Embedded Vision Intelligent Laboratory

多媒體技術與應用
Spring 2021

Instructor： Yen-Lin Chen(陳彥霖), Ph.D.
Professor
Dept. Computer Science and Information Engineering
National Taipei University of Technology

Lecture 3

OpenCV物件檢測技術

物件檢測概述和應用

物件檢測概述

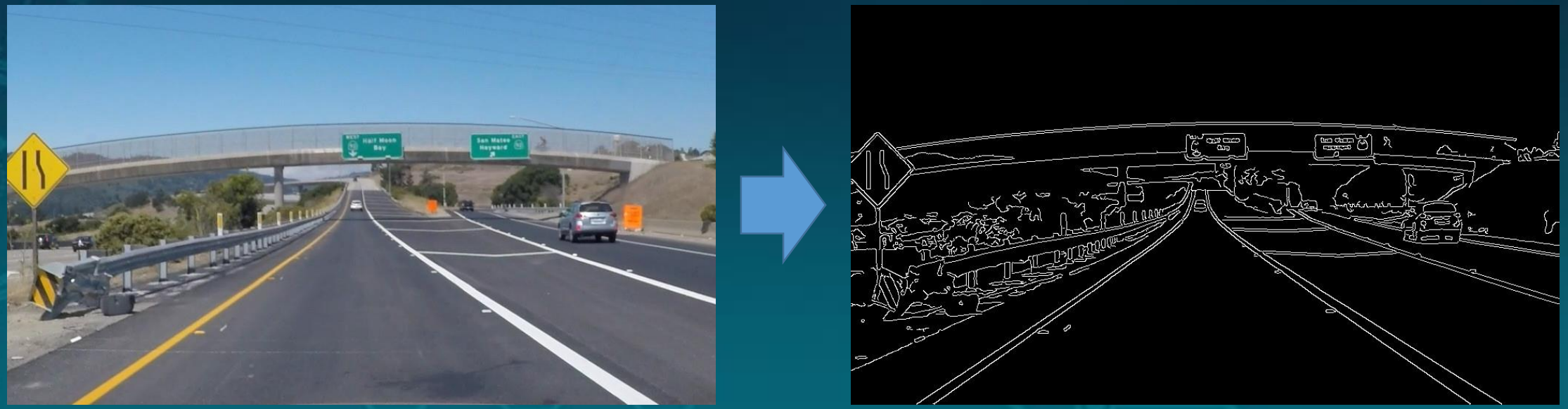
- 物件檢測顧名思義就是要找出影像中感興趣的物件，這個方法可以透過基本的影像處理，如：二值化或影像相減等技術來達成，近年也流行使用深度學習的方式，如：YOLOv4來辨識影像中的物件。
- 本次課程將介紹如何使用連通物件或邊緣檢測等基本的影像處理技術來找出影像中感興趣的物件。

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象。

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。



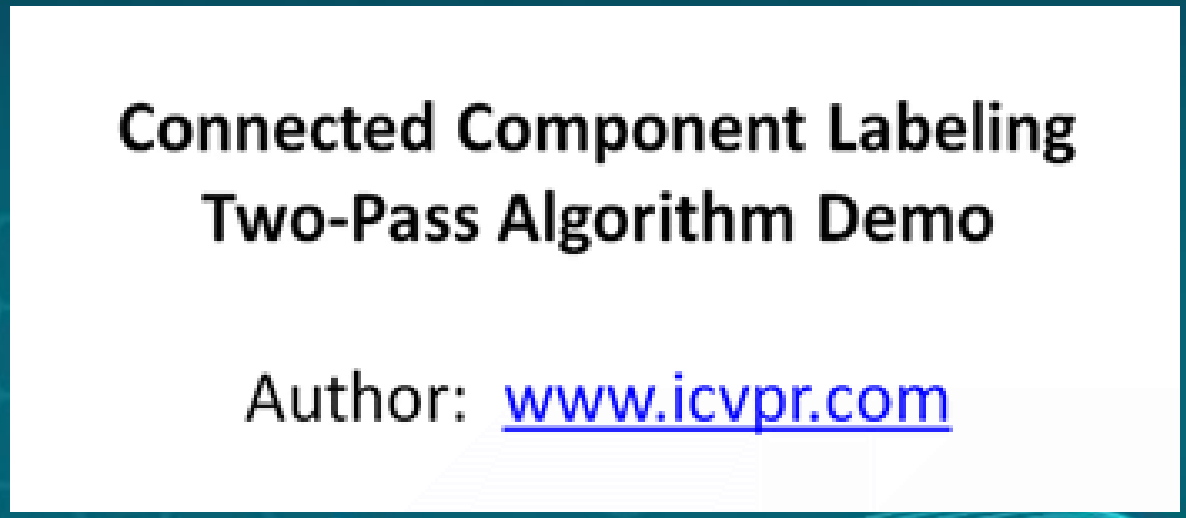
圖為經過Canny處理後的結果

連通物件

連通物件(Connected Component)-概述

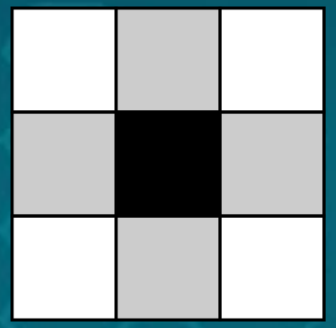
- 連接物件標記演算法(connected component labeling algorithm)是影像分析中最常用的演算法之一，演算法的實質是掃描二值影像的每個圖元點，對於圖元值相同的而且相互連通分為相同的組(group)，最終得到影像中所有的圖元連通元件。
- 連通區域一般是指影像中具有相同圖元值且位置相鄰的前景圖元點組成的影像區域。連通區域分析是指將影像中的各個連通區域找出並標記。

https://img-blog.csdn.net/20130825153036000?fbclid=IwAR21RPkU2wtijoUGt8U6rfWWdyvnsCMLKSHq7nb0EyIHiRAzzznwIlSMjdk

