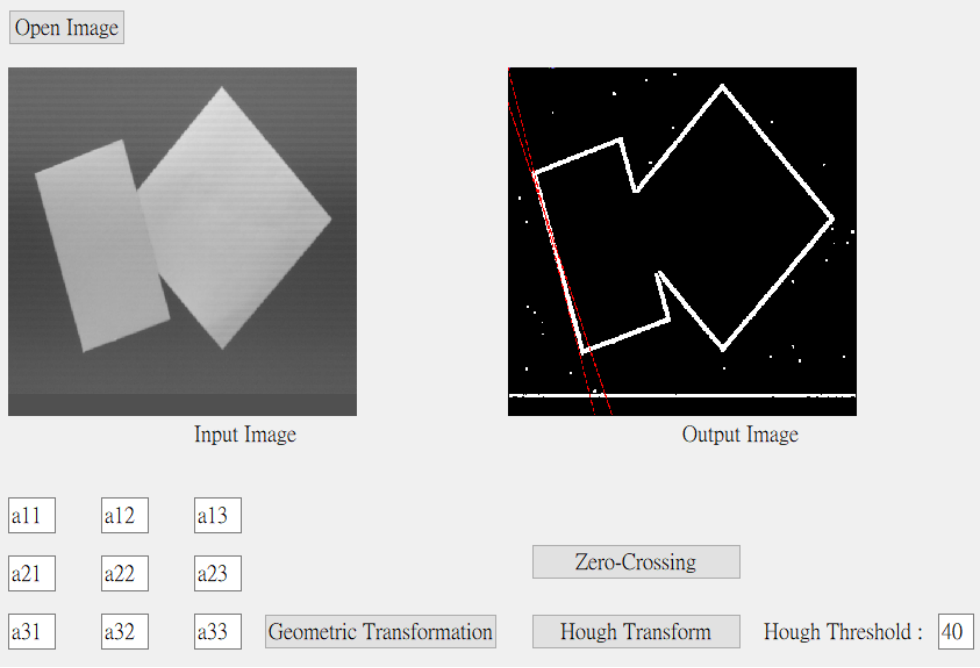
Digital Image Processing HW06 Report

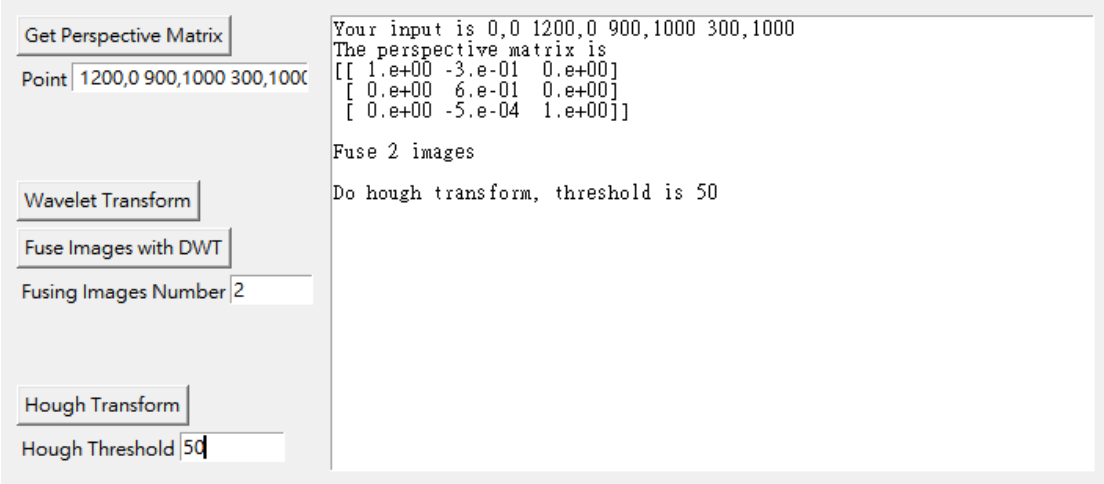
生機碩一 R09631007 吳乙澤

一、軟體簡介

由於我使用 C++ 寫該作業時，第二題小波轉換沒寫出來，第三題霍夫變換寫得不夠完整，因此我額外用到 Python 來撰寫本作業。 C++ 的演算法皆是我自己手刻的， Python 則有用到 pywt、OpenCV 兩個影像處理函數庫。見下圖，兩種語言我都有做出使用者介面。首先，C++的部分，需先開啟圖片後，才能做後續的動作。見左下方的九個輸入框，這是透視變換的矩陣，使用者可自行輸入係數，並按下「Geometric Transformation」按鈕，來達成各種幾何變換。介面的右方則是霍夫變換，需先按下「Zero-Crossing」找完邊緣後，在「Hough Threshold」輸入閥值並按下「Hough Transform」，稍待片刻，軟體便會在圖中以紅色顯示出直線。另外， Python 的部分不須先開啟圖片，按下各按鈕便會自動跳出圖片選擇框。須注意「Get Perspective Matrix」、「Fuse Images with DWT」、「Hough Transform」等按鈕，需先輸入相對應的值後，才能起作用。「Get Perspective Matrix」搭配本作業的第一題，可以找出透視變換的矩陣係數。找出後，將係數輸入 C++ 介面的九個輸入框，便能完成透視變換。值得注意的是，「Point」須以「數字,數字 數字,數字 數字,數字 數字,數字」的格式輸入四組座標點，如1,1 2,2 3,3 4,4。「Fusing Images Number」也只能輸入 2 或 3，輸入其他的數字會出錯。



軟體介面圖。該圖是 C++ 配合 QT



軟體介面圖，該圖是 Python 配合 Tkinter

二、演算法說明與結果討論

本作業有三個功能需完成。以下依序說明這些功能的演算法並討論結果。

2-1 Geometric Transformation

參照課本公式 2-45 以及網路(https://zhuanlan.zhihu.com/p/36082864)的資料，寫出一個可以讓使用者自訂轉換矩陣之參數介面。見下方 C++ 程式碼，a11、a12、a13...是使用者輸入的係數，根據透視變換公式，便能將這些係數與原座標相乘得到新座標。最後，再以該新座標設定新圖片的像素即可。

```
// Do geometric transform
for (int i = 0 ; i < imgCols ; i++)
   for (int j = 0 ; j < imgRows ; j++)
      pixelB = MatImgIn.at<Vec3b>(j,i)[0];
