lines
284 left_curverad, right_curverad = measure_curvature_real()
285 curverad = (left_curverad + right_curverad)/2.0 # Resulting curvature is average of left & right
286
287 #print(left_curverad, right_curverad, curverad)
288
289
290 #(2)Measuring vehicle position
291 center = result.shape[1]/2.0*xm_per_pix
292 position = (left_fitx_pix[-1] + right_fitx_pix[-1])/2.0 - center
```

Seventh, I warped the left and right lane lines (colored green between 2 lines) from bird view to the original images.

I also put texts of curvature and vehicle position.

```
300 #7. Warp back onto original
301 # Warp the detected lane boundaries back onto the original image.
302
303 #(1)Unwarp function (referred warp function)
304 def unwarp(img):
305     # Define calibration box in source and destination
306     img_size = (img.shape[1], img.shape[0])
307
308     # Get src from undistorted images
309     src1 = np.float32([[708,462],[1053,688],[246,688],[577,462]]) # undist_straight_lines1
310     src2 = np.float32([[678,441],[1040,675],[280,675],[607,441]]) # undist_straight_lines2
311     src = (src1 + src2)/2 # Use average to get more robust src
312     offsetx = 300
313     offsety = 0
314     dst = np.float32([[img_size[0]-offsetx,offsety],[img_size[0]-offsetx,img_size[1]-offsety],[offsetx,img_size[1]],[offsetx,offsety]])
315
316     # Compute the perspective transform, M
317     M = cv2.getPerspectiveTransform(dst, src)# Switch src and dst to calculate unwarp
318
319     # Create warped image
320     unwarped = cv2.warpPerspective(img, M, img_size, flags = cv2.INTER_LINEAR)
321
322     return unwarped
```



```

324 # (2) Unwarp lane finding image and plot onto the original image
325 unwarped = unwarp(result)
326 # final_result = unwarped
327 final_result = cv2.addWeighted(original_img, 1.0, unwarped, 0.3, 0)
328
329 # (3) Show the result
330 text1 = "Curvature[m]: " + str(round(curverad, 1))
331 if position > 0:
332     text2 = "Position to Left[m]: " + str(round(abs(position), 3))
333 else:
334     text2 = "Position to Right[m]: " + str(round(abs(position), 3))
335 cv2.putText(final_result, text1, (350, 100), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 5, cv2.LINE_AA)
336 cv2.putText(final_result, text2, (350, 200), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 5, cv2.LINE_AA)
337
338 plt.imshow(final_result)
339 plt.title('Final result')
340 cv2.imwrite('output_images/Final_result_'+file_name+'.jpg', cv2.cvtColor(final_result, cv2.COLOR_BGR2RGB))

```

The lane lines were detected well from the resulting images.

Data folder is "output_images\6_Result".



Image after unwarp

Final_result_test1.jpg



Image after unwarp

Final_result_test2.jpg



Image after unwarp

Final_result_test3.jpg



Image after unwarp

Final_result_test4.jpg



Image after unwarp

Final_result_test5.jpg



Image after unwarp

Final_result_test6.jpg

Finally eighth, I ran the video file "project_video.mp4".

(2) Result of my pipeline

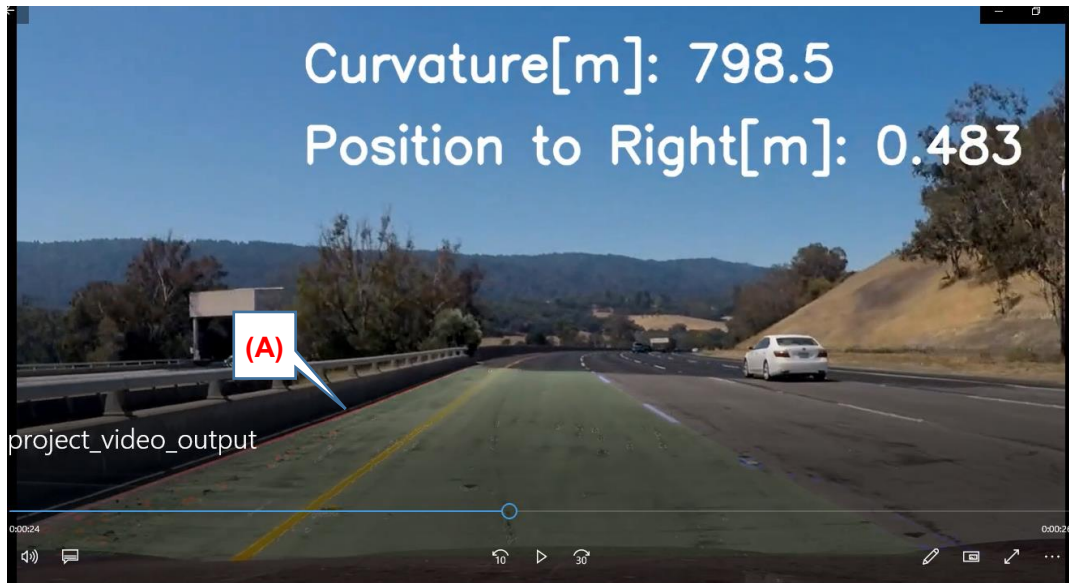
My pipeline looked mostly well with the video file "project_video.mp4".

The folder is "test_videos_output/before_countermeasure" and the video file name is

"**project_video_output.mp4**". (1st submit on July 18th)



However, there was just one moment when the left lane detected the edge of the wall as below (A). It needs to be improved.



project_video_output.mp4

[Countermeasures]

(1) Avoid the sudden change of lane

In the function “fit_polynomial()”, I added the following part to restrict the coefficients of polynomial lines.

