VTC STEM Education Centre
stem@vtc.edu.hk

Python Application

電腦視覺

(Computer Vision)

參考工具

Python 官方網站

(https://www.python.org/)

StackOverflow

(https://stackoverflow.com/)

Visual Studio Code

(https://code.visualstudio.com/download)

GitHub

(https://github.com/kwgarylam/pythonWorkshop)

1. Python 簡介

Python 是當今最流行的程式語言之一，應用範圍十分廣的「萬用語言」，包括：人工智慧、大數據分析、用戶圖形接口、遊戲、Web、硬體自動控制等等。其缺點是執行速度比某些程式語言緩慢，如 C++。

2019 十大程式語

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.5
4. C++		97.1
5. C#		87.7
6. R		87.7
7. JavaScript		85.6
8. PHP		81.2
9. Go		75.1
10. Swift		73.7

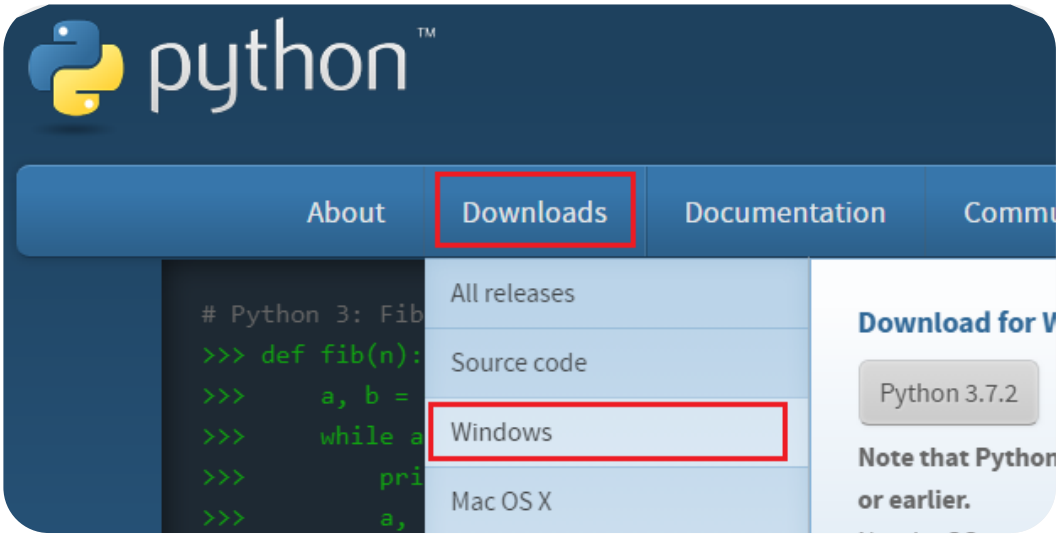
資料來源：IEEE Spectrum

A. 安裝 Python

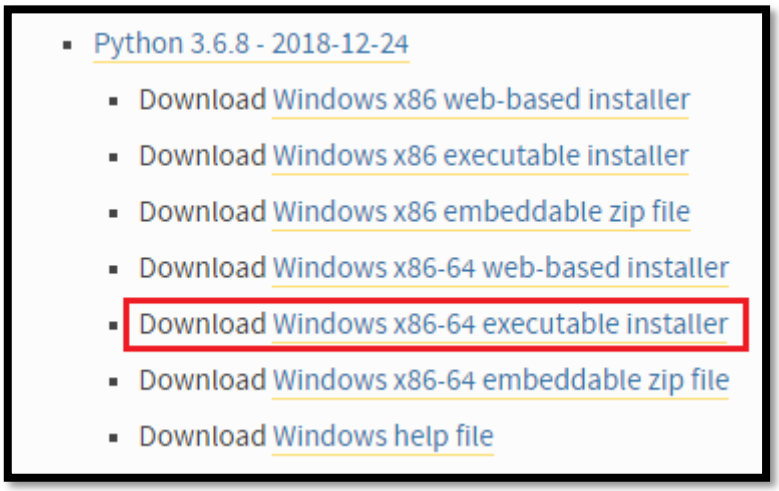
安裝 Python 3.6.8

Python 最新的版本是 3.7.2 ，但有些 Framework (如 TensorFlow) 暫時未支援此版本，所以本課程是使用 3.6.8 版本。

在 Python 官方網站 (www.python.org) ，點擊 “**Windows**”



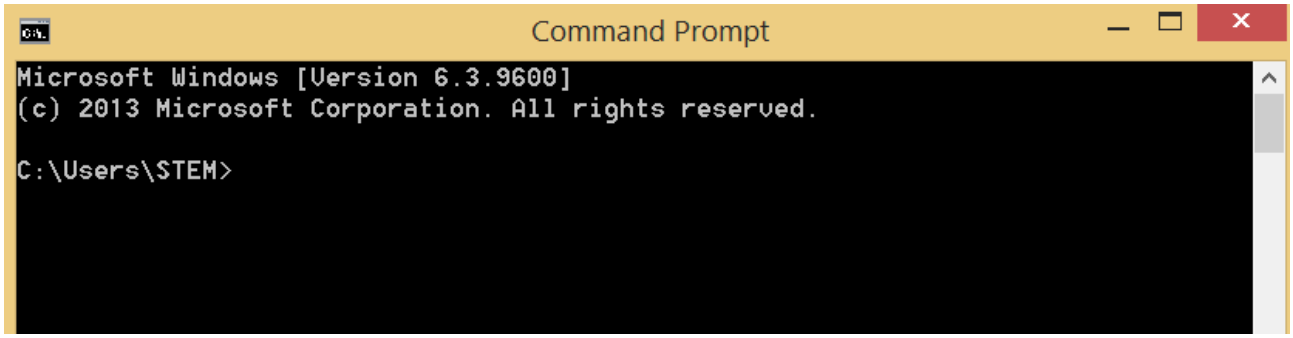
尋找 Python 3.6.8，點擊 “**Window x86-64 executable installer**” 下載及執行安裝程式。



選取 “**Add Python3.6 to PATH**” 及按 “**Install Now**” 安裝。

安裝相關 Packages

在 Python 中，安裝 Packages 最方便的方法是使用 pip (軟體包管理系統)
開啟 **Command Prompt** (Window) 或 **Terminal** (Mac)



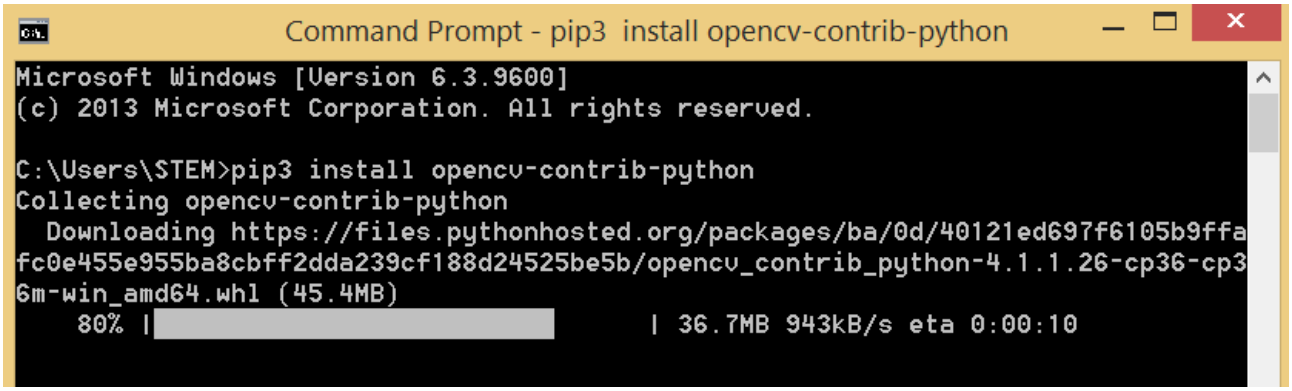
安裝 numpy:
numpy 是一個開源科學計算 (scientific computing) 的 Package。

```
pip3 install numpy
```

安裝 OpenCV:

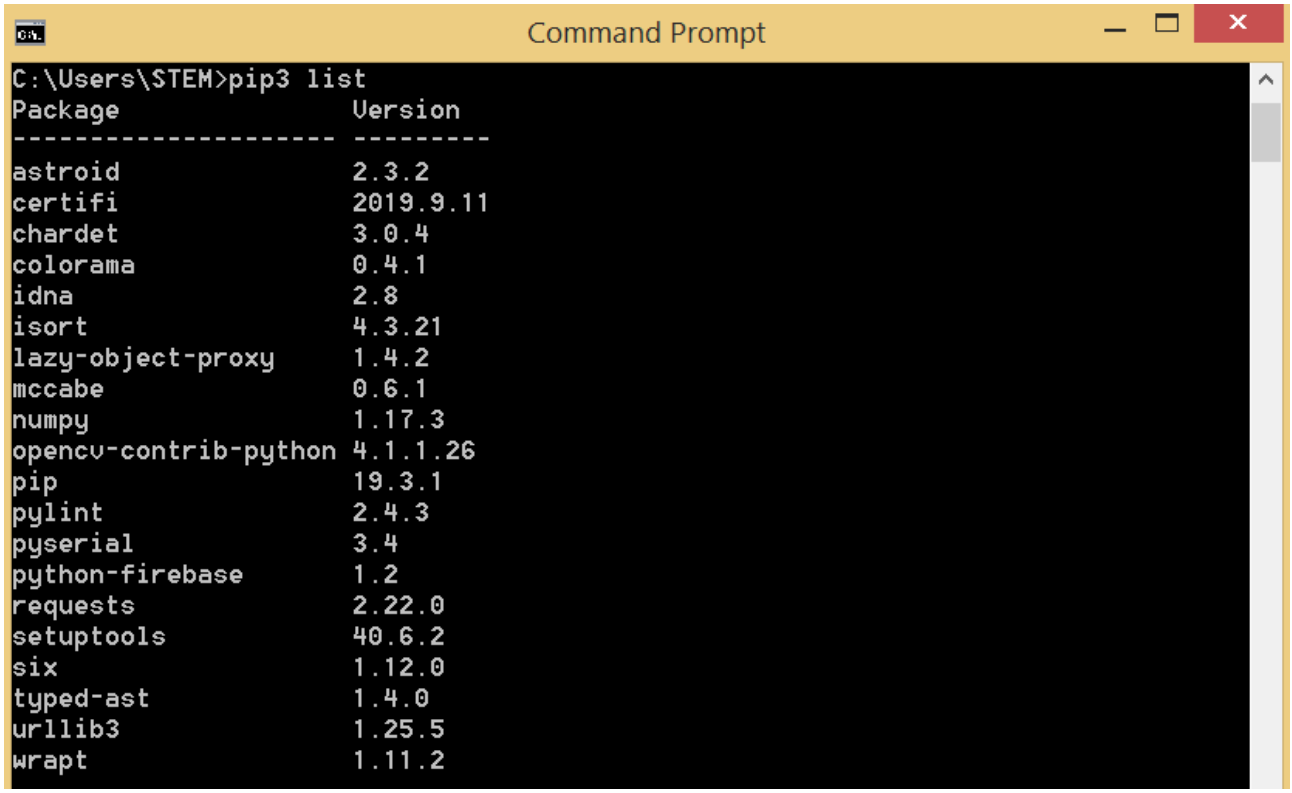
OpenCV (Open Source Computer Vision Library) 是一個開源的電腦視覺的程式庫。

```
pip3 install opencv-contrib-python
```



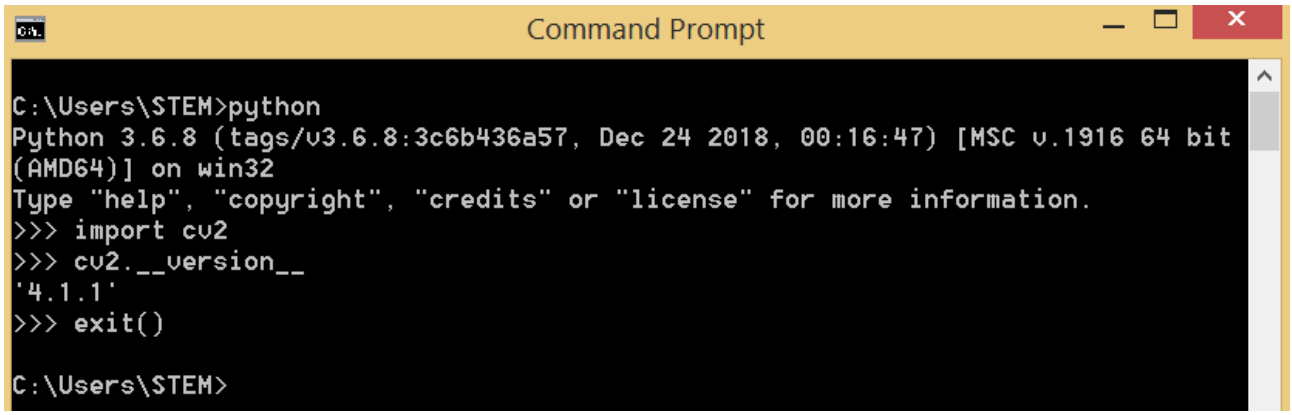
使用 pip3 list ，列出安裝了的 Packages。

```
pip3 list
```



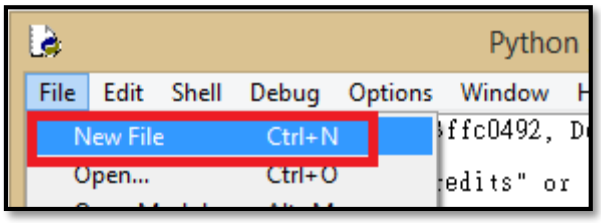
匯入 openCV 測試是否安裝成功。

```
import cv2
cv2.__version__
```



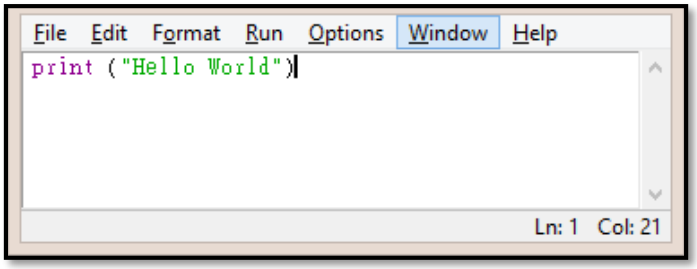
B. Hello World 程式

在 **IDLE** 程式，選取 **“File” => “New File”**

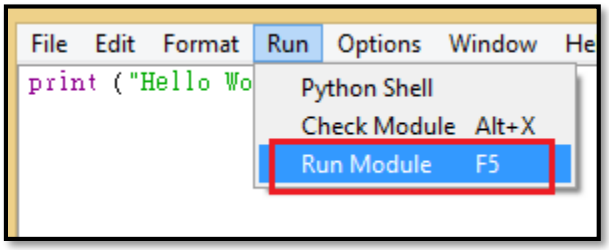


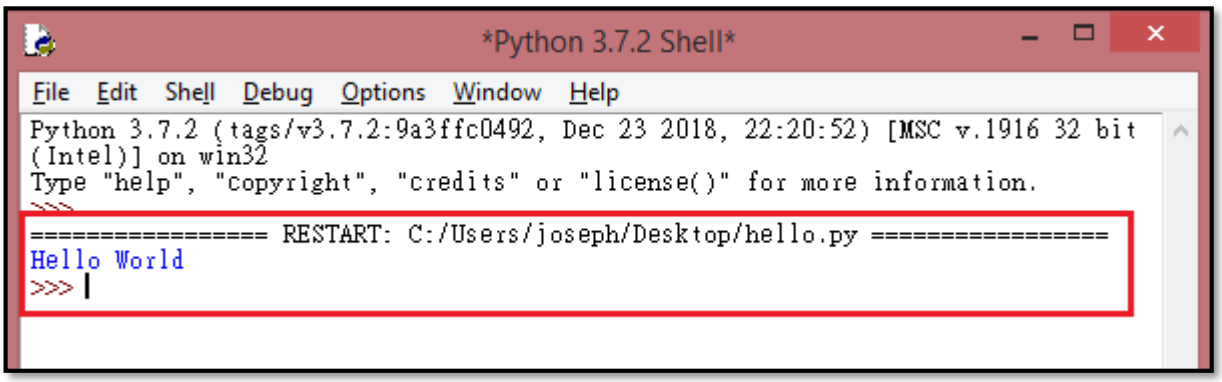
鍵入以下程式：

```
print ("Hello World")
```



將程式儲存為 **“hello.py”**。選取 **“Run” => “Run Module”** 或按 **F5** 鍵去運行程式。





C. 變數（variable）與資料類型（type）

變數是程式中用來暫存數據以作計算、輸入和輸出之用。
鍵入以下程式及運行：

```
strName = input ("Please enter your name:")
print ("Hello " + strName)
```

輸出結果為：

```
Please enter your name:STEM
Hello STEM
>>>
```

input() 函数是讀取使用者的數據。
strName 是一個變數，用來暫存使用者的名字 (例子中的 **STEM**)。而它的類型 (type) 是字串 (string)。

練習一

寫一個程式 (sayHi.py)：讀取使用者的**名字**及**姓氏**，然後輸出 Hi [名字]及[姓氏]

```
Please enter your first name:Joseph
Please enter your last name:Siu
Hi Joseph Siu
>>>
```

鍵入以下程式及運行 (add.py)：

```
iNum1 = int(input ("Please enter a number:"))
iNum2 = int(input ("Please enter another number:"))
answer = iNum1 + iNum2
print (iNum1, "+", iNum2, " = " + str(answer))
```

