```python
import cv2 as cv
import numpy as np

img1 = cv.imread('1.jpg')
img2_ori = cv.imread('2.jpg')
img2 = cv.resize(img2_ori,(1640,590))
img3 = cv.imread('3.jpg')
```

首先，一開始按照第一次作業所學的，讀取照片，而第 5、6 行的程式碼是我在修改照片 2.jpg 的大小，因為如果沒有調整大小，2.jpg 本身太大，會導致 cv2.imshow()的時候只會跑出一小部分，如下圖，因此我將其改成跟 1.jpg 大小一樣的圖片，讓其可以正常顯示

```python
gray_1 = cv.cvtColor(img1,cv.COLOR_BGR2GRAY)
gray_2 = cv.cvtColor(img2,cv.COLOR_BGR2GRAY)
gray_3 = cv.cvtColor(img3,cv.COLOR_BGR2GRAY)
cv.imwrite('gray_1.jpg',gray_1)
cv.imwrite('gray_2.jpg',gray_2)
cv.imwrite('gray_3.jpg',gray_3)

filter_1 = cv.blur(gray_1, (5, 5))
filter_2 = cv.blur(gray_2, (5, 5))
filter_3 = cv.blur(gray_3, (5, 5))
cv.imwrite('filter_1.jpg',filter_1)
cv.imwrite('filter_2.jpg',filter_2)
cv.imwrite('filter_3.jpg',filter_3)
```

這邊跟作業一的步驟一樣，將原圖轉成灰階再進行濾波。

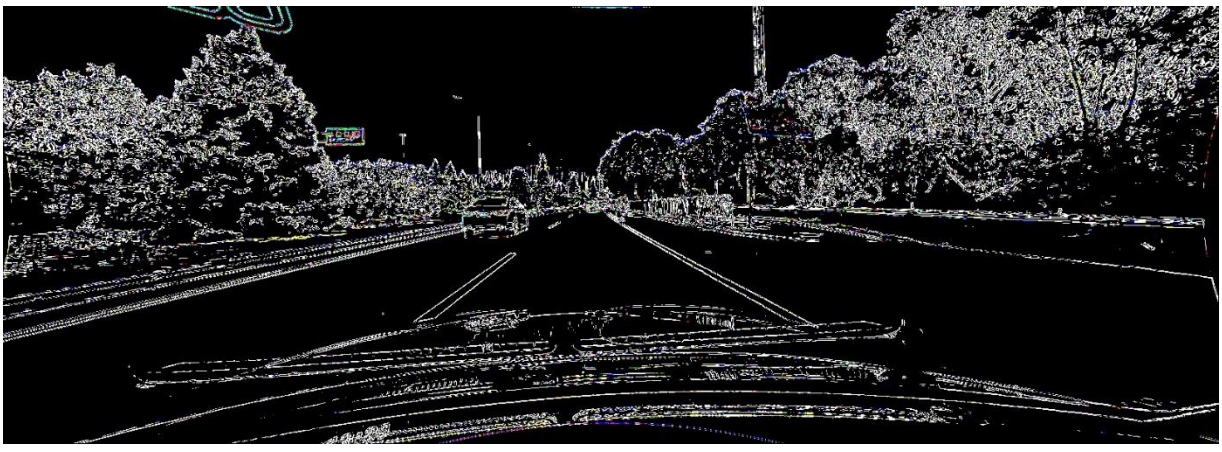
```
29    sobelX_1 = cv.Sobel(img1,cv.CV_64F,1,0)
30    sobelY_1 = cv.Sobel(img1,cv.CV_64F,0,1)
31    sobelX_2 = cv.Sobel(img2,cv.CV_64F,1,0)
32    sobelY_2 = cv.Sobel(img2,cv.CV_64F,0,1)
33    sobelX_3 = cv.Sobel(img3,cv.CV_64F,1,0)
34    sobelY_3 = cv.Sobel(img3,cv.CV_64F,0,1)
35
36    sobelX_1 = np.uint8(np.absolute(sobelX_1))
37    sobelY_1 = np.uint8(np.absolute(sobelY_1))
38    sobelX_2 = np.uint8(np.absolute(sobelX_2))
39    sobelY_2 = np.uint8(np.absolute(sobelY_2))
40    sobelX_3 = np.uint8(np.absolute(sobelX_3))
41    sobelY_3 = np.uint8(np.absolute(sobelY_3))
42
43    sobelCombined_1 = cv.bitwise_or(sobelX_1,sobelY_1)
44    sobelCombined_2 = cv.bitwise_or(sobelX_2,sobelY_2)
45    sobelCombined_3 = cv.bitwise_or(sobelX_3,sobelY_3)
```