      pixelG = MatImgIn.at<Vec3b>(j,i)[1];
      pixelR = MatImgIn.at<Vec3b>(j,i)[2];
      perspectI = abs(i*a11) + abs(j*a12) + abs(a13);
      perspectJ = abs(i*a21) + abs(j*a22) + abs(a23);
      perspectK = abs(i*a31) + abs(j*a32) + abs(a33);
      perspectI /= perspectK;
      perspectJ /= perspectK;
      QImgOut.setPixel(int(perspectI), int(perspectJ), qRgb(pixelR, pixelG,
pixelB));
```

另外，為了快速求解透視變換矩陣的係數，我有以 Python 程式語言和 getPerspectiveTransform 函數獲得變換矩陣係數。見下方程式碼，需對使用者輸入座標點做些前處理後，才能丟到 getPerspectiveTransform 算出矩陣結果。

```
pointStr = pointStr.split()
for i in range(4):
    point = [int(pointStr[i].split(',')[0]), int(pointStr[i].split(',')
    [1])]
    pointList.append(point)

dst = np.float32(pointList)

matrix = cv2.getPerspectiveTransform(dst, imgPoint)
```

見下方結果圖，左上為原圖，右上為範例圖，中間為我的 C++ 程式繪出的圖，最下為 Python 求得的透視變換矩陣。由於我有對輸出結果做座標點限縮，因此跑出的圖片和範例圖不太一樣。但本質上都是 Trapezoidal Transformation。

Trapezoidal Transformation

其他如 Wavy Transformation、Circular Transformation 等變換的係數我沒有求出。但是，由 Trapezoidal Transformation 的結果來看，我第一題的演算法應該是沒有問題的。另外，若固定 a31、a32 為零，a33 為一，則該矩陣能做到 Affine Transformation 的效果。見下圖所示，一個是 shear，另一個是 rotation。

Shear Transformation (a11 = 1, a12 = 0.5, a13 = 0, a21 = 0, a22 = 1, a23 = 0)

Rotation Transformation (a11 = 0, a12 = -1, a13 = 0, a21 = 1, a22 = 0, a23 = 0)

