教育部教學實踐研究計畫成果報告
Project Report for MOE Teaching Practice Research Program

計畫編號/Project Number：PMS1080040
學門分類/Division：數理
執行期間/Funding Period：2019-08-01-2020-07-31

計畫名稱：
數學影像處理的相關理論及實作/
Mathematical Image Processing and Implementations

配合課程名稱：
相關數學影像處理的相關理論及實作專題 I (MA5103) 及
數學影像處理的相關理論及實作專題 II (MA5104)

計畫主持人(Principal Investigator)：陳建隆

執行機構及系所(Institution/Department/Program)：國立中央大學數學系

繳交報告日期(Report Submission Date)：109-08-01

目錄

1. 報告內文(Content)

(1) 研究動機與目的(Research Motive and Purpose)

中央數學系並沒有畢業專題的課程，在系上必選修學分上規定也十分寬鬆，這樣寬鬆的規定希望數學系學生可以跨系修課、培養跨領域能力；但實際上，跨系所修課有著不少的困難點，一是衝堂、二是課程的先後關係、三是不知道該往哪個方向發展。

對於一、二點問題來說，通常希望跨系修課的數學系學生會在大二開始選修其他系的課程，但在大二時仍有一部份的必修要修，會造成沒辦法選修其他系大一的課程，造成許多數學系學生在大三結束時才剛完成它系的基礎課程，而在大四時面對考研究所壓力時，才剛開始修習它系的進階課程，這對希望跨領域發展同時也想念研究所的學生造成極大的阻礙。

而第三點問題則是最嚴重的，學生不知道自己該去修習哪方面的課程，故數學系自身應該開設一些跨領域的課程，提供給數學系學生一些不同領域的知識，開拓學生的視野，提供理論數學以外的知識，才不會讓有潛力的學生被制度所侷限住。

有鑑於影像處理領域在近年來高速發展，產業界急需大量擁有影像處理專長的人才，同時影像處理演算法所需的數學程度對數學系學生來說也恰好適中，故本計畫預計以影像處理方面的課程作為使力點。一方面可以將低學生跨領域學習的門檻，另一方面也可以消弭產學落差。

所以本計畫預計針對數學系大三、大四的學生，設立一門一學年的影像課程，將原本資電學院的影像處理課程，帶到數學系來，並應用數學系課程(線性代數、微分方程、機率統計、程式設計、離散數學等)的知識，以建立模型的觀點來理解影像處理，提供一門專為數學系學生量身打造的跨領域課程，讓學生接觸不同領域的知識，並培養學生發掘問題、解決問題的能力。

同時，與課程並進，預計會編寫出一份影像處理教材(以網頁形式呈現)，彙整基礎的影像處理技巧與學生實作的主題，提供對影像處理方面有興趣的人學習，也能使此課程成為中央數學系的一門特色課程。

計畫的主題是希望提供一門跨領域課程給數學系學生，並建立一份影像處理的學習文件，帶領學生亦或是對此方面有興趣的人入門影像處理。

研究目的是希望除了教授數學理論給學生，而且能夠讓學生實際應用所學過的數學理論，同時培養學生的跨領域技能、知識，讓學生可以以更寬廣的視

野來發掘、解決問題。

未來目標則著重於建立一套完整的影像處理課程，培養學生發展影像演算法、建立模型的能力，以便在未來可以投入產業界，提升整體產業實力；抑或是投入學術界，研究更尖端影像處理技巧、理論。

(2) 文獻探討(Literature Review)

在網路上有些影像處理教學網頁，但絕大多數的內容都較缺少完備的數學理論介紹，而有介紹數學理論的網頁常常又會缺乏實作程式碼，再者是這類型的資源通常是以英文編寫，對於想要涉足這塊領域的中文讀者也是一道不小的阻礙。

在網路上有一份影像的教學網頁《演算法筆記》，裡面涵蓋了影像處理技巧的基本介紹，可以帶領讀者快速入門影像處理領域，但是文章內大多是介紹這個方法的效果，對於理論及其實作沒有深入介紹，故未來要編寫的影像教學網頁除了介紹影像處理技巧外，同時也會著重在理論與實作的介紹。

另外，因為此份影像處理教材《演算法筆記》並不是設計面向課程，在未來編寫的影像教學網頁會以主題式的方式編寫，將主題切分成一個一個小單元，配合教學使用。

而在 An Introduction to Mathematical Image Processing 這篇則是以數學方面切入影像處理的範本，將影像處理領域的問題用數學模型來思考，並以數學理論、技巧來解釋。未來編寫的教學網站理論方面預期上是以這種方式呈現。

透過結合數學理論與影像實作，培養數學系的學生以一個新的角度切入影像處理領域，培養出不同於一般資電學院影像課程的技能，讓學生在各種影像處理問題上能有更寬廣的視野，在解決問題上能有自己獨特的見解。

(3) 研究問題

教導學生影像處理技巧，透過分組專題實作經驗，培養發掘問題、解決問題的能力，同時也將補足學生影像處理方面的知識，為以後的理論研究工作或是就業能力奠下基礎。

(4) 研究設計與方法

針對計畫欲規劃的課程，召集數位學生，訓練影像處理技巧及實作專題，詳列如下。

(4-1) 邊緣偵測在車牌辨識中的應用

此主題冀求學生在實作成功以後，讓學生實際演練車牌辨識系統應用於智慧停車場之出入口車輛識別，以作為智慧化停車場部屬練習。

車牌辨識系統主要有兩個問題：一為車牌之位置偵測、二為車牌上之文字辨識。針對前者問題，由於車牌位置相對於攝影機 (停車場出入口) 是固定的，因此在處理上先假設車牌的大致位置，再透過邊界偵測、形態學中的影像運算等處理，抓出照片中的物件形狀，最後以車牌之固定形狀在假設位置附近尋找最適配的物件，擷取出圖像中的車牌。

而後者問題屬於光學文字識別的範疇。由於台灣的車牌內字元落在英文字母與數字兩類，因此在辨識上不考慮兩類以外的字元，另外針對易混淆字元也會被排除在外 (如容易與數字 “0” 混淆的字母 “O”)；在辨識上，主流而言多使用透過機器學習所訓練出之網路作為辨識依據，我們可以藉由後處理中過濾掉排除在外的字元，以提升車牌辨識的成功率。

學生可以透過此實作的過程中，理解影像、以及影像處理中的一些基本運算有更多的認識。

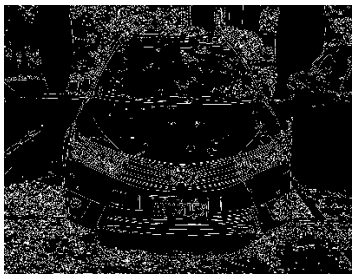
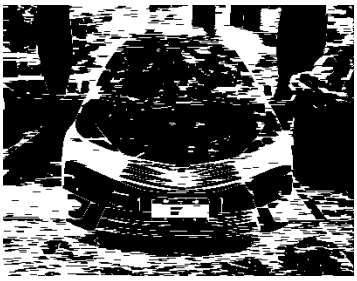
<table>
<tr><td>  </td><td>  </td><td>  </td></tr>
<tr><td>(a) 原圖</td><td>(b) 邊界偵測</td><td>(c) 型態運算 (close)</td></tr>
<tr><td colspan="2">  </td><td>ANC-2059</td></tr>
<tr><td colspan="2">(d) 於假定位置附近中找出適配物件</td><td>(e) 進行文字辨識與後處理</td></tr>
</table>

圖 車牌辨識流程 (a)-(e)

(4-2) 浮水印

在現代的通訊系統中，有許多資料傳輸都是以加密形式進行，這些加密的目的都是：即使攻擊者知道東西被加密，也無法破解，這些都是屬於資訊加密

的範疇。

就像是「把東西鎖在箱子」裡，但是也有一些需求是「把東西給別人看，但別人拿不走」，以及「檢查東西的完整性」。在影像浮水印的章節裡，這些東西就是影像；添加浮水印，就能完成這兩個需求。

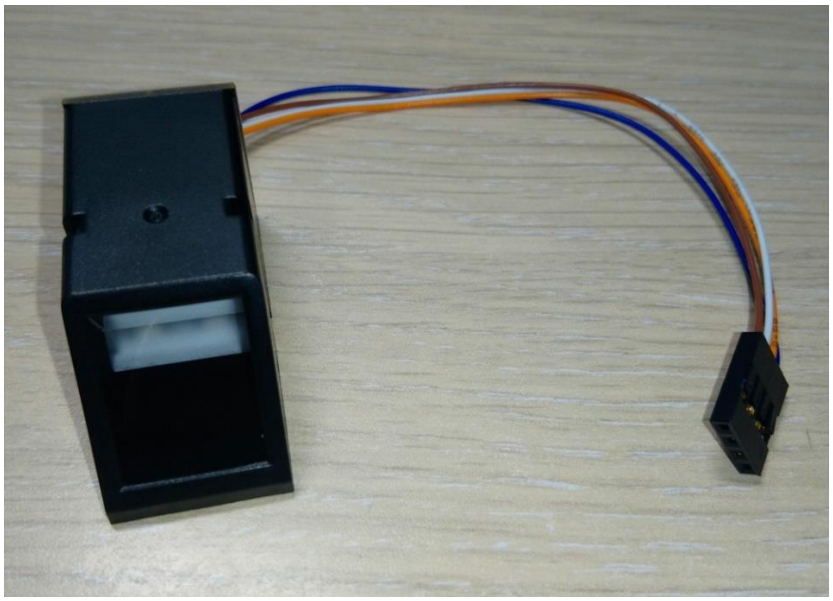
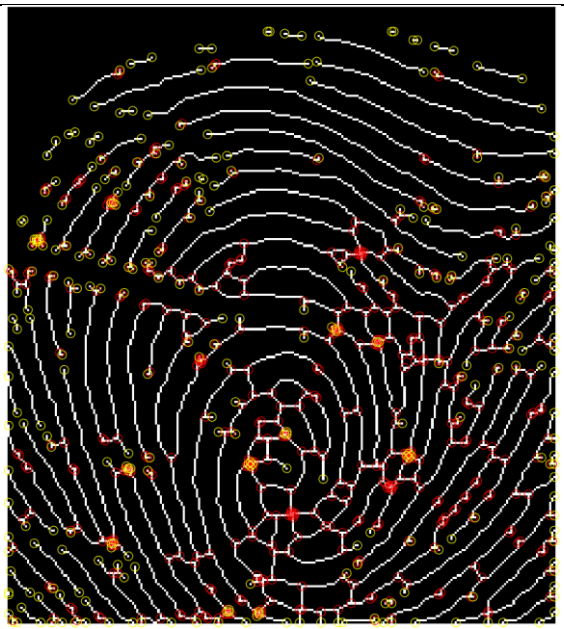
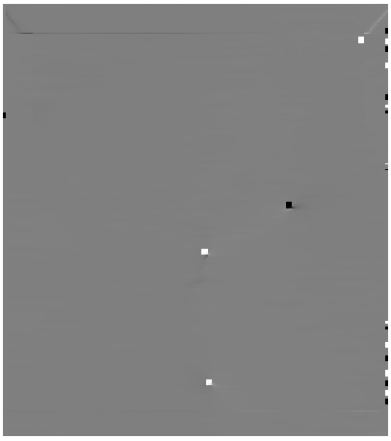
對於「把東西給別人看，但別人拿不走」，就加入一個難以移除 (強健型) 的浮水印。 對於「檢查東西的完整性」，就加入一個容易被破壞 (易碎型) 的浮水印

學生可以透過實作浮水印的過程理解影像資料的組成，理解一些簡單的浮水印演算法。

	破壞後的影像	取出的浮水印
LSB 浮水印		
VQ 浮水印		

(4-3) 指紋辨識

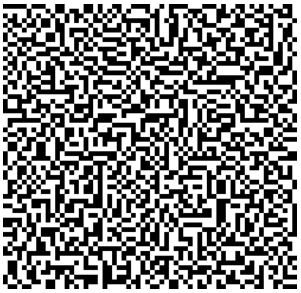
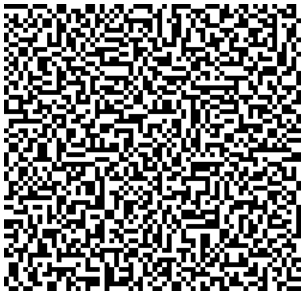
指紋辨識在影像處理領域發展甚久，已經有相當成熟的技術來處理這個問題，在這個課題中，教導學生實作影像強化、特徵擷取等技巧來實作一個指紋辨識系統。

硬體設備	
指紋特徵抓取	
指紋奇異點抓取 左：原圖 右：奇異點	 

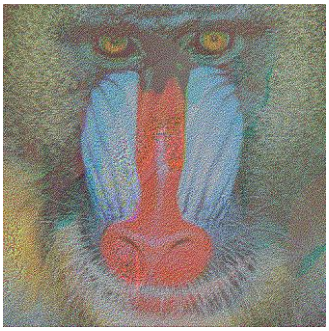
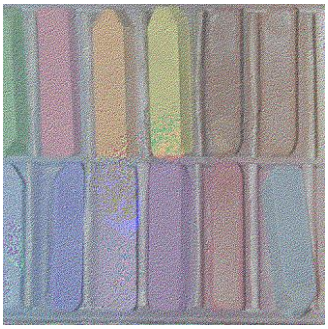
(4-4) 視覺密碼

傳統密碼學是加密者透過金鑰，再使用特定演算法來將資訊加密；而解密者也需要一把金鑰，並使用特定演算法來解密；而視覺密碼的加密則是生成許多張圖片，透過"疊合圖片"來解密。

黑白視覺密碼

疊合圖片 1	疊合圖片 2	疊合結果
		

彩色視覺密碼

疊合圖片 1	疊合圖片 2	疊合結果
		

(4-5) 立體網頁

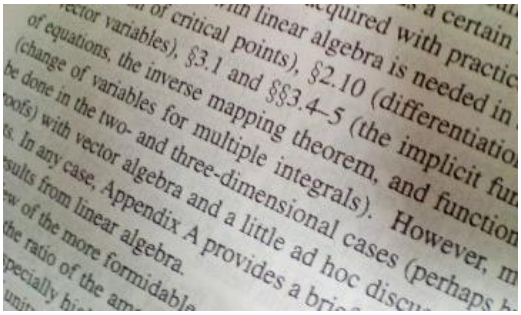
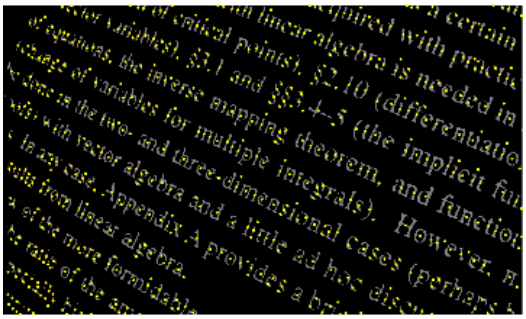
在立體網頁的章節中，我們使用 html,css 等網頁語言在網頁上實作一個立體投影影像，以此向學生講解立體投影相關內容。

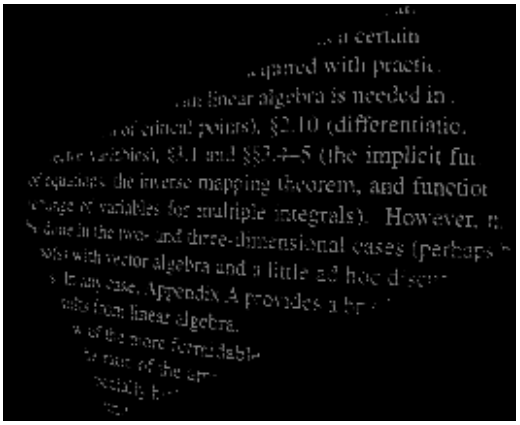
(4-6) 文章彎曲字體校正與辨識

藉由相機直接拍攝文件，進行文件中文字辨識，並轉譯為音訊輸出，是一種可以做為有閱讀困難的讀者藉由聽覺來理解文件的方案，這同時也是實作該主題背後的重要意涵。

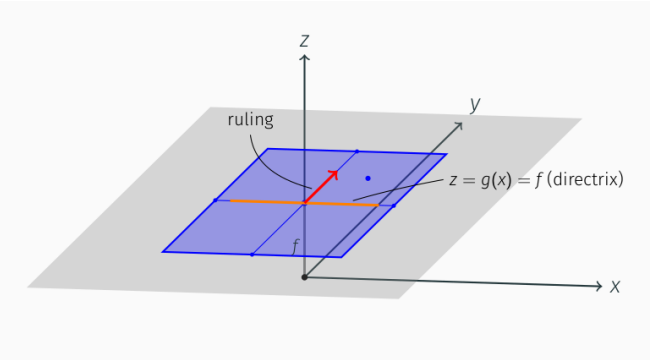
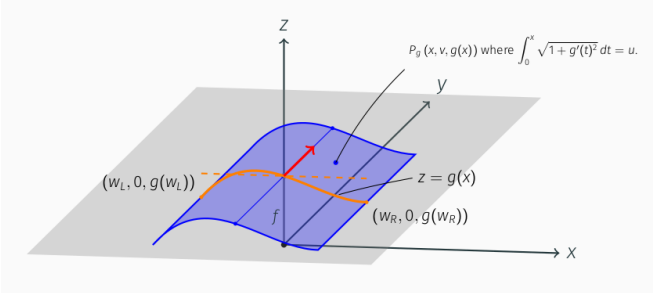
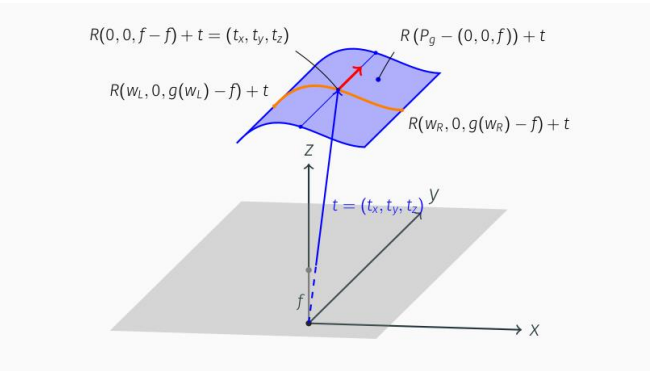
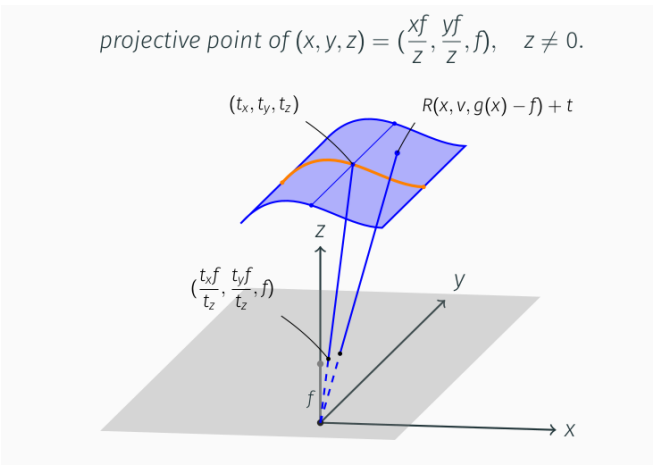
然而透過相機直接拍攝紙本文章時，容易有文件在影像中扭曲的現象，其中這個扭曲可能是攝影角度的影響，也可能是紙本自身因裝訂等因素所導致的變形。當這個現象產生時，最直接地會導致文字辨識效果降低。因此達成彎曲文章中的高精度文字辨識，首先要處理的議題即為彎曲文章校正。

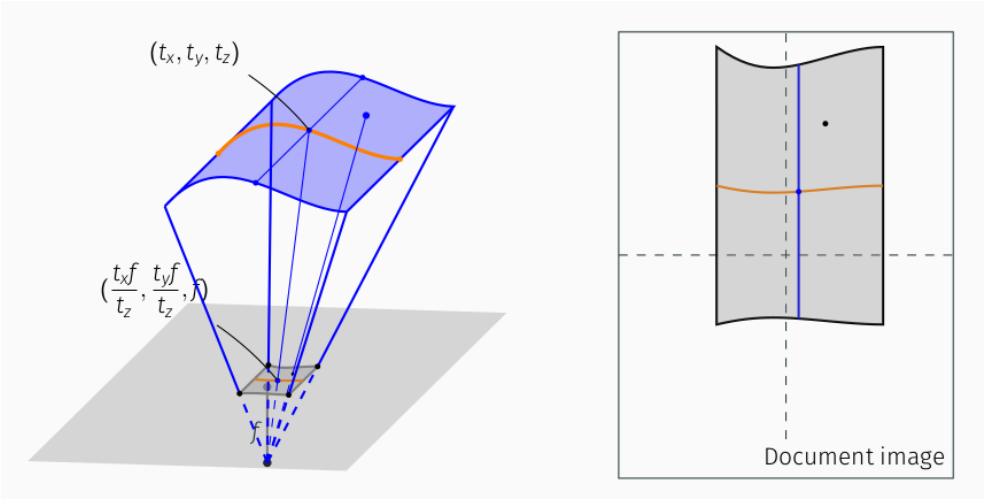
若討論扭曲的所有變因，前者問題是相對單純的──大抵上我們可以透過某種仿射變換以將文件校正。在以下實驗中，我們首先透過區域二值化找出文字塊，藉由一次回歸線運算找出扭曲角度，並將整張影像轉正。

(a) 原圖	(b) 區域二值化運算
	
(c) 找出字元質心	(d) 計算一次回歸線找出扭曲角度
	
(e) 將整張影像轉正	

	圖 仿射校正 (a)-(e)

然而當文件不單只受到前者變因影響時，此時就會衍生成一個複雜的問題。我們首先將問題以模型的形式進行描述：

	
(a) 文件在投影幕上 (藍色：文件、灰色：投影幕)	(b) 在 xz 平面上做扭曲變換
	
(c) 對文件進行平移變換	(d) 計算文件上投射至投影幕的對應點



(e) 投影結果

此時我們便可基於上述模型對此問題進行探討與推算。從方法上大致可分成以下兩類：

(1) 透過設計能量函數以衍生最佳化問題，

[Lai Kang, etc. (2016). Rectification of curved document images based on single view three-dimensional reconstruction]

或是

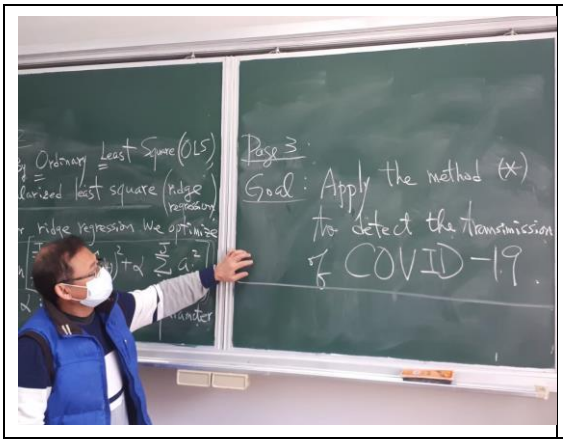
(2) 以幾何的觀點出發，找出變換當中的不變量並還原扭曲文件。

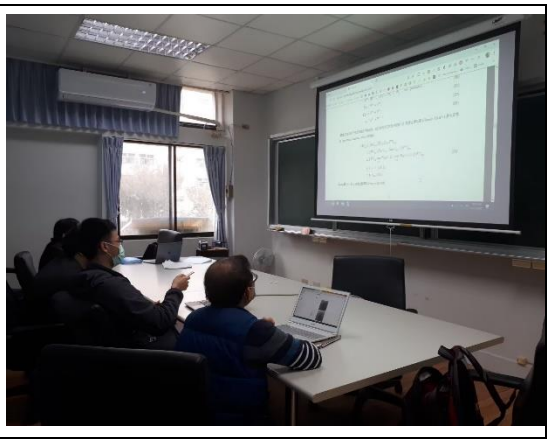
[Gaofeng Meng, Chunhong Pan, etc. (2012). Metric Rectification of Curved Document Images]

在課程中，期許學生從方法探索中找到解決方法的可能性，並且透過理解與實作來驗證解決方法的可行性。

(5) 教學暨研究成果(Teaching and Research Outcomes)

(5-1) 課堂攝影





(5-2) 上課簡報：詳見附件

(5-3) 網站內容：詳見附件

(5-4) 學生學習回饋

教學評量表格

數學影像處理的相關理論及實作專題 I

基本資料					
1.這門課我的缺席次數	從不缺課	缺課 1-3 週	缺課 4-6 週	缺課 7-12 週	缺課 13 週以上
	6	2	0	0	0
2.除上課時間外，平均一星期修讀本課程付出的時間	10 小時以上	6-9 小時	4-5 小時	1-3 小時	1 小時以下
	1	0	2	3	2
對本科目之教學意見	非常同意	同意	普通	不同意	非常不同意
3.我有修習本課程	5	2	1	0	0
4.我對這門課的學習態度很認真	2	3	3	0	0
5.教師提供完整的課程大綱，且教學內容與課程大綱相符	4	3	0	0	0
6.教師的課前準備充分，上課內容豐富充實	3	4	1	0	0

7.我在課程中感受到教師的教學熱忱	4	4	0	0	0
8.本課程授課內容組織完善，有助學習	2	4	1	0	0
9.教材設計能顧及學生的學習狀況	2	3	3	0	0
10.教師具備講授本課程之專業知識	4	4	0	0	0
11.教師講解表達方式良好，使課程容易瞭解	2	4	1	0	0
12.教師樂於協助學生解決有關本課程之疑問	4	4	0	0	0
13.教師(或助教)對試卷、作業或報告的評分公平合理	4	3	0	0	0
14.本科目教師之整體教學表現值得讚許	2	5	1	0	0

數學影像處理的相關理論及實作專題 II

基本資料					
1.這門課我的缺席次數	從不缺課	缺課 1-3 週	缺課 4-6 週	缺課 7-12 週	缺課 13 週以上
	2	2	0	0	0
2.除上課時間外，平均一星期修讀本課程付出的時間	10 小時以上	6-9 小時	4-5 小時	1-3 小時	1 小時以下
	1	0	1	2	0
對本科目之教學意見	非常同意	同意	普通	不同意	非常不同意
3.我有修習本課程	2	1	1	0	0
4.我對這門課的學習態度很認真	2	2	0	0	0
5.教師提供完整的課程大綱，且教學內容與課程大綱相符	3	1	0	0	0
6.教師的課前準備充分，上課內容豐富充實	3	1	0	0	0
7.我在課程中感受到教師的教學熱忱	3	1	0	0	0
8.本課程授課內容組織完善，有助學習	3	1	0	0	0
9.教材設計能顧及學生的學習狀況	3	1	0	0	0

10.教師具備講授本課程之專業知識	3	1	0	0	0
11.教師講解表達方式良好，使課程容易瞭解	3	1	0	0	0
12.教師樂於協助學生解決有關本課程之疑問	3	1	0	0	0
13.教師(或助教)對試卷、作業或報告的評分公平合理	3	1	0	0	0
14.本科目教師之整體教學表現值得讚許	3	1	0	0	0

(6) 建議與反思

因為這次課程時間安排每周兩小時，為課程上教授理論，讓學生回去參考教材實作，但後續發現學生實作能力跟不上，所以需要安排助教及實作課程及派發作業，讓學生具備有基礎的實作能力。在助教的選擇上，則可以考慮過去修課情況較好的學生來擔任。

學生實作成果

在學期末成果報告中，學生可藉由閱讀相關文件並結合上課內容進行實作探討，以下以部分修課同學之學習成果為例。

- **傅立葉運算應用於影像濾波技術**

學生一在習得傅立葉轉換之必要數學背景知識後，實際理解快速傅立葉變換之演算過程、複雜度分析、以及程式實作，並對於轉換至頻譜空間上之圖像處理對應到影像過濾高頻資訊之結果。

