Lesson 03: Advanced Techniques for Lane Finding/Advanced Techniques for Lane Finding.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2

global fig , axis_array

def warp(img):

    image_size = (img.shape[1] , img.shape[0])

    # Four Source Coordinates
    src = np.float32([
        [560,485],
        [745,475],
        [1064,684],
        [221,689]
    ])

    # Four Desired Coordinates
    dst = np.float32([
        [350,0],
        [950,0],
        [950,700],
        [350,700]
    ])

    # Compute the prespective transform, M
    M = cv2.getPerspectiveTransform(src,dst)

    # Could compute the inverse also by swaping the input parameters
    Minv  = cv2.getPerspectiveTransform(dst,src)

    # Create transformed image - uses linear interpolation
    warped = cv2.warpPerspective(img,M,image_size,flags=cv2.INTER_LINEAR)

    return warped , Minv



def color_and_gradient_threshold(img):

    hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
    s_channel = hls[:,:,2]
    l_channel = hls[:,:,1]

    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    v_channel = hsv[:,:,2]

    # Grayscale image
    # NOTE: we already saw that standard grayscaling lost color information for the lane lines
    # Explore gradients in other colors spaces / color channels to see what might work better
```

```
53      gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
54
55      # Sobel x
56      sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0) # Take the derivative in x
57      abs_sobelx = np.absolute(sobelx) # Absolute x derivative to accentuate lines away
    from horizontal
58      scaled_sobel = np.uint8(255*abs_sobelx/np.max(abs_sobelx))
59
60      # Threshold x gradient
61      thresh_min = 20
62      thresh_max = 100
63      sxbinary = np.zeros_like(scaled_sobel)
64      sxbinary[(scaled_sobel >= thresh_min) & (scaled_sobel <= thresh_max)] = 1
65
66      # Threshold color channel - saturation used to best detect lanes
67      s_thresh_min = 115
68      s_thresh_max = 255
69      s_binary = np.zeros_like(s_channel)
70      s_binary[(s_channel >= s_thresh_min) & (s_channel <= s_thresh_max)] = 1
71
72      # lightness threshold was used to better detect white lines
73      l_thresh_min = 200
74      l_thresh_max = 255
75      l_binary = np.zeros_like(l_channel)
76      l_binary[(l_channel >= l_thresh_min) & (l_channel <= l_thresh_max)] = 1
77
78      # value threshold was used to better detect yellow lines
79      v_thresh_min = 230
80      v_thresh_max = 255
81      v_binary = np.zeros_like(s_channel)
82      v_binary[(v_channel >= v_thresh_min) & (v_channel <= v_thresh_max)] = 1
83
84      # Stack each channel to view their individual contributions in green and blue
    respectively
85      # This returns a stack of the two binary images, whose components you can see as
    different colors
86      color_binary = np.dstack(( np.zeros_like(sxbinary), sxbinary, s_binary))
87
88      # Combine the two binary thresholds
89      combined_binary = np.zeros_like(sxbinary)
90      combined_binary[(s_binary == 0.5) & (l_binary == 0.5)| (v_binary == 1) |
    (sxbinary == 1)] = 1
91      #plt.imshow(combined_binary,cmap='gray')
92      #plt.show()
93      return combined_binary
94
95
96
97  def find_lane_line(binary_warped, return_img=False):
98
99      # Take a histogram of the bottom half of the image
100     histogram = np.sum(binary_warped[binary_warped.shape[0]//3:,:], axis=0)
101     axis_array[0,1].plot(histogram)
102
103     # Create an output image to draw on and visualize the result
104     out_img = np.dstack((binary_warped, binary_warped, binary_warped))
105     out_img = (out_img*255).astype('uint8')
```

