Udacity Self-Driving Car Engineer

Report

- **Project:** Finding Lane Lines
- **Name:** Shao Hongxu

Let me just use some brief sentence to explain the thing I did, basically these are 3 big parts here:

Part 1: import necessary lib
Part 2: identify process_image function
Part 3: convert clips into video. (only for video)

To be specific as below, you will find some note on Script, key take away and Area could be improved:

Part 1: import necessary lib:

In [15]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
from moviepy.editor import VideoFileClip
from IPython.display import HTML
```

Part 2: identify process_image function:

```
def process_image(image):

    color_select = np.copy(image)
    line_image = np.copy(image)
    image_original = image
    ysize = image.shape[0]
    xsize = image.shape[1]

    # MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION
    red_threshold = 190
    green_threshold = 190
    blue_threshold = 0
    rgb_threshold = [red_threshold, green_threshold, blue_threshold]
    left_bottom = [0, 539]
    right_bottom = [960, 539]
    #apex = [480, 270]
    apex = [480, 300]
```

To archive the goal of detecting lane Lines, below is what I have done under process_image function:

2. 1: Using colour selection function to decouple image area which is not lane lines.

```
    color_select = np.copy(image)
    line_image = np.copy(image)
    image_original = image
    ysize = image.shape[0]
    xsize = image.shape[1]

    # MODIFY THESE VARIABLES TO MAKE YOUR COLOR SELECTION
    red_threshold = 190
    green_threshold = 190
    blue_threshold = 0
    rgb_threshold = [red_threshold, green_threshold, blue_threshold]
    left_bottom = [0, 539]
    right_bottom = [960, 539]
    #apex = [480, 270]
    apex = [480, 300]

    # Perform a linear fit (y=Ax+B) to each of the three sides of the triangle
    fit_left = np.polyfit((left_bottom[0], apex[0]), (left_bottom[1], apex[1]), 1)
    fit_right = np.polyfit((right_bottom[0], apex[0]), (right_bottom[1], apex[1]), 1)
    fit_bottom = np.polyfit((left_bottom[0], right_bottom[0]), (left_bottom[1], right_bottom[1]), 1)

    # Mask pixels below the threshold
    color_thresholds = (image[:,:,0] < rgb_threshold[0]) | \
                       (image[:,:,1] < rgb_threshold[1]) | \
                       (image[:,:,2] < rgb_threshold[2])
```

-> Key take away: it has to be a very basic colour selection process considering different colour of line detection, like wine, yellow etc.

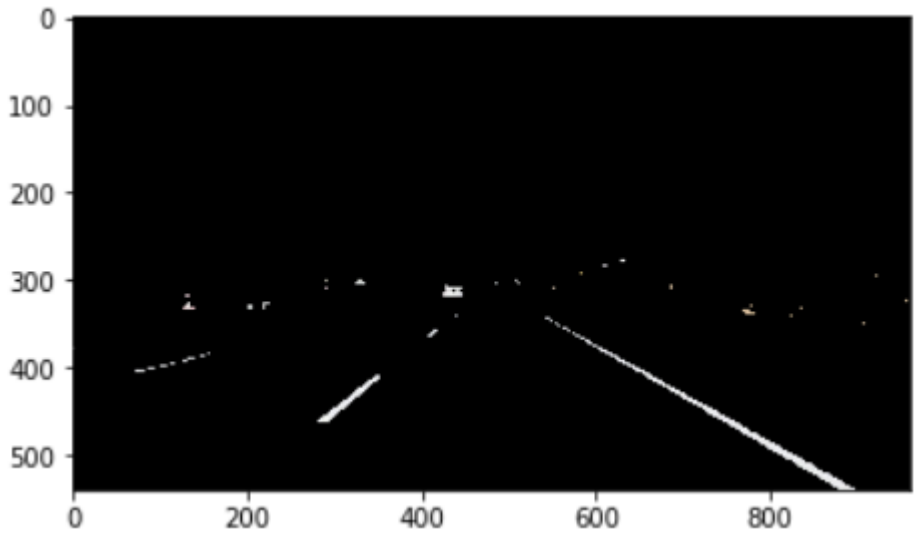
red_threshold = 190
green_threshold = 190
blue_threshold = 0 (as yellow is made by red & green only)

-> Area could be improved: Here we can do a research on how many colours could be for line, and use an "or" logic to cover all different colour conditions. And parameter here not working for all conditions, like in chanllenge.mp4, few scenconds the road is in white and line is in light yellow, in that case, this parameter can't detect the line. This is an important area has to be improved but not yet have any idea. Will update later in case some thought/ idea coming out.

-> Result:

Original image:

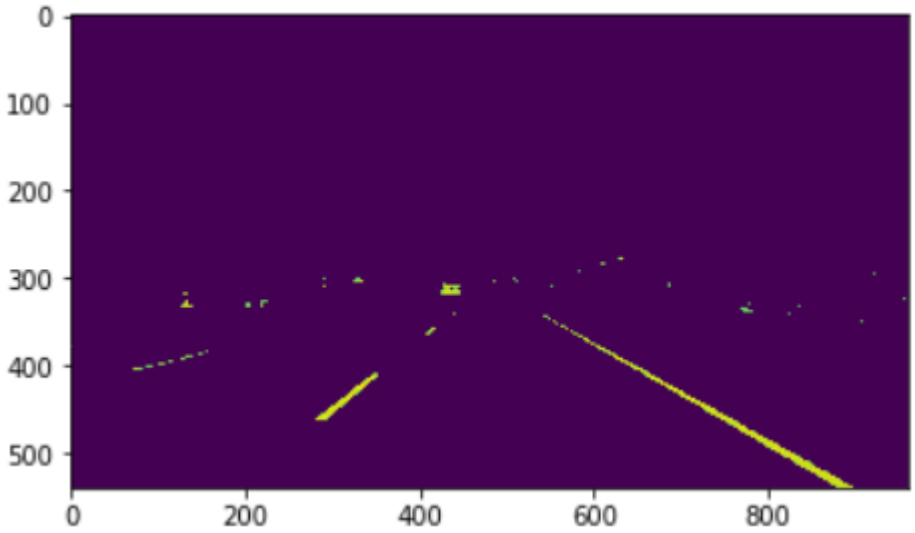
Image after using colour selection function:



2. 2: Gray the image

-> Result:

Image after using Gray function:



2. 3: using Canny function to find edges between low_threshold and high_threshold