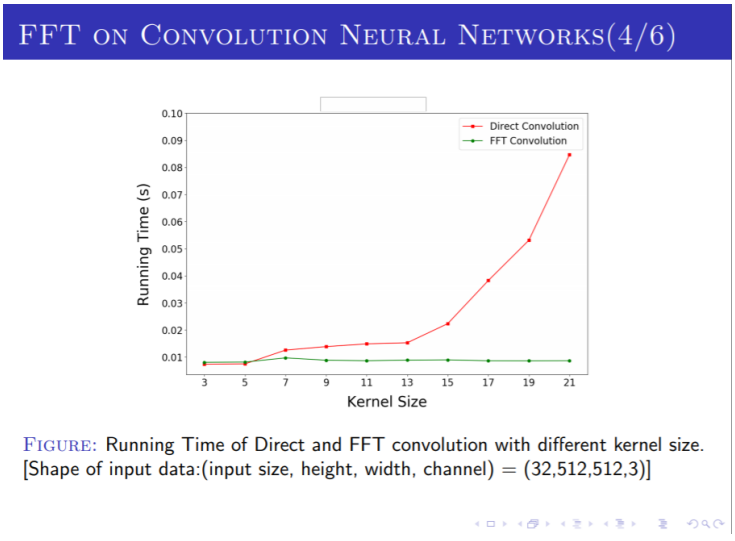
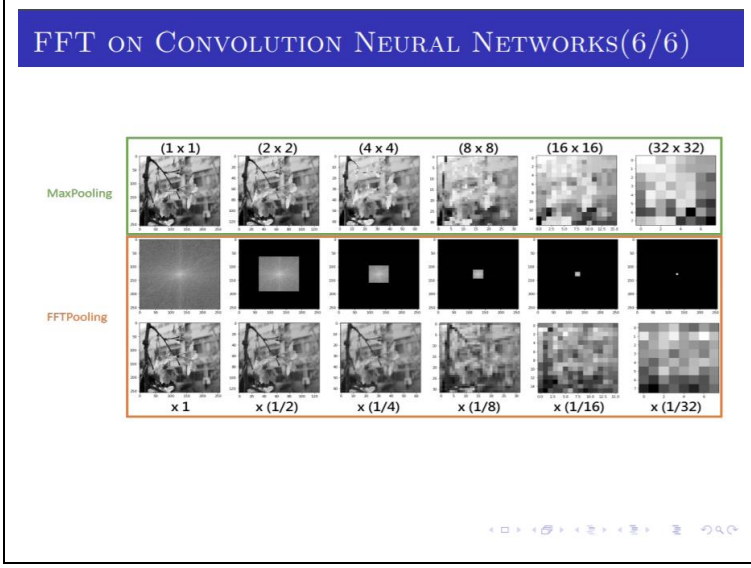


FIGURE: Running Time of Direct and FFT convolution with different kernel size. [Shape of input data:(input size, height, width, channel) = (32,512,512,3)]

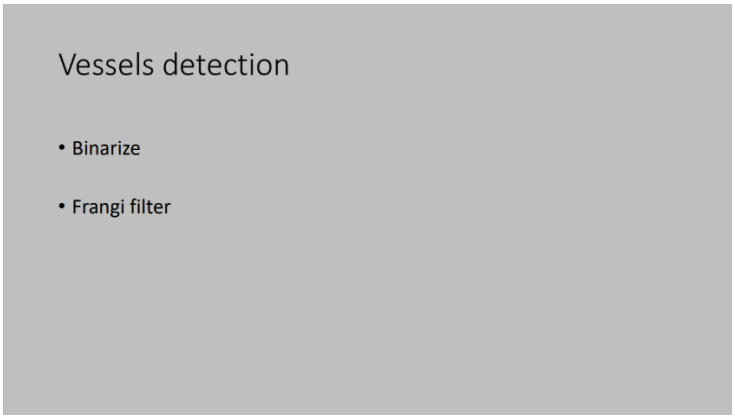
分析原始傅立葉轉換 (DFT) 與快速傅立葉轉換 (DFT) 之執行時間比較

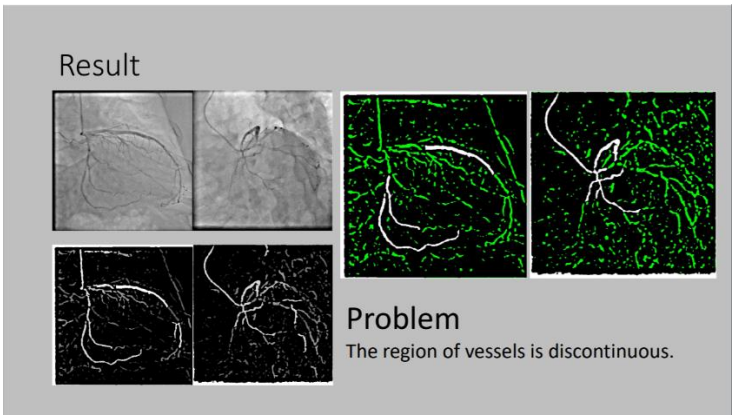
	於頻譜空間之處理對應到影像濾波效果

- **Vessels Detection 應用於醫學血管偵測處理**

學生二建立在影像二值化處理與基礎濾波等相關背景知識上，閱讀相關文獻並實際於真實資料上透過全域大津二值化運算 (Otsu’s Method) 與 Frangi 濾波器實現心臟血管偵測。

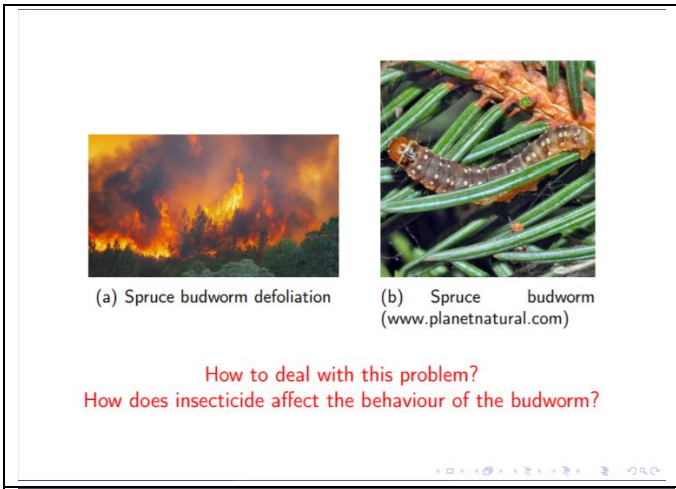
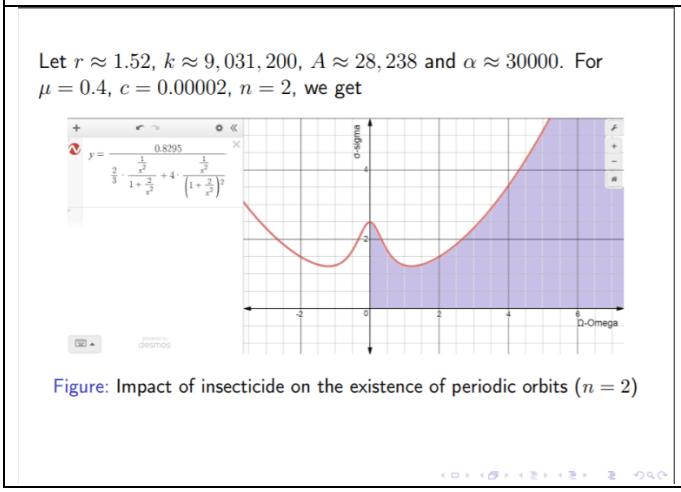
實作過程中需要藉由人工調校參數以得到適當成果，並且在雜訊等外在環境干擾下可能會得到品質不好的結果，學生不僅在實作報告中提到相關問題，並與課程成員討論在品質提升與自動化等未來工作上可能的解決方向。

	列舉 Vessels Detection 中之必要技術

	實際應用於心臟血管偵測之結果

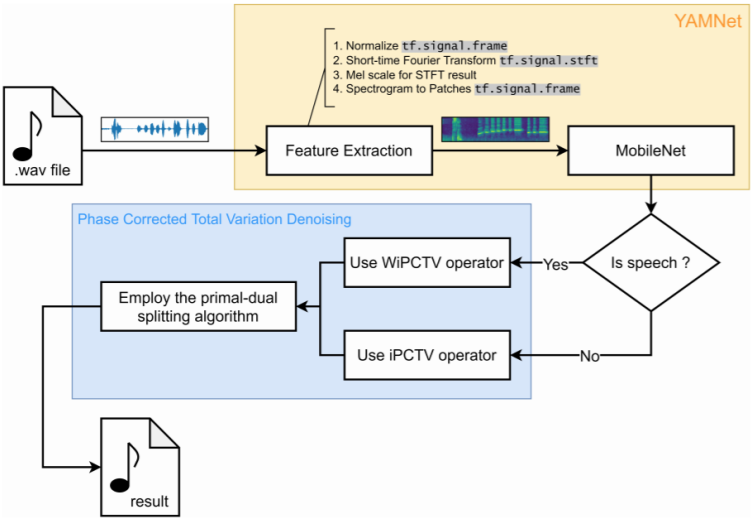
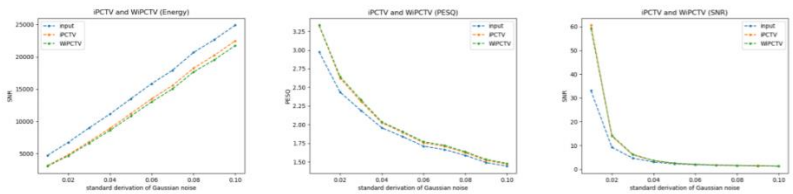
- **數學建模應用於雲杉芽蟲、鳥類、與殺蟲劑之動態系統**

學生三體會數學在影像處理上的建模思維，應用於森林生態圈中之動態系統。森林大火的事件，反思人類在介入生態圈的同時，其不當行為可能造成之反撲。學生透過數學建模與微分方程等工具建構其數學問題，並探討在各項條件下，說明在人類不同程度的干預下在模型上所反應出來的結果。

	說明模型探討動機
	演示其數值結果並說明對應之人類行為對環境之影響

- **基於頻譜與機器學習之聲紋分析及音訊強化**

學生四在理解全變異數去雜訊之實作方法下，結合短時距傅立葉轉換 (STFT) 與機器學習技術，實現對於不同音訊類型之自適應性音訊強化成果。

<table>
<tr>
<td> 
</td>
<td>結合機器學習與全變異數之方法流程</td>
</tr>
<tr>
<td> 
(a) 能量函數比較 (b) PESQ 比較 (c) SNR 比較</td>
<td>實作成果反應於指標 PESQ 與 SNR 之比較</td>
</tr>
</table>

(7) 參考文獻(References)

[1] An Introduction to Mathematical Image Processing IAS, Park City Mathematics Institute, Utah Undergraduate Summer School 2010

[2] Lai Kang, etc. (2016). Rectification of curved document images based on single view three-dimensional reconstruction

[3] Gaofeng Meng, Chunhong Pan, etc. (2012). Metric Rectification of Curved Document Images

[4] Ivan W. Selesnick and ˙Ilker Bayram (2010). Total Variation Filtering.

[5] Laurent Condat (2012). A Direct Algorithm for 1D Total Variation Denoising

[6] C.Y. Yang; Y.C. Zhang; Y.H. Chen and C.W. Huang (2018). Toward semantic loop closure in simultaneous localization and mapping systems

[7] V. Blanz; T. Vetter (1999). A Morphable Model For The Synthesis of 3D Faces

[8] W. Hariri; H. Tabia; N Farah; A Benouareth and D. Declercq (2016). 3D face recognition using covariance based descriptors

[9] F. Schroff; D. Kalenichenko; and J. Philbin (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering

[10] 演算法筆記 http://www.csie.ntnu.edu.tw/~u91029/Image.html

[11] 價創計畫：CIVIS 首頁 https://www.civis.com.tw/
[12] 數位影像處理 (Digital Image Processing, 3/e) Rafael C. Gonzalez, Richard E. Woods 著、繆紹綱 譯，ISBN: 9866534103
[13] QR code 2006 標準: http://download.adamas.ai/dlbase/Stuff/%21ISO/ISO_IEC-18004-2006.pdf
[14] 曾定章教授影像處理課程 http://ip.csie.ncu.edu.tw/course/course.htm
[15] 徐子仁(2016) 基於深度學習的人臉辨識系統
[16] Keras 文件: https://keras.io/
[17] 2016, 戴維良, 廖子鈞,“基於視覺密碼之彩色機密隱藏機制”
[18] 2007, Davide Maltoni,Dario Maio,Anil K. Jain,Salil Prabhakar,Handbook of Fingerprint Recognition
[19] 2007,潘正祥、張真誠、林詠章 ,挑戰影像處理：數位浮水印技術
[20] 2012,楊任芯, 指紋辨識
[21] 1985, Malcolm K. Sparrow and Penelope J. Sparrow, A Topological Approach to the Matching of Single Fingerprints: Development of Algorithms for Use on Latent Fingermarks
[22] 2002, Asker M. Bazen and Sabih H. Gerez, Systematic Methods for the Computation of the Directional Fields and Singular Points of Fingerprints
[23] 2007, 黃啟禎, 指紋辨識系統之設計與實現
[24] 1995, KALLE KARU and ANIL K. JAIN, FINGERPRINT CLASSIFICATION
[25] 2004, 侯永昌, 杜淑芬, 像素不擴展之彩色視覺密碼技術
[26] 2008, 黃樹乾、黃酩閔, 基於視覺密碼學之影像浮水印技術

2. 附件(Appendix)

Note for FFT on Image Processing

Bo-Cyuan Lin

Department of Mathematics, National Central University, Taiwan

kobayasiyuli1833@g.ncu.edu.tw

Orthonormal Family

Definition

(Normalized)
$(\nu, <,>)$m inner product space. $\varphi \in \nu$ is called normalized, if $||\varphi|| = \sqrt{<\varphi, \varphi>} = 1$.

Definition

(Orthonormal Family)
$(\nu, <,>)$, inner product space, $\varphi_0, \varphi_1, \varphi_2, ... \in \nu$ is called an orthonormal family, if it satisfies:

$$\begin{cases} <\varphi_i, \varphi_j> = 1, & \text{if } i = j \\ <\varphi_i, \varphi_j> = 0, & \text{if } i \neq j \end{cases}$$

Fourier Series and Fourier Coefficients

Definition

(Complete)
An orthomormal family $\varphi_0, \varphi_1, \varphi_2, ...$ in an inner product space ν is complete, if $\forall\ f \in \nu,\ f = \sum_{k=0}^{\infty} c_k \varphi_k$.

Note that, $\{\varphi_k\}$ is complete
$\Leftrightarrow \forall\ f \in \nu,\ ||f - \sum_{k=0}^{n} < f, \varphi_k > \varphi_k|| \to 0,\ as\ n \to \infty$.

Definition

(Fourier Series and Fourier Coefficient)
We call $\sum_{k=0}^{\infty} < f, \varphi_k > \varphi_k$ Fourier Series of f with respect to $\varphi_0, \varphi_1, \varphi_2, ...$ and $< f, \varphi_k >$ is the Fourier coefficients.

Conception of Fourier Transform I

Definition

(Exponential Fourier Series)

$F(t) = \sum_{k=-\infty}^{\infty} c_k e^{\frac{2\pi ikt}{T}}$, where

$c_k = \frac{1}{T}\int_{\frac{T}{2}}^{-\frac{T}{2}} f(t)e^{\frac{-2\pi ikt}{T}}dt,\ k \in \mathbb{Z},\ t \in [-\frac{T}{2}, \frac{T}{2}],\ f \in L^2$, satifies Dirichlet Lemma (i.e. $||f - F||^2 = 0$).

(Derivation of the Fourier Transform)

Let $f_T(t)$ be the $T - period$ function, $t \in [-\frac{T}{2}, \frac{T}{2}],\ f_t(t) = f(t)$.

Define $\nu_k = \frac{k}{T}$ $s.t.$ $\nabla\nu = \nu_{k+1} - \nu_k = \frac{1}{T}$. Then the Fourier Series of $f_T(t)$:

$F(t) = \sum_{k=-\infty}^{\infty} c_k(T)e^{\frac{2\pi ikt}{T}}$, where $c_k(T) = \frac{1}{T}\int_{\frac{T}{2}}^{-\frac{T}{2}} f(t)e^{\frac{-2\pi ikt}{T}}dt,\ k \in \mathbb{Z}$.

$$\Rightarrow F(t) = \sum_{k=-\infty}^{\infty} \frac{1}{T}\int_{\frac{T}{2}}^{-\frac{T}{2}} f(t)e^{\frac{-2\pi ikt}{T}}dte^{\frac{2\pi ikt}{T}}$$

Conception of Fourier Transform II

$$= \sum_{k=-\infty}^{\infty} \int_{\frac{T}{2}}^{-\frac{T}{2}} f(t) e^{\frac{-2\pi ikt}{T}} dt \; e^{\frac{2\pi ikt}{T}} \frac{1}{T}$$

$$= \sum_{k=-\infty}^{\infty} \int_{\frac{T}{2}}^{-\frac{T}{2}} f(t) e^{-2\pi i \nu_k t} dt \; e^{2\pi i \nu_k t} \bigtriangledown \nu$$

As $T \to \infty \Rightarrow \bigtriangledown \nu \to d\nu$ with $\sum_{k=-\infty}^{\infty} \to \int_{-\infty}^{\infty}$

$$\Rightarrow f(t) = F(t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t) e^{-2\pi i \nu t} dt \; e^{2\pi i \nu t} d\nu.$$

Note that $\hat{f}(\nu) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \nu t} dt$, where ν is the frequency and $f(t) = \int_{-\infty}^{\infty} \hat{f}(\nu) e^{2\pi i \nu t} d\nu$.

Conception of Fourier Transform III

Definition

(Fourier Transform)
Let f be a period function and bounded Riemann integrable, then Fourier Transform:
$\hat{f}(\nu) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i\nu t}dt$, where ν is the frequency.

Definition

(Inverse Fourier Transform)
Let f be a period function and bounded Riemann integrable and $\hat{f}$ be the Fourier Transform of f, then Inverse Fourier Transform:
$f(t) = \int_{-\infty}^{\infty} \hat{f}(\nu)e^{2\pi i\nu t}d\nu$.

Matrix Form

Use the Euler Formula we have $\omega = \omega(N) = e^{\frac{-2\pi i}{N}}$, and express the DFT with the form $y = Fx$ as the following:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{pmatrix}_{N\times 1} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{(N-1)} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & & & & \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)^2} \end{pmatrix}_{N\times N} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{pmatrix}_{N\times 1}$$

Where F is a symmetric complex matrix.
However, if we compute the DFT directly with computer, it may cost much of time. The computational complexity of general DFT is $O(N^2)$.

FFT Operations

Consider the general cases $m = 2^k,\ k \in \mathbb{Z}^+$, since $R_m^{-1} = R_m^T$.

$$\Rightarrow F_m = \begin{pmatrix} F_{\frac{m}{2}} & D_{\frac{m}{2}} F_{\frac{m}{2}} \\ F_{\frac{m}{2}} & -D_{\frac{m}{2}} F_{\frac{m}{2}} \end{pmatrix}_{m \times m} R_m, \text{where}$$

$$R_m^T = \begin{pmatrix} e_0 & e_2 & e_{m-2} & \cdots & e_1 & e_3 & e_{m-1} & \cdots \end{pmatrix}_{m \times m}, \text{ and}$$

$$D_{\frac{m}{2}}^T = diag(1, \omega^{\frac{N}{m}}, \omega^{\frac{2N}{m}}, ..., \omega^{\frac{N}{2} - \frac{N}{m}})_{\frac{m}{2} \times \frac{m}{2}}$$

Operations:

1. Rearrange the sequences.

2. Divide-and-Conquer until with size 2.

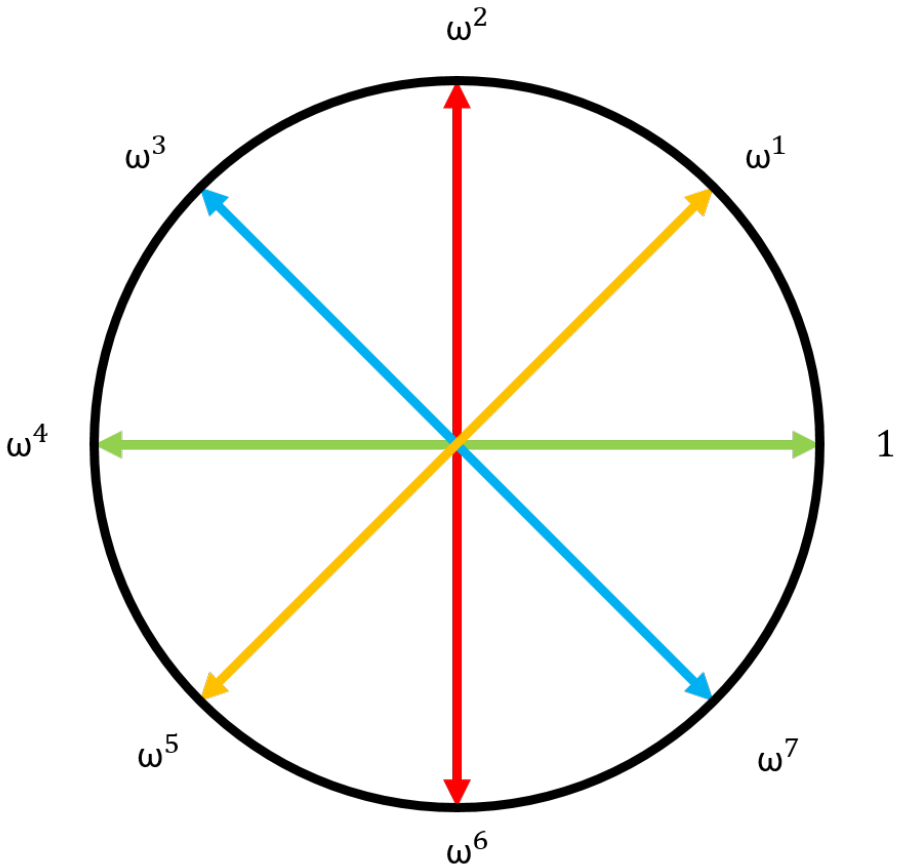
Some Properties about complex roots:

1. Let the complex set $\{1, \omega, \omega^2, ..., \omega^{N-1}\}$ be the roots of $z^N = 1$, actually they are lie on the unit circle in the complex plane. Moreover, $\{1, \omega^2, \omega^4, ..., \omega^{N-2}\}$ is the roots of $z^{\frac{N}{2}}$. Hence, based on this conception we have $m = 2^k,\ k \in \mathbb{Z}^+,\ \{1, \omega^m, \omega^{2m}, ..., \omega^{N-m}\}$ is a root of $z^{\frac{N}{m}}$.

2. Since the ω has period, if $k \geq N$ we have $\omega^k = \omega^{k \ mod \ N}$.
For example, If $N = 8$, then $\omega^8 = 1, \omega^9 = \omega, ...$

3. ω is symmetric (i.e. $\omega^{k+\frac{N}{2}} = -\omega^k$). Since Euler Formula we have
$\omega^p = e^{\frac{2\pi ip}{N}} = cos(\frac{2\pi ip}{N}) + isin(\frac{2\pi ip}{N})$.
$\Rightarrow \omega^{k+\frac{N}{2}} = e^{2\pi \frac{k+\frac{N}{2}}{N}} = e^{2\pi \frac{k}{N}+\pi} = -e^{2\pi \frac{k}{N}} = -\omega^k$.
For example, If $n = 8$, then $\omega^4 = -1, \omega^5 = -\omega, \omega^6 = -\omega^2, \omega^7 = -\omega^3$.

Root for $n = 8$



Example for $n = 8$ (1/5)

$$\Rightarrow F_8 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^3 & \omega^3 & \omega^2 & \omega \end{pmatrix}_{8\times 8}$$

Example for $n = 8$ (2/5)

$$= \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & -1 & -\omega & -\omega^2 & -\omega^3 \\ 1 & \omega^2 & -1 & -\omega^2 & 1 & \omega^2 & -1 & -\omega^2 \\ 1 & \omega^3 & -\omega^2 & \omega & -1 & -\omega^3 & \omega^2 & -\omega \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -\omega & \omega^2 & -\omega^3 & -1 & \omega & -\omega^2 & \omega^3 \\ 1 & -\omega^2 & -1 & \omega^2 & 1 & -\omega^2 & -1 & \omega^2 \\ 1 & -\omega^3 & -\omega^2 & -\omega & -1 & \omega^3 & \omega^2 & \omega \end{pmatrix}_{8\times 8}$$

Example for $n = 8$ (3/5)

Consider the rearrange matrix:

$$\Rightarrow R_8^T = \begin{pmatrix} e_0 & e_2 & e_4 & e_6 & e_1 & e_3 & e_5 & e_7 \end{pmatrix}_{8\times 8}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}_{8\times 8}$$

Example for $n = 8$ (4/5)

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^6 & \omega^4 & \omega^2 \end{pmatrix}_{4\times 4} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & -1 & -\omega^2 \\ 1 & -1 & 1 & -1 \\ 1 & -\omega^2 & -1 & \omega^2 \end{pmatrix}_{4\times 4}$$

Example for $n = 8$ (5/5)

$$F_8 R_8^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^2 & -1 & -\omega^2 & \omega & \omega^3 & -\omega & -\omega^3 \\ 1 & -1 & 1 & -1 & \omega^2 & -\omega^2 & \omega^2 & -\omega^2 \\ 1 & -\omega^2 & -1 & \omega^2 & \omega^3 & \omega & -\omega^3 & -\omega \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & \omega^2 & -1 & -\omega^2 & -\omega & -\omega^3 & \omega & \omega^3 \\ 1 & -1 & 1 & -1 & -\omega^2 & \omega^2 & -\omega^2 & \omega^2 \\ 1 & -\omega^2 & -1 & \omega^2 & -\omega^3 & -\omega & \omega^3 & \omega \end{pmatrix}_{8\times 8}$$

$$= \begin{pmatrix} F_4 & D_4 F_4 \\ F_4 & -D_4 F_4 \end{pmatrix}_{8\times 8}, (D_4 = diag(1, \omega, \omega^2, \omega^3)_{4\times 4})$$

Cooley Tukey Algorithm (FFT)

Algorithm

(FFT Radix-2)

Let $\omega_N = e^{-\frac{2\pi i}{N}}$, then both of y_k and $y_{k+\frac{N}{2}}$ has $\frac{N}{2}$ terms.

$$\begin{aligned} y_k &= \sum_{n=2t} \omega_N^{kn} x_n + \sum_{n=2t+1} \omega_N^{kn} x_n \\ &= \sum_t \omega_{\frac{N}{2}}^{kt} x_{2t} + \omega_N^k \sum_t \omega_{\frac{N}{2}}^{kt} x_{2t+1} \\ &= F_{even}(k) + \omega_N^k F_{odd}(k) \end{aligned}$$

$$y_{k+\frac{N}{2}} = F_{even}(k) - \omega_N^k F_{odd}(k), \text{ and } y_k = F_{even}(k) + \omega_N^k F_{odd}(k)$$

$\mathcal{O}(N^2 log(N^2)) = \mathcal{O}(2N^2 log(N)) = C2N^2 log(N)$, *where* C is a hidden constant of $\mathcal{O}$