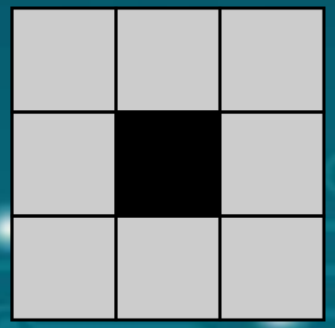


連通物件(Connected Component)-概述

- 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。



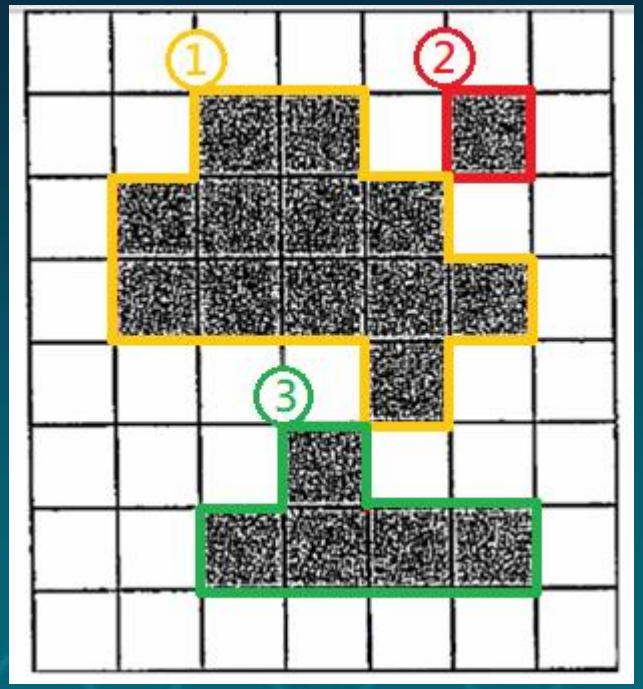
4-connected



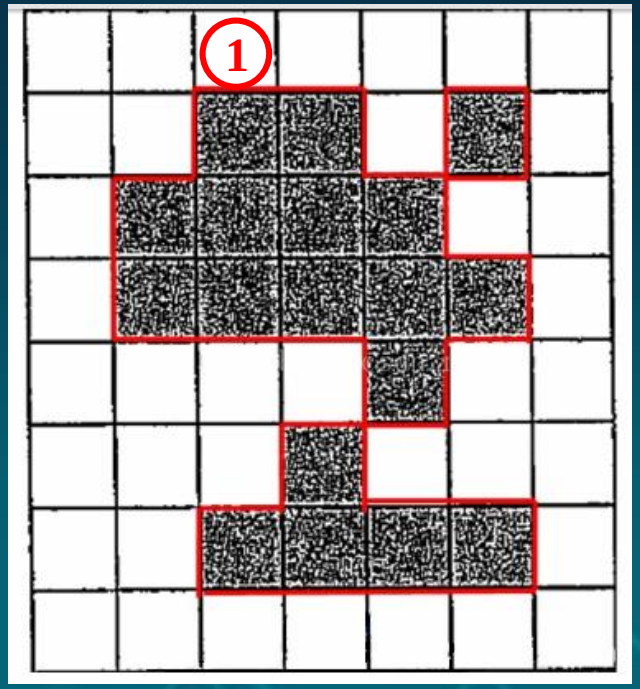
8-connected

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

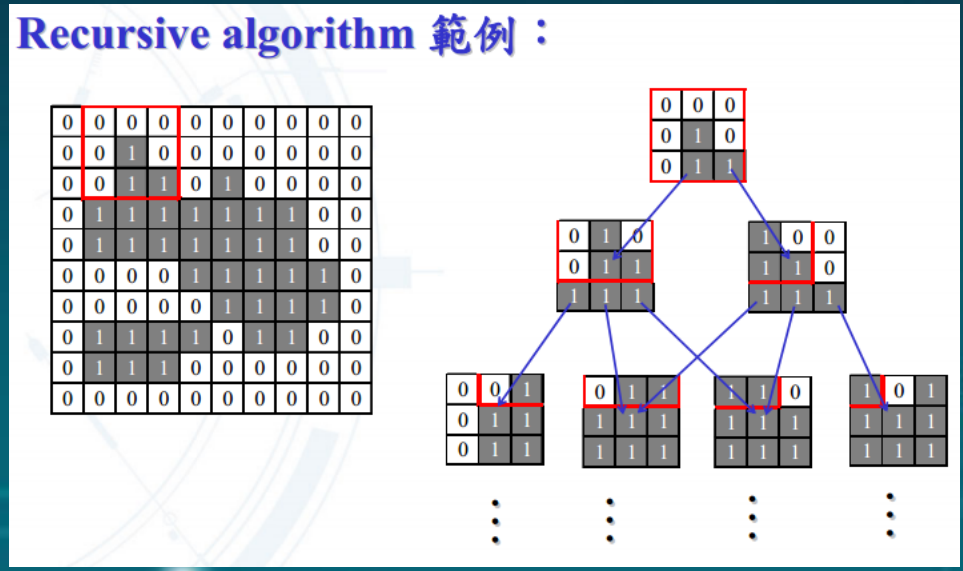


8-connected

連通物件(Connected Component)-演算法

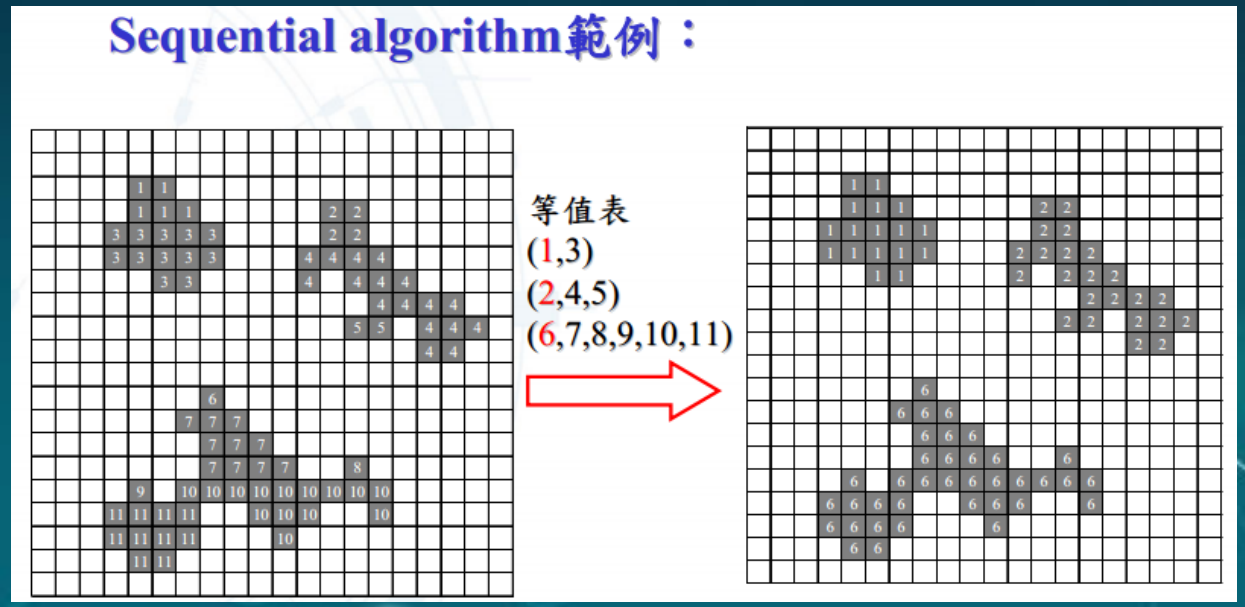
- 一、Recursive algorithm (遞歸法)
  1. 遞歸法首先掃瞄二值化影像，找出灰階值為1且未經標號的像素，並指定一個未經使用的號碼給該像素。
  2. 接下來利用8-connected 連通的原理，找出灰階值同為1之近鄰像素，並指定同一個標號給該像素，然後以遞迴的方式持續進行，直到沒有任何近鄰之像素值為1為止。

