高等數位訊號處理

期末報告

影像的幾何變換
Geometric Transformation of Images

學號：B10505047

系級：電機三

姓名：邱郁喆

一、摘要

影像的幾何變換(Geometric Transformations)是常用的數位影像處理技術，此技術主要是將一張影像中的座標位置映射到另一張影像中的新座標位置。透過幾何變換會改變影像中的像素空間，但不改變其灰階或色彩值，也就是說這種幾何變換不改變像素值，只是在平面上進行像素位置的重新排列。常見的幾何變換主要包括影像的平移(Translation)、旋轉(Rotation)、縮放(Scaling)、翻轉(Flipping)等基本變換，以及更複雜的仿射變換(Affine Transformation)、透視變換(Perspective Transformation)和非線性變換(Non-linear Transformation)等等。本書面報告旨在對於這些常見的影像幾何轉換的基本原理與概念進行介紹，並且會透過 OPENCV 這個電腦視覺函式庫(模組)對於每種介紹到的幾何轉換進行些許實作範例，透過 OPENCV 對應的函式實作來體現這些幾何變換對於影像處理的實際效果，而程式碼(geometric_transformation.py)也會隨此份文件一同繳交。

二、簡介

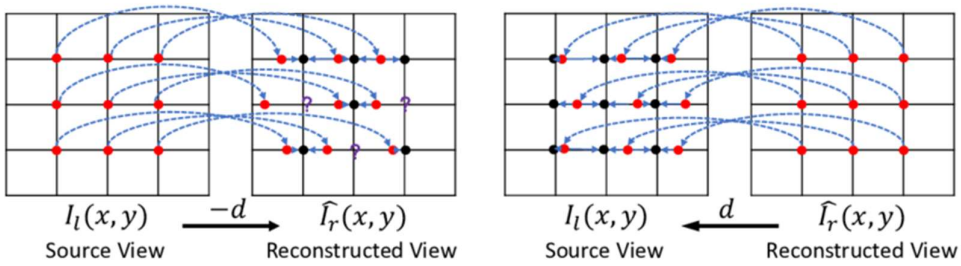
1. 影像幾何變換的基本概念：

影像幾何變換的核心在於改變像素的空間位置，建立原影像像素與變換後影像像素之間的映射關係，且此轉換通常不改變像素的灰階值或色彩值。而影像幾何變換一般會涉及以下三種基本概念：

(1)**座標系統**：座標系統是用於定義影像中像素空間位置的參考系統，在數位影像處理中，最常用的座標系統是笛卡兒座標系，它使用兩條互相垂直的軸（x 軸和 y 軸）來表示影像中像素的水平和垂直位置。

(2)**空間轉換**：空間轉換是指透過空間轉換函式將影像中的像素從一個空間位置映射到另一個空間位置的過程，空間轉換的方法包含正向映射(Forward Mapping)與反向映射(Inverse Mapping)。正向映射主要是根據輸入影像的空間座標(x, y)以及轉換函數 T，計算輸出影像映射的空間座

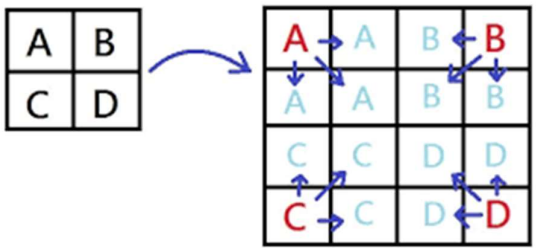
標(x', y')，也就是只要給定原影像上的任意像素座標，都能透過對應的映射關係獲得該像素在變換後影像的座標位置。反向映射則是跟正向映射相反，是根據輸出影像的空間座標(x', y')以及反轉換函數 T'，計算其在輸入影像的空間座標(x', y')，也就是使用輸出影像的座標反過來推算對應於原影像中的座標位置，以下正向映射與反向映射的範例圖：



圖一、正向映射(左)與反向映射(右)示意圖

(3) **影像插值**：無論採用正向或反向映射，在決定輸出影像像素的灰階色彩值，都有可能發生無法對應到整數空間座標的情形，這個時候通常就會使用插值法。插值法是估算影像在未明確定義位置的像素值的過程，而插值法通常是根據周圍像素的值來估算新位置的像素值。常用的插值演算法為最鄰近插值法(Nearest Neighbor Interpolation)、雙線性插值法(Bilinear Interpolation)、雙立方插值法(Bicubic Interpolation)。以下簡單介紹這三種常用的插值演算法：