Butterfly Diagram

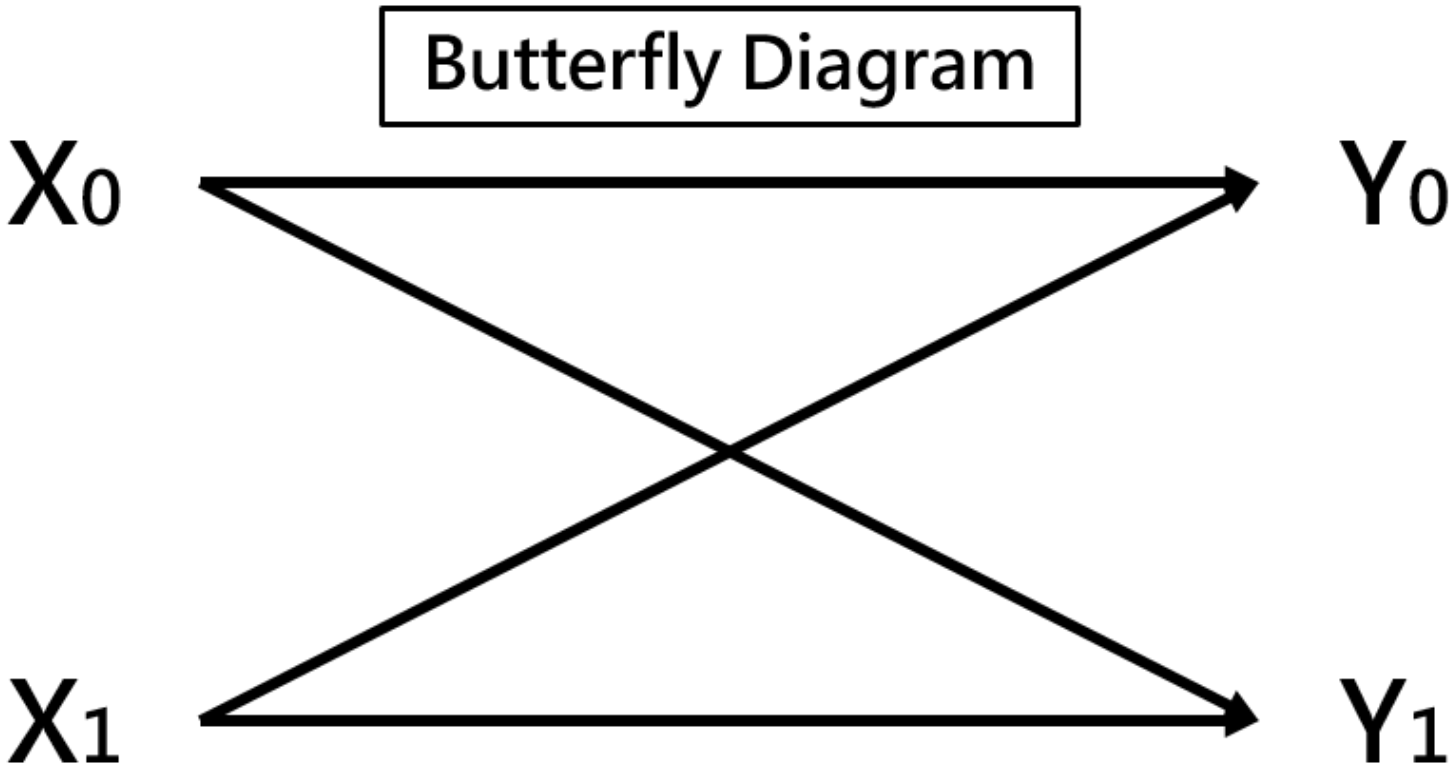


Figure: Butterfly Diagram

Parallel FFT

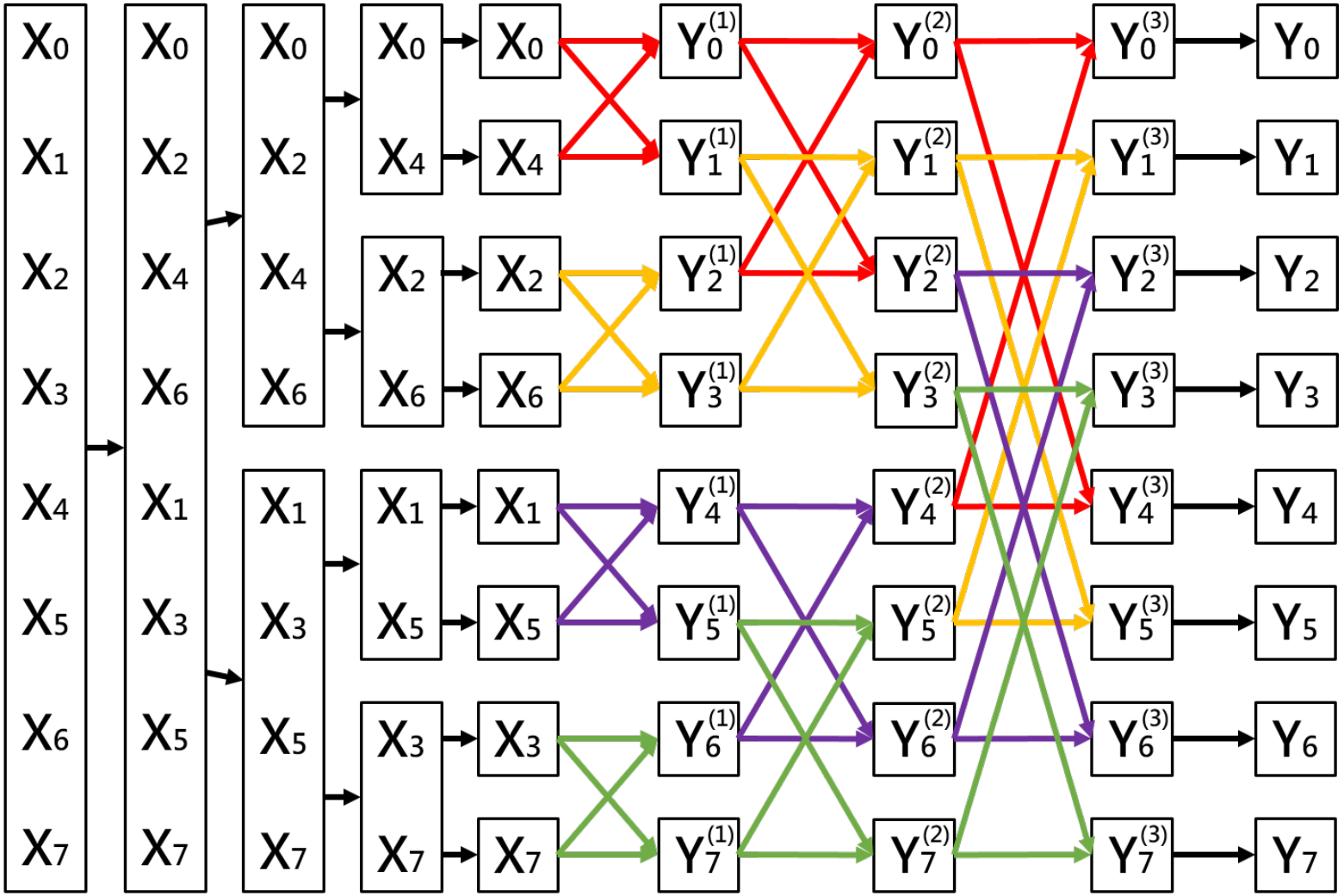


Figure: Different colors in same layer can be calculated parallel.

Complexity

For the $n \times n$ image with $n = 2^p$ (Radix-2), where p is the amounts of the layer about butterfly diagram, we have the complexity as the following:

FT	Complexity of Computation
DFT	n^2
FFT	$\frac{n}{2} log(n)$

Table: Computing Complexity of DFT and FFT

FFT on Convolution Neural Networks(1/6)

Theorem

(Convolution)
Let $f, g \in L^1(\mathbb{R}^n)$ be two function with convolution $f * g$, and $\mathcal{F}$ denotes Fourier Transform, where $\mathcal{F}(f), \mathcal{F}(g)$ are the Fourier Transform of f, g. Then,

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$$

where $\cdot$ denotes point-wise multiplication.

(Fourier Convolution Layer)
Note that discrete analogue of the Convolution Theorem:

$$\mathcal{F}([kernel] * [data]) = \mathcal{F}([kernel]) \cdot \mathcal{F}([data])$$

where $\mathcal{F}$ denotes the Fourier Transform.

FFT on Convolution Neural Networks(2/6)

Algorithm

(Fourier Convolution Layer)

1. $[kernel]^i_{\mathcal{F}} = \mathcal{F}([kernel]^i),\ i = 1, ..., N_{[kernel]},\ where\ N_{[kernel]}$: kernel amount.

2. $[data]^j_{\mathcal{F}} = \mathcal{F}([data]^j),\ j = 1, ..., N_{[data]},\ where\ N_{[data]}$: data amount.

3. $[output]^{i,j}_{\mathcal{F}} = [kernel]^i_{\mathcal{F}} \odot [data]^j_{\mathcal{F}},\ i = 1, ..., N_{[kernel]},\ j = 1, ..., N_{[data]},\ where \odot : Hadamard\ Product.$

4. $[output]^{i,j} = \mathcal{F}^{-1}([output]^{i,j}_{\mathcal{F}}),\ i = 1, ..., N_{[kernel]},\ j = 1, ..., N_{[data]}.$

The algorithm reduces the number of operations gives an increasing relative speed-up for larger kernels.

FFT on Convolution Neural Networks(3/6)

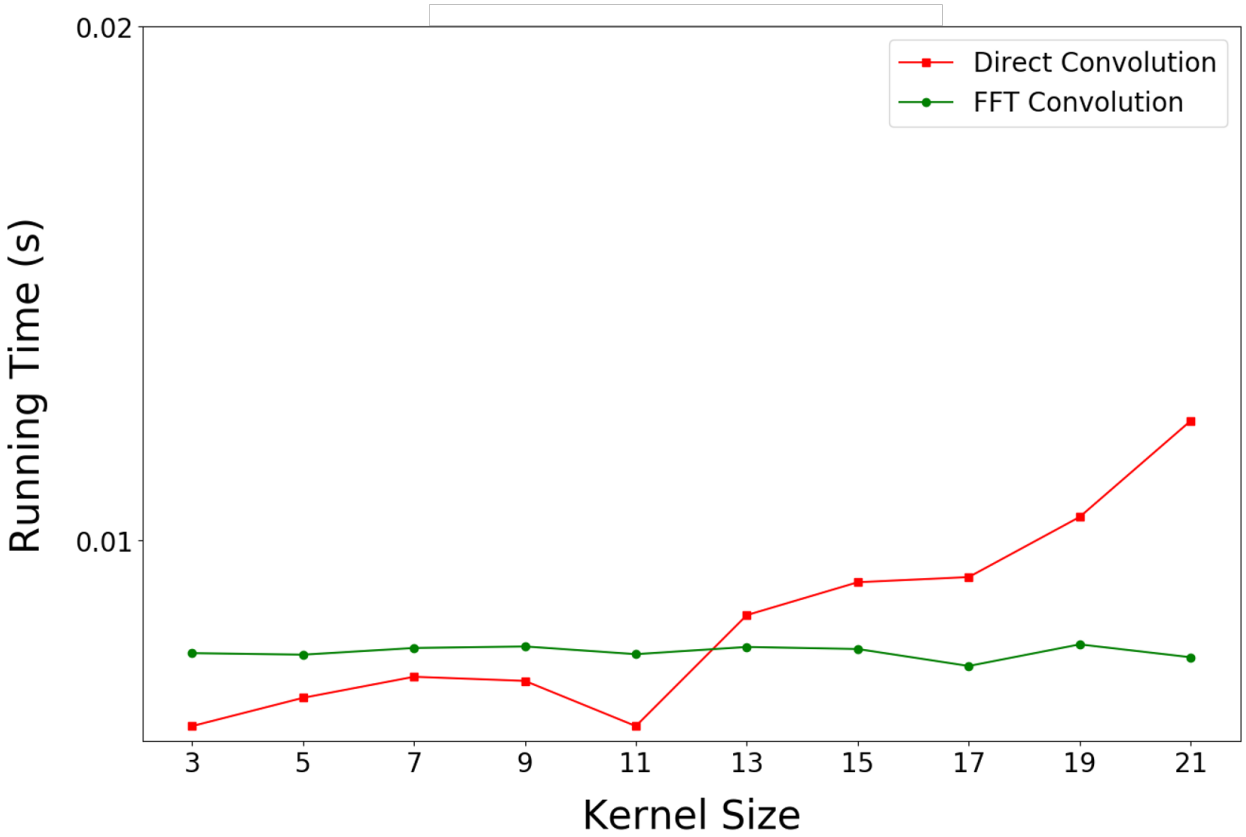


Figure: Running Time of Direct and FFT convolution with different kernel size. [Shape of input data:(input size, height, width, channel) = (32,256,256,3)]

FFT on Convolution Neural Networks(4/6)

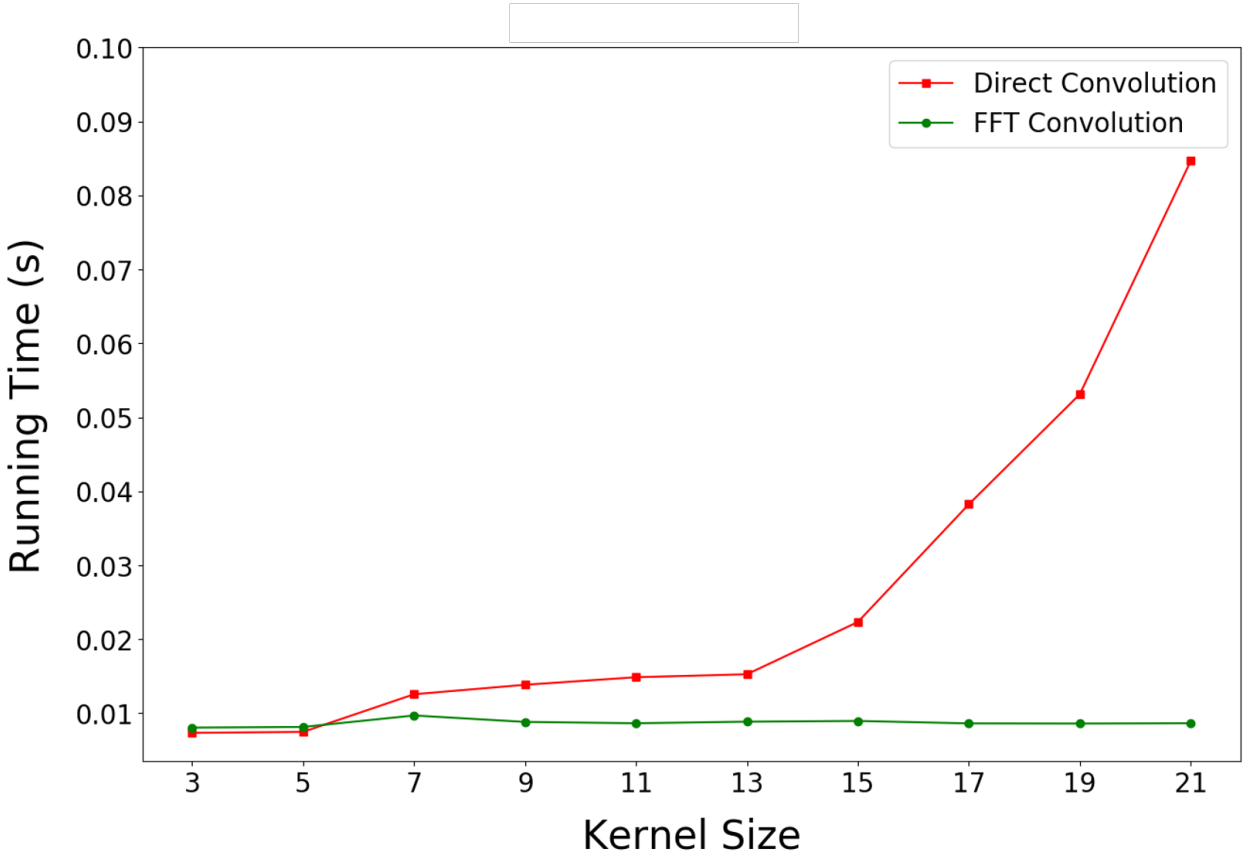


Figure: Running Time of Direct and FFT convolution with different kernel size. [Shape of input data:(input size, height, width, channel) = (32,512,512,3)]

Algorithm

(Fourier Pooling)
Given a complex 3D tensors (x: data set, y: data height, z: data width), $\forall$ *image* $\in x$ we retain the region $[y_{p\ min}, y_{p\ max}] \times [z_{p\ min}, z_{p\ max}]$ as follows:

$$y_{p\ min} = (0.5 - \frac{poolsize}{2}) \times y,\ y_{p\ max} = (0.5 + \frac{poolsize}{2}) \times y$$

$$z_{p\ min} = (0.5 - \frac{poolsize}{2}) \times z,\ z_{p\ max} = (0.5 + \frac{poolsize}{2}) \times z$$

$$y_{p\ min} = 0.375 \times y,\ y_{p\ max} = 0.625 \times y$$

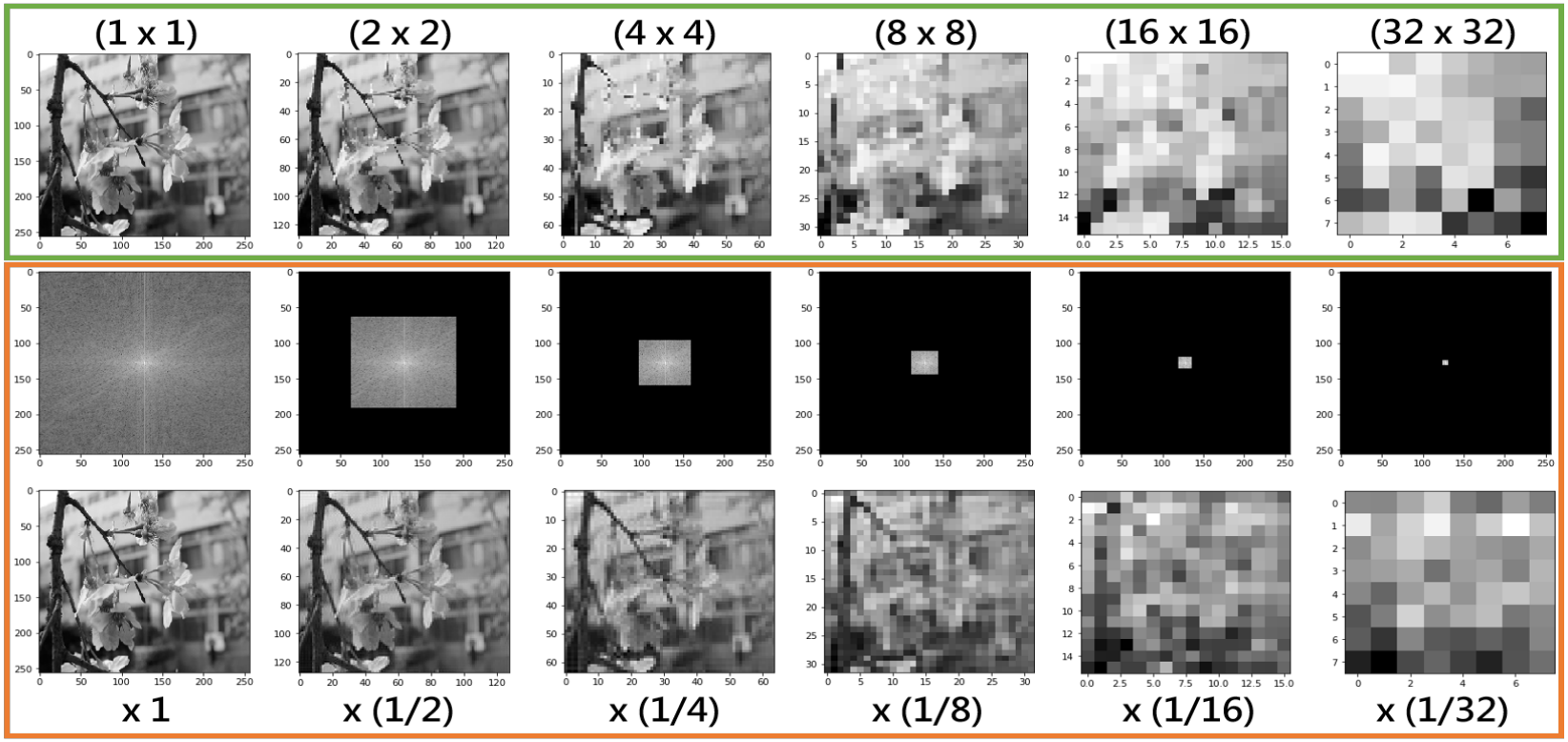
$$z_{p\ min} = 0.375 \times z,\ z_{p\ max} = 0.625 \times z$$

which reduces the data by a quarter.

FFT on Convolution Neural Networks(6/6)

MaxPooling

FFTPooling



Reference I

[1] Elementary Classical Analysis (2nd Edition); Jerrold E. Marsden, Michael J. Hoffman (1974)

[2] http://www.math.ncu.edu.tw/ cchsiao/Course/Fourier_Analysis_051/Fourier_Analysis.pdf

[3] https://ccjou.wordpress.com/

[4] A. James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation* **19**(90), pp. 297–301

[5] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *CoRR(2013)*

[6] Oren Rippel, Jasper Snoek and Ryan P. Adams, Spectral Representations for Convolutional Neural Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), *Advances in neural information processing systems, 28*, pp. 2449–2457

Reference II

[7] Pratt H., Williams B., Coenen F., Zheng Y. (2017) FCNN: Fourier Convolutional Neural Networks. In: Ceci M., Hollmén J., Todorovski bibitL., Vens C., Džeroski S. (eds) *Machine Learning and Knowledge Discovery in Databases*, pp. 786–798

THE DYNAMIC OF SPRUCE BUDWORM - BIRDS INTERACTION WITH THE EFFECT OF INSECTICIDE

Le Nguyen Thuy Van [1]

[1,2]Department of Mathematics
National Central University
Advisor: Prof. Jann-Long Chern [2]

June 15th, 2020

1. Introduction to Spruce budworm - birds interaction
2. Statements of the model
3. Stability analysis
   - Equilibrium points
   - Periodicity of solutions
4. Biological considerations

1. Introduction

Introduction

- The spruce budworm, Choristoneura fumiferana, is prevalent in eastern North America. Budworm outbreaks cause various negative effects such as loss of productivity of forests, defoliation.
- Studying about the behaviours of spruce budworm has attracted many researchers (eg. Ludwig (1978), Jacai deNeveu (2015), Raina Robeva and David Murrugarra (2016), Xiaofeng Xu and Junjie Wei (2017))

(a) Spruce budworm defoliation

(b) Spruce budworm (www.planetnatural.com)

How to deal with this problem?
How does insecticide affect the behaviour of the budworm?

Statements of the model

Statements of the model

$$\frac{dB}{dt} = -\mu B + c\alpha \frac{B\omega^2}{A^2 + \omega^2} \tag{2.1}$$

$$\frac{d\omega}{dt} = r\omega(1 - \frac{\omega}{k}) - \alpha B \left(\frac{\omega^2}{A^2 + \omega^2} \right) - \frac{\sigma\Omega \left(\frac{\omega}{\Omega} \right)^{n+1}}{1 + \left(\frac{\omega}{\Omega} \right)^{n}} \tag{2.2}$$

where B is bird density, ω is budworm density, r is maximum growth rate of budworm, k is carrying capacity of the forest, A is budworm population when the predation rate is at half of its maximum, μ is death rate of birds, σ is maximum budworm mortality rate due to the action of insecticide, Ω is critical density.

We introduce new quantities by writing $\omega = A\bar{\omega}$, $t = \dfrac{\tau}{c\alpha}$, $B = cA\bar{B}$. Then

$$\frac{d\bar{B}}{d\tau} = -\bar{\mu}\bar{B} + \frac{\bar{B}\bar{\omega}^2}{1+\bar{\omega}^2} \tag{2.3a}$$

$$\frac{d\bar{\omega}}{d\tau} = \bar{r}\bar{\omega}\left(1 - \frac{\bar{\omega}}{\bar{k}}\right) - \bar{B}\frac{\bar{\omega}^2}{1+\bar{\omega}^2} - \frac{\bar{\sigma}\bar{\Omega}\left(\dfrac{\bar{\omega}}{\bar{\Omega}}\right)^{n+1}}{1+\left(\dfrac{\bar{\omega}}{\bar{\Omega}}\right)^{n}} \tag{2.3b}$$

For the simplicity, we use B, ω,... instead of $\bar{B}$, $\bar{\omega}$,... respectively.

Equilibrium points

The system (3) has three equilibrium points $E_1 = (0, 0)$, $E_2 = (0, \omega_\Omega)$, $E_3 = (B^*, \omega^*)$.
Note:

- ω_Ω satisfies $r\left(1 - \frac{\omega_\Omega}{k}\right) - \frac{\sigma\left(\frac{\omega_\Omega}{\Omega}\right)^n}{1 + \left(\frac{\omega_\Omega}{\Omega}\right)^n} = 0$
- $\omega^* = \frac{\mu}{\sqrt{\mu - \mu^2}}$
- $B^* = \frac{1 + (\omega^*)^2}{\omega^*}\left(r(1 - \frac{\omega^*}{k}) - \frac{\sigma\left(\frac{\omega^*}{\Omega}\right)^n}{1 + \left(\frac{\omega^*}{\Omega}\right)^n}\right)$

Stability analysis

Jacobian matrix: $J = \begin{bmatrix} \dfrac{\omega^2}{1+\omega^2} - \mu & \dfrac{2B\omega}{(1+\omega^2)^2} \\ -\dfrac{\omega^2}{1+\omega^2} & \Phi \end{bmatrix}$

Where $\Phi = r(1 - \dfrac{2\omega}{k}) - \dfrac{2B\omega}{(1+\omega^2)^2} - \dfrac{\sigma\left(\dfrac{\omega}{\Omega}\right)^n}{1+\left(\dfrac{\omega}{\Omega}\right)^n} - \dfrac{\sigma n\left(\dfrac{\omega}{\Omega}\right)^n}{\left(1+\left(\dfrac{\omega}{\Omega}\right)^n\right)^2}$

Stability of E_1 and E_2

$$J(E_1) = \begin{bmatrix} -\mu & 0 \\ 0 & r \end{bmatrix} \Rightarrow E_1 \text{ is a saddle point.}$$

$$J(E_2) = \begin{bmatrix} \frac{\omega_\Omega^2}{1+\omega_\Omega^2} - \mu & 0 \\ -\frac{\omega_\Omega^2}{1+\omega_\Omega^2} & \Phi \end{bmatrix}$$

where $\Phi(E_2) = r(1 - \frac{2\omega_\Omega}{k}) - \frac{\sigma\left(\frac{\omega_\Omega}{\Omega}\right)^n}{1+\left(\frac{\omega_\Omega}{\Omega}\right)^n} - \frac{\sigma n\left(\frac{\omega_\Omega}{\Omega}\right)^n}{\left(1+\left(\frac{\omega_\Omega}{\Omega}\right)^n\right)^2}$

Hence, E_2 is stable if $\mu > \frac{\omega_\Omega^2}{1+\omega_\Omega^2}$. Otherwise, E_2 is unstable.

Stability of E_3

Note that: If E_2 is stable, E_3 does not exist.

For $\mu < \dfrac{\omega_\Omega^2}{1+\omega_\Omega^2}$, E_2 is unstable and there exists E_3.

$$J(E_3) = \begin{bmatrix} 0 & \dfrac{2B^*\omega^*}{(1+(\omega^*)^2)^2} \\ -\dfrac{(\omega^*)^2}{1+(\omega^*)^2} & \Phi \end{bmatrix}$$

where

$$\Phi(E_3) = r(1-\frac{2\omega^*}{k}) - \frac{2B^*\omega^*}{(1+(\omega^*)^2)^2} - \frac{\sigma\left(\dfrac{\omega^*}{\Omega}\right)^n}{1+\left(\dfrac{\omega^*}{\Omega}\right)^n} - \frac{\sigma n\left(\dfrac{\omega^*}{\Omega}\right)^n}{\left(1+\left(\dfrac{\omega^*}{\Omega}\right)^n\right)^2}$$

Stability of E_3

$$tr(J) = r(1 - \frac{2\omega^*}{k}) - \frac{2B^*\omega^*}{(1+(\omega^*)^2))^2} - \frac{\sigma\left(\frac{\omega^*}{\Omega}\right)^n}{1+\left(\frac{\omega^*}{\Omega}\right)^n} - \frac{\sigma n\left(\frac{\omega^*}{\Omega}\right)^n}{\left(1+\left(\frac{\omega^*}{\Omega}\right)^n\right)^2}$$

$$= -\frac{r\omega^*}{k} - \frac{2B^*\omega^*}{(1+(\omega^*)^2)^2} + \frac{B^*\omega^*}{1+(\omega^*)^2} - \frac{\sigma n\left(\frac{\omega^*}{\Omega}\right)^n}{\left(1+\left(\frac{\omega^*}{\Omega}\right)^n\right)^2}$$

If $trJ(B^*, \omega^*) < 0$, E_3 is stable.
If $trJ(B^*, \omega^*) > 0$, E_3 is unstable.

Periodicity of solutions

Theorem 2.1

Any solution of system

$$\frac{d\bar{B}}{d\tau} = -\bar{\mu}\bar{B} + \frac{\bar{B}\bar{\omega}^2}{1+\bar{\omega}^2}$$

$$\frac{d\bar{\omega}}{d\tau} = \bar{r}\bar{\omega}\left(1-\frac{\bar{\omega}}{\bar{k}}\right) - \bar{B}\frac{\bar{\omega}^2}{1+\bar{\omega}^2} - \frac{\bar{\sigma}\bar{\Omega}\left(\frac{\bar{\omega}}{\bar{\Omega}}\right)^{n+1}}{1+\left(\frac{\bar{\omega}}{\bar{\Omega}}\right)^{n}}$$

is bounded.

Proof of theorem 2.1.

- For $\omega \geq K$, $\omega^{'} \leq 0$.
- For $\omega < K$, $\omega < \omega^{*}$, $B^{'} \leq 0$.
- For $\omega < K$, $\omega > \omega^{*}$, we have the following lemma:

Lemma 2.1

Assume that for some $t_0 > 0$, $\epsilon > 0$ satisfy $B(t_0) = \bar{B}$, $\omega(t_0) = \bar{\omega}$ with $\bar{B} \geq \frac{rk}{4\mu} + \frac{\epsilon}{\mu}$ and $\bar{\omega} > \omega^{*}$. Then there exists $t^{*} > t_0$ such that $\omega(t^{*}) = \omega^{*}$.

Periodic solutions

Since the solution is bounded, we get the following theorems

Theorem 2.2

For $\mu > \dfrac{\omega_\Omega^2}{1+\omega_\Omega^2}$, E_2 is globally asymptotically stable.

Theorem 2.3

If $trJ(B^*, \omega^*) > 0$, the system has periodic solutions.

Proof. By using Poincaré-Bendixson Theorem.

