自動駕駛實務

Project_1

Lane Line Detection

電機所 碩一

廖聲宇

N26121143

1.Import Library

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
import math
from PIL import Image
from moviepy.editor import VideoFileClip
```

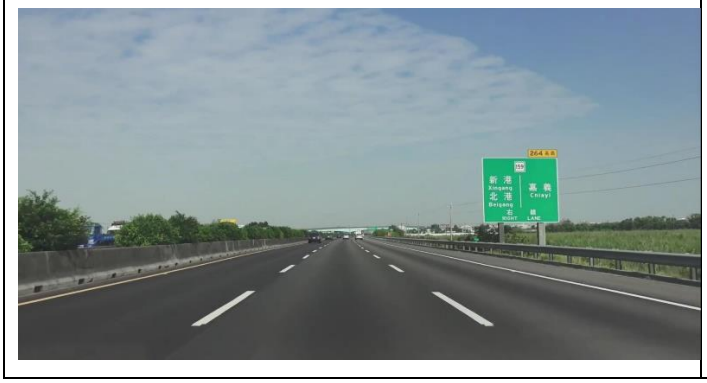
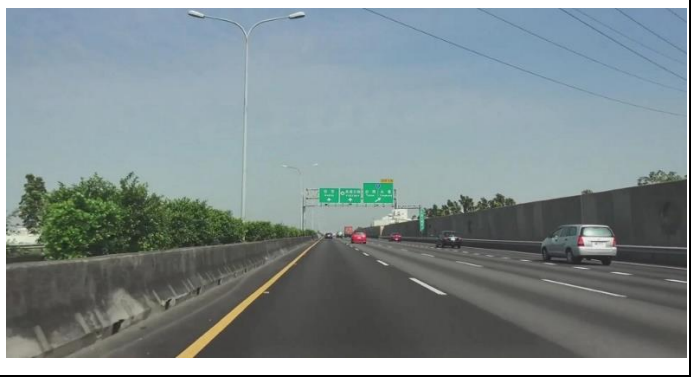
引入需要用到的函示庫

2.Define image process function and show image

```
def pipeline(image):
    ysize = image.shape[0]
    xsize = image.shape[1]

    def apply_smoothing(image, kernel_size = 3):
        return cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)

    def select_white_yellow(image):
        converted = convert_hls(image)
          # white color mask
          lower = np.uint8([  0, 160,   0])
          upper = np.uint8([255, 255, 255])
          white_mask = cv2.inRange(converted, lower, upper)
          # yellow color mask
          lower = np.uint8([  10,   0, 100])
          upper = np.uint8([  40, 255, 255])
          yellow_mask = cv2.inRange(converted, lower, upper)
          # combine the mask
          mask = cv2.bitwise_or(white_mask, yellow_mask)
          return cv2.bitwise_and(image, image, mask = mask)

    picking_white_yellow = select_white_yellow(image)