```
# Define a kernel size and apply Gaussian smoothing
kernel_size = 3
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)

#plt.imshow(blur_gray)
#plt.show()

# Define our parameters for Canny and apply
low_threshold = 255/3
high_threshold = 255

edges = cv2.Canny(blur_gray, low_threshold, high_threshold)

#plt.imshow(edges)
#plt.show()

# Next we'll create a masked edges image using cv2.fillPoly()
mask = np.zeros_like(edges)
ignore_mask_color = 255

#plt.imshow(mask)
#plt.show()
```

And draw the line in Red:

-> Key take away: low_threshold (Research told me on practice experience say low could be 1/3 of high, even author of function says so, but i need to research in deep especially explanation from math angle)
high_threshold (we should make it as highest 255)

low_threshold = 255/3
high_threshold = 255

-> Area could be improved: how to do it better on selecting below parameters:

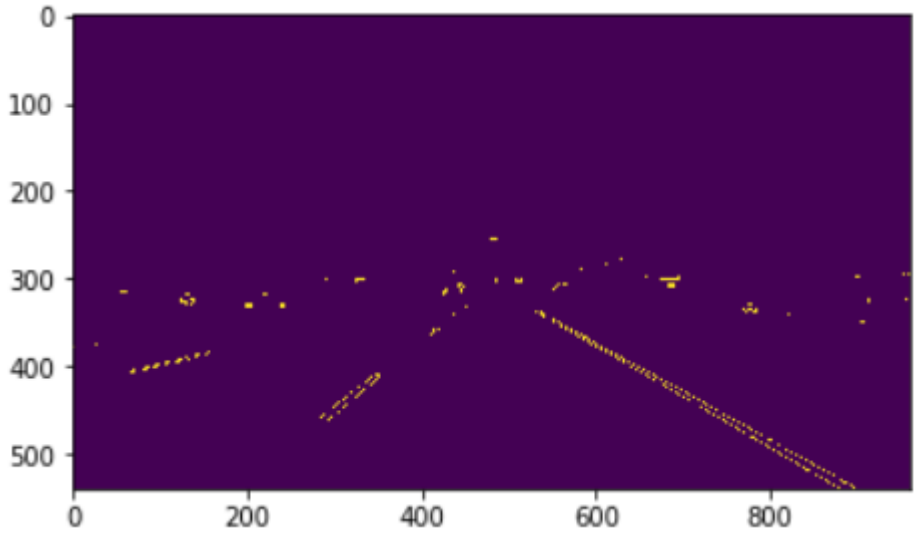
low_threshold

high_threshold

But can we rely on computer vision to choose parameter by itself? like we think about a rule to tell the quality of the result given from different parameter. or machine learning to handle that.

-> Result:

Image after using Canny function:

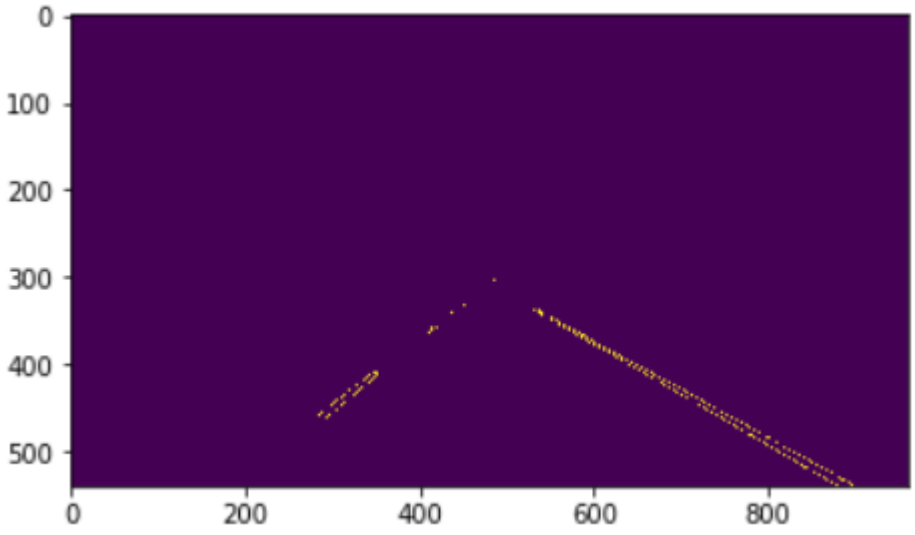


2. 4: to restrict analysis area into a triangle

