嵌入式智慧影像分析與實境界面
Fall 2021

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

Lecture 6

OpenCV車道辨識

Image Processing Edge Detection

邊緣檢測-簡介

- 影像邊緣檢測，去除了大部分認為與影像不相關的資訊，保留了影像重要的紋理特徵。
- 邊緣點通常是影像中亮度變化明顯的點。影像紋理中的顯著變化通常反映了紋理的重要事件和變化。有以下幾種情況 ：
  - 深度上的不連續。
  - 表面方向不連續。
  - 物質紋理變化。
  - 場景照明變化。

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矩陣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。

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

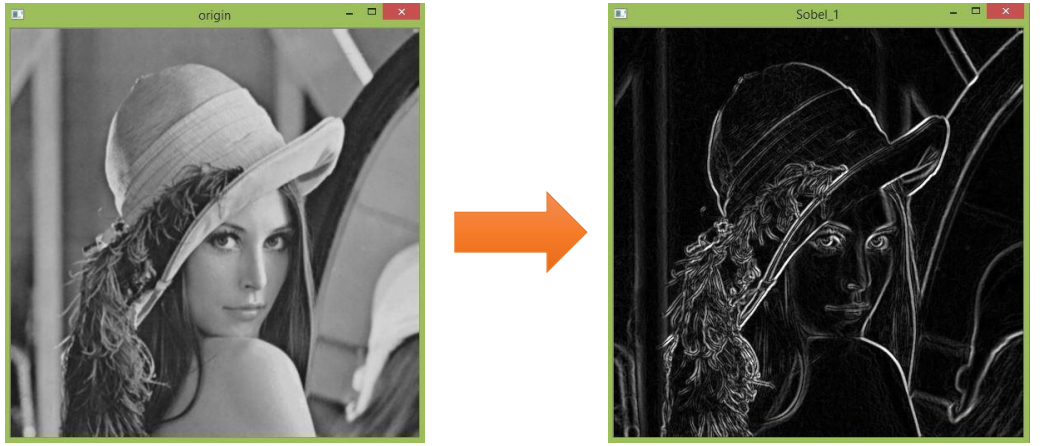
- 影像中每一個像素的橫向及縱向梯度近似值，可用以下公式計算梯度的大小。

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

邊緣檢測 – Sobel – OpenCV(C++)

OpenCV Sobel()函式

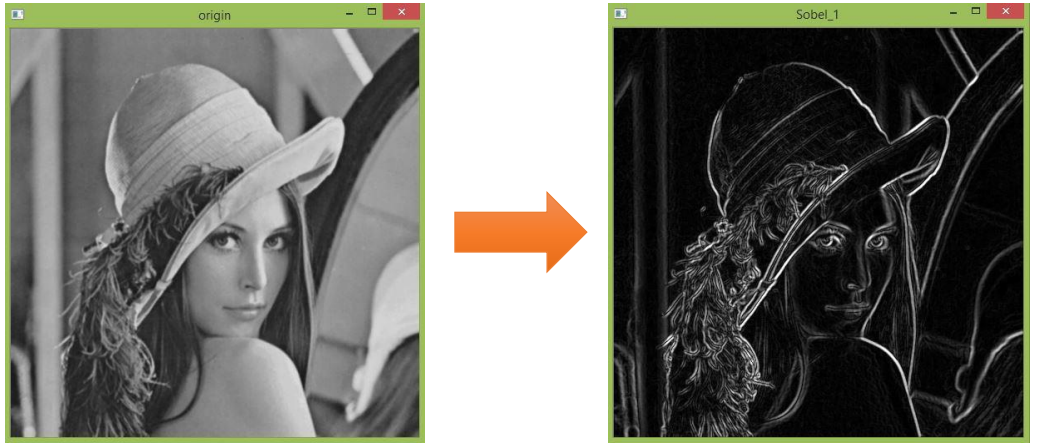
- void Sobel(const Mat &src, Mat dst, int ddepth, int dx, int dy, int ksize = 3, double scale = 1, double delta = 0, int borderType = BORDER_DEFAULT)
  - src：輸入影像。
  - dst：輸出影像，和輸入影像有相同的尺寸和通道數。
  - ddepth：輸出影像的深度，假設輸入影像為CV_8U, 則支援CV_8U、CV_16S、CV_32F、CV_64F，假設輸入影像為 CV_16U, 則支援CV_16U、CV_32F、CV_64F。
  - 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。



邊緣檢測 – Sobel – OpenCV(Python)