- Recursive algorithm 執行步驟：
  1. 掃瞄找出灰階值為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。



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，依序掃瞄影像。
  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」掃瞄影像。
  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. 若第一回合的掃瞄未完成，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. 進行第二回合的掃瞄，將各組等值標號以其最小值取代。



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)

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。

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

- 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。

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

邊緣檢測 – Sobel - OpenCV

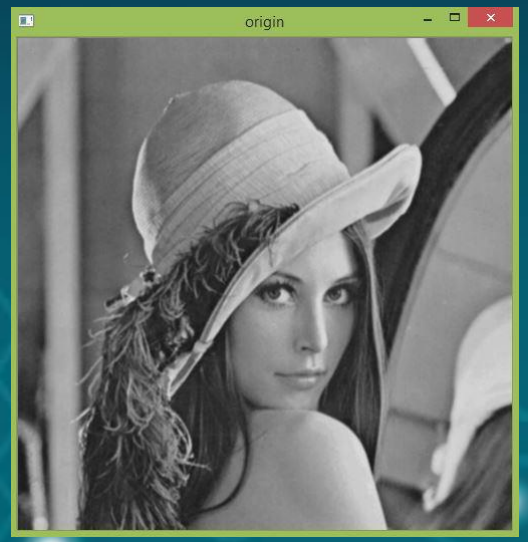
OpenCV Sobel()函式

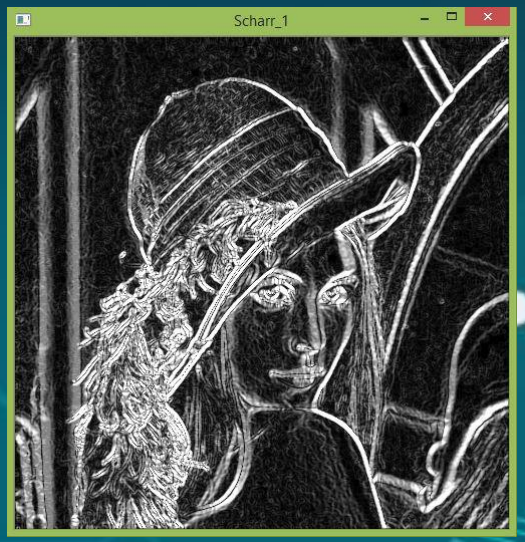
- dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])
  - src：輸入影像。
  - dst：輸出影像，和輸入影像有相同的尺寸和通道數。
  - ddepth：輸出影像的深度，假設輸入影像為CV_8U, 則支援CV_8U、CV_16S、CV_32F、CV_64F，假設輸入影像為 CV_16U, 則支援CV_16U、CV_32F、CV_64F 。
  - dx：x方向的微分階數。
  - dy：y方向的微分階數。
  - ksize：核心大小，必須為1、3、5或7。
  - 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3x3的模板。

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





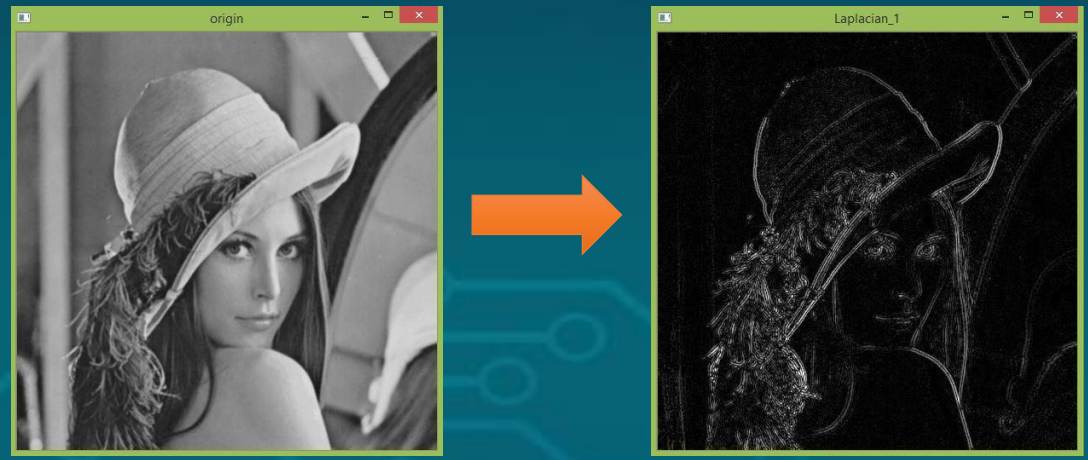
邊緣檢測 – Scharr - OpenCV

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：

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

- 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：

$$\nabla^2 f = \nabla_x^2 f + \nabla_y^2 f$$
$$= f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

邊緣檢測 – Laplacian算法

- 將計算結果代入3x3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

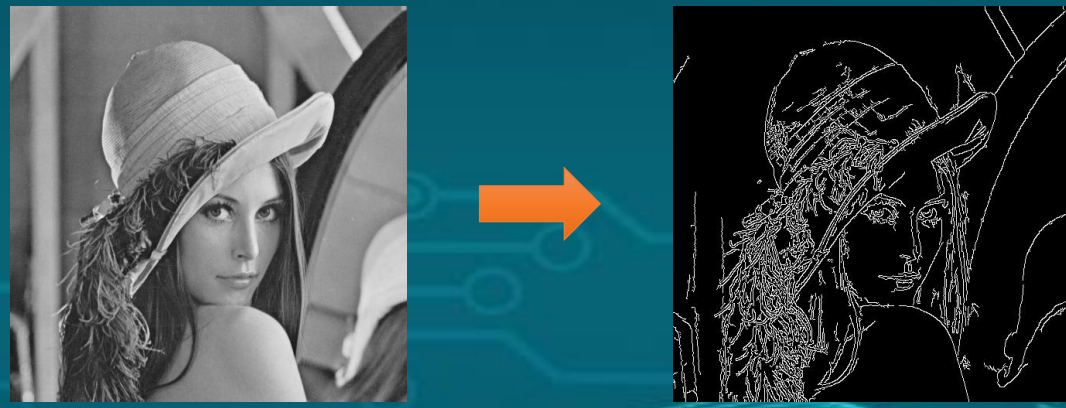
OpenCV Laplacian()函式

- 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：

- 好的檢測-演算法能夠標識出影像中的實際邊緣。
- 好的定位-標識出的邊緣要與實際影像中的邊緣相近。
- 最小的響應-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。