```
masked_edges[~region_thresholds] = 0
#plt.imshow(masked_edges)
#plt.show()
```

-> Result:

Image after using restrict function:



2. 5: using HoughLinesP to find target line.

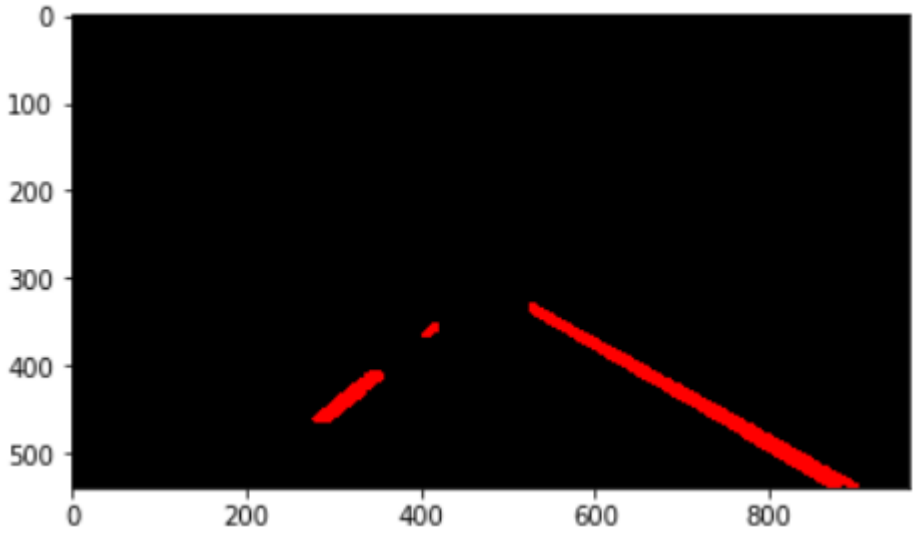
```
# This time we are defining a four sided polygon to mask
imshape = image.shape
vertices = np.array([[(0,imshape[0]),(0, 0), (imshape[1], 0), (imshape[1],imshape[0])]], dtype=np.int32)
cv2.fillPoly(mask, vertices, ignore_mask_color)
masked_edges = cv2.bitwise_and(edges, mask)

masked_edges[~region_thresholds] = 0
#plt.imshow(masked_edges)
#plt.show()

# Define the Hough transform parameters
# Make a blank the same size as our image to draw on
rho = 1 # distance resolution in pixels of the Hough grid
theta = np.pi/180 # angular resolution in radians of the Hough grid
threshold = 15     # minimum number of votes (intersections in Hough grid cell)
min_line_length = 10 #minimum number of pixels making up a line
max_line_gap = 6    # 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)


# Iterate over the output "lines" and draw lines on a blank image
for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)

# Create a "color" binary image to combine with line image
color_edges = np.dstack((edges, edges, edges))
```

-> Key take away: again, parameter...

rho = 1 # distance resolution in pixels of the Hough grid
theta = np.pi/180 # angular resolution in radians of the Hough grid
threshold = 15   # minimum number of votes (intersections in Hough grid cell)
min_line_length = 30 #minimum number of pixels making up a line
max_line_gap = 10   # maximum gap in pixels between connectable line segments

-> Result:

Image after using HoughLinesP function:



2. 6: Club /addWeighted together with original image

```
    # Draw the lines on the edge image
    lines_edges = cv2.addWeighted(image_original, 0.8, line_image, 1, 0)

    image = lines_edges
    return image
```

-> Result:

Image after using HoughLinesP function:

Part 3: convert clips into video. (only for video)

```
#video
white_clip = clip.fl_image(process_image) #NOTE: this function expects color images!!
white_clip.write_videofile(white_output, audio=False)
```

Summary:

These are the steps I could think of for project 1, as a base line. Still a lot of improvement needed to achieve a better accuracy & more effective exceptional handling model.

The key area has to be improved is few second inside challenge.mp4 video, the road changed from black into white, and a lot of dirty line made by car wheel, and system detected them as line as well, will continue thinking about the solution on it, in case any new idea, will update here accordingly.