Let $r \approx 1.52$, $k \approx 9,031,200$, $A \approx 28,238$ and $\alpha \approx 30000$. For $\mu = 0.4$, $c = 0.00002$, $n = 2$, we get

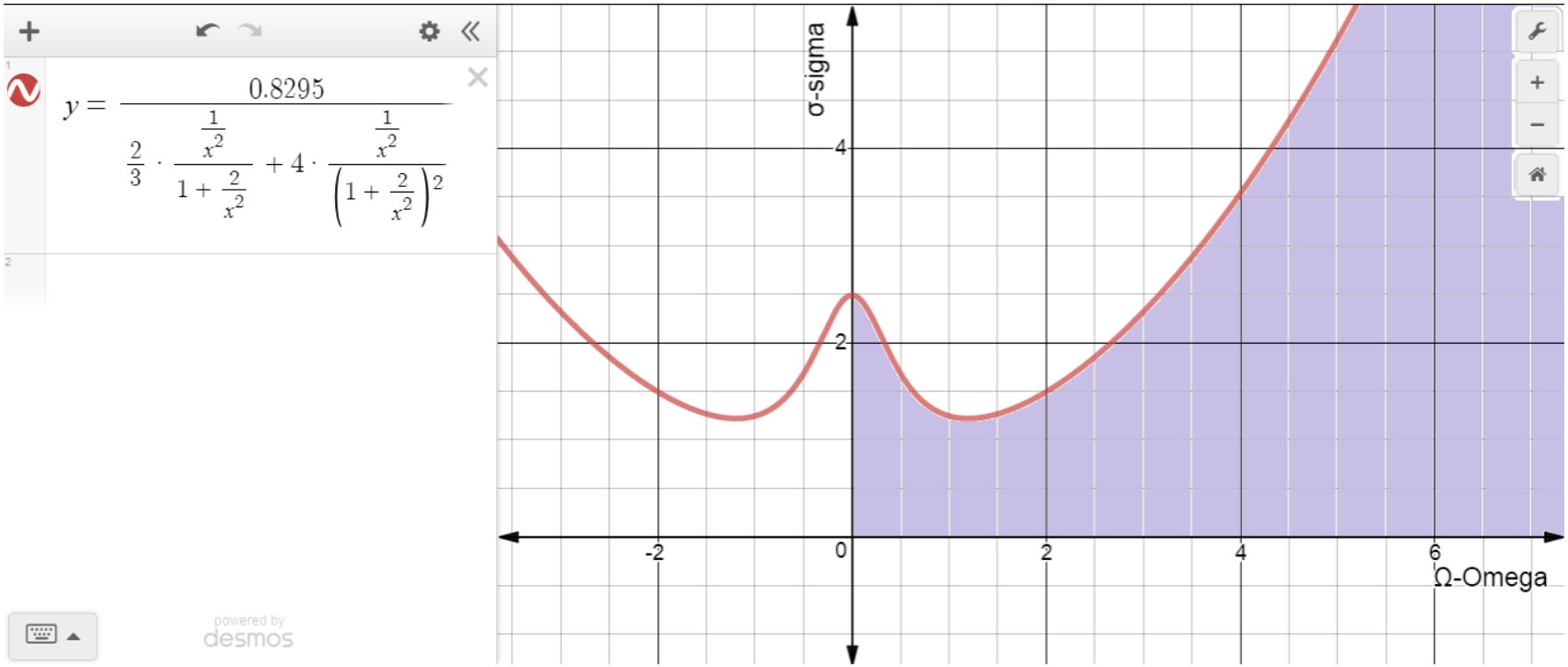


Figure: Impact of insecticide on the existence of periodic orbits ($n = 2$)

Periodic solutions

Remark. If insecticide is not applied, i.e, $\sigma = 0$, we reduce the system (3) to

$$\frac{d\bar{B}}{d\tau} = -\bar{\mu}\bar{B} + \frac{\bar{B}\bar{\omega}^2}{1+\bar{\omega}^2}$$
$$\frac{d\bar{\omega}}{d\tau} = \bar{r}\bar{\omega}\left(1 - \frac{\bar{\omega}}{\bar{k}}\right) - \bar{B}\frac{\bar{\omega}^2}{1+\bar{\omega}^2}.$$

Then we get the following corollaries.

Corollary 2.1

For $\mu > \dfrac{k^2}{1+k^2}$, $E_2(0,k)$ is globally asymptotically stable.

Corollary 2.2

If $\mu > 1/2$ and $k > \dfrac{2\mu^2}{(2\mu-1)\sqrt{\mu-\mu^2}}$, the system has periodic solutions. Moreover, the periodic solution is unique.

Proof. By using Green's theorem, we obtain

$$\oint_{\Gamma} div(g,h)\,dt < 0$$

.

Thus, periodic solution is orbitally stable with asymptotic phase.

Biological considerations

The analysis shows the limited effect of insecticide. Insecticide overuse does not break the periodic orbit and even causes pollution.

- We figured out some conditions for existence of periodic solutions of spruce budworm - bird interation system with the impact of insecticide.
- We are looking for conditions for parameters which yields the uniqueness of limit cycle.
- In the future, budworm-bird-forest interaction model should be considered.

Bibliography

1. Tzy-Wei Hwang. *Global analysis of the predator–prey system with Beddington–DeAngelis functional response*. Journal of mathematical analysis and applications, March 2002.
2. Tzy-Wei Hwang. *Uniqueness of limit cycles of the predator–prey system with Beddington–DeAngelis functional response*. Journal of mathematical analysis and applications, December 2002.
3. Kuo-Shung Cheng. *Uniqueness of a limit cycle for a predator-prey system*. Society for industrial and applied mathematics, 1981.
4. Patricia Arriola,Irene Mijares-Bernal, Juan Ariel Ortiz-Navarro, Roberto A. Saenz. *Dynamics of the Spruce Budworm Population Under the Action of Predation and Insecticides*, 2000.
5. Ludwig, D., D. D. Jones, C. S. Holling, Qualitative analysis of Insect Outbreaks system. *The Spruce budworms and Forrest*. Journal of Animal Ecology, 1978.

6. Xun-Cheng Huang, Stephen J. Merrill. *Conditions for uniqueness of limit cycles in gerenal predator-prey systems*, 1998.
7. Jitsuro Sugie, Rie Kohno, Rinko Miyazaki. *On a predator-prey system of Holling type*, 1997.
8. Sze-Bi Hsu, Tzy-Wei Hwang. *Uniqueness of limit cycles for a predator-prey system of Holling and Leslie type*, 1998.
9. Yang Kuang, H. O. Freedman. *Uniqueness of limit cycles in Gause-Type models of predator-prey systems*, 1987.
10. Zhang Zhifen. *Proof of uniqueness theorem of limit cycles of Generalized Liénard equation*, 1986.
11. S. B. Hsu, S. P. Hubbell, Paul Waltman. *Competing predators*, 1978

Thank you so much for your attention!

基於頻譜與機器學習之聲紋分析及音訊強化

成果報告

林育愷

國立中央大學數學系 學士班四年級

一、 摘要

在理想條件下，將懲罰項加入即時相位修正全變異數去噪模型時，可以看見音訊擁有更佳的去噪表現。然而從懲罰項的設計、能量的表現模式、演算法之收斂性、到參數最佳解良莠與否之判斷方法，各層面皆有許多議題尚待解決。本次計畫除了面對以上困境之外，亦希望能基於以上工作建構完整之加權即時相位修正全變異數去噪模型，並嘗試納入基於機器學習方法之音訊之分類器，以搭建依據音訊種類配給適當處置之適應性音訊去噪工具。

二、 即時相位修正全變異數模型及其跌代演算法

(一) 基於音訊雜訊處理之能量函數建構演進

在西元 1992 年，由 Rudin et al. 提出全變異數去噪模型 (total variation denoising) (式 1)：對於受高斯雜訊干擾之訊號 y，試圖找尋 $x^\star$ 使滿足以下能量最小值條件:

$$x^\star = \arg\min_x \left[\frac{1}{2}\|y-x\|_2^2 + \mathrm{TV}(x)\right], \tag{1}$$

其中

$$\mathrm{TV}(x) = \|\nabla x\|_1 \tag{2}$$

稱爲去噪能量函數中的平滑項。其模型搭配適應之最佳化演算法下，可被廣泛應用於訊號 (一維) 與影像 (二維) 去噪問題 (如圖 7、2 所示) [4, 10, 11]。

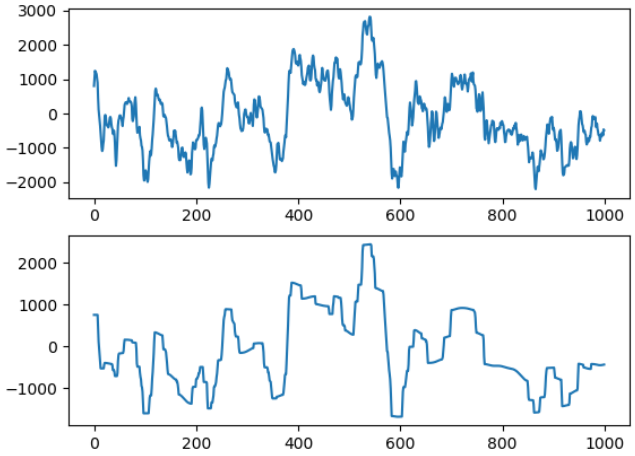


圖 1. 全變異數去噪模型應用於訊號去噪問題 (上：原訊號；下：去噪訊號) [11]

(a) 原始圖片　(b) 加入雜訊的原始圖片 (測資)　(c) 基於測資與去噪模型之迭代解

圖 2. 全變異數去噪模型應用於影像去噪問題 [4]

然而在訊號屬音訊的情境下，作爲音訊品質參考之訊噪比 (SNR) 顯示出音訊品質在能量下降過程中並沒有隨之提昇，反而是逐漸下降 (如圖 3 所示) [7]。

基於修改平滑項 (式 2) 的精神，在 2013 年，兩名科學家，Bayram and Kamasak，提出以對時間在時頻空間上的相位修正頻譜的導數，並取其 1-norm 作爲音訊去噪模型中的平滑項設計 [1]。具體來說，令

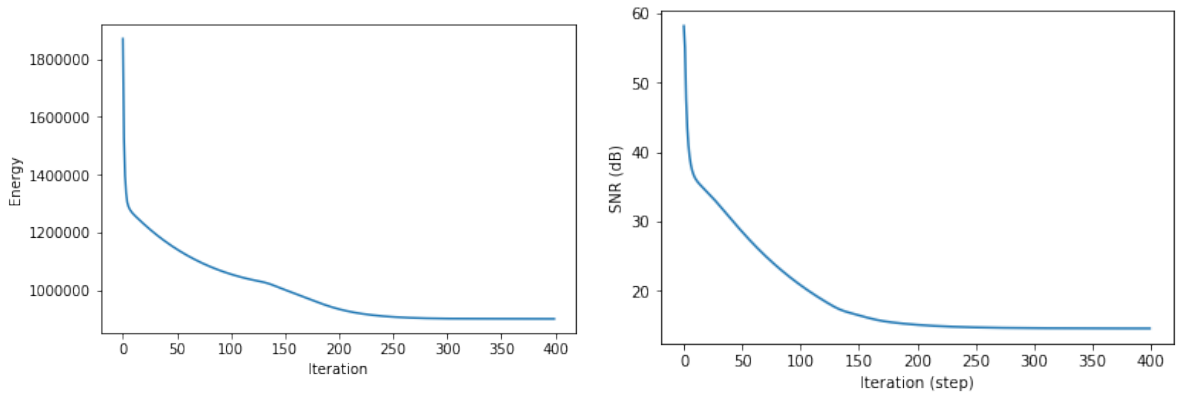
$$(\mathscr{F}^w(y))(m,n) = \sum_{l=0}^{L-1} y(l+an)\overline{w(l)e^{2\pi ibml/L}}, \tag{3}$$

爲基於窗函數 w 之短時距傅立葉轉換 (在實驗中皆以 Hann 函數作爲窗函數的選定)。其中 m 與 n 分別爲頻率與時間的索引數，b 與 a 分別爲頻率與時間的間隔長度。考慮任意固定振幅的正弦音訊在頻譜中的表現，在時間變化上會出現固定的相位差，此現象被稱爲鄰近關係 (neighborhood relation) (式 4)

$$(\mathscr{F}^w s)(f,n+1) \underbrace{e^{-2\pi ifa/L}}_{\text{phase correction}} = (\mathscr{F}^w s)(f,n). \tag{4}$$

針對該頻譜做相位修正算子運算 (式 5)，

$$(E_{\text{PC}}z)(m,n) = z(m,n)e^{-2\pi ibman}. \tag{5}$$



(a) 能量隨迭代過程之表現

(b) 訊噪比隨迭代過程之表現

圖 3. 全變異數在音訊去噪問題之驗證 [7]

(測試訊號：$y(t) = 8\sin(1600\pi t) + \sin(16000\pi t)$，雜訊：$n \sim \mathrm{N}(0, 0.2)$，$\lambda = 200$)

取對時間的偏微分後，最後取 1-norm，取其值並命名爲音訊 y 的相位修正之全變異數 (phase corrected total variation, PCTV) (式 6)：

$$\mathrm{TV}_{\mathrm{PC}}(y) = \|D_t E_{\mathrm{PC}} \mathscr{F}^w x\|_1. \tag{6}$$

而以此平滑項設計之去噪模型

$$x^\star = \arg\min_x \left[\frac{1}{2}\|y - x\|_2^2 + \mathrm{TV}_{\mathrm{PC}}(x)\right] \tag{7}$$

便被稱爲相位修正全變異數去噪模型。

在 2018 年，另二名科學家，Yatabe and Oikawa，將相位修正全變異數 (phase corrected total variation) 與短時距傅立葉轉換之重分配方法 (reassignment method) [5] 結合，發展出即時相位修正全變異數 (instantaneous phase corrected total variation) [12]。細節而言，由於音訊 s 的固定頻率 f 可能與頻譜之中心頻率不一致，導致在頻譜上的表現會出現誤差。重分配方法適用於以短時距傅立葉轉換形成之頻譜，藉由對於頻率項的重新賦值以抑制頻率誤差的展現 (式 8)：

$$s[n] = \exp\left(2\pi i \quad \underbrace{b(m + \delta[m,n])}_{\text{instantaneous frequency}} \quad an/L\right), \tag{8}$$

其中在數值上，

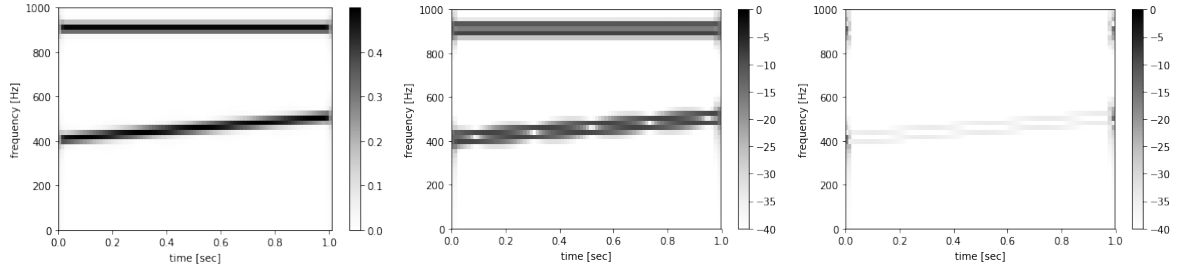
$$\delta[m,n] = -\frac{1}{b}\mathrm{Im}\left[\frac{(\mathscr{F}^{w_t} s)[m,n]}{(\mathscr{F}^w s)[m,n]}\right]. \tag{9}$$

此時重新產生的頻率 $b(m + \delta[m,n])$ 被稱爲即時頻率 (instantaneous frequency)。將此概念與鄰近關係 (neighborhood relation) 結合後，即時相位修正之全變異數 (instantaneous phase corrected total variation, iPCTV) 便呼之欲出 (式 10)：

$$\mathrm{TV}_{\mathrm{iPC}}(x) = \|D_t E_{\mathrm{iPC}} E_{\mathrm{PC}} \mathscr{F}^w x\|_1, \tag{10}$$

其中 $(E_{\mathrm{iPC}} z)(m,n) = z(m,n) e^{-2\pi i b a \tilde{\delta}(m,n)}$，且

$$\tilde{\delta}(m,n) = \sum_{l=0}^{n-1} \frac{\delta(m,l+1) + \delta(m,l)}{2}. \tag{11}$$



(a) 原始訊號在頻譜上之表現　(b) 相位修正之頻譜對時間之導數　(c) 即時相位修正之頻譜對時間之導數

圖 4. 不同處理後頻譜對時間之導數比較 [1, 12]

(測試音訊: $y(t) = \sin(1800\pi t) + \sin((100t + 800)\pi t)$，雜訊: $n \sim \mathrm{N}(0, 0.2)$)

以此平滑項產生之去噪模型

$$x^\star = \arg\min_x \left[\frac{1}{2}\|y - x\|_2^2 + \lambda \cdot \|D_t E_{\mathrm{iPC}} E_{\mathrm{PC}} \mathscr{F}^w x\|_1 \right], \tag{12}$$

便稱爲即時相位修正全變異數去噪模型。

在筆者於去年二月底參加於日本龍谷大學舉辦的台日青年學者應用數學研討會 (The 10th Taiwan-Japan Joint Workshop for Young Scholars in Applied Mathematics) 並於台上演講時，提出一個看法爲：當音訊在頻率表現上只出現在特定區間時，我們可以在頻率上加入懲罰算子，設計概念在於給予較高的懲罰值在頻率區間外；反言之給予較低的懲罰值在頻率區間內。筆者命其名爲加權之即時相位修正全變異數 (weighted instantaneous phase corrected total variation, WiPCTV) [7]。舉例來說，該全變異數可描述如下:

$$x^\star = \arg\min_x \left[\frac{1}{2}\|y - x\|_2^2 + \lambda \cdot \|W D_t E_{\mathrm{iPC}} E_{\mathrm{PC}} \mathscr{F}^w x\|_1 \right], \tag{13}$$

其中懲罰項 W 在較頻繁出現的頻率區間應給予相對低的權重，以增強非尋常頻率區間的雜訊性。在後續章節將討論 W 的選擇流程以及演算法上應注意的事項。

(二)　主-對偶分離演算法之概念與應用方法

在處理相位修正全變異數去噪模型的最小值求解時，可以使用一種迭代演算法，是爲主-對偶分離演算法 (a primal-dual splitting algorithm)，以求去噪能量函數之參數最小化找值 [2, 12]。該方法在建構上給定如下:

$$x^\star = \arg\min_{x \in \mathscr{X}} \left[f(x) + g(x) + h(\Phi x) \right], \tag{14}$$

其中

1. $\mathscr{X}$ 與 $\mathscr{Y}$ 爲二希爾伯特空間 (完備內積空間)。
2. $f : \mathscr{X} \to \mathbb{R}$ 爲一可微分之凸函數，並且存在一非負實數 β 使得 ∇f 爲 β-Lipschitz 函數。
3. $\Gamma_0(\mathscr{H}) := \{f : \mathscr{H} \to \mathbb{R} \cup \{+\infty\} : f \text{ is a proper, lower semicontinuous, convex function}\}$.
4. $g \in \Gamma_0(\mathscr{X})$ 且 $h \in \Gamma_0(\mathscr{Y})$.
5. $\Phi : \mathscr{X} \to \mathscr{Y}$ 爲有界線性算子。

針對該問題，存在一主公式與對偶公式合併的表達式:

$$(x^\star, z^\star) \in \arg\min_{x\in\mathscr{X}} \max_{z\in\mathrm{dom}(h^*)} [f(x) + g(x) - h^*(z) + \langle \Phi x, z\rangle], \tag{15}$$

其中

$$\begin{aligned} \mathrm{dom}(f) &:= \{s \in \ : f(s) < +\infty\}, \quad f \in \Gamma_0(\mathscr{H}), \\ f^* &:= \sup_{s'\in\mathscr{H}} [\langle s, s'\rangle - f(s')], \quad f \in \Gamma_0(\mathscr{H}). \end{aligned} \tag{16}$$

並存在一迭代過程以計算 $x^\star$ 與 $z^\star$ 的數值解:

$$\begin{aligned} x^{[n+1]} &= \mathrm{prox}_{\sigma_1 g}[x^{[n]} - \sigma_1(\nabla f(x^{[n]}) + \Phi^* z^{[n]})], \\ z^{[n+1]} &= \mathrm{prox}_{\sigma_2 h^*}[z^{[n]} + \sigma_2 \Phi(2x^{[n+1]} - x^{[n]})]. \end{aligned} \tag{17}$$

其中 $\mathrm{prox}_{\sigma f}[z] = \arg\min_x \left[f(x) + \frac{1}{2\lambda}\|z - x\|_2^2\right]$. 該演算法中存在兩個參數 σ_1 與 σ_2，此組參數須滿足以下不等式以保持解的收斂性 [2]:

$$\frac{1}{\sigma_1} - \sigma_2 \|\Phi\|_{\mathrm{op}}^2 \geq \frac{1}{2}. \tag{18}$$

應用於去噪模型，PCTV、iPCTV、與 WiPCTV，時，設定 $f = \frac{1}{2}\|y - x\|_2^2$、$g(x) = 0$、$h(x) = \lambda\|x\|_1$、且 $h(\Phi x)$ 爲模型給定之平滑項; 意即，Φx 爲不取範數之平滑項值。便可嘗試套用迭代方法 (式 17) 並求解之 [7, 12]。

(三) 收斂條件中之範數估計

以 iPCTV 模型而言，在式 18 中，由於算子的空間對應爲 $\mathbb{R}^n$ 與 $\mathbb{R}^{n\times(m-1)}$，根據定義之算子範數，其值應由下式決定:

$$\begin{aligned} \|\Phi\|_{\mathrm{op}} &= \max_{x\neq 0\in\mathbb{R}^n} \frac{\|D_t E_{\mathrm{iPC}} E_{\mathrm{PC}} \mathscr{F}^w x\|_2}{\|x\|_2} \\ &= \max_{x\neq 0\in\mathbb{R}^n} \max_{y\neq 0\in\mathbb{C}^{(m-1)}} \frac{\|(D_t E_{\mathrm{iPC}} E_{\mathrm{PC}} \mathscr{F}^w x) y\|_2}{\|x\|_2 \cdot \|y\|_2}. \end{aligned} \tag{19}$$

然而直接計算式 19 不是一件容易的事情，因此我們採用估計上界的方式去決定跌代演算法中參數的給定標準: 不考慮短時距傅立葉轉換的交疊 (overlap) 下，令 $\mathbb{V} = \mathbb{R}^n$，則 Φ 內各算子映射關係爲

$$\mathscr{F}^w : (\mathbb{R}^n, \ldots, \mathbb{R}^n) \to (\mathbb{V}, \ldots, \mathbb{V}) \qquad (m-\text{piecewise}), \tag{20}$$

$$E_{\mathrm{PC}} : \mathbb{V}^m \to \mathbb{V}^m, \tag{21}$$

$$E_{\mathrm{iPC}} : \mathbb{V}^m \to \mathbb{V}^m, \tag{22}$$

$$D_t : \mathbb{V}^m \to \mathbb{V}^{m-1}. \tag{23}$$

這樣的宣告對以下估計來說是有幫助的，在於其利用了對差分矩陣 D_t 與傅立葉矩陣 (Fourier Matrix) 最大奇異值 (maximum singular value) 的理解:

$$\begin{aligned} \|\Phi\|_{\mathrm{op}} &\leq \|D_t\|_{\mathrm{op}} \|E_{\mathrm{iPC}} E_{\mathrm{PC}} \mathscr{F}^w\|_{\mathrm{op}} \\ &\leq \|D_t\|_{\mathrm{op}} \|E_{\mathrm{iPC}}\|_{\mathrm{op}} \cdot \|E_{\mathrm{PC}}\|_{\mathrm{op}} \cdot \|\mathscr{F}^w\|_{\mathrm{op}} \\ &\leq \|D_t\|_{\mathrm{op}} \sup_{i,j} |E_{\mathrm{iPC}}(i,j)| \cdot \sup_{i,j} |E_{\mathrm{PC}}(i,j)| \cdot \|\mathscr{F}^w\|_{\mathrm{op}} \\ &\leq 4 \cdot 1 \cdot 1 \cdot \|M_F\|_2 \\ &\leq 4 \cdot \sigma_{\max}(M_F). \end{aligned} \tag{24}$$

其中方陣 M_F 爲 n 階傅立葉矩陣 (Fourire Matrix).

三、 輕量級卷積神經網路應用於音訊分類器

(一) 深度可分離卷積網路

深度可分離卷積 (depthwise separable convolution) [6] 是卷積網路的一種輕量級形式，它將標準卷積層分解爲深度卷積和 1×1 卷積 (pointwise convolution, weight factor)。留意深度可分離卷積和傳統卷積層在函數觀點上是不一樣的：深度可分離卷積可視爲傳統卷積層的特殊情況，而這種分解也可以在不降低精度的情況下加速大量計算，這將在接續內容中進行更多討論。

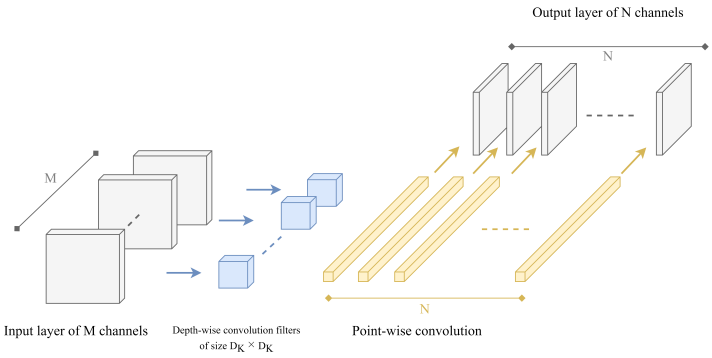


圖 5. 深度可分離卷積結構

回顧一下傳統卷積層的計算複雜度，對於輸入 $D_F \times D_F \times M$ 維度之特徵層 F 並生成 $D_G \times D_G \times N$ 維度之特徵層 G，其中 D_F 是輸入層 (input layer) 的空間寬度與高度、M 是輸入通道 (input channel) 數，D_G 是輸出層 (output layer) 的空間寬度和高度，N 是輸出通道 (output channel) 數，標準卷積層的參數是卷積核 (convolution kernel) K 的大小爲 $D_K \times D_K \times M \times N$，其中 D_K 是給定之濾波器 (kernel) 的維度。考慮跨度 (stride) 爲 1 的條件下，此方法的輸出特徵圖的計算方式爲

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} \cdot F_{k+i-1,l+j-1,m}, \tag{25}$$

呼應其計算複雜度爲

$$D_K \times D_K \times M \times N \times D_F \times D_F. \tag{26}$$

而深度可分離卷積由兩層組成：深度卷積 (depthwise convolution) 和逐點卷積 (pointwise convolution)：深度卷積爲對於每個輸入通道套用單個濾波器，而逐點卷積則產生一組 1×1 濾波器，用於創造出對應深度層輸出通道數的線性組合。精確來說，對於輸入層數 M 之深度卷積之計算方式爲

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot F_{k+i-1,l+j-1}, \tag{27}$$

其中 $\hat{K}$ 爲 $D_K \times D_K \times M$ 維度之濾波器，在 m 引數下輸入通道與濾波器一一對應。註記逐點卷積等價於應用 1×1 濾波器之標準卷積，因此綜合來說深度可分離卷積的計算複雜度爲

$$D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F, \tag{28}$$

且不考慮偏移向量 (bias vector) 下，其參數維度爲 $D_K \times D_K \times M + M \times N$.