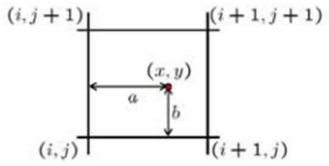
最鄰近插值法：最鄰近插值法是最簡單的方法，也稱為零階內插法(Zero-Order Interpolation)。這個方法是根據離映射點最近的空間座標像素取其灰階值或色彩值，如下圖所示:



圖二、最鄰近插值法示意圖

最鄰近插值幾乎沒有多餘的運算，速度相當快，但是這種鄰近取值的方法是比較粗糙的，當要放大的倍率越多，此方法的效果就會越差，因此造成影像的馬賽克、鋸齒等現象。

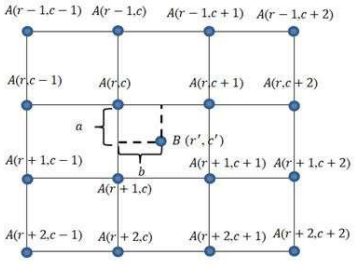
雙線性插值法：此方法又稱為二階內插法(Second-Order Interpolation)，其概念就是無論經過轉換函式後得到的座標是多少，其一定會介於原圖中某四個整數點之間，如下圖所示，根據這個點(x, y)與四個點距離的權重來決定此點的值。此方法計算複雜度不高，並且也有不錯的效果。



$$
\begin{aligned}
f(x,y) = \; & (1-a)(1-b) \quad f[i,j] \\
& +a(1-b) \quad f[i+1,j] \\
& +ab \quad f[i+1,j+1] \\
& +(1-a)b \quad f[i,j+1]
\end{aligned}
$$

圖三、雙線性插值法示意圖

雙立方插值法：此方法又稱為三階內插法(Third-Order Interpolation)，它和 bilinear interpolation 類似，只是將參考點從周圍四點提高到周圍十六個點，如下圖所示。由於參考點更多，因此此方法的計算量大，但可以保留較精確的影像細節品質，但在即時影像處理較不實用。



圖四、雙立方插值法示意圖

除了這三個常見的插值演算法之外，還有許多更複雜的插值法像是 Lanczos 插值法、Hermite 插值及高階插值法等等。這些插值法通常計算

量比較大，因此在實作上較少被使用。

三、常見的幾何變換介紹與實作範例

1. 平移(Translation)：平移算是最簡單的一種幾何變換，基本上影像的平移變換就是將影像中的座標加上指定的水平和垂直偏移量。假設(x, y)是平移前的座標，dx, dy 是座標要加上的水平和垂直偏移量，平移後的座標為(x', y')，則平移的關係式為：$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}$

   接著用 OPENCV 實作的程式碼如圖五，先取得需要進行平移的矩陣 M。程式碼中以 x 軸平移 100（向右），y 軸平移 50(向左)為例，之後透過 warpAffine 函式即可得到圖六做平移變換後的結果(圖六)。

```
def translation():
    M = np.float32([[1, 0, 100], [0, 1, 50]]) # shift along x-axis 100, shift along y-axis 50
    output = cv2.warpAffine(img, M, (w, h)) # img size is h x w
```

圖五、平移變換程式碼範例

圖六、原圖

圖七、對圖六平移後的結果

2. 旋轉(Rotation)：影像的旋轉變換就是將影像繞著某個中心按照指定的角度進行旋轉。影像旋轉後不會變形，但是其垂直對稱軸和水平對稱軸都會發生改變，旋轉後的影像的坐標和原影像坐標之間的關係已不能通過簡單的加減乘法得到，而需要通過一系列的複雜運算。而且影像在旋轉後的寬度和高度都會發生變換，其座標原點會發生變換。

