Finding Lane Lines on the Road

Write up

July 10th, 2021 Kenta Kumazaki

1. Purpose

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road. (figure1)
- Reflect on my work in a written report.

2. Submission

(1) GitHub

https://github.com/kkumazaki/Self-Drivig-Car_Project1_Finding-Lane-Lines.git

(2) Directory

<folder: main>

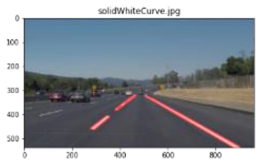
- **Writeup_of_Lesson3.pdf**: This file
- **P1.jpynb**: Pipeline

<folder: test_images_output>

<folder: without_average_extrapolate>

“image file name”_low_”tuning value”_high_”tuning value”.png:

Lane lines without average or extrapolate.



<folder: with_average_extrapolate>

final_“image file name”_low_”tuning value”_high_”tuning value”.png:

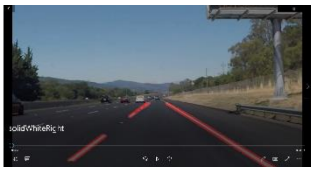
Lane lines with average and extrapolate.

<folder: test_video_output>

<folder: without_average_extrapolate>

“video file name”.mp4:

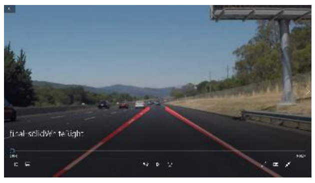
Lane lines without average or extrapolate.



<folder: with_average_extrapolate>

final_“video file name”.mp4:

Lane lines with average and extrapolate.



3. Reflection

(1)Description of my pipeline

My pipeline consisted of 7 steps.

First, I converted the images to grayscale.

```
#reading in an image
#I try each file at first --> If OK, expand to other files.
image = mpimg.imread("test_images/"+image_list[i])
gray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
```

Second, I applied Gaussian smoothing to decrease noise.

```
# Define a kernel size and apply Gaussian smoothing
kernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)
```

Third, I got edges from the images by using Canny Edges.

I tried 3 sets of parameters, and I chose 1st one after checking pictures and movies.

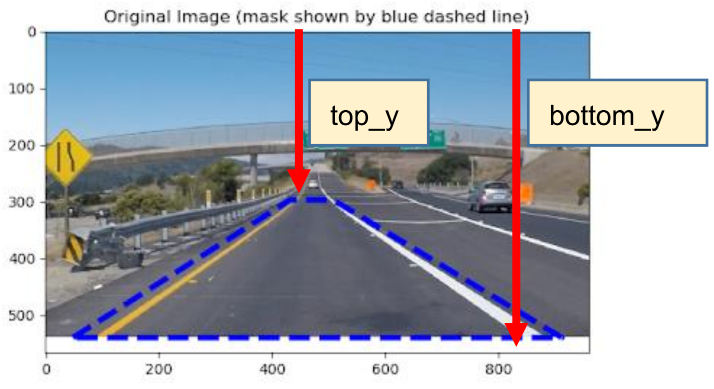
I will explain why I chose it later.

```
# Define our parameters for Canny and apply
#Try 1: Not good at the far point in "whiteCarLaneSwitch.jpg"
low_threshold = 50
high_threshold = 150
#Try 2: Overall, Good.
#low_threshold = 70
#high_threshold = 200
#Try 3: Not good. Some lines are not detected.
#low_threshold = 100
#high_threshold = 300
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
```

Forth, I created a masked area whose shape is four sided polygon.

I tuned that area in Lesson 3.