OpenCV Sobel()函式

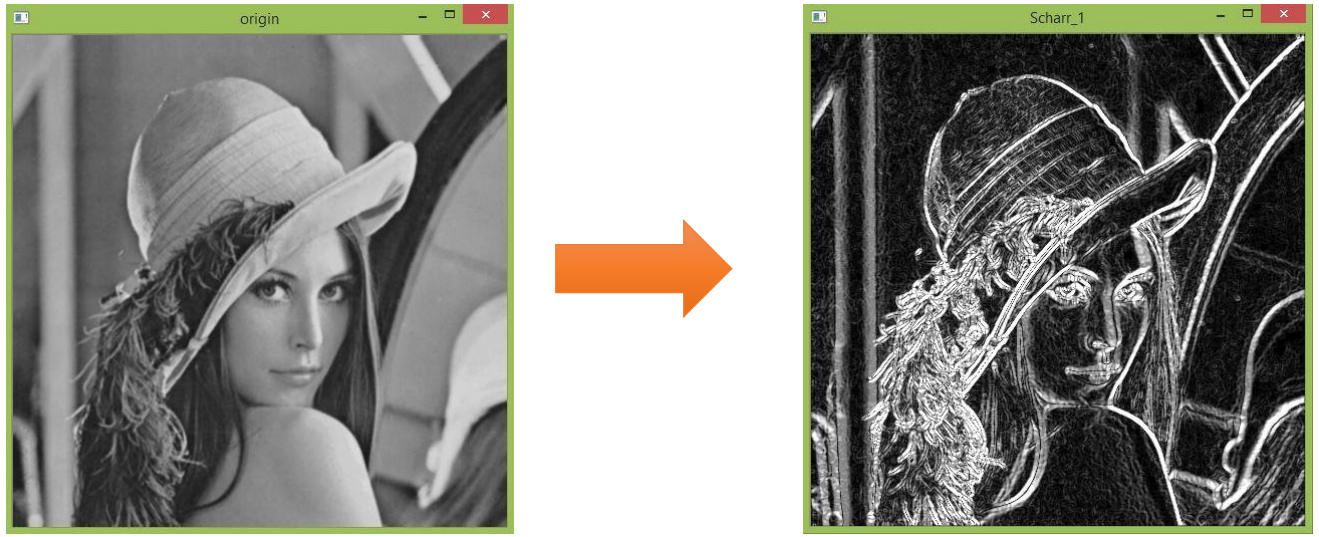
- dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])
  - src：輸入影像。
  - dst：輸出影像，和輸入影像有相同的尺寸和通道數。
  - ddepth：輸出影像的深度，假設輸入影像為CV_8U, 則支援CV_8U、CV_16S、CV_32F、CV_64F，假設輸入影像為 CV_16U, 則支援CV_16U、CV_32F、CV_64F。
  - dx：x方向的微分階數。
  - dy：y方向的微分階數。
  - ksize：核心大小，必須為1、3、5或7。
  - scale：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



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矩陣，但Scharr只有3x3的矩陣。

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



邊緣檢測 – Scharr – OpenCV(C++)

OpenCV Scharr()函式

- void Scharr(InputArray src, OutputArray dst, int ddepth, int dx, int dy, double scale=1, double delta=0, int borderType=BORDER_DEFAULT)
  - 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。

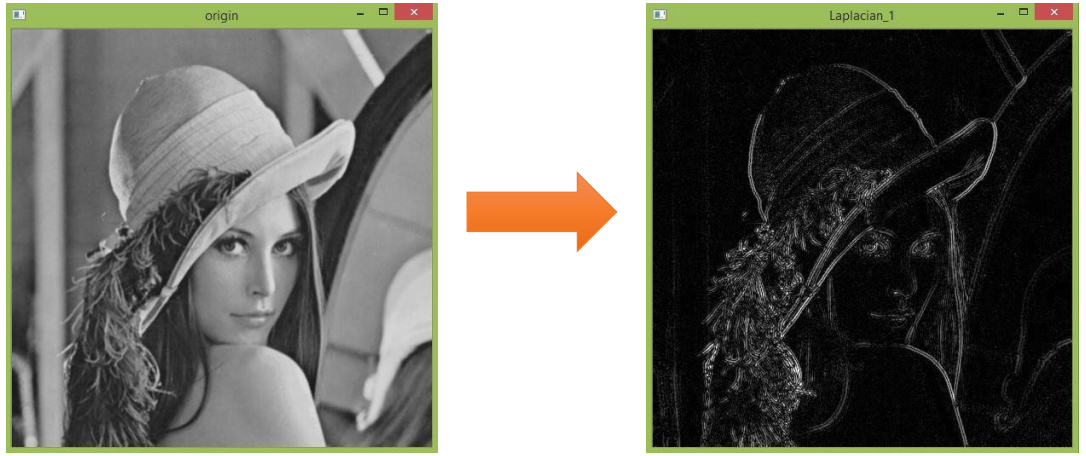
邊緣檢測 – Scharr – OpenCV(Python)