   旋轉的關係式為：$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$，其中 θ 為指定的旋轉角度

接著用 OPENCV 實作的程式碼如圖八，先透過 getRotationMatrix2D 函式取得需要的旋轉矩陣 M。程式碼中以座標(240, 180)為旋轉中心，旋轉角度為 45 度，之後透過 warpAffine 函式即可得做旋轉變換後的結果(圖九)。

```
def rotation():
    M = cv2.getRotationMatrix2D((240, 180), 45, 1) # center point (240, 180), spin 45 degrees, size 1
    output = cv2.warpAffine(img, M, (w, h)) # img size is h x w
```

圖八、旋轉變換程式碼範例

圖九、對圖六做旋轉變換後的結果

3. 縮放(Scaling)：影像的縮放轉換主要是按照特定的縮放比例改變影像的大小，縮放比例可以是整數或浮點數，縮放後影像的寬度和高度會發生變換。縮放的關係式為：$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$，其中$s_x$、$s_y$ 為縮放比例

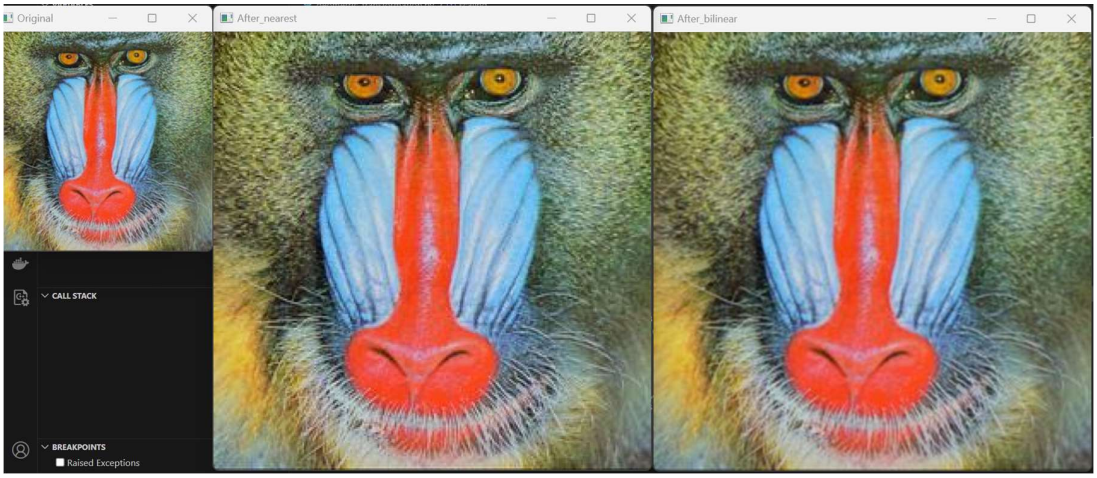
若縮放後得到了浮點坐標，就要用前面提及的插值法取得近似的像素值。

接著用 OPENCV 實作的程式碼如圖九，透過 resize 函式可以輕鬆對原圖進行縮放。resize 中的參數 dsize 為輸出影像大小，fx, fy 分別為水平跟垂直軸的縮放比例，而 interpolation 則是要使用的插值法。此範例中分別使用最鄰近插值法與雙線性插值法對原圖縮放兩倍：圖十中左邊為原圖，中間為使用最鄰近插值法縮放後的結果，右邊則為使用雙線性插值法縮放後的結果。雖然乍看之下兩者差異不大，但若仔細看的話會發現其實使用雙線性插值法

的效果還是好一點，而最鄰近內插的效果則稍有較多的馬賽克區塊。

```
def scaling():
    output1 = cv2.resize(img, dsize=None, fx=2, fy=2, interpolation=cv2.INTER_NEAREST) # nearest
    cv2.imwrite('scaling.jpg', output1)
    output2 = cv2.resize(img, dsize=None, fx=2, fy=2, interpolation = cv2.INTER_LINEAR) # bilinear
    cv2.imwrite('scaling.jpg', output2)
```