    gaussian_image = apply_smoothing(picking_white_yellow)
    # Call Canny Edge Detection here.
    cannyed_image = cv2.Canny(gaussian_image, 50, 220)
    plt.figure()
    plt.imshow(cannyed_image)
```

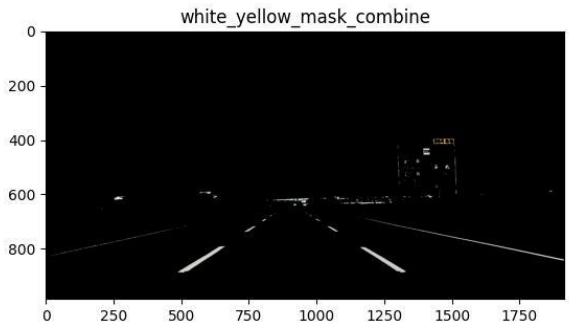
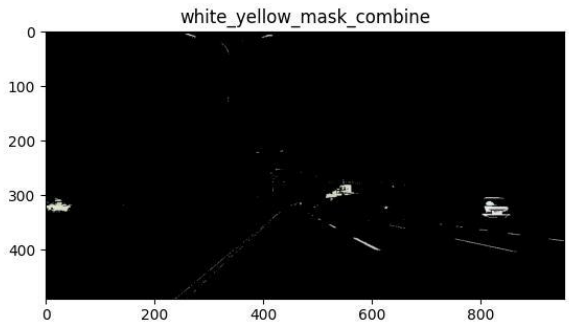
首先將需要的影像處理撰寫成 function，並建立一個 pipeline 包含所有 function，由第 2、3 行來獲得影像的尺寸，接著說明影像處理的步驟及函示。

以下測試圖片為 test.jpg/test02.jpg

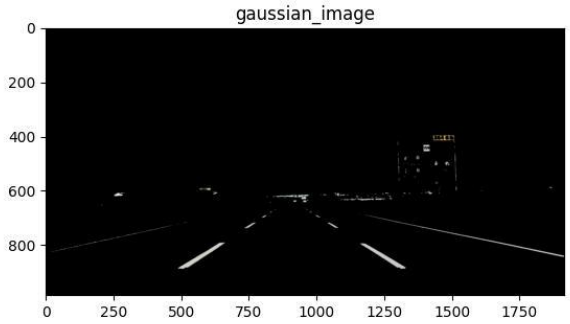
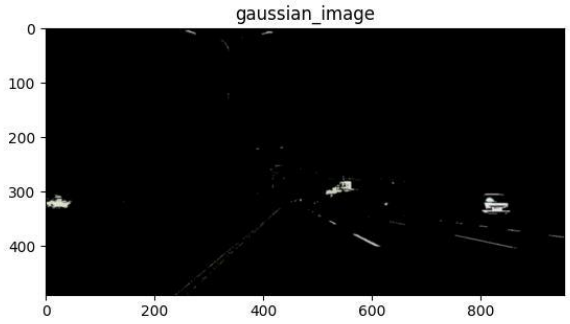
test.jpg	test02.jpg
	

第 5、6 行建立一個**高斯模糊的函示**，來調整影像中的噪點。

