Lane detection

Import libraries that will be needed in the program

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import io
import base64
import cv2
import tensorflow as tf
import sys
#import imageio
#imageio.plugins.ffmpeg.download()
from moviepy.editor import VideoFileClip
from IPython.display import HTML
%matplotlib inline
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import io
import base64
import cv2
import sys
#import imageio
#imageio.plugins.ffmpeg.download()
from moviepy.editor import VideoFileClip
from IPython.display import HTML
%matplotlib inline
```

Convert the initial image to either yuv volor scheme or Gray color in order to highlight the lanes.
Note the YUV transform shows improvements of detecting the yellow lane under shadow.
Next apply GaussianBlur to blur the image a little bit.
Then use Canny function to get a binary image of all edges, which is also saved and visualized in a small window in the final video.
Next we apply a mask to focus on the region of interests: the road ahead of the car.
Next we apply Hough transform to output all the possible lines in this region.

```
def generate_lines(image):
    # convert to YUV color scheme
    yuv_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
    # convert the image to grayscale
    #gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    kernel_size = 7
    blur_gray = cv2.GaussianBlur(yuv_image,(kernel_size, kernel_size),0)
    # Define parameters for Canny
    low_threshold = 40
    high_threshold = 80
    edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
    # create a masked edges image using cv2.fillPoly()
    mask = np.zeros_like(edges)
    ignore_mask_color = 255
    mask = np.zeros_like(edges)
    imshape = image.shape
    vertices = np.array([[(int(imshape[1]*0.52),int(imshape[0]*0.58)),(int(imshape
        [1]*0.1), int(imshape[0])), (int(imshape[1]*0.9), int(imshape[0]*0.98)), (int(
        imshape[1]*0.55),int(imshape[0]*0.6))]], dtype=np.int32)
    cv2.fillPoly(mask, vertices, 255)
    masked_edges = cv2.bitwise_and(edges, mask)
    # Define the Hough transform parameters
    # Make a blank the same size as our image to draw on
    rho = 0.8 # distance resolution in pixels of the Hough grid
    theta = np.pi/180 # angular resolution in radians of the Hough grid
    threshold = 25     # minimum number of votes (intersections in Hough grid cell)
    min_line_length = 50 #minimum number of pixels making up a line
    max_line_gap = 200    # maximum gap in pixels between connectable line segments
    line_image = np.copy(image)*0 # creating a blank to draw lines on
    # Run Hough on edge detected image
    # Output "lines" is an array containing endpoints of detected line segments
    lines = cv2.HoughLinesP(masked_edges, rho, theta, threshold, np.array([]),
                            min_line_length, max_line_gap)
    return lines, edges
```

The following function is to perform two actions:
First, a initial filter is applied to remove all the short horizontal lines.
Second, we separate the left and right lines based on their slope, which is also saved in the variables.

```
def separate_lanes (lines):
    # separate the left and right lanes
    left_lane = []
    right_lane = []
    for x1,y1,x2,y2 in lines[:,0]:
        if abs(y1-y2)>20: # filter out the horizontal lines
            k = (float(y2) - y1) / (x2 - x1)
            if k > 0:
                right_lane.append([x1,y1,x2,y2,k])
            else:
                left_lane.append([x1,y1,x2,y2,k])
    return np.array(left_lane), np.array(right_lane)
```

Next we further filter the lines based on their slope.

```
def filter_lanes(lane, minimum, maximum):
    # filter the lanes based on the slope
    lane = lane[ (maximum>lane[:,4]) & (lane[:,4]>minimum) ]
    return lane
```

Here we can combine all the lines using linear regression.

```
def combine_lane(points):
    # use linear regression to get one line for each lane
    points = points[:,0:4]
    points = points.reshape(points.shape[0]*2,2)
    m,b = np.polyfit(points[:,0], points[:,1], 1)
    return m, b
```

Next we can draw lines on an empty image.

```
def draw_line(lines, y_left, y_right, line_image):
    left_lane, right_lane = separate_lanes (lines)
    left_lane = filter_lanes(left_lane, -0.9, -0.6)
    right_lane = filter_lanes(right_lane, 0.45, 0.75)
    m_left, b_left = combine_lane(left_lane)
    m_right, b_right = combine_lane(right_lane)
    x_left = [int((xx - b_left)/m_left) for xx in y_left]
    x_right = [int((xx - b_right)/m_right) for xx in y_right]
    cv2.line(line_image,(x_left[0],y_left[0]),(x_left[1],y_left[1]),(0,0,255),10)
    cv2.line(line_image,(x_right[0],y_right[0]),(x_right[1],y_right[1]),(0,0,255),10)
    return line_image
```

The following function is to insert a small image (e.g., image of edges or lanes) inside another image.