圖九、縮放變換程式碼範例



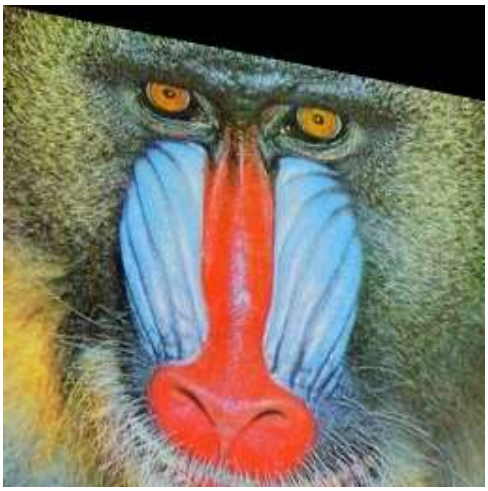
圖十、對圖六使用最鄰近插值法與雙線性插值法對原圖縮放兩倍的結果

4. 剪切(Shearing)：剪切變換主要是將影像中的像素沿特定方向進行平行移動，從而改變影像的形狀。剪切可分成水平剪切和垂直剪切，水平剪切的關係式為 $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$，垂直剪切的關係式為 $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

接著用 OPENCV 實作的程式碼如圖十一，一樣先取得剪切需要的矩陣 M (此範例為垂直剪切)，再透過 warpAffine 函式得到剪切變換後的結果(圖十二)。

```
def shearing():
    M = np.float32([[1, 0, 0], [0.2, 1, 0]])
    output = cv2.warpAffine(img, M, (w, h))
```

圖十一、剪切變換程式碼範例



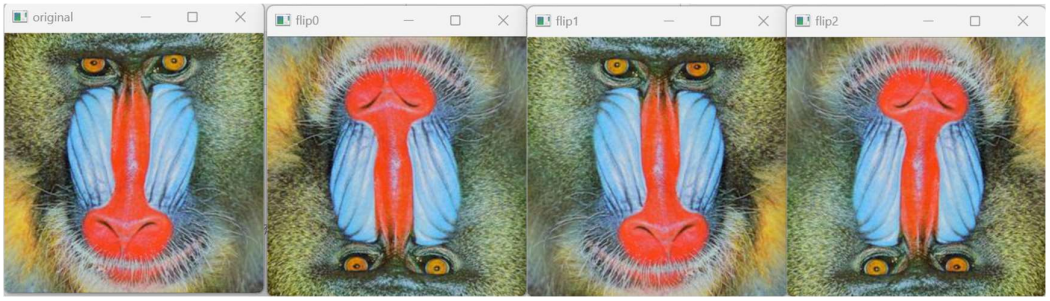
圖十二、對圖六做垂直剪切變換後的結果

5. 翻轉(Flipping)：翻轉是將影像沿著指定軸進行反轉，分成水平翻轉和垂直翻轉。水平翻轉的關係式為 $x' = -x$，垂直翻轉的關係式為 $y' = -y$

接著用 OPENCV 實作的程式碼如圖十三，透過 flip 函式可得到垂直和水平翻轉變換後的結果如圖十四，其中由左到右依序為原圖、垂直翻轉、水平翻轉、垂直跟水平方向都翻轉的結果。

```
def flipping():
    output0 = cv2.flip(img, 0)  # vertical flip
    output1 = cv2.flip(img, 1)  # horizontal flip
    output2 = cv2.flip(img, -1) # both vertical and horizontal flip
```

圖十三、翻轉變換程式碼範例



圖十四、翻轉變換後的結果

6. 仿射變換(Affine Transformation)：仿射變換是一種混合的線性二維幾何轉換，其實就是混合了平移、旋轉、縮放、剪切等基本變換。仿射變換保持影像中直線和平行線的性質，也就是說經過仿射轉換後點、直線和平面仍然是