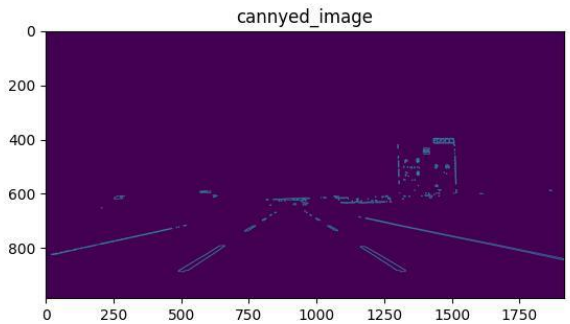
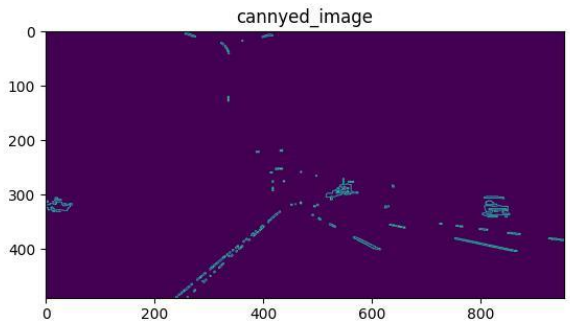
第 8~20 行建立一個**擷取出黃色/白色邊線的函示**，其中第 11~13 行為給定需要擷取的黃色上下限範圍，第 15~17 行為給定需要擷取的白色上下限範圍，針對只有白線的 test.jpg 僅設定白色範圍即可，接著由第 20 行執行將兩個顏色擷取結果合併成一張影像，結果如下圖。

test.jpg	test02.jpg
	

將上面的影像由第 24 行執行**高斯模糊**處理，可以得到下圖。

test. jpg	test02. jpg
	

將上面的影像由第 26 行執行**邊緣偵測**處理找出影像中的輪廓，可以得到下圖。

test. jpg	test02. jpg
	

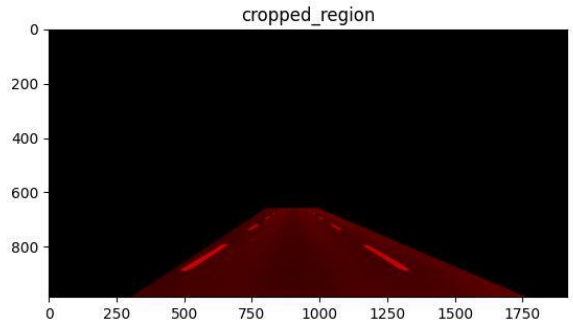
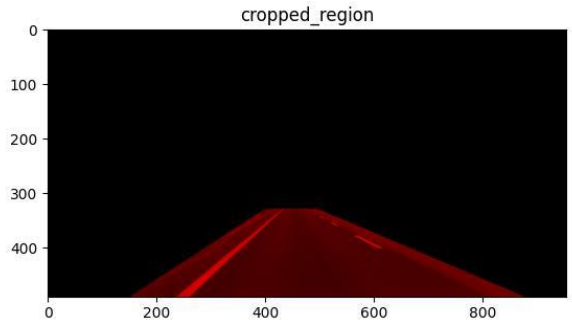
說明完影像處理的步驟及函示，接著說明影像想要**裁切(辨識)的區塊**，如此可以減少道路邊線以外的物體影響。

