

多媒體技術與應用 Spring 2022

Instructor : Yen-Lin Chen(陳彥霖), Ph.D.

Professor

Dept. Computer Science and Information Engineering

National Taipei University of Technology



Lecture 2

OpenCV影像處理及物件檢測技術



OpenCV

影像金字塔(Image Pyramid)



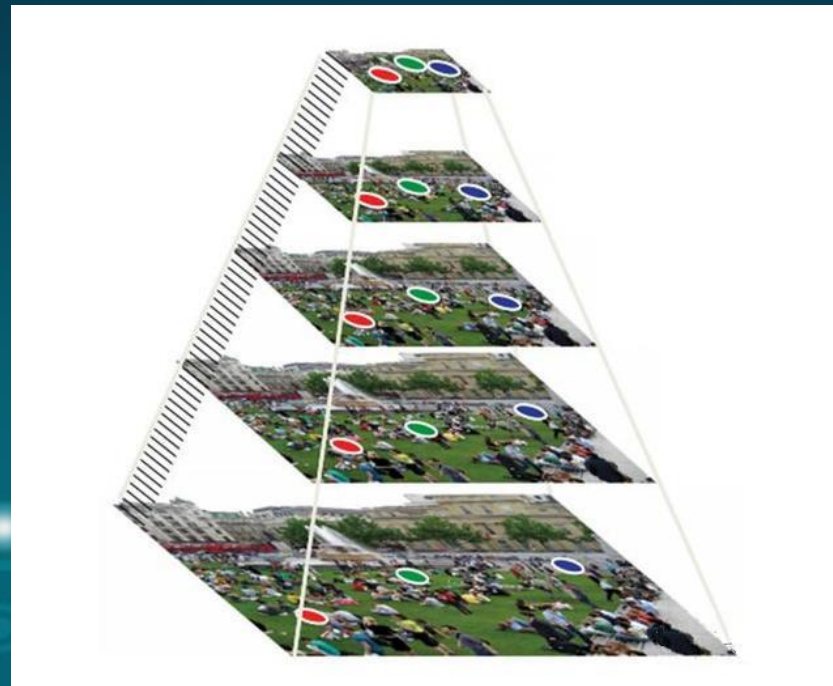
影像金字塔(Image Pyramid)

- 影像金字塔是指一組影像且不同解析度的子圖集合，它是影像多尺度表達的一種，以多解析度來解釋影像的結構，主要用於影像的分割或壓縮。



影像金字塔(Image Pyramid)

- 一幅影像的金字塔是一系列以金字塔性質排列的解析度逐步降低，且來源於同一張原始圖的影像集合，如下圖所示，它包括了五層影像，將這一層一層的影像比喻成金字塔。影像金字塔可以通過梯次向下取樣獲得，直到達到某個終止條件才停止取樣，在向下取樣中，層次越高，解析度越低。

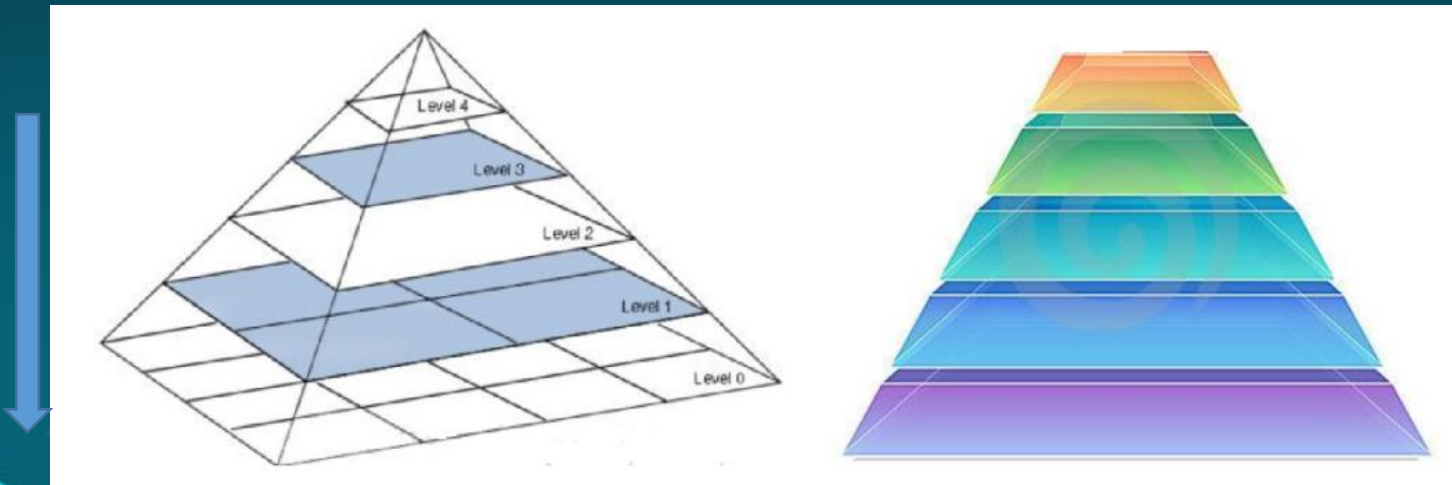




影像金字塔(Image Pyramid)

- 生成影像金字塔主要包括兩種方式——向下取樣，向上取樣。
- 將level0級別的影像轉換為 level1，level2，level3，level4，影像解析度不斷降低的過程稱為向下取樣；
- 將level4級別的影像轉換為 level3，level2，level1，level0，影像解析度不斷增大的過程稱為向上取樣。

向上取樣



向下取樣



高斯金字塔(Gaussian Pyramid)

- 高斯金字塔用於下取樣。高斯金字塔是最基本的影像塔。原理：首先將原影像作為最底層影像 level0（高斯金字塔的第0層），利用高斯核（ 5×5 ）對其進行卷積，然後對卷積後的影像進行下取樣（去除偶數行和列）得到上一層影像G1，將此影像作為輸入，重複卷積和下取樣操作得到更上一層的影像，反覆迭代多次，形成一個金字塔形的影像數據結構，即高斯金字塔。



高斯金字塔(Gaussian Pyramid)

- 高斯金字塔是通過高斯平滑和亞取樣獲取一些列下取樣影像，也就是說第 K 層高斯金字塔通過平滑，亞取樣就可以獲得 $K+1$ 層高斯影像，高斯金字塔包含了一系列低通濾波器，其截止頻率從上一層到下一層是以因子 2 逐漸增加，所以高斯金字塔可以跨越很大的頻率範圍。



拉普拉斯金字塔(Laplacian Pyramid)

- 拉普拉斯金字塔用於重建圖形，也就是預測殘差，對影像進行最大程度的還原。比如一幅小影像重建為一幅大圖。原理：用高斯金字塔的每一層影像減去其上一層影像上取樣並高斯卷積之後的預測影像，得到一系列的差值影像，即為Laplacian分解影像。



拉普拉斯金字塔(Laplacian Pyramid)

- 拉普拉斯影像的形成過程大致為對原影像進行低通濾波和降取樣得到一個粗尺度的近似影像，即分解得到的低通近似影像，把這個近似影像經過插值，濾波，再計算它和原影像的插值，就得到分解的帶通分量。下一級分解是在得到的低通近似影像上進行，迭代完成多尺度分解。可以看出拉普拉斯金字塔的分解過程包括四個步驟：
 1. 低通濾波
 2. 降取樣（縮小尺寸）
 3. 內插（放大尺寸）
 4. 帶通濾波（影像相減）



影像向下取樣(高斯金字塔—縮小)

- 在影像向下取樣中，使用最多的是高斯金字塔。它將堆影像 G_i 進行高斯核卷積，並刪除原圖中所有的偶數行和列，最終縮小影像。其中，高斯核卷積運算就是對整幅影像進行加權平均的過程，每一個像素點的值，都由其本身和鄰域內的其他像素值（權重不同）經過加權平均後得到。常見的 $3*3$ 和 $5*5$ 高斯核如下：

$$K(3,3) = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$K(5,5) = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

$$\frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- 高斯核卷積讓臨近中心的像素點具有更高的重要度，對周圍像素計算加權平均值，如下圖所示，其中心位置權重最高為 0.4。
- 顯而易見，原始影像 G_i 具有 $M*N$ 個像素，進行向下取樣之後，所得到的影像 G_{i+1} 具有 $M/2 * N/2$ 個像素，只有原圖的四分之一。通過對輸入的原始影像不停迭代以上步驟就會得到整個金字塔。注意，由於每次向下取樣會刪除偶數行和列，所以它會不停地丟失影像的資訊。