點、直線和平面，平行線仍然保持平行。

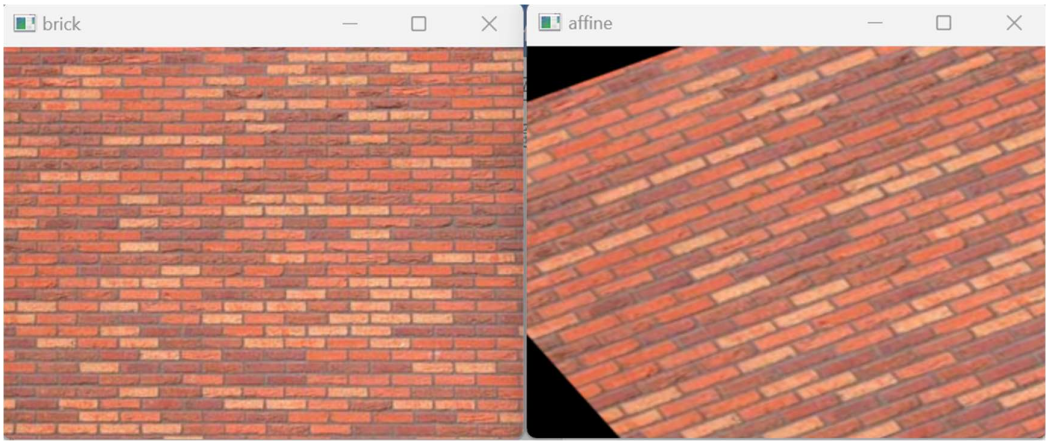
仿射變換的關係式為 $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$，即 $\begin{cases} x' = a_{11}x + a_{12}y + a_{13} \\ y' = a_{21}x + a_{22}y + a_{23} \end{cases}$

由此可知一組對應點可以貢獻兩個式子，所以我們需要三組對應點(原圖像中的點及其映射在目標圖像的對應點)來解出仿射變換矩陣中的六個未知數 a_{11}~a_{23}，也就是說仿射變換需要至少三組對應點來確定變換矩陣。

接著用 OPENCV 實作的程式碼如圖十五，基本上就是將三組對應點代入 getAffineTransform 這個函式即可得到仿射變換矩陣，再透過 warpAffine 函式得到仿射變換後的結果(圖十六)。

```
def affine():
    pts1 = np.float32([[50,50],[150,50],[50,150]])
    pts2 = np.float32([[10,100],[150,50],[100,200]])
    M = cv2.getAffineTransform(pts1, pts2)
    output = cv2.warpAffine(img, M, (w, h))
```

圖十五、仿射變換程式碼範例



圖十六、左為原圖，右為仿射變換後的結果

7. 透視變換(Perspective Transformation)：又稱為單應性(Homography)變換，影像透視變換的概念是將影像投影到一個新的視平面，是一種非線性的幾何變換，它可以將一個二維空間中的點映射到另一個二維空間中。透視變換使用一個 3x3 矩陣進行映射，其關係式為 $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$，而此矩

陣稱為 Homography Matrix。

在實際上運算時我們通常會令 $h_{33}=1$，並得到兩個式子 $x'=\frac{h_{11}x+h_{12}y+h_{13}}{h_{31}x+h_{32}y+1}$ 及 $y'=\frac{h_{21}x+h_{22}y+h_{23}}{h_{31}x+h_{32}y+1}$，化簡後可得 $\begin{cases}x'=h_{11}x+h_{12}y+h_{13}-h_{31}xx'-h_{32}yx'\\y'=h_{21}x+h_{22}y+h_{23}-h_{31}xy'-h_{32}yy'\end{cases}$

因此我們可以得出每一組對應點(原圖像中的點及其映射在目標圖像的對應點)可以貢獻兩個式子：$\begin{bmatrix}x & y & 1 & 0 & 0 & 0 & -xx' & -yx'\\0 & 0 & 0 & x & y & 1 & -xy & -yy'\end{bmatrix}\begin{bmatrix}h_{11}\\h_{12}\\h_{13}\\h_{21}\\h_{22}\\h_{23}\\h_{31}\\h_{32}\end{bmatrix}=\begin{bmatrix}x'\\y'\end{bmatrix}$