```
def embed_image(image, embed_image, x_offset, y_offset):
    image[y_offset:y_offset+embed_image.shape[0], x_offset:x_offset+embed_image.shape
        [1]] = embed_image
    return image
```

Here is the main function to process a frame in the video.

```
def add_lanes(image):
    imshape = image.shape
    lines, edges = generate_lines(image)
    line_image = np.copy(image)*0 # creating a blank to draw lines on
    y_left = [int(imshape[0]*0.95), int(imshape[0]*0.62)] # y_coordinates of left lane
    y_right = [int(imshape[0]*0.95), int(imshape[0]*0.62)] # x_coordinates of right
        lane
    line_image = draw_line(lines, y_left, y_right, line_image)
    small_edges = cv2.resize(edges, (0,0), fx = 0.2, fy = 0.2)
    small_edges = np.dstack((small_edges, small_edges, small_edges))
    small_line = cv2.resize(line_image, (0,0), fx = 0.2, fy = 0.2)
    x_offset_edge = 50
    y_offset_edge = 30
    x_offset_line = x_offset_edge + small_edges.shape[1] + 10
    y_offset_line = y_offset_edge
    image = embed_image(image, small_edges, x_offset_edge, y_offset_edge)
    image = embed_image(image, small_line, x_offset_line, y_offset_line)
    lines_edges = cv2.addWeighted(image, 0.8, line_image, 1, 0)
    cv2.putText(lines_edges,"Edges", (x_offset_edge + 90, y_offset_edge + small_edges.
        shape[0] + 30), cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),2)
    cv2.putText(lines_edges,"Lanes", (x_offset_line + 90, y_offset_edge + small_edges.
        shape[0] + 30), cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),2)
    return lines_edges
```

Here comes the scripts to load and process the initial video.

```
test_clip = VideoFileClip("test.mp4").subclip(0,-1)
new_clip = test_clip.fl_image(add_lanes)
```

Here is the script to make a new video output.

```
new_clip_output = 'test_output.mp4'
%time new_clip.write_videofile(new_clip_output, audio=False)
```

Here is the script to see the final video.

```
# see the movie after processing
video = io.open('test_output.mp4', 'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''
<video width="600" height="500" controls>
    <source src="data:video/mp4;base64,{0}" type="video/mp4" />
</video>'''.format(encoded.decode('ascii')))
```

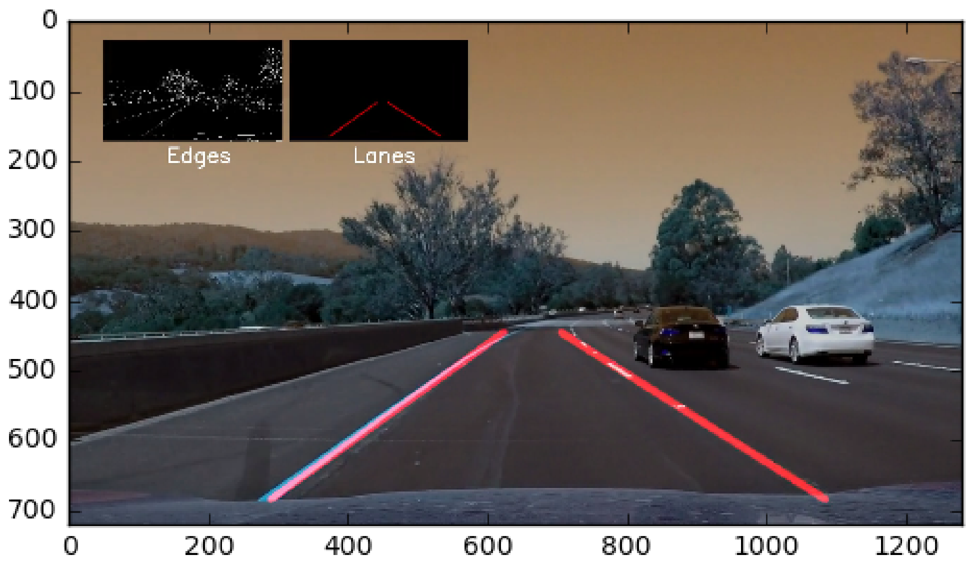

Figure 1: A frame of the final output video for lane detection. The left and right lanes are highlighted. Embedded small images on top-left corner are image of all edges and the two lanes only.

Future improvement:
The above scripts has a potential to fail to detect the lanes under the following situation: 1) under dart environment where contrast between the lane and surrounding places is not clear enough, 2) in places where long and thin vertical shadows exists between the left and right lanes.

In addition, future improvement includes implement another function to record the pipelines in previous frame to improve stability and avoid errors such as no line detected.