OpenCV Scharr()函式

- dst = cv2.Scharr(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。

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和相加，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（以下為常用矩陣）：

0	1	0
1	-4	1
0	1	0

邊緣檢測 – Laplacian算法 – OpenCV(C++)

OpenCV Laplacian()函式

- void Laplacian(const Mat &src, Mat dst, int ddepth, int ksize = 1, double scale = 1, double delta = 0, intborderType = BORDER_DEFAULT)
  - 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, 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。
  - ksize：核心大小，輸入值必須為正整數。
  - borderType: 判斷影像邊界的模式。

邊緣檢測 – Laplacian算法 – OpenCV(Python)

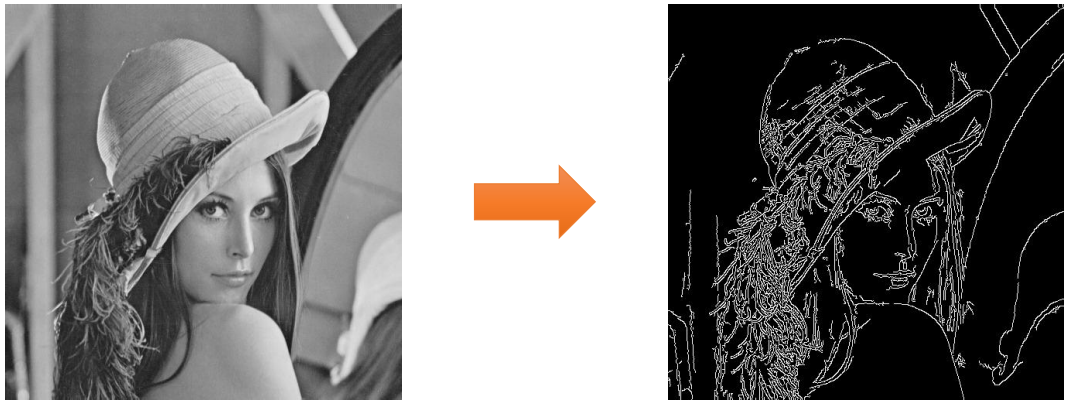
OpenCV Laplacian()函式

- dst = cv2.Laplacian(src, ddepth[, 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, 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。
  - ksize：核心大小，輸入值必須為正整數。
  - borderType: 判斷影像邊界的模式。

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。

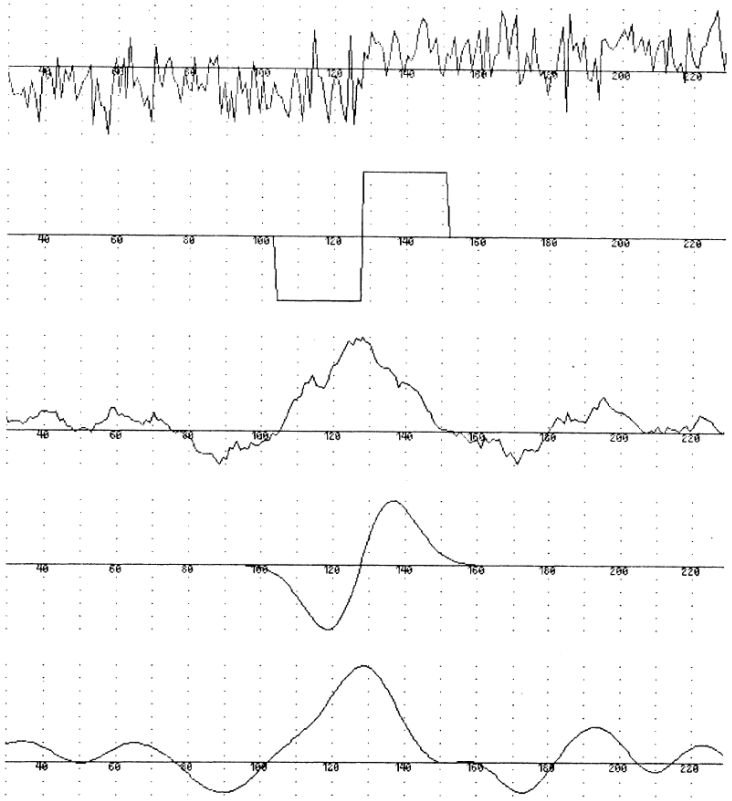


邊緣檢測 - Canny

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。
- 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)