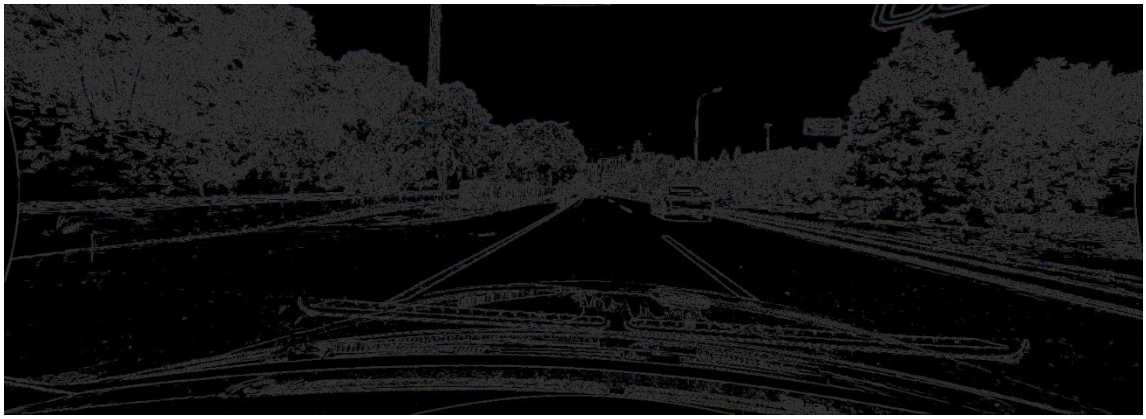
在做邊緣檢測的時候，我選擇使用 cv2.Sobel()的方式，首先 29-34 行是在對圖形的 X、Y 軸分別進行邊緣抓取，36-41 則是在計算邊緣梯度值，43-45 則是讓其針對 XY 軸進行邊緣抓取。但是這個方式我覺得他的邊緣檢測不是很好，因此我改用 canny

```
31    canny1 = cv.Canny(filter_1,120,150)
32    canny2 = cv.Canny(filter_2,120,150)
33    canny3 = cv.Canny(filter_3,120,150)
```

```
70    ret, bin_1 = cv.threshold(sobelCombined_1,127,255,cv.THRESH_BINARY)
71    ret, bin_1test = cv.threshold(sobelCombined_1,50,50,cv.THRESH_BINARY)
72    ret, bin_2 = cv.threshold(sobelCombined_2,127,255,cv.THRESH_BINARY)
73    ret, bin_3 = cv.threshold(sobelCombined_3,127,255,cv.THRESH_BINARY)
74    cv.imwrite('bin_1.jpg',bin_1)
75    cv.imwrite('bin_1test.jpg',bin_1test)
76    cv.imwrite('bin_2.jpg',bin_2)
77    cv.imwrite('bin_3.jpg',bin_3)
```

這邊進行二值化，其中 71 是我在用於觀察不同的分類閥值、最大值所形成的圖片是什麼，因為像素的灰度值如果小於閥值 = 前景(白色)，反之大於閥值 = 背景(黑色)，下列兩張圖分別為 70、71 程式碼所生成的圖形，用以分別不同閥值所造成的差異，最後我選擇是使用 70 行所設定的閥值。



