# 自動駕駛實務 #1 Report

| Department | Name | Student ID |
|---|---|---|
| 電機112C | 謝宗翰 | E14084078 |

- 車道辨識後成果影片
  1. Part I
     - 白線辨識：https://youtu.be/UPzABzYvC1c
     - 黃線辨識：https://youtu.be/MM14Qbq0S08
     - 挑戰題　：https://youtu.be/-PF1t4AVIgg
  2. Part II
     - 加分題　：https://youtu.be/FjSRAa72Ag4

- 遇到的困難

△ **問題 1. 版本衝突：**

因為剛接觸 OpenCV 等等影像辨識模組，所以在安裝上一直出問題。此次作業遇到最困難的問題是在安裝 OpenCV 跟 Moviepy 時，因為版本問題耗了我半天的時間，使用老師的方法安裝 OpenCV 需要在Python 3.5.*的環境下面安裝，但是接下來要安裝 Moviepy 的 Setup.py 時會需要在Python 3.7以上才可以安裝，兩者的安裝環境是互相衝突的。

✓ **解決辦法：**

最後是直接到 OpenCV 的官網下載開源程式碼，自己手動安裝到我自己 Python39 的資料夾內，略過了在Anaconda PowerShell 下環境限制問題。

△ **問題 2. Package 缺失：**

安裝完老師PPT上面所提供的Package後，去跑主程式檔(P1.ipynb)發現還是出現很多問題。ffmpeg與Moviepy在我所建置的環境下其實是不完整的，估計應該是版本之間的衝突。

✓ **解決辦法：**

把上網查到的所有關於 ffmpeg 跟 Moviepy 的Package都裝到我自己的

Python39資料夾內和 anaconda 建置出的環境，而非單單裝到建置的環境中。

△ **問題 3. 核心涵式不好寫：**

自己在寫 draw_lines 涵式時，一直因為資料型態而出現error。在寫影像處理時，常常因為矩陣大小不匹配而出現error，但這也是要磨練的地方。

△ **問題 4. 從網路上下載的影片跟 Test 的影片大小不一樣**

這遇到的問題應該每個人在寫挑戰題時都會遇到，我在調整各個參數時，都是針對 Test.mp4 的大小在調整(960*540)，但是我自己下載的challenge2.mp4的大小是1920*1080，導致偵測的Lane line不正確。

✓ **解決辦法：**

在寫涵式時增加更多參數，讓使用者可以自行控制變數增加。或用Image.shape去針對圖片大小來做Lane line的控制。

△ **問題5. 參數調整問題**

在下面「如何調整參數」的部分一併說明。

- 使用的工具

1. 環境工具與IDE：Jupyter notebook、anaconda
2. 涵式工具：NumPy, math, moviepy, IPython.display, matplotlib, cv2

- 如何調整參數

## 1. 高斯模糊與Canny演算法(影像邊緣)

Canny演算法跟高斯模糊需要放在一起調整，Canny 的 Threshold 越高影像銳畫的部分越少，反之而增多。而高斯模糊可以使影像的邊緣大大減少，能圖像比較明顯有邊緣的部分留下。影像邊緣的部分我就慢慢手動調到最佳化。

以下是我設定的參數(4組參數都差不多)，數字不要太誇張影響都不大。

```
# gaussian blur
blur_gray = gaussian_blur(gray, 5)
# canny edge detection
edges = canny(blur_gray, 30, 150)
```

## 2. 影像遮罩

不同的影片有不同的像素大小，如何找出好的範圍對於接下來的draw_line涵式有著非常很大的影響。我是用手動的方式找出梯形的最佳座標位置。

畫白線及畫黃線的遮罩參數設定(960*540)

```
# create region of interest
imshape = image.shape
vertices = np.array([[(50, imshape[0]), (460, 310), (490, 310),
                      (imshape[1]-50, imshape[0])]], dtype=np.int32)
masked_edges = region_of_interest(edges, vertices)
```

畫白線及畫黃線的影片相對簡單，車道上無其他雜物，所以就直接用範例給的大小作參考。

挑戰題的參數設定(1280*720)

```
# create region of interest
vertices = np.array([[(200,650), (1100, 650), (580, 420),
                      (720, 420)]], dtype=np.int32)  # 建立陣列
masked_edges = region_of_interest(edges, vertices)
# return masked_edges
```

影片的大小為1280*720，地上也有許多陰影，一開始在嘗試設定參數時遇到比較大的問題應該就是黃色的線經過灰階處理後很不明顯，再加上旁邊護欄的邊緣也很明顯，所以應該要畫在黃線上的線會被吸過去如下圖