```
#Calculate fitting polynomial
def fit_polynomial(binary_warped):
    # Find our lane pixels first
    leftx, lefty, rightx, righty, out_img = find_lane_pixels(binary_warped)

    ### TO-DO: Fit a second order polynomial to each using 'np.polyfit' ###
    left_fit = np.polyfit(leftx, lefty, 2)
    right_fit = np.polyfit(rightx, righty, 2)

    #####
    # If the left_fit or right_fit changes a lot, use the global fit #
    #####
    global left_fit_global, right_fit_global, cnt1_global, cnt2_global, cnt3_global

    if (left_fit_global[2]!=0 and right_fit_global[2]!=0):
        if ((abs(left_fit_global[0] - left_fit[0])>0.001 or (abs(right_fit_global[0] - right_fit[0])>0.001)%
        or (abs(left_fit_global[1] - left_fit[1])>1.0 or (abs(right_fit_global[1] - right_fit[1])>1.0)%
        or (abs(left_fit_global[2] - left_fit[2])>200 or (abs(right_fit_global[2] - right_fit[2])>200)):
            left_fit = left_fit_global
            right_fit = right_fit_global

        else:
            left_fit_global = left_fit
            right_fit_global = right_fit
            cnt2_global += 1

    else:
        left_fit_global = left_fit
        right_fit_global = right_fit
        cnt3_global += 1

    #####

    # Generate x and y values for plotting
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )
```

If any coefficients of left_fit or right_fit changes a lot, use the previous coefficient.

If no coefficients changed a lot, set them for the next control step.

If it's the first control step, just set the coefficients.

(2) Smoothing the line

In the function “fit_poly()”, I added the following part for smoothing the coefficients of polynomial lines.

```
#(2)Search lines from Polynomial fit values from the previous frame
def fit_poly(img_shape, leftx, lefty, rightx, righty):
    def fit_poly(img_shape, leftx, lefty, rightx, righty):
        ### T0-D0: Fit a second order polynomial to each with np.polyfit() ###

        left_fit = np.polyfit(leftx, lefty, 2)
        right_fit = np.polyfit(rightx, righty, 2)

        #####
        # Smoothing left_fit and right_fit
        #####
        global left_fit_global2, right_fit_global2, left_fit_global3, right_fit_global3

        if (len(leftx)==0 or len(rightx)==0):
            left_fit = left_fit_global2
            right_fit = right_fit_global2
        else:
            left_fit = np.polyfit(leftx, lefty, 2)
            right_fit = np.polyfit(rightx, righty, 2)

        if (left_fit_global2[2]!=0 and right_fit_global2[2]!=0 and left_fit_global3[2]!=0 and right_fit_global3[2]!=0):
            for i in range(3):
                left_fit[i] = left_fit_global2[i]*0.4 + left_fit_global3[i]*0.4 + left_fit[i]*0.2
                right_fit[i] = right_fit_global2[i]*0.4 + right_fit_global3[i]*0.4 + right_fit[i]*0.2
                #left_fit[i] = left_fit_global2[i]*0.2 + left_fit_global3[i]*0.2 + left_fit[i]*0.6
                #right_fit[i] = right_fit_global2[i]*0.2 + right_fit_global3[i]*0.2 + right_fit[i]*0.6