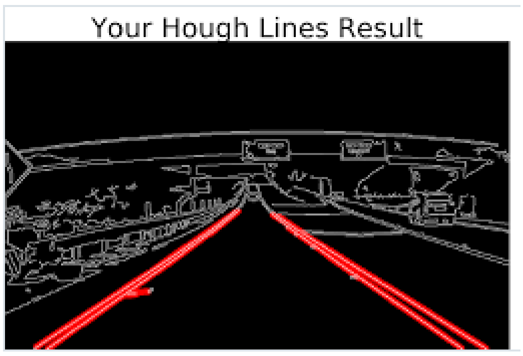
```
# Next we'll create a masked edges image using cv2.fillPoly()
mask = np.zeros_like(edges)
ignore_mask_color = 255

# This time we are defining a four sided polygon to mask
imshape = image.shape
top_y = int(imshape[0]*0.6) #use it for lane calculation
bottom_y = imshape[0] #use it for lane calculation
vertices = np.array([[(imshape[1]*0.05,bottom_y),(imshape[1]*0.45, top_y), (imshape[1]*0.53, top_y), (imshape[1]*0.95,bottom_y)]], dt
cv2.fillPoly(mask, vertices, ignore_mask_color)
masked_edges = cv2.bitwise_and(edges, mask)
```



Fifth, I applied Hough transform.

I tuned the parameters in Lesson 3.

```
# Define the Hough transform parameters
# Make a blank the same size as our image to draw on
rho = 2 # 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 = 5 #minimum number of pixels making up a line
min_line_length = 40 #minimum number of pixels making up a line
#max_line_gap = 1    # maximum gap in pixels between connectable line segments
max_line_gap = 20    # 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)
```



Sixth, I outputted the detected lane lines. I separated 2 modes here.

[Mode1] Just show the Hough Lines results.

The results of this are in the folder “without_average_extrapolate”.

```
# Mode1: Iterate over the output "lines" and draw lines on a blank image
if mode == 1:
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)
```

[Mode2] Extrapolate every Hough Line, then separate them into left side and right side and calculate the average in each side.

The results of this are in the folder “with_average_extrapolate”.

```
    # Mode2: Make only 2 lines by average or extrapolate!! * need to be cahnged
    else:
        #Initialize to calculate average
        count_left= 0
        count_right = 0
        bottom_x_left= 0
        bottom_x_right= 0
        top_x_left= 0
        top_x_right = 0
```