輸出結果為：

```
Please enter a number:1
Please enter another number:3
1 + 3  =  4
>>>
```

int () 函数是將字串轉換為整數。
str () 函数是將整數轉換為字串。
iNum1 和 **iNum2** 是整數類型的變數，用來暫存使用者輸入的兩個整數。
answer 是用來暫存運算的結果及輸出時使用。

D. 選擇 (if ... else ...)

編寫程式時，很多時需要作分流處理。例如測驗成績達 40 分為合格，否則為不合格；使用者為男或女，會有不同的處理。這時可以使用 if - else 敘述。

鍵入以下程式及運行 (checkPass.py)：

```
fMark = float(input ("Your test mark:"))
# check the test mark
if fMark >= 40:
    print("Pass!")
else:
    print("Fail!")
```

縮排

C++ version

```
if (fMark >=40)
{
    printf ("Pass!");
}else
{
    printf ("Fail!");
}
```

輸出結果為：

```
Your test mark:45.5
Pass!
>>>
```

```
Your test mark:37
Fail!
>>>
```

float () 函数是將字串轉換為實數(有小數位)。

符號是用來加入注解，編程器會跳過這部份。

如果 fMark >= 40 的檢查結果為真時，程式會執行 **if** 以下的敘述，上例是：print("Pass!") 否則會執行 **else** 以下的敘述，上例是： print("Fail!")。

注意：程式中的 ：及 縮排(Indentation)。

Operator	Description
>	greater than
<	less than
==	equal to
<=	less or equal to
>=	greater or equal to
!=	Inot equal to

如果要檢測多個條件，可以用 **and** 或 **or**。

練習二

寫一個程式 (checkGrade.py)：讀取測驗成績，然後根據下列表格輸出適當的等級：
(可以使用： **elif**)

測驗成績	等級
>=80	A
60 - <80	B
40 - <60	C
< 40	F

```
Your test mark:85
Your grade is A
>>>
```

```
Your test mark:76
Your grade is B
>>>
```

```
Your test mark:40
Your grade is C
>>>
```

```
Your test mark:37
Your grade is D
>>>
```

E. 迴圈 (loop)

編寫程式時，有時需要重複或搜索某一記錄，可以用 **for** 或 **while**。例如要列印 5 行 Hello World，可以執行以下敘述：

```
print ("Hello World")
print ("Hello World")
print ("Hello World")
print ("Hello World")
print ("Hello World")
```

但如果要列印 100 行，以上方法便不太方便。

使用迴圈 **for**：鍵入以下程式及運行 (forHello.py)：

```
for i in range (0,5):
    print ("Hello World")
```

或者

使用迴圈 **while**：鍵入以下程式及運行 (whileHello.py)：

```
i = 0
while i < 5:
    print ("Hello World")
    i = i + 1
```

輸出結果為：

```
Hello World
Hello World
Hello World
Hello World
Hello World
>>>
```

C++ version

```
for (int i=0; i<5; i ++)
{
    printf ("Helllo World")
}
```

C++ version

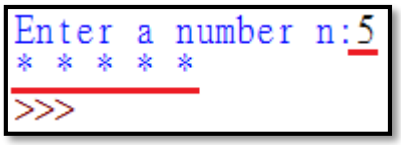
```
int i=0
while (i<5)
{
   printf ("Helllo World")
}
```

兩個迴圈

鍵入以下程式及運行 (printStar.py)：

```
n = int (input("Enter a number n:"))
for i in range (0, n):
    print("* ", end="")
```

輸出結果為：

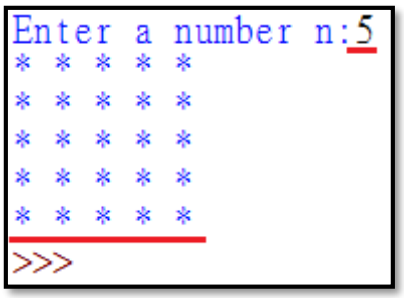


print("* ", end="") 是列印完 "*" 後保留在同一行。

加多一個迴圈

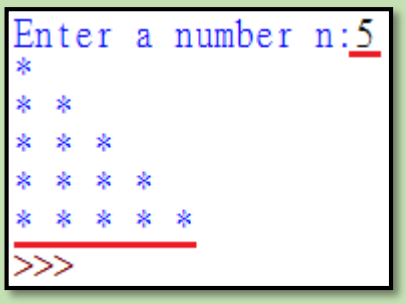
```
n = int (input("Enter a number n:"))
for j in range (0, n):
    for i in range (0, n):
        print("* ", end="")
    print ("")
```

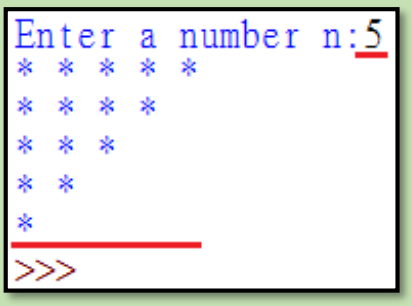
輸出結果為：



練習三

修改以上程式去列印下列結果：



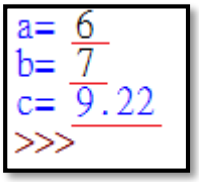


F. 函式庫 (Library)

鍵入以下程式及運行 (pythagorean.py)：

```
import math
a = float(input("a= "))
b = float(input("b= "))
c = math.sqrt(a*a + b*b)
print ("c= %.2f" % c)
```

輸出結果為：



import math 載入函式庫 **math**，使用 **math.sqrt** 來計算開方。

G. 函数 (Function)

編程心得
勿將重複的代碼放在程式的地方

函数可以解決上述問題。例如現在要列印任何字串及任何次數，鍵入以下程式及運行

```
def printString( strPrint, n):
    for i in range (0, n):
        print (strPrint)

printString ("Hello World", 5)
```

輸出結果為：

C++ version

```
void printString (char *str, int n)
{
    for (int i=0; i<n; i++){
        printf ("%s", str);
    }
}
```

但現在我們可以改變參數得出不同的結果：

```
printString ("STEM Education Centre", 3)
```

輸出結果為：

```
STEM Education Centre
STEM Education Centre
STEM Education Centre
>>>
```

除了輸出外，有時函数需要返回計算的結果，這時可以使用 **return**

鍵入以下程式及運行

```
def addition(no1, no2):
    answer = no1 + no2
    return answer
a = float(input("a="))
b = float(input("b="))
print (a, "+", b, "= %.2f" % addition(a, b))
```

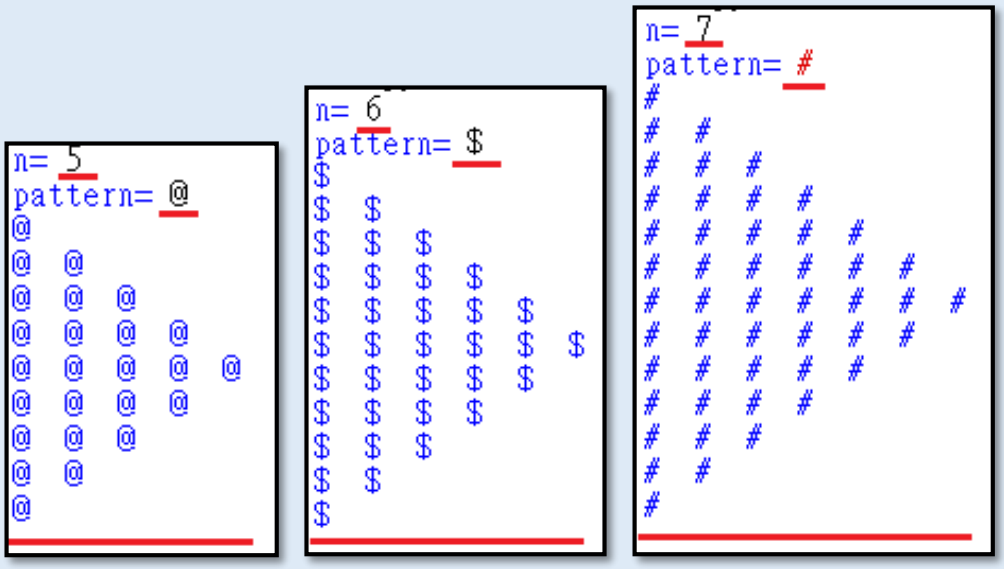
輸出結果為：

```
a=13.5
b=11
13.5 + 11.0 = 24.50
```

H. 練習

Q1

寫一個程式 (printPattern.py)：讀取一個整數及符號，然後輸出以下圖案：



Q2

寫一個程式 (factorial.py)：讀取一個整數，然後計算該數的階乘(factorial)：

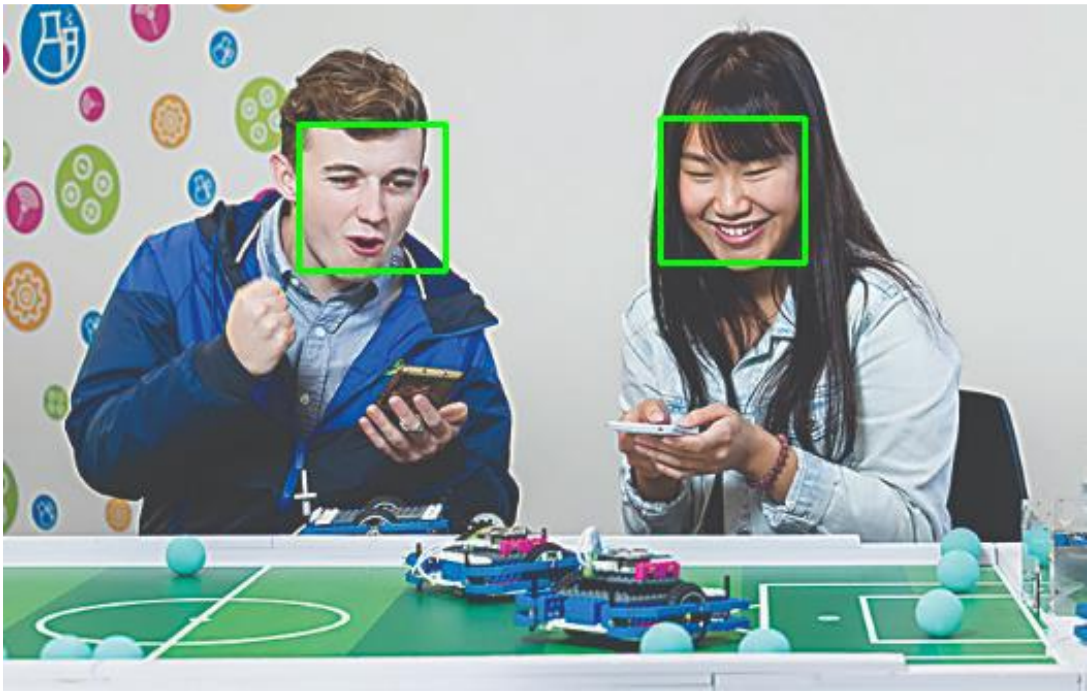
[注意：程式中需要使用遞迴函数(Recursive Function)。]

```
Please enter a number: 4
4 != 24
```

```
Please enter a number: 6
6 != 720
```

```
Please enter a number: 9
9 != 362880
>>>
```

2. 基本影像處理



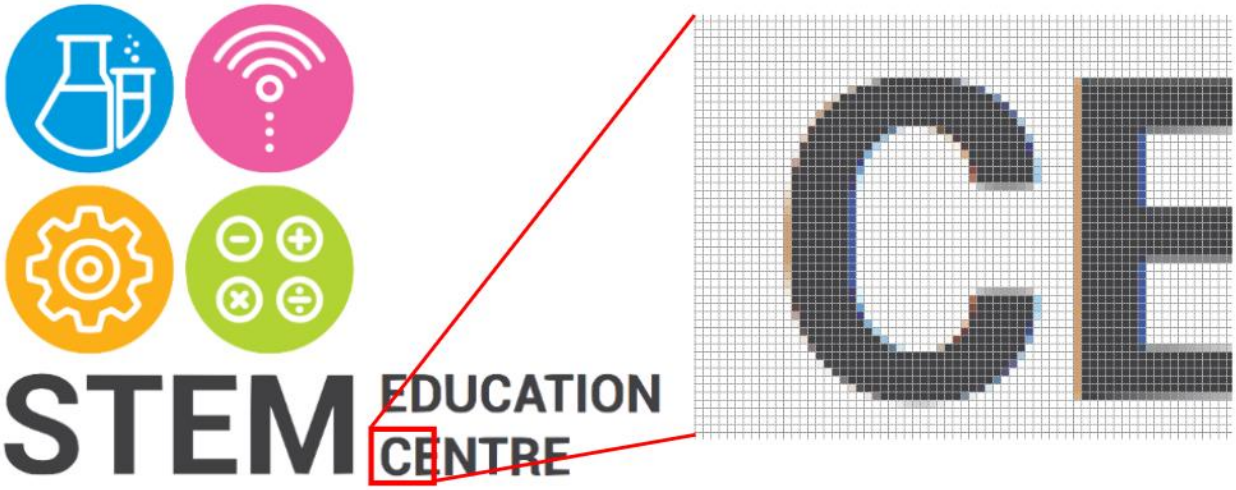
電腦視覺的目的就是為了了解影像與影片的內容，從中提取需要的資訊。人類的視覺系統顯然很容易就做到，但是怎樣可以令電腦做到同樣的事呢？

人類的眼睛會捕捉環境的光度，顏色，形象等等，對於影像中的顯點（Salient Point）十分敏銳。例如見到一個人面上有一粒痣，下次見到同樣的情況時，馬上就能把兩者連繫起來。當我們見到面前的筆記本時，馬上就可以辨認出來。但是另一方面，電腦就很難完成類似的工作。

要令電腦也能做到近似人類視覺的功能，就要令電腦能夠在畫面中抽取一些特質，讓電腦能透過機器學習演算法學會這些特質，並且對影像中的物體進行放大，遮蔽等等的影像處理。因此，我們需要建立一個有效率的函式庫，透過不同的數學運算和複雜的電腦演算法，完成影像處理和操作所需要的基元（Primitive），同時對執行速度和記憶體做了最佳化。

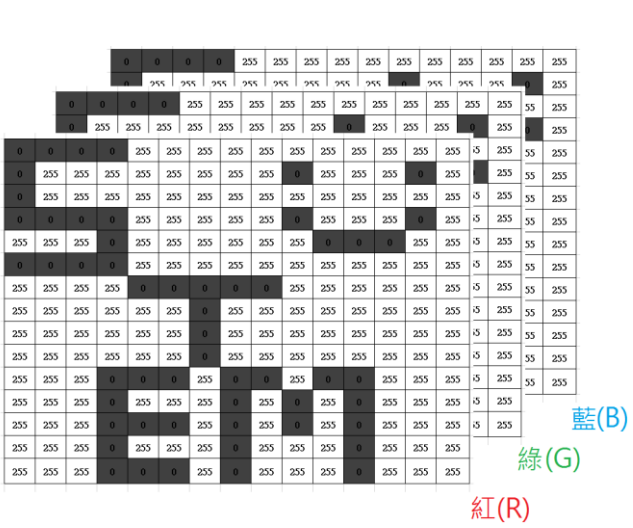
其中 OpenCV 就提供了許多這樣的功能，它是開發電腦視覺應用程式時最受歡迎的函式庫，能夠即時執行很多不同的電腦視覺演算法，現時已經成為了電腦視覺領域的標準。因為函式庫的高度最佳化，它能夠讓人們能夠即時，有效率地執行電腦視覺演算法。

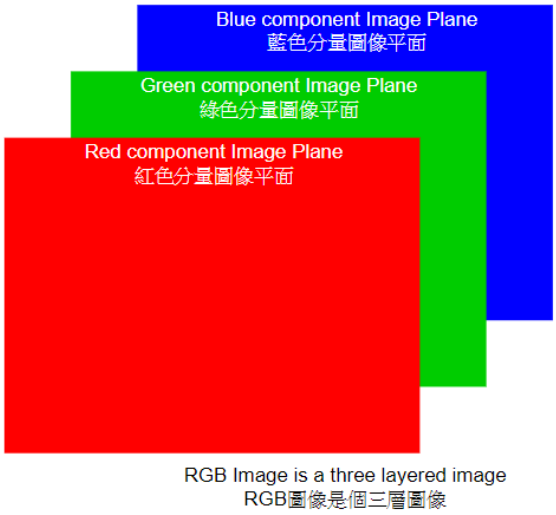
2-1 像素與影像



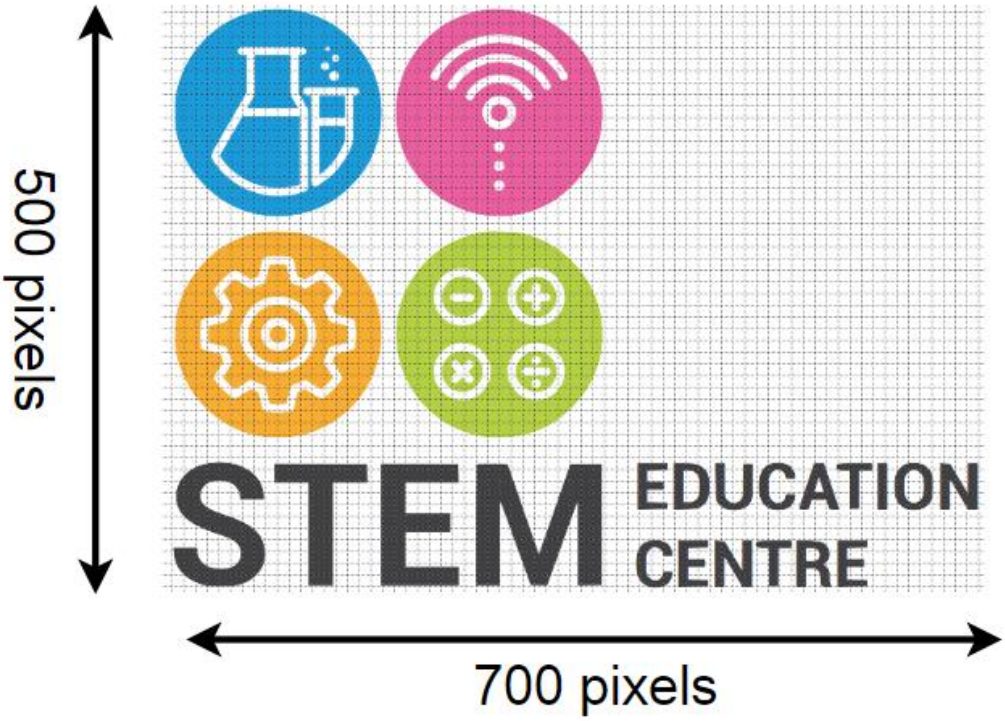
在電腦視覺中，影像是最重要的元素。它可以是從數位設備，例如網絡鏡頭，捕捉現實世界的影像，並且轉換成數碼資訊。如下圖所示，影像會以矩陣的格式儲存在電腦，每個數字代表特定波長的光線強度。圖片中每個點稱為像素（Pixel），對於灰階的影像每個像素只需要儲存一個數值，通常是在 0 到 255 之間的整數。