```
106
107     # Find the peak of the left and right halves of the histogram
108     # These will be the starting point for the left and right lines (right : 955,
    left : 326, mid : 426)
109     midpoint = int(histogram.shape[0]//3)
110     leftx_base = np.argmax(histogram[:midpoint])
111     rightx_base = np.argmax(histogram[midpoint:]) + midpoint
112
113     # HYPERPARAMETERS
114     nwindows = 9 # Choose the number of sliding windows
115     margin = 100 # Set the width of the windows +/- margin
116     minpix = 50 # Set minimum number of pixels found to recenter window
117     window_height = int(binary_warped.shape[0]//nwindows) # Set height of windows -
    based on nwindows above and image shape
118
119     # Identify the x and y positions of all nonzero pixels in the image
120     nonzero = binary_warped.nonzero()
121     nonzeroy = np.array(nonzero[0])
122     nonzerox = np.array(nonzero[1])
123
124     # Current positions to be updated later for each window in nwindows
125     leftx_current = leftx_base
126     rightx_current = rightx_base
127
128     # Create empty lists to receive left and right lane pixel indices
129     left_lane_inds = []
130     right_lane_inds = []
131
132
133     for window in range(nwindows): # Step through the windows one by one
134
135         # Identify window boundaries in x and y (and right and left)
136         win_y_low = binary_warped.shape[0] - (window+1)*window_height
137         win_y_high = binary_warped.shape[0] - window*window_height
138
139         #Find the four below boundaries of the window
140         win_xleft_low = leftx_current - margin
141         win_xleft_high = leftx_current + margin
142         win_xright_low = rightx_current - margin
143         win_xright_high = rightx_current + margin
144
145         # Draw the windows on the visualization image
146         cv2.rectangle(out_img,(win_xleft_low,win_y_low),
147         (win_xleft_high,win_y_high),(0,255,0), 5)
148         cv2.rectangle(out_img,(win_xright_low,win_y_low),
149         (win_xright_high,win_y_high),(0,255,0), 5)
150
151         # Identify the nonzero pixels in x and y within the window
152         good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
153         (nonzerox >= win_xleft_low) &  (nonzerox < win_xleft_high)).nonzero()[0]
154         good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
155         (nonzerox >= win_xright_low) &  (nonzerox < win_xright_high)).nonzero()[0]
156
157         # Append these indices to the lists
158         left_lane_inds.append(good_left_inds)
159         right_lane_inds.append(good_right_inds)
160
```

```
        # If you found > minpix pixels, recenter next window on their mean position
        if len(good_left_inds) > minpix:
            leftx_current = int(np.mean(nonzerox[good_left_inds]))
        if len(good_right_inds) > minpix:
            rightx_current = int(np.mean(nonzerox[good_right_inds]))


    # Concatenate the arrays of indices (previously was a list of lists of pixels)
    try:
        left_lane_inds = np.concatenate(left_lane_inds)
        right_lane_inds = np.concatenate(right_lane_inds)
    except ValueError:
        # Avoids an error if the above is not implemented fully
        pass

    # 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]

    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0] )

    try:
        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]
    except TypeError:
        # Avoids an error if `left` and `right_fit` are still none or incorrect
        print('The function failed to fit a line!')
        left_fitx = 1*ploty**2 + 1*ploty
        right_fitx = 1*ploty**2 + 1*ploty

    ## Visualization ##
    # Colors in the left and right lane regions
    out_img[lefty, leftx] = [255, 0, 0]
    out_img[righty, rightx] = [100, 200, 255]
    if return_img:

        # Plots the left and right polynomials on the lane lines
        axis_array[0,1].plot(left_fitx, ploty, color='yellow')
        axis_array[0,1].plot(right_fitx, ploty, color='yellow')