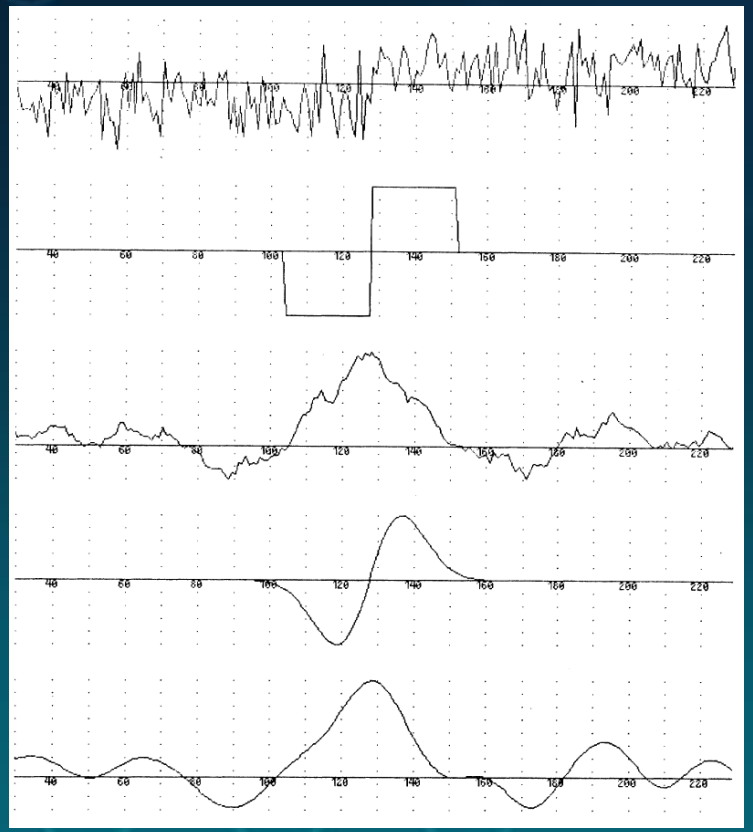


邊緣檢測 - Canny

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



(a) Noisy edge

(b) Impulse response (box operator)

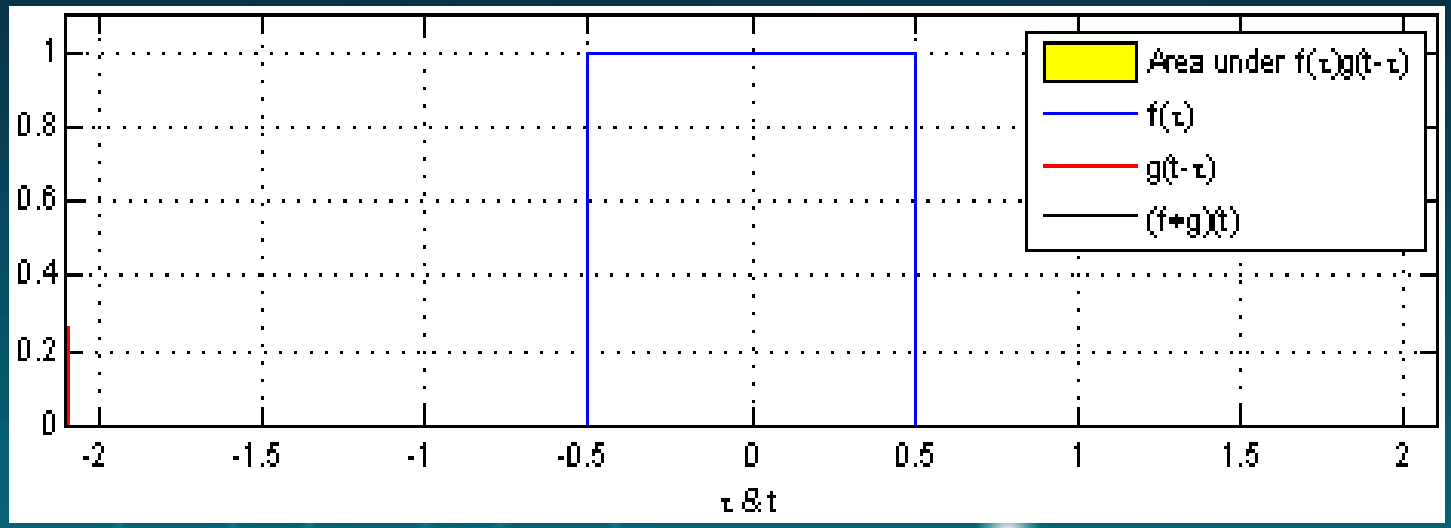
(c) Convolution (a)(b)

(d) First derivative of Gaussian operator

(e) Convolution (c)(d)

邊緣檢測 – Canny - Gradient Intensity of Image

令函數f,g是定義在$\mathbb{R}^N$上的可測函數，f與g的卷積(Convolution)記作$f*g$，它是其中一個函數翻轉並平移後與另一個函數的乘積的積分，是一個對平移量的函數。



$$(f*g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

邊緣檢測 – Canny - Gradient Intensity of Image

但由於對影像對x、y方向套用上述的Convolution進行偏微分過程過於複雜，因此透過使用Irwin Sobel（即Sobel Operator的提出者之一）於1968年提出的Gradient Operator for Image Processing，可以達到相近且快速的效果。

$$\Delta X_1(v,w) = \sum_{i=0}^{2}\sum_{j=0}^{2} \Delta x_1(i,j) \cdot I(v+i-1, w+j-1)$$

$$\Delta Y_{1(v,w)} = \sum_{i=0}^{2}\sum_{j=0}^{2} \Delta y_1(i,j) \cdot I(v+i-1, w+j-1)$$

$$M(v,w) = \sqrt{\Delta X_1(v,w)^2 + \Delta Y_1(v,w)^2}$$

$$\Delta x_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\Delta y_1 = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

邊緣檢測 – Canny - Non-maximum Suppression

於梯度強度圖$M(v,w)$上的任一(v,w)之值越大，僅代表該點上有著較大的梯度強度，並不能代表該點即為邊緣點。因此將該點之梯度方向映射回該點之八相鄰的點，透過比較該點與其映射點的梯度強度，而從決定該點是否為邊緣點。

邊緣檢測 – Canny - Non-maximum Suppression

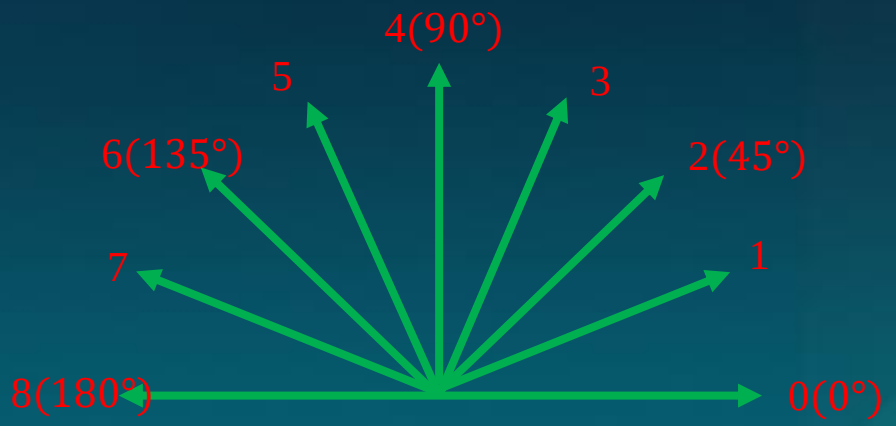
$$\theta(v,w) = \tan^{-1}\left(\frac{\Delta Y_1(v,w)}{\Delta X_1(v,w)}\right)$$