0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
0	255	255	255	255	255	255	255	255	0	255	255	255	0	255
0	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	255	255	255	255	255	0	255	255	255	0	255
255	255	255	0	255	255	255	255	255	255	0	0	0	255	255
0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	0	0	0	0	0	255	255	255	255	255	255
255	255	255	255	255	255	0	255	255	255	255	255	255	255	255
255	255	255	255	255	255	0	255	255	255	255	255	255	255	255
255	255	255	255	255	255	0	255	255	255	255	255	255	255	255
255	255	255	0	0	0	255	0	0	255	0	0	255	255	255
255	255	255	0	255	255	255	0	255	0	255	0	255	255	255
255	255	255	0	0	0	255	0	255	0	255	0	255	255	255
255	255	255	0	255	255	255	0	255	255	255	0	255	255	255
255	255	255	0	0	0	255	0	255	255	255	0	255	255	255





因為彩色的影像可以從紅（R），綠（G），藍（B）這三原色組成，所以彩色圖片每個點（像素）需要儲蓄三個數值，分別儲存紅／綠／藍的光線強度。每個像素都有其位置，並且以行，列的數值作定位，對於灰階的圖片只需使用一個長 x 寬的矩陣，而彩色的影像即需要長 x 寬 x 顏色數目的矩陣。



2-2 基本讀取、顯示與儲存圖片

學過基本影像矩陣的概念後，這部分會介紹如何使用 Python 與 OpenCV 讀取影像圖檔，以及儲存處理好的圖形。

讀取圖片

首先引入 NumPy 與 OpenCV 的 Python 模組：

```
1. import numpy as np
2. import cv2
```

讀取一般的圖片檔，只要呼叫 cv2.imread 即可將圖片讀取進來，並且儲存成一個 NumPy 的陣列。此 NumPy 陣列的前兩個維度分別是圖片的高度與寬度，第三個維度則是圖片的 channel（RGB 彩色圖片的 channel 是 3，灰階圖片則為 1）。

```
3. # 讀取圖檔
4. img = cv2.imread('image.jpg')
```

在讀取圖片時，可以在第二個參數指定圖片的格式

```
5. # 以灰階的方式讀取圖檔
6. img_gray = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
```

顯示圖片

圖片讀取進來之後，可以使用 cv2.imshow 來顯示圖片。

```
7.  # 顯示圖片
8.  cv2.imshow('My Image', img)
```

使用 cv2.waitKey 函數等待並讀取按下的按鍵，其參數是等待時間，若設定為 0 ，就持續等待至使用者按下任意按鍵為止，並且關閉所有視窗。

```
9.  # 按下任意鍵則關閉所有視窗
10. cv2.waitKey(0)
11. cv2.destroyAllWindows()
```

(如果希望可以自由縮放視窗的大小，可以使用 cv2.namedWindow 設定視窗為 cv2.WINDOW_NORMAL)

```
# 讓視窗可以自由縮放大小
cv2.namedWindow('My Image', cv2.WINDOW_NORMAL)
cv2.imshow('My Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

儲存圖片

儲存圖片可以使用 cv2.imwrite 來儲存檔案

```
12. # 儲存圖檔
13. cv2.imwrite('output.jpg', img)
```

完整程式碼

```
1.  import numpy as np
2.  import cv2
3.  # 讀取圖檔
4.  img = cv2.imread('image.jpg')
5.  # 以灰階的方式讀取圖檔
6.  img_gray = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
7.  # 顯示圖片
8.  cv2.imshow('My Image', img)
9.  # 按下任意鍵則關閉所有視窗
10. cv2.waitKey(0)
11. cv2.destroyAllWindows()
12. # 儲存圖檔
13. cv2.imwrite('output.jpg', img)
```

2-3 擷取網路攝影機串流影像

在擷取攝影機的影像之前，要先呼叫 cv2.VideoCapture 並建立一個 VideoCapture 物件，這個物件會連接到一隻網路攝影機，後面的參數可以指定要使用那一隻攝影機（0 代表第一、1 代表第二）。

擷取影像

```
3.  cap = cv2.VideoCapture(0)
```

建立好 VideoCapture 物件之後，就可以使用它的 read 函數來擷取一張張連續的畫面影像了，以下是程式的範例：

完整程式碼

```
1.  import cv2
2.  # 選擇第一個攝影機
3.  cap = cv2.VideoCapture(0)
4.
5.  while (cap.isOpened()):
6.    #從攝影機擷取一張影像
7.    ret, image = cap.read()
8.
9.    # 顯示圖片
10.   cv2.imshow('Video Capture', image)
11.
12.   # 若按下 q 鍵則離開迴圈
13.   if cv2.waitKey(1) & 0xFF == ord('q'):
14.     Break
15.
16. # 釋放攝影機資源
17. cap.release()
18. # 關閉所有 OpenCV 視窗
19. cv2.destroyAllWindows()
```

在這迴圈中，每次呼叫 cap.read() 就會讀取一張畫面，其第一個傳回值 ret 代表成功與否，而第二個傳回值 frame 就是攝影機擷取的單張畫面。

2-4 加入線條圖案與文字

在影像處理的程式中，時常會需要在圖片上加上一些標示的圖形或是文字，用來呈現處理的結果。例如在人面偵測後，我們會使用方框將偵測到的人面框起來，並加註一些文字描述。

直線

若要加入直線，可以使用 cv2.line 函數

cv2.line(影像, 開始座標, 結束座標, 顏色, 線條寬度)

```
#  在圖片上畫一條紅色的對角線， 寬度為 5 px
cv2.line(img, (0, 0), (255, 255), (0, 0, 255), 5)
```

方框

繪製方框可以使用 cv2.rectangle 函數，線條寬度參數若設定為正的值，則代表正常的線條寬度，而若設定為負的值，則代表畫實心的方框：

cv2.rectangle(影像, 頂點座標, 對向頂點座標, 顏色, 線條寬度)

```
#  在圖片上畫一個綠色方框， 線條寬度為 2 px
cv2.rectangle(img, (20, 60), (120, 160), (0, 255, 0), 2)
# 綠色實心方框
cv2.rectangle(img, (40, 80), (100, 140), (0, 255, 0), -1)
```

圓圈

繪畫圓圈可以使用 cv2.circle 函數：

cv2.circle(影像, 圓心座標, 半徑, 顏色, 線條寬度)

```
#  黃色圓圈， 線條寬度為 3 px
cv2.circle(img, (90, 210), 30, (0, 255, 255), 3)
#  藍色實心圓圈
cv2.circle(img, (140, 170), 15, (255, 0, 0), -1)
```

文字

若要在圖片上加上文字，可以使用 cv2.putText 函數：

cv2.putText(影像, 文字, 座標, 字型, 大小, 顏色, 線條寬度, 線條種類)

```
#  文字字串
text = 'Hello world!'
#  在圖片上加上文字
cv2.putText(img, text, (10, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 1, cv2.LINE_AA)
```

畫布程式碼

```
1.  import cv2
2.  import numpy as np
3.  # Create a blank canvas with w=512, h=512
4.  img = np.ones((512,512,3))
5.
6.  ### Code at here: ###
7.
8.  cv2.imshow("Canvas", img)
9.  cv2.waitKey(0)
10. cv2.destroyAllWindows()
```

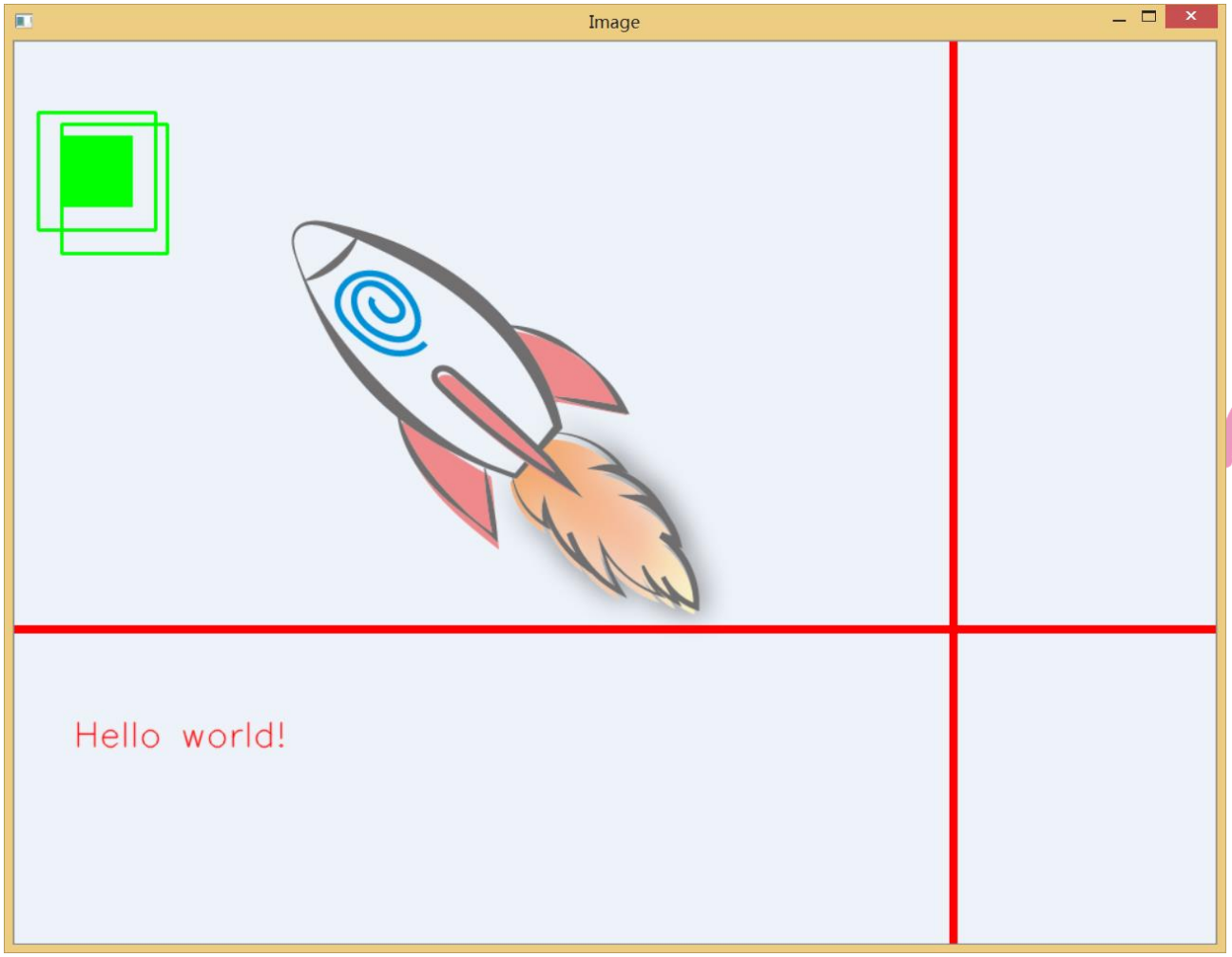
實驗一

1） 讀取 rocket.png 圖像，

2） 並加上線條，方塊和文字，

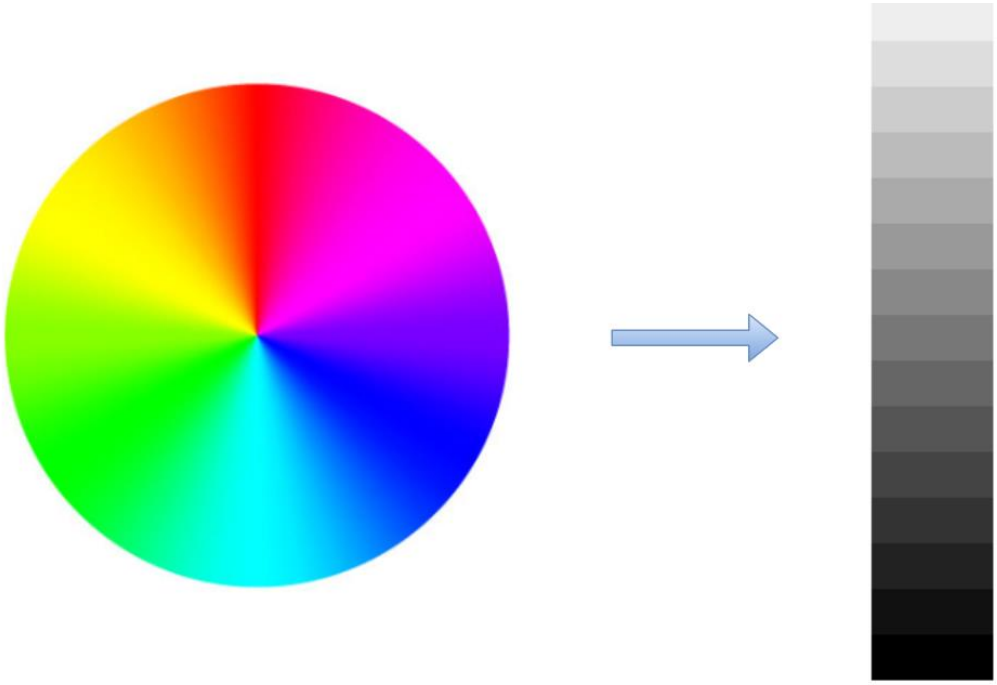
3） 最後顯示並儲存完成圖片

參考：



2-5 色彩空間轉換

OpenCV 的 cvtColor()可以令影像在不同色彩空間之中轉換，要主要的是 OpenCV 的內存為 BGR 而不是 RGB 格式，影像是由 BGR 色彩空間轉到其他色彩空間，或從其他色彩空間轉到 BGR 色彩空間。



Grayscale image = ((0.3 * R) + (0.59 * G) + (0.11 * B))

灰階圖像

色彩空間轉換

要轉換為灰階圖像可使用 cv2.cvtColor 函數：

cv2.cvtColor(輸入影像, 轉換類型)

首先我們導入 cv2 模塊，並將圖像讀取到名為“image”的變量

```
1. import cv2
2. image = cv2.imread('testImg.png')
```

現在，要轉換為灰階圖像並將其存儲到另一個名為“gray_image”的變量：

```
3. gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

使用函數“cv2.imwrite（）”把轉換後的圖像存儲到硬盤

```
4. cv2.imwrite('gray_image.png', gray_image)
```

為了顯示原始和灰階的影像，我們使用帶有參數的函數“cv2.imshow（）”作為“窗口標題”和“圖像變量”：

```
5. cv2.imshow('color_image', image)
6. cv2.imshow('gray_image', gray_image)
7. # Waits forever until press any key
8. cv2.waitKey(0)
9. # Closes displayed windows
10. cv2.destroyAllWindows()
```

2-6 影像尺寸改變 (Resize)

如果只是單純的影像縮小放大，可以使用 resize()函式。

影像縮小放大

cv2.resize(輸入影像, (長, 寬), 插值方法)

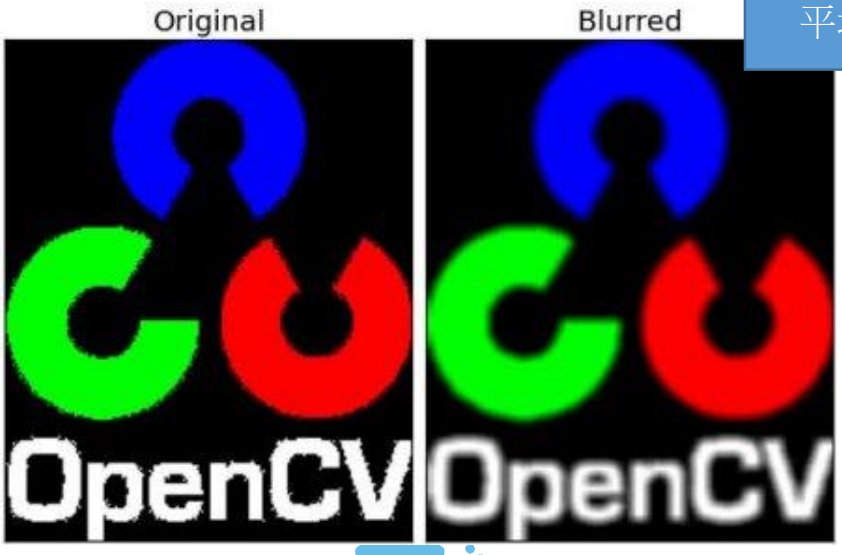
```
1. import cv2
2. image = cv2.imread('testImg.png')
3. resizedImg = cv2.resize(image, (400, 400), interpolation = cv2.INTER_CUBIC)
4. cv2.imshow('Resized Image', resizedImg)
5. cv2.waitKey(0)
6. cv2.destroyAllWindows()
```