影像向下取樣—OpenCV

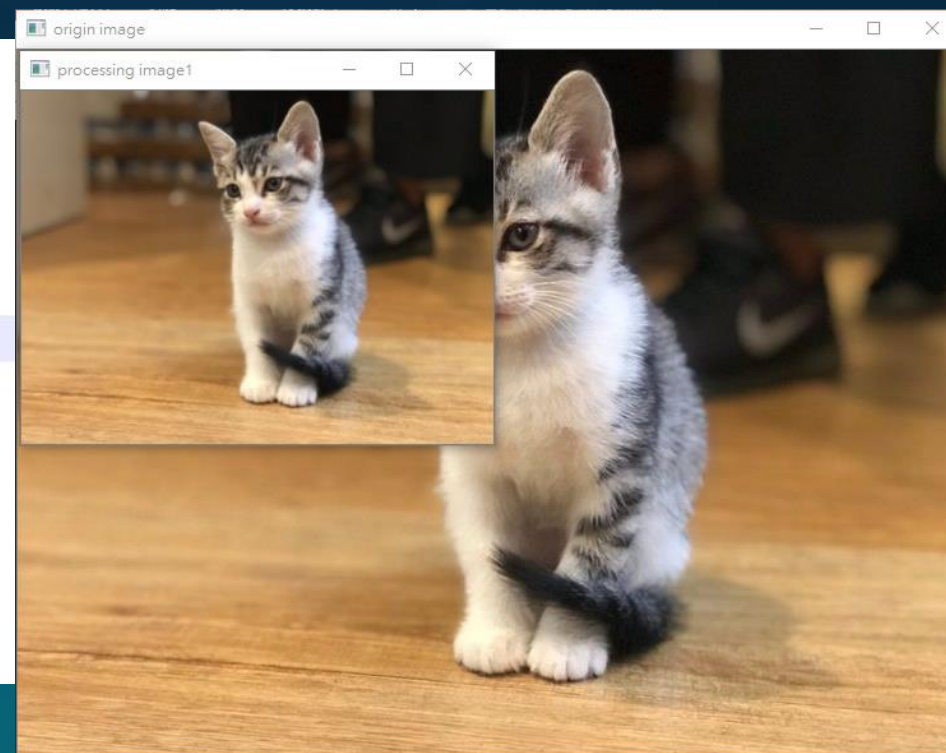
- `dst = pyrDown(<src>[, <dst>[, <dstsize>[, <borderType>]])`
 - `src`:表示輸入影像，
 - `dst`:表示輸出影像，和輸入影像具有一樣的尺寸和類型
 - `dstsize`:表示輸出影像的大小，默認值為`Size(5*5)`
 - `borderType`:表示像素外推方法，詳見`cv::bordertypes`
 - 例：`r1 = cv2.pyrDown(img)`



影像向下取樣—OpenCV

- 範例程式:

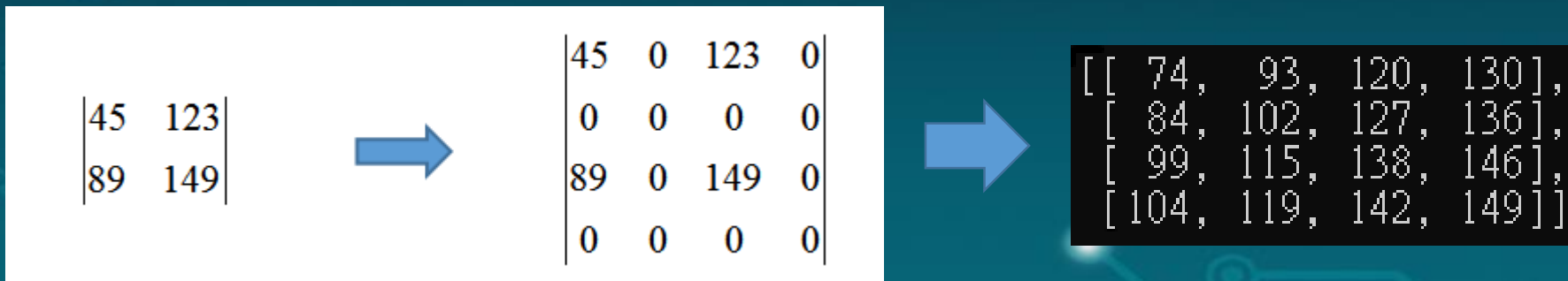
```
1 import cv2
2 import numpy as np
3
4 # 讀取原始圖片
5 img = cv2.imread('cat.jpg')
6
7 # 影像向下取樣
8 r1 = cv2.pyrDown(img)
9
10 # 顯示圖形
11 cv2.imshow('origin image', img)
12 cv2.imshow('processing image1', r1)
13 cv2.waitKey(0)
14 cv2.destroyAllWindows()
```





影像向上取樣(高斯金字塔—放大)

- 在影像向上取樣是由小影像不斷放大影像的過程，它將影像在每個方向上擴大為原影像的2倍，新增的行和列均使用0來填充，並使用於「向下取樣」相同的卷積核乘以4，再與放大後的影像進行卷積運算，以獲得「新增像素」的新值。如下所示，它在原始像素45，123，89，149之間各新增了一行和一列值為0的像素。





影像向上取樣—OpenCV

- `dst = cv2.pyrUp(<src>[, <dst>[, <dstsize>[, <borderType>]])`
 - `src`:表示輸入影像，
 - `dst`:表示輸出影像，和輸入影像具有一樣的尺寸和類型
 - `dstsize`:表示輸出影像的大小，默認值為`Size()`
 - `borderType`:表示像素外推方法，詳見`cv::bordertypes`
 - 例：`r1 = cv2.pyrUp(img)`



影像向上取樣—OpenCV

- 範例程式:

```
1 import cv2
2 import numpy as np
3
4 # 讀取原始圖片
5 img = cv2.imread('cat.jpg')
6
7 # 影像向上取樣
8 r1 = cv2.pyrUp(img)
9
10 # 顯示圖形
11 cv2.imshow('origin image', img)
12 cv2.imshow('processing image1', r1)
13 cv2.waitKey(0)
14 cv2.destroyAllWindows()
```





影像金字塔總結

- **向上取樣**：就是圖片放大，使用pyrUp函數。上取樣的步驟：先將影像在每個方向放大為原來的兩倍，新增的行和列用0填充，再使用先前同樣的內核與放大後的影像卷積，獲得新增像素的近似值。
- **向下取樣**：就是圖片縮小，使用pyrDown函數。下取樣步驟：先將圖片進行高斯內核卷積，再將所有偶數列去除。
- **注意**：pyrUp() 和 pyrDown() 不是互逆的，即上取樣和下取樣的不是互為逆操作。
- 上，下取樣都存在一個嚴重的問題，那就是影像變模糊了，因為縮放的過程中發生了資訊丟失的問題。要解決這個問題，就得用拉普拉斯金字塔。

物件檢測概述和應用



物件檢測概述

- 物件檢測顧名思義就是要找出影像中感興趣的物件，這個方法可以透過基本的影像處理，如：二值化或影像相減等技術來達成，近年也流行使用深度學習的方式，如：YOLOv4來辨識影像中的物件。
- 本次課程將介紹如何使用連通物件或邊緣檢測等基本的影像處理技術來找出影像中感興趣的物件。



物件檢測概述與應用

- 物件檢測可以辨識字元或進行字元的分割提取（如：車牌識別、文字識別、字幕識別等）、視覺跟蹤中的運動前景目標分割與提取（行人偵測、遺留物體偵測、基於視覺的車輛偵測與追蹤等）、醫學影像處理（感興趣目標區域(ROI)提取）、等等。
- 也就是說，在需要將前景目標提取出來，以便後續進行處理的應用場景中都能夠用到連通區域分析方法來進行物件檢測。通常連通區域分析處理的物件是一張二值化後的影像。



物件檢測應用 - 車道線辨識

在自駕車的技術中，常利用邊緣檢測的手法，我們可以更容易地萃取出車道線的特徵。



圖為經過Canny處理後的結果



連通物件



連通物件(Connected Component)-概述

- 連接物件標記演算法(connected component labeling algorithm)是影像分析中最常用的演算法之一，演算法的實質是掃描二值影像的每個像素點，對於像素值相同的而且相互連通分為相同的組(group)，最終得到影像中所有的像素連通元件。
- 連通區域一般是指影像中具有相同像素值且位置相鄰的前景像素點組成的影像區域。連通區域分析是指將影像中的各個連通區域找出並標記。

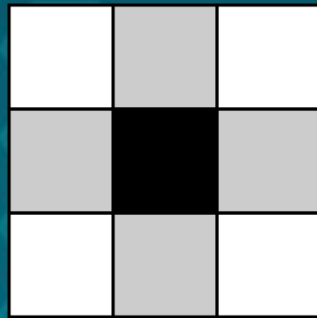
**Connected Component Labeling
Two-Pass Algorithm Demo**

Author: www.icvpr.com

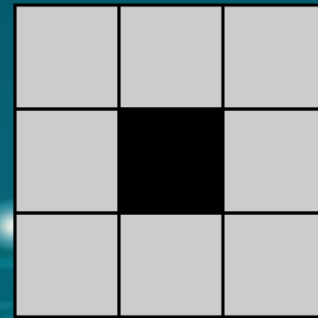


連通物件(Connected Component)-概述

- 連通物件演算法中連通類型大致可分為2種，分別為4-connected(4-連通)、8-connected(8-連通)。
- 4-connected是指對應像素位置的上、下、左、右，是緊鄰的位置。共4個方向，所以稱之為四連通，又稱四鄰域。
- 8-connected是指對應位置的上、下、左、右、左上、右上、左下、右下，是緊鄰的位置和斜向相鄰的位置。共8個方向，所以稱之為8連通或八鄰域。