$$Partition\ Number(PN) = \frac{((\theta(v,w) + \pi)\ mod\ \pi)}{\pi} \times 8$$

$$\begin{cases} d1 = dir1_i, d2 = dir2_i \ \ if\ LBi < PN \le HBi, i = 0 \cdots 2 \\ d1 = dir13, d2 = dir23\ , else\ if\ LB_3 < PN\ or\ PN \le HB_3 \end{cases}$$

$$dir1 \leftarrow \{ne, nn, nw, ee\}, dir2 \leftarrow \{se, ss, sw, ww\}$$

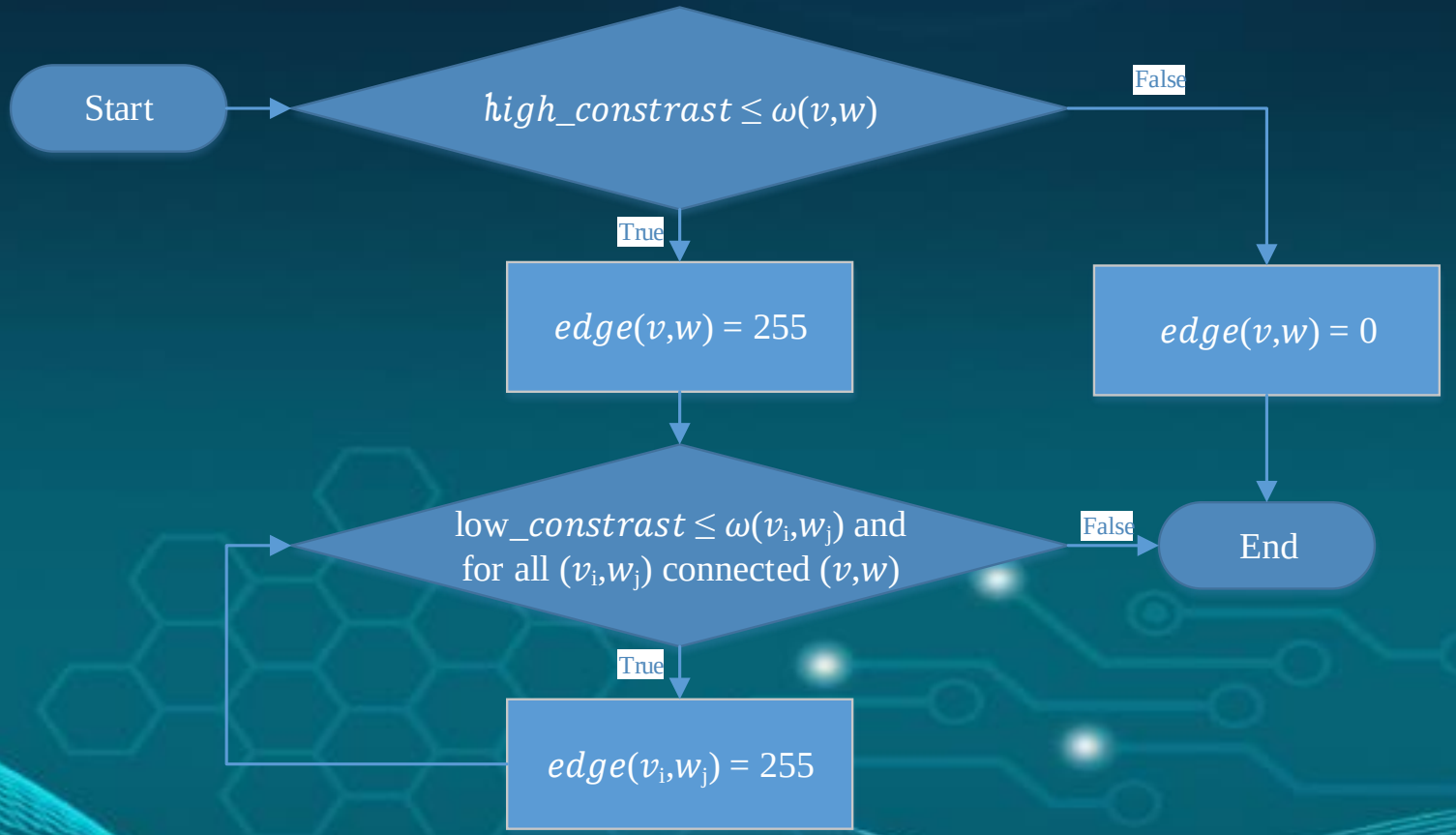
$$LB \leftarrow \{1,3,5,7\}, HB \leftarrow \{3,5,7,1\}$$



$$\begin{cases} \omega(v,w) = M(v,w), if\ M(d1) \le M(v,w)\ and\ M(d2) \le M(v,w) \\ \omega(v,w) = 0, otherwise \end{cases}$$

邊緣檢測 – Canny - Double threshold

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。



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，用於檢測影象中明顯的邊緣，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，而原影象是uint8，即8位無符號數，所以Sobel建立的影象位數不夠，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，否則將無法顯示影象，而只是一副灰色的視窗。
  - [11]將x, y兩個方向的sobel結果(已轉回uint8形式)結合
  - [14, 15, 16, 18]分別秀出原圖、x方向sobel結果圖、y方向sobel結果圖、 x, y兩個方向的sobel結合結果