2-7 影像平滑 (平均平滑 blur、高斯平滑 GaussianBlur)

有時我們收到的影像雜訊過多，就需要進行平滑化去除雜訊，常見的有四種平滑方式，分別是平均平滑、高斯平滑、中值濾波、雙邊濾波。

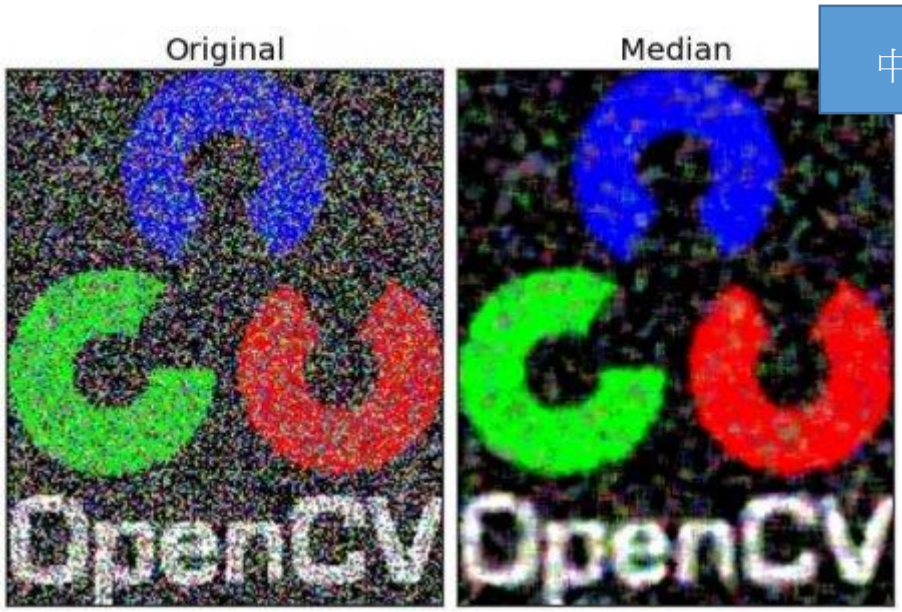
$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

平均平滑



圖片來源:https://opencv-python-tutorials.readthedocs.io/

中值濾波



圖片來源:https://opencv-python-tutroals.readthedocs.io/

平均平滑 Averaging Filtering

cv2.blur(輸入影像, (核心大小))

高斯平滑 Gaussian Filtering

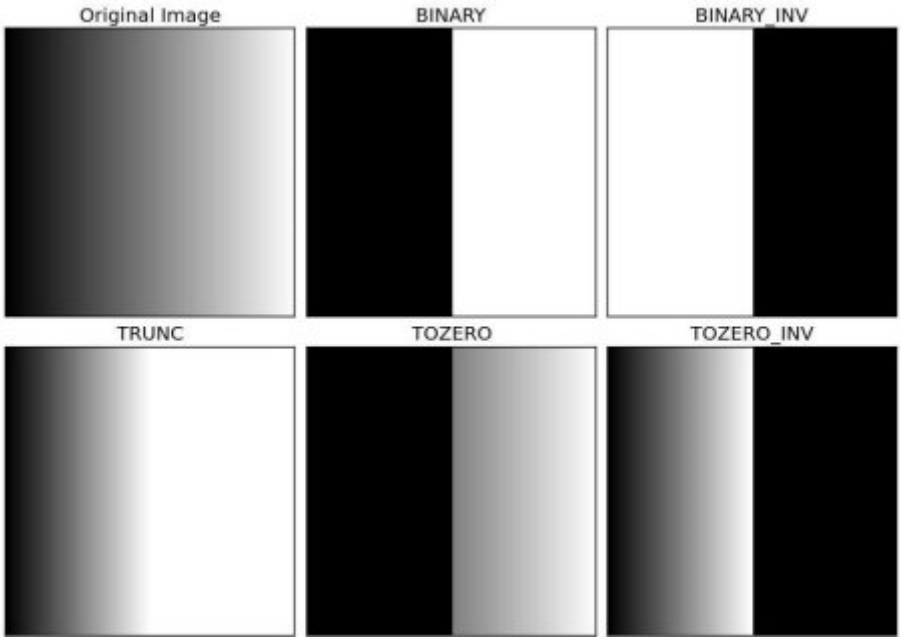
cv2.GaussianBlur(輸入影像, (核心大小), 類型)

```
1.  import cv2
2.  import numpy as np
3.  image = cv2.imread('testImage.png')
4.  cv2.imshow("Original ", image)
5.
6.  AvgBlur = cv2.blur(image, (5, 5))
7.  cv2.imshow("Averaged blur ", AvgBlur)
8.
9.  GausBlur = cv2.GaussianBlur(image, (5, 5), 0)
10. cv2.imshow("Gaussian Blur ", GausBlur)
11. cv2.waitKey(0)
12. cv2.destroyAllWindows()
```

2-8 二值化

在進行影像處理時，為了方便及明確的分離目標物，或者為了減少圖片維度，我們會將一張彩色圖片轉為灰階後，再轉為黑白來處理。

threshold 指令讓我們執行二值化處理，它需要四個參數：第一個是待處理的相片。第二個是 threshold (T 值)，第三個是最大值，即當某點超過 T 值時則設為該最大值，由於我們想要全黑白相片，故此值設為 255。第四個為 Thresholding 的處理方式，表示計算後輸出後影像的呈現型態。



圖片來源: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html

二值化

(T, thresh) = cv2.threshold(輸入影像, 閾值, 最大值, cv2.THRESH_BINARY)

完整程式碼

```
1.  import cv2
2.  import numpy as np
3.  image = cv2.imread('testImg.png')

4.  #讀取相片並裁切需要的區域 format-> Y1: Y2 X1: X2
5.  cropped = image[100: 600, 920: 1490]
6.  cv2.imshow("Cropped", cropped)
7.  #將裁切的影像轉為灰階，再使用 Gaussian Blurring 將其模糊化，減少相片中複雜的物體，讓它們較為圓滑一
    致。
8.  imgTest = cv2.cvtColor(cropped, cv2.COLOR_BGR2GRAY)
9.  imgTestBlur = cv2.GaussianBlur(imgTest, (11, 11), 0)
10. #若超過 T 值，則將該點設為最大值，否則為 0
11. (T, thresh) = cv2.threshold(imgTestBlur, 35, 255, cv2.THRESH_BINARY)
12.
13. cv2.imshow(“Threshold Binary ", thresh)
14. cv2.waitKey(0)
15. cv2.destroyAllWindows()
```

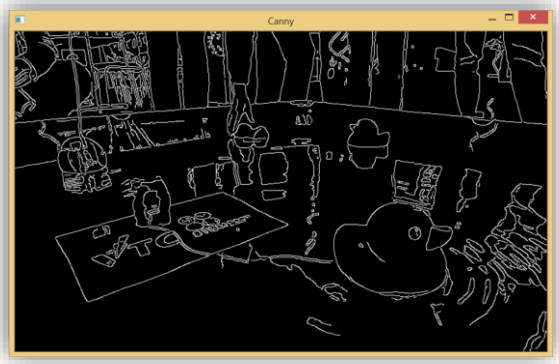
2-9 找邊緣 (Canny)

圖像邊緣是一個重要特徵，邊緣檢測是影像處理的一個重要的元素，能夠減少數據量，剔除了不重要的信息，保留圖像重要的結構屬性。第一個參數是需要處理的原圖像，該圖像必須為單通道的灰階圖；第二和第三個參數是閾值，用於檢測圖像中明顯的邊緣，最後把斷斷續續的邊緣連接。

邊緣

canny = cv2.Canny(灰階影像, 低閾值, 高閾值)

```
1.  import cv2
2.  import numpy as np
3.
4.  img = cv2.imread('duck.png')
5.  img = cv2.GaussianBlur(img, (3, 3), 0)
6.
7.  canny = cv2.Canny(img, 30, 100)
8.
9.  cv2.imshow('Canny', canny)
10. cv2.waitKey(0)
11. cv2.destroyAllWindows()
```

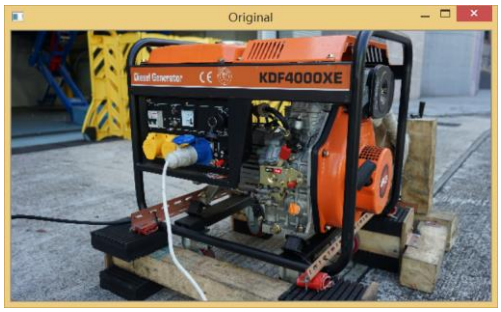


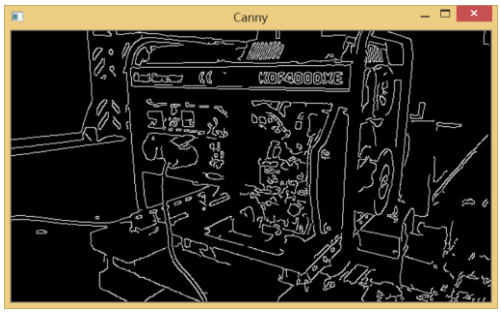
實驗二

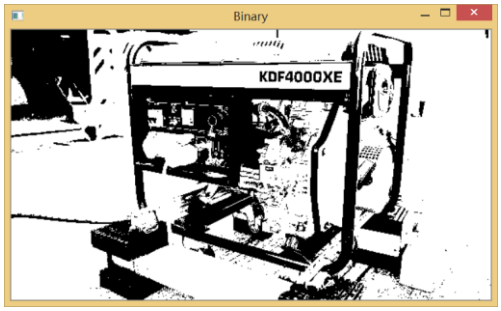
1） 讀取 engine.png 圖像，

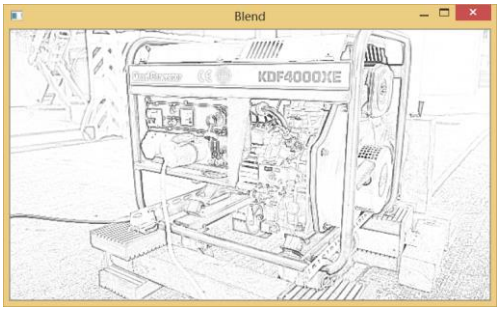
2） 轉換成以下色彩的圖像，

3） 最後顯示並儲存完成圖片









Blend 提示: imgBlend = cv2.divide(imgGrayscale, imgBlurred, scale=256)

其他有趣色彩轉換：

https://www.pyimagesearch.com/2014/06/30/super-fast-color-transfer-images/

2-10 輪廓 (findContours、drawContours)

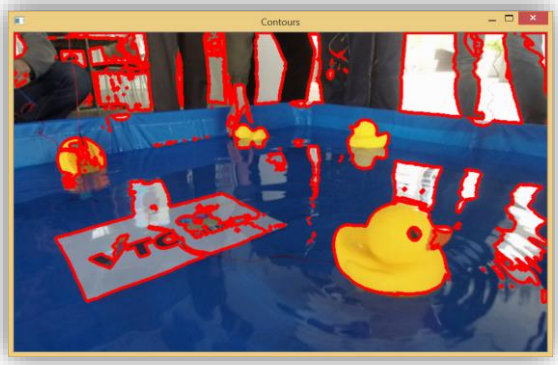
當我們做物件辨識時，透過輪廓可得到特定物件的資訊，協助我們做判斷。而輪廓和找邊緣的處理不同，findContours 是在經過邊緣處理之後，將只有邊緣的影像的各個邊緣點做分類，連結的邊緣點儲存在相同的容器內，當我們找到輪廓後，可用 drawContours()劃出輪廓線，檢查是否取得正確或適合的輪廓。

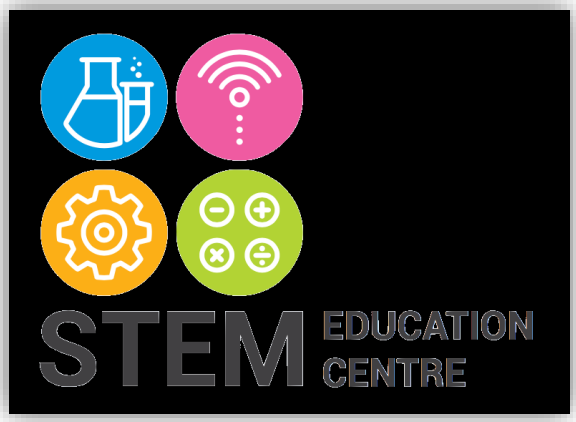
第一個參數是尋找輪廓的二值化圖像； 第二個參數表示輪廓的檢索模式，第三個參數 method 為輪廓的近似辦法存儲所有的輪廓點。函數返回兩個值，一個是輪廓本身（contours），並以 list 儲存，list 中每個元素都是圖像中的一個輪廓，另外還有一個（hierarchy）是每條輪廓對應的屬性。

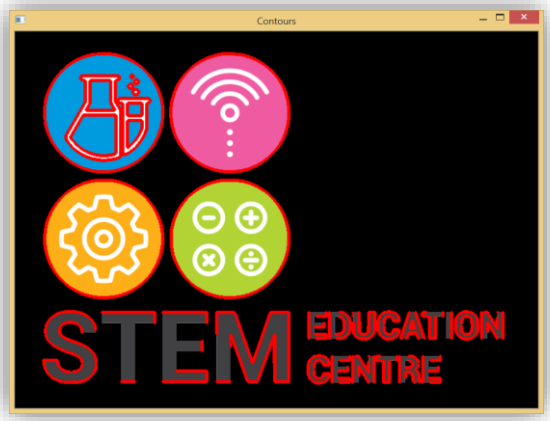
輪廓

contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

```
1.  import cv2
2.  img = cv2.imread('duck.png')
3.  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
4.  ret, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
5.
6.  contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
7.  cv2.drawContours(img, contours, -1, (0, 0, 255), 3)
8.
9.  cv2.imshow("Contours", img)
10. cv2.waitKey(0)
11. cv2.destroyAllWindows()
```



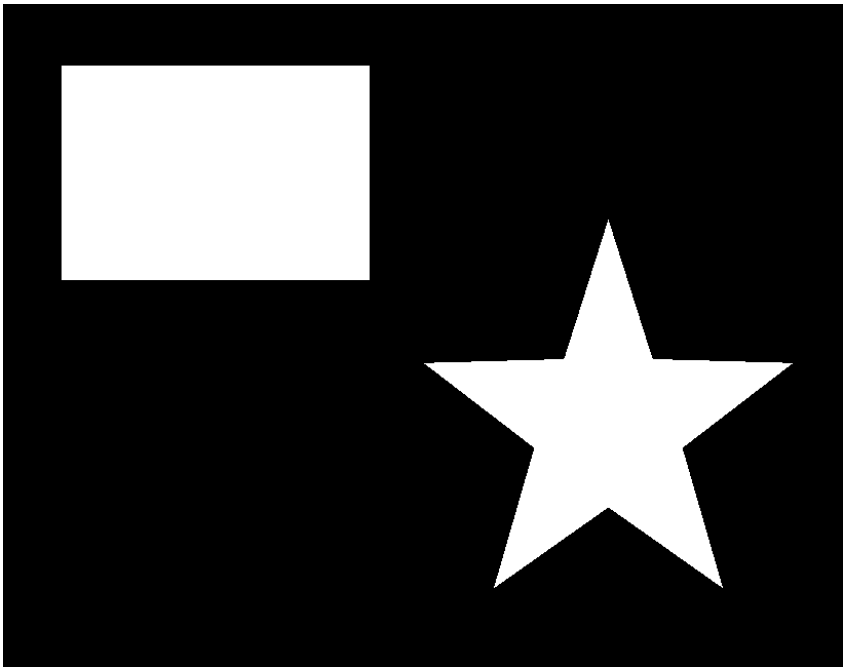




對比色越高的時候，輪廓會越明顯。所以大部份的圖像處理都需要大量的預先處理程序，所以我們都會先對圖像做一些預先程序。例如會先把圖像轉成黑白，再做閾值（threshold），邊緣（canny） 等邊緣檢測處理，能提高輪廓的辨識效果。

2-11 結構分析和形狀描述

輪廓的線條可以簡單地解釋為連接具有相同顏色或強度的所有連續點（沿邊界）的曲線。運用 FindContour 的指令，我們可以找出圖像的所有緣檢。



完整程式碼

```
1.  import numpy as np
2.  import cv2
3.  # Read the image and convert to gray tone
4.  img = cv2.imread('polygon.png')
5.  imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
6.  # Find the contours
7.  ret,thresh = cv2.threshold(imgray,127,255,0)
8.  contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
9.
10. print ("there are " + str(len(contours)) + " contours")
11. # The first polygon
12. cnt = contours[0]
13. print ("there are " + str(len(cnt)) + " points in contours[0]")
14. print (cnt)
```

```
15.  # The second polygon
16. cnt = contours[1]
17. print ("there are " + str(len(cnt)) + " points in contours[1]")
18. print (cnt)
19.
20. cv2.imshow('Image', img)
21. cv2.waitKey(0)
22. cv2.destroyAllWindows()
```

輸出結果

```
C:\Users\STEM\Documents\Python workshop>python polygon.py
there are 2 contours
there are 579 points in contours[0]
[[[589 210]]

 [[589 212]]

 [[588 213]]

 ...

 [[590 215]]

 [[590 213]]

 [[589 212]]]
there are 4 points in contours[1]
[[[ 57  60]]

 [[ 57 268]]
```

