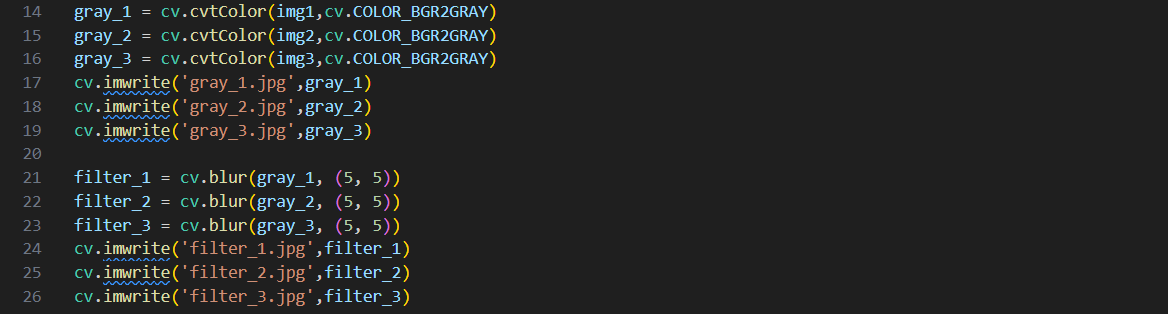
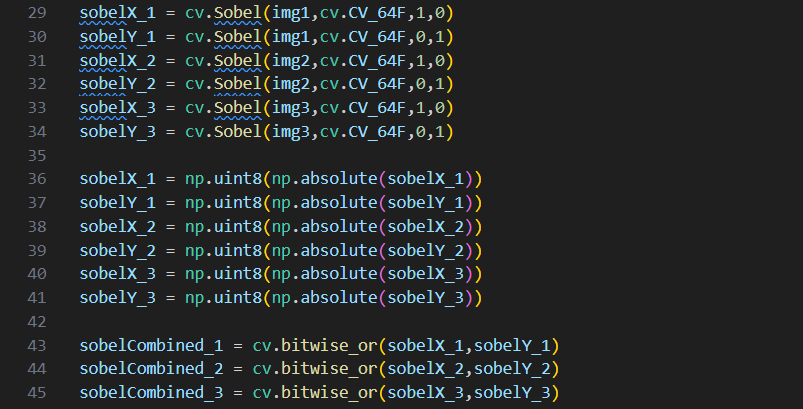
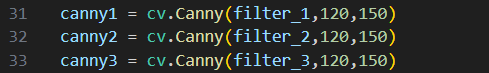


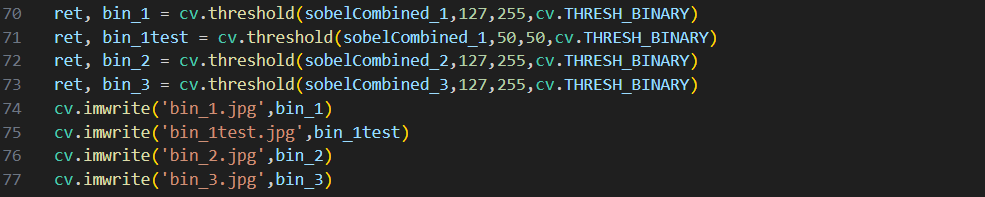
首先，一開始按照第一次作業所學的，讀取照片，而第5、6行的程式碼是我在修改照片2.jpg的大小，因為如果沒有調整大小，2.jpg本身太大，會導致cv2.imshow()的時候只會跑出一小部分，如下圖，因此我將其改成跟1.jpg大小一樣的圖片，讓其可以正常顯示