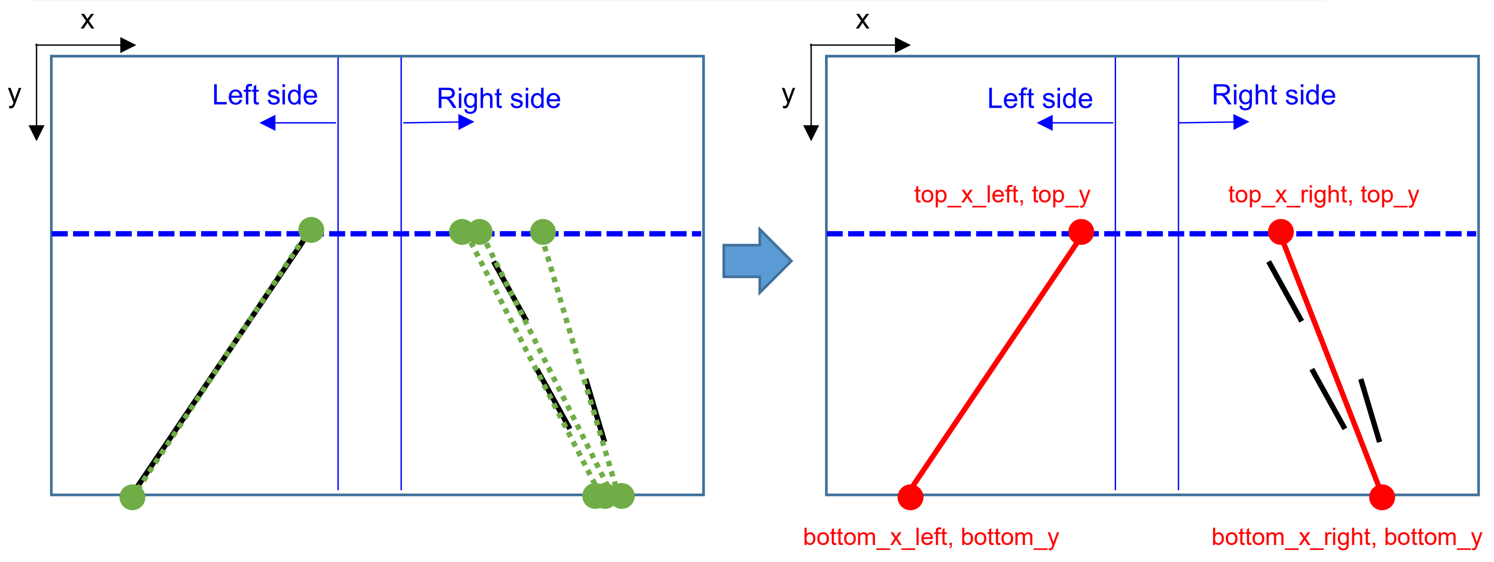
```
#Calculation
for line in lines:
    for x1,y1,x2,y2 in line:
        bottom_x = int((bottom_y - y1)/(y2-y1)*(x2-x1)+x1)
        top_x = int((top_y - y1)/(y2-y1)*(x2-x1)+x1)
        #Eliminate exception
        if bottom_x > 0 and bottom_x < imshape[1]:
            #Left side
            if bottom_x < imshape[1]*0.4:
                count_left += 1
                bottom_x_left += bottom_x
                top_x_left += top_x
            #Right side
            elif bottom_x > imshape[1]*0.6:
                count_right += 1
                bottom_x_right += bottom_x
                top_x_right += top_x
#Calculate average (cv2.line accept "int")
bottom_x_left = int(bottom_x_left/count_left)
bottom_x_right = int(bottom_x_right/count_right)
top_x_left = int(top_x_left/count_left)
top_x_right = int(top_x_right/count_right)

#Show left average line and right average line
cv2.line(line_image,(bottom_x_left,bottom_y),(top_x_left,top_y),(255,0,0),10)
cv2.line(line_image,(bottom_x_right,bottom_y),(top_x_right,top_y),(255,0,0),10)
```

Calculate extrapolate line.

Only inside image size.

Separate left side and right side.
I don't count the lines whose bottom position is around the center.

Calculate average positions of "x".
Positions of "y" are fixed to "top_y" and "bottom_y", so no need to calculate "y" here.



Finally seventh, I drew the lines of the edge image.

I will show the results in the next section.

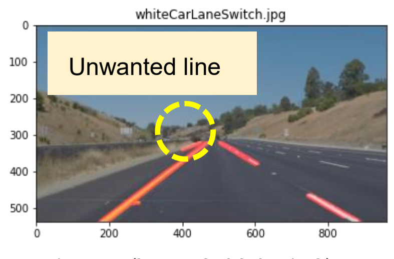
```
# Create a "color" binary image to combine with line image
color_edges = np.dstack((edges, edges, edges))

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

(2) Result of my pipeline

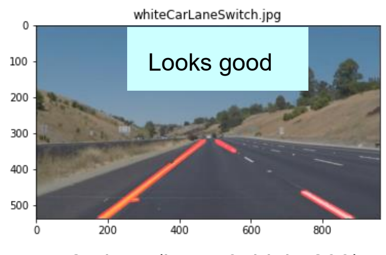
I tuned the parameters of Canny Edges.