從輸出結果可以發現程式能夠找到 2 個輪廓，分別有 4 個和 579 個點(座標串列)，這是因為輪廓的細微毛邊或雜點都被程式掃瞄出來。這時候就可以對輪廓做多邊形逼近（approxPolyDP），用粗一點的線來描述邊線，以忽略掉細微的毛邊或雜點。

多邊形逼近（approxPolyDP）

cv2.approxPolyDP(curve, epsilon, closed[, approxCurve])

完整程式碼

```
1.  import numpy as np
2.  import cv2
3.  # Read the image and convert to gray tone
4.  img = cv2.imread('polygon.png')
5.  imgray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
6.  # Find the contours
7.  ret,thresh = cv2.threshold(imgray,127,255,0)
8.  contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
9.
10. print ("there are " + str(len(contours)) + " contours")
11. # The first polygon
12. cnt = contours[0]
13. print ("there are " + str(len(cnt)) + " points in contours[0]")
14. approx = cv2.approxPolyDP(cnt,30,True)
15. print ("after approx, there are " + str(len(approx)) + " points")
16. print (approx)
17.
18.  # The second polygon
19. cnt = contours[1]
20. print ("there are " + str(len(cnt)) + " points in contours[1]")
21. approx = cv2.approxPolyDP(approx,30,True)
22. print ("after approx, there are " + str(len(approx)) + " points")
23. print (approx)
24.
25. cv2.imshow('Image', img)
26. cv2.waitKey(0)
27. cv2.destroyAllWindows()
```

輸出結果

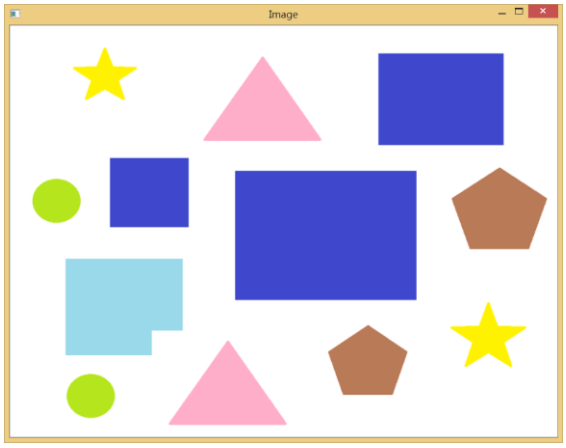
```
C:\Users\STEM\Documents\Python workshop>python polygon.py
there are 2 contours
there are 579 points in contours[0]
after approx, there are 10 points
[[[589 210]]

 [[545 346]]

 [[410 350]]

 [[517 432]]

 [[478 568]]

 [[588 490]]

 [[700 568]]

 [[661 432]]

 [[768 350]]

 [[633 346]]]
there are 4 points in contours[1]
after approx, there are 4 points
[[[ 57  60]]

 [[ 57 268]]

 [[356 268]]

 [[356  60]]]
```

實驗三

1） 讀取 pattern.png 圖像，

2） 用 drawContours 標示出所有三角形，

3） 最後計算出三角形的面積



提示:

1) ret, binary = cv2.threshold(imgBlurred, 230, 255, cv2.THRESH_BINARY)

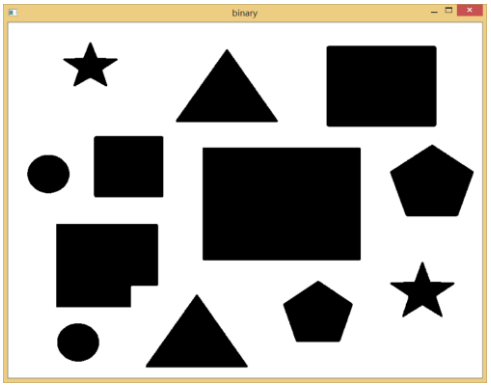
2) for cnt in contours:

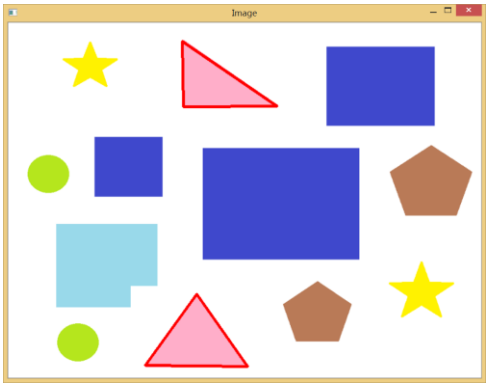
3) if len(approxVertex)==3:

4) approxVertex = cv2.approxPolyDP(cnt,0.01*cv2.arcLength(cnt,True),True)

5) area = cv2.contourArea(cnt)

輸出結果:



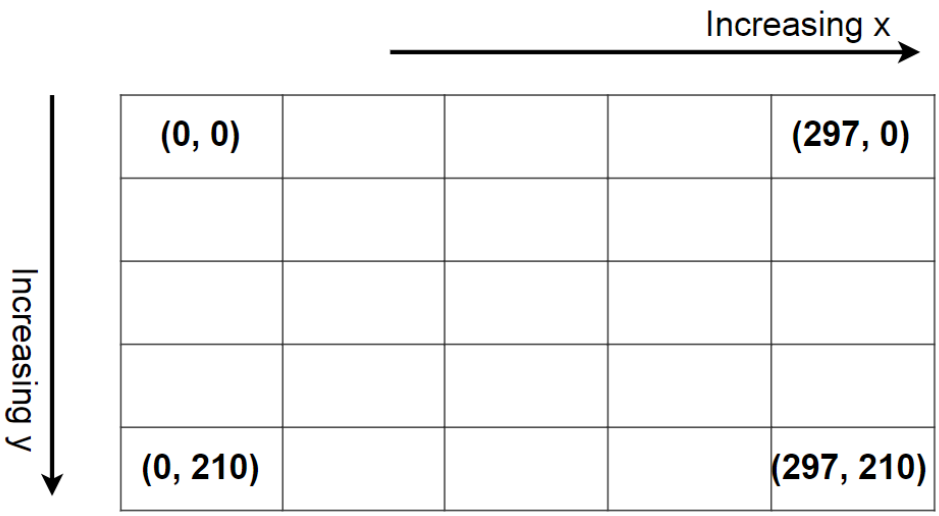



```
C:\Users\STEM\Documents\Python workshop\Lab\Lab3>python lab3_ans.py
Total Numbers of contours: 13
There are 150 points in contour 1
There are 3 vertics in contour 1
The area is:  11660.5  in contour

There are 108 points in contour 2
There are 3 vertics in contour 2
The area is:  10043.0  in contour
```

2-12 透視變換(Perspective Transform)

透視變換是我們要更改對象的透視時所使用的操作。例如用相機拍攝桌上紙，離相機較近的部分比距離更遠的大。所以當我們要對平面圖像作分析前，可以使用透視變換的變換功能。

要留意的是，在 OpenCV 的坐標系統中，左上角是(0,0)，越近右下角數值越大。所以只要找到圖案中 4 個角的坐標，便可以利用數學方法把原來扭曲的圖案轉換到新的坐標系統。



完整程式碼

```
1.  import cv2
2.  import numpy as np
3.  # load the image we want to transform
4.  img = cv2.imread('perspective.png')
5.  img = cv2.resize(img, None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
6.
7.  # We need to select 4 points, in order: top-left, top-right, bottom-left, bottom-right
8.  cv2.circle(img, (243, 24), 5, (0, 0, 255), -1)
9.  cv2.circle(img, (116, 267), 5, (0, 0, 255), -1)
10. cv2.circle(img, (572, 408), 5, (0, 0, 255), -1)
11. cv2.circle(img, (633, 124), 5, (0, 0, 255), -1)
12.
13. # The points to apply the transformation
14. old_pts = np.float32([[243, 24], [116, 267], [572, 408], [633, 124]])
15.
16. # The points (size of windows) to display the image transformed
17. # The size of an A4 paper is 297 x 210 mm
18. new_pts = np.float32([[0, 0], [0, 210], [297, 210], [297, 0]])
19.
20. # Apply the perspective transform to create the matrix
21. matrix = cv2.getPerspectiveTransform(old_pts, new_pts)
22.
23. # Warp the image into using the original frame and the matrix
24. result = cv2.warpPerspective(img, matrix, (297, 210))
25.
26. # Show it on the screen
27. cv2.imshow("Image", img)
28. cv2.imshow("Perspective transformation", result)
29. cv2.waitKey(0)
30. cv2.destroyAllWindows()
```

可是，在以上的程式需要人手找出 4 個點(坐標)來作出變換，事實上我們可以透過運算計出這 4 個頂點。

在程式的開始部分加上搜尋頂點的函数。目的要尋找有 4 個頂點，而且面積又是最大的輪廓。函数程式先假定第 0 個輪廓最大，然後拿這個和後面的輪廓比，找到大的就更新面積，最後會回傳 4 個坐標。

```
12. # Find the largest contour with 4 vertix
13. def find_vertix(contours):
14.     cnt = contours[0]
15.     max_area = cv2.contourArea(cnt)
16.
17.     for cont in contours:
18.         pts = cv2.approxPolyDP(cont,30,True)
19.         # If the contours have 4 vertix
20.         if len(pts) == 4:
21.             # Find the biggest contour
22.             if cv2.contourArea(cont) > max_area:
23.                 points = pts
24.                 max_area = cv2.contourArea(cont)
25.     return points
26.
27. vertix = find_vertix(contours)
28. # Since the contours returned are wrapped by 3 taples (x,y,z),
29. # it can be simplified to a 4 by 2 matrix (x,y)
30. reshapedVertix = vertix.reshape(4, 2)
```

以下的程式自動的計算出 4 個頂點，並且用紅圈標示出來。

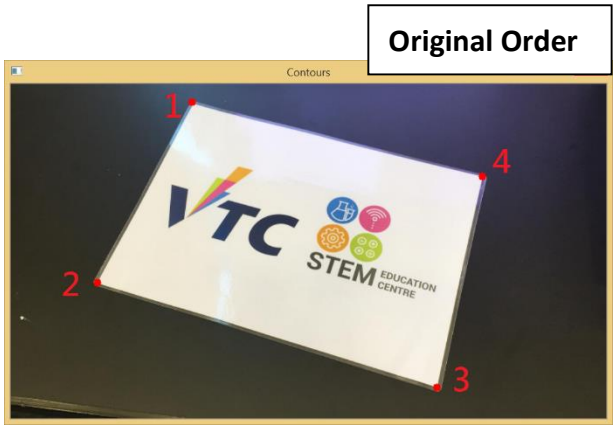
完整程式碼

```
1.  import cv2
2.  import numpy as np
3.  # load the image we want to transform
4.  img = cv2.imread('perspective.png')
5.  img = cv2.resize(img, None,fx=0.5, fy=0.5, interpolation = cv2.INTER_CUBIC)
6.
7.  # Find the contours
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Find the largest contour with 4 vertix
def find_vertix(contours):
    cnt = contours[0]
    max_area = cv2.contourArea(cnt)

    for cont in contours:
        pts = cv2.approxPolyDP(cont,30,True)
        # If the contours have 4 vertix
        if len(pts) == 4:
            # Find the biggest contour
            if cv2.contourArea(cont) > max_area:
                points = pts
                max_area = cv2.contourArea(cont)
    return points

vertix = find_vertix(contours)
# Since the contours returned are wrapped by 3 taples (x,y,z),
# it can be simplified to a 4 by 2 matrix (x,y)
reshapedVertix = vertix.reshape(4, 2)

# Label the vertix on the image
cv2.circle(img, (reshapedVertix[0][0], reshapedVertix[0][1]), 5, (0, 0, 255), -1)
cv2.circle(img, (reshapedVertix[1][0], reshapedVertix[1][1]), 5, (0, 0, 255), -1)
cv2.circle(img, (reshapedVertix[2][0], reshapedVertix[2][1]), 5, (0, 0, 255), -1)
cv2.circle(img, (reshapedVertix[3][0], reshapedVertix[3][1]), 5, (0, 0, 255), -1)

cv2.imshow("Contours", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

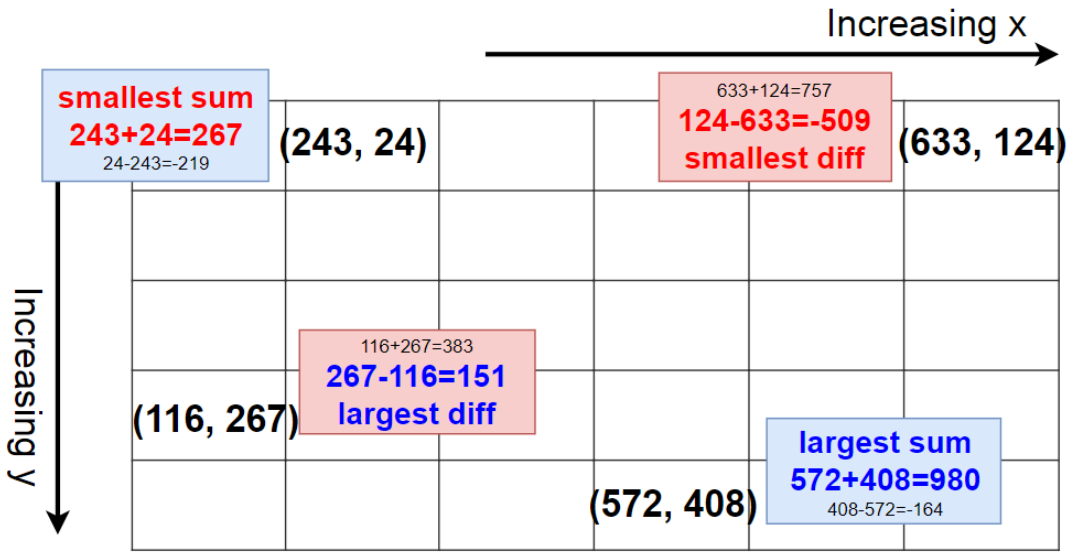
2-13 坐標換算

在矩形中具有一致的點順序十分重要，而實際順序本身可以是任意的，只要在整個實施過程中保持一致即可。用輪廓程式找出來的點(座標)是以逆轉方向標示的，事實上我們也可以根據需要改變點順序。




```
C:\Users\STEM\Documents\Python workshop>python auto_perspect_trans_reorder.py
Original Order:
1: [243  24]   2: [116 267]   3: [572 408]   4: [633 124]

Sum:  [267 383 980 757] Index:  0 2

Diff:  [[-219  151 -164 -509]] Index:  3 1

Re-ordered:
1: [243.  24.] 2: [633. 124.] 3: [572. 408.] 4: [116. 267.]
```

運用之前的程式找出 4 個坐標後，可以利用座標幾何的數式換算出輪廓的上下左右等 4 個方位。例如把 x，y 軸相加後，左上角的座標擁有最少的和，右下角的座標擁有最大的和；把 x，y 軸相減後，右上角的座標擁有最少的差，左下角的座標擁有最大的差。最後把找到的座標按需要放入 rect[ ]數組便可以更改坐標順序。



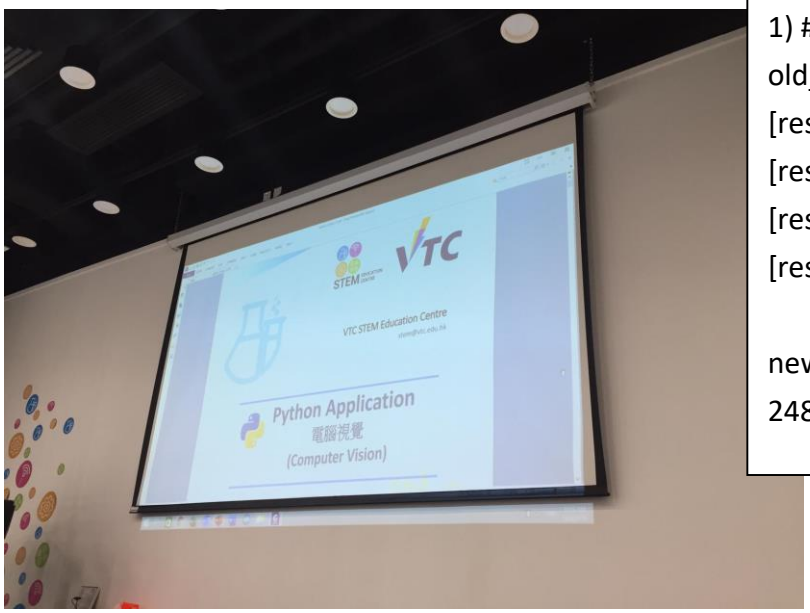