4-connected

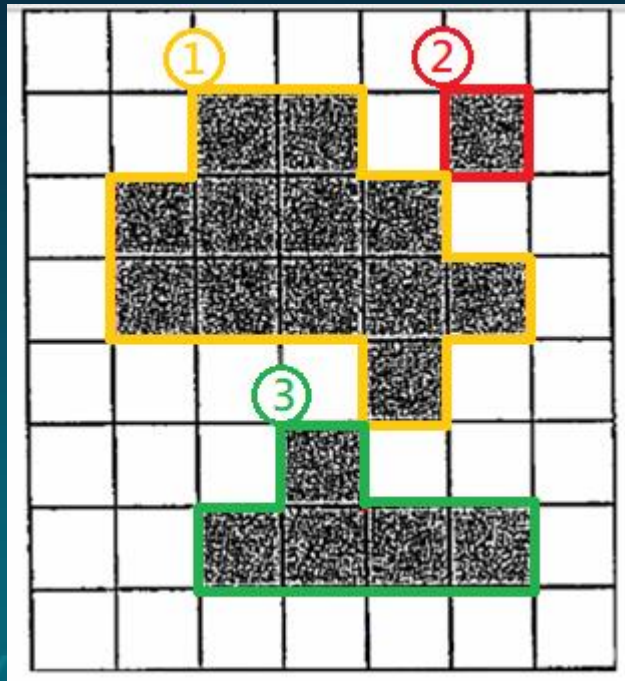


8-connected

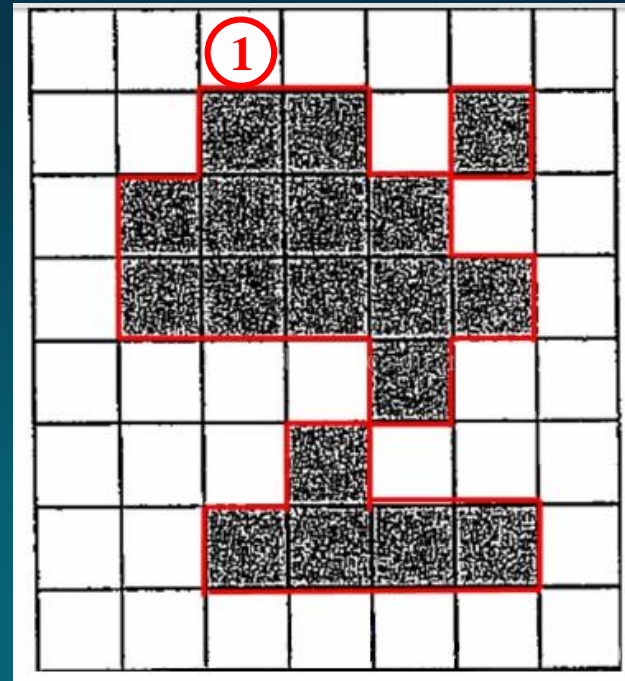


連通物件(Connected Component)-概述

- 如下圖所示，若為4-connected(4-連通)則會分成3塊不同的連通物件，若為8-connected(8-連通)則只有1塊連通物件。



4-connected



8-connected



連通物件(Connected Component)-演算法

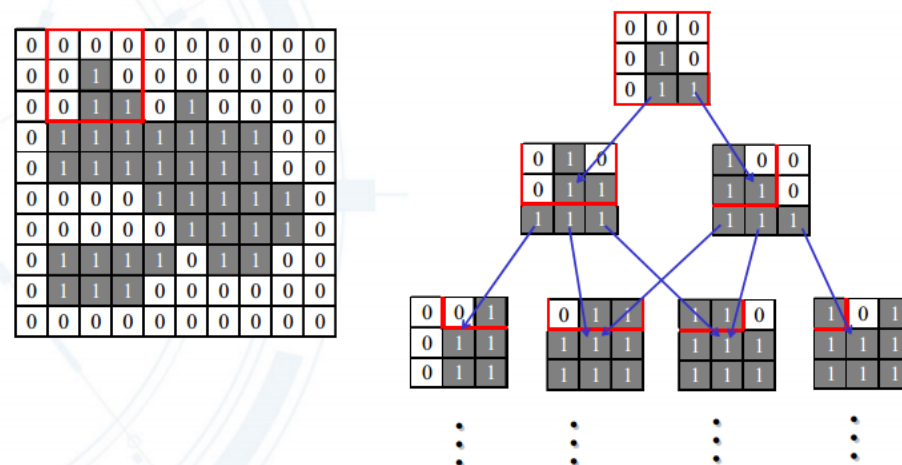
• 一、Recursive algorithm (遞歸法)

1. 遞歸法首先掃描二值化影像，找出灰階值為1且未經標號的像素，並指定一個未經使用的號碼給該像素。
2. 接下來利用8-connected 連通的原理，找出灰階值同為1之近鄰像素，並指定同一個標號給該像素，然後以遞迴的方式持續進行，直到沒有任何近鄰之像素值為1為止。

• Recursive algorithm 執行步驟：

1. 掃描找出灰階值為1且未經標號的像素，並指定一個未經使用的號碼給該像素。
2. 以遞迴的方式指定同一個標號給該像素之8-connected中像素值為1的像素。
3. 當不再有未經標號且像素值為1之像素即停止。
4. 回到步驟1。

Recursive algorithm 範例：



連通物件(Connected Component)-演算法

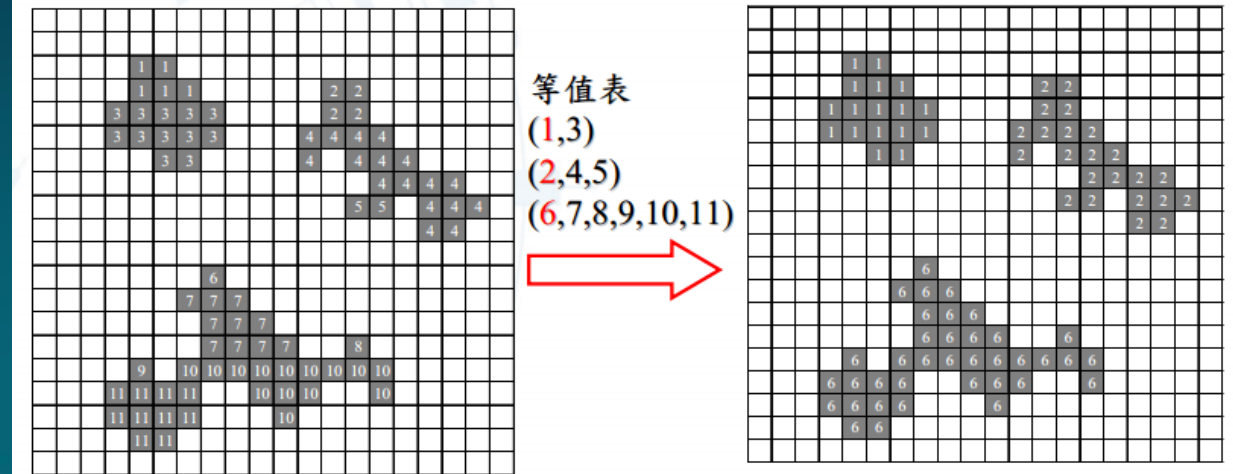
• 二、Sequential algorithm (循序法)

1. 循序法是以4-connected連通為基礎，以「由左而右，由上而下」的順序，依序掃描影像。
2. 遇到像素值為1之像素時，則根據該像素左邊及正上兩個像素之像素值，依照事先設定之規則賦予該像素值一個標號。

• Sequential algorithm 執行步驟：

1. 「由左而右，由上而下」掃描影像。
2. 若遇到像素值為1則：
 - a. 若目前處理中的像素的上面或左邊鄰近像素只有一個有標號，則將該標號複製給目前處理中的像素。
 - b. 若上述兩個鄰近像素的標號相同，則將該標號複製給目前處理中的像素。
 - c. 若上述兩個鄰近像素的標號不同，則將值較大的標號複製給目前處理中的像素，並將該標號記錄在等值標號表內。
 - d. 若上述兩個鄰近像素都沒有標號，則指定一個新的標號給目前處理中的像素。
3. 若第一回合的掃描未完成，則回到步驟2。
4. 搜尋等值標號表內，各組等值標號的最小值。
5. 進行第二回合的掃描，將各組等值標號以其最小值取代。

Sequential algorithm範例：





連通物件-OpenCV

- `num_labels, labels = cv2.connectedComponents(image[, connectivity[, ltype]])`
 - `num_labels`：回傳連通物件的數量。
 - `labels`：回傳連通物件的標記，用數字1、2、3...表示。
 - `image`：輸入影像，必須是二值圖，即8位元單通道影像。（因此輸入影像必須先進行二值化處理才能被這個函式接受）
 - `connectivity`：連通域，可以為8-連通或是4-連通，預設是8-連通
 - `ltype`：輸出影像標記的類型，目前支援CV_32S 和 CV_16U。
 - 例：`num_labels, labels = cv2.connectedComponents(image, connectivity=8, ltype=None)`



連通物件-OpenCV

- `num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(image [, connectivity[, ltype]])`
 - `num_labels`：回傳連通物件的數量。
 - `labels`：回傳連通物件的標記，用數字1、2、3...表示。
 - `stats`：回傳各個連通物件的座標，以[x, y, width, height, area]五列的矩陣表示，分別代表x座標、y座標、長、寬、面積。
 - `centroids`：回傳各個連通物件的中心點
 - `image`：輸入影像，必須是二值圖，即8位元單通道影像。（因此輸入影像必須先進行二值化處理才能被這個函式接受）
 - `connectivity`：連通域，可以為8-連通或是4-連通，預設是8-連通
 - `ltype`：輸出影像標記的類型，目前支援CV_32S 和 CV_16U。
 - 例：`num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(image, connectivity=8, ltype=None)`



連通物件-OpenCV

- 範例程式：
 - [6]以灰階讀取影像
 - [7]二值化
 - [9, 10]以8-連通取得連通物件數量、編號、位置資訊(x, y, width, height, area)、中心點座標
 - [12]以for迴圈拜訪每個連通物件(注意：編號0的連通物件是黑色部分，因此i從1開始)
 - [13]取得每一個連通物件的座標資訊
 - [14, 15]在每一個連通物件左上角標上座標(x, y)

```
1  import cv2
2  import numpy as np
3
4  #連通物件
5  img_rgb = cv2.imread("circle.jpg")
6  img = cv2.imread("circle.jpg", 0)
7  ret, img = cv2.threshold(img, 80, 255, cv2.THRESH_BINARY)
8
9  num_labels, labels, stats, centroids = \
10     cv2.connectedComponentsWithStats(img, connectivity=8, ltype=cv2.CV_32S)
11
12  for i in range(1, num_labels):
13      x, y, w, h, area = stats[i]
14      cv2.putText(img_rgb, str("(")+str(x)+","+str(y)+str(")"), \
15                  (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255), 2)
16
17  cv2.imshow("circle_out", img_rgb)
18  cv2.waitKey(0)
19  cv2.destroyAllWindows()
```