	Standard convolution	Separable convolution	Rate (separable / standard)
Param #	$D_K^2 MN$	$D_K^2 M + MN$	$\frac{1}{N} + \frac{1}{D_K^2}$
Cost	$D_K^2 MND_F^2$	$D_K^2 MD_F^2 + MND_F^2$	$\frac{1}{N} + \frac{1}{D_K^2}$

表 1: 深度可分離卷積與傳統卷積的比較 [6]

(二) 深度可分離卷積應用於 YAMNet 音訊分類器

在 2019 年間，Manoj Plakal and Dan Ellis 發表了針對音訊資料集 AudioSet [3] 之分類器 YAMNet [8]。YAMNet 的運作流程分成特徵擷取與神經網路兩部分。特徵擷取部分有以下步驟:

1. 將音訊重採樣 (resamping) 爲頻率 16 kHz 之音訊
2. 透過短時距傅立葉轉換計算音訊之時頻譜 (spectrum) 並取其量鋼 (magnitude)
3. 將線性頻率時頻譜轉換成梅爾倒頻譜 (Mel-Frequency Cpectrum) 並取對數
4. 分割整段音訊爲正規且批次化資料

計算出來的數個特徵將會批次作爲 MobileNet [6] 的輸入層進行演算。MobileNet 結構是基於深度可分離卷積 (depthwise separable convolution) 構建的，除了第一層是標準卷積，最後依序爲池化層 (pooling layer)、全連結層 (full-connected layer) 和分類 (classification)。結構陳述中，以單次深度卷積和單次逐點卷積爲單位，並在每單位運算後進行批次正規化 (batch normalization) 和 ReLU 正規化 (見表 2)。由於每次單位運算中之 D_K 皆爲 3，此意味著 MobileNet 的計算效率約爲標準卷積神經網路的八到九倍之間。

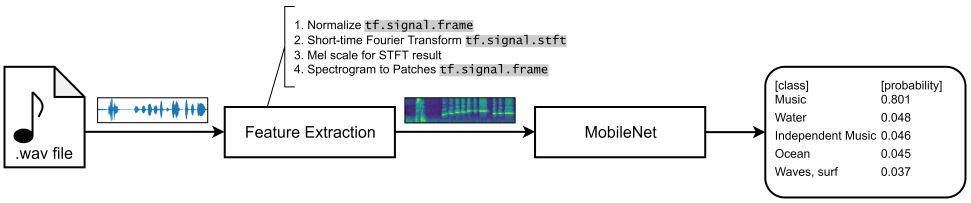


圖 6. YAMNet 音訊分類流程

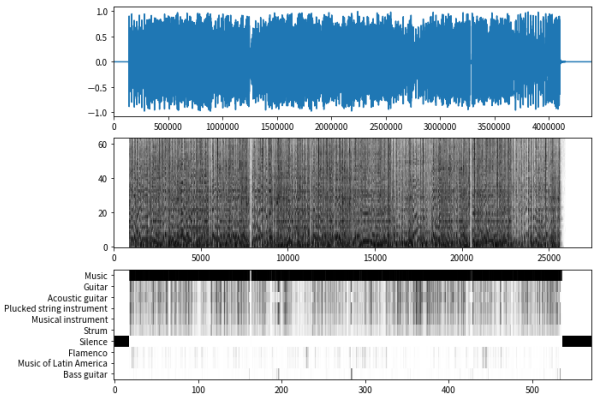


圖 7. YAMNet 音訊分類範例: 在圖中分成三個列，第一列爲音訊原始資料、第二列爲特徵擷取後之數值、第三列則爲 YAMNet 預測之音訊類別的機率視覺圖 (顏色深淺代表機率高低) [8]。

表 2: YAMNet 與 MobileNet 網路架構 [6, 8]

Model		Type / Stride	Filter Shape	Input Size
		(Feature extraction)		
MobileNet	5 times	Conv / s2	$3 \times 3 \times 3 \times 32$	$96 \times 64 \times 1$
		Conv dw / s1	$3 \times 3 \times 32$ dw	$48 \times 32 \times 32$
		Conv / s1	$1 \times 1 \times 32 \times 64$	$48 \times 32 \times 32$
		Conv dw / s2	$3 \times 3 \times 64$ dw	$48 \times 32 \times 64$
		Conv / s1	$1 \times 1 \times 64 \times 128$	$24 \times 16 \times 64$
		Conv dw / s1	$3 \times 3 \times 128$ dw	$24 \times 16 \times 128$
		Conv / s1	$1 \times 1 \times 128 \times 128$	$24 \times 16 \times 128$
		Conv dw / s2	$3 \times 3 \times 128$ dw	$24 \times 16 \times 128$
		Conv / s1	$1 \times 1 \times 128 \times 256$	$12 \times 8 \times 128$
		Conv dw / s1	$3 \times 3 \times 256$ dw	$12 \times 8 \times 256$
		Conv / s1	$1 \times 1 \times 256 \times 256$	$12 \times 8 \times 256$
		Conv dw / s2	$3 \times 3 \times 256$ dw	$12 \times 8 \times 256$
		Conv / s1	$1 \times 1 \times 256 \times 512$	$6 \times 4 \times 512$
		Conv dw / s1	$3 \times 3 \times 512$ dw	$6 \times 4 \times 512$
		Conv / s1	$1 \times 1 \times 512$ dw	$6 \times 4 \times 512$
		Conv dw / s2	$3 \times 3 \times 512$ dw	$6 \times 4 \times 512$
		Conv / s1	$1 \times 1 \times 512 \times 1024$	$3 \times 2 \times 512$
		Conv dw / s2	$3 \times 3 \times 1024$ dw	$3 \times 2 \times 1024$
		Conv / s1	$1 \times 1 \times 1024 \times 1024$	$3 \times 2 \times 1024$
		Avg Pool / s1	Pool 3×2	$3 \times 2 \times 1024$
		FC / s1	1024×521	$1 \times 1 \times 1024$
		Softmax / s1	Classifier	$1 \times 1 \times 521$

四、 自適應性音訊強化工具

(一) 結合音訊分類器與即時相位修正全變異數之音訊強化工具

此適應性音訊去雜訊工具結合了「音訊分類器」與「加權即時相位修正全變異數去噪」兩項工具 (流程見圖 8)：透過前者分類出輸入音訊之類型後，在後者去噪模型上引用對應的權重以加強非主流頻率段的雜訊合理性。本文以演說 (speech) 類型音訊爲例，藉由在時頻空間對頻率的平均量鋼 (magnitude) 得到機率密度分布函數 p 後，並施以相對應的權重值:

$$f = 1 - \mu p, \tag{29}$$

其中 $\mu \in [0, 1]$ 爲權重的突顯程度，且

$$(Wz)(m, n) = z(m, n) f(m). \tag{30}$$

留意到式 29 之值域爲閉區間 $[0,1]$，此特性可避免跌代演算法上的收斂問題。精確來說，從式 24 可推知:

$$\begin{aligned}\left\|WD_tE_{\mathrm{iPC}}E_{\mathrm{PC}}\mathscr{F}^w x\right\|_{\mathrm{op}} &\leq \|W\|_{\mathrm{op}}\left\|D_tE_{\mathrm{iPC}}E_{\mathrm{PC}}\mathscr{F}^w\right\|_{\mathrm{op}} \\ &\leq \sup_i |f(i)|\cdot 4\cdot\sigma_{\max}(W) \qquad \leq 4\cdot\sigma_{\max}(W).\end{aligned}$$

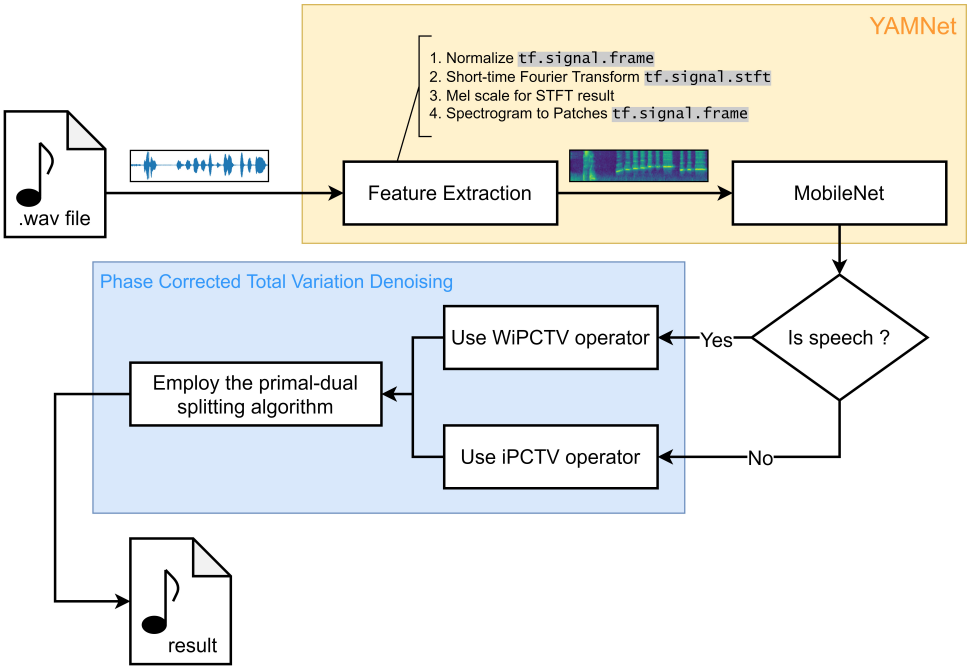
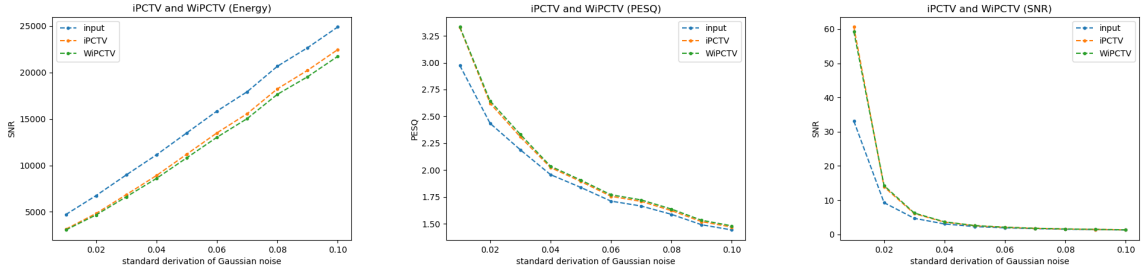


圖 8. 自適應音訊去噪演算流程

(二) 去噪音訊品質判定與結果分析

由於語音類型音訊的特性，除一般訊號常用之訊噪比 (Signal-Noise Ratio, SNR) 之外，亦可使用 PESQ (Perceptual Evaluation of Speech Quality) [9] 作爲判定去噪結果良莠的依據之一，另外我們也透過式 10 去判斷不同方法上的效果比較。對於眞實音訊給定 $\sigma_1 = 0.00039$ 與 $\sigma_2 = 0.001$，跌代 6000 次此固定條件下，從圖 9 與表 3 中，我們發現在不同雜訊程度的語音類音訊，自適應式音訊強化工具 (WiPCTV) 在式 10 與 PESQ 上的表現皆略優於 iPCTV 的結果，而在訊噪比的表現則多數略優於 iPCTV 的結果。



(a) 能量函數比較 (b) PESQ 比較 (c) SNR 比較

圖 9. WiPCTV ($\mu = 0.5$) 與 iPCTV 應用於眞實音訊的效果比較

表 3: WiPCTV ($\mu = 0.5$) 與 iPCTV 應用於眞實音訊的效果比較

	能量函數 (式 10)			PESQ			SNR		
SD	Input	iPCTV	WiPCTV	Input	iPCTV	WiPCTV	Input	iPCTV	WiPCTV
0.01	4.712e+03	3.139e+03	**3.053e+03**	2.974	3.327	**3.332**	33.11	**60.63**	59.31
0.02	6.75e+03	4.824e+03	**4.67e+03**	2.436	2.623	**2.644**	9.269	13.88	**14.25**
0.03	8.974e+03	6.849e+03	**6.616e+03**	2.19	2.309	**2.333**	4.689	6.075	**6.243**
0.04	1.113e+04	8.907e+03	**8.598e+03**	1.958	2.024	**2.036**	3.049	3.585	**3.665**
0.05	1.349e+04	1.12e+04	**1.081e+04**	1.839	1.895	**1.908**	2.274	2.529	**2.575**
0.06	1.584e+04	1.349e+04	**1.302e+04**	1.712	1.757	**1.771**	1.911	2.058	**2.088**
0.07	1.792e+04	1.556e+04	**1.503e+04**	1.666	1.709	**1.723**	1.666	1.755	**1.775**
0.08	2.066e+04	1.823e+04	**1.764e+04**	1.589	1.624	**1.638**	1.511	1.569	**1.583**
0.09	2.265e+04	2.022e+04	**1.953e+04**	1.492	1.521	**1.534**	1.407	1.447	**1.458**
0.1	2.489e+04	2.246e+04	**2.173e+04**	1.443	1.47	**1.482**	1.292	1.316	**1.323**

五、 結論與未來工作

(一) 結論

自適應式音訊強化工具結合了頻譜處理、全變異數模型的變體、最佳化方法的使用，並結合機器學習之神經網路音訊分類器，以作爲自調配權重以優化音訊去噪演算。本文以語音類型音訊爲例，使用上述方法設計之權重進行演算，除確保其收斂性外，亦透過能量函數 (式 10)、PESQ、與 SNR 三種指標驗證其有效性。

(二) 未來工作

自適應式方法尚有部分細項可更加深入探討，除可探討音訊類型以外的去噪權重設計外，也可尋找在給定之明確收斂下，賦予適切之跌代參數使得整體演算法有更佳的收斂速率。另外基於 YAMNet 架構的特性，亦可探索單一音訊中，分段分類權重之適應性方法。

參考文獻

[1] I. Bayram and M. E. Kamasak. A simple prior for audio signals. *IEEE Trans. Audio, Speech, Language Process*, 21(6):1190–1200, 2013.

[2] Laurent Condat. A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms. *Journal of Optimization Theory and Applications*, 158, 08 2013. doi: 10.1007/s10957-012-0245-9.

[3] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 776–780, March 2017. doi: 10.1109/ICASSP.2017.7952261.

[4] Pascal Getreuer. Rudin-osher-fatemi total variation denoising using split bregman. *Image Processing On Line*, 2012.

[5] N. Holighaus, Z. Průša, and P. L. Søndergaard. Reassigment and synchrosqueezing for general time-frequency filter banks, subsampling and processing. *Signal Process*, 125:1–8, 2016.

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv e-prints*, art. arXiv:1704.04861, Apr 2017.

[7] Yu-Kai Lin. Optimization problems in signal and audio denoising. In *10th Japan-Taiwan Joint Workshop For Young Scholars in Applied Mathematics*, 2019.

[8] Manoj Plakal and Dan Ellis. YAMNet. `https://github.com/tensorflow/models`, 2019.

[9] A. W. Rix, J. G. Beerends, M. P. Hollier, and A. P. Hekstra. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 2, pages 749–752 vol.2, May 2001. doi: 10.1109/ICASSP.2001.941023.

[10] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60:259–268, 1992.

[11] I.W. Selesnick and I. Bayram. Total variation filtering. *Technical Report*, 2010.

[12] K. Yatabe and Y. Oikawa. Phase corrected total variation for audio signals. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018-April: 656–660, 2018. doi: 10.1109/ICASSP.2018.8461541.

Vessels detection

ME4B 林彥成

Vessels detection

- Binarize
- Frangi filter

Binarize

- Otsu's Method

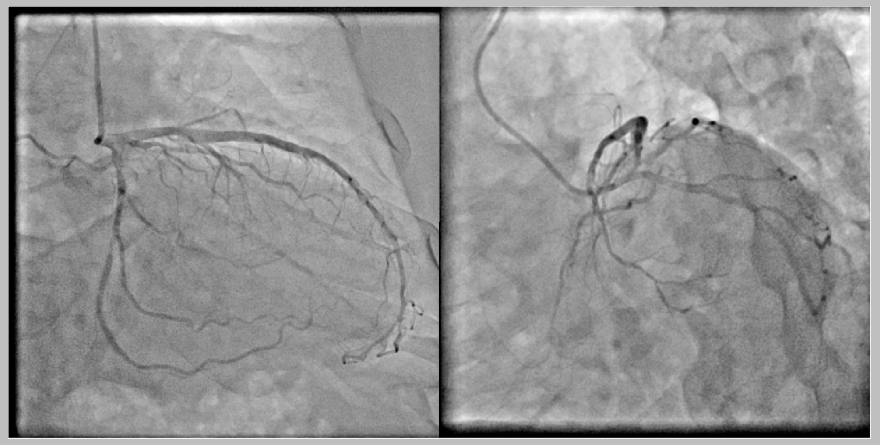
A single intensity threshold that separate pixels into two classes, foreground and background.

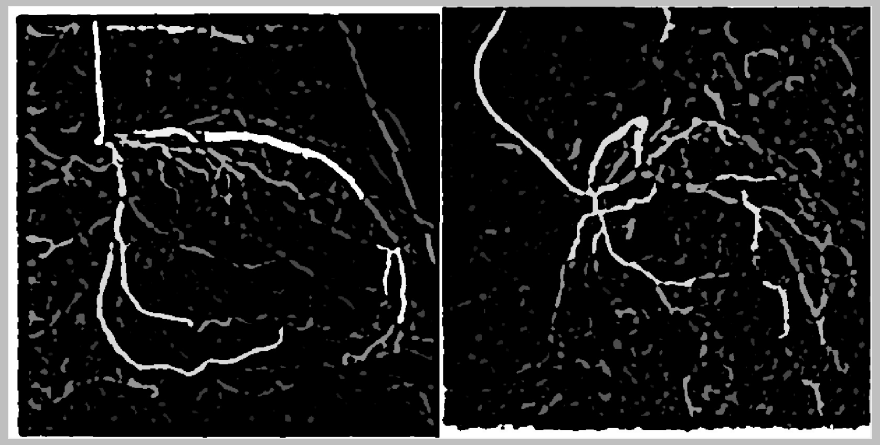
This threshold is determined by minimizing intra-class intensity variance, or equivalently, by maximizing inter-class variance.

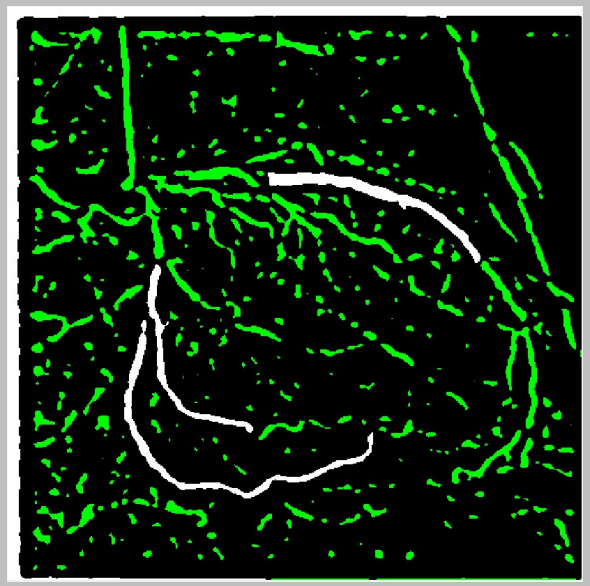
Frangi filter

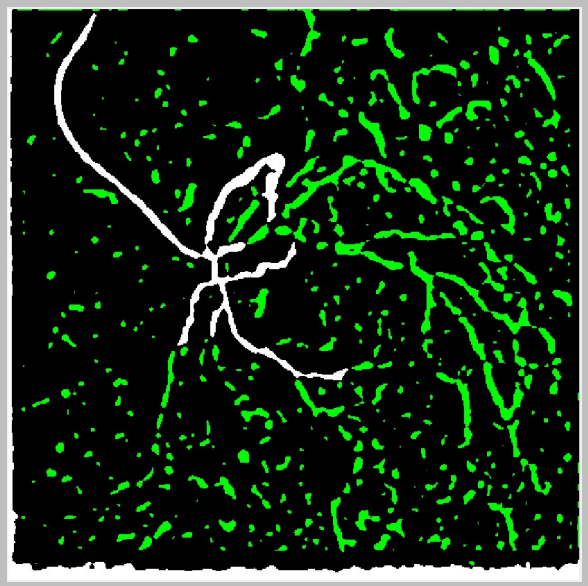
- The Frangi filter is typically used to detect vessel-like or tube-like structures and fibers in volumetric image data.
- Multiscale vessel enhancement filtering.

Result









Problem

The region of vessels is discontinuous.

Possible Reason

- parameter setting
- Problems with the image itself
  - X-ray image

Reference

- 199809_Multiscale vessel enhancement filtering
- 201000_Optimized Anisotropic Rotational Invariant Diffusion Scheme on Cone-Beam CT
- 201306_Optimized Coronary Artery Segmentation using Frangi filter and Anisotropic Diffusion Filtering

python 環境安裝 (使用 anaconda)

- Windows 環境
  - 使用 anaconda 整合包 下載頁面
    - 使用 python 3.7 版本
    - 【注意】安裝時會詢問是否 **將 anaconda 加入環境變數** 要打勾
      - 沒有打勾會不能在 power shell (命令提示字元)使用相關的指令，如 pip, conda 等
      - 可能會蓋過舊安裝的 python
- Linux 環境
  - 使用 anaconda 整合包 下載頁面
    - 使用 python 3.7 版本
    - 待補

1. 開始使用文字介面操作

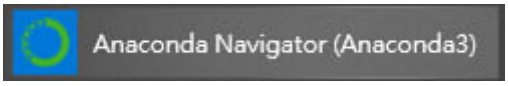
- 為什麼不使用圖形介面(GUI[1])，而要使用文字介面(CLI[2])？

雖然 anaconda 有提供圖形介面 `anaconda navigator`[3] 來操作，但有時候會需要使用文字介面來操作，尤其是跑大量計算的程式 (ex. 神經網路) 就常常需要連線到遠端伺服器來操作，這時候通常就沒有圖形介面；所以接下來會介紹使用文字介面的操作方式。

在 Windows 下命令提示字元主流有兩種，一是 `cmd`，一是 `power shell` (新版的 `cmd`，支援更多指令，更接近 linux 系統)，以下會以 `power shell` 的來介紹。

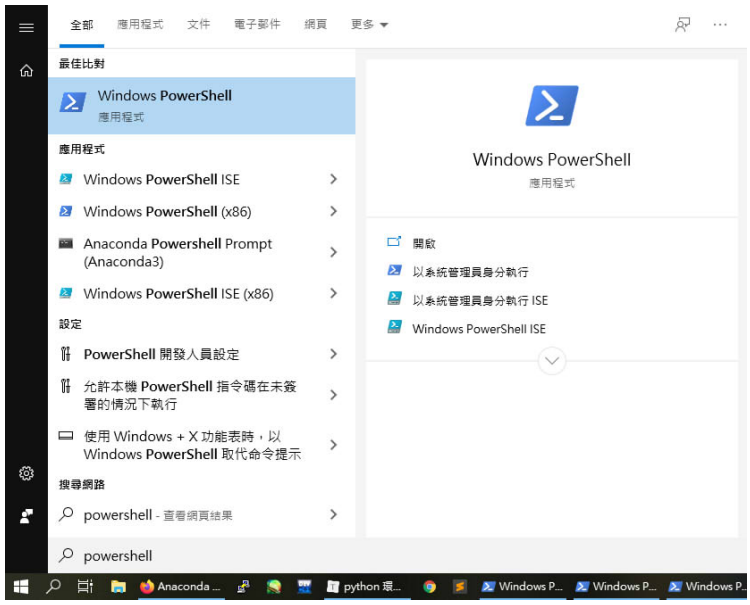
[1] GUI: *Graphical User Interface* 使用者圖形介面

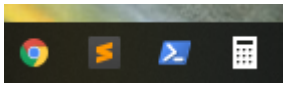
[2] CLI: *Command-Line Interface* 命令列介面

[3] `anaconda navigator`**: 可以在"開始"裡面找到** 

1.1. 打開文字介面的方式

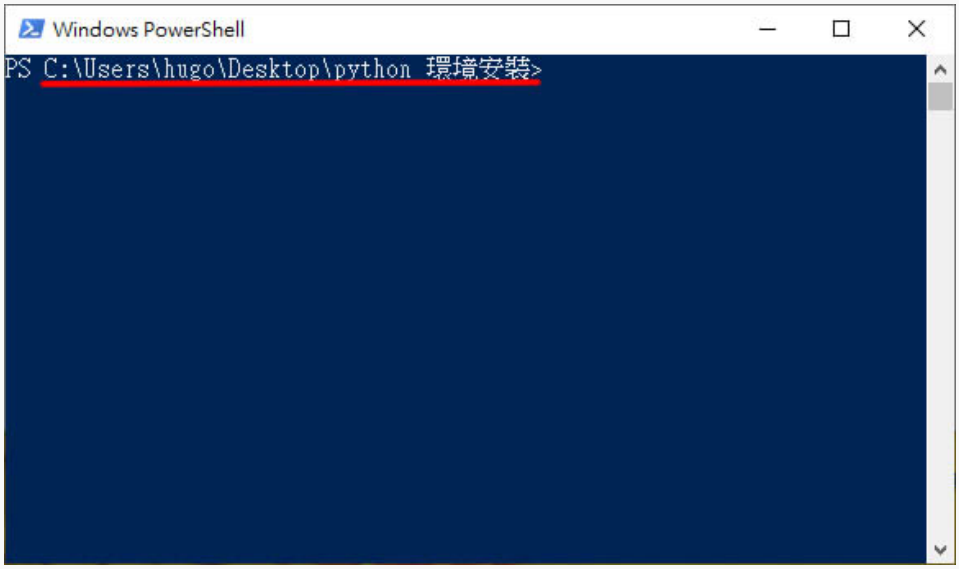
- Windows

按住 shift，對畫面空白的地方按右鍵。 **(一般的右鍵會跳出新增資料夾、文件的選單)**	
在開始工具列搜尋 power shell	

- 兩者的差別在於**路徑不同**，一個隨開啟的位置改變，一個固定在家目錄。
- 可以把固定在工具列上 

- Linux 則是叫出終端機的指令 `ctrl`+`alt`+`t`

1.2. 文字介面的樣式

Windows	
Linux	

`power shell` 會顯示路徑 `C:\Users\hugo\Desktop\python 環境安裝`

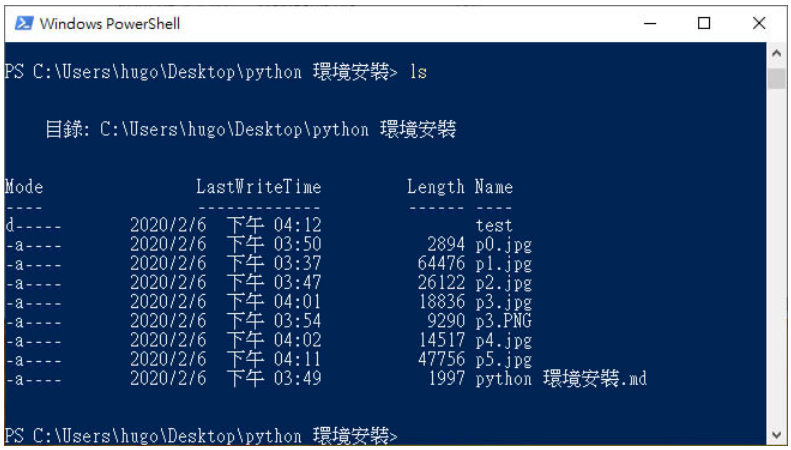
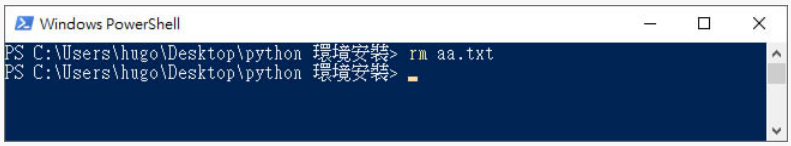
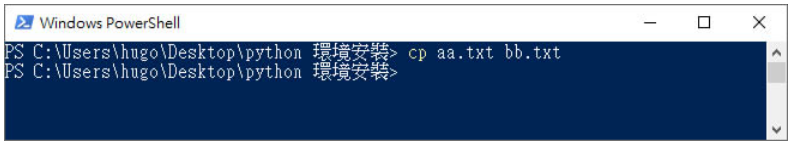
`bash` 會顯示使用者 `hugo` 、主機名 `math` 、以及路徑 `~/imweb`

1.3. 基本指令

打開文字介面後，就可以開始輸入指令，常用的指令有下列五個：

指令名	全稱	作用
`ls`	list	列出資料夾下所有檔案
`cd`	change directory	切換資料夾
`mv`	move	移動檔案，也可用作改名
`rm`	remove	刪除檔案
`cp`	copy	複製檔案
`mkdir`	make directory	建立資料夾

另外也有 秀出檔案內容的 `cat`, 改變權限的 `chmod` 等等，也有額外安裝文字編輯器才會有的 `vim` 等等。

指令	圖解
`ls` 注意 Mode 那邊第一位是 d 的，就是資料夾	
`cd`	
`cd ..` 往上一層	
`rm` 檔案存在就會刪除	
`cp` 複製一份 `aa.txt` 叫做 `bb.txt`	
`mkdir` 建立一個資料夾叫 `c`	

2. 使用 anaconda 的虛擬環境

我們可以使用 anaconda 來建立一個一個的**虛擬環境**，之後再將需要的套件安裝在這個**虛擬環境**內，就可以避免不同專案安裝的套件產生衝突。

使用這個**虛擬環境**，你就可以在**虛擬環境A**裝 `tensorflow` 1.1版，在**虛擬環境B**裝 `tensorflow` 1.2版，而不會產生衝突。