```
    def region_of_interest(img, vertices):
        mask = np.zeros_like(img)
        # channel_count = img.shape[]   * channel_count/下面的
        match_mask_color = (255,)
        cv2.fillPoly(mask, vertices, match_mask_color)
        masked_image = cv2.bitwise_and(img, mask)
        return masked_image
    #
    region_of_interest_vertices = [
         (xsize*0.16, ysize),
         (xsize*0.42, ysize*0.65),
         (xsize*0.52, ysize*0.65),
         (xsize*0.92, ysize),
      ]

```

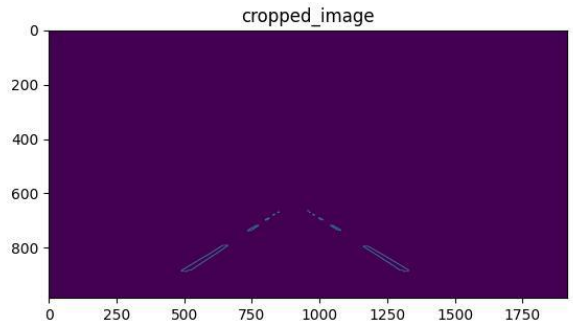
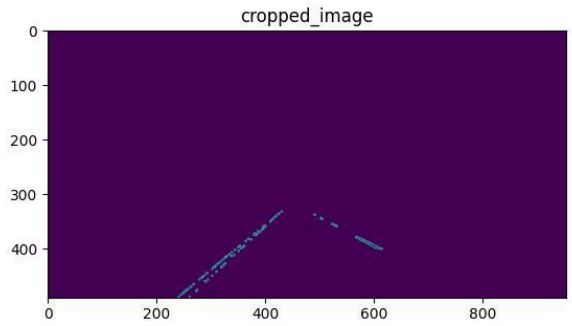
```
16.     cropped_image = region_of_interest(
17.         cannyed_image,
18.         np.array([region_of_interest_vertices], np.int32),)
19.     plt.figure()
20.     plt.imshow(cropped_image)
```

第 1~20 行為定義出想要**裁切的影像區塊**，給定四個頂點，裁出一個梯形區塊，並對上面邊緣偵測完的影像進行裁切，結果如下圖。

實際裁切的區域

test.jpg	test02.jpg
	

將邊緣偵測影像進行裁切的結果

test.jpg	test02.jpg
	

再來對上面的影像進行**霍夫轉換**處理，來找出車道邊線的線段。**霍夫轉換**(Hough Transform)是一種用於檢測幾何形狀的經典圖像處理技術，特別是用於檢測直線和圓形。

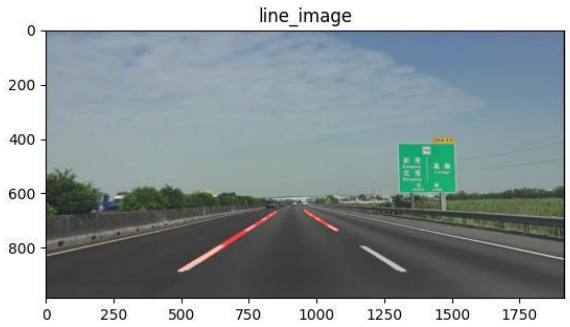
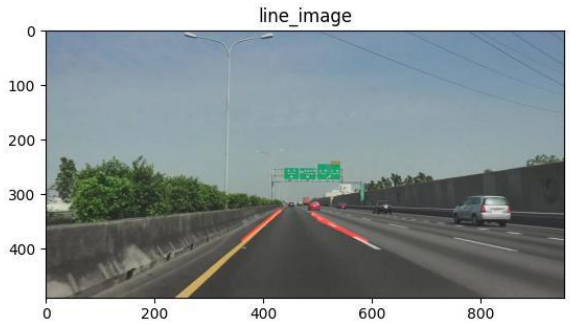
```
def draw_lines(img, lines, color=[255, 0, 0], thickness=6):
    # If there are no lines to draw, exit.
    if lines is None:
        return
    # Make a copy of the original image.
    img = np.copy(img)
    # Create a blank image that matches the original in size.
    line_img = np.zeros(
        (
            img.shape[0],
            img.shape[1],
            3
        ),
        dtype=np.uint8,
    )
    # Loop over all lines and draw them on the blank image.
    for line in lines:
        for x1, y1, x2, y2 in line:
            cv2.line(line_img, (x1, y1), (x2, y2), color, thickness)
    # Merge the image with the lines onto the original.
    img = cv2.addWeighted(img, 0.95, line_img, 1.0, 0.0)
    # Return the modified image.
    return img