後來發現有一大原因是遮罩也框到旁邊護欄的範圍所致。經手動設定參數把梯形的底邊縮小及不做灰階處理後就成功了。

| 加分題的參數設定(1920*1080) |
| --- |
| ```<br>imshape = image.shape<br>vertices = np.array([[(50,1000), (1500, 1000), (900, 620),<br>                    (1000, 620)]], dtype=np.int32)  # 建立陣列<br>masked_edges = region_of_interest(edges, vertices)<br>``` |
| 台灣高速公路的影片十分的清晰，一樣找出適合的梯形座標後就可以完美標出正確的線段。 |

3. Hough Lines 及 Draw Line 涵式

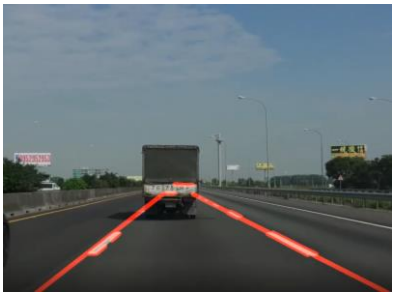
```python
# y_min 為線的最高高度
def draw_lines(img, lines, color=[255, 0, 0], thickness=12, y_min=330):
    # initialize lists to hold line formula values
    bLeftValues = []  # b of left lines
    bRightValues = []  # b of Right lines
    mPositiveValues = []  # m of Left lines
    mNegitiveValues = []  # m of Right lines

    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(img, (x1, y1), (x2, y2), color, thickness)
                # calculate slope and intercept
                m = (y2-y1)/(x2-x1)
                b = y1 - x1*m
                # threshold to check for outliers
                if m >= 0 and (m < 0.2 or m > 0.8):
                    continue
                elif m < 0 and (m < -0.8 or m > -0.2):
                    continue

                # seperate positive line and negative line slopes
                if m >= 0:
                    mPositiveValues.append(m)
                    bLeftValues.append(b)
                else:
                    mNegitiveValues.append(m)
                    bRightValues.append(b)
```

```
# Get image shape and define y region of interest value
imshape = img.shape
y_max = imshape[0]  # lines initial point at bottom of image

# Get the mean of all the lines values
AvgPositiveM = mean(mPositiveValues)
AvgNegitiveM = mean(mNegitiveValues)
AvgLeftB = mean(bLeftValues)
AvgRightB = mean(bRightValues)

# use average slopes to generate line using ROI endpoints

global x1_Left, y1_Left, x2_Left, y2_Left, x1_Right, y1_Right, x2_Right, y2_Right
if AvgPositiveM != 0:
    x1_Left = (y_max - AvgLeftB)/AvgPositiveM
    y1_Left = y_max
    x2_Left = (y_min - AvgLeftB)/AvgPositiveM
    y2_Left = y_min
if AvgNegitiveM != 0:
    x1_Right = (y_max - AvgRightB)/AvgNegitiveM
    y1_Right = y_max
    x2_Right = (y_min - AvgRightB)/AvgNegitiveM
    y2_Right = y_min

    # define average left and right lines
    cv2.line(img, (int(x1_Left), int(y1_Left)), (int(x2_Left),
             int(y2_Left)), color, thickness)  # avg Left Line
    cv2.line(img, (int(x1_Right), int(y1_Right)), (int(x2_Right),
             int(y2_Right)), color, thickness)  # avg Right Line
```

(Code解說放在註解)

整個Code的邏輯就是用傳入經過霍夫轉換的線(input 為 lines)，然後剔除不合理的部分，如斜率界在±0.2~ ± 0.8之間的線段才採納，找到點之後就將其用OpenCV的涵式cv2.line去將線段畫出來。值得注意的是，我有將線段的最大值做了限制，以便套用在各種大小影片或圖片上，下面會再更進一步說明。

最難調整的部分應該是跟把線畫出來有直接相關的Hough Line跟 Draw Line 方程式，Hough Line 的 Threshold 會大大影響線的精確程度，如果太小，會辨識到周



圍不相干的物體，例如：切換車道的車或前面車子的車體等等，如下圖的情況。這部分也是需要手動去調整。而 Draw Line 部分的寫法參考了網路上的做法，也考慮了線的長度，限制線長設定在涵式裡面，需要根據影像大小去做設定。例如把960*540的線長套用進1920*1080的影像中會過長等等。

