Day 06

深度學習與電腦視覺

學習馬拉松

cupoy 陪跑專家：楊鎮銘

CUPOY

基礎影像處理

仿射變換的概念與實作
(Affine Transformation)

重要知識點

CUPOY

- 了解仿射變換包含的重要特性與概念
- 以旋轉變換為例了解仿射變換矩陣的設計

OpenCV 畫圖

CUPOY

前面我們介紹透過 Transformation matrix 的概念做平移轉換
這邊進一步介紹線性變換加上平移的仿射變換

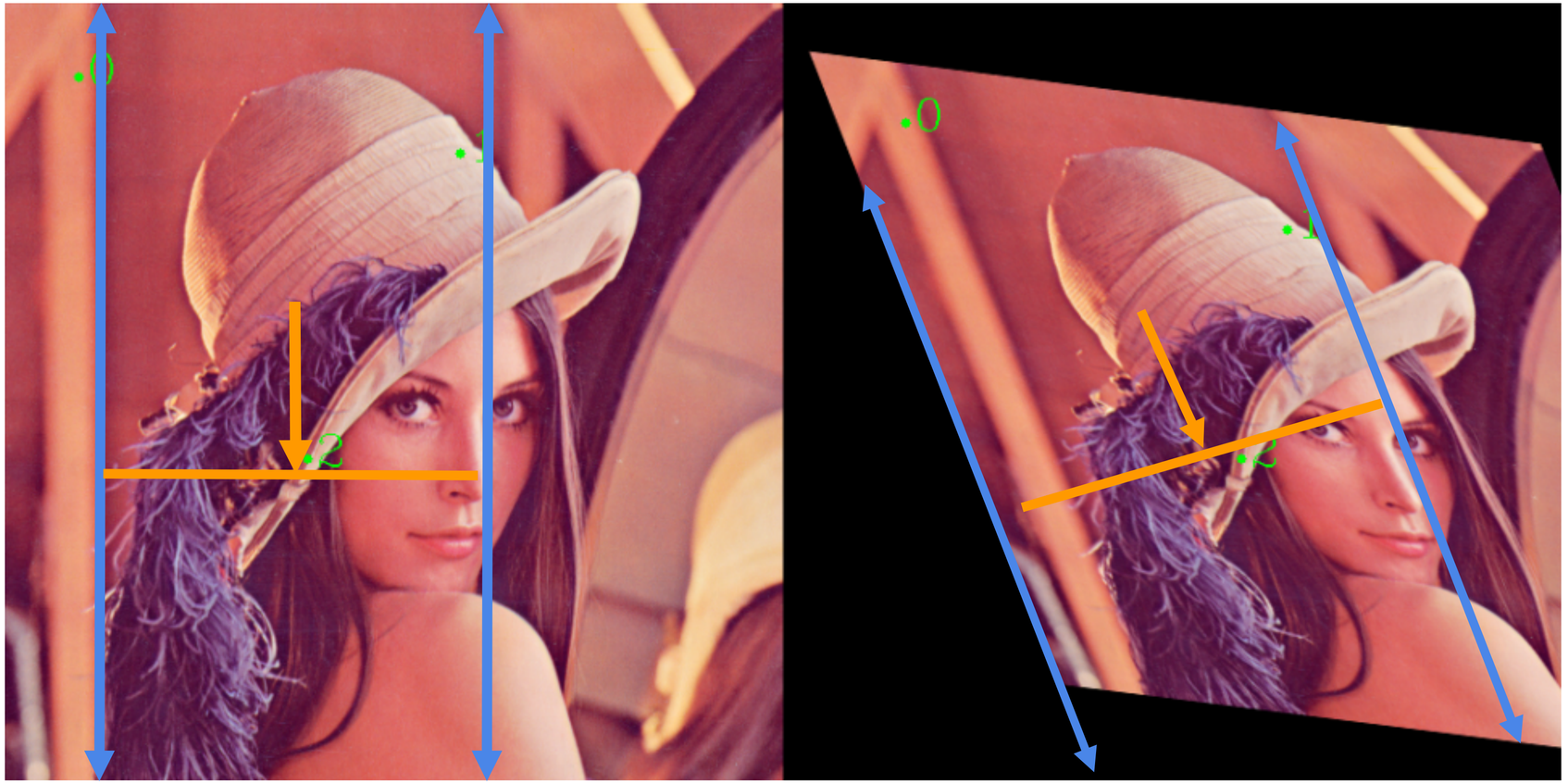
仿射不一定會保持物體的面積與大小，但是具有以下兩種特性

- 共線不變性：平行的線經過轉換後還是會平行