#Hough
lines = cv2.HoughLinesP(
    cropped_image,
    rho=2,
    theta=np.pi / 180,
    threshold=10,
    lines=np.array([]),
    minLineLength=15,
    maxLineGap=300
)
line_image = draw_lines(image, lines)
plt.figure()
plt.imshow(line_image)
# plt.show()

left_line_x = []
left_line_y = []
right_line_x = []
right_line_y = []
for line in lines:
    for x1, y1, x2, y2 in line:
        slope = (y2 - y1) / (x2 - x1) # <-- Calculating the slope.
        if math.fabs(slope) < 0.5: # <-- Only consider extreme slope
            continue
        if slope <= 0: # <-- If the slope is negative, left group.
            left_line_x.extend([x1, x2])
            left_line_y.extend([y1, y2])
        else: # <-- Otherwise, right group.
            right_line_x.extend([x1, x2])
            right_line_y.extend([y1, y2])
```

```
55.     min_y = int(image.shape[0] * 3.25 / 5 )# <-- Just below the horizon
56.     max_y = image.shape[0] # <-- The bottom of the image
57.     poly_left = np.poly1d(np.polyfit(
58.           left_line_y,
59.           left_line_x,
60.           deg=1
61.     ))
62.     left_x_start = int(poly_left(max_y))
63.     left_x_end = int(poly_left(min_y))
64.     poly_right = np.poly1d(np.polyfit(
65.           right_line_y,
66.           right_line_x,
67.           deg=1
68.     ))
69.     right_x_start = int(poly_right(max_y))
70.     right_x_end = int(poly_right(min_y))
71.     line1_image = draw_lines(image,[[
72.                 [left_x_start, max_y, left_x_end, min_y],
73.                 [right_x_start, max_y, right_x_end, min_y]]], thickness=6,)
74.
75.     plt.figure()
76.     plt.imshow(draw_lines_image)
77.     plt.show()
78.     return draw_lines_image
```

第 1~78 行為在**影像畫出車道線**。

第 3~23 行，首先建立複製影像 img，透過霍夫轉換找出裁切影像上直線的集合，並在 line_img 上繪出檢測到的線條，最後將 line_img 與 img 疊合，即可得到繪有線條的影像，如下圖。

霍夫轉換後的結果

test. jpg	test02. jpg
	

第 26~34 行為**霍夫轉換的參數設置**，其中:

rho -累加器的距離解析度，以像素為單位。

theta -累加器的角度解析度，以弧度為單位。

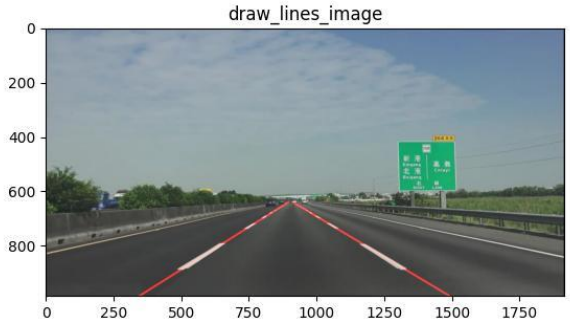
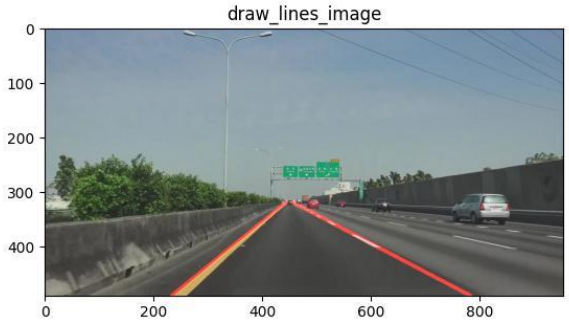
threshold -累加器的閾值參數。只有獲得足夠投票的直線（> 閾值）才會被返回。

minLineLength -最小線段長度。 小於此長度的線段將被拒絕。

maxLineGap -同一直線上兩點之間允許的最大間隙。

第 40~73 行為**擬合出車道線**，首先會檢測霍夫轉換得到的直線，對於每條直線先計算其斜率（slope），並檢查斜率的絕對值是否大於某個閾值（0.5），排除過於平緩的直線，只保留較陡的車道線。 根據斜率的正負將直線劃分為左右兩組，將各組的端點座標（x1，y1）和（x2，y2）新增到對應的清單中。 然後將左右兩組座標點進行多項式擬合，以估計車道線的位置，這裡使用了 np.polyfit 函數進行一次多項式擬合。擬合後得到車道線在圖像中的起始點和結束點，使用擬合後的多項式函數來計算車道線在影像底部和視野最遠處的位置。

最後對上面的影像**擬合出視野中的車道線**

test.jpg	test02.jpg
	

讀取影像及影片的部分

```
1.  #圖片
2.  white_output = 'test_output.jpg'
3.  clip1 = VideoFileClip("test.jpg")
4.  #white_output = 'test02_output.jpg'
5.  #clip1 = VideoFileClip("test02.jpg")
6.  #黃線
7.  # white_output = 'solidYellowLeft_output.mp4'
8.  # clip1 = VideoFileClip("solidYellowLeft.mp4")
9.  #白線
10.  # white_output = 'solidWhiteRight_output.mp4'
11.  # clip1 = VideoFileClip('solidWhiteRight.mp4')
12.  #挑戰
13.  # white_output = 'challenge_output.mp4'
14.  # clip1 = VideoFileClip("challenge.mp4")
15.  #國道
16.  # white_output = '/guodao_output.mp4'
17.  # clip1 = VideoFileClip("/guodao.mp4")
18.
19.  white_clip = clip1.fl_image(pipeline)
20.  white_clip.write_videofile(white_output, audio=False)
```

第 2~5 行為讀取影像的部分。

第 7~17 行為讀取影片的部分。

第 19、20 行為對影片中的每一幀進行 pipeline 函示的處理，並輸出成影片。

最終結果:

影片辨識結果	
白線	自駕車實務_solidWhiteRight output (youtube.com)
黃線	自駕車實務_solidYellowLeft output (youtube.com)
挑戰	自駕車實務_challenge output (youtube.com)
國道	自駕車實務_guodao output (youtube.com)