基本上，只要上面遮罩做得好的話就可以很成功的把線畫出來。換句話說，Drawline 涵式寫完沒出問題，遮罩的參數設定應該就是這次作業最關鍵的部分。

## 4. Process_image() 及 Challenge_image()

A) Process_image() (實作SolidWhite 和 SolidYellow)

```
def process_image(image):
    # NOTE: The output you return should be a color image (3 channel) for processing video below
    # TODO: put your pipeline here,
    # you should return the fMinal output (image where lines are drawn on lanes)
    # grayscale the image
    gray = grayscale(image)

    # gaussian blur
    blur_gray = gaussian_blur(gray, 5)
    # canny edge detection
    edges = canny(blur_gray, 30, 150)

    # create region of interest
    imshape = image.shape
    vertices = np.array([[(50, imshape[0]), (460, 310), (490, 310),
                          (imshape[1]-50, imshape[0])]], dtype=np.int32)  # 建立陣列
    masked_edges = region_of_interest(edges, vertices)

    # hough tranform
    line_image = hough_lines(masked_edges, 1, np.pi/180, 22, 18, 1, 330)

    # draw lines
    lines_edges = weighted_img(line_image, image)

    # return masked_edges
    return lines_edges

    # return result
```

這是影像處理的總涵式：

灰階 → 高斯模糊 → 影像锐畫→ 遮罩→ 霍夫轉換→ 畫線

參數設定可參考前四點所述

B) Challenge_image() (實作SolidWhite 和 SolidYellow)

```
def challenge_image(image):
    # NOTE: The output you return should be a color image (3 channel) for processing video below
    # TODO: put your pipeline here,
    # you should return the fMinal output (image where lines are drawn on lanes)
    # grayscale the image
    # gray = grayscale(image)

    # gaussian blur
    blur_gray = gaussian_blur(image,15)
    # canny edge detection
    edges = canny(blur_gray, 30, 150) #canny(img, low_threshold, high_threshold)
    # plt.imshow(edges)

    # create region of interest
    vertices = np.array([[(200,650), (1100, 650), (580, 420),
                        (720, 420)]], dtype=np.int32)  # 建立陣列

    vertices = np.array([[(50, imshape[0]), (460, 310), (490, 310),
                        (imshape[1]-50, imshape[0])]], dtype=np.int32)  # 建立陣列

    masked_edges = region_of_interest(edges, vertices)
    # return masked_edges

    # hough tranform
    # hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    line_image = hough_lines(masked_edges, 0.8, np.pi/180, 70, 22, 1,460)
    plt.imshow(line_image)
    # return line_image
    # draw lines
    lines_edges = weighted_img(line_image, image)

    return lines_edges

    # return result
```

這是影像處理的總涵式，邏輯與Process_image()差不多，只是參數的不同，但值得注意的是，此涵式沒有加入灰階處理，原因是這會讓挑戰題的黃線辨識度下降，所以剔除。

參數設定可參考前四點所述

- 心得與討論

在做這份作業時，遇到比較大的問題就是物件的型態問題，有時Package提供的涵式 return 結果都跟想的不太一樣，需要自己去trace資料型態，但相信這應該都是剛接觸新套件時會遇到的問題。

自己實作出加分題跟draw_line方程式其實還滿有成就感的，花在debug的時間應該超過30小時，主要應該因為對於 Jupyter notebook 跟 OpenCV 套件的不熟悉，希望下次影像處理作業能夠更上手。

在做挑戰題時，黃色線會貼過去柵欄的問題卡了我一段時間。我原本的作法是先把高斯模糊調大，讓旁邊柵欄的線變得很不明線，但是相對的，要偵測的車道線也會變得很模糊，本末倒置，來來回回也是調了很多參數，最後是Return edge的輸出結果才發現，黃線的edge在某些路段會因為顏色太淺而消失，而旁邊

柵欄的edge會變得很明顯，最後才發現是因為灰階跟遮罩沒有調好才導致車道辨識失敗。

寫到最後會發現，其實只要自己手動把遮罩遮好，就能處理9成的問題，感覺是有點偷吃步的，因為在現實生活中，要辨識的大小會常常變來變去的，例如挑戰題的柵欄，車距大小改變，攝相機位置等等，都會影響需要做遮罩的大小。接下來的作業是Advance Finding Lane line ，利用梯形區域來做判斷，好像也可以利用相同的做法解決，但我給自己一個期許，不用手動標的方式去取得該遮罩的位置參數，希望可以成功。

我的GitHub連結：Lucashien/Finding-Lane-Line (github.com)