```
# Define our parameters for Canny and apply
#Try 1: Not good at the far point in "whiteCarLaneSwitch.jpg"
low_threshold = 50
high_threshold = 150
#Try 2: Overall, Good.
#low_threshold = 70
#high_threshold = 200
#Try 3: Not good. Some lines are not detected.
#low_threshold = 100
#high_threshold = 300
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
```

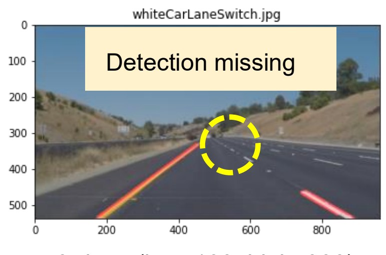
When I checked the figures in "test_images" folder, second set (low: 70, high: 200) looked better than 1st set (low: 50, high: 150) because there were unwanted lines. Second set also looked better than 3rd set (low: 100, high: 300) because there were some lane detection missing.



1st set (low: 50, high: 150)

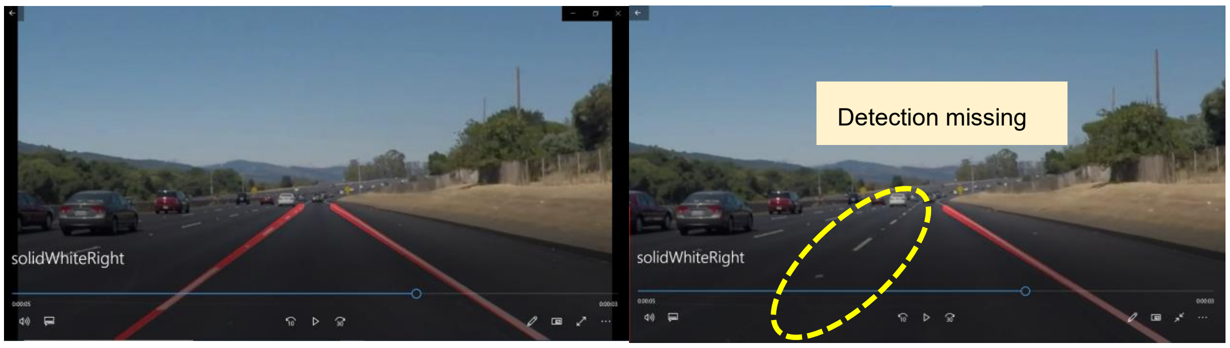


2nd set (low: 70, high: 200)

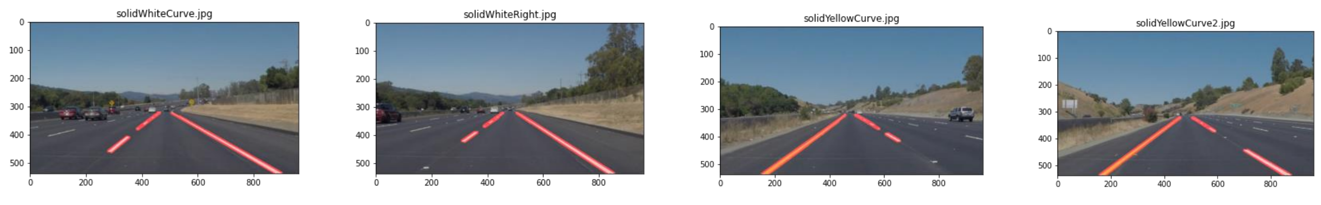


3rd set (low: 100, high: 300)

However, when I ran the movie file (with average and extrapolate), the lane lines disappeared often.
It means that the thresholds of 2nd set (low: 70, high: 200) were too high to detect enough lane lines.



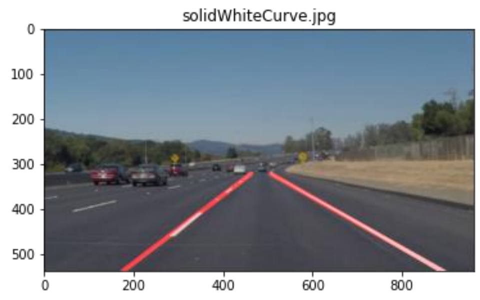