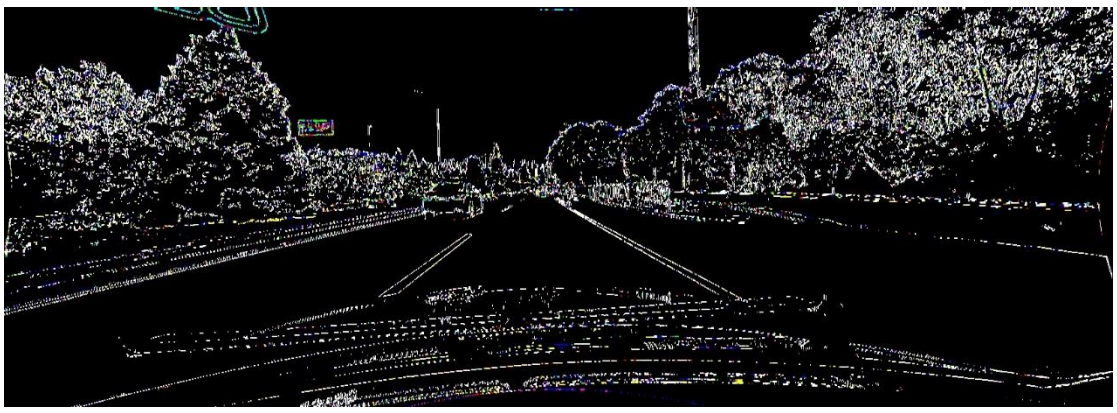


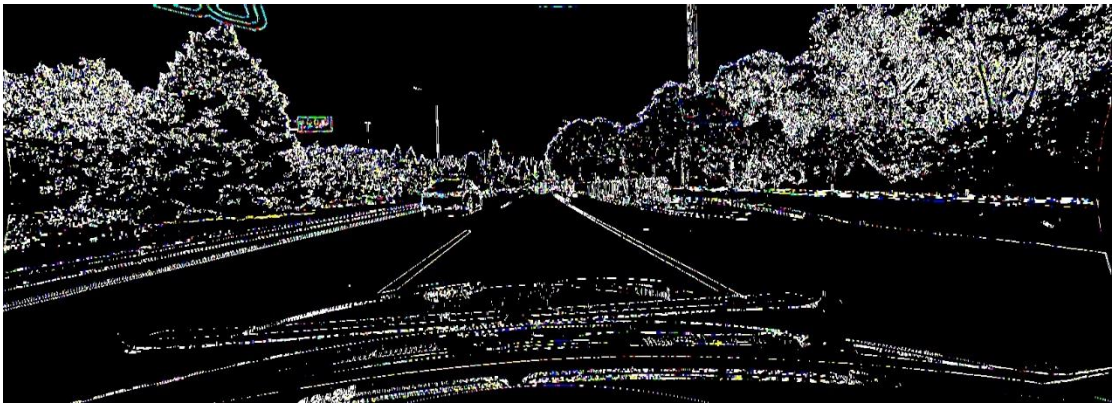
```python
# morphology_1_smallkernalopen = cv.morphologyEx(bin_1,cv.MORPH_OPEN,(1,1))          測試kernal大小的影響
# morphology_1_largekernalopen = cv.morphologyEx(bin_1,cv.MORPH_OPEN,(500,500))
# cv.imshow('test1',morphology_1_smallkernalopen)
# cv.waitKey(0)
# cv.imshow('test2',morphology_1_largekernalopen)
# cv.waitKey(0)

morphology_1open = cv.morphologyEx(bin_1,cv.MORPH_OPEN,(10,10))
morphology_open_close = cv.morphologyEx(morphology_1open,cv.MORPH_CLOSE,(10,10))
# open =cv.morphologyEx(morphology_1open,cv.MORPH_OPEN,(10,10))
morphology_1close = cv.morphologyEx(bin_1,cv.MORPH_CLOSE,(10,10))
morphology_close_open = cv.morphologyEx(morphology_1close,cv.MORPH_OPEN,(10,10))
cv.imwrite('open_then_close.jpg',morphology_open_close)
cv.imwrite('close_then_open.jpg',morphology_close_open)
# cv.imshow('test',open)                                  觀察進行多次開運算的結果
# cv.waitKey(0)
# cv.imshow('test1',morphology_open_close)                通過觀察開運算以及閉運算的先後，選擇較適合的
# cv.waitKey(0)
# cv.imshow('test2',morphology_close_open)
# cv.waitKey(0)
```

```python
mor11 = cv.morphologyEx(bin_1,cv.MORPH_OPEN,(10,10))
mor22 = cv.morphologyEx(bin_2,cv.MORPH_OPEN,(10,10))
mor33 = cv.morphologyEx(bin_3,cv.MORPH_OPEN,(10,10))
mor1 = cv.morphologyEx(mor11,cv.MORPH_OPEN,(10,10))
mor2 = cv.morphologyEx(mor22,cv.MORPH_OPEN,(10,10))
mor3 = cv.morphologyEx(mor33,cv.MORPH_OPEN,(10,10))
cv.imwrite('mor1.jpg',mor1)
cv.imwrite('mor2.jpg',mor2)
cv.imwrite('mor3.jpg',mor3)
```