```
25. def order_points(pts):
26.         # initialzie a list of coordinates
27.         # first entry in the list is the top-left, the second entry is the top-right,
28.         # the third is the bottom-right, and the fourth is the bottom-left
29.         rect = np.zeros((4, 2), dtype = "float32")
30.
31.         # the top-left point will have the smallest sum, whereas
32.         # the bottom-right point will have the largest sum
33.         s = np.sum(pts, axis = 1)
34.         print ("Sum: ", s, "Index: ", np.argmin(s),np.argmax(s), end='\n\n')
35.         rect[0] = pts[np.argmin(s)]
36.         rect[2] = pts[np.argmax(s)]
37.
38.         # top-right point will have the smallest difference,
39.         # whereas the bottom-left will have the largest difference
40.         diff = np.diff(pts, axis = 1)
41.         print ("Diff: ", diff.reshape(1,4), "Index: ", np.argmin(diff),np.argmax(diff), en
    d='\n\n')
42.         rect[1] = pts[np.argmin(diff)]
43.         rect[3] = pts[np.argmax(diff)]
44.
45.         # return the ordered coordinates
46.         return rect
```

實驗四

1） 讀取 screen.png 圖像，

2） 用 cv2.circle 標示出銀幕的四個座標，

3） 最後用透視變換把圖形轉換到新的座標



提示:

1) # Set the coordinates

old_pts = np.float32([

[reshapedVertix[0][0],reshapedVertix[0][1]],

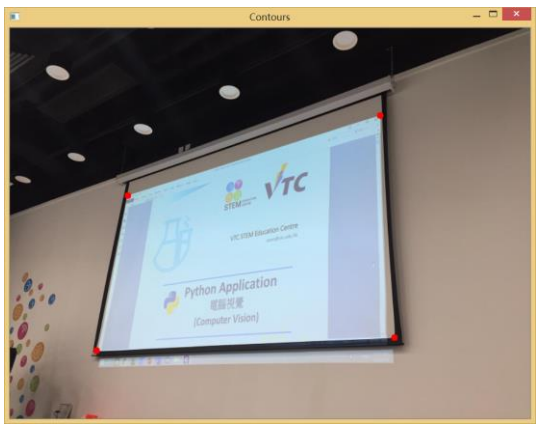
[reshapedVertix[1][0],reshapedVertix[1][1]],

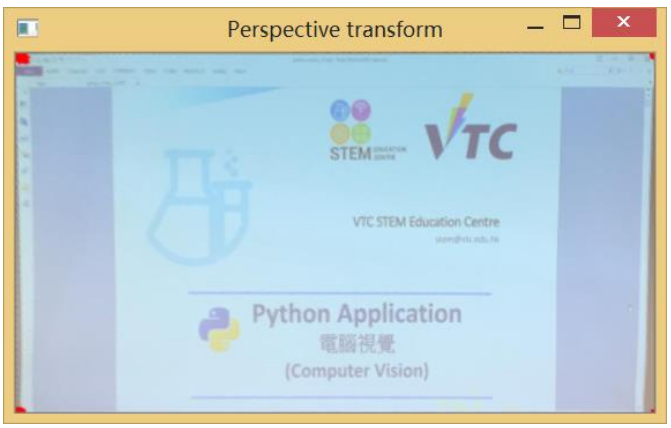
[reshapedVertix[2][0],reshapedVertix[2][1]],

[reshapedVertix[3][0],reshapedVertix[3][1]]])

new_pts = np.float32([[442, 0], [0, 0], [0, 248], [442, 248]])

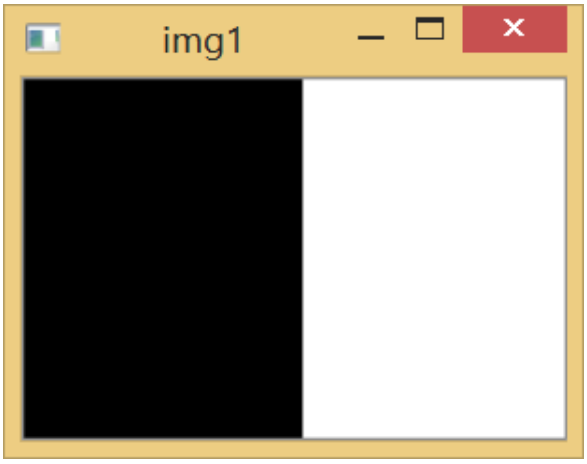
輸出結果:

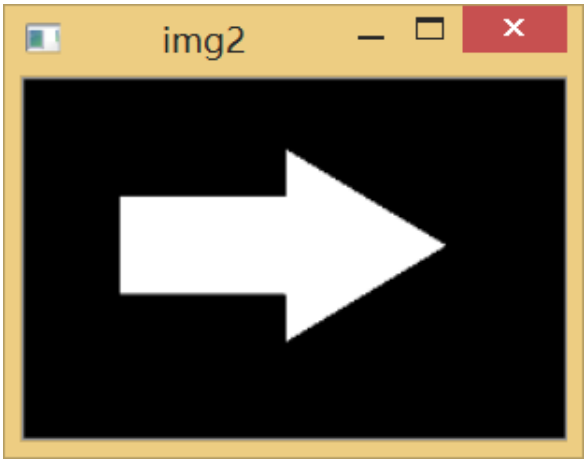


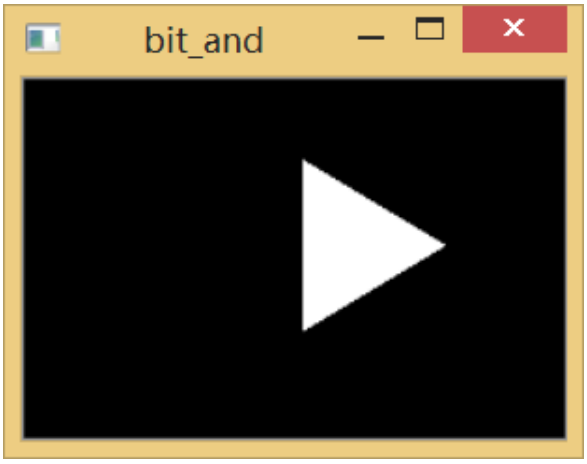


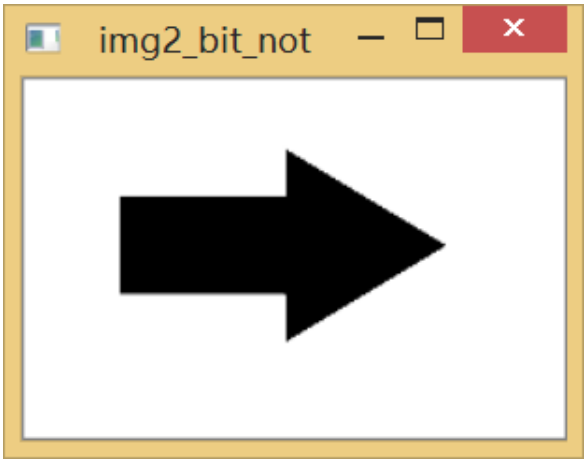
2-14 按位運算符 (Bitwise operator)

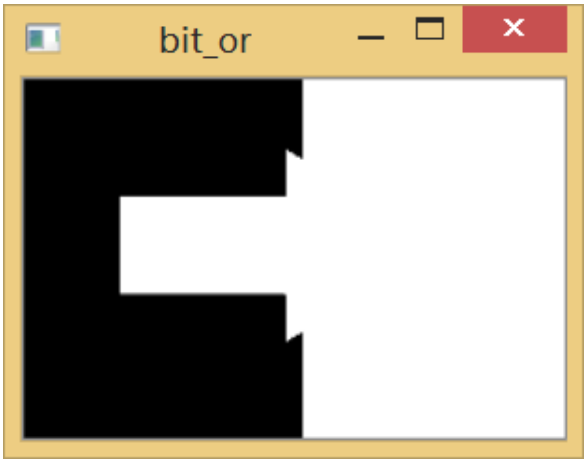
按位運算符是一種常用的圖像處理技巧。例如對 img1 和 img2 進行不同的按位運算可以產生以下的效果：

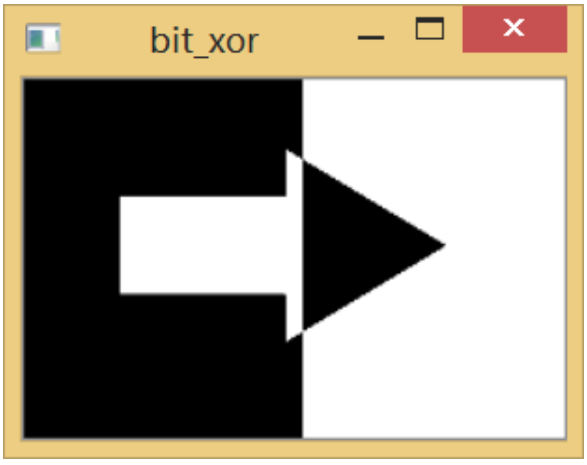




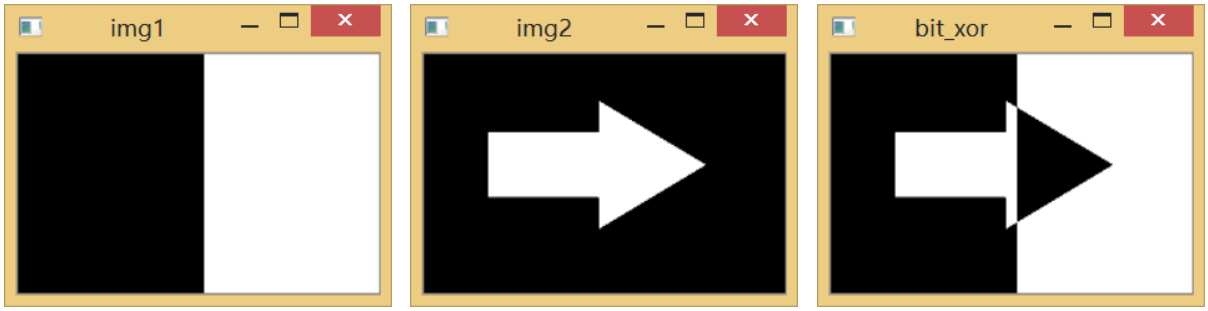




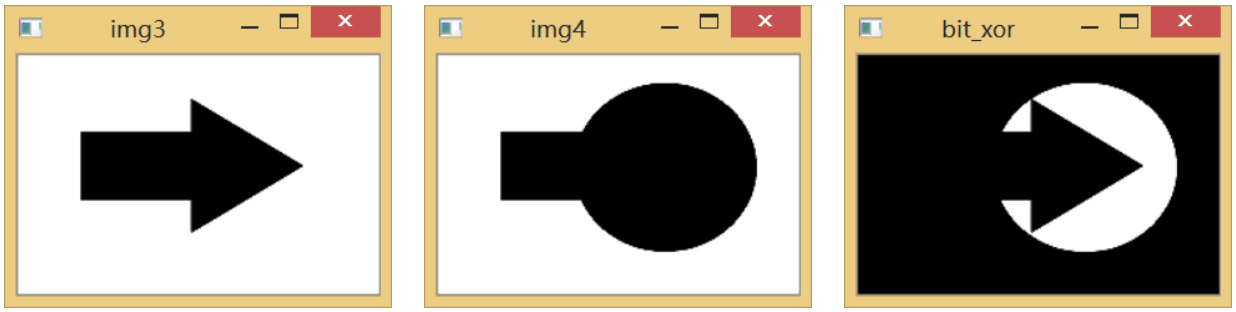




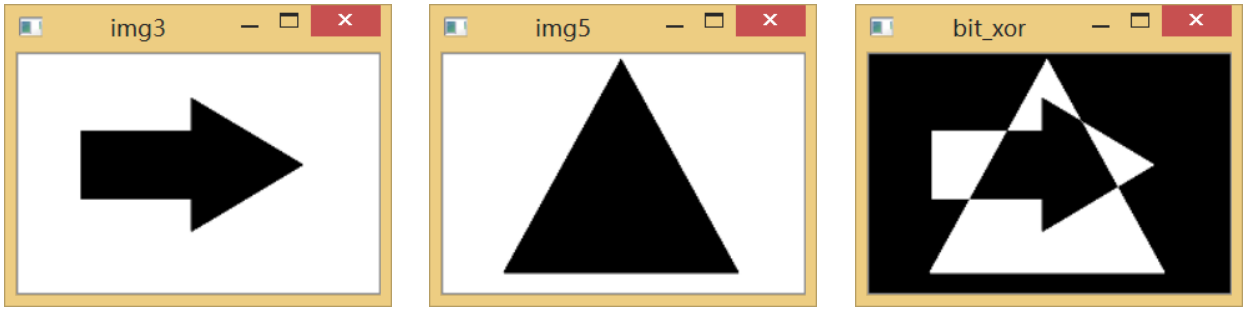
使用 XOR 的時候，可以發現兩張圖片相同的部分會變成黑色(0)，不同的部分會變成白色(1)。所以只要計算 1 的數目(像素)，就可以分別兩張圖片相異的程度(Error)，相異程度越大，兩張圖片越不相似。



從以下例子可見，兩者越不相似，Error 越大，反之亦然。




```
C:\Users\STEM\Documents\Python workshop>python bitwise.py
Image 3 and Image 4
Error:  6018
```