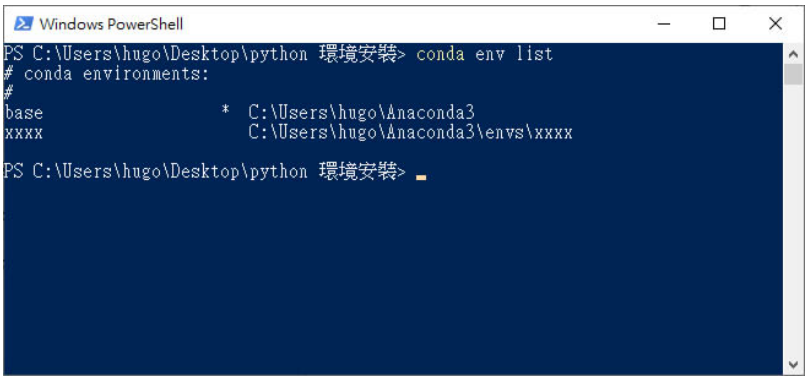
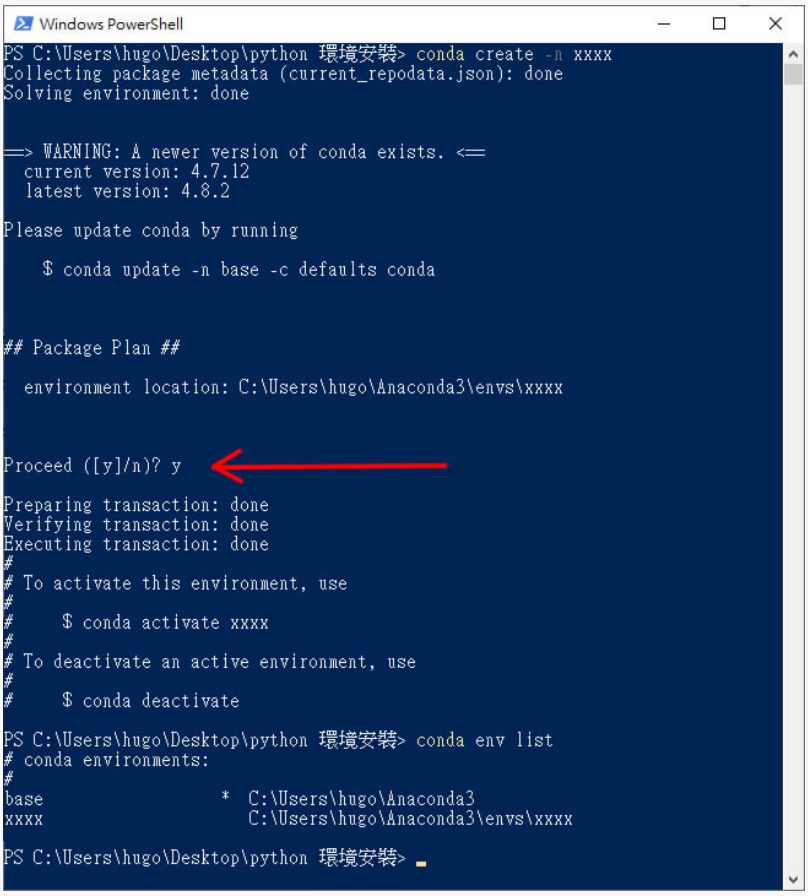
了解如何操作文字介面後，現在開始使用 anaconda 的虛擬環境功能。

通常操作如下：

時間點	開發流程
開始新專案	建立環境 > 進入環境 > 寫程式 ...
直到專案完成前	進入環境 > 寫程式 ...
(加入新套件)	進入環境 > 安裝新套件 > 匯出一份環境設定做紀錄
(專案和別人分工)	進入環境 > 匯出一份環境設定給別人
專案完成	(進入環境 > 匯出一份環境設定做紀錄 >) 刪除環境

2.1 建立/刪除/列出環境

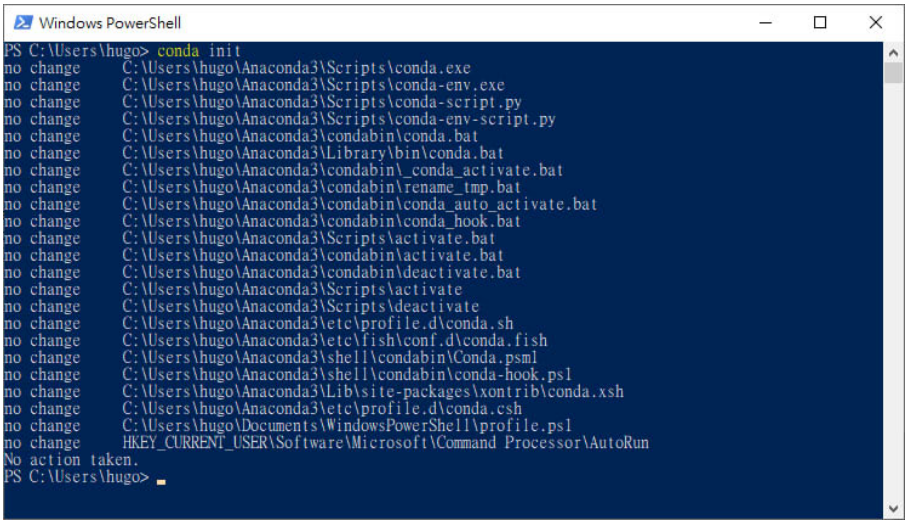
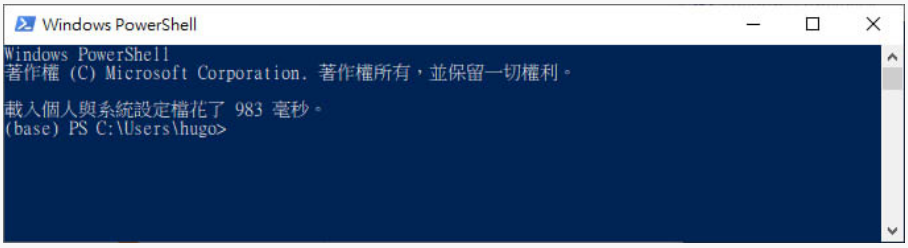
指令	作用
`conda env list`	列出所有建立的環境
`conda create -n xxxx`	建立名為 `xxxx` 的環境
`conda env remove -n xxxx`	刪除名為 `xxxx` 的環境

指令	圖解
`conda env list` 會看到目前所屬的環境 (打 * 的)	
`conda create -n xxxx` 中間有一步需要輸入 `y` 確認	
`conda env remove -n xxxx`	

2.2.0 設定 power shell (power shell 限定)

有些 `conda` 指令在 `power shell` 上會不能正常作用 (ex. `conda activate`)，所以需要額外處理。

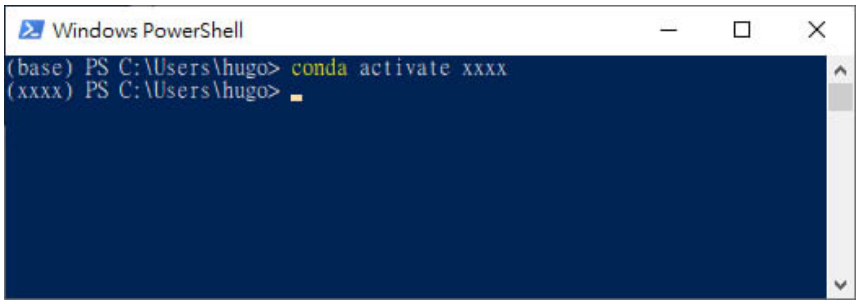
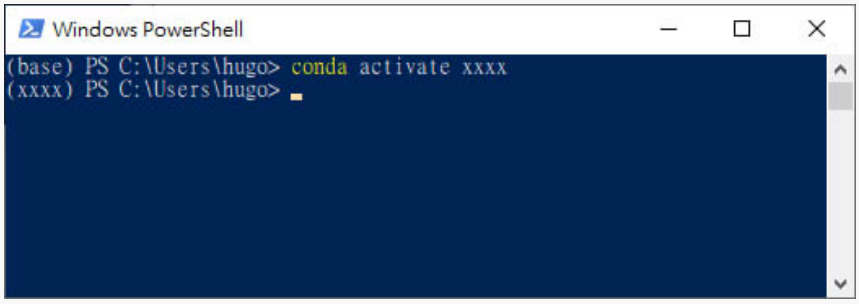
指令	作用
`conda init`	讓 `conda` 指令可以正確作用在 `powershell` 上
`conda config --set auto_activate_base false`	不要一打開 `power shell` 就進入 `base` 環境
`conda config --set auto_activate_base true`	一打開 `power shell` 就進入 `base` 環境

指令	圖解
`conda init`	
`conda init` (重開 power shell) (前面多了 base)	

2.2 進入/退出環境

在輸入指令的最前方小括號的部分，就是目前所在的環境的名稱。

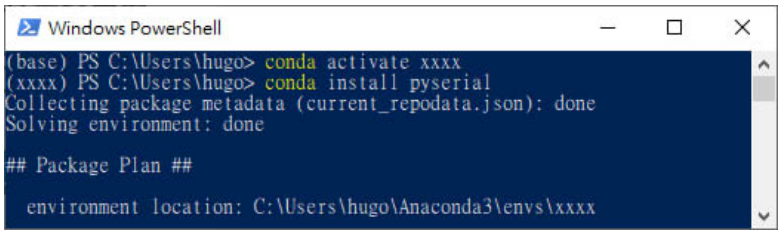
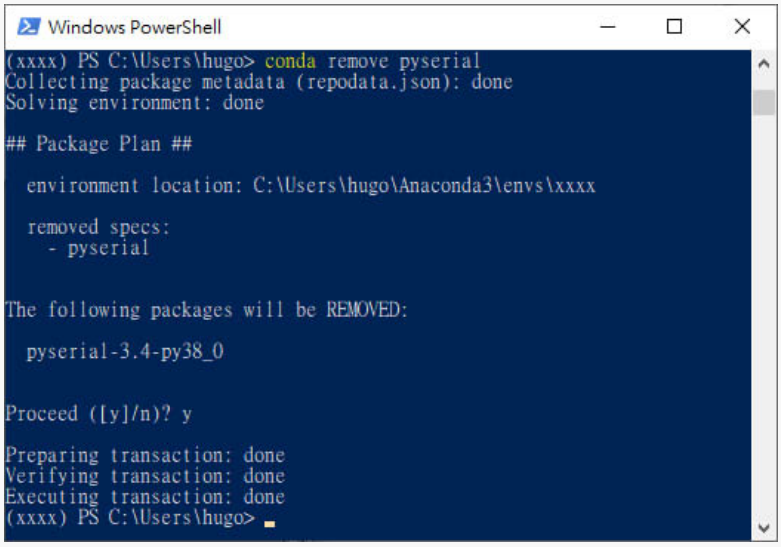
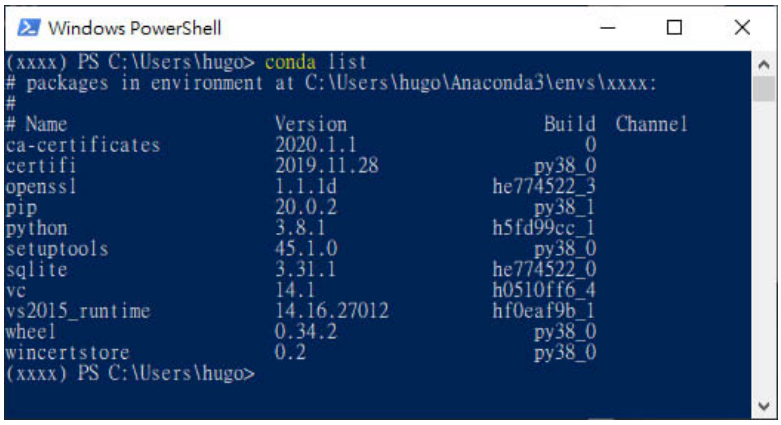
指令	作用
`conda activate xxxx`	進入 `xxxx` 環境
`conda deactivate`	離開目前的環境

指令	圖解
`conda activate xxxx`	
`conda deactivate`	

2.3 列出/安裝/移除套件

【注意】安裝套件前要確認目前所在的環境

指令	作用
`conda install`	安裝套件
`conda remove`	移除套件
`conda list`	列出已安裝的套件

指令	圖解
`conda install pyserial`	
`conda remove pyserial`	
`conda list`	

2.4 匯出/使用環境設定檔

當環境安裝好必要的套件之後，可以匯出環境設定檔。

【注意】匯出的設定檔是依據當前所在的環境

指令	作用
`conda env export > req.yml`	匯出環境安裝的套件，檔名為 `req.yml`
`conda env create -f req.yml`	使用 `req.yml` 的設定來建立環境

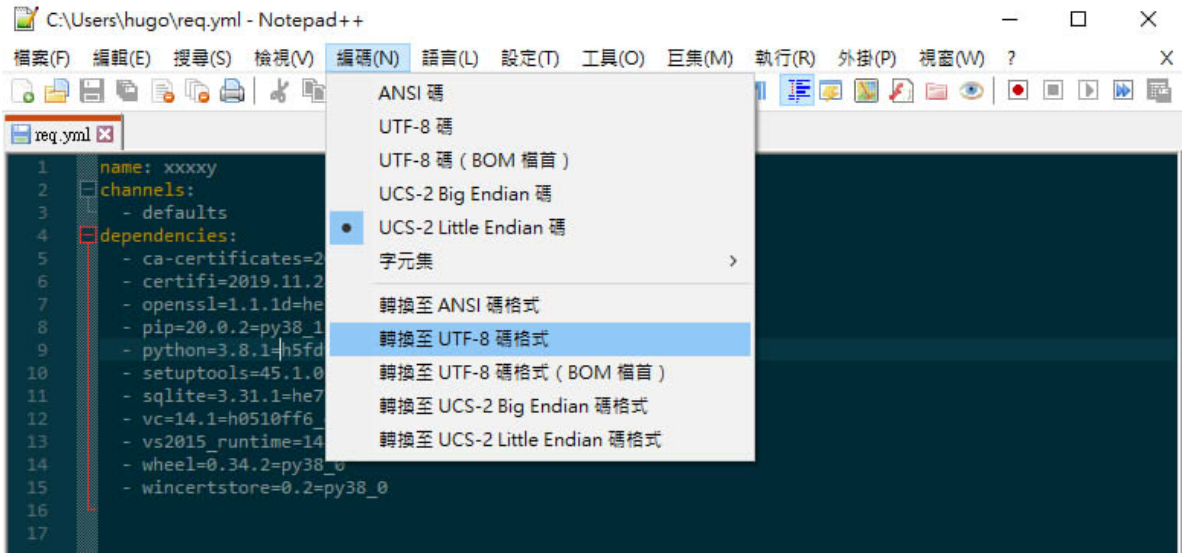
`req.yml` 檔案如下

```
name: xxxx
channels:
```

```
  - defaults
dependencies:
  - ca-certificates=2020.1.1=0
  - certifi=2019.11.28=py38_0
  - openssl=1.1.1d=he774522_3
  - pip=20.0.2=py38_1
  - python=3.8.1=h5fd99cc_1
  - setuptools=45.1.0=py38_0
  - sqlite=3.31.1=he774522_0
  - vc=14.1=h0510ff6_4
  - vs2015_runtime=14.16.27012=hf0eaf9b_1
  - wheel=0.34.2=py38_0
  - wincertstore=0.2=py38_0
prefix: C:\Users\hugo\Anaconda3\envs\xxxx
```

【注意】要去掉 `prefix` 那行後再使用

【注意】有時候會產生編碼問題 (ex. cp950)，可以下載 `notepad++` ，使用他的`轉換至 UTF-8 碼格式`



3. 使用 jupyter notebook 來寫 python

因為 `anaconda` 內建 `jupyter notebook`，同時也有人寫了套件整合 `jupyter` 和 `conda` 的虛擬環境，所以這裡建議大家使用使用 `jupyter notebook` 來編寫 `python`。

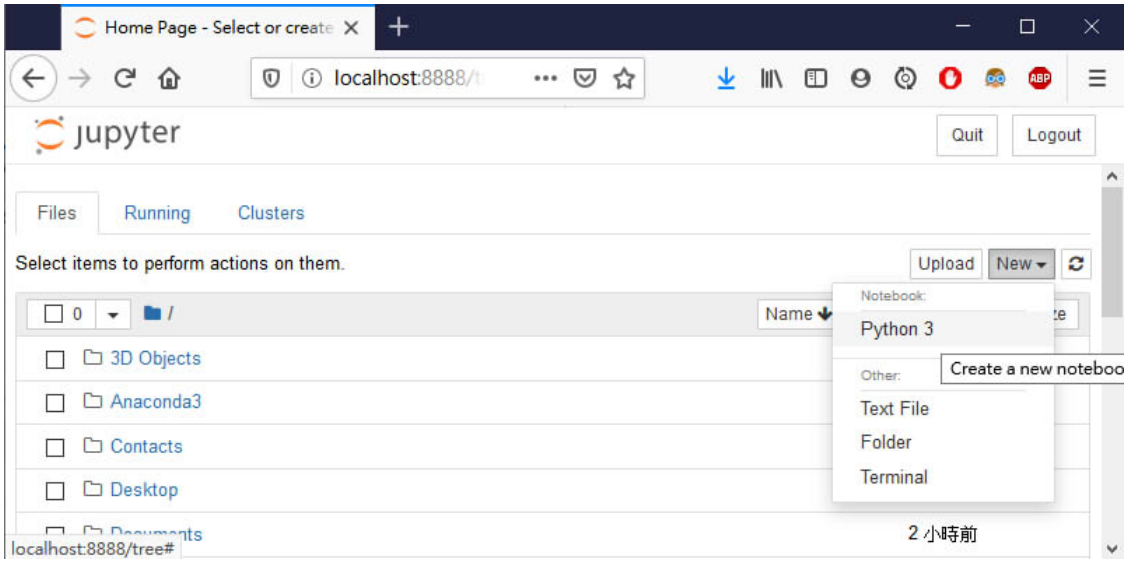
`jupyter` 的優點還有可以分塊執行，不用一次執行所有的程式碼；以及很方便的移動程式碼區塊。

3.1 使用 jupyter notebook (base 環境下)

- 打開 `power shell` ，打 `jupyer notebook` 就會自動打開一個瀏覽器頁面 `localhost:8888`。

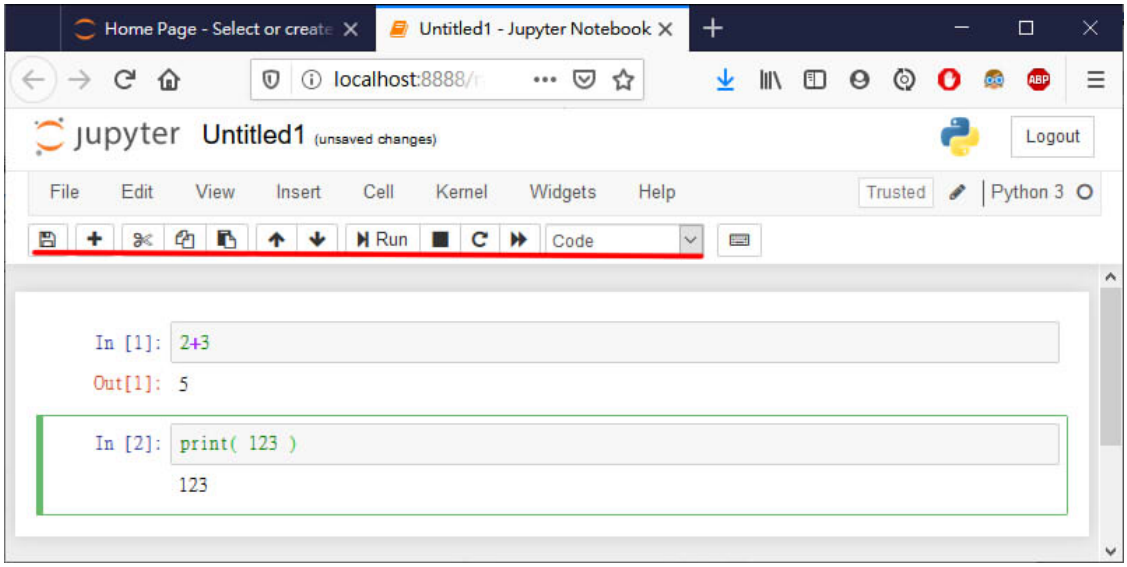
  【注意】執行指令的視窗關掉時，就會連不上這個頁面。
  【注意】會根據執行指令的目錄來決定 `jupyter notebook` 開啟的資料夾位置。

- 右上角的 `New` > `Python 3` 就會開啟一個新的頁面



- 直接在框框內就可以打 `python` 程式碼，並使用上方工具列來操作 (紅線)，實地操作一下就能明白了。

  特別值得注意的是最右方的 Code，也可以使用 Markdown，你可以使用 Markdown 來寫註解。(他會轉換整個格子)



3.2 使用 jupyter notebook (虛擬環境下)

當需要使用虛擬環境的時候，需要額外安裝一個套件 `nb_conda`。

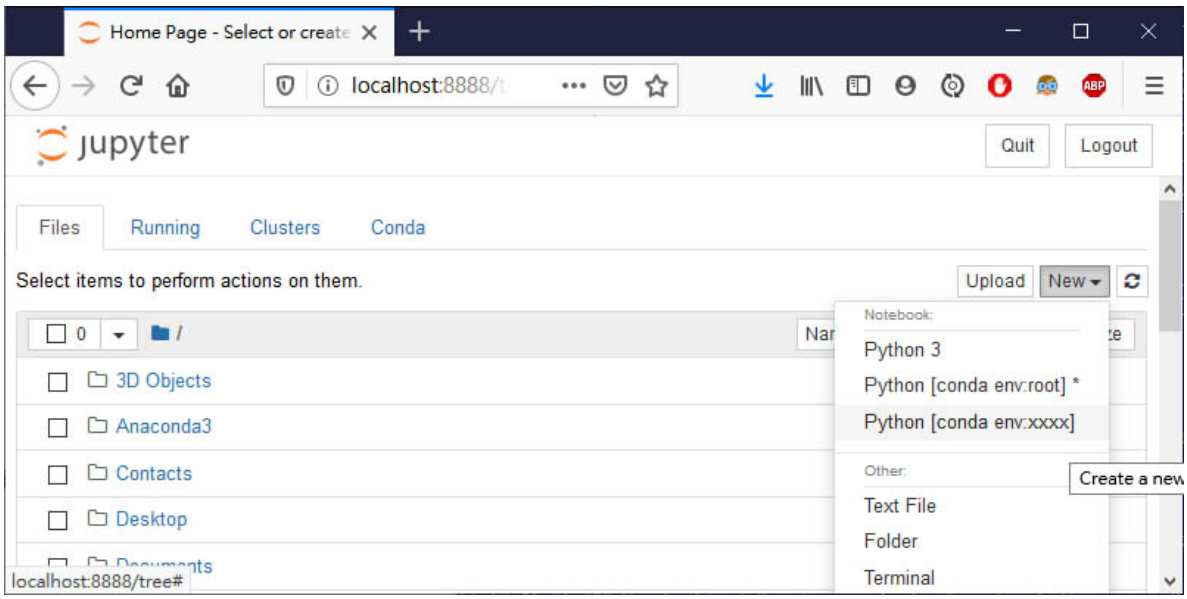
3.2.1 安裝 nb_conda (在 base 環境底下)

執行 `conda install nb_conda`

3.2.2 打開 juypter notebook

執行 `jupyter notebook`

安裝後再打開就會多出已經建立的環境可以選。



4. 使用記事本軟體搭配 CLI 來寫 python

在記事本軟體 (ex. Sublime, VScode) 完成程式碼 ...

打開 `power shell`

1. 打 `python a.py` 執行檔案 (在 `base` 環境下執行一個 `.py` 檔案)
2. 先 `conda activate xxxx`，再打 `python a.py` 執行檔案 (在 `xxxx` 環境下執行一個 `.py` 檔案)

通常這類專門寫程式的記事本軟體都可以在編寫介面裡面打開 CLI，不用額外開一個視窗；通常也可以設置熱鍵，讓使用者直接 `ctrl` + `B` 之類的熱鍵就能完成編譯、執行。

基礎

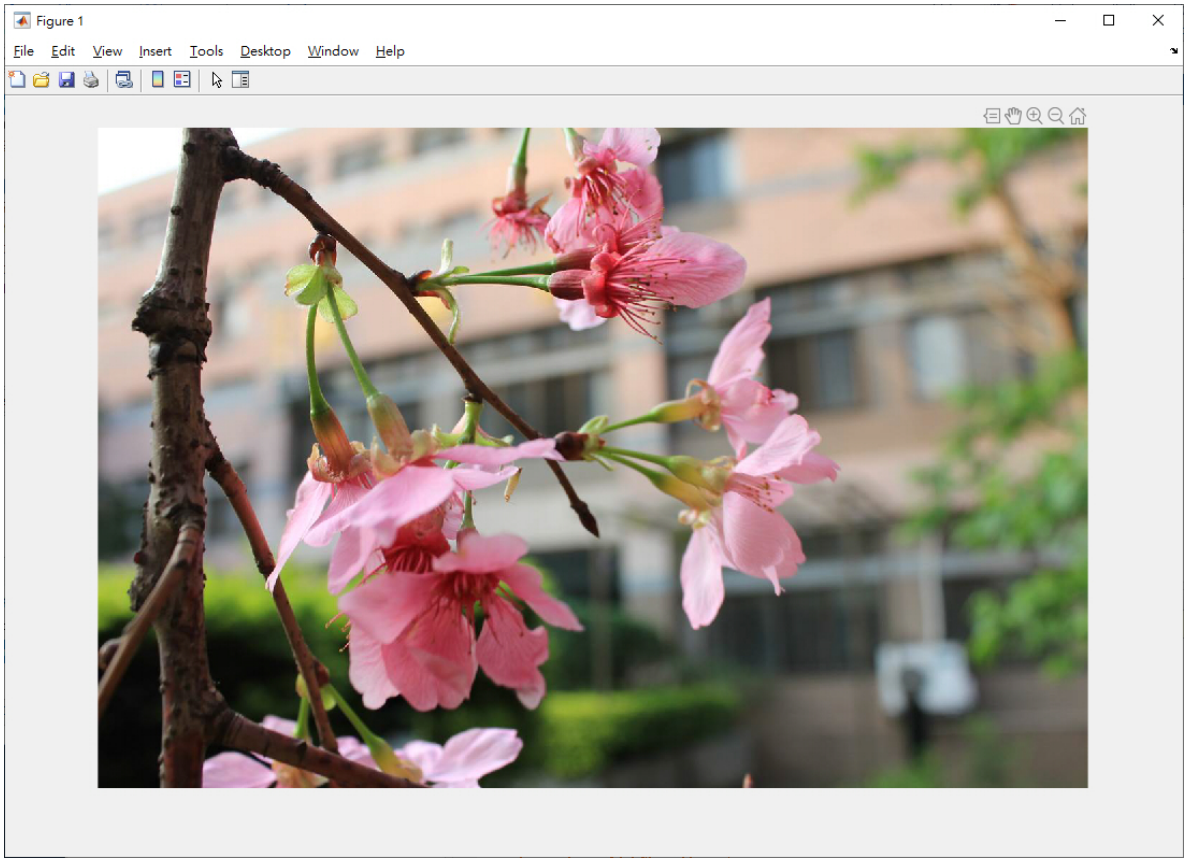
章節重點

影像讀取、顯示

使用 `imread` 就可以讀取影像，讀取進來的資料就會以矩陣形式儲存。

以下圖的彩色圖片，矩陣大小為 $921 \times 1382 \times 3$ (高、寬、三色)。

```
%-- example1.m --%
% 讀取並存入變數 img
img = imread('demo_color.bmp');

% 開新視窗
figure

% 顯示讀入的圖片
imshow(img)
```



影像大小、組成、修改

讀取進來的圖片就可以直接修改矩陣，修改圖像的值。

- 注意在 `matlab` 中，矩陣編號是從 `1` 開始。(`python`, `c++` 等等大部分程式語言都是從 `0` 開始)。
- 矩陣的第一個維度是 **高**，第二個是 **寬**
- 如果是彩色圖像的話，還會有第三個維度，分別代表 RGB。(在 `opencv` 中順序為 BGR)

```
%-- example2 --%
% 讀入圖片 (彩色與灰階)
cimg = imread('demo_color.bmp');

% 顯示圖片大小 (h,w,d) 高, 寬, 深
size(cimg)
size(gimg)

% 色彩分離 (Red)
figure(1)
tmp = cimg;
tmp(:,:,[2,3]) = 0;
imshow( tmp )

% 色彩分離 (Green)
figure(2)
tmp = cimg;
tmp(:,:,[1,3]) = 0;
imshow( tmp )

% 色彩分離 (Blue)
```

```
figure(3)
tmp = cimg;
tmp(:,:,[1,2]) = 0;
imshow( tmp )

% 修改圖片左上角 300*400 (高*寬) pixels
ae_img = cimg;
ae_img( 1:300 , 1:400 , : ) = 0;
figure(4)
imshow( ae_img )
```

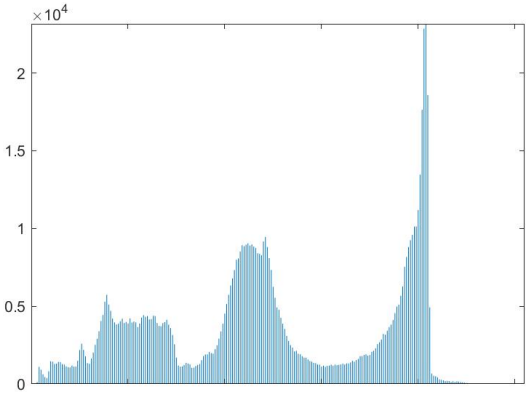
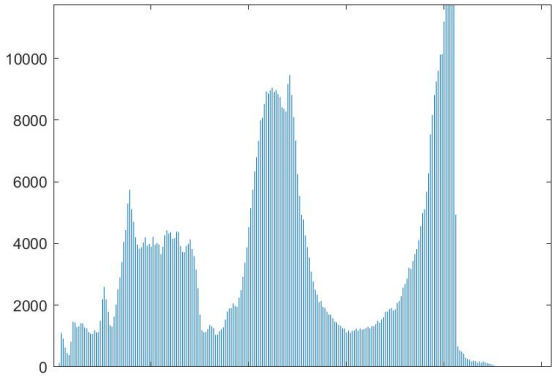
亮度直方圖

對圖片統計各個亮度的出現次數，可以得到亮度直方圖，可以看出大概的亮度分布。

在 `matlab` 中使用 `imhist` 。

- 對灰階圖片來說，會有一個亮度直方圖。
- 對彩色圖片來說，因為有 RGB 三個，所以理論上會有三個亮度直方圖，但一般而言，對彩色圖片取亮度直方圖時，通常是先把圖片灰階化後再取直方圖。

```m
%-- example3 --%
% 讀入圖片
img = imread('demo_gray.bmp');

% 顯示強度的直方圖
imhist(img)

% 讓直方圖頂端不碰到方框，副作用是當值差距過大時會看不太出大致的形狀。
axis tight

figure
imhist(img)
```

有 `tight`	沒有 `tight`
	

原圖 (demo_gray.bmp)	
	.....