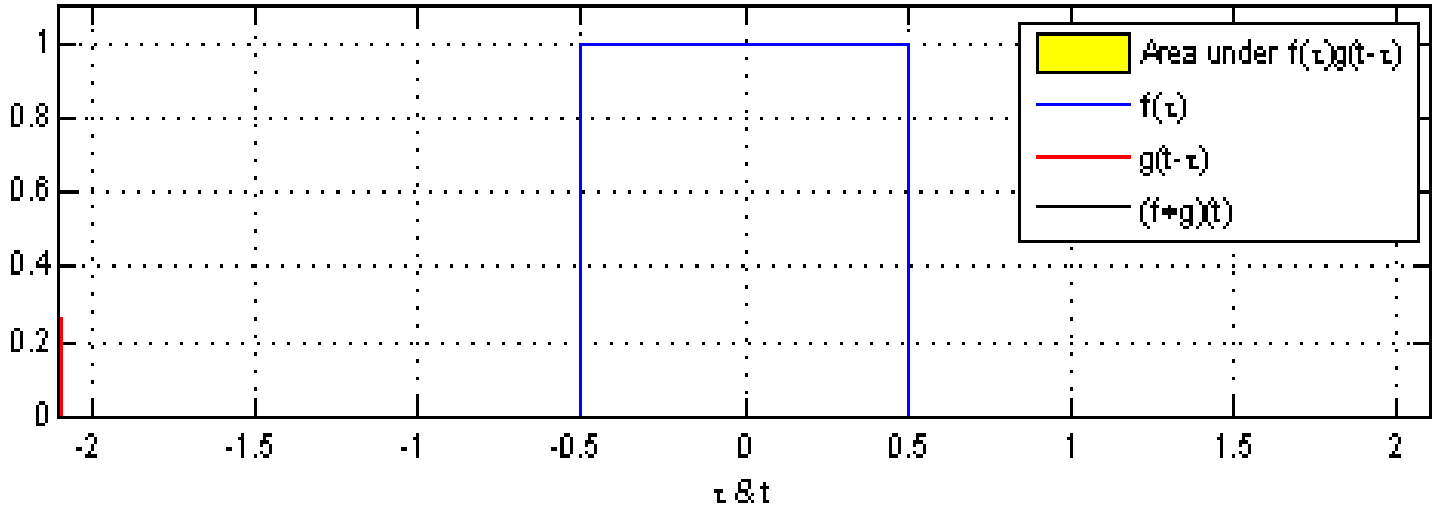
(c) Convolution (a)(b)

(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。



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

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

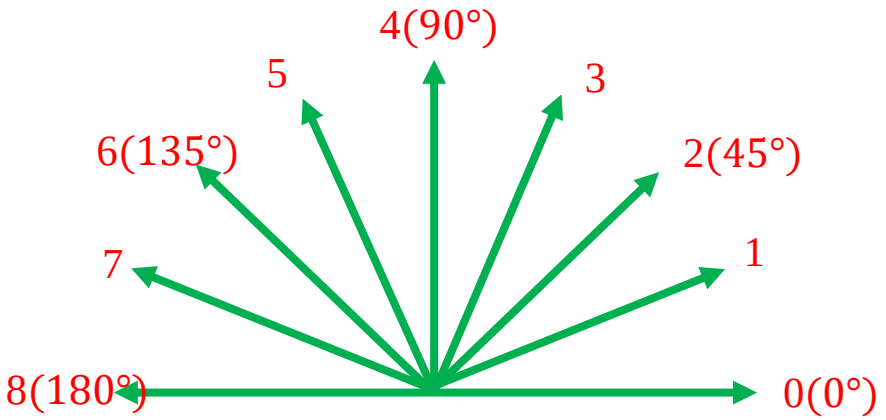
$$\theta(v,w) = \tan^{-1}\left(\frac{\Delta Y_1(v,w)}{\Delta X_1(v,w)}\right)$$

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

$$\begin{cases} d1 = dir1_i, d2 = dir2_i \ \ if\ LBi < PN \le HBi, i = 0 \cdots 2 \\ d1 = dir13, d2 = dir2\grave{3}\ , else\ if\ LB_3 < PN\ or\ PN \le HB_3 \end{cases}$$

