HW1_透視變形校正 Perspective Distortion Correction

一、透視變形

透視變形是由相機與目標物體之間的距離與拍攝角度所造成，當相機與目標物體之間的距離有所變化時，會造成影像發生彎曲或變形，造成結果與所期待的相左，就是所謂的透視變形。

二、如何校正

在數學中，透視變形能透過以下聯立方程式表示：

$$x = ax' + by' + cx'y' + d$$
$$y = ex' + fy' + gx'y' + h$$

其中x, y為原始影像的座標，x'、y'為校正後的座標，a、b、c、d、e、f、g、h為常數，表示變形的關係。

透過影像變形區域的四個角點代入聯立方程式，就可以計算出 a、h 共八個參數並得到變形關係。

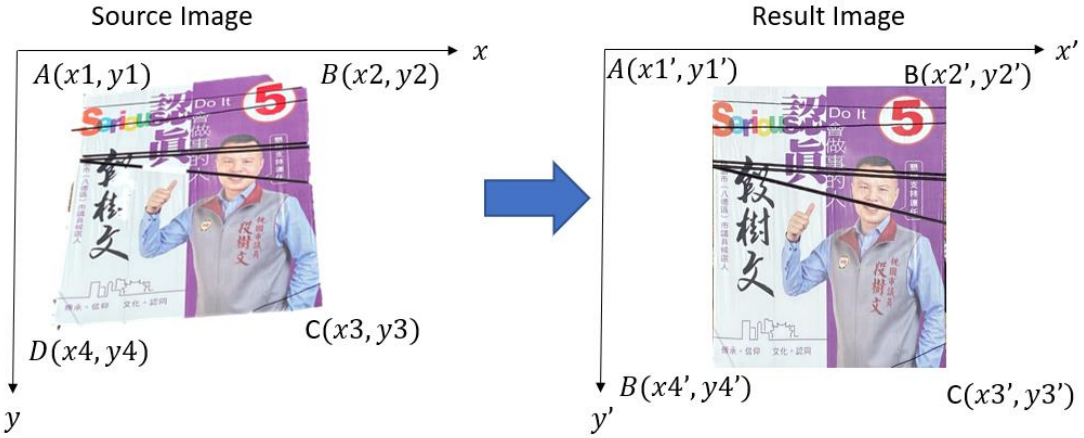

圖 一　座標點關係圖

三、如何計算

可以透過解多元一次方程組求出八參數，八個方程式八個未知數，必定有解。

可以將八參數寫成以下矩陣

$$\begin{bmatrix} x1' & y1' & x1'y1' & 1 & 0 & 0 & 0 & 0 \\ x2' & y2' & x2'y2' & 1 & 0 & 0 & 0 & 0 \\ x3' & y3' & x3'y3' & 1 & 0 & 0 & 0 & 0 \\ x4' & y4' & x4'y4' & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x1' & y1' & x1'y1' & 1 \\ 0 & 0 & 0 & 0 & x2' & y2' & x2'y2' & 1 \\ 0 & 0 & 0 & 0 & x3' & y3' & x3'y3' & 1 \\ 0 & 0 & 0 & 0 & x4' & y4' & x4'y4' & 1 \end{bmatrix} X \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \\ y1 \\ y2 \\ y3 \\ y4 \end{bmatrix}$$

$$A \qquad\qquad x \quad = \quad b$$

計算出 a、b、c、d、e、f、g、h 後，就能套用 Invert mapping +bilinear

Interpolation 將變形區的每一個點校正過去新的 image。

Invert mapping + bilinear Interpolation：

$$f(x,y) = (1-a)(1-b)img[l,k] + a(1-b)img[l+1,k] + (1-a)b * img[l,k+1] + ab * img[l+1,k+1]$$

四、程式實作

Step1：讀取圖片

首先設定圖片的路徑，透過 search_file 函數將路徑內所有的圖片讀入，將圖片

檔案名稱存進 image 陣列內

```
img_path = './'
image = search_file(img_path)
if 'desktop' in image:image.remove('desktop')
```