連通物件-OpenCV

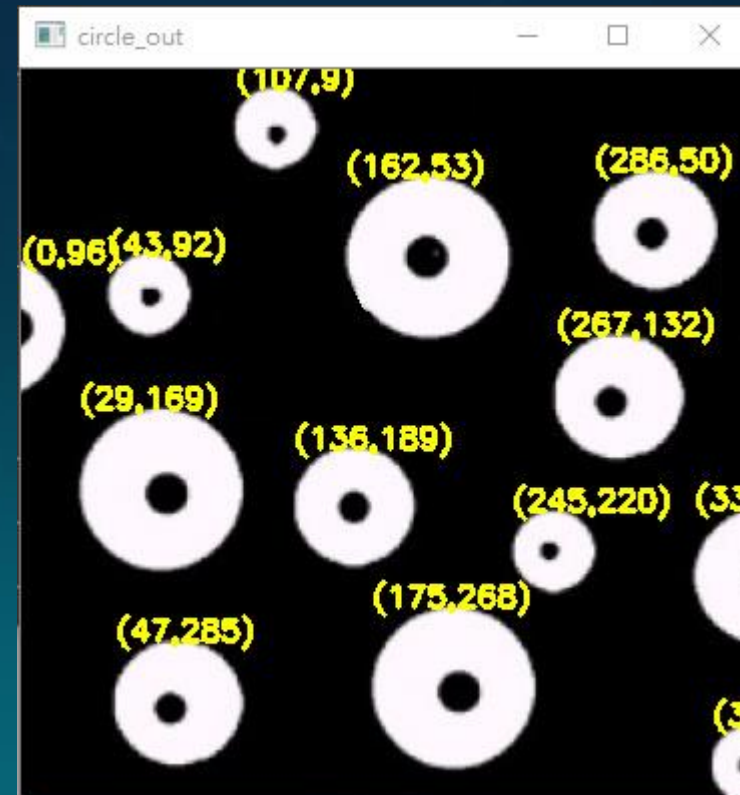
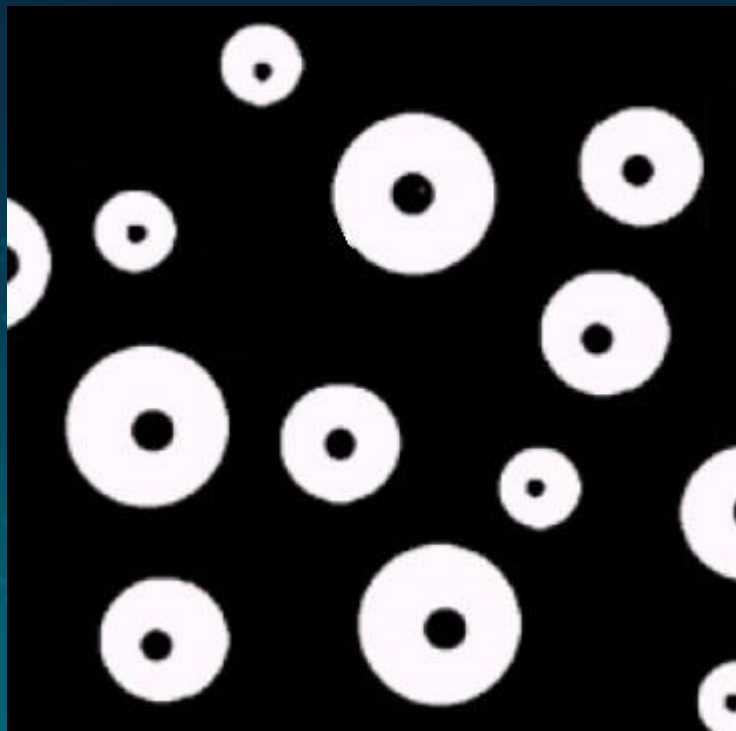




Image Processing Edge Detection



影像梯度(Image gradient)-簡介

- 影像梯度(Image gradient)是指影像強度和顏色的方向性變化。影像的梯度在影像處理中是基礎的一環。舉例來說，Canny邊緣檢測器(Canny edge detector)用計算影像梯度來做邊緣檢測。在數位影像處理的軟體中，影像的梯度也可以用來將顏色做漸進式的混合，例如左下圖中可以將黑色從上到下漸進的變成白色。



梯度變化微弱



梯度變化強烈



邊緣檢測(Edge Detection)-簡介

- 影像邊緣檢測，剔除了大部分認為與影像不相關的訊息，保留了影像重要的結構屬性。
- 邊緣點通常是影像中亮度變化明顯的點。影像屬性中的顯著變化通常反映了屬性的重要事件和變化。有以下幾種情況：
 - 深度上的不連續。
 - 表面方向不連續。
 - 物質屬性變化。
 - 場景照明變化。
- 因此，可透過影像中梯度變化強烈的地方來提取出邊緣線條。



邊緣檢測-常見方法

- Sobel
- Scharr
- Laplacian
- Canny



邊緣檢測 - Sobel

- Sobel是一離散性差分算子，用來運算影像亮度函數的梯度之近似值。
- 以下為Gx和Gy模板會與影像進行平面卷積，Gx用來檢測垂直邊緣，Gy用來檢查水平邊緣，分別對影像進行水平和垂直模板的運算，得到像素的梯度向量。

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

- 影像中每一個像素的橫向及縱向梯度近似值，可用以下公式計算梯度的大小G及方向 Θ 。

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$



邊緣檢測 — Sobel - OpenCV

OpenCV Sobel()函式

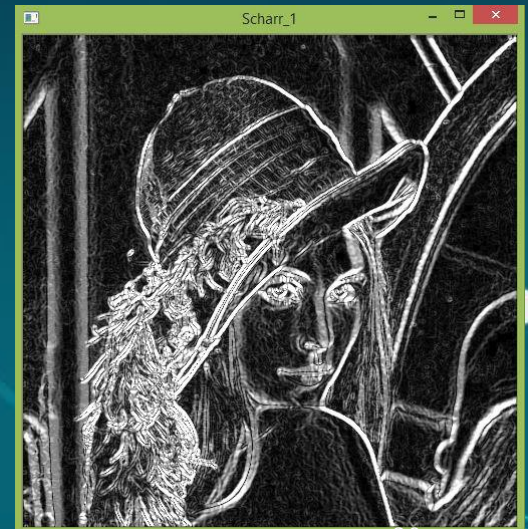
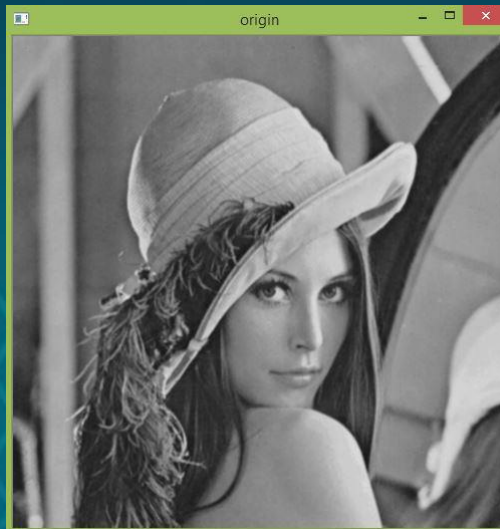
- `dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]])`
 - `src`：輸入影像。
 - `dst`：輸出影像，和輸入影像有相同的尺寸和通道數。
 - `ddepth`：輸出影像的深度，假設輸入影像為CV_8U，則支援CV_8U、CV_16S、CV_32F、CV_64F，假設輸入影像為CV_16U，則支援CV_16U、CV_32F、CV_64F。
 - `dx`：x方向的微分階數。
 - `dy`：y方向的微分階數。
 - `ksize`：核心大小，必須為1、3、5或7。
 - `scale`：縮放值
 - `delta`：偏移量。
 - `borderType`：邊界類型，邊界模式用來推斷影像外的像素
 - 例：`x = cv2.Sobel(img, cv2.CV_16S, 1, 0)`
 - 例：`y = cv2.Sobel(img, cv2.CV_16S, 0, 1)`



邊緣檢測 - Scharr

- Scharr運算出的梯度方向較Sobel精確，與Sobel的濾波係數不同。Scharr和Sobel相似，都擁有水平和垂直方向的模板，但Scharr只有3×3的模板。

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} \quad G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$





邊緣檢測 — Scharr - OpenCV

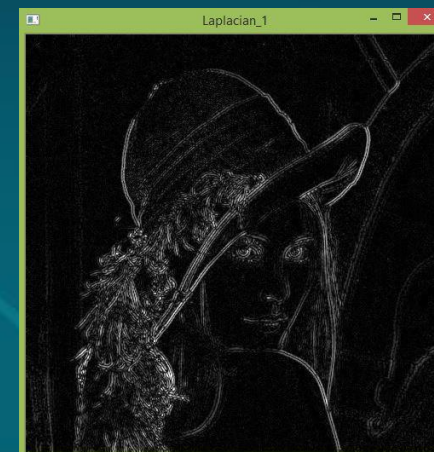
OpenCV Scharr()函式

- `dst = cv2.Scharr(src, ddepth, dx, dy[, dst[, scale[, delta[, borderType]]])`
 - src：輸入影像。
 - dst：輸出影像，和輸入影像有相同的尺寸和通道數。
 - ddepth：輸出影像的深度，使用方式和Sobel相同。
 - dx：x方向的微分階數。
 - dy：y方向的微分階數。
 - scale：縮放值
 - delta：偏移量。
 - borderType：邊界類型，邊界模式用來推斷影像外的像素
 - 例：`scharrx=cv2.Scharr(img,cv2.CV_64F,1,0)`
 - 例：`scharry=cv2.Scharr(img,cv2.CV_64F,0,1)`