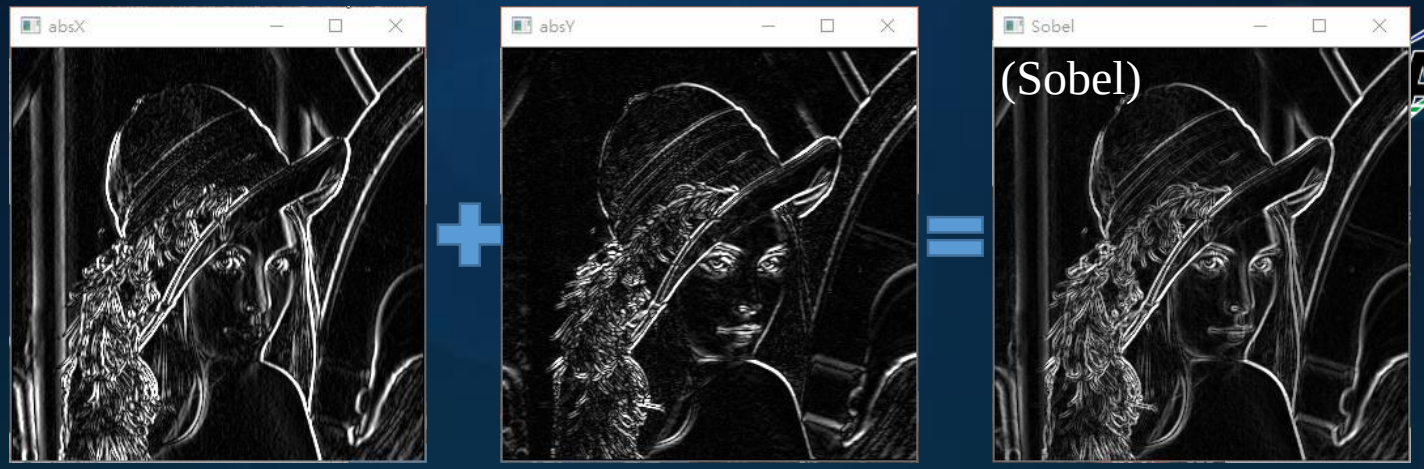
```python
import cv2
import numpy as np

img = cv2.imread("lenna.jpg", 0)

#sobel
x = cv2.Sobel(img, cv2.CV_16S, 1, 0)
y = cv2.Sobel(img, cv2.CV_16S, 0, 1)
absX = cv2.convertScaleAbs(x)# 轉回uint8
absY = cv2.convertScaleAbs(y)
sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#laplacian
gray_lap = cv2.Laplacian(img, cv2.CV_16S, ksize=3)
laplacian = cv2.convertScaleAbs(gray_lap) # 轉回uint8

#canny
img_c = cv2.GaussianBlur(img, (3, 3), 0)
canny = cv2.Canny(img_c, 50, 150)

#scharr
scharrx=cv2.Scharr(img,cv2.CV_64F,1,0)
scharry=cv2.Scharr(img,cv2.CV_64F,0,1)
scharrx=cv2.convertScaleAbs(scharrx)# 轉回uint8
scharry=cv2.convertScaleAbs(scharry)
scharrxy=cv2.addWeighted(scharrx,0.5,scharry,0.5,0)

cv2.imshow("orign", img)
cv2.imshow("absX", absX)
cv2.imshow("absY", absY)
cv2.imshow("Sobel", sobel)
cv2.imshow('Laplacian', laplacian)
cv2.imshow('Canny', canny)
cv2.imshow("scharrx",scharrx)
cv2.imshow("scharry",scharry)
cv2.imshow("scharrxy",scharrxy)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

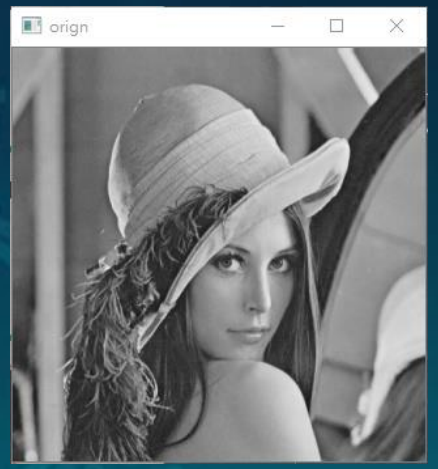
邊緣檢測-OpenCV

- 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，而原影象是uint8，即8位無符號數，所以Sobel建立的影象位數不夠，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，否則將無法顯示影象，而只是一副灰色的視窗。
  - [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Schaar邊緣檢測。
  - [24, 25]分別將x, y方向的Schaar結果轉回uint8形式
  - [26]將x, y兩個方向的Schaar結果(已轉回uint8形式)結合

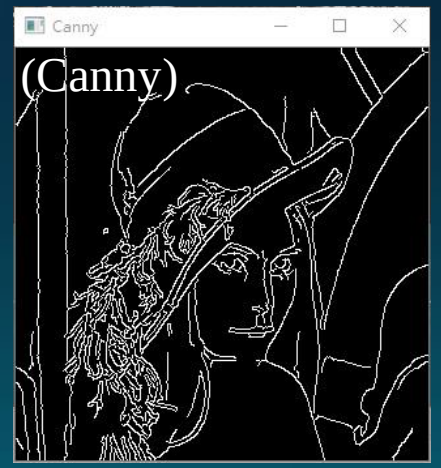
```python
import cv2
import numpy as np

img = cv2.imread("lenna.jpg", 0)

#sobel
x = cv2.Sobel(img, cv2.CV_16S, 1, 0)
y = cv2.Sobel(img, cv2.CV_16S, 0, 1)
absX = cv2.convertScaleAbs(x)# 轉回uint8
absY = cv2.convertScaleAbs(y)
sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

#laplacian
gray_lap = cv2.Laplacian(img, cv2.CV_16S, ksize=3)
laplacian = cv2.convertScaleAbs(gray_lap) # 轉回uint8

#canny
img_c = cv2.GaussianBlur(img, (3, 3), 0)
canny = cv2.Canny(img_c, 50, 150)

#scharr
scharrx=cv2.Scharr(img,cv2.CV_64F,1,0)
scharry=cv2.Scharr(img,cv2.CV_64F,0,1)
scharrx=cv2.convertScaleAbs(scharrx)# 轉回uint8
scharry=cv2.convertScaleAbs(scharry)
scharrxy=cv2.addWeighted(scharrx,0.5,scharry,0.5,0)

cv2.imshow("orign", img)
cv2.imshow("absX", absX)
cv2.imshow("absY", absY)
cv2.imshow("Sobel", sobel)
cv2.imshow('Laplacian', laplacian)
cv2.imshow('Canny', canny)
cv2.imshow("scharrx",scharrx)
cv2.imshow("scharry",scharry)
cv2.imshow("scharrxy",scharrxy)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

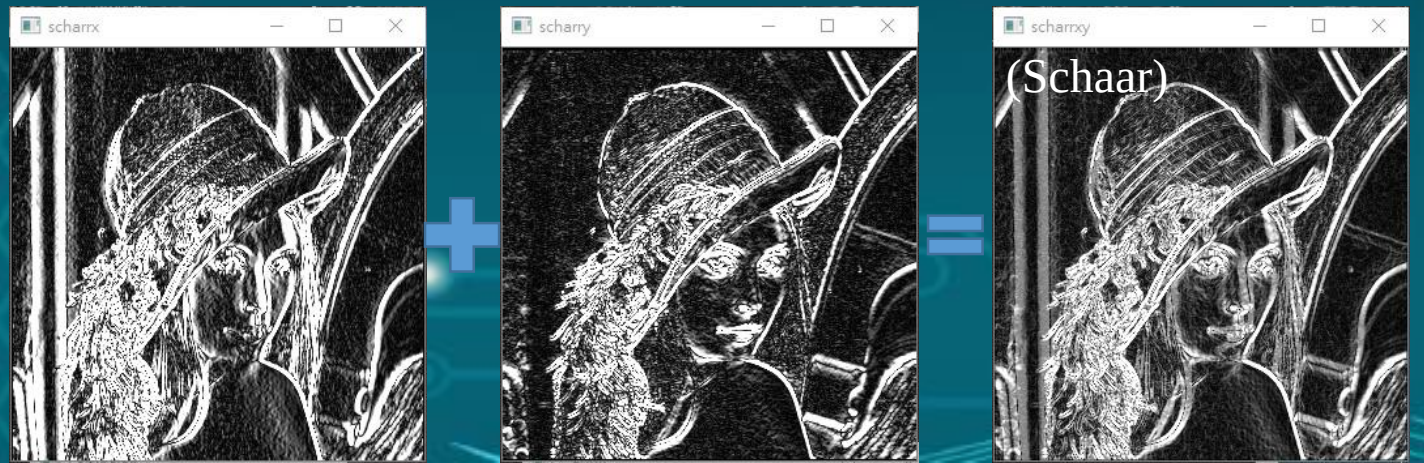
邊緣檢測
-OpenCV


absX
absY
Sobel
(Sobel)


orign


Laplacian
(Laplacian)