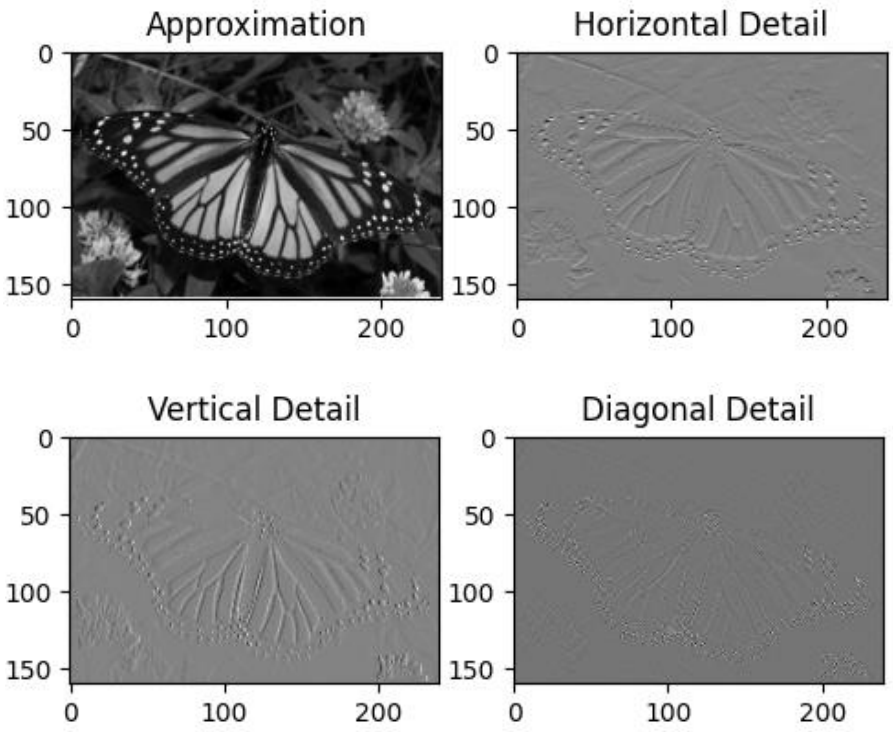
2-2 Image Fusion Using Wavelet Transform

該題只有 Python 部分的程式碼。如下方所示，我以 dwt2 函數對圖片做小波轉換，找到該圖的approximation、horizontal detail、vertical detail、diagonal detail 等四種資訊。圖片融合的部分，我先利用 wavedec2 函數對圖片做小波轉換找到係數後，再以 np.maximum 搭配 for 迴圈，找出不同圖片間係數較大的部分後，將之 append 到 fuseList 中。最後，以 waverec2 函數做逆小波轉換得到原圖。

```
# Wavelet Transform
cA, (cH, cV, cD) = pywt.dwt2(imgGray_part2, 'haar')
```

```
# Fuse Images with Wavelet Transform
coeffs  = pywt.wavedec2(imgGray_part2[:,:], 'haar')
if fuseNum == 2:
    for i in range(len(coeffsList[0])-1):
        if(i == 0):
            c = np.maximum(coeffsList[0][0], coeffsList[1][0])
            fuseList.append(c)
            continue
        c1 = np.maximum(coeffsList[0][i][0], coeffsList[1][i][0])
        c2 = np.maximum(coeffsList[0][i][1], coeffsList[1][i][1])
        c3 = np.maximum(coeffsList[0][i][2], coeffsList[1][i][2])
        c  = (c1, c2, c3)
            fuseList.append(c
elif fuseNum == 3:
...

fuseImg = pywt.waverec2(fuseList, 'haar')
```

見下方結果圖，這是以 dwt2 函數做圖的結果。可以從 horizontal detail、vertical detail、diagonal detail 的圖形中看到不同的直線趨勢。



小波轉換結果圖

見下方，這是以小波轉換做圖片融合的結果圖。原本的圖片中，一張是背景模糊、鬧鐘清晰，另一張則是背景清晰、鬧鐘模糊。通過圖片融合，背景中的 3M 字樣、鬧鐘數字的清晰度皆比原來模糊的部分好。但是經過融合後，圖片的解析度也下降了，細看的話，圖片像是由一個一個小方塊組合成的，推測這是小波轉換的特性所造成的。

Image Set 1 (左方兩張為原圖，最右張是融合結果)