圖 二　　讀取圖片程式碼

```
def search_file(path): #return all file name
    file=[]
    for i in os.listdir(path):
        if (i[-4:]=='.jpg' or i[-4:]=='.JPG' or i[-4:]=='.png') and i[:-4] not in file:
            i = i[:-4]
            file.append(i)
    return file
```

圖 三　　搜尋檔案函數內容

此時陣列內容與資料夾內容顯示如下：

```
['IMG_5308', 'IMG_5309', 'IMG_5310']
```

圖 四　　資料夾圖片名稱

圖 五　　資料夾內的圖片

就能使用迴圈將每張圖片讀入並做處理。

Step2：圖片前處理

進入迴圈後，使用者可以先設定預設的圖片大小，如本程式預設期望大小為 1200*1000，再使用 cv2.imread 讀取圖片 img，並使用 cv2.resize 將圖片縮小成原本的 3 成，供使用者方便點選四個角點，否則圖片過大會超出螢幕範圍造成不方便點選。

再來會將 img 使用 copy 的方式給予 img2，以免修改的時候動到原圖(img)，再使用 cv2.imshow 將 img2 展示出來，並將其與 show_xy 涵式傳入 setMouseCallback，供使用者可以點擊圖片的四個角點。

show_xy 函數會先判斷使用者是否點擊(if event ==1)，若使用者點擊，則會 print 出該點的 x,y 座標，並將 x,y 座標加入進 point 陣列內，代表此座標點為四角點之一，在點擊的同時會呼叫 cv2.circle 將使用者所點擊的點畫上紅圈，供使用者查看，當使用者點滿四個點，就將 img 關掉，並開始計算八參數。

```
def show_xy(event,x,y,flags,param):
    global point, img2
    if event ==1:
        print(x,y)
        point.append([x,y])
        img_circle = cv2.circle(img2, (x,y), 10, (0,0,225), -1)
        cv2.imshow("img",img2)
    if len(point)==4:
        cv2.destroyAllWindows()
```

圖 六　　show_xy 函數

圖 七　　圖片點擊後顯示紅點示意圖

Step3：計算八參數

當使用者點選完四個角點後，會將 1.圖片 2.期望大小 3.四個角點 傳入

calculate 函數中，此函數中會先判斷四個角點的座標值，並將四個角點重新排序為[左上、右上、右下、左下]之順序，以免產出的效果不如預期，排序原理如下圖：

```
def calculate(img , expect_size ,point):
    right_up = []
    right_down = []
    left_up = []
    left_down = []
    x_max = 0
    x_max_index=0
    x_second=0
    x_second_index = 0
    for i in range(len(point)):
        if point[i][0]>x_max:
            x_max=point[i][0]
            x_max_index = i
    for i in range(len(point)):
        if point[i][0]>x_second and i!=x_max_index:
            x_second = point[i][0]
            x_second_index = i
    if point[x_max_index][1]>point[x_second_index][1]:
        right_down=(point[x_max_index])
        right_up=(point[x_second_index])
    else:
        right_up=(point[x_max_index])
        right_down=(point[x_second_index])
    point.remove(right_up)
    point.remove(right_down)
    if point[0][1] < point[1][1]:
        left_down=point[1]
        left_up = point[0]
    else:
        left_down=point[0]
        left_up = point[1]
    point = [left_up,right_up,right_down,left_down]
```

圖 八　　防呆機制

程式首先判斷 x 的最大值與其 index，再來判斷第二大的 x 值與其 index，藉此可以判斷這兩個座標點為右邊，再來判斷這兩個座標點的 y 值，y 值較大的為右下角，反之則為右上角。

之後將這兩個已經確定為右上及右下角的座標點從陣列 point 中移除，此時 point 陣列剩下兩個座標點，我們只需要再比較 y 值，若 y 值較大的則為左

下，反之為左上，當左上、右上、右下、左下皆確定後，再將這四個點依序放進 point 陣列中，以此達到防呆的效果，因此使用者點選的順序不會造成程式出現錯誤。

待四個座標點順序確定後，即可開始計算八參數，首先我建立代表我們所預期的圖片的四個座標(x1_,y1_)、(x2_,y2_)、(x3_,y3_)、(x4_,y4_)