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

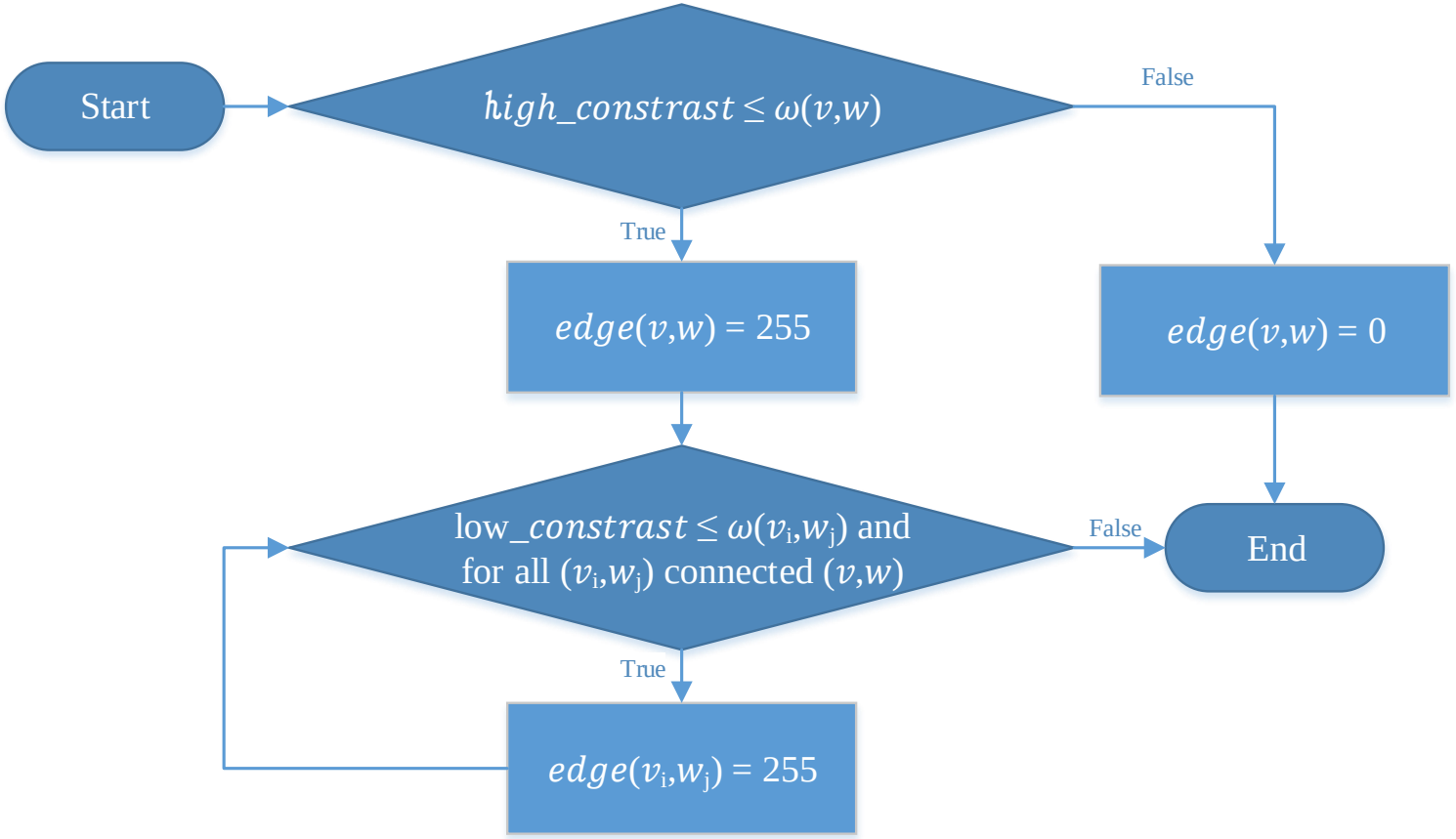
$$LB \leftarrow \{1,3,5,7\}, HB \leftarrow \{3,5,7,1\}$$



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

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(C++)

OpenCV Canny()函式:

- void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false)
  - src：輸入影像，單通道8位元圖。
  - dst：輸出圖，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
  - apertureSize ：Sobel算子的核心大小。
  - L2gradient ：梯度大小的算法，預設為false。

邊緣檢測 – Canny – OpenCV(Python)

OpenCV Canny()函式:

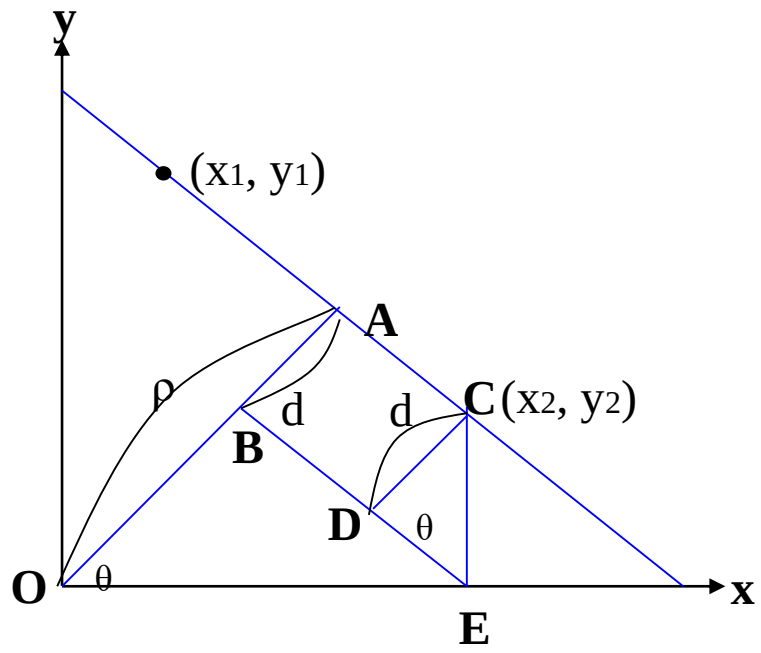
- dst = cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])
  - src：輸入影像，單通道8位元圖。
  - dst：輸出圖，尺寸、型態和輸入影像相同。
  - threshold1：第一個門檻值
  - threshold2：第二個門檻值
  - apertureSize ：Sobel算子的核心大小。
  - L2gradient ：梯度大小的算法，預設為false。

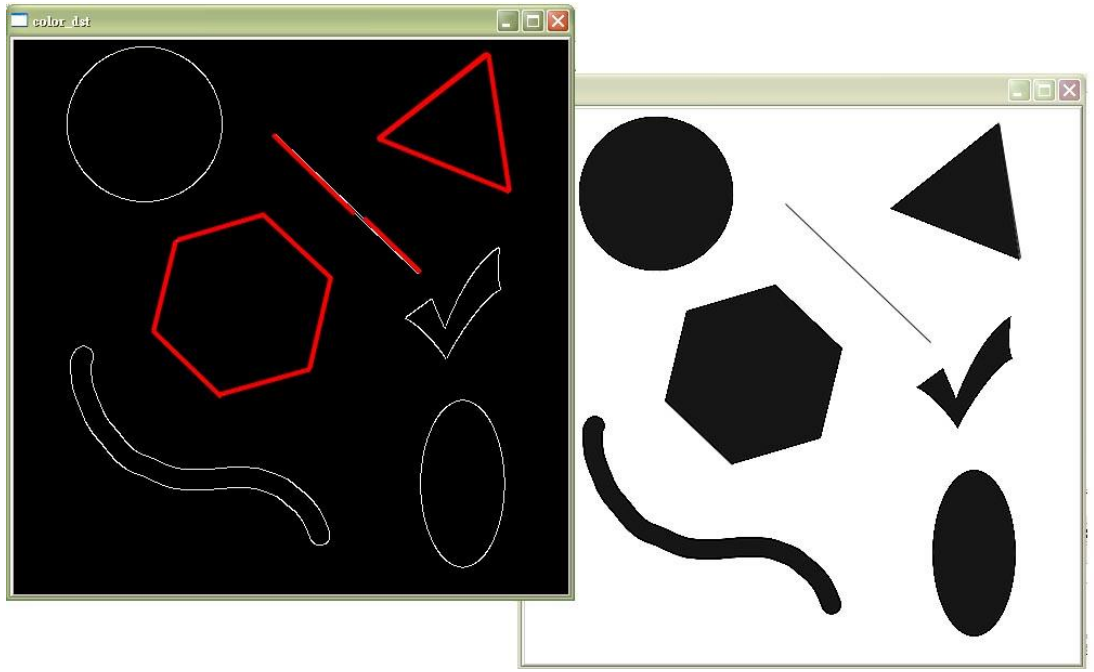
影像特徵的研究

- 「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：

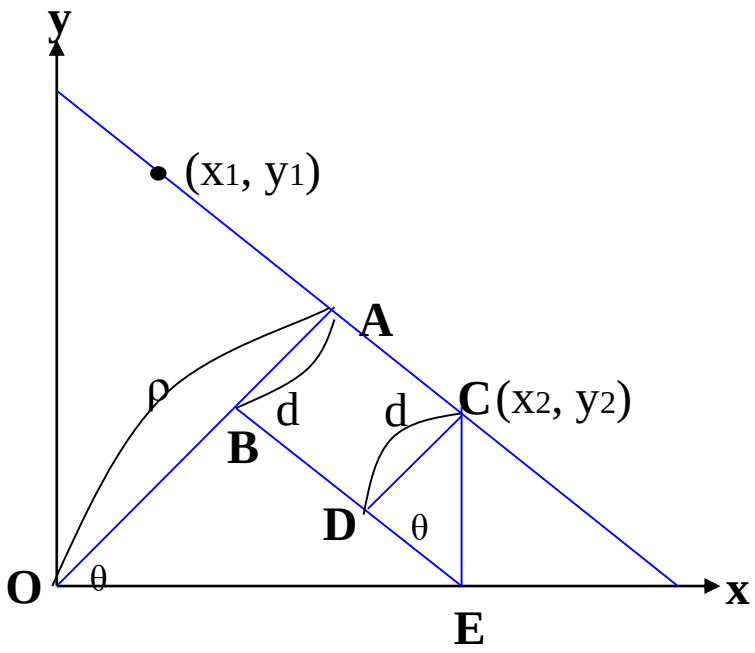
$$d = y_2 sin\theta = \overline{AB} \text{ .....(1)}$$