因此我們可以知道要解這個矩陣中的八個未知數需要四組對應點，並且無論是在原影像或是輸出影像的那四個點之中都不能有三點共線。

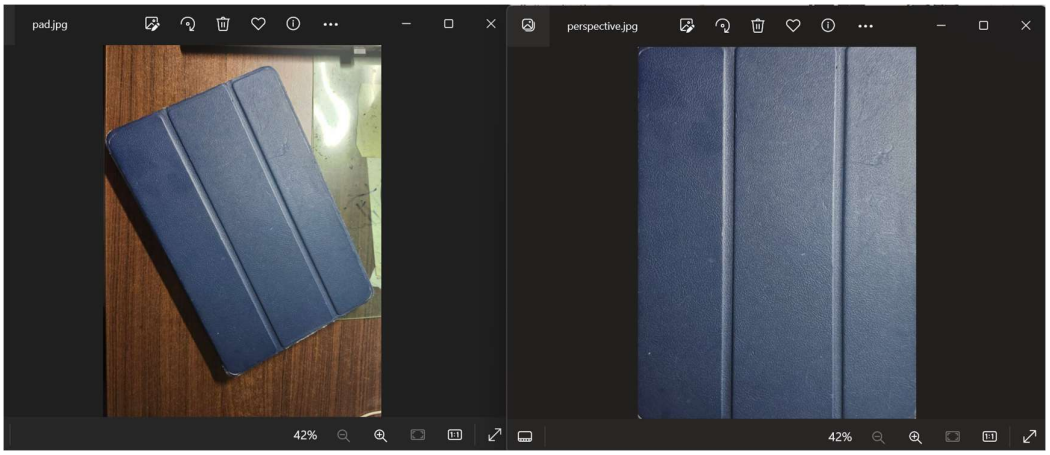
這裡要介紹一個概念叫做齊次座標(Homogeneous Coordinate)：笛卡兒座標空間適合描述二維和三維空間，但無法描述透視空間，例如平行線在笛卡兒座標空間中無法相交，但在透視空間中的無窮遠處是會相交於同一點。因此，在透視變換空間使用的是齊次座標，它使用 N+1 維來表示 N 維空間座標，例如對於一個二維笛卡兒座標上的點(x, y)，它對應的齊次座標為(x, y, 1)；對於一個三維笛卡兒座標上的點(x, y, z)，它對應的齊次座標為(x, y, z, 1)。而透視變換簡單來說就是透過 Homography Matrix 將原影像中的點(x, y, 1)映射到輸出影像中的點(x', y', 1)。

此外，透視變換經常被拿來跟前一點介紹的仿射變換作比較，兩者最大的不同是透視變換不會使平行線保持平行，並且會讓距離、角度產生變化。

接著用 OPENCV 實作的程式碼如圖十七，首先找出四組對應點代入 getPerspectiveTransform 函式中，取得 homography 矩陣 H，接著將矩陣代入 warpPerspective 函式，使原圖透過矩陣映射到輸出影像上。圖十八為透視轉換實作結果的範例。

```python
def perspective():
    pts1 = np.float32([[10,371],[665,91],[513,1310],[1070,983]])
    pts2 = np.float32([[0,0],[1107,0],[0,1476],[1107,1476]])
    H = cv2.getPerspectiveTransform(pts1, pts2)
    output = cv2.warpPerspective(img, H, (w, h))
```

圖十七、透視變換程式碼範例



圖十八、左為原圖，右為針對左邊物體四個點進行透射變換後的結果

四、結論

影像幾何變換是一項在數位影像處理當中簡單卻又重要的技術此技術改變影像中的像素空間來達到不同的視覺效果，但不改變其灰階或色彩值。本篇報告中介紹的這些幾何變換技巧都非常實用，並且配合 OPENCV 這個非常方便的函式庫使用能夠輕鬆地達到想要的效果。而雖然隨著科技的發展，現在有越來越多新穎複雜的影像處理技術，然而有時候傳統、簡單的方法還是有好用的地方。因此，以影像處理這個領域來說，我想學習像是本篇介紹的幾何變換技巧這些基本技術還是重要的一件事。

五、參考文獻

1. https://codingnote.cc/zh-tw/p/259306/#google_vignette

2. https://ithelp.ithome.com.tw/articles/10289372

3. OPENCV 函式庫之 geometric transformations 教學：

https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html

4. Affine Transformation 與 Perspective Transformation 的比較：

https://medium.com/@fearless_fusion_snake_755/affine-transformation-v-s-perspective-transformation-950c86a8e0b4

5. 為什麼需要齊次座標(Homogeneous Coordinate)

https://flyhead.medium.com/%E7%82%BA%E4%BB%80%E9%BA%BC%E9%9C%80%E8%A6%81%E9%BD%8A%E6%AC%A1%E5%BA%A7%E6%A8%99-homogeneous-coordinate-bd86356f67b1