I checked the other figures in "test_images" folder with 1st set, and other resulting figures looked good.
I thought that I can decrease the effect of unwanted lines by extrapolate and average, so I finally chose the tuning parameter as 1st set (low: 50, high: 150).

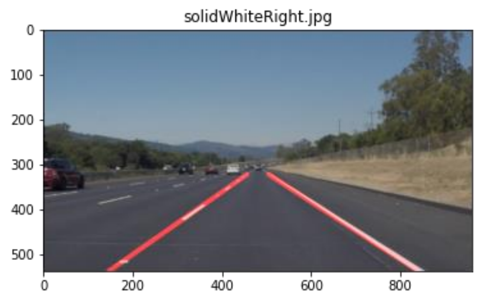


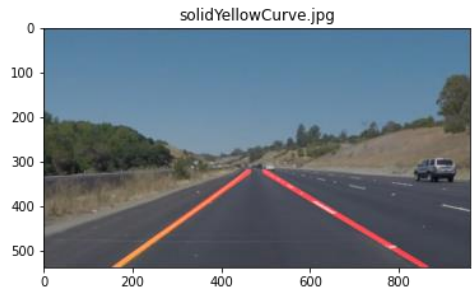
Other resulting figures with tuning parameters: 1st set (low: 50, high: 150)

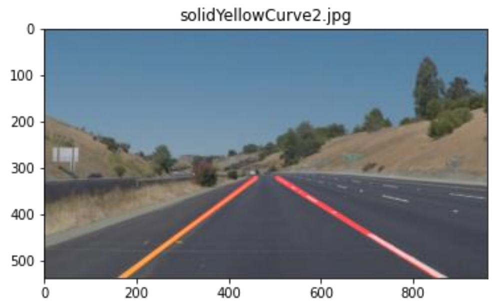
*: They are saved in the folder "without_average_extrapolate", file names are **"file name"_low_50_high_150.png**

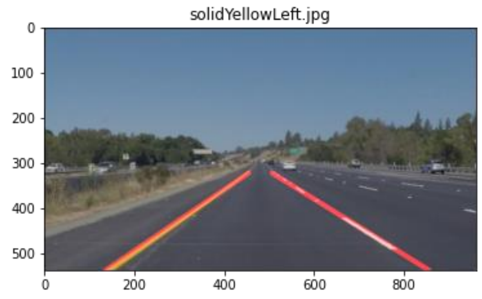
The following figures are the results of extrapolate and average. As I assumed, the unwanted line in the figure "whiteCarLaneSwitch.jpg" disappeared after extrapolate and average.

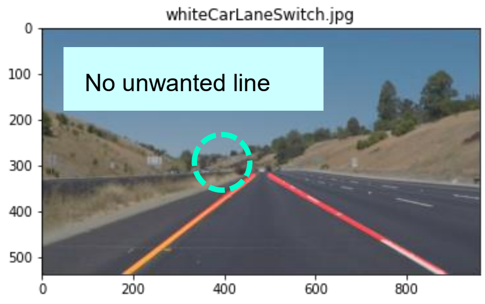










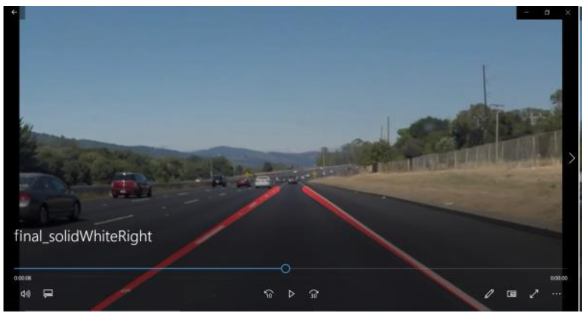


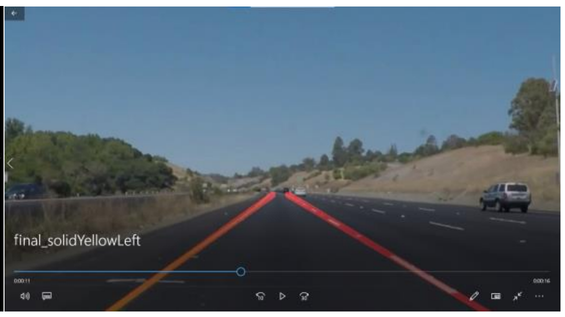
<u>Final resulting figures with tuning parameters: 1st set (low: 50, high: 150)</u>

*: They are saved in the folder "with_average_extrapolate", file names are **final_"file name"_low_50_high_150.png**