    # leftx and rightx is the predition from polynominal fit points
    # left_fit and right_fit are the coefficients

    return out_img,left_fitx,right_fitx,ploty,left_fit,right_fit



def search_around_poly(binary_warped, left_fit, right_fit, return_img=False):

    margin = 100

    # Grab activated pixels
```

```
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    # Set the area of search based on activated x-values within the +/- margin of our polynomial function
    left_lane_inds = ((nonzerox >
                       (left_fit[0]*(nonzeroy**2) + left_fit[1]*nonzeroy + left_fit[2] - margin)
                     ) & (nonzerox <
                          (left_fit[0]*(nonzeroy**2) + left_fit[1]*nonzeroy + left_fit[2] + margin)))
    right_lane_inds = ((nonzerox > (right_fit[0]*(nonzeroy**2) + right_fit[1]*nonzeroy +
                    right_fit[2] - margin)) & (nonzerox < (right_fit[0]*(nonzeroy**2) +
                    right_fit[1]*nonzeroy + right_fit[2] + margin)))

    # 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]

    # Fit new polynomials
    ploty,left_fit, right_fit, left_fitx, right_fitx = fit_poly(binary_warped.shape, leftx, lefty, rightx, righty)

    ## Visualization ##
    # Create an image to draw on and an image to show the selection window
    out_img = (np.dstack((binary_warped, binary_warped, binary_warped))*255).astype('uint8')
    window_img = np.zeros_like(out_img)

    # Color in left and right line pixels
    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [100, 200, 255]

    # Generate a polygon to illustrate the search window area
    # And recast the x and y points into usable format for cv2.fillPoly()
    left_line_window1 = np.array([np.transpose(np.vstack([left_fitx-margin, ploty]))])
    left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx+margin, ploty])))])
    left_line_pts = np.hstack((left_line_window1, left_line_window2))
    right_line_window1 = np.array([np.transpose(np.vstack([right_fitx-margin, ploty]))])
    right_line_window2 = np.array([np.flipud(np.transpose(np.vstack([right_fitx+margin, ploty])))])
    right_line_pts = np.hstack((right_line_window1, right_line_window2))


    # Draw the lane onto the warped blank image
    cv2.fillPoly(window_img, np.int_([left_line_pts]), (0, 255, 0))
    cv2.fillPoly(window_img, np.int_([right_line_pts]), (0, 255, 0))
    out_img = cv2.addWeighted(out_img, 1, window_img, 0.3, 0)

    if return_img:
        # Plot the polynomial lines onto the image
```

```
265         axis_array[0,1].plot(left_fitx, ploty, color = 'yellow')
266         axis_array[0,1].plot(right_fitx, ploty, color = 'yellow')
267         ## End visualization steps ##
268 
269     return out_img, ploty, left_fit, right_fit, left_fitx, right_fitx
270 
271 
272 
273 def generate_data(ploty, left_fitx, right_fitx, ym_per_pix, xm_per_pix):
274 
275     # generate coefficient values for lane datapoints in meters
276 
277     left_fit_cr = np.polyfit(ploty*ym_per_pix, left_fitx*xm_per_pix, 2)
278     right_fit_cr = np.polyfit(ploty*ym_per_pix, right_fitx*xm_per_pix, 2)
279 
280     return ploty*ym_per_pix, left_fit_cr, right_fit_cr
281 
282 
283 
284 def measure_curvature_real(ploty, left_fitx, right_fitx):
285     '''
286     # calculate average radius of curvature of left and right lane lines
287     # and calculate centre offset of vehicle within lane assuming camera is mounted
    directly in the middle centreline of vehicle
288 
289     Calculates the curvature of two polynomial lane lines in meters from pixel space.
290     '''
291     # define conversions in x and y from pixels space to meters
292     ym_per_pix = 30/720 # meters per pixel in y dimension
293     xm_per_pix = 3.7/700 # meters per pixel in x dimension
294 
295     lane_centre = (left_fitx[-1] + right_fitx[-1])/2
296     centre_offset_pixels = img_size[0]/2 - lane_centre
297     # convert to metres from pixels using conversion
298     centre_offset_metres = xm_per_pix*centre_offset_pixels
299 
300     # generate data points for left and right curverad
301     ploty, left_fit_cr, right_fit_cr = generate_data(ploty, left_fitx, right_fitx,
    ym_per_pix, xm_per_pix)
302 
303     # define y-value where we want radius of curvature from the bottom of the image
304     y_eval = np.max(ploty)
305 
306     ##### Implement the calculation of R_curve (radius of curvature) #####
307     left_curverad = ((1+(2*left_fit_cr[0]*y_eval + left_fit_cr[1])**2)**1.5)
    //(2*abs(left_fit_cr[0]))  ## Implement the calculation of the left line here
308     right_curverad = ((1+(2*right_fit_cr[0]*y_eval + right_fit_cr[1])**2)**1.5)
    //(2*abs(right_fit_cr[0]))  ## Implement the calculation of the right line here
309     average_curvature = (left_curverad + right_curverad)/2
310 
311     return average_curvature, centre_offset_metres, left_curverad, right_curverad
312 
313 
314 
315 
316 def fit_poly(img_shape, leftx, lefty, rightx, righty):
317 
```