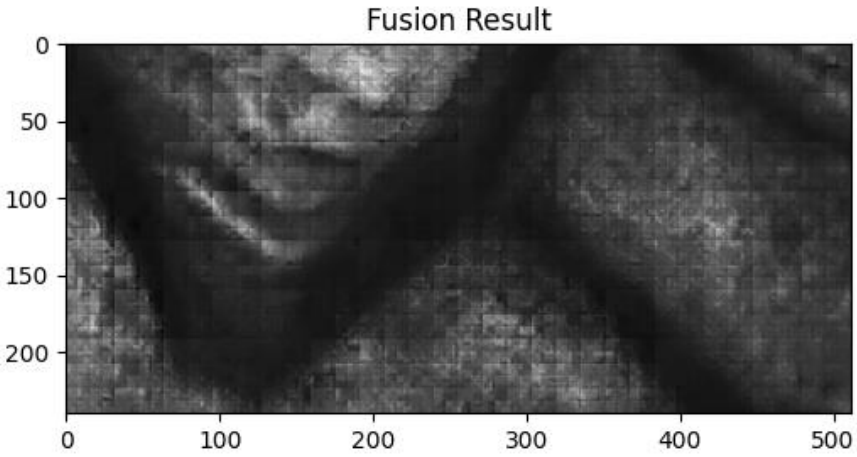


Image Set 2 (融合結果)

2-3 Hough Transform

該題 C++ 部分寫的不夠完整，我沒有把空間轉到角座標系統。見下方程式碼，先利用第三次作業的 Zero-Crossing 程式碼找完邊緣，再針對邊緣做潛在直線的投票，使相對應的一維矩陣 forHough[k] 之 k 值處加一。投完票後，根據閥值，選出比閥值大的部分做圖，即可找到直線。

```
// Vote
for (int i = 1 ; i < imgCols-maskSize ; i++)
   for (int j = 1 ; j < imgRows-maskSize ; j++)
      if ((MatForZero.at<double>(i - 1, j) * MatForZero.at<double>(i + 1, j)
<= 0) || (MatForZero.at<double>(i, j + 1) * MatForZero.at<double>(i, j - 1)
```

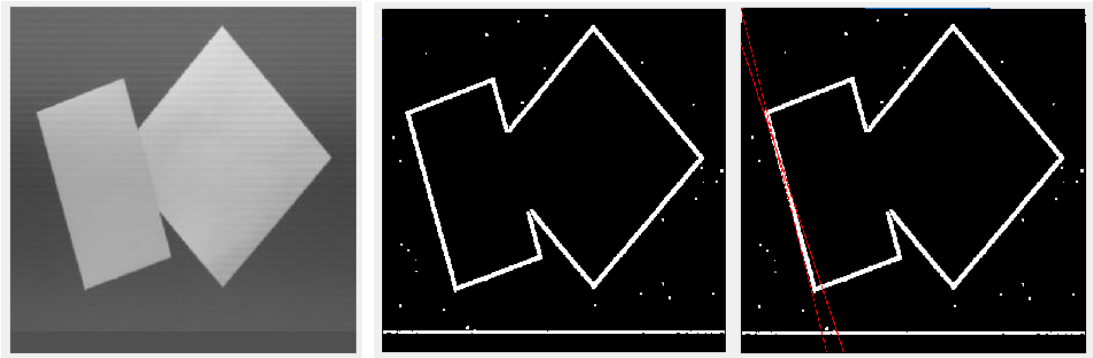
```
<= 0) || (MatForZero.at<double>(i + 1, j - 1) * MatForZero.at<double>(i - 1,
j + 1) <= 0) || (MatForZero.at<double>(i - 1, j - 1) * MatForZero.at <double>
(i + 1, j + 1) <= 0))
        for (int k = 0 ; k < (imgCols-maskSize)*(imgRows-maskSize); k++)
        {
           houghI = k - k/(imgCols-maskSize)*imgCols;
           houghJ = k/(imgCols-maskSize);
           if (houghI*i + houghJ == j)
              forHough[k]++;
        }

// Choose the one bigger than threshold
for (int i = 0; i < (imgCols-maskSize)*(imgRows-maskSize) ; i++)
   if (forHough[i] > threshold)
   {
      cout << forHough[i] << endl;
      line[index] = i;
      index ++;
   }
   if (forHough[i] > max)
      max = forHough[i];
```