並由程式自動依據使用者所期望的圖片大小建立前述的八參數矩陣(此範例 A 矩陣為期望大小 1200*1000 所建立出的矩陣，b 矩陣為使用者所點選的四個角點)：

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1200 & 1000 & 1200000 & 1 & 0 & 0 & 0 & 0 \\ 1200 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1000 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1200 & 1000 & 1200000 & 1 \\ 0 & 0 & 0 & 0 & 1200 & 0 & 0 & 1 \end{bmatrix} X \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} 419 \\ 845 \\ 864 \\ 347 \\ 340 \\ 306 \\ 550 \\ 574 \end{bmatrix}$$

$$A \quad x = b$$

有了 A 矩陣與 b 矩陣確切的值後，即可透過 np.linalg.solve 套件替我們計算出 x 的值，此時 x 的值為$[-6.00000000e-02, 4.26000000e-01, 7.58333333e-05\ \ 4.19000000e+02, 1.95000000e-01, -3.40000000e-02, 8.33333333e-06, 3.40000000e+02]$

即可將 x 值回傳至主程式，程式碼如下：

```
x1_ = y1_ = x2_ = y4_ = 0
x4_ , x3_ , y2_ ,y3_ = expect_size[0] , expect_size[0] , expect_size[1] ,expect_size[1]
A = np.array([(x1_,y1_,x1_*y1_,1,0,0,0,0),(x2_,y2_,x2_*y2_,1,0,0,0,0),(x3_,y3_,x3_*y3_,1,0,0,0,0),
              (x4_,y4_,x4_*y4_,1,0,0,0,0),(0,0,0,0,x1_,y1_,x1_*y1_,1),(0,0,0,0,x2_,y2_,x2_*y2_,1),
              (0,0,0,0,x3_,y3_,x3_*y3_,1),(0,0,0,0,x4_,y4_,x4_*y4_,1)])
b = np.array([(point[0][0]),(point[1][0]),(point[2][0]),(point[3][0]),(point[0][1]),(point[1][1]),\
              (point[2][1]),(point[3][1])])
x = np.linalg.solve(A, b)
return x
```

圖 九　　計算八參數

Step4：Invert mapping + bilinear Interpolation

前項步驟所算的 x 回傳至主程式後，會建立一個全為 0 的圖片 output_img，大小為我們所期望的大小，也就是 1200*1000，建立後會將我們所算出來的 x、原圖 img 及 output_img 傳入 inverse_mapping 函數，此函數會透過兩個迴圈將 output_img 的長寬跑一輪，並且將每個 output_img 的座標點依據以下公式將原圖 img 的座標點像素值傳入 output_img 相對應的座標點

$$f(x,y) = (1-a)(1-b)img[l,k] + a(1-b)img[l+1,k] + (1-a)b * img[l,k+1] + ab * img[l+1,k+1]$$

待 output_img 全部的座標點都有像素值後，會使用 cv2.imshow 將透視變形校正後的圖片 show 出來給使用者查看。

程式碼如下：

```
def inverse_mapping(b,input_img,output_img):
    print(input_img.shape , output_img.shape)
    row ,col = output_img.shape[:2]
    for i in range(row):
        for j in range(col):
            d_y = ((b[0]*i) + (b[1]*j) + (b[2]*i*j) + b[3])
            d_x = ((b[4]*i) + (b[5]*j) + (b[6]*i*j) + b[7])
            y , x = int(d_y) , int(d_x)
            u , v = d_y-y , d_x-x
            for c in range(3):
                output_img[i,j,c] = (1-u) * (1-v) * input_img[x,y][c] + u *(1-v) * input_img[x,y+1][c] + \
                v * (1-u) * input_img[x+1,y][c] + u*v*input_img[x+1,y+1][c]
    cv2.imshow("img",output_img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

圖 十　　inverse_mapping 函數內容

最終結果如下圖：

圖 十一　原始圖片點選四個角點

圖 十二　透視變形校正後結果

整體程式碼如下：