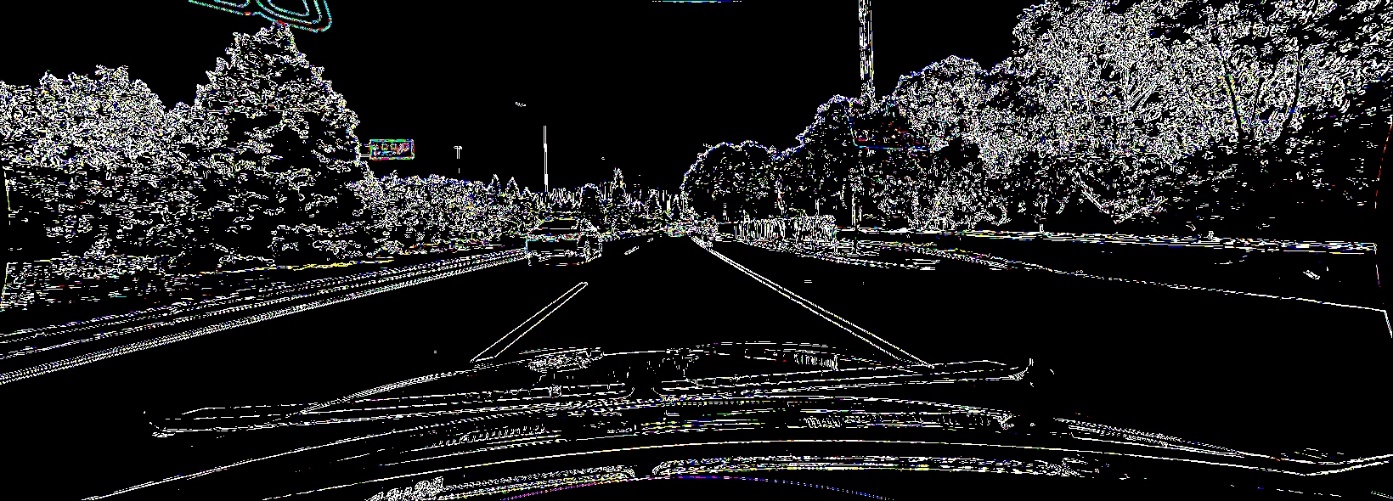
這邊跟作業一的步驟一樣，將原圖轉成灰階再進行濾波。



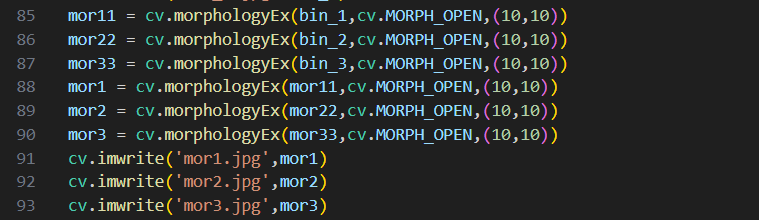
在做邊緣檢測的時候，我選擇使用cv2.Sobel()的方式，首先29-34行是在對圖形的X、Y軸分別進行邊緣抓取，36-41則是在計算邊緣梯度值，43-45則是讓其針對XY軸進行邊緣抓取。但是這個方式我覺得他的邊緣檢測不是很好，因此我改用canny