見下方所示，這是 Python 部分的程式碼，我用到 OpenCV 的 Canny 和 HoughLines 函數。以 HoughLines 函數得到 imgLine_part3 後，使用 for 迴圈取出對應資料並將直線畫在原圖上，即可完成該題。

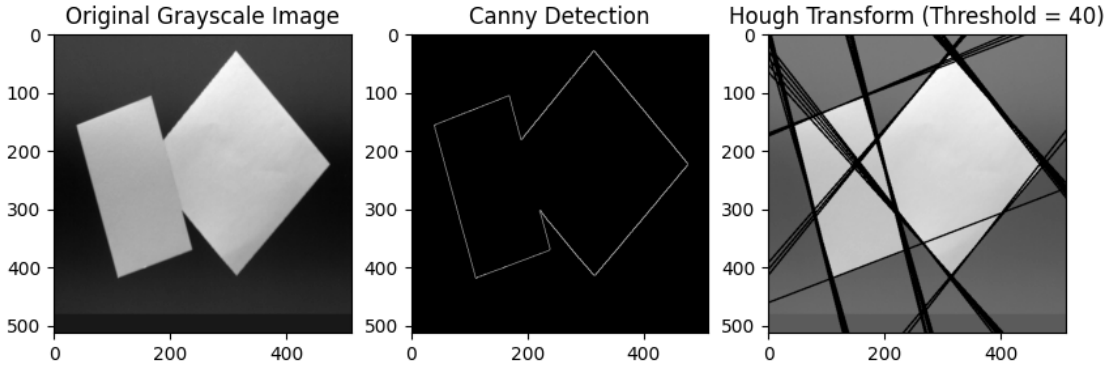
```
# Hough transform
imgEdge_part3 = cv2.Canny(imgGray_part3, threshold1 = 50, threshold2 =
150, apertureSize = 3)
imgLine_part3 = cv2.HoughLines(imgEdge_part3, rho = 1, theta = np.pi/18
0, threshold = threshold)
```

見下面結果圖，C++ 和 Python 在相同的 threshold 下，有不同的輸出結果。猜測這是我自己手刻的演算法不夠完整所導致的，或是我 C++ 的 Zero-Crossing 方法不夠好。比較 Zero-Crossing 和 Canny 對該張圖片的濾波結果，可明顯地看出 Canny 幾乎沒有雜點，找出的邊緣也比 Zero-Crossing 來的細，更沒有 Zero-Crossing 圖的水平直線。另外，Python 版本的 threshold，經過幾次測試，我發現本圖的最佳值為 80 左右。在該 threshold 下，兩個矩形共八條邊皆有被找出，且

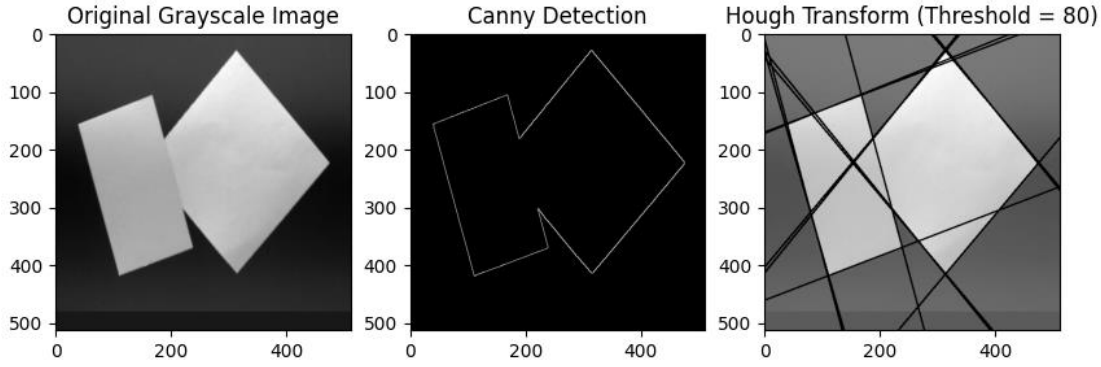
沒有其他的雜線產生。



C++ 的霍夫變換結果，threshold = 40



Python 的霍夫變換結果，threshold = 40



Python 的霍夫變換結果，threshold = 80