I ran my pipeline in the 2 video files, solidWhiteRight.mp4 and solidYellowLeft.mp4.

I checked my pipeline worked well with these sample video files.





<u>Final resulting videos with tuning parameters: 1st set (low: 50, high: 150)</u>

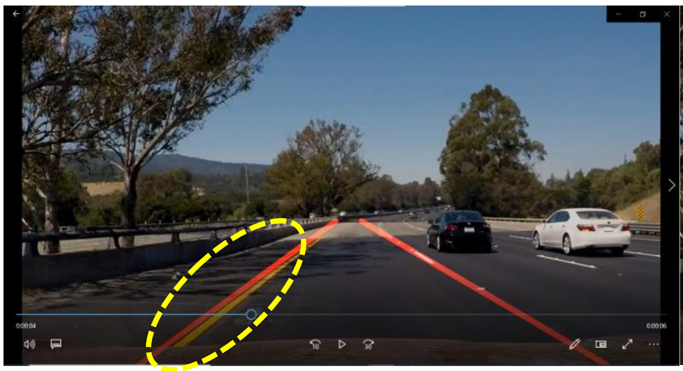
*: They are saved in the folder "with_average_extrapolate", file names are **final_"file name".mp4**

(3) Identification of potential shortcomings with my current pipeline

I ran my pipeline with “challenge.mp4”. I found 3 potential shortcomings with my current pipeline as bellow:

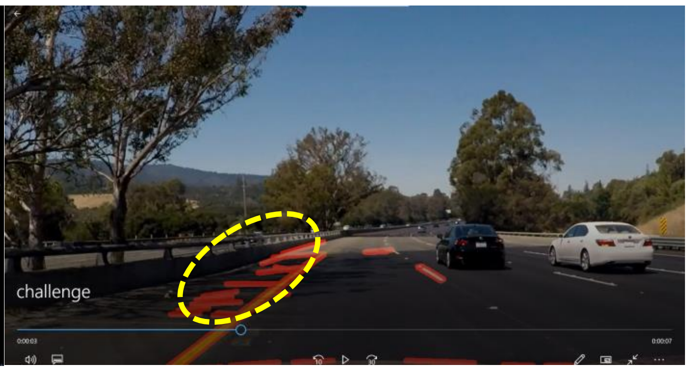
(i) Robustness against shades

The lane line in the left side was detached nearby the tree. I checked Hough Edges in this scene, and there are a lot of unwanted lines which match with tree shades. It affected the result of averaged line in the left side.



Final resulting videos with challenge.mp4

*: File names is **final_challenge.mp4**

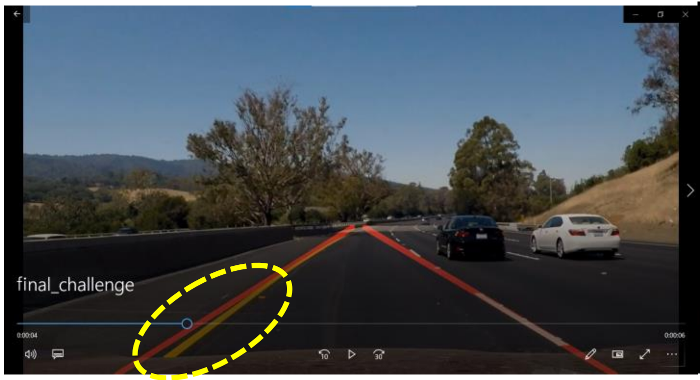


Resulting videos with challenge.mp4 (no extrapolate, average)

*: File names is **challenge.mp4**

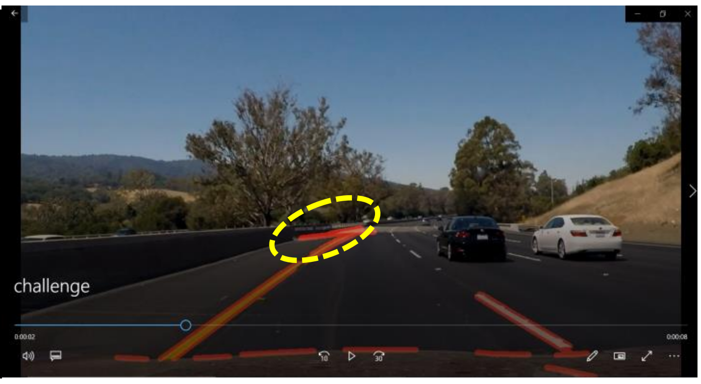
(ii) Robustness against curves