- 比例不變性：兩點的中點經過轉換還會是中點

仿射變換 Affine Transformation

CUPOY

共線不變性: 一樣是平行的線



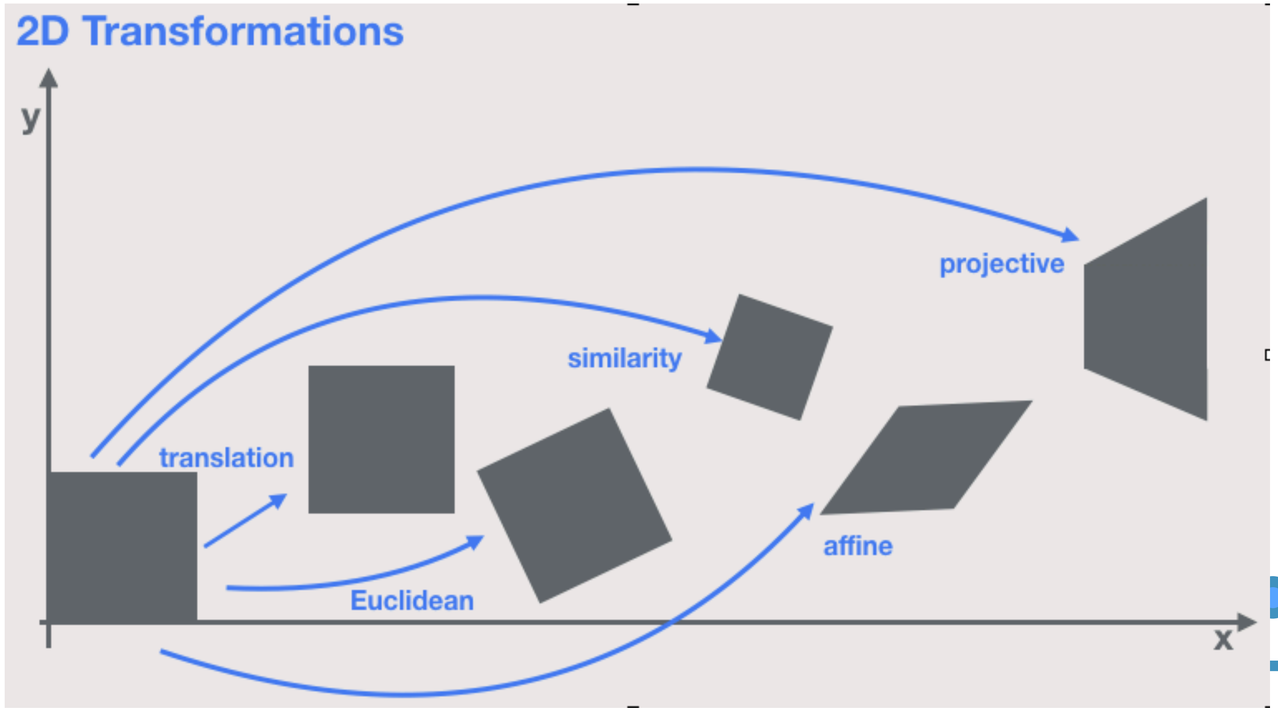
比例不變性: 一樣是兩點的中點

仿射變換矩陣概念

CUPOY

前面我們提到仿射變換是線性變換後再做平移的結果
線性變化可以套上各種線性變換矩陣，所以仿射變換非固定格式
常見的線性變換

- 平移 (Translation)
- 旋轉 (Rotation)
- 鏡射 (Reflection)
- 伸縮 (Stretching/ Squeezing)
- 切變 (Shearing)