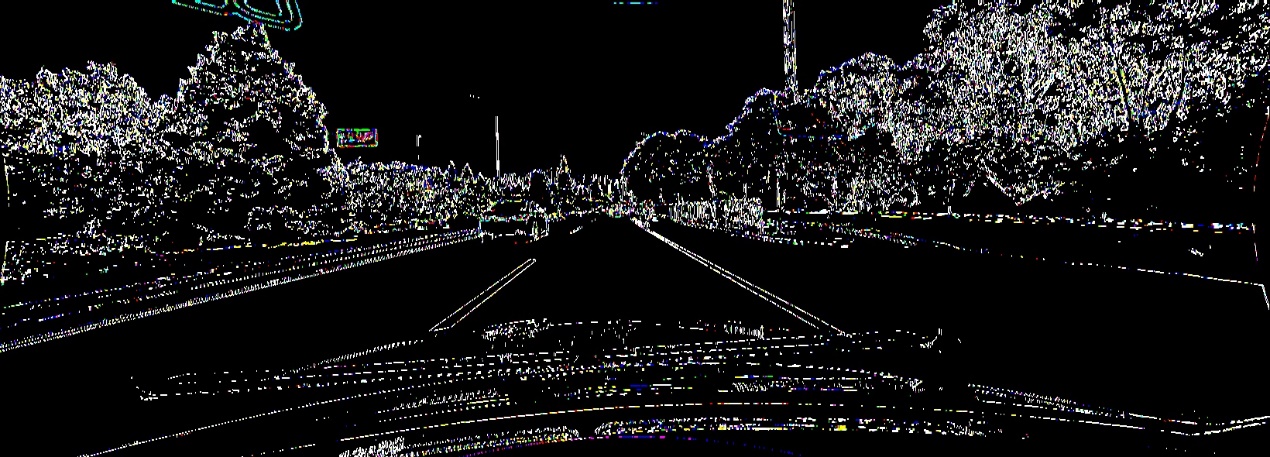
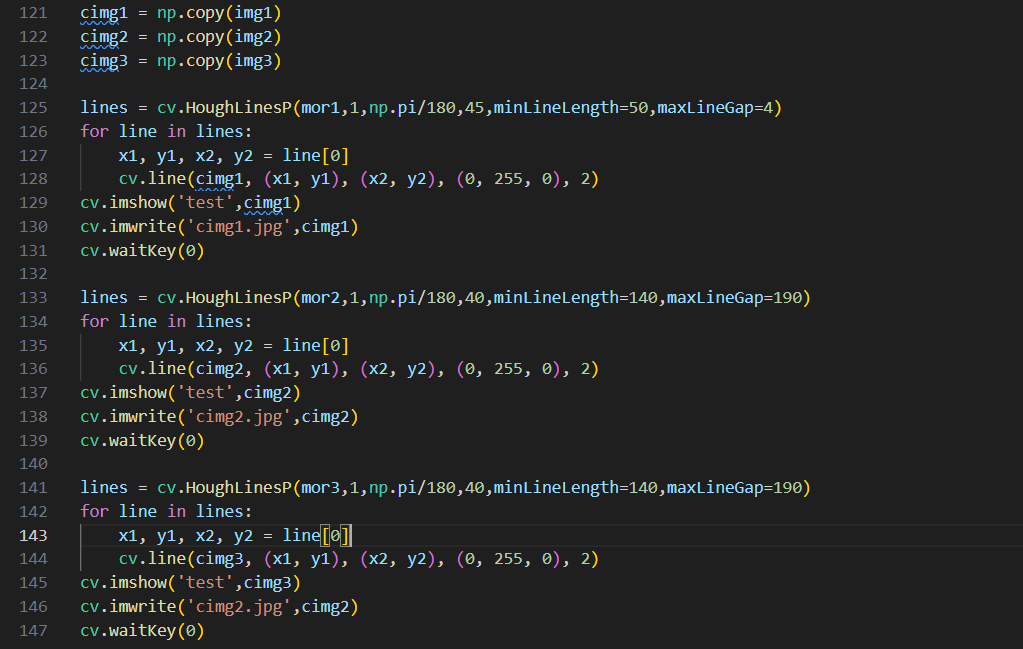
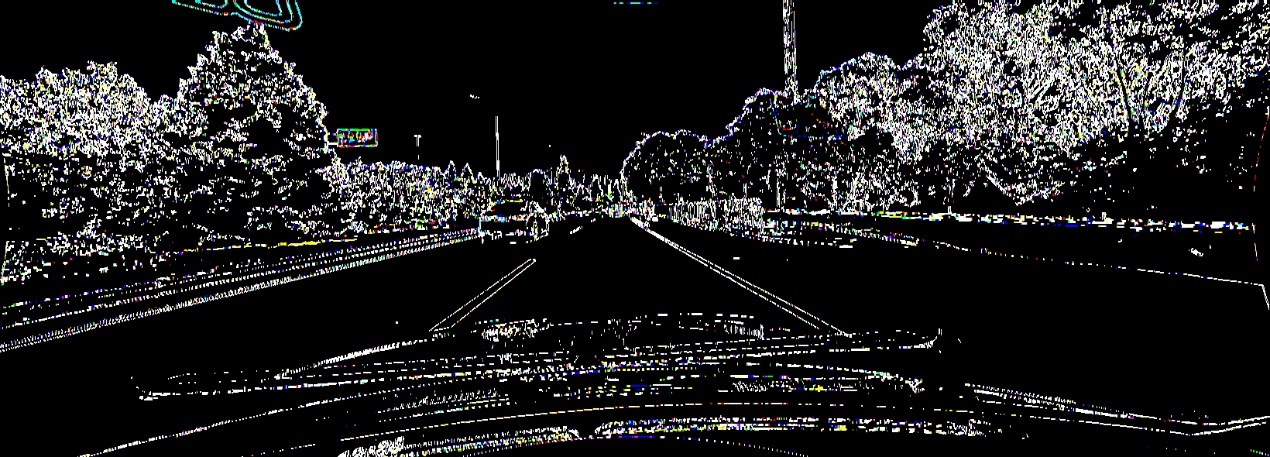
這邊進行二值化，其中71是我在用於觀察不同的分類閥值、最大值所形成的圖片是什麼，因為像素的灰度值如果小於閥值 = 前景(白色)，反之大於閥值 = 背景(黑色)，下列兩張圖分別為70、71程式碼所生成的圖形，用以分別不同閥值所造成的差異，最後我選擇是使用70行所設定的閥值。





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  
  
121-123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