Canny
(Canny)

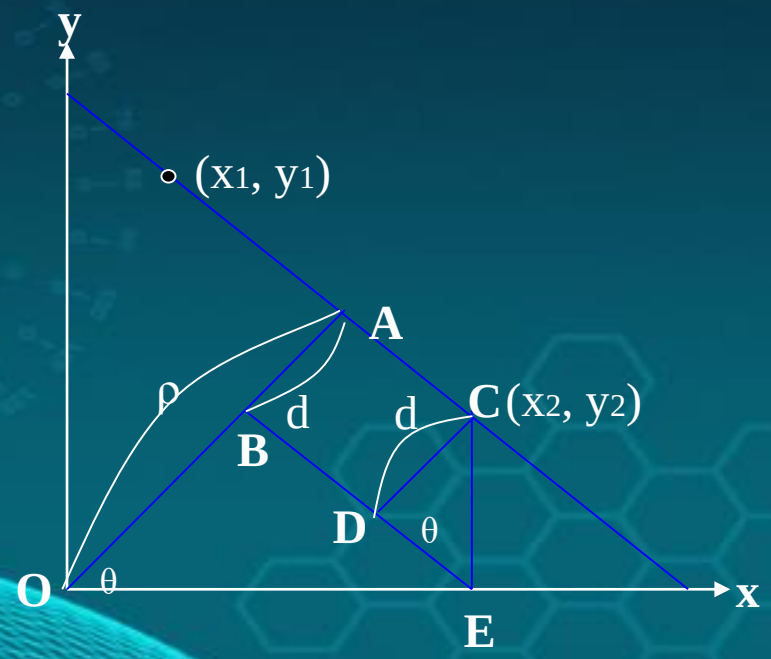

scharrx
scharry
scharrxy
(Schaar)

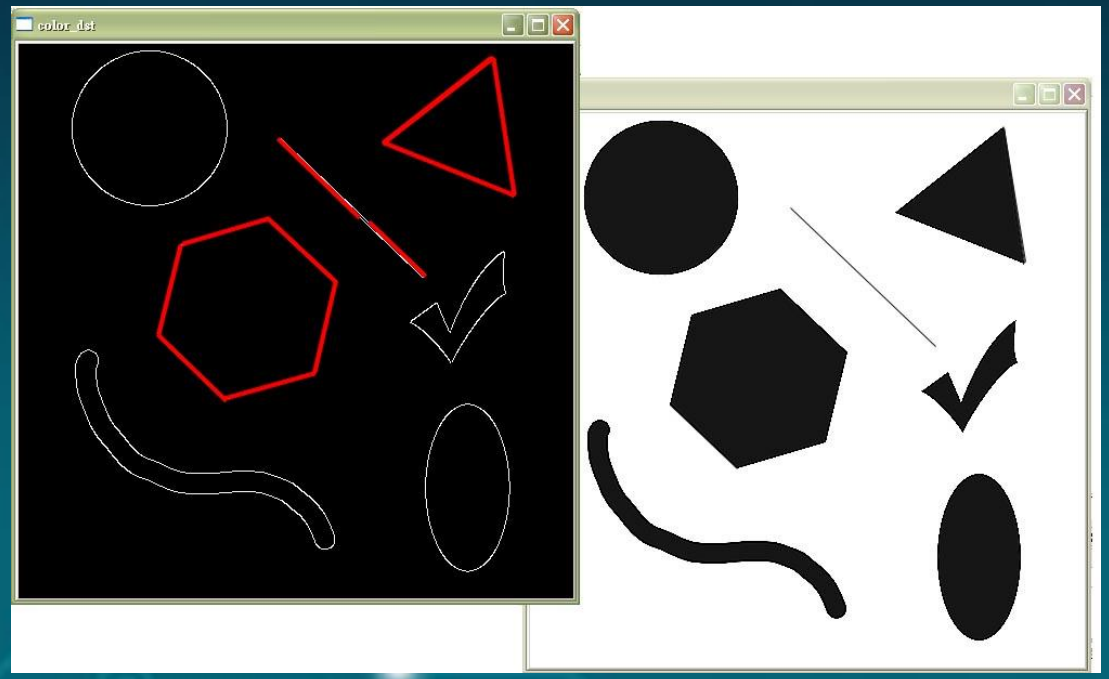
影像特徵的研究

- 「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, x_2)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△CDE可得知：

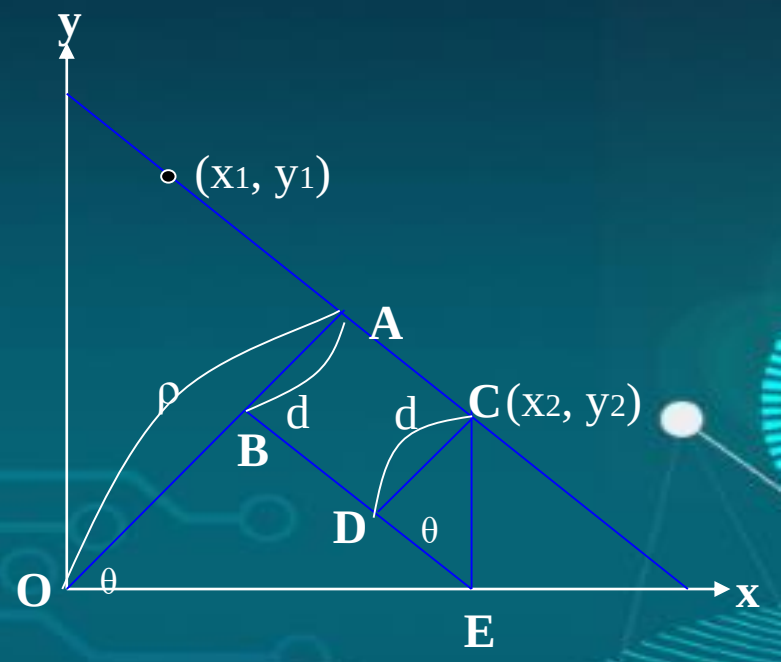
$$d = y_2 sin\theta = \overline{AB} \text{ .....(1)}$$