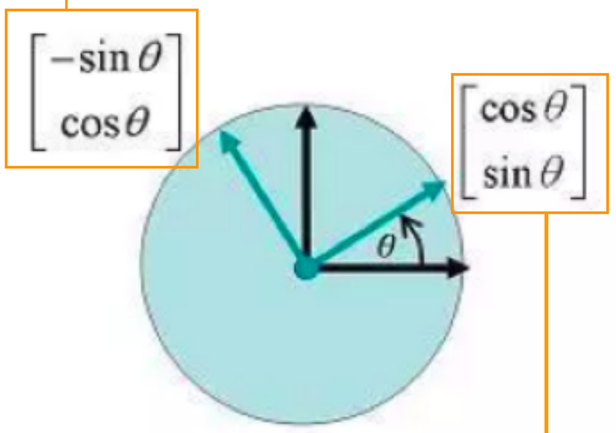
線性變換 - 以旋轉矩陣為例

線性變換有許多種方式，這邊以前面沒有提過的旋轉矩陣為例，
跟 1-4 章節提到的 Transformation 類似，一樣透過齊次座標 (homogenious) 表示

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

x , y 軸轉為兩個單位的向量

y 軸：(0, 1) 的點旋轉 θ 後變為 (-sinθ, cosθ)



x 軸：(1, 0) 的點旋轉 θ 後變為 (cosθ, sinθ)

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

旋轉矩形操作　x旋轉　y旋轉

齊次座標後面章節會介紹，這邊先以
Transformation 理解即可

仿射變換 - 以旋轉矩陣為例 (optional)

CUPOY

旋轉矩陣的一般式可以寫為

$$T(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$$

n*n 線性變換矩陣

平移向量

其中線性變換矩陣我們以旋轉矩陣為例

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R(\theta)Tr(t_x, t_y) = T(x) = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