回 第 4 課 亮度直方圖再提

彩色與灰階圖片

一般最常接觸到的兩類圖片，一個是灰階圖片(黑白圖片)，一個是彩色圖片。

從維度上來看，灰階圖片是 400×600 (高寬)，彩色圖片則是 $400 \times 600 \times 3$ 。

可以透過把彩色圖片的 RGB 三個值相加除以 3 來得到灰階圖片。

可以透過把灰階圖片的值複製三份，放到 RGB 的位置來得到彩色圖片 (即使實際上看起來還是灰階圖)。

在 `matlab` 中使用 `rgb2gray` 及 `gray2rgb` 來處理。

彩色	灰階
	

```
%-- example4 --%
% 讀入圖片
img = imread('demo_color.bmp');

imshow( rgb2gray(img ) )
```

要注意的是，`matlab` 在 `rgb2gray` 的實作中並不是採用 **相加除以 3** 的作法，而是加權平均。

實際上的係數為 $0.2989 \times R + 0.5870 \times G + 0.1140 \times B$ ，這個根據人眼對不同顏色感光程度不同，實驗得到的數值，一般不會特別在意究竟是用 **平均** 還是 **加權平均** 。(從上面的係數你可以s發現人眼認為綠色比較亮、藍色比較暗)

要注意轉成灰階圖的過程會遺失色彩的資訊，圖片使用 $\begin{cases} R = [255, 20, 20] \\ G = [0, 153, 0] \\ B = [69, 69, 255] \end{cases}$

資料格式

在影像上常見的格式有幾種，在寫程式處理圖片的過程中盡量使用統一的格式，減少麻煩。

有時候圖片顯示不正常可能就是數值的範圍是 [0,255] 的，但卻用 `double` 型態來存了。

可以用 `whos` 接變數名稱來查詢類別。

類別	uint8	uint16	double	logical
數值範圍	[0,255]	[0,65535]	[0,1]	{0,1}
影像轉換函式	im2uint8()	...	im2double()	...
轉換函式	uint8()	uint16()	double()	boolean()

其中影像轉換函式會做數值映射，原本是 double 數值介在 [0,1] 之間，再轉換成 uint8 時，數值會等比例映射到 [0,255]。

一般的類別轉換函式不會改變數值。

常用的有 `im2double` 和 `im2uint8` 。

影像寫入、影像格式

在把處理後的影像儲存有很多格式，但常用的是 `jpg` 、 `png` 、 `bmp` 。

在 `matlab` 中儲存影像使用 `imwrite( 影像矩陣 , 檔名 )` ，會根據檔案的副檔名自動設定好檔案格式，所以要存成 `jpg` 就 `.jpg` 結尾這樣。

實際上 `imwrite` 還有很多可以設定的參數，可以查看 `help imwrite` ，後續用到的時候會在介紹進階的參數。

```
%-- example7 --%
% 讀入圖片
img = imread('demo_color.bmp');

gimg = rgb2gray(img );

% 寫入圖片
imwrite( gimg , 'gray.jpg'  );
```

這裡有一些儲存格式優缺點。

檔案格式	優點	缺點
`.jpg`	檔案小	破壞性壓縮，像素值會稍微跑掉 不支援透明度
`.png`	無失真壓縮，像素值不會跑掉 支援透明度 檔案比 jpg 大，比 bmp 小	
`.bmp`	無失真壓縮，像素值不會跑掉 儲存格式很簡單。	檔案非常大 不支援透明度

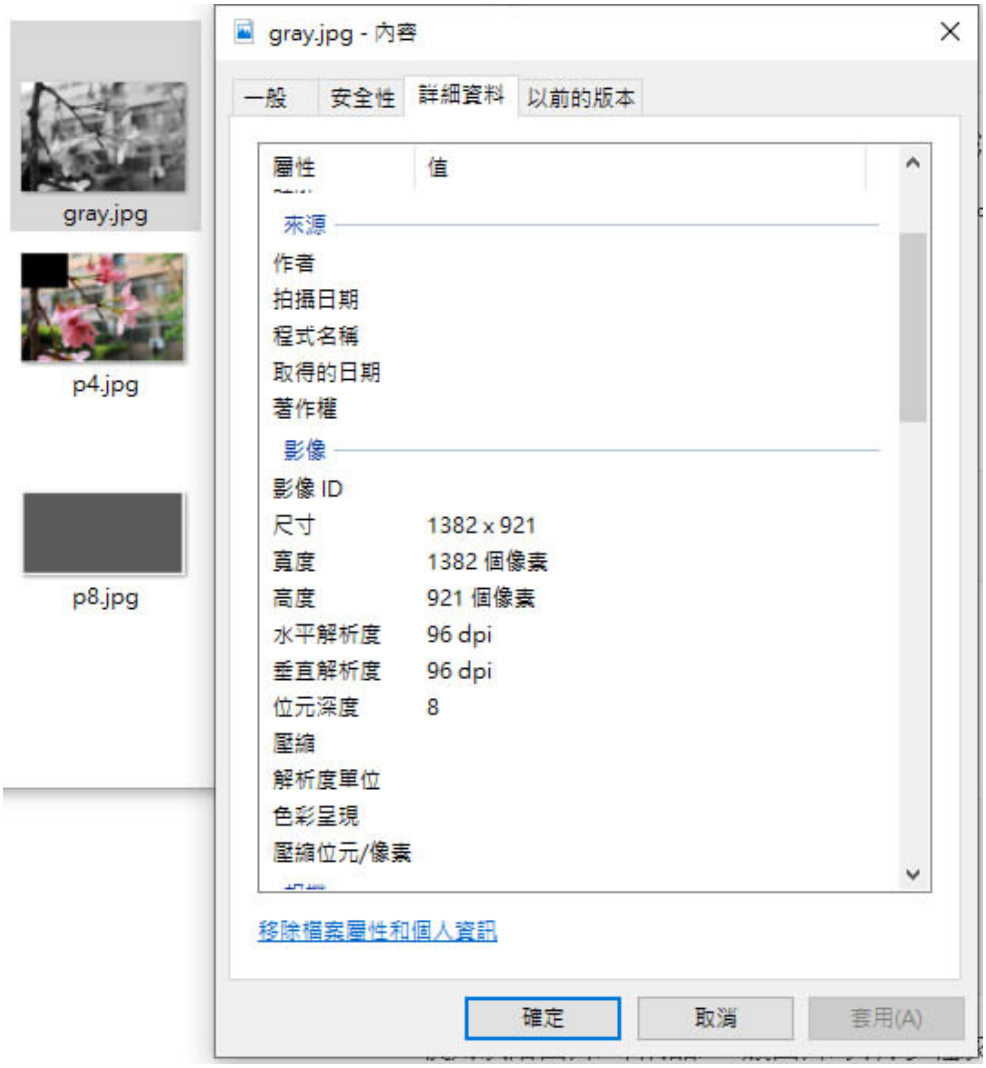
強度解析度與位元深度

在影像儲存方面，有一個重要的資訊：位元深度。

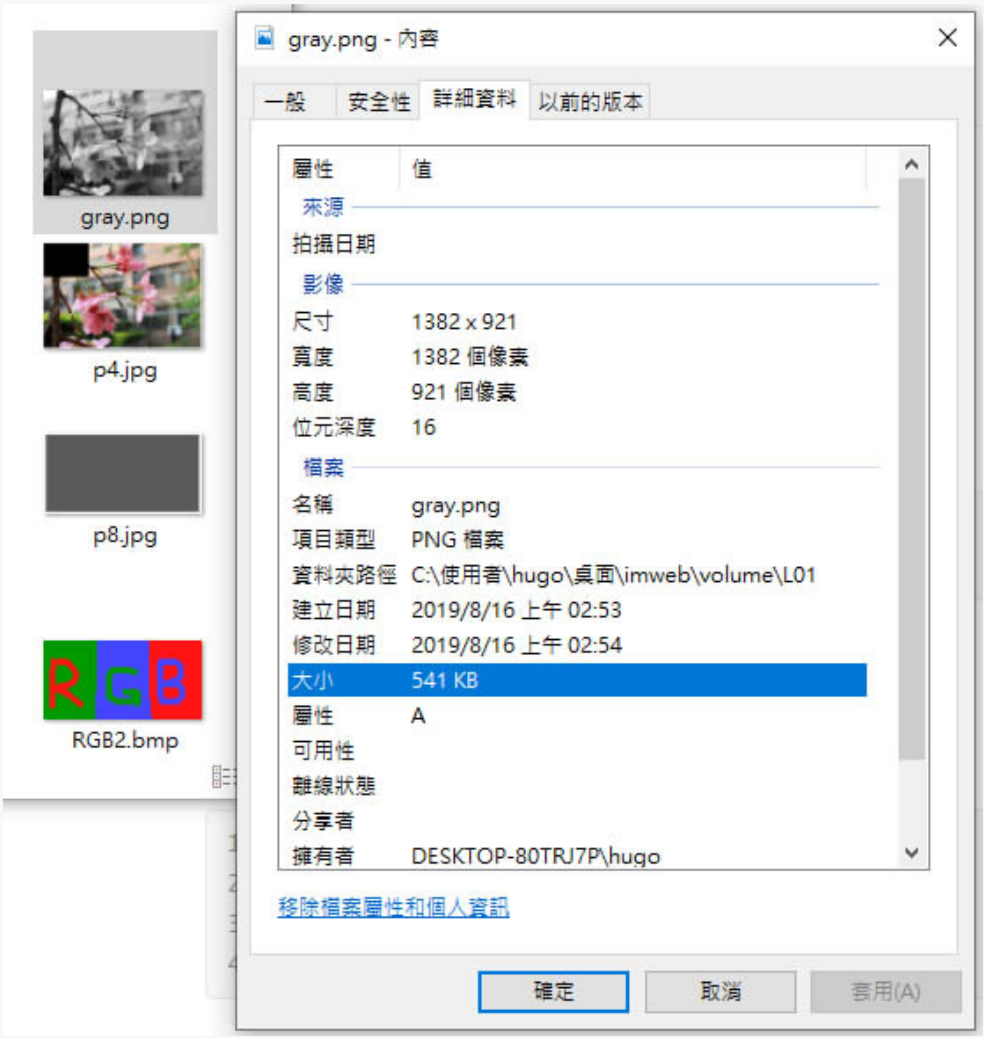
它代表每個像素實際上用了幾個 bit 來儲存。

一般彩色影像的位元深度是 24

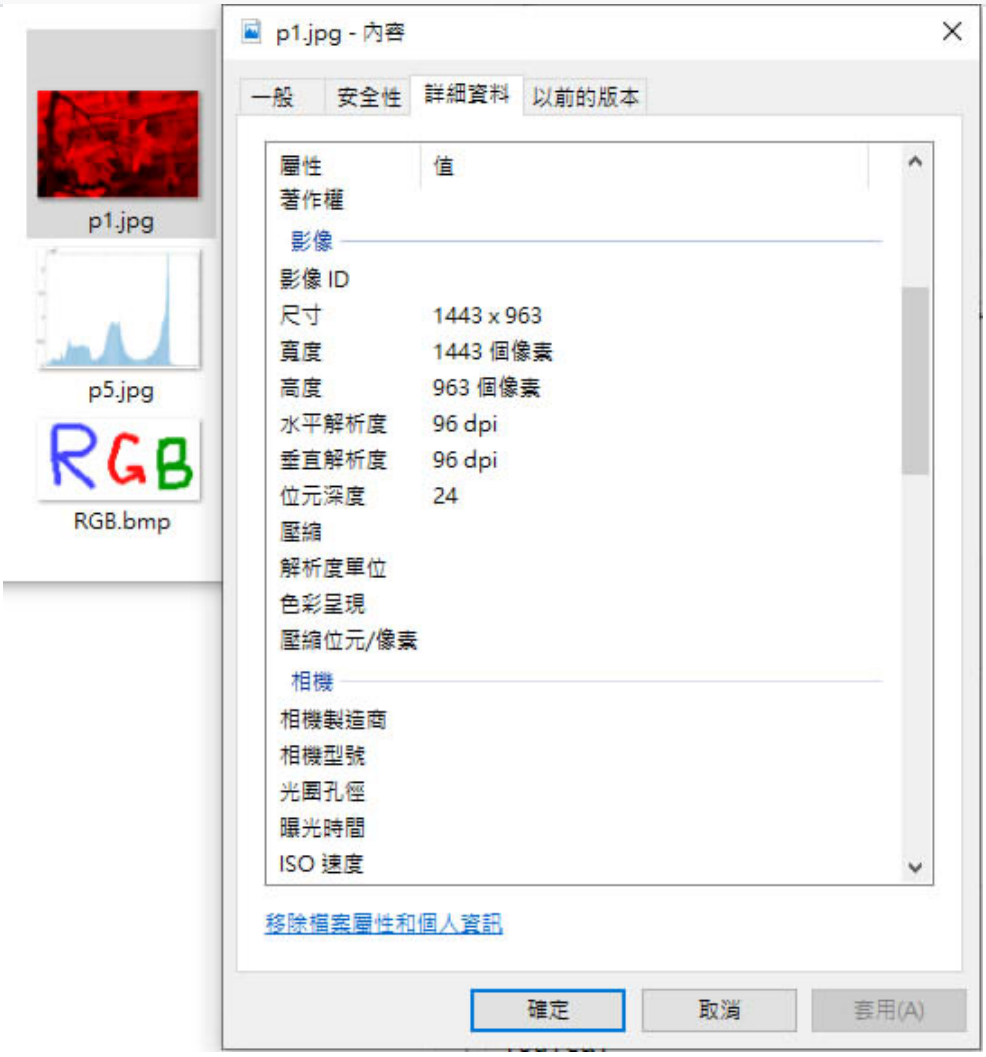
灰階影像常見的位元深度的則有 8 和 16 。(16 通常是 `.png` 常見在 3D 相機擷取出的圖片)

..... 位元深度	
8	

gray.jpg
p4.jpg
p8.jpg
gray.jpg - 內容
一般
安全性
詳細資料
以前的版本
屬性
值
來源
作者
拍攝日期
程式名稱
取得的日期
著作權
影像
影像 ID
尺寸 1382 x 921
寬度 1382 個像素
高度 921 個像素
水平解析度 96 dpi
垂直解析度 96 dpi
位元深度 8
壓縮
解析度單位
色彩呈現
壓縮位元/像素
移除檔案屬性和個人資訊
確定
取消
套用(A)

位元深度	
16	 gray.png p4.jpg p8.jpg RGB2.bmp gray.png - 內容 一般 安全性 詳細資料 以前的版本 屬性 值 來源 拍攝日期 影像 尺寸 1382 x 921 寬度 1382 個像素 高度 921 個像素 位元深度 16 檔案 名稱 gray.png 項目類型 PNG 檔案 資料夾路徑 C:\使用者\hugo\桌面\imweb\volume\L01 建立日期 2019/8/16 上午 02:53 修改日期 2019/8/16 上午 02:54 大小 541 KB 屬性 A 可用性 離線狀態 分享者 擁有者 DESKTOP-80TRJ7P\hugo 移除檔案屬性和個人資訊 確定 取消 套用(A)
24	

..... 位元深度	



實際上透過 `matlab` 也可以儲存成有支援的位元深度。

位元深度	圖片
位元深度 16	
位元深度 8 (原本的大小)	

位元深度	圖片
位元深度 4	
位元深度 2 (4 種顏色)	

位元深度	圖片
位元深度 1 (2 種顏色)	

```
%-- example8 --%
% 讀入圖片
img = imread('demo_color.bmp');

gimg = rgb2gray(img );

imwrite( gimg , 'gray16.png' , 'BitDepth', 16 );
imwrite( gimg , 'gray8.png' , 'BitDepth', 8 );
imwrite( gimg , 'gray4.png' , 'BitDepth', 4 );
imwrite( gimg , 'gray2.png' , 'BitDepth', 2 );
imwrite( gimg >128 , 'gray1.png' , 'BitDepth', 1 );
```

遮罩處理

章節重點

- ROI-建立遮罩
- ROI-設定座標
- 透明背景疊合 (合成)
- 色彩增值
- 濾色
- 覆蓋
- 差異化
- 色彩混合/調色

ROI-建立遮罩

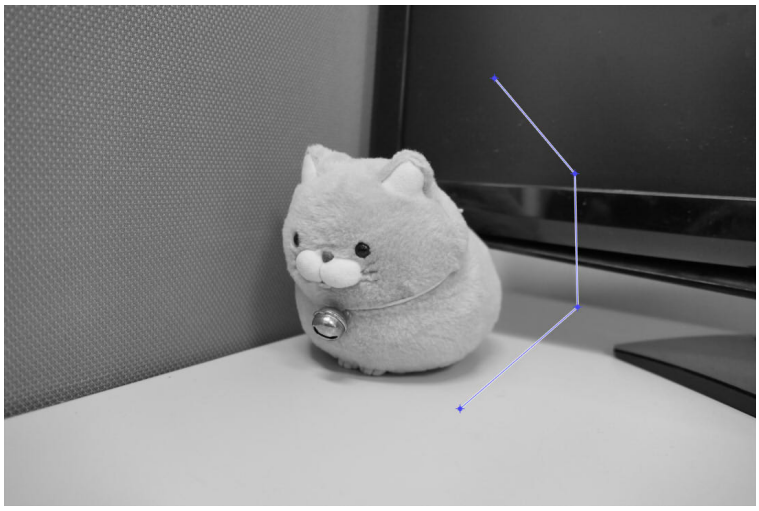
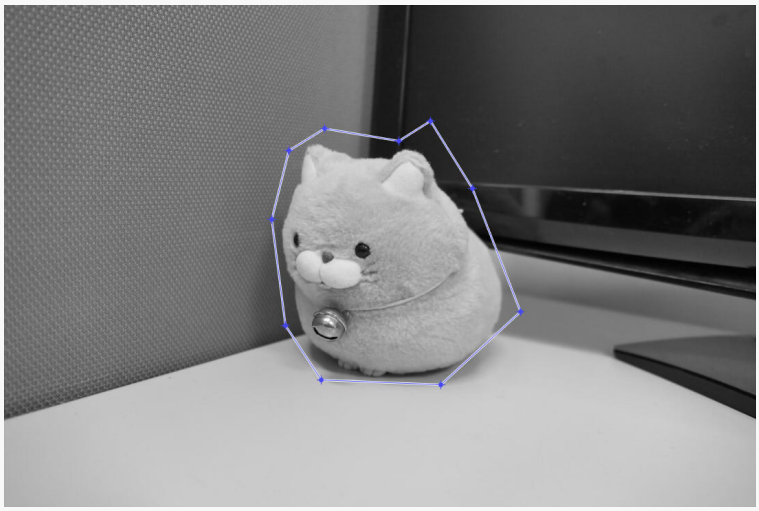
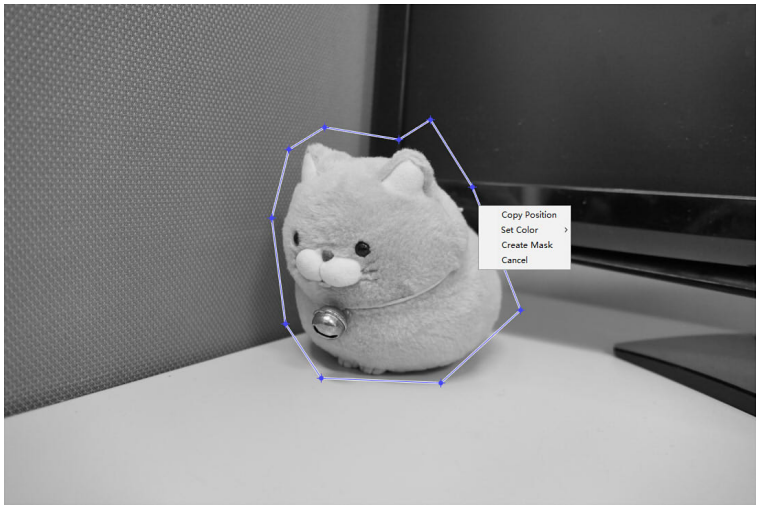
在影像處理的過程中，有時候會需要選取一個範圍(可能是前面過程中擷取到的範圍)，稱為 ROI (Region Of interest)，再接下去做處理。

但有時候前面的處理程式還沒有寫好，在測試後續程式的時候就會造成困擾。

這種情況就可以使用 `roipoly` 來生成一個選取範圍。

先編寫程式碼如下

```
%-- example1 --%
gimg = imread('demo_gray.png');
mask = roipoly(gimg);
```

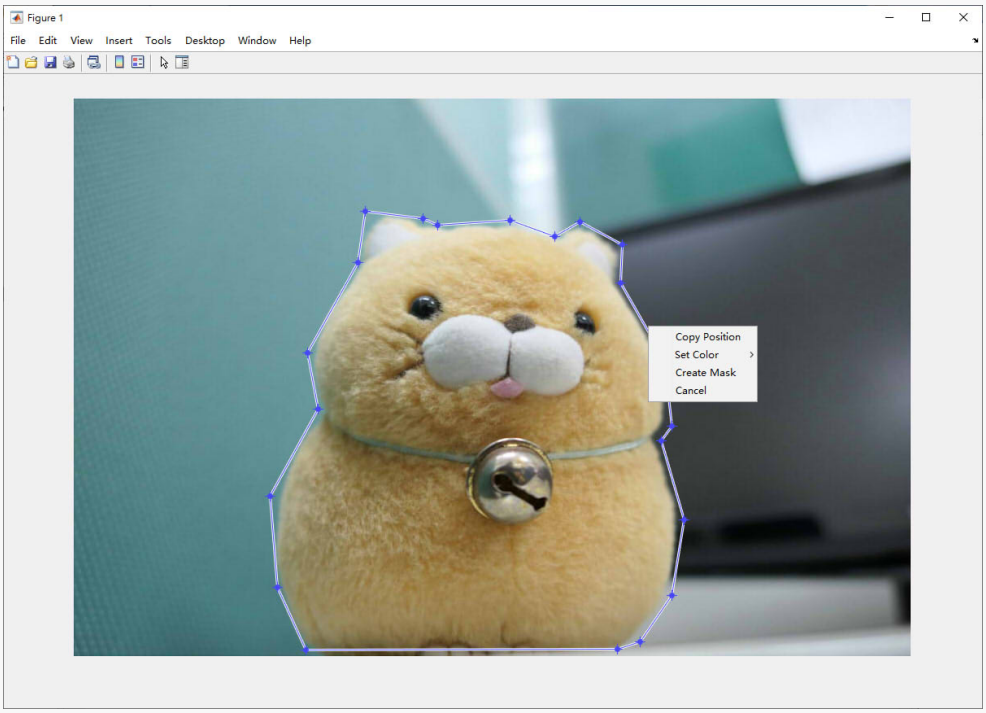
流程	參考圖片
執行後會打開互動介面，可以在上面點，會跑出點和連接線。點可以事後微調、刪除。	
完成後點兩下或點右鍵就會封閉線段。	
點右鍵會有 `Copy Position` ：把選取的點座標存到剪貼簿 會是一個 $n\times 2$ 的矩陣 以及 `Create Mask` 建立一個與原圖相等大小的 {0,1} 遮罩存到變數裡。	

實際上操作可能會如下：

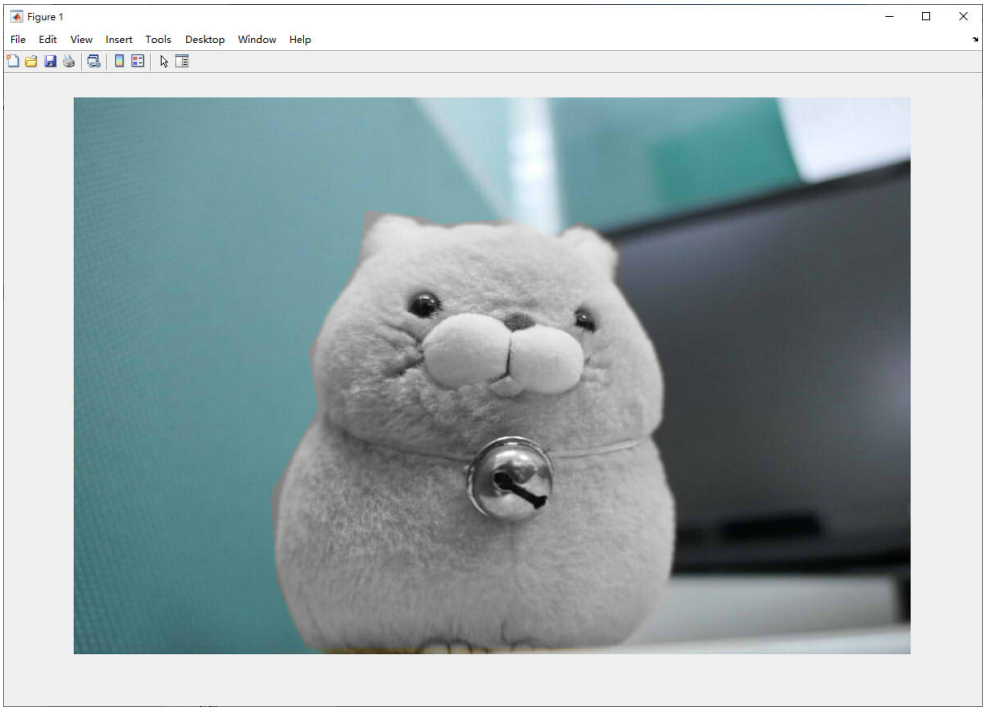
```
%-- example2 --%
cimg = imread('demo_color.jpg');
mask = roipoly(cimg);

gimg = rgb2gray( cimg );

%-- RGB 分開處理，用灰階的顏色代替。
```

```
for i=1:3
    tmp = cimg( : , : , i );
    tmp( mask ) = gimg( mask );
    cimg( : , : , i ) = tmp;
end

imshow( cimg );
```