這邊因為不同的 kernal 值會影響所產生的圖片，因此我分別測試了不同的 kernal 值所產生的圖片，但我覺得沒什麼差別，最後就一律使用(10,10)

接著我分別生成兩張圖片，分別按照順序進行，開運算-閉運算、閉運算-開運算，所產生的圖，因為開運算是去除圖像中的小對象，而閉運算是填充小空洞，使空隙增大。下列分別是先進行開運算以及先進行閉運算的圖片





因為我認為先進行開運算讓其去除一些小對象，像是平滑車道邊緣、分開相連的車道，再用閉運算填補一些小空間

```
121  cimg1 = np.copy(img1)
122  cimg2 = np.copy(img2)
123  cimg3 = np.copy(img3)
124
125  lines = cv.HoughLinesP(mor1,1,np.pi/180,45,minLineLength=50,maxLineGap=4)
126  for line in lines:
127      x1, y1, x2, y2 = line[0]
128      cv.line(cimg1, (x1, y1), (x2, y2), (0, 255, 0), 2)
129  cv.imshow('test',cimg1)
130  cv.imwrite('cimg1.jpg',cimg1)
131  cv.waitKey(0)
132
133  lines = cv.HoughLinesP(mor2,1,np.pi/180,40,minLineLength=140,maxLineGap=190)
134  for line in lines:
135      x1, y1, x2, y2 = line[0]
136      cv.line(cimg2, (x1, y1), (x2, y2), (0, 255, 0), 2)
137  cv.imshow('test',cimg2)
138  cv.imwrite('cimg2.jpg',cimg2)
139  cv.waitKey(0)
140
141  lines = cv.HoughLinesP(mor3,1,np.pi/180,40,minLineLength=140,maxLineGap=190)
142  for line in lines:
143      x1, y1, x2, y2 = line[0]
144      cv.line(cimg3, (x1, y1), (x2, y2), (0, 255, 0), 2)
145  cv.imshow('test',cimg3)
146  cv.imwrite('cimg2.jpg',cimg2)
147  cv.waitKey(0)
```

121-123 複製一份新的原圖，之後就是利用 HoughLinesP, 裡面內容依序為，毒入圖片、像素單位的精度、可能搜尋的角度、閥值、接受的最小長度直線、線段間最小間隔，再來 128 中最後兩個參數一個是第一次作業的顏色，另一個是線條粗度