由直角三角形△OBE又得知：

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

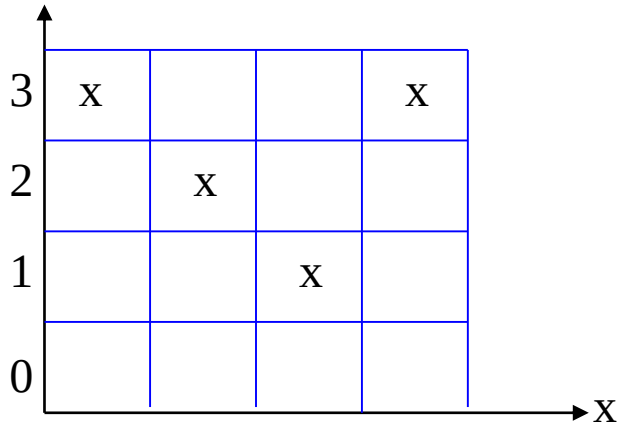
我們進而得知下列的式子：

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



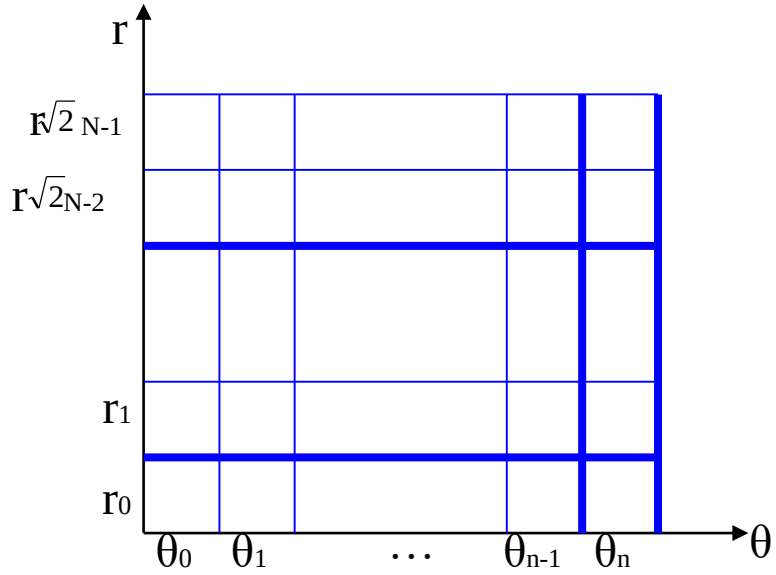
直線偵測 - Hough轉換法

- 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，將四點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θ_0 = 0、θ_1 = 5、θ_2 = 10、…和θ_{36} =π
- 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(C++)

OpenCV HoughLinesP()函式:

- void HoughLinesP(InputArray image, OutputArray lines, double rho, double theta, int threshold, double minLineLength=0, double maxLineGap=0)
  - 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。

直線偵測 - Hough – OpenCV(Python)

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。
  - 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。

圓形偵測 - Hough – OpenCV(C++)

OpenCV HoughCircles()函式：

- void HoughCircles(InputArray image, OutputArray circles, int method, double dp, double minDist, double param1=100, doubleparam2=100, int minRadius=0, int maxRadius=0)
  - 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。