選取範圍



灰階處理



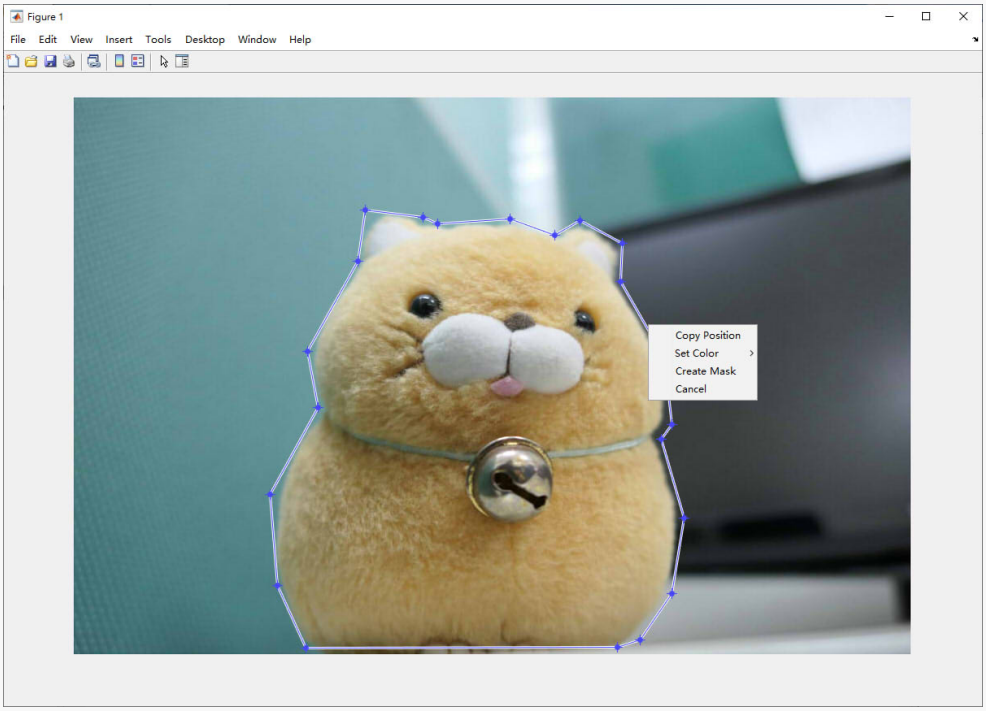
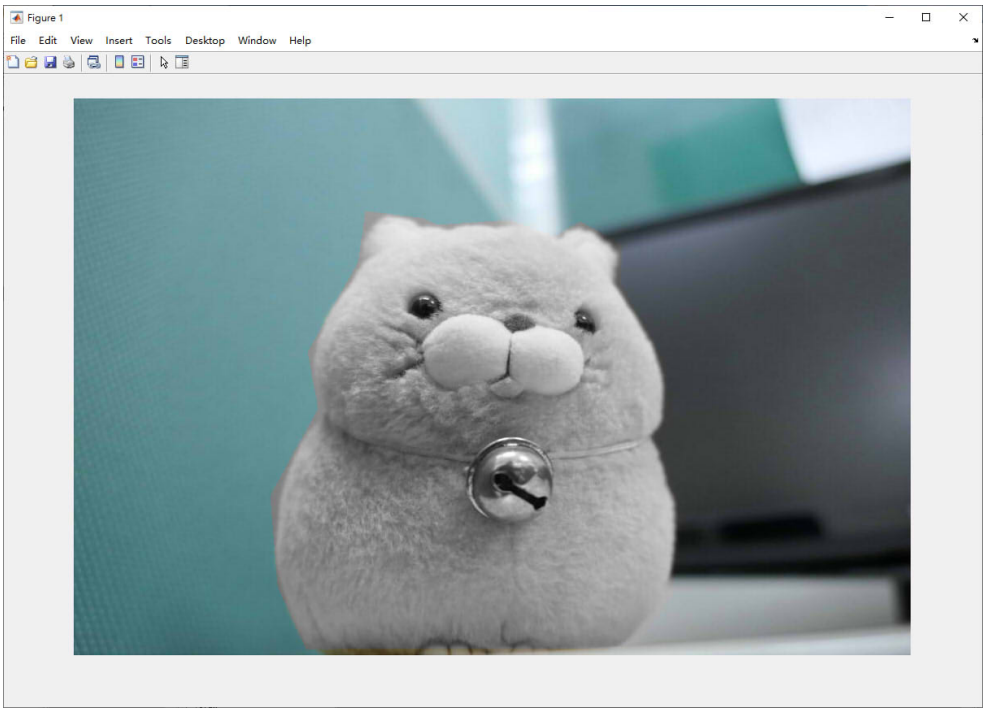
ROI-設定座標

使用 `Copy Position` 就可以只框選一次，之後都用之前框選的結果。

`pos(:,1)` 是點的 x 座標， `pos(:,2)` 是點的 y 座標。

```
%-- example3 --%
cimg = imread('demo_color.jpg');
pos = [
    287 683;245 586;247 472;299 389;280 311;
    338 195;363 139;415 135;442 151;522 150;
    598 169;623 150;666 172;679 205;671 224;
    710 280;729 331;742 402;713 417;736 458;
    748 515;738 608;716 650;679 675;654 680];

mask = roipoly( gimg , pos(:,1) , pos(:,2) );

gimg = rgb2gray( cimg );

%-- RGB 分開處理，用灰階的顏色代替。
for i=1:3
    tmp = cimg( : , : , i );
    tmp( mask ) = gimg( mask );
    cimg( : , : , i ) = tmp;
end

imshow( cimg );
```

產生和上面相同的結果

選取範圍	
灰階處理	

透明度

透明背景疊合 (合成)

色彩增值

濾色

覆蓋

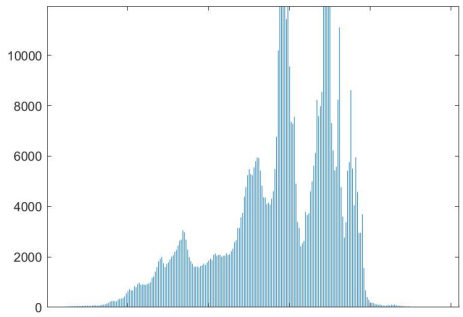
差異化

色彩混合/調色

單點強度轉換

亮度直方圖再提

```
%-- example1 --%
gimg = imread('demo_gray.jpg');
imhist(gimg)
```

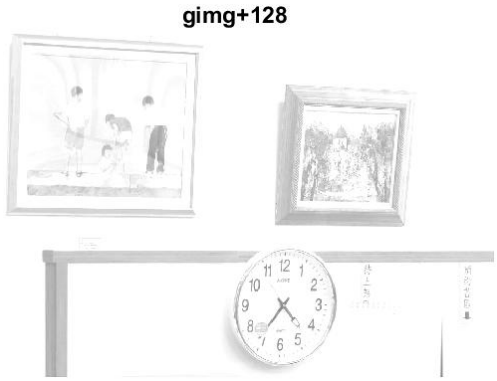
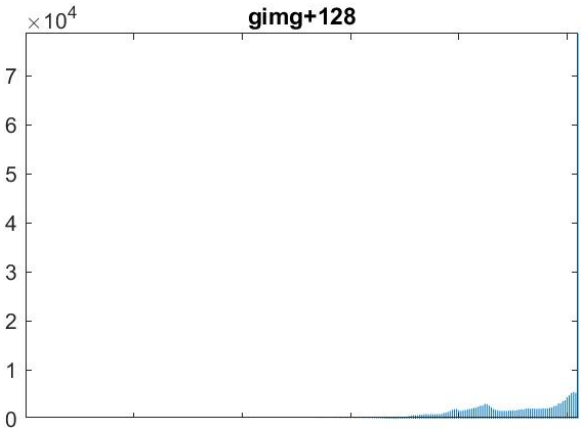
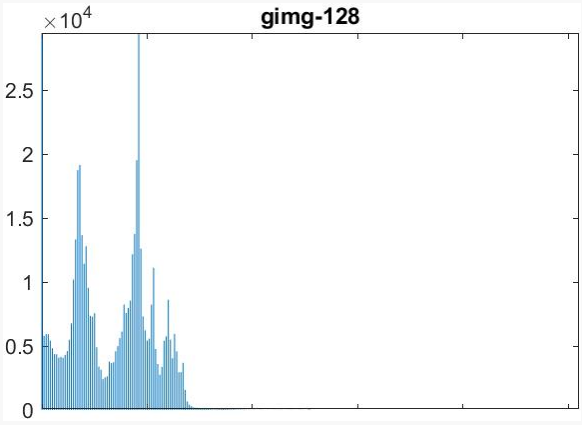
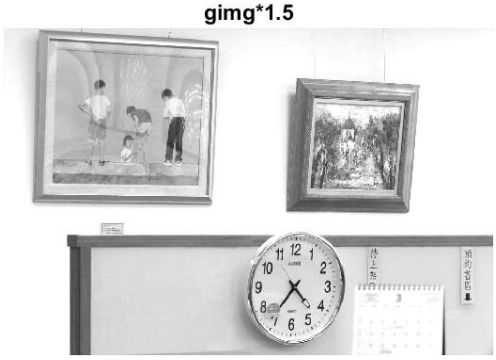
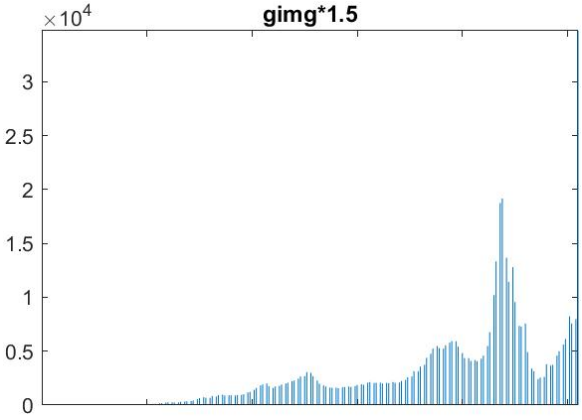
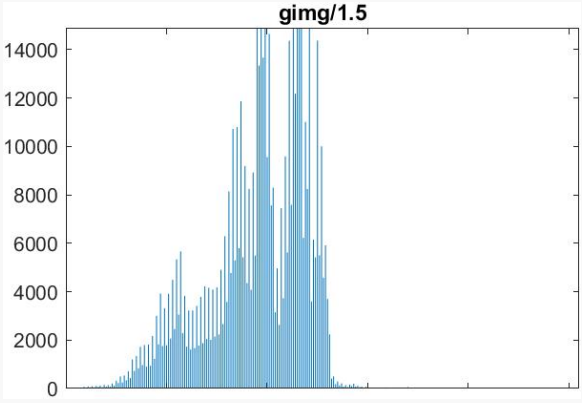
圖片	亮度直方圖
	

相關函數 `imhist(img)`

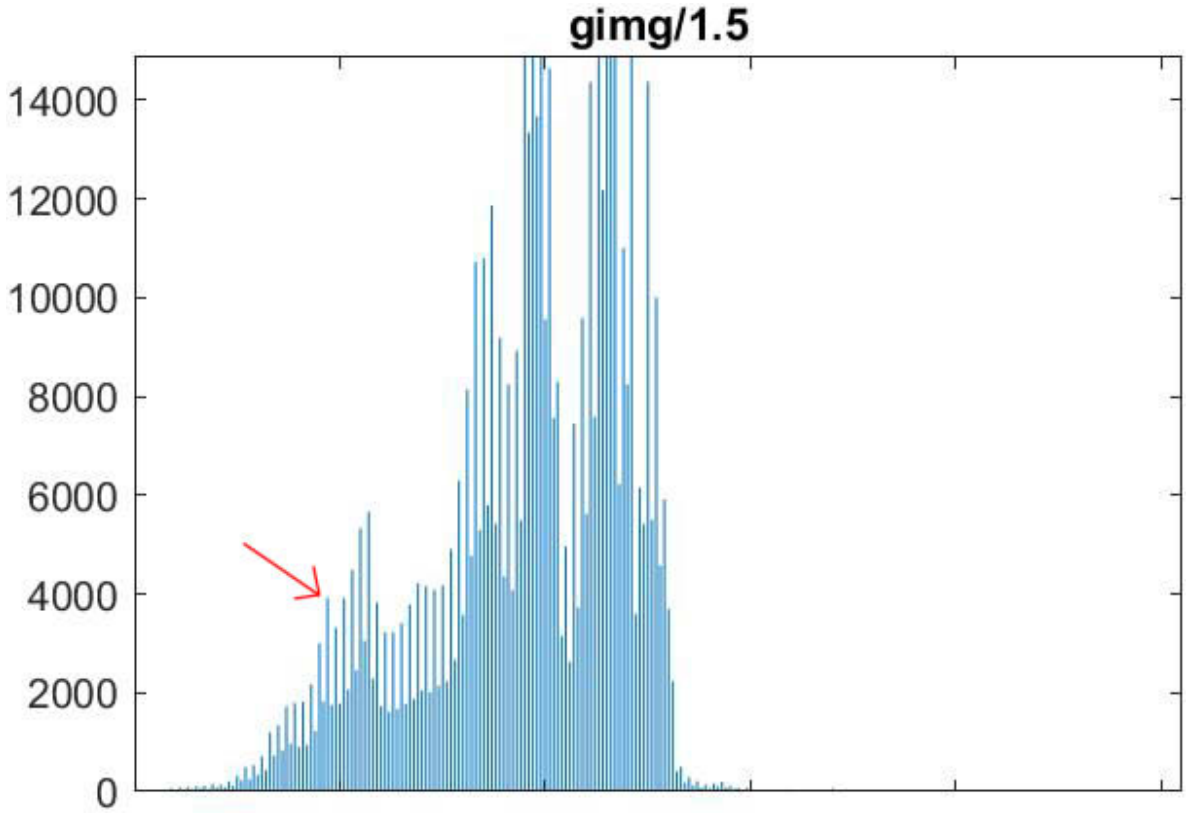
詳見 第 1 章亮度直方圖

點亮度轉換

對同一張圖全部像素做加減乘除。

圖片	亮度直方圖
	
	
	
	

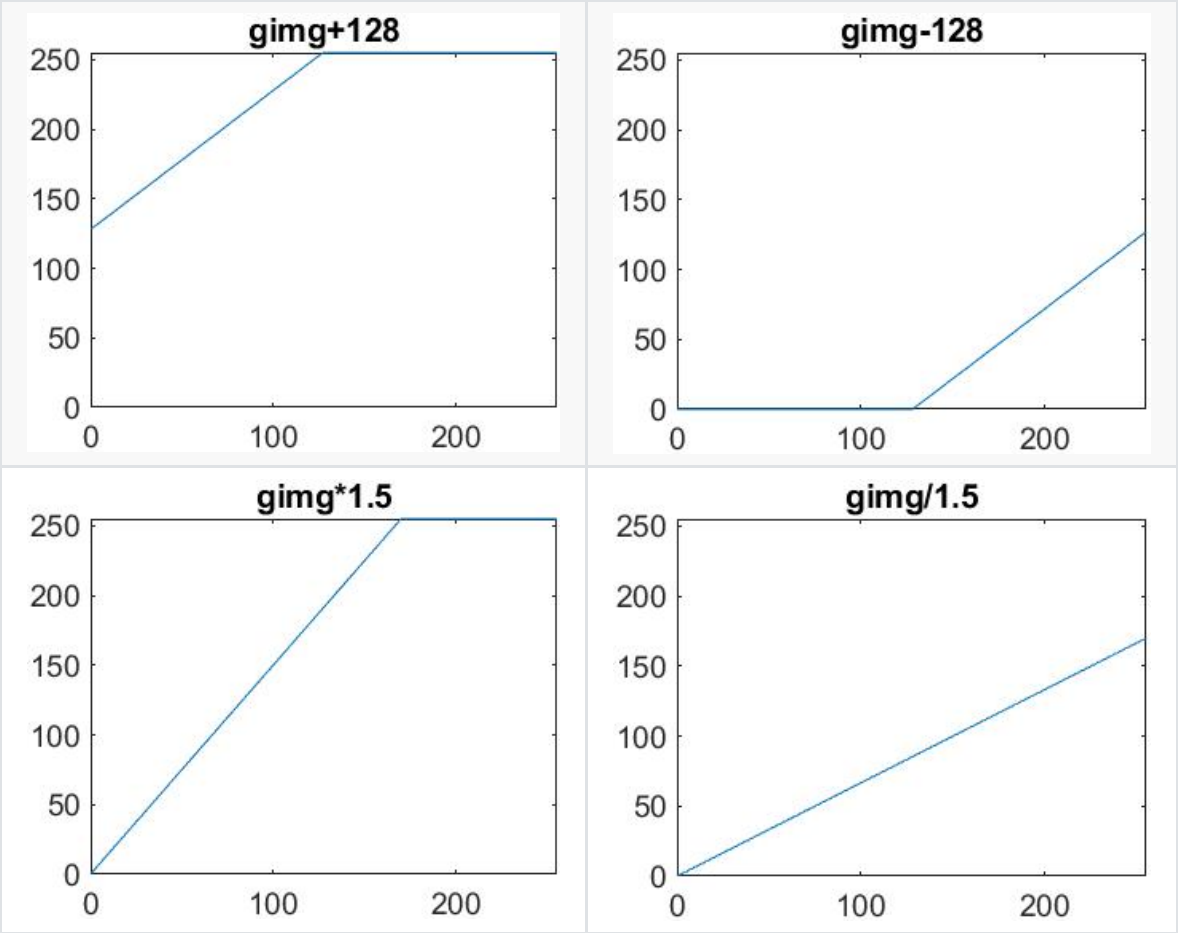
- 加減會讓原本的直方圖左右**平移**。
- 乘大於 1 的數值會讓直方圖左右**延展**。
- 乘小於 1 的數值則會**壓縮**直方圖，同時每隔一段距離就會一個特別高的值(註)。



- 註：以除以 1.5 來說，原本強度 192 與 193 的像素，轉換之後都會變成 128, 強度 194 的轉換到 129 ...。
- 如果除的數值不是剛好 2 的次方倍，就會造成每個數值分配到的"強度數量"不一樣。

原本	轉換後
192,193	128
194	129
195,196	130
...	...

- 這樣的轉換關係也可以畫成函數 (x軸是原本的強度, y軸是轉換後的強度)。



點亮度轉換的 `matlab` 程式碼：

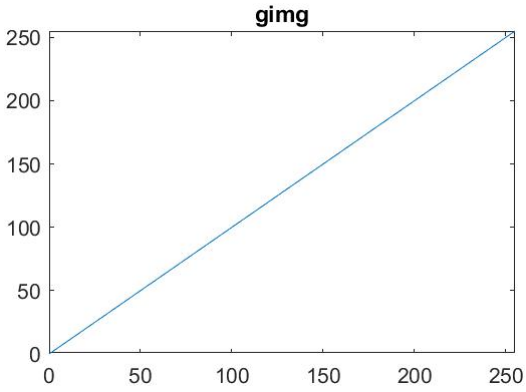
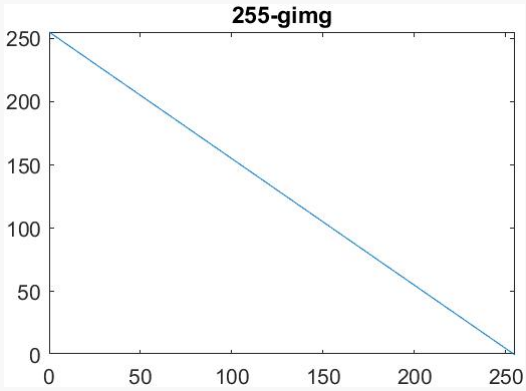
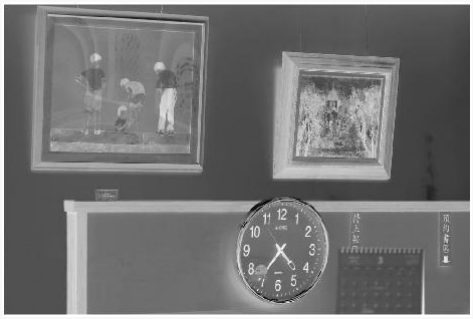
```matlab
%-- example2 --%
gimg = imread('demo_gray.jpg');

figure
subplot(2,2,1)
imshow(gimg+128)
title('gimg+128')

subplot(2,2,2)
imshow(gimg-128)
title('gimg-128')

subplot(2,2,3)
imshow(gimg*1.5)
title('gimg*1.5')

subplot(2,2,4)
imshow(gimg/1.5)
title('gimg/1.5')

figure
subplot(2,2,1)
imhist(gimg+128)
title('gimg+128')

subplot(2,2,2)
imhist(gimg-128)
```

```
title('gimg-128')

subplot(2,2,3)
imhist(gimg*1.5)
title('gimg*1.5')

subplot(2,2,4)
imhist(gimg/1.5)
title('gimg/1.5')
```

負片

把原本 {0,1,...,254,255} 映射到 {255,254,...,1,0}。

在影像處理過程常常會碰到，尤其是在二值圖的部分。

變換	轉換圖	結果
不變		
負片		

```
%-- example4 --%
gimg = imread('demo_gray.jpg');
figure
x = uint8(0:255);

%-- 不做變換 gimg
subplot(2,2,1)
y = x;
plot(x,y)
axis( [0,255,0,255] );
```

```
title('gimg')

subplot(2,2,2)
imshow(gimg)

%-- 負片 255-gimg
subplot(2,2,3)
y = 255-x;
plot(x,y)
axis( [0,255,0,255] );
title('255-gimg')

subplot(2,2,4)
imshow(255-gimg)
```

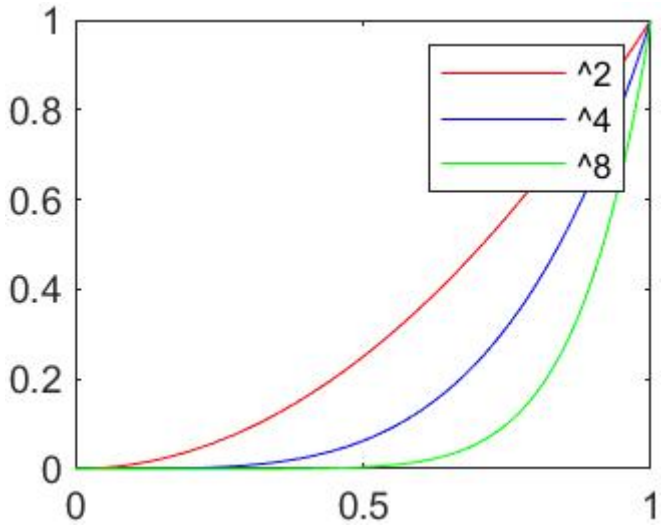
gamma 變換

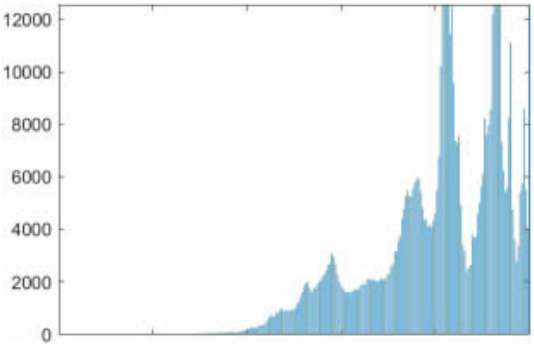
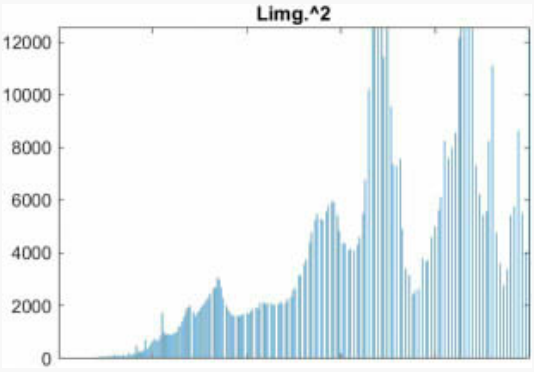
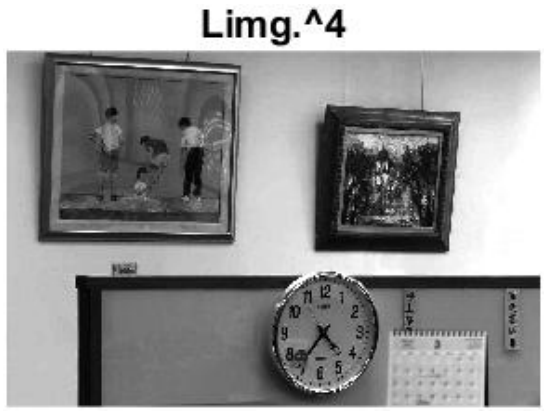
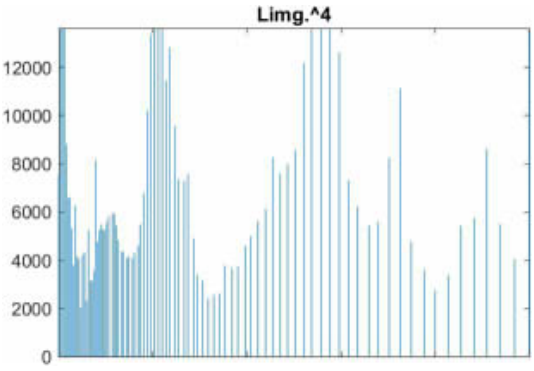
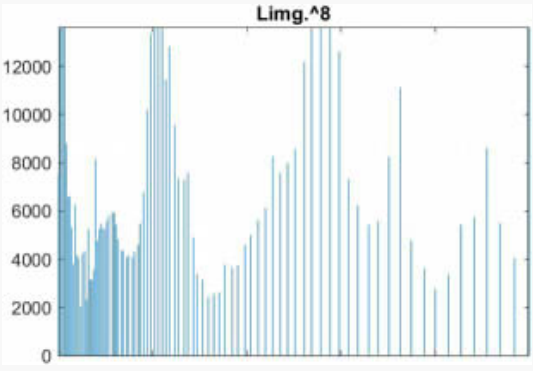
將影像數值 x 重設為 cx^γ，其中 c 為常數。

如果亮度集中在一小段區域，可以透過 gamma 變換重新分配到較大的區段。

要注意運算的數值範圍是 [0,1]，不是 [0,255]。

讓 $\gamma > 1$ ，相當於調暗。

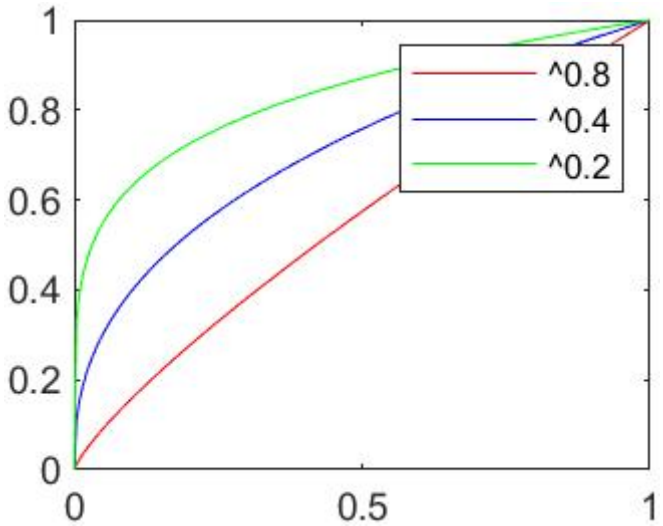
變換	轉換圖
gamma 變換 (調暗)	

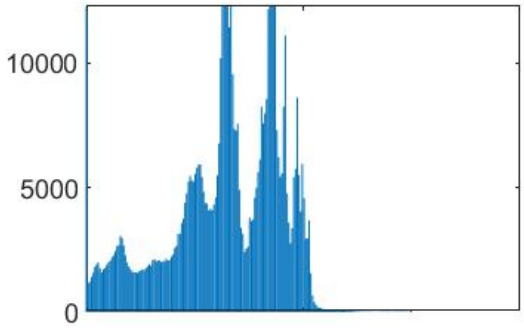
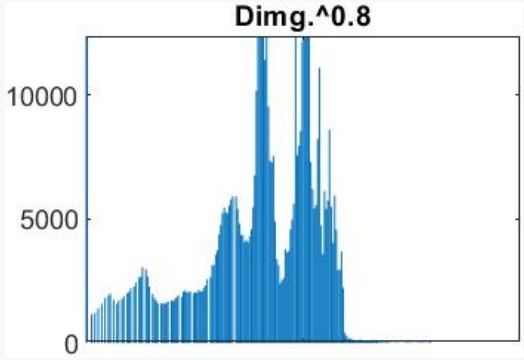
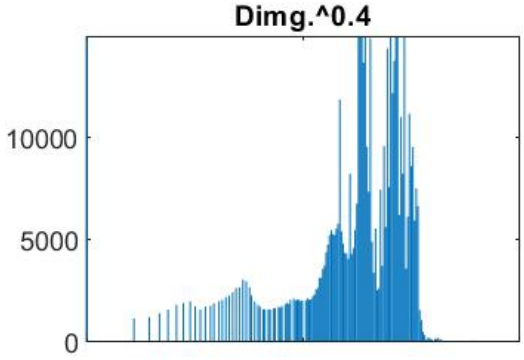
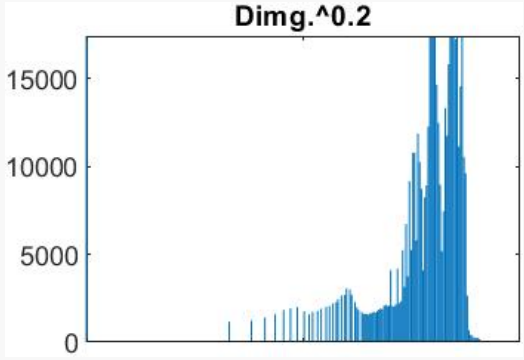
圖片	亮度直方圖
原圖 	
	
	
	

```
%-- example5 --%
gimg = imread('demo_gray.jpg');
% 亮的圖片
Limg = gimg + 64;
% 轉到 [0,1]
Limg = im2double(Limg);

```

```
figure
%-- 畫線
x = linspace(0,1,256);

subplot(2,2,1)
y = x.^2;
plot(x,y,'r')
axis( [0,1,0,1] );
hold on;

y = x.^4;
plot(x,y,'b')
axis( [0,1,0,1] );

y = x.^8;
plot(x,y,'g')
axis( [0,1,0,1] );

%-- 圖例
legend('\^2','\^4' ,'\^8');

%-- 畫圖
subplot(2,2,2)
imshow(Limg.^2)
title('gimg.\^2')

subplot(2,2,3)
imshow(Limg.^4)
title('gimg.\^4')

subplot(2,2,4)
imshow(Limg.^8)
title('gimg.\^8')
```

讓 $\gamma < 1$ ，相當於調亮。

變換	轉換圖
gamma 變換 (調亮)	

圖片	亮度直方圖
原圖 	
Dimg.^0.8 	Dimg.^0.8 
Dimg.^0.4 	Dimg.^0.