```
import os
import cv2
import numpy as np
def search_file(path): #return all file name
    file=[]
    for i in os.listdir(path):
        if (i[-4:]=='.jpg' or i[-4:]=='.JPG' or i[-4:]=='.png') and i[:-4] not in file:
            i = i[:-4]
            file.append(i)
    return file

def show_xy(event,x,y,flags,param):
    global point, img2
    if event ==1:
        print(x,y)
        point.append([x,y])
        img_circle = cv2.circle(img2, (x,y), 10, (0,0,225), -1)
        cv2.imshow("img",img2)
    if len(point)==4:
        cv2.destroyAllWindows()

def calculate(img , expect_size ,point):
    right_up = []
    right_down = []
    left_up = []
    left_down = []
    x_max = 0
    x_max_index=0
    x_second=0
    x_second_index = 0
    for i in range(len(point)):
        if point[i][0]>x_max:
            x_max=point[i][0]
            x_max_index = i
    for i in range(len(point)):
        if point[i][0]>x_second and i!=x_max_index:
            x_second = point[i][0]
            x_second_index = i
    if point[x_max_index][1]>point[x_second_index][1]:
        right_down=(point[x_max_index])
        right_up=(point[x_second_index])
    else:
        right_up=(point[x_max_index])
        right_down=(point[x_second_index])
    point.remove(right_up)
    point.remove(right_down)
    if point[0][1] < point[1][1]:
        left_down=point[1]
        left_up = point[0]
    else:
        left_down=point[0]
        left_up = point[1]
    point = [left_up,right_up,right_down,left_down]
    x1_ = y1_ = x2_ = y4_ = 0
    x4_ , x3_ , y2_ ,y3_ = expect_size[0] , expect_size[0] , expect_size[1] ,expect_size[1]
    A = np.array([(x1_,y1_,x1_*y1_,1,0,0,0,0),(x2_,y2_,x2_*y2_,1,0,0,0,0),(x3_,y3_,x3_*y3_,1,0,0,0,0),
                  (x4_,y4_,x4_*y4_,1,0,0,0,0),(0,0,0,0,x1_,y1_,x1_*y1_,1),(0,0,0,0,x2_,y2_,x2_*y2_,1),
```

```
                    (0,0,0,0,x3_,y3_,x3_*y3_,1),(0,0,0,0,x4_,y4_,x4_*y4_,1)])
    b = np.array([(point[0][0]),(point[1][0]),(point[2][0]),(point[3][0]),(point[0][1]),(point[1][1]),\
                  (point[2][1]),(point[3][1])])
    x = np.linalg.solve(A, b)
    return x

def inverse_mapping(b,input_img,output_img):
    print(input_img.shape , output_img.shape)
    row ,col = output_img.shape[:2]
    for i in range(row):
        for j in range(col):
            d_y = ((b[0]*i) + (b[1]*j) + (b[2]*i*j) + b[3])
            d_x = ((b[4]*i) + (b[5]*j) + (b[6]*i*j) + b[7])
            y , x = int(d_y) , int(d_x)
            u , v = d_y-y , d_x-x
            for c in range(3):
                output_img[i,j,c] = (1-u) * (1-v) * input_img[x,y][c] + u *(1-v) * input_img[x,y+1][c] + \
                v * (1-u) * input_img[x+1,y][c] + u*v*input_img[x+1,y+1][c]
    cv2.imshow("img",output_img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

img_path = './'
image = search_file(img_path)
if 'desktop' in image:image.remove('desktop')


for i in image:
    expect_size = [1200,1000]
    point=[]
    img = cv2.imread(f'{img_path}{i}.JPG')
    img = cv2.resize(img,(int(img.shape[0]*0.3),int(img.shape[1]*0.3)))
    img2 = img.copy()
    cv2.imshow("img",img2)
    cv2.setMouseCallback('img',show_xy)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    x = calculate(img ,expect_size, point)
    output_img = np.zeros((expect_size[0],expect_size[1],3),dtype=img.dtype)
    inverse_mapping(x,img,output_img)
```

圖 十三 整體程式碼

心得：

我本身做圖片的處理通常都是直接呼叫 opencv 套件，第一次自己將完整影像處理的 code 寫完，蠻有成就感的，其中我遇到最大的困難是 opencv 與 numpy 的座標系統是相反的，導致我在寫程式時出現了些問題，但好在後來 debug 過後結果已經是正確的了，相信經過這次作業對於我未來在影像處理實作上會有很大的幫助。