The lane line in the left side was detached in a curve. I checked Hough Edges in this scene, and there was an unwanted edge which matches with the outside lane. It affected the result of averaged line in the left side.
And also, another reason is that I plotted straight line but real lane line is a curve.



Final resulting videos with challenge.mp4

*: File names is **final_challenge.mp4**

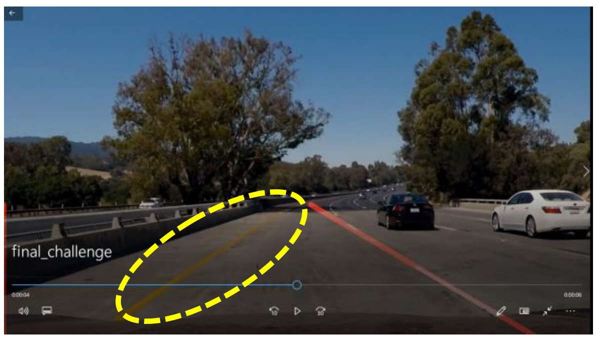


Resulting videos with challenge.mp4 (no extrapolate, average)

*: File names is **challenge.mp4**

(iii) Robustness against road color

The lane line in the left side disappeared when the road color was changed. Yellow lane line was difficult to detect because the road color was similar to yellow.



Final resulting videos with challenge.mp4

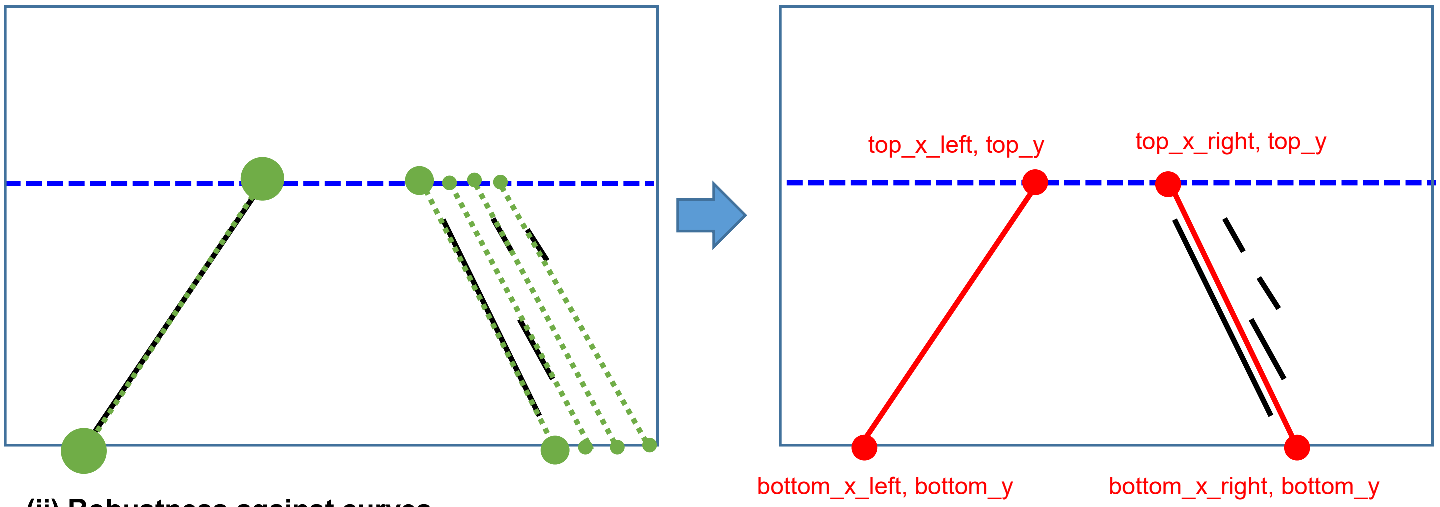
*: File names is **final_challenge.mp4**

(4) Suggestion of possible improvements to my pipeline

(i) Robustness against shades

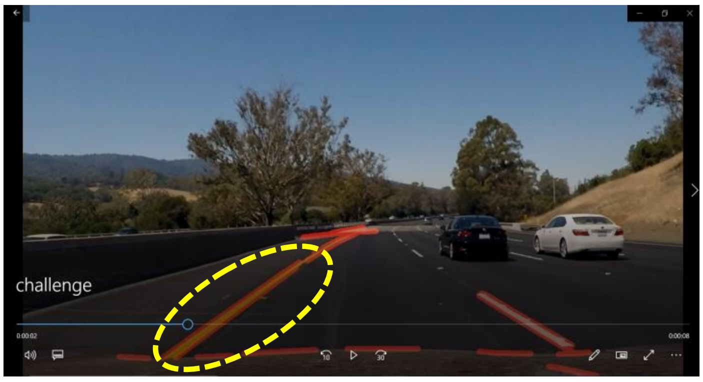
I simply calculated average of Hough Lines in page 4.