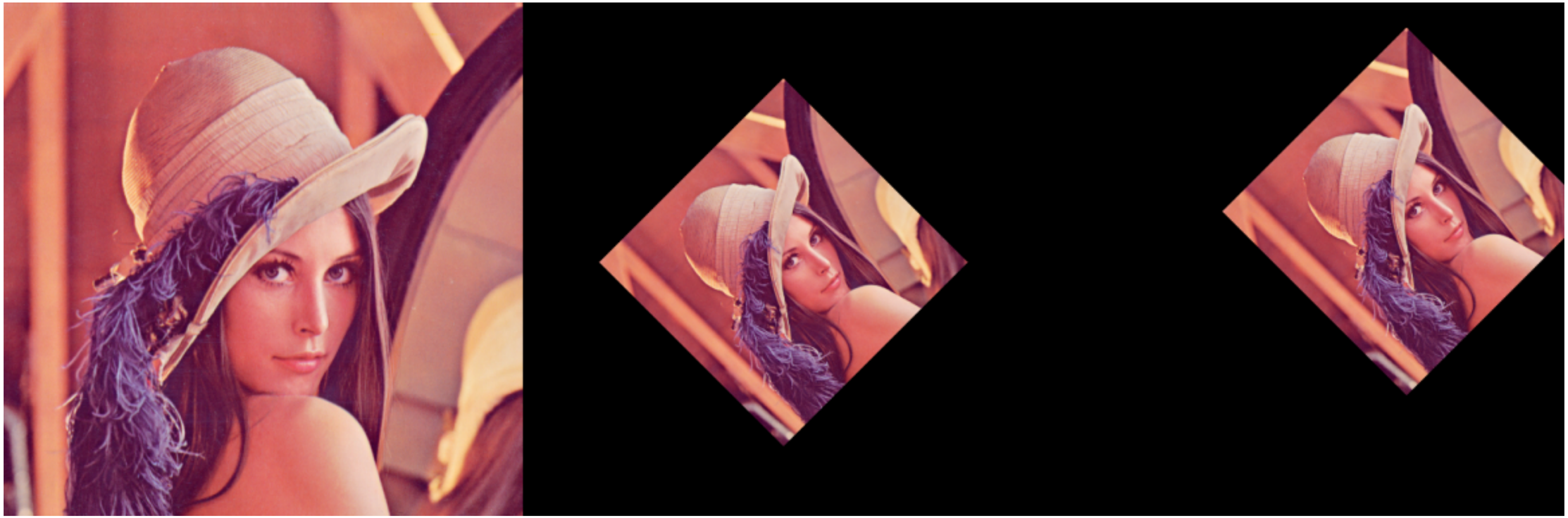
x , y 向量格式 → **齊次座標表示** → **加上平移向量**

這頁希望透過數學推導解釋如何透過 Transformation 來做旋轉

可以當作參考教材視情況閱讀

仿射變換 - 以旋轉矩陣為例 (範例)

CUPOY



原圖旋轉 45 度 → 縮小 0.5 倍 → x 平移 +100 y 平移 -50

仿射變換 (optional)

另外一個觀點，只要有一對圖片包含三組點就可以直接建構出矩陣
e.g. A 圖的三個點 (p1, p2, p3) 對應到 B 圖的三個點 (p1', p2', p3')

仿射矩陣的一般式可以改寫為

$$T(\mathbf{x}) = A\mathbf{x} + \mathbf{b} \longrightarrow T(x, y) = (ax + by + c, dx + ey + f)$$

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ i \end{bmatrix}$$

給予 3 點 A, B, C 解方程式直接計算矩陣值

$$\begin{aligned} A'_1 &= aA_1 + bA_2 + c \\ A'_2 &= dA_1 + eA_2 + f \\ B'_1 &= aB_1 + bB_2 + c \\ B'_2 &= dB_1 + eB_2 + f \\ C'_1 &= aC_1 + bC_2 + c \\ C'_2 &= dC_1 + eC_2 + f \end{aligned} \quad \begin{bmatrix} A_1 & A_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & A_1 & A_2 & 1 \\ B_1 & B_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & B_1 & B_2 & 1 \\ C_1 & C_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & C_1 & C_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} A'_1 \\ A'_2 \\ B'_1 \\ B'_2 \\ C'_1 \\ C'_2 \end{bmatrix}$$

參考來源： StackExchange

知識點回顧

- 仿射變換可以是多種線性變換的組合，沒有固定格式
  - 了解旋轉矩陣的設計方式
  - 複習平移矩陣 (Translation Transformation)

範例

- 練習以旋轉變換 + 平移變換來實現仿射變換
  - 旋轉 45 度 + 縮放 0.5 倍 + 平移 (x+100, y-50)
- 透過旋轉矩陣處理

```
# 旋轉矩陣
cv2.getRotationMatrix2D(中心位置, 旋轉角度, 縮放倍率)
```