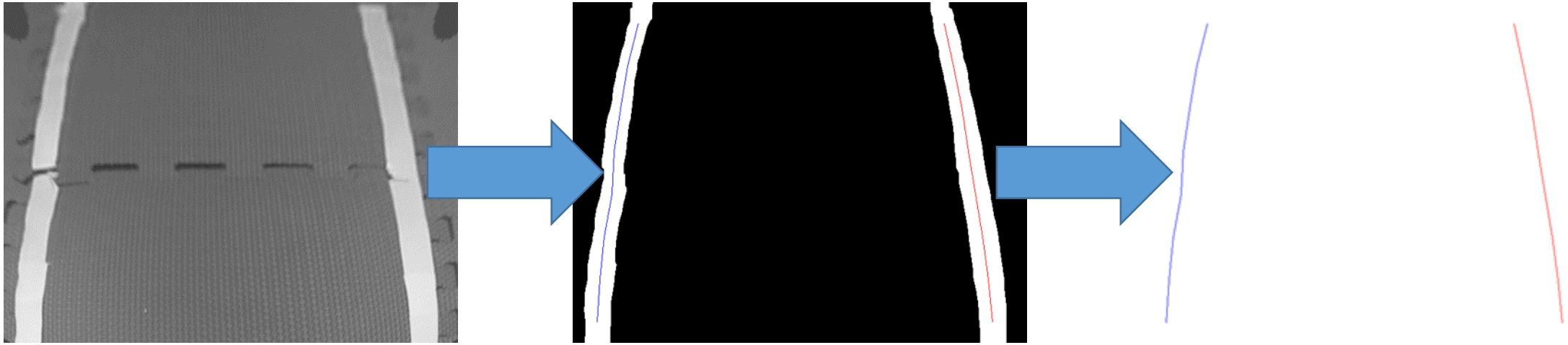
圓形偵測 - Hough – OpenCV(Python)

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。
  - 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。

中線擷取

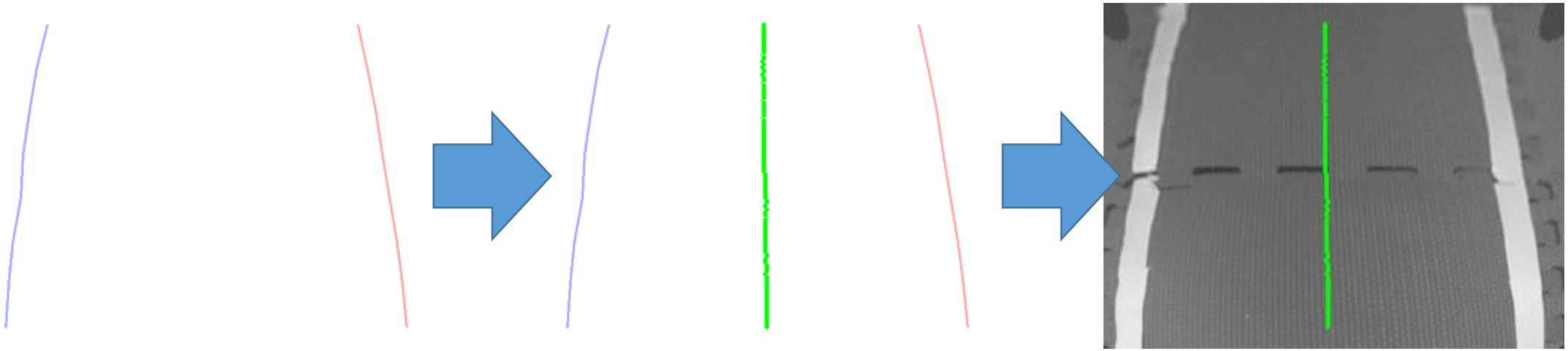
中線擷取



利用影像前處理擷取出車子左右兩條車道線，而為了辨識出車子於車道中的相對位置，我們可以先利用左右兩條線擷取出中間的線條。

中線擷取－方法一

要擷取出中間的線，我們可以直接利用圖片的特性，用一個for迴圈抓出對於每個y值，左邊線條的x值(x_l)與右邊線條的x值(x_r)，取兩者平均後畫出中間點$(\frac{x_l+x_r}{2}, y)$，即可畫出中間的線條。



OpenCV line(C++)

將左線頭點與右線頭點取平均作為中線頭點，並將左線尾點與右線尾點取平均作為中線尾點，再使用cv::line()在圖形上畫出直線。

OpenCV line()函式(C++)：

- void cv::line(InputOutputArray img, Point pt1, Point pt2, const Scalar &color, int thickness)
  - img: 輸入影像
  - pt1: 直線端點1，格式為(int x, int y)
  - pt2: 直線端點2，格式為(int x, int y)
  - color: 直線顏色，格式為[int B, int G, int R]
  - thickness: 直線寬度，預設為1 pixel

OpenCV line(Python)

將左線頭點與右線頭點取平均作為中線頭點，並將左線尾點與右線尾點取平均作為中線尾點，再使用cv::line()在圖形上畫出直線。

OpenCV line()函式(Python)：

- cv2.line(img, pt1, pt2, color[, thickness[, lineType[, shift]]])
  - img: 輸入影像
  - pt1: 直線端點1，格式為(int x, int y)
  - pt2: 直線端點2，格式為(int x, int y)
  - color: 直線顏色，格式為[int B, int G, int R]
  - thickness: 直線寬度，預設為1 pixel