Usually real lane lines are longer than unwanted lines, so I think the robustness will be improved if I set bigger weights on the longer Hough lines when I calculate the average.



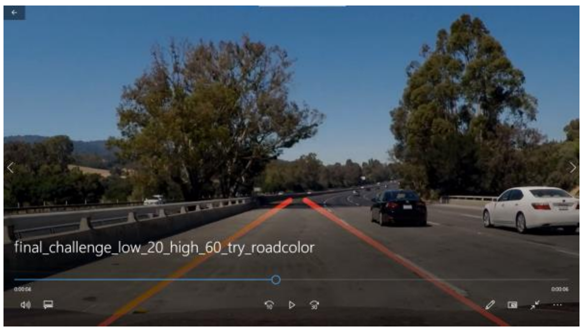
(ii) Robustness against curves

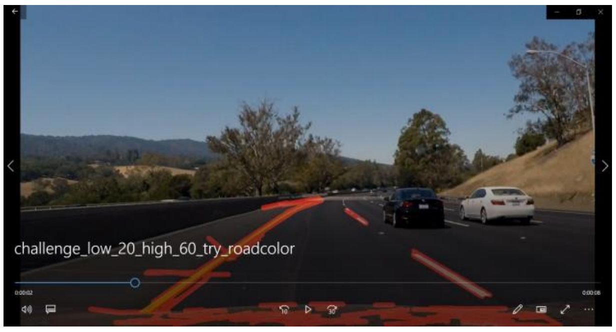
I'd better not to use straight line but use approximate curve by python function "polyfit" in numpy library because Hough lines match well with the real road. And also, we have to care about unwanted edge which matches with the outside lane same as (i). (Bellow figure is the same one as the previous page)



(iii) Robustness against road color

When I changed the parameters of Canny Edge to low=20, high=60, I was able to detect the left lane as below left movie. However, there are more unwanted Hough Lines on the usual road color area as below right movie. I think it's better to change the Canny Edge parameters dynamically according to the brightness of the current road color.





Parameter change video with challenge.mp4

*: File names is **final_challenge_low_20_high_60_try_roadcolor.mp4**

Parameter change video with challenge.mp4

*: File names is **challenge_low_20_high_60_try_roadcolor.mp4**