```
C:\Users\STEM\Documents\Python workshop>python bitwise.py
Image 3 and Image 5
Error:  8982
```

完整程式碼

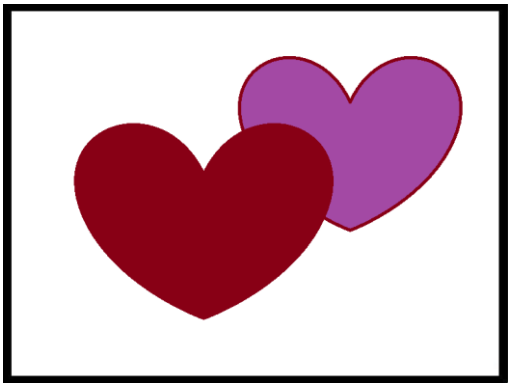
```
1.  import cv2
2.  import numpy as np
3.  # Read the image in gray scale
4.  img1 = cv2.imread("drawing_1.png", cv2.IMREAD_GRAYSCALE)
5.  img2 = cv2.imread("drawing_2.png", cv2.IMREAD_GRAYSCALE)
6.  # Resize the image to 1/4, in order to speed up the calculation
7.  img1 = cv2.resize(img1, None,fx=0.25, fy=0.25, interpolation = cv2.INTER_CUBIC)
8.  img2 = cv2.resize(img2, None,fx=0.25, fy=0.25, interpolation = cv2.INTER_CUBIC)
9.
10. bit_and = cv2.bitwise_and(img2, img1)
11. bit_or = cv2.bitwise_or(img2, img1)
12. bit_xor = cv2.bitwise_xor(img1, img2)
13. bit_not = cv2.bitwise_not(img2)
14.
15. # Remerber the image must be in binary format for non-zero counting
16. #diff = cv2.countNonZero(bit_xor)
17. #print("Image 3 and Image 4 ")
18. #print("Error: ", diff)
19.
20. # Show the results
21. cv2.imshow("img1", img1)
22. cv2.imshow("img2", img2)
23. cv2.imshow("bit_and", bit_and)
24. cv2.imshow("bit_or", bit_or)
25. cv2.imshow("bit_xor", bit_xor)
26. cv2.imshow("img2_bit_not", bit_not)
27.
28. cv2.waitKey(0)
29. cv2.destroyAllWindows()
```

實驗五

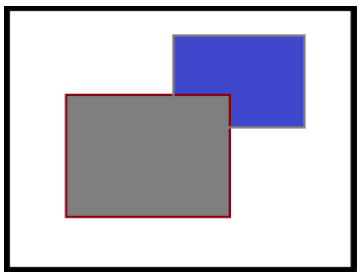
1） 讀取 image.png 圖像，

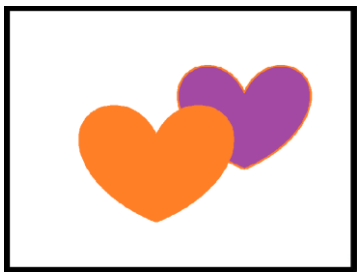
2） 用 XOR 計算出原圖和比較的圖像的像素差，

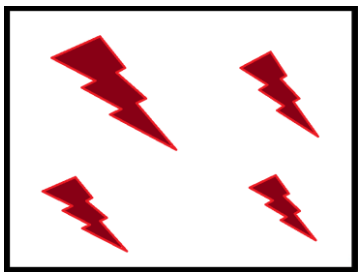
原圖:



比較的圖像:

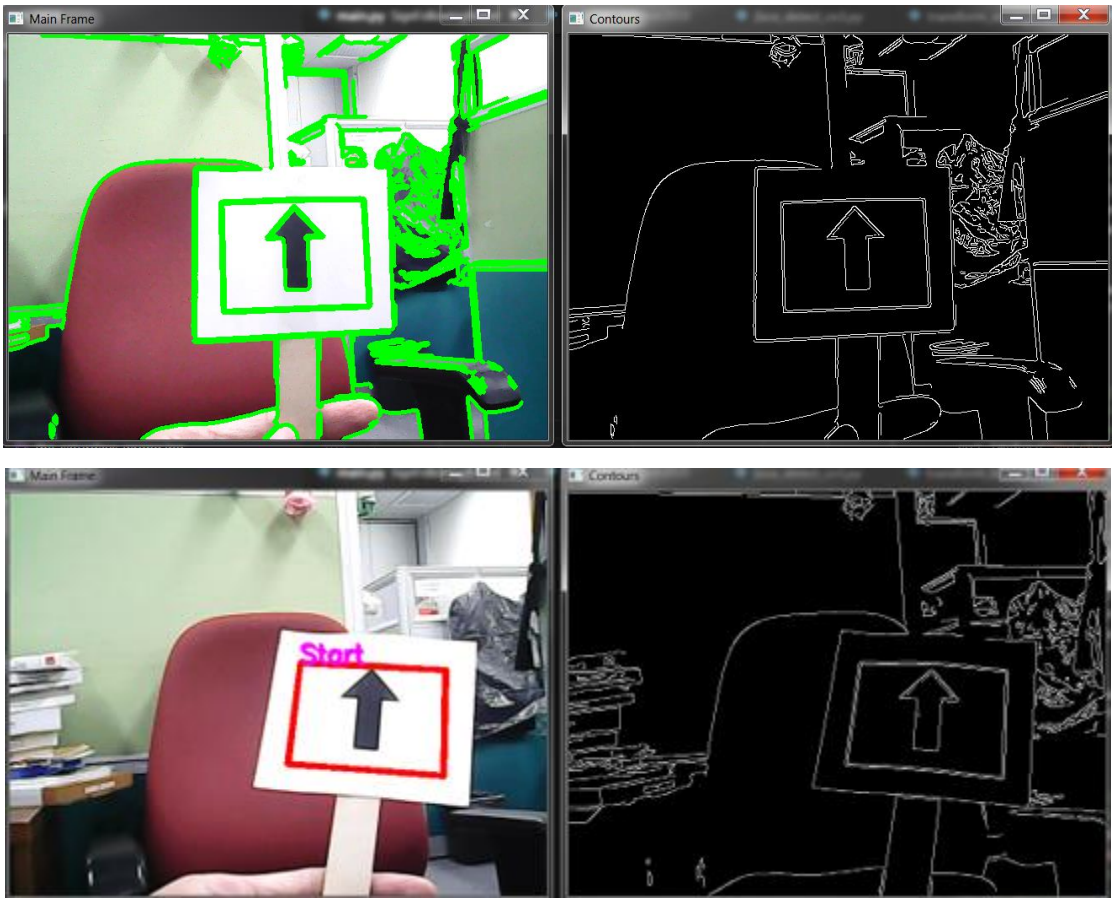






3 物體分割與特徵萃取

Object Segmentation and Feature Extraction



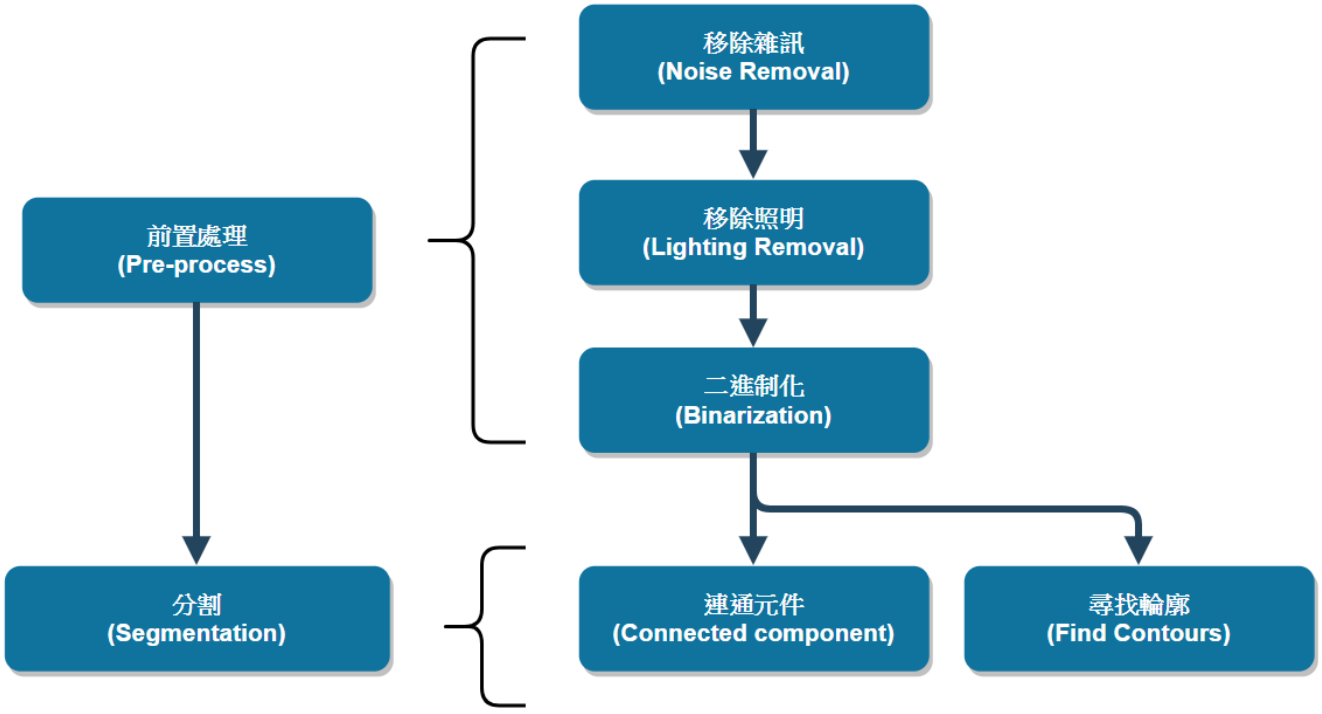
3.1 物體分割與偵測 Object Segmentation and Detection

3.2 特徵萃取 Feature Extraction

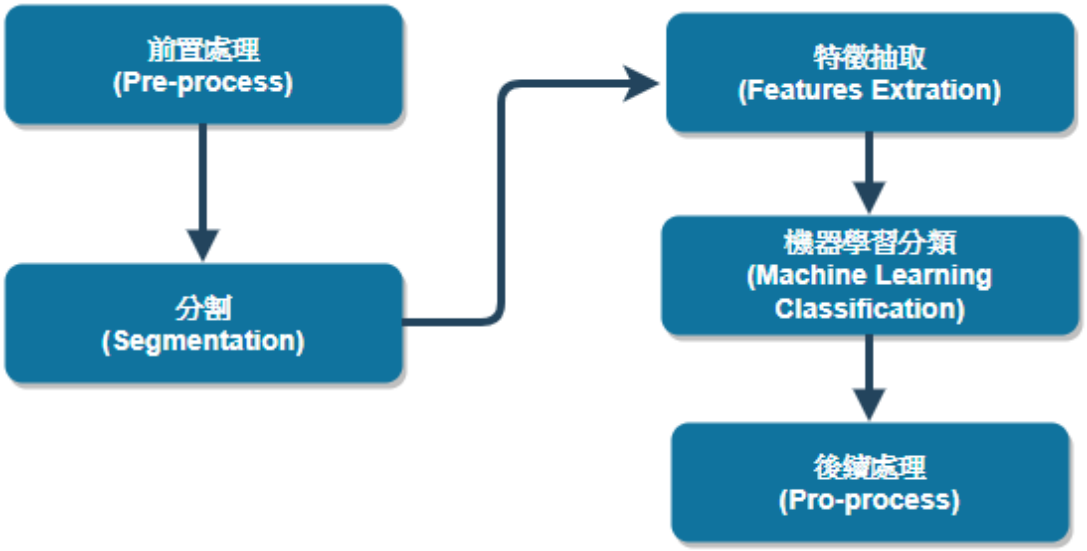
本節會涵蓋以下主題：

- 移除雜訊
- 在物體分割時找出相連的部分
- 在物體分割時找出輪廓

要能做到這樣的結果，比必須依循幾個不同的步驟。可以從下圖看出程式的流程。一般而言，前置處理步驟試著減少影像的雜訊，降低偵測物體或分割影像時發生的錯誤。



大部分電腦視覺的程式都有類似的結構和步驟，下圖就是程式結構所包涵的步驟。



幾乎所有的電腦視覺應用程式都會對輸入影像作前置處理，這包括移除影像的背景，雜訊和閥值處理。

完成對影像的前置處理步驟後，第二個階段就是分割。在例子中我們便是要將圖案從輸入影像分割出來。

取得影像中的物體後會進入下一個階段，萃取每個物體的特徵。每個特徵都是物體的描述，可能是物體的面積，輪廓和紋路等等。

3.1 物體分割與偵測 Object Segmentation and Detection

在這個部份會介紹物體分割與偵測的基本知識，物體分割與偵測是指獨立出影像中的物體供後續處理與分析。

3.1.1 移除雜訊

雜訊通常在影像中會收小點呈現，因此可以被分割為一個物體，如果不先移除雜訊，就會偵測到比預期多的物體。現實上有很多不同的技術可以移除影像中的雜訊，根據不同種類的雜訊，我們可以使用中間值過濾器（median filter）或高斯濾波器（Guassian filter）來實現。

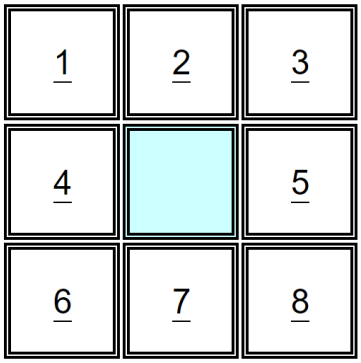
完整程式碼