邊緣檢測 – Laplacian

- 影像銳化分成一階微分及二階微分，兩者的參數都是由數學算式推導而成，拉普拉斯(Laplace)運算子是一種二階導數，且與方向無關的邊緣檢測運算子。
- 使用時，會先對原始影像進行拉普拉斯運算後，取絕對值得到輸出影像，再將輸出影像和原始影像進行混合相加，得到一個相似於原始影像，但是細節被強調的影像。





邊緣檢測 – Laplacian 算法

- 首先將 $f(x, y)$ 對 x 進行微分：

$$\nabla_x f = f(x + 1, y) - f(x, y)$$

- 再次對 x 微分得到二次微分：

$$\begin{aligned}\nabla_x^2 f &= f(x + 2, y) - f(x + 1, y) - [f(x + 1, y) - f(x, y)] \\ &= f(x + 2, y) - 2f(x + 1, y) + f(x, y)\end{aligned}$$

- 令 $x=x+1$ ，可得：

$$\nabla_x^2 f = f(x + 1, y) - 2f(x, y) + f(x - 1, y)$$



邊緣檢測 – Laplacian 算法

- 同理對y二次微分可得以下公式：

$$\nabla_y^2 f = f(x, y + 1) - 2f(x, y) + f(x, y - 1)$$

- 合併兩個二次微分，可得拉普拉斯算子公式：

$$\begin{aligned}\nabla^2 f &= \nabla_x^2 f + \nabla_y^2 f \\ &= f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y)\end{aligned}$$



邊緣檢測 – Laplacian 算法

- 將計算結果代入 3×3 濾波：

$f(x - 1, y + 1)$	$f(x, y + 1)$	$f(x + 1, y + 1)$
$f(x - 1, y)$	$f(x, y)$	$f(x + 1, y)$
$f(x - 1, y - 1)$	$f(x, y - 1)$	$f(x + 1, y - 1)$

- 由結果式可得到以下結果（以下為常用模板）：

0	1	0
1	-4	1
0	1	0



邊緣檢測 – Laplacian 算法 - OpenCV

OpenCV Laplacian() 函式

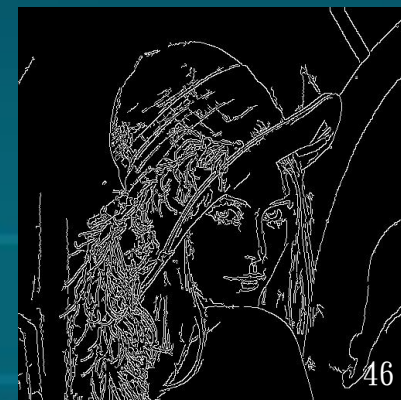
- `dst = cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]])`
 - `src`：輸入影像。
 - `ddepth`：輸出影像的深度，假設輸入影像為 `CV_8U`，支援 `CV_8U`、`CV_16S`、`CV_32F`、`CV_64F`，假設輸入影像為 `CV_16U`，支援 `CV_16U`、`CV_32F`、`CV_64F`。
 - `dst`：輸出影像，和輸入影像有相同的尺寸和通道數。
 - `ksize`：核心大小，輸入值必須為正整數。
 - `dx`：x 方向的微分階數。
 - `dy`：y 方向的微分階數。
 - `scale`：縮放值
 - `delta`：偏移量。
 - `borderType`：邊界類型，邊界模式用來推斷影像外的像素
 - 例：`gray_lap = cv2.Laplacian(img, cv2.CV_16S, ksize=3)`



邊緣檢測 - Canny

Canny邊緣檢測的目標是找到一個最佳的邊緣檢測演算法，最佳的邊緣檢測定義是：

- 演算法能夠標識出影像中的實際邊緣。
- 標識出的邊緣要與實際影像中的邊緣相近。
- 影像中的邊緣只能被標識一次，並且影像雜訊不能被標識為邊緣。





邊緣檢測 - Canny

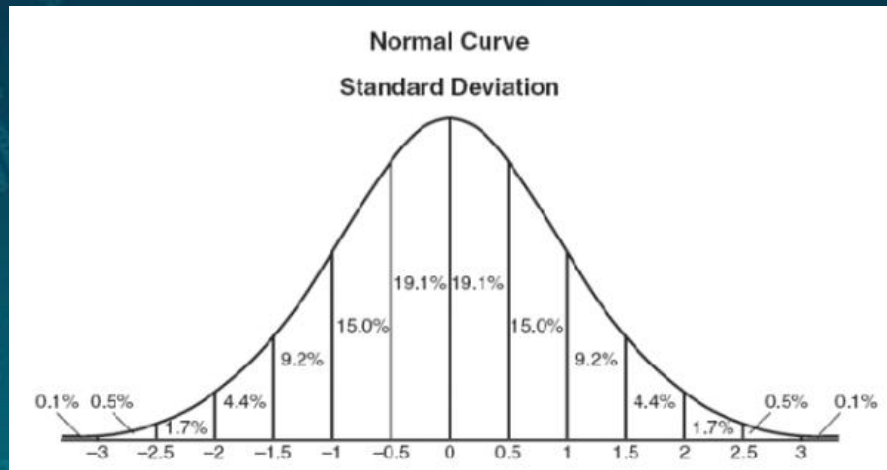
Canny演算流程：

- 計算邊緣點(Gradient intensity of image)。
- 對梯度Gradient進行**非極大值抑制**(NMS，抑制不是極大值的元素，可以理解為區域性最大搜索)。
- 使用兩個門檻值來檢測與連接邊緣輪廓特徵。



邊緣檢測 – Canny - Gradient Intensity of Image

- 透過高斯濾波將圖片進行模糊處理，原始像素為中心，有最大的高斯分布值與權重，相鄰像素權重隨著距離衰退。為一可以有效保留的邊緣濾波器。



二維高斯率波函式

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

5X5高斯濾波Kernel

$\frac{1}{159}$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2



邊緣檢測 – Canny - Gradient Intensity of Image

- 透過Sobel Operator的 G_x 和 G_y 模板與影像進行平面卷積， G_x 用來檢測垂直邊緣， G_y 用來檢查水平邊緣，得到像素的梯度向量。

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

- 並利用以下公式計算各像素梯度向量的大小 G 及方向 Θ 。

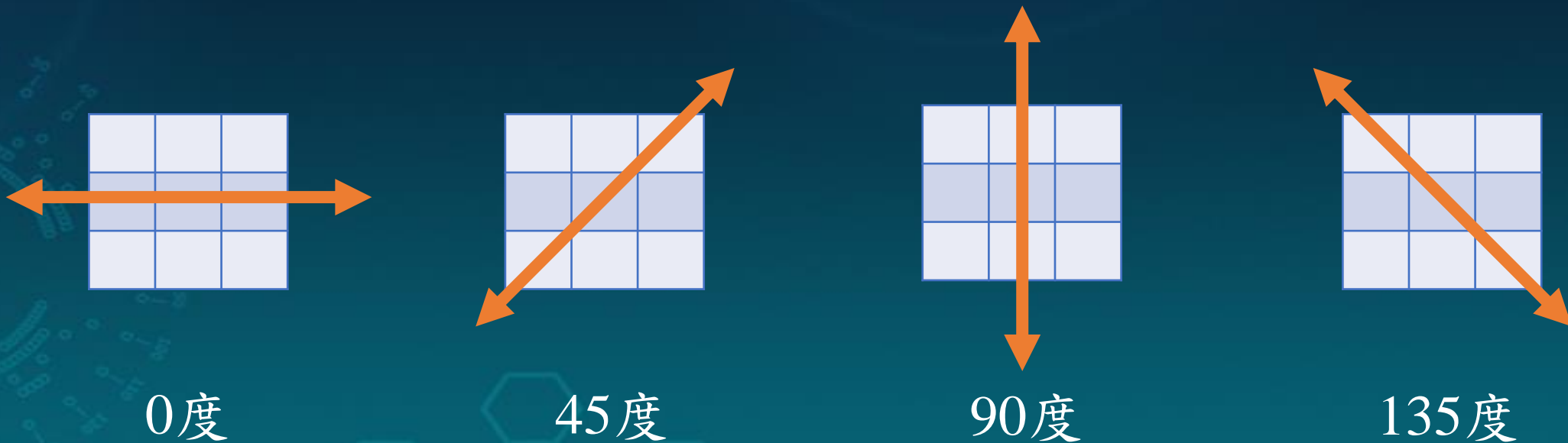
$$G = \sqrt{G_x^2 + G_y^2}$$

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$



邊緣檢測 – Canny - Gradient Intensity of Image

- 將梯度方向 Θ 簡化歸類為四個方向(0度、45度、90度、135度)





邊緣檢測 – Canny - Non-maximum Suppression

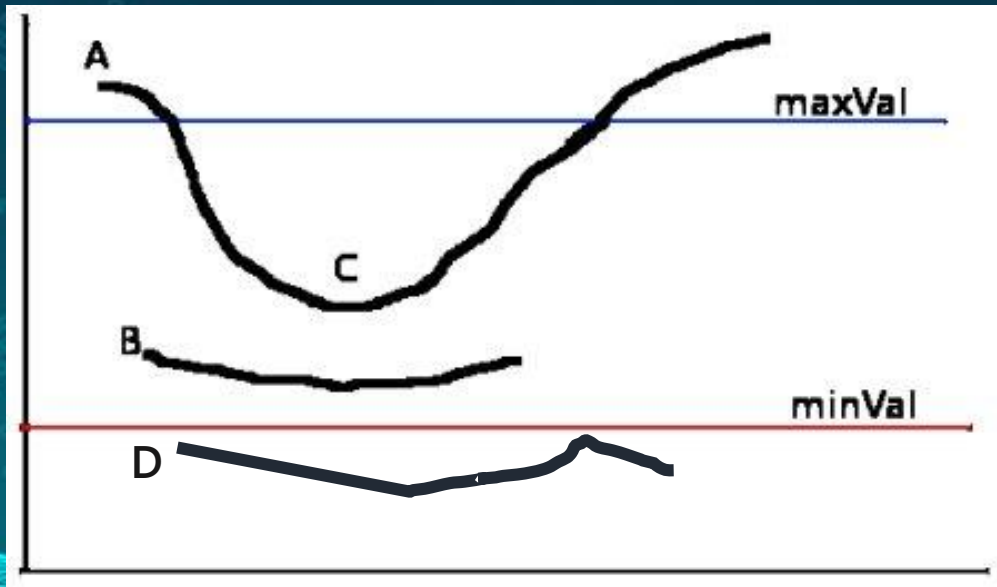
- 存在邊緣附近的像素，通常梯度強度不為0，若將其歸納為邊緣點，會產生非常粗的邊。因此若已經確定梯度方向，僅保留矩陣內方向強度最大的點，其餘該梯度方向的像素強度皆歸0。