```
    # Fit a second order polynomial to each with np.polyfit() ###
    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, img_shape[0]-1, img_shape[0])

    # Calc both polynomials using ploty, left_fit and right_fit ###
    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 ploty, left_fit, right_fit, left_fitx, right_fitx




def generate_data(ploty, left_fitx, right_fitx, ym_per_pix, xm_per_pix):

    # generate coefficient values for lane datapoints in meters

    left_fit_cr = np.polyfit(ploty*ym_per_pix, left_fitx*xm_per_pix, 2)
    right_fit_cr = np.polyfit(ploty*ym_per_pix, right_fitx*xm_per_pix, 2)

    return ploty*ym_per_pix, left_fit_cr, right_fit_cr



def draw_shade(img, warped, left_fit, right_fit, ploty, Minv):

    # Create an image to draw the lines on
    warp_zero = np.zeros_like(warped).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

    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]

    # 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))

    # Warp the blank back to original image space using inverse perspective matrix
(Minv)
    newwarp = cv2.warpPerspective(color_warp, Minv, (img_size[0], img_size[1]))

    return newwarp


fig , axis_array = plt.subplots(3,3, figsize=(24, 9))


img_path = '/home/fayo/Udacity - Self Driving Car Nanodegree/Part 1 _ Computer Vision
and Deep Learning/Module 04_ Computer Vision/Lesson 03: Advanced Techniques for Lane
Finding/color-shadow-example.jpg'
```

```
372 img = plt.imread(img_path)
373 img_size = (img.shape[1], img.shape[0])
374
375 warped_image , Minv= warp(img)
376
377
378 combined_binary = color_and_gradient_threshold(warped_image)
379
380 axis_array[0,0].imshow(img)
381 axis_array[1,0].imshow(warped_image)
382 axis_array[2,0].imshow(combined_binary,cmap='gray')
383
384
385 out_img, left_fitx, right_fitx, ploty, left_fit, right_fit =
    find_lane_line(combined_binary, return_img=True)
386 axis_array[1,1].imshow(out_img,cmap='gray')
387
388 out_img, ploty, left_fit, right_fit, left_fitx, right_fitx =
    search_around_poly(combined_binary, left_fit, right_fit, return_img=True)
389 axis_array[2,1].imshow(out_img,cmap='gray')
390
391
392 average_curvature, centre_offset_metres, left_curverad, right_curverad =
    measure_curvature_real(ploty, left_fitx, right_fitx)
393 print("Average Curvature: " + str(average_curvature) + " m")
394 print("Vehicle Offset from Centre of Lane: " + str(centre_offset_metres) + " m")
395
396
397 shade_lane = draw_shade(img, combined_binary, left_fit, right_fit, ploty, Minv)
398 original_and_shade = cv2.addWeighted(img, 1, shade_lane, 0.3, 0)
399 axis_array[0,2].imshow(original_and_shade)
400
401
402 axis_array[0,1].imshow(out_img)
403
404 plt.show()
```