```
1. import numpy as np
2. import cv2
3. img = cv2.imread('SaltAndPepperNoise.jpg')
4. imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
5. median = cv2.medianBlur(imgray,5)
6. cv2.imshow("Salt_PepperNoise", img)
7. cv2.imshow("Median filter", median)
8. cv2.waitKey(0)
9. cv2.destroyAllWindows()
```

3.1.2 分割輸入影像 Object Segmentation

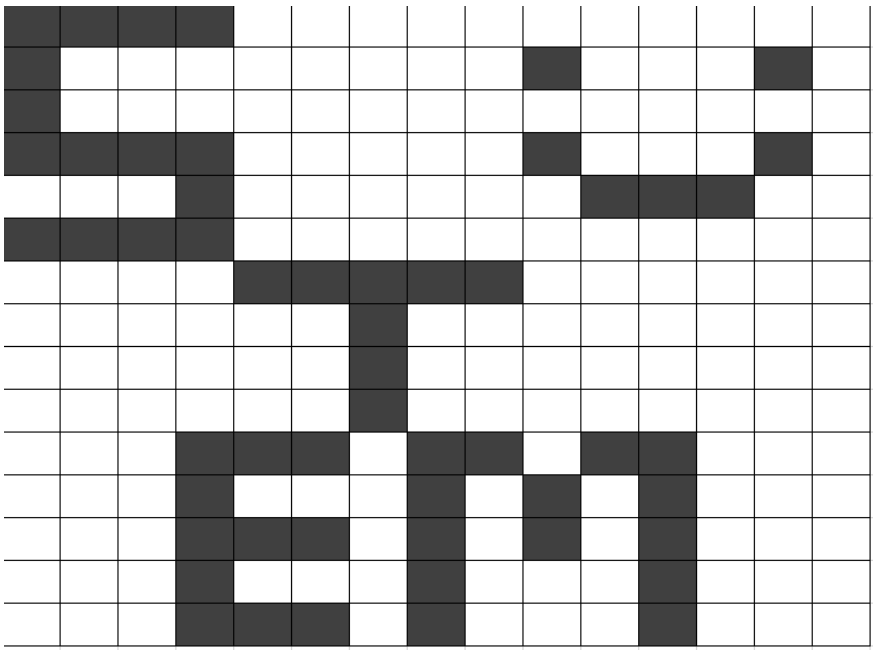
演算法的基本概念

接下來就要進入物體分割的部分，連結組件是個很常用在對二進位影像分割以及找出元件部分的演算法。可以使用 8- 或 4- 相連像素標記的迭代演算法（iterative algorithm），兩個像素互相連結的條件是兩個像素相同且彼此相鄰。

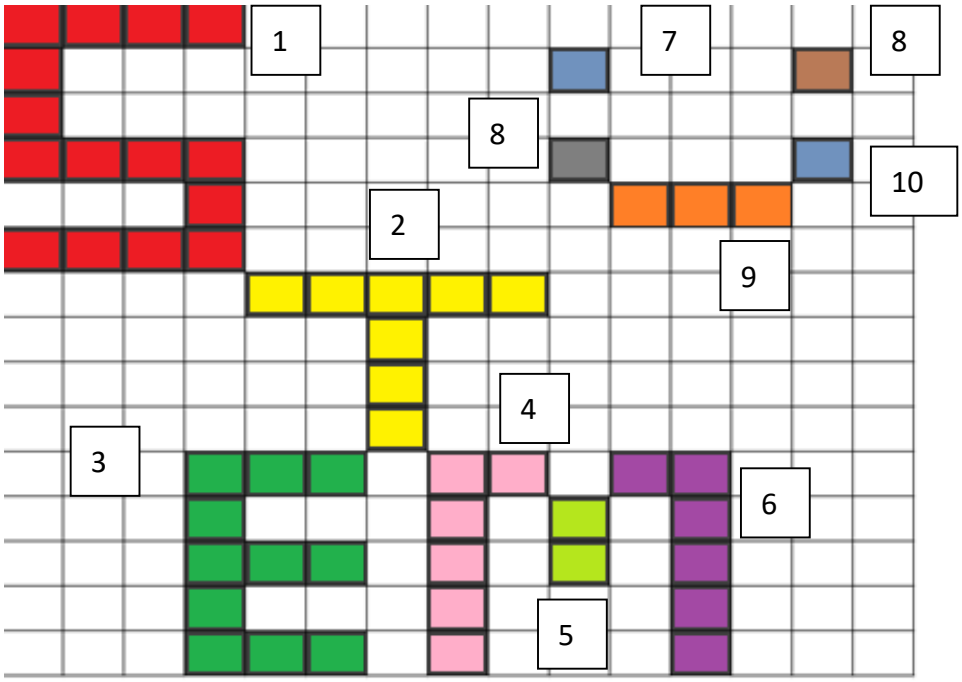


4- 連結是只有 2,4,5,7 四個相鄰像素與中心顏色相同時，兩個像素才算是互相連結。而 8- 連結是 1,2,3,4,5,6,7,8 的顏色興中心相同都可以視為互相連結。

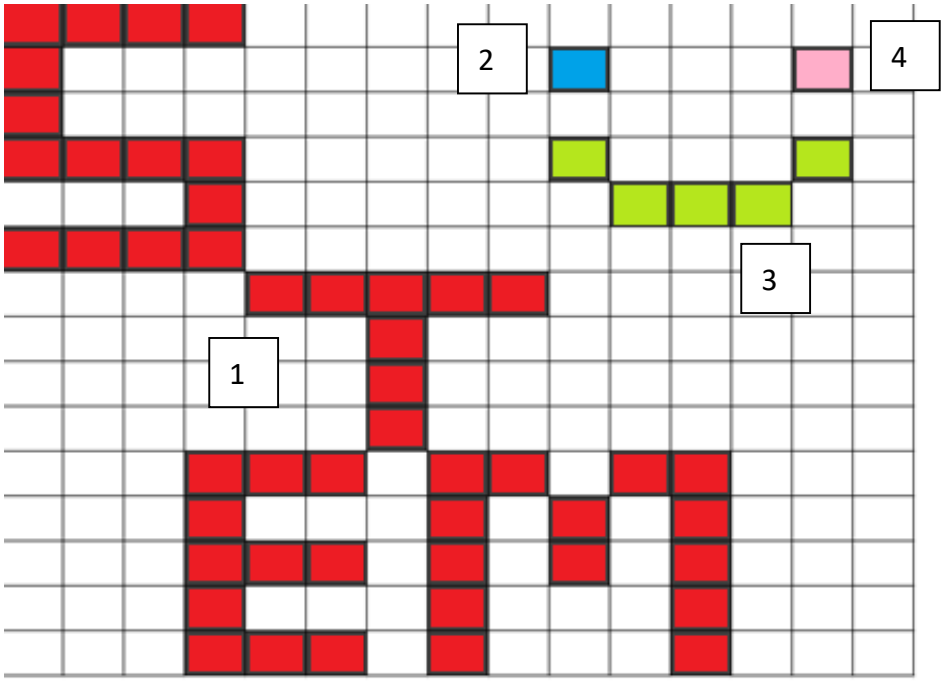
以下的物件可以分別用 4- 連結和 8- 連結演算法進行分割：



4- 連結演算法



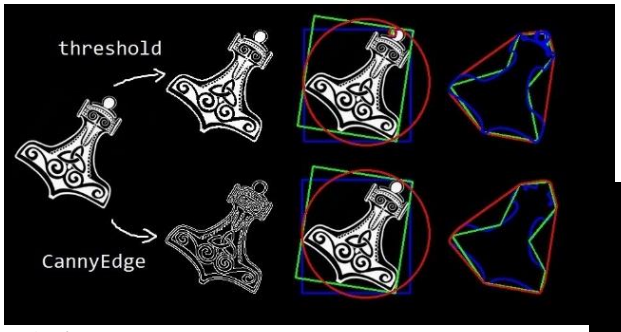
8- 連結演算法



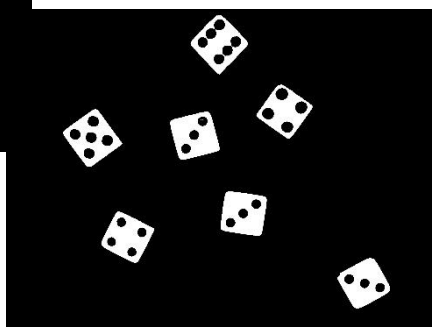
3.2 特徵萃取 Feature Extraction

接下來會介紹萃取個別物體特徵的方法。這些特徵一般都比較簡單，但已經足夠獲得良好的結果。以下介紹一些能幫助電腦辨別個別物體的良好特徵，這類特徵有：

- 輪廓的邊數
- 物體的面積
- 物體的長寬比
- 空洞的數目



圖片來源: https://zhuanlan.zhihu.com/p/38739563



這些特徵都能夠適當地描述物體，而接下來的練習會使用前兩個特徵，輪廓的邊數和物體的面積，因為這些特徵都能夠直接正確的描述物體。

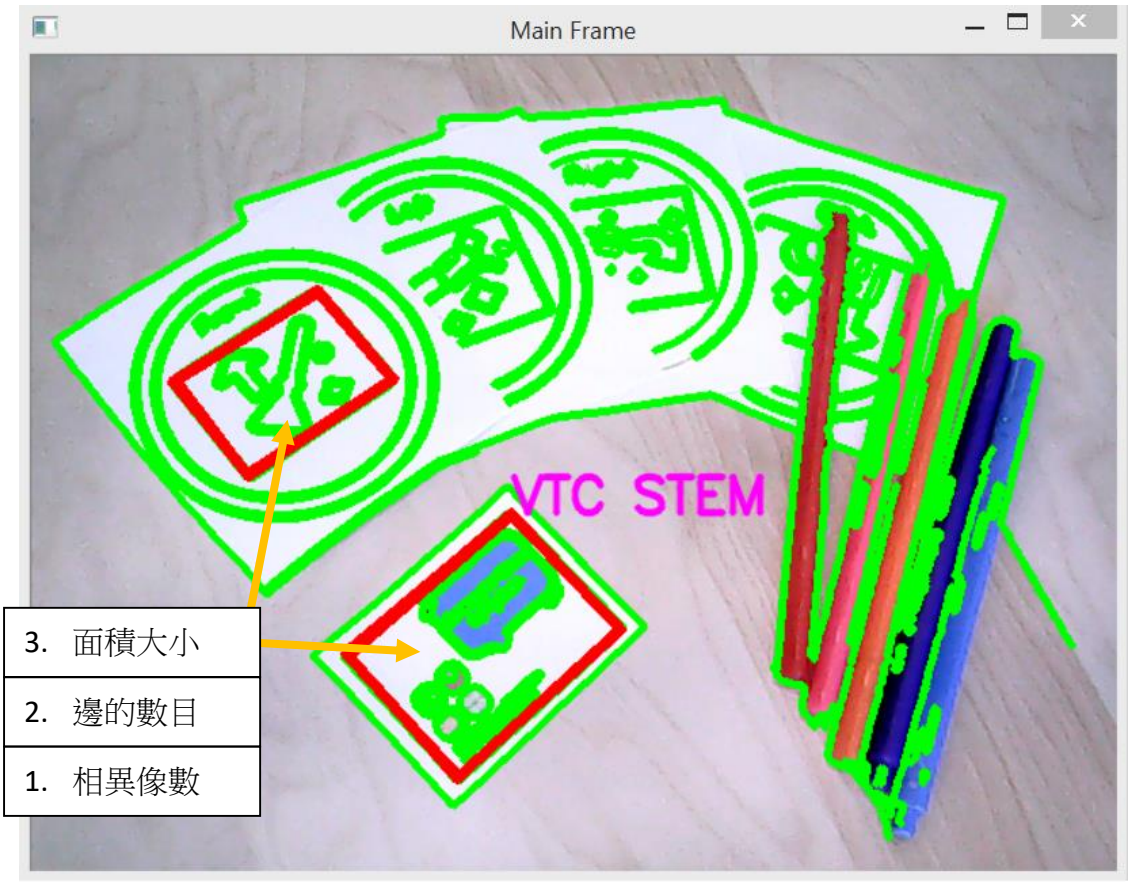
面積就是第一個特徵，接下來會使用面積作為過濾器，移除需要排除的小物體（雜訊），所有小於特定面積的物體都會被忽略。通過過濾條件後，接著會建立第二個特徵，從尋找輪廓的過程我們可以找出擁有 4 條邊的矩形，從而區分出輸入影像中，尋找的圖案與其他物體。

程式最後會利用位元運算子(Bitwise operator) ，計算未知樣本(輸入影像)與訓練資料的相似度，從而產生分類結果。位元運算子是用來計算變數的位元，運作得宜可以提高程式的運作效率。

程式利用 XOR 運算，比對輸入影像和訓練資料，計算出兩者相異像數的數目，當兩者的相異像數低過某個數值時，就會判定兩者為相同的物體。

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

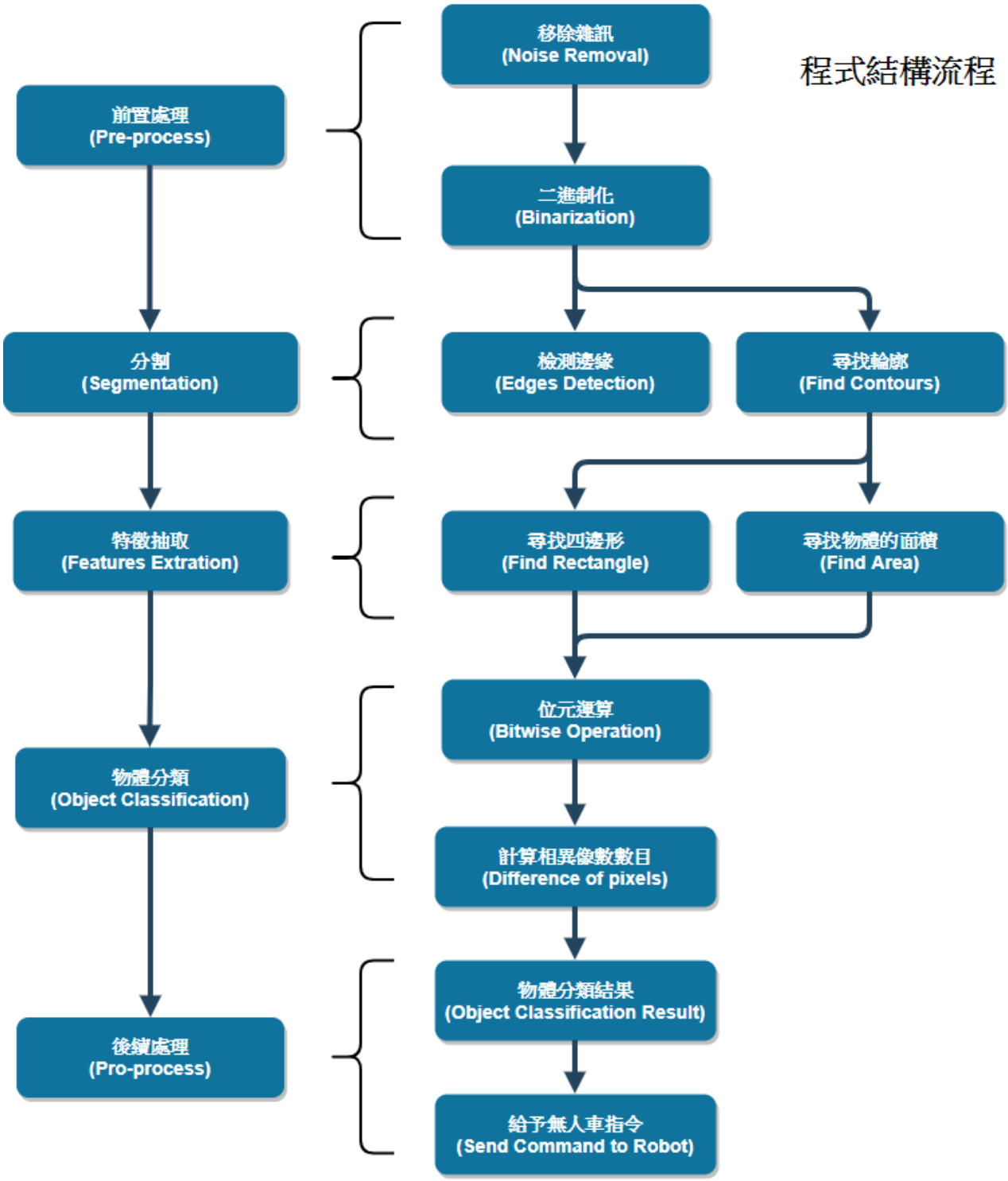
XOR 運算邏輯圖表

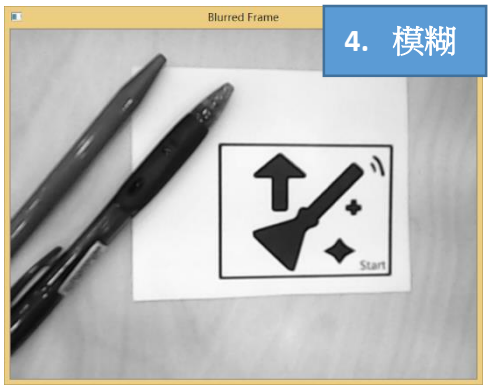


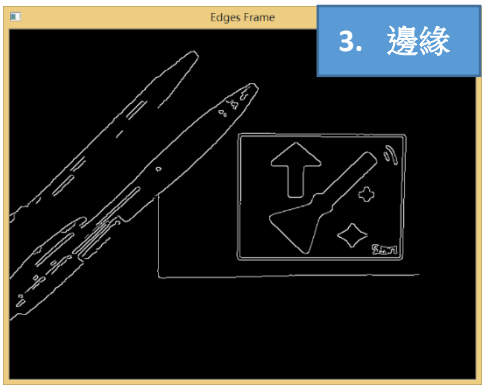
描述物體的特徵

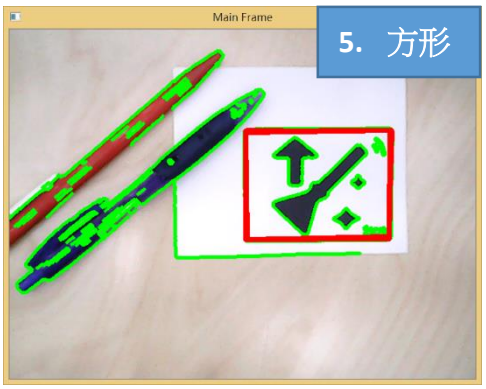
3.3 程式流程圖

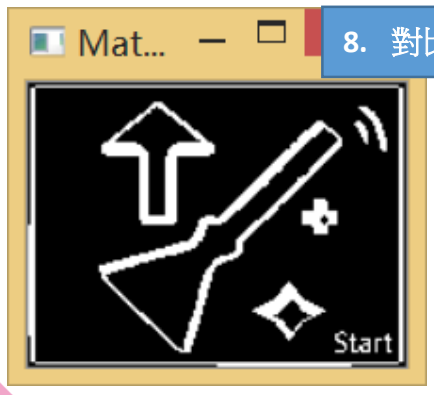
以下圖表總結了程式的結構流程，當程式完成分類的過程後，之後的後續處理就會利用分類結果，對自動駕駛車作出不同的指令。











3.3 程式整合

1 Import the necessary packages

2 Set Variables e.g.

3 Load the image to database

4 Turn on the webcam

```
while True:
```

5 Load the video frame from camera

6 Convert the color image to gray

7 Convert the gray image to blur image

8 Detecting Edges

9 Contour Detection & checking for squares based on the square area

10 Send the Image for checking

11 Show Resulted Image

```
    # Press q from keyboard to exit
      if cv2.waitKey(1) & 0xFF == ord('q'):
              break
```

12 Release program resources

- **A**

```
# Load the image to database
sign.readRefImages()
```

- **B**

```
# Import the necessary packages
import time
import cv2
import numpy as np
import os
import signDetectionLib as sign
```

- **C**

```
# Turn on the webcam
video = cv2.VideoCapture(0)
```

- **D**

```
# Convert the gray image to blur image
blurred = cv2.GaussianBlur(gray,(3,3),0)
```

- **E**

```
# Load the video frame from camera and convert the frame to gray
color
ret, OriginalFrame = video.read()
```

- **F**

```
# Variable
minDiff = 2000
minSquareArea = 2000
vertex = 4
```

- **G**

```
# Send the Image for checking
sign.checkImage(OriginalFrame, contours, vertex, minSquareArea,
minDiff)
```

- **H**

```
# Contour Detection & checking for squares based on the square area
contours, hierarchy =
cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
#cv2.drawContours(OriginalFrame, contours, -1, (0,255,0), 3)
```

- **I**

```
# Convert the color image to gray
gray = cv2.cvtColor(OriginalFrame, cv2.COLOR_BGR2GRAY)
```

- **J**

```
# Release program resources
video.release()
cv2.destroyAllWindows()
```

- **K**

```
# Show Resulted Image
cv2.imshow("Main Frame", OriginalFrame)
```

- **L**

```
# Detecting Edges
edges = sign.auto_canny(blurred)
```

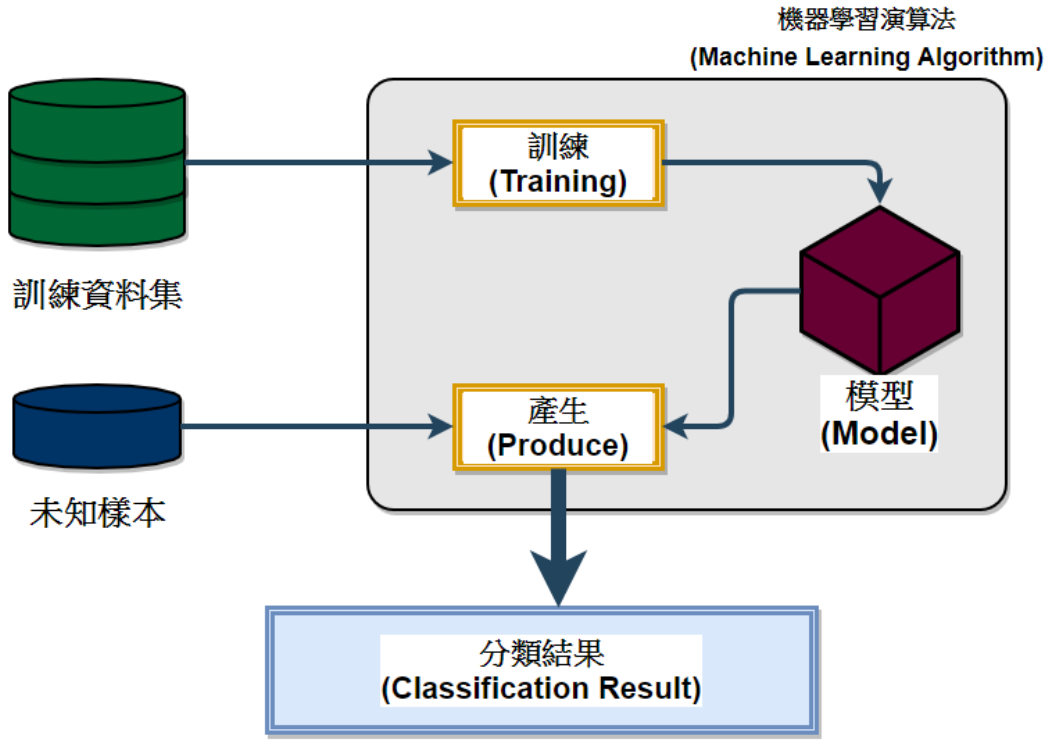
完整程式碼

```
# ======================================================== #
# Author: VTC STEM Centre/ Gary Lam
# Date: 5/12/2019
# Program Name: RoadSign Game version 1.3
# Description: Program code for sign detection
# ======================================================== #

# import the necessary packages
import time
import cv2
import numpy as np
import os
import signDetectionLib as sign

#Difference Variable
minDiff = 3000
minSquareArea = 2000
vertex = 4

video = cv2.VideoCapture(0)

sign.readRefImages()

print ("Please use the Road Sign to play this demo.")
print ("Please press q to exit")

while True:
        #Load the video frame from camera and convert the frame to gray color
        ret, OriginalFrame = video.read()
        gray = cv2.cvtColor(OriginalFrame, cv2.COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(gray,(3,3),0)

        # Detecting Edges
        edges = sign.auto_canny(blurred)

```

```
36.         # Contour Detection & checking for squares based on the square area
37.         contours, hierarchy = cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIM
    PLE)
38.         #cv2.drawContours(OriginalFrame, contours, -1, (0,255,0), 3)
39.
40.         # Send the Image for checking
41.         sign.checkImage(OriginalFrame, contours, vertex, minSquareArea, minDiff)
42.
43.         # Show Image
44.         cv2.imshow("Main Frame", OriginalFrame)
45.         #cv2.imshow("Gray", gray)
46.         #cv2.imshow("Blurred", blurred)
47.         #cv2.imshow("Edges", edges)
48.
49.         if cv2.waitKey(1) & 0xFF == ord('q'):
50.                 break
51.
52. video.release()
53. cv2.destroyAllWindows()
```

延伸程式: 機器學習與物體分類 Machine Learning and Object Classification

從上述的步驟計算到物體的特徵後，就會得到物體的描述子（Descriptor），描述子是描述物體的特徵，程式會利用描述子來訓練系統的模型或進行預測。為了令預測的準確度提高，就必須對影像進行上千，甚至上萬次的前置處理和萃取特徵，利用這些特徵來訓練電腦模型，並且建立巨大的特徵資料集。



為了建立模型，我們會使用到機器學習（Machine Learning）演算法。機器學習是人工智慧（Artificial Intelligence）的一個分支，機器學習會從資料中自動分析獲得規律，並利用規律對未知資料進行預測。訓練的時候，模型會學習到預測未知標記的特徵向量時所需要的所有參數。當輸入未知樣本的時候，機器就能夠抽取物體的特徵，分類各個物體。

參考連結/書籍

OpenCV-Python 中文教程
https://www.kancloud.cn/aollo/aolloopencv

pyimagesearch
https://www.pyimagesearch.com/

Python 入門教學課程
https://www.youtube.com/watch?v=wqRlKVRUV_k&list=PL-g0fdC5RMboYEyt6QS2iLb_1m7QcgfHk

OpenCV computer vision with Python : learn to capture videos, manipulate images and track objects with Python using the OpenCV library / Joseph Howse Howse, Joseph.

OpenCV by example : enhance your understanding of computer vision and image processing by developing real-world projects in OpenCV 3 / Prateek Joshi, David Millán Escrivá, Vinicius Godoy Joshi, Prateek.

世界排名第一的視覺資料庫 : OpenCV 開發一本搞定 / 李立宗, 著 李立宗. | 佳魁數位 | 2017

Raspberry Pi 入門與機器人實作應用 / 王進德 著 王進德. | 博碩文化股份有限公司 | 2017 | 初版

會 Python : 從不懂, 到玩上手! / 陳會安 著 陳會安. | 旗標科技股份有限公司 | 2017 | 初版