邊緣檢測 – Canny - Hysteresis Thresholding

- 根據梯度方向留下適當的梯度強度對應的邊緣點後，尚需進行進行過濾的動作，使得找出來的邊緣能夠更連通且較無雜訊。
- 使用兩個閾值將像素梯度點以梯度強度進行分類



A: 大於maxVal閾值，為強邊緣點

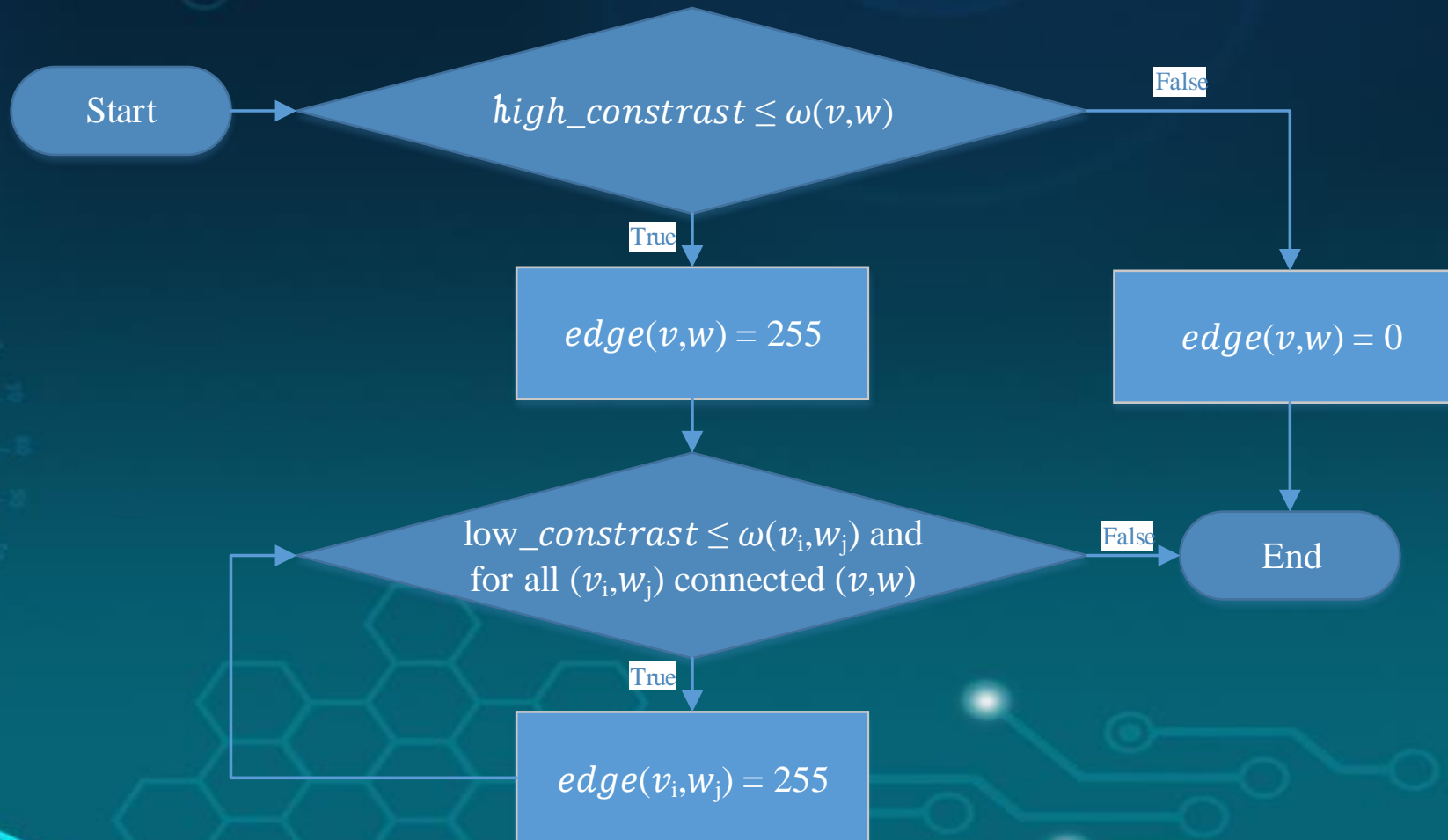
B: 介於兩閾值間，沒有與任何強邊緣點連接，不為邊緣點

C: 介於兩閾值間，但與強邊緣點A連接，為弱邊緣點

D: 小於minVal閾值，不為邊緣點



邊緣檢測 – Canny - Hysteresis Thresholding





邊緣檢測 – Canny - OpenCV

OpenCV Canny()函式:

- `edges = cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])`
 - image：輸入影像，單通道8位元圖。
 - edges：輸出影像，尺寸、型態和輸入影像相同。
 - threshold1：第一個門檻值
 - threshold2：第二個門檻值，用於檢測影像中明顯的邊緣，但一般情況下檢測的效果不會那麼完美，邊緣檢測出來是斷斷續續的。所以這時候用較小的第一個閾值用於將這些間斷的邊緣連線起來。
 - apertureSize：Sobel算子的核心大小。
 - L2gradient：梯度大小的算法，預設為false。
 - 例：`canny = cv2.Canny(img, 50, 150)`



邊緣檢測-OpenCV

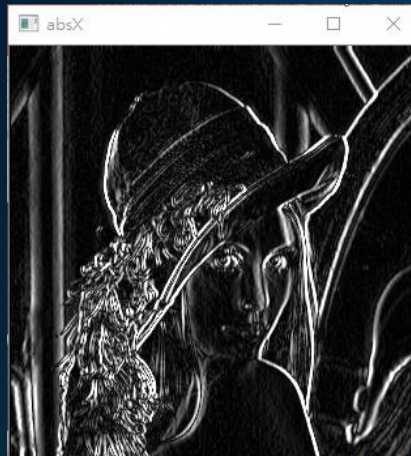
• 範例程式：

- [4]以灰階讀取原始影像
- [6, 7]分別對x, y方向做Sobel邊緣檢測
- [9, 10]Sobel函式求完導數後會有負值，還有會大於255的值，而原影像是uint8，即8位無符號數，所以Sobel建立的影像位數不夠，會有截斷，因此要使用16位有符號的資料型別，即cv2.CV_16S，在經過處理後，別忘了用convertScaleAbs()函式將其轉回原來的uint8形式，否則將無法顯示影像。
- [11]將x, y兩個方向的Sobel結果(已轉回uint8形式)結合
- [14]以3x3的kernel進行Laplacian邊緣檢測
- [15]將Laplacian結果轉回uint8形式
- [18]將3x3的kernel對灰階影像進行高斯模糊預處理
- [19]進行canny邊緣檢測
- [22, 23]分別對x, y方向做Scharr邊緣檢測
- [24, 25]分別將x, y方向的Scharr結果轉回uint8形式
- [26]將x, y兩個方向的Scharr結果(已轉回uint8形式)結合

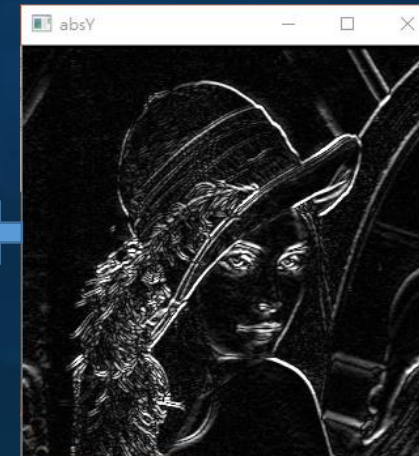
```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread("lenna.jpg", 0)
5
6 #sobel
7 x = cv2.Sobel(img, cv2.CV_16S, 1, 0)
8 y = cv2.Sobel(img, cv2.CV_16S, 0, 1)
9 absX = cv2.convertScaleAbs(x) # 轉回uint8
10 absY = cv2.convertScaleAbs(y)
11 sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)
12
13 #laplacian
14 gray_lap = cv2.Laplacian(img, cv2.CV_16S, ksize=3)
15 laplacian = cv2.convertScaleAbs(gray_lap) # 轉回uint8
16
17 #canny
18 img_c = cv2.GaussianBlur(img, (3, 3), 0)
19 canny = cv2.Canny(img_c, 50, 150)
20
21 #scharr
22 scharrx=cv2.Scharr(img,cv2.CV_64F,1,0)
23 scharry=cv2.Scharr(img,cv2.CV_64F,0,1)
24 scharrx=cv2.convertScaleAbs(scharrx) # 轉回uint8
25 scharry=cv2.convertScaleAbs(scharry)
26 scharrxy=cv2.addWeighted(scharrx,0.5,scharry,0.5,0)
27
28 cv2.imshow("origin", img)
29 cv2.imshow("absX", absX)
30 cv2.imshow("absY", absY)
31 cv2.imshow("Sobel", sobel)
32 cv2.imshow('Laplacian', laplacian)
33 cv2.imshow('Canny', canny)
34 cv2.imshow("scharrx",scharrx)
35 cv2.imshow("scharry",scharry)
36 cv2.imshow("scharrxy",scharrxy)
37 cv2.waitKey(0)
38 cv2.destroyAllWindows()
```