由直角三角形△OBE又得知：

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

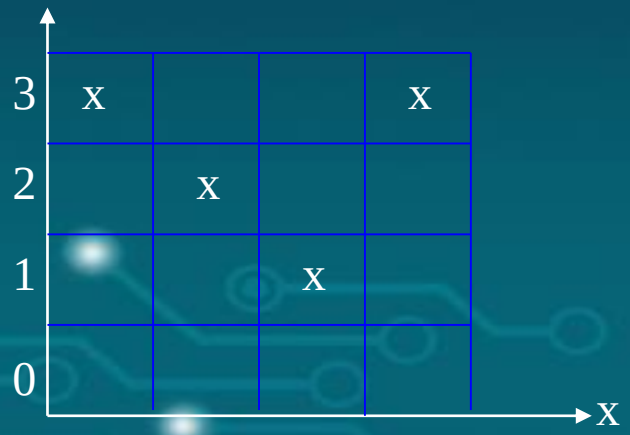
我們進而得知下列的式子：

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



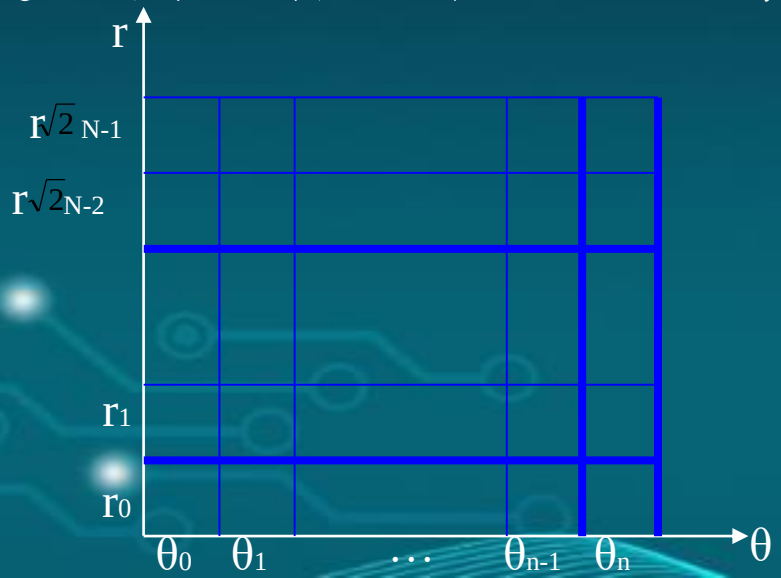
直線偵測 - Hough轉換法

- 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θ= 45度，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θ＝45度是合適的角度猜測值。基本上我們可將角度範圍[0, π]切割成n份。例如每隔5度切一份，則在[0, π]之間可切割出37份角度，這些角度分別為θ0＝0、θ_1 = 5、θ_2 = 10、…和θ_{36} =π
- 如下圖所示，先從θ0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θ1至θｎ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： 累積個數門檻值，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:

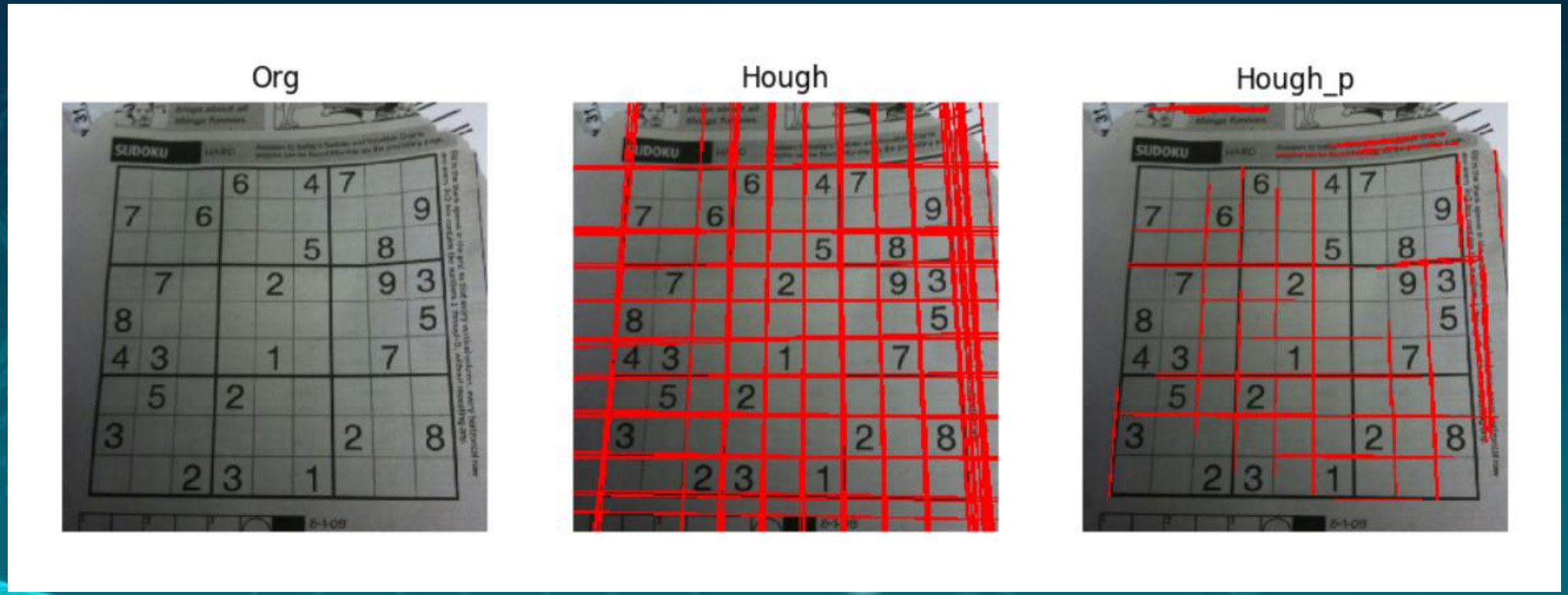
- lines = cv2.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])
  - image：輸入影像，8位元單通道二值化圖。
  - lines：將所有線的資料存在vector< Vec4i >，Vec4i為每個線段的資料，分別有x1、y1、x2、y2這四個值，(x1，y1)和(x2，y2)分別表示線段的頭尾頂點。
  - rho：距離解析度，越小表示定位要求越準確，但也較易造成應該是同條線的點判為不同線。
  - theta：角度解析度，越小表示角度要求越準確，但也較易造成應該是同條線的點判為不同線。
  - threshold：累積個數門檻值，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：最大間隔。
  - 例：lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 100, 100, 10)

直線偵測 - Hough - OpenCV

- 概率霍夫變換(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)，概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圖元點構成了一條直線，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圖元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圖元點間距。如果有超過門檻值個數的圖元點構成了一條直線，但是這組圖元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圖元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vector< Vec3f >記錄所有圓的資訊，每個Vec3f紀錄一個圓的資訊，包含3個浮點數資料，分別表示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：計數門檻值，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