推薦延伸閱讀

CUPOY

1. 仿射變換 | 線代啟示錄 連結

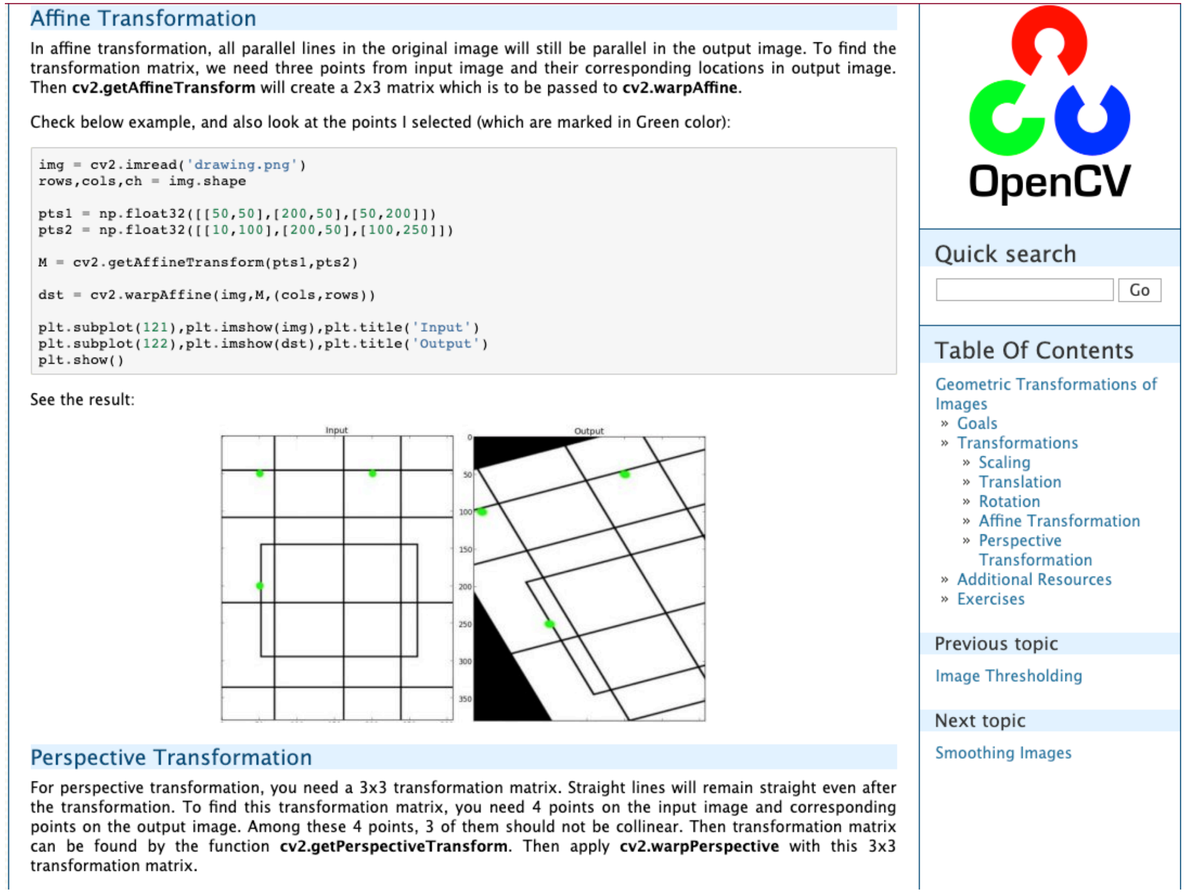
透過數學證明仿射變換為甚麼具有比例不變性，
以及依序推導仿射變換矩陣的設計

2. 幾和變換矩陣設計 | 線代啟示錄 連結

裏面包含其他線性變換的證明與講解，
其中切變變換 (Shearing) 屬於 optional 的內容

推薦延伸閱讀

CUPOY



Affine Transformation

In affine transformation, all parallel lines in the original image will still be parallel in the output image. To find the transformation matrix, we need three points from input image and their corresponding locations in output image. Then **cv2.getAffineTransform** will create a 2x3 matrix which is to be passed to **cv2.warpAffine**.

Check below example, and also look at the points I selected (which are marked in Green color):

```
img = cv2.imread('drawing.png')
rows,cols,ch = img.shape

pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])

M = cv2.getAffineTransform(pts1,pts2)

dst = cv2.warpAffine(img,M,(cols,rows))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

See the result:

Perspective Transformation

For perspective transformation, you need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, you need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then transformation matrix can be found by the function **cv2.getPerspectiveTransform**. Then apply **cv2.warpPerspective** with this 3x3 transformation matrix.

OpenCV

Quick search

Go

Table Of Contents

Previous topic

Image Thresholding

Next topic

Smoothing Images

Geometric Transformation - Affine Transformation

如果對 OpenCV 實作不夠清楚可以參考官方實作程式碼

連結

解題時間 Let’s Crack It

CUPOY

請跳出 PDF 至官網 Sample Code & 作業開始解題