            left_fit_global3 = left_fit_global2
            right_fit_global3 = right_fit_global2
            left_fit_global2 = left_fit
            right_fit_global2 = right_fit
        #####
```

If there're no candidate dots, use the previous coefficients.

Else, set the coefficient same as before I changed.

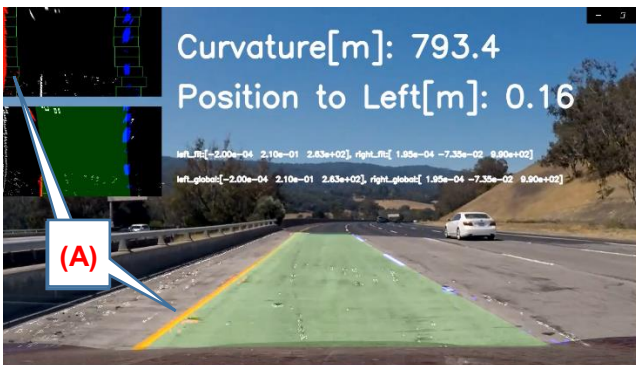
I calculate moving average by using the 1st previous and 2nd previous coefficients.

I ran the new pipeline with the project video.

The folder is “test_videos_output/after_countermeasure” and the video file name is

“project video output after.mp4”.

The lane line became much more robust, and the sudden lane change didn't occur even though the Histogram Detect the wrong line suddenly (A).



project video output after.mp4



project video output after.mp4

(3) Identification of potential shortcomings with my current pipeline, and suggestion of possible improvements

I also ran my pipeline with “challenge_video.mp4” and “harder_challenge_video.mp4”.

I found other 3 potential shortcomings with my current pipeline as bellow:

(i) Robustness against bad road condition

If there are clack lines on the road, my pipeline mismatched them (A).

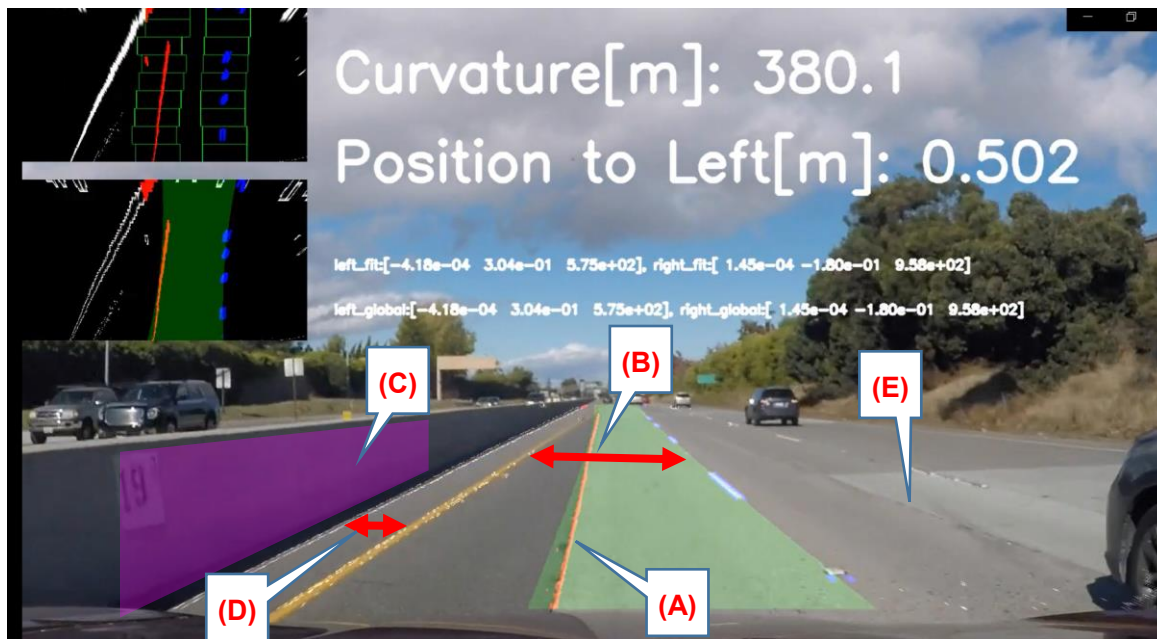
One of countermeasures is assuming the width of lane by map information if the lane width is different from it (B).

However, if the clack lines are deep there's no idea which lane is real lane.

We may be able to assume the left lane by Sensor Fusion. We can detect the position of wall by Radar or LiDAR (C), and we know how much distance is there between wall and left lane by High Definition Map Information (D), so we can set the higher weight around the real left lane during lane detection algorithm.