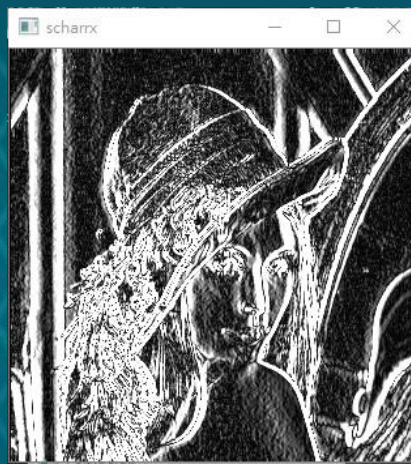
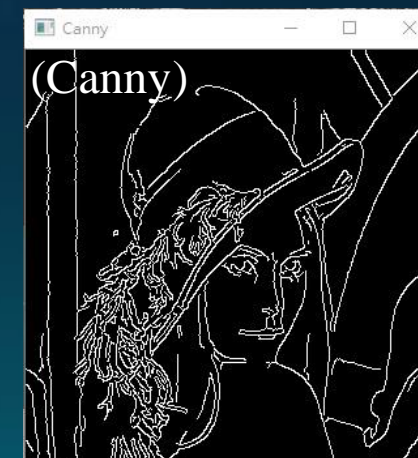
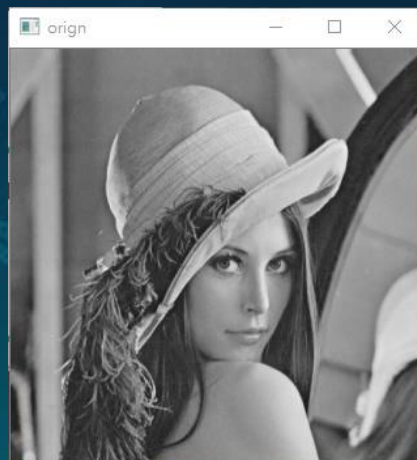

邊緣檢測 -OpenCV



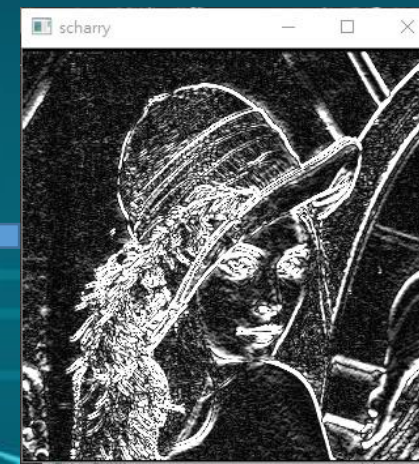
+



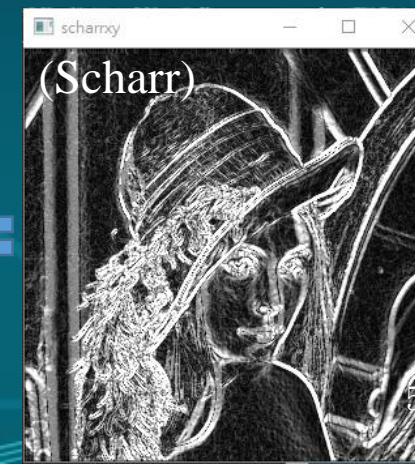
=



+



=





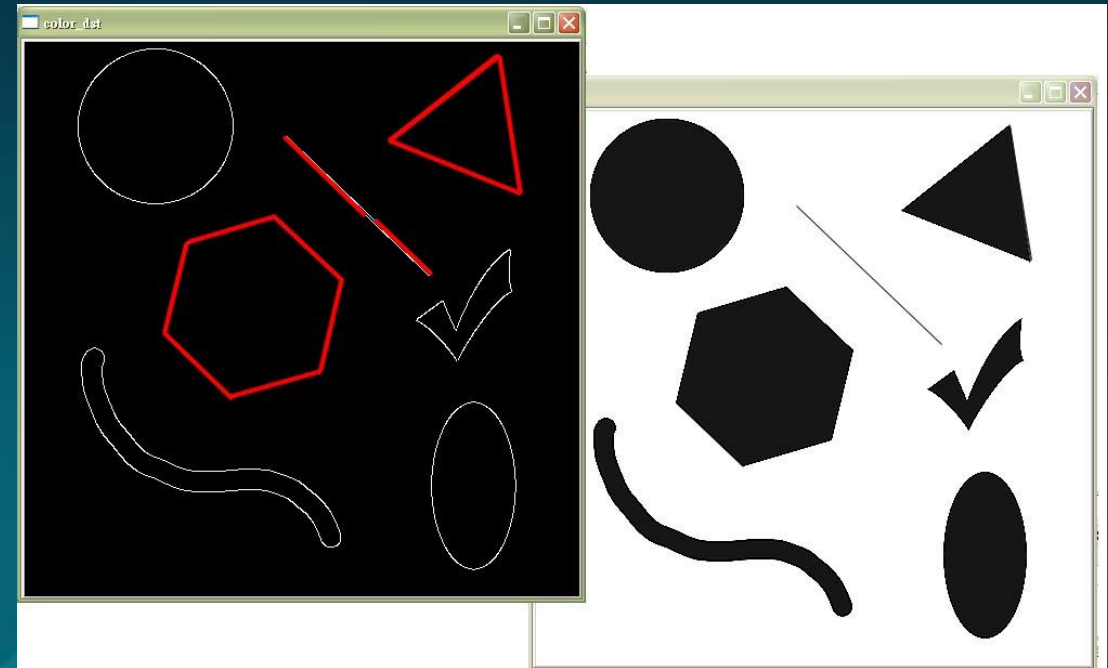
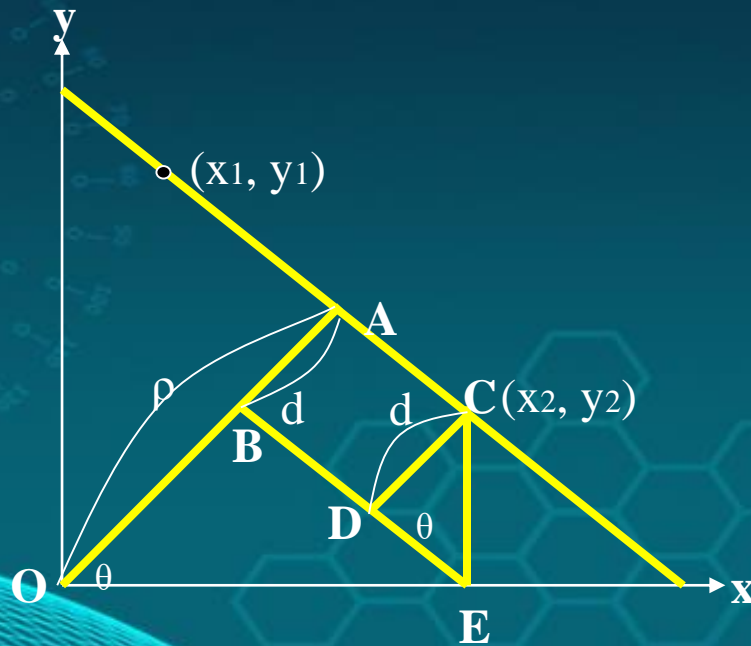
影像特徵的研究

- 「電腦自動地研究影像的特徵，從而判斷是否是某某人的面孔」，這可能做到嗎？隨著影像處理技術的進步，沒有什麼完全辦不到的事情。現在，自動售貨機已可以準確地區別一百元或千元的紙鈔了。在工廠中，利用攝影機也能自動判別不合格產品。利用自動識別指紋影像，代替電子鑰匙的設備也已經出現。
- 本章將先介紹影像處理中的直線特徵偵測方法，接著再以一影像為例，透過對物體形狀和大小特徵的研究，對擷取必要物體，去除不必要雜物的有關方法，做些基本說明。



直線偵測 - Hough轉換法

基本上，Hough轉換法的精神為將 $x-y$ 空間轉換為 $\rho-\theta$ 參數空間（Parameter Space），即所謂的法距—法角空間（Normal Distance-Normal Angle Space）。 $x-y$ 及 $\rho-\theta$ 空間的關係如下圖所示。





直線偵測 - Hough轉換法

令線段 \overline{AB} 長為 d ，線段 \overline{OA} 的長度為 r 。邊點 (x_1, y_1) 和邊點 (x_2, y_2) 為共線。從邊點 (x_2, y_2) 得知 $\overline{CE} = y_2$ 和 $\overline{OE} = x_2$ 。又由直角三角形 $\triangle CDE$ 可得知：

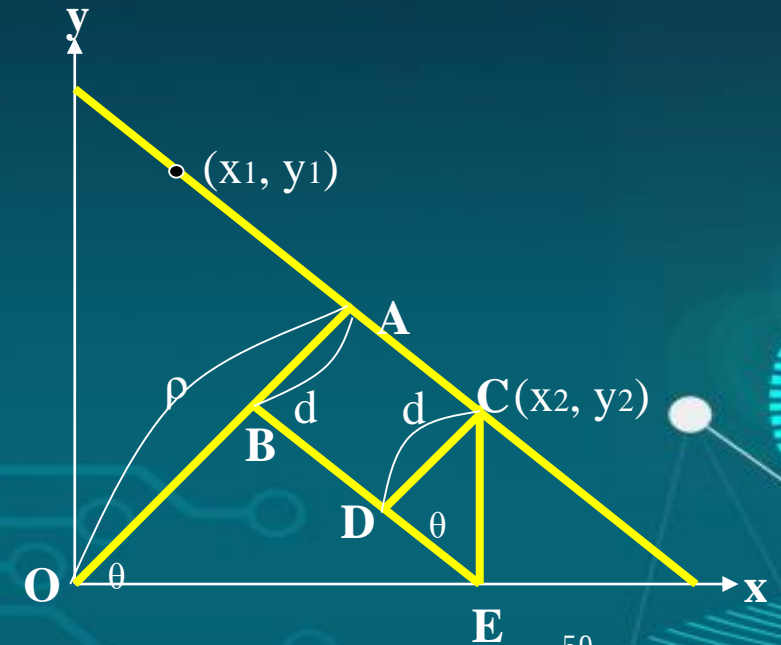
$$d = y_2 \sin \theta = \overline{AB} \dots\dots(1)$$