We also may be able to assume the right lane by detecting another vehicle on the adjacent lane.

If there's another vehicle in the right side, we can set lower priority on the right side clack line area (E) to detect the real lane.



challenge video output after need improvement.mp4

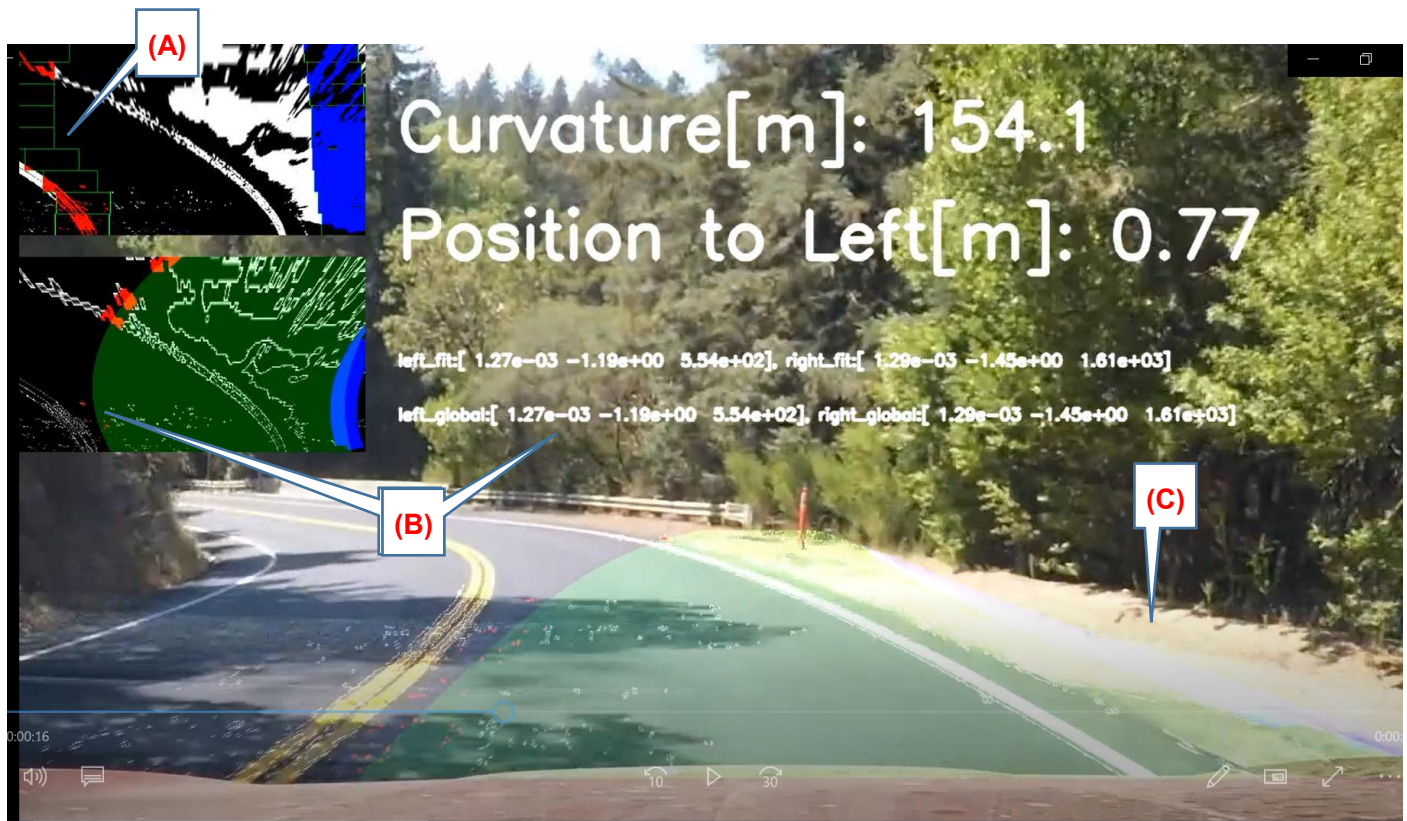
(ii) Robustness against frequent curvature change

Red dots are left side and blue dots are right side.

When there's frequent curvature change, Histogram detected left side (A) but the countermeasure "Avoid the sudden change of lane" worked and final curve detected the opposite side (B).

I think it will be more robust if I set the restriction threshold of "Avoid the sudden change of lane" according to the road shape.

There are some area without wall (C) and this is one side lane, so the Sensor Fusion countermeasures written in the previous page don't work in this scene.



harder challenge video output after bad need improvement.mp4

(iii) Robustness against steep curve

In case of steep curve, the lane detection doesn't match the real lane at all.

My pipeline covers only straight roads and gradual curves (A).

However, on steep curves the coverage area should be changed like (B) and I should not detect dots in other area.

We can detect whether it's steep curve or not by the following information:

- (i) High Definition Map Information
- (ii) Current angle of steering wheel
- (iii) Current lateral acceleration

And also, decreasing vehicle speed helps self-driving in this scene by the following viewpoints:

- The curvature changes slowly, so lane detection by camera image will be more robust.
- Even if lane detection mismatches sometimes, the vehicle behavior will be slowly so it's easier to recover to the stable status.



harder challenge video output after bad need improvement.mp4