由直角三角形 $\triangle OBE$ 又得知：

$$\overline{OB} = \overline{OE} \cos \theta = x_2 \cos \theta \dots\dots(2)$$

我們進而得知下列的式子：

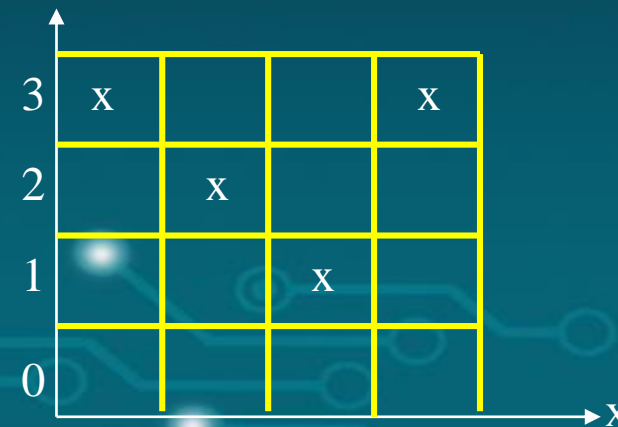
$$\rho = \overline{OB} + \overline{AB} = x_2 \cos \theta + y_2 \sin \theta \dots\dots(3)$$





直線偵測 - Hough轉換法

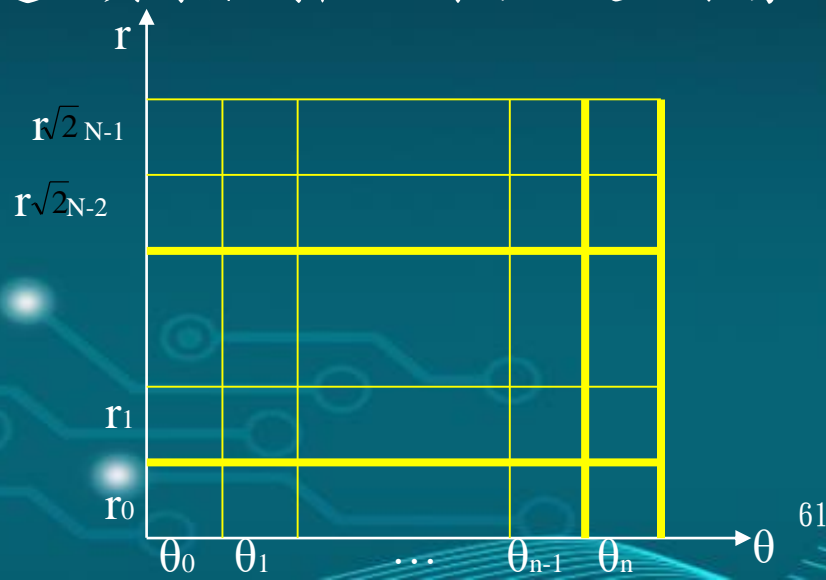
- 上式(3)在Hough轉換法中為共線上的邊點進行投票時的重要依據。這裡舉個例子以便大家更了解所謂的Hough轉換法。
- 假設給一4x4的影像，如下圖所示，符號X表示邊點所在。下圖的四個邊點座標分別為(2, 1)、(1, 2)、(0, 3)和(3, 3)。令 $\theta = 45^\circ$ ，將四個點的座標帶入上式並分別求得 ρ 值為 $\frac{3\sqrt{2}}{2}$ 、 $\frac{3\sqrt{2}}{2}$ 、 $\frac{3\sqrt{2}}{2}$ 以及 $3\sqrt{2}$ ，依照這四個 ρ 值，我們可以得知(2, 1)、(1, 2)和(0, 3)為共線，假設邊點共線數門檻值為2，可得知在該圖中有一條角度為45度之線段通過，符合我們視覺所見。





直線偵測 - Hough轉換法

- 另一問題是我們如何得知 $\theta = 45$ 度是合適的角度猜測值。基本上我們可將角度範圍 $[0, \pi]$ 切割成 n 份。例如每隔5度切一份，則在 $[0, \pi]$ 之間可切割出37份角度，這些角度分別為 $\theta_0 = 0$ 、 $\theta_1 = 5$ 、 $\theta_2 = 10$ 、...和 $\theta_{36} = \pi$
- 如下圖所示，先從 θ_0 開始，將所有的邊點一一代入公式，可得 $|V|$ 個 r 值，在這 $|V|$ 個法距值中有些值是相同的，而且這些近似法距值的邊點會掉在同一個位置上，這些位置通常稱為小區間（Cell）。同理，我們繼續計算 θ_1 至 θ_n 並進行投票動作，每個小房子會紀錄那些共線的邊點數，若在某一個小房子內，其紀錄的邊點數超過邊點共線數門檻，則可設定該小房子內的邊點數為一條可接受之直線。





直線偵測 - Hough - OpenCV

OpenCV HoughLines()函式:

- `lines = cv2.HoughLines(image, rho, theta, threshold)`
 - `image`: 輸入影像，8位元單通道二值化圖。
 - `lines`: `lines` 中的每個元素都是一對浮點數，表示檢測到的直線的參數，即 (r, θ) ，是 `numpy.ndarray` 類型。
 - `rho`: 距離解析度，越小表示定位要求越準確，但也較易造成應該是同條線的點判為不同線。
 - `theta`: 角度解析度，越小表示角度要求越準確，但也較易造成應該是同條線的點判為不同線。
 - `threshold`: 門檻值，超過此值的線才會存在`lines`這個容器內。
 - 例: `lines = cv2.HoughLines(edges, 1, np.pi/180, 200)`



直線偵測 - Hough - OpenCV

OpenCV HoughLinesP()函式:

- `lines = cv2.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]])`
 - `image`: 輸入影像，8位元單通道二值化圖。
 - `lines`: 將所有線的資料存在`numpy.array()`，每列代表每個線段的資料，分別有`x1`、`y1`、`x2`、`y2`這四個值，`(x1, y1)`和`(x2, y2)`分別表示線段的頭尾頂點。
 - `rho`: 距離解析度，越小表示定位要求越準確，但也較易造成應該是同條線的點判為不同線。
 - `theta`: 角度解析度，越小表示角度要求越準確，但也較易造成應該是同條線的點判為不同線。
 - `threshold`: 門檻值，超過此值的線才會存在`lines`這個容器內。
 - `minLineLength`: 線段最短距離，超過此值的線才會存在`lines`這個容器內。
 - `maxLineGap`: 同一條直線上點與點之間的最大允許間隔。
 - 例: `lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 100, 100, 10)`



直線偵測 - Hough - OpenCV

- 機率霍夫變換(`cv2.HoughLinesP()`)對基本霍夫變換(`cv2.HoughLines()`)演算法進行了一些修正，是霍夫變換演算法的改善。它沒有考慮所有的點。相反，它只需要一個足以進行線檢測的隨機點子集即可。
- 為了更好地判斷直線(線段)，機率霍夫變換演算法還對選取直線的方法作了兩點改進：
 - 所接受直線的最小長度：如果有超過門檻值個數的像素點構成了一條直線，但是這條直線很短，那麼就不會接受該直線作為判斷結果，而認為這條直線僅僅是圖像中的若干個像素點恰好隨機構成了一種演算法上的直線關係而已，實際上原圖中並不存在這條直線。
 - 接受直線時允許的最大像素點間距：如果有超過門檻值個數的像素點構成了一條直線，但是這組像素點之間的距離都很遠，就不會接受該直線作為判斷結果，而認為這條直線僅僅是圖像中的若干個像素點恰好隨機構成了一種演算法上的直線關係而已，實際上原始圖像中並不存在這條直線。



直線偵測 - Hough - OpenCV

- `cv2.HoughLines()`與`cv2.HoughLinesP()`函式的差別





圓形偵測 - Hough - OpenCV

OpenCV HoughCircles()函式:

- `circles = cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]])`
 - `image`: 輸入影像，8位元單通道圖
 - `circles`: 以`numpy.array()`記錄所有圓的資訊，包含`x`、`y`、`radius`。
 - `method`: 偵測圓的方法，目前只能使用`CV_HOUGH_GRADIENT`。
 - `dp`: 偵測解析度倒數比例，假設`dp=1`，偵測圖和輸入影像尺寸相同，假設`dp=2`，偵測圖長和寬皆為輸入影像的一半。
 - `minDist`: 圓彼此間的最短距離，太小的話可能會把鄰近的幾個圓視為一個，太大的話可能會錯過某些圓。
 - `param1`: 圓偵測內部會呼叫`Canny()`尋找邊界，`param1`就是`Canny()`的高門檻值，低門檻值自動設為此值的一半。
 - `param2`: 門檻值，超過此值的圓才會存入`circles`。
 - `minRadius`: 最小的圓半徑。
 - `maxRadius`: 最大的圓半徑。
 - 例: `circle1 = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 100, param1=100, param2=30, minRadius=100, maxRadius=200)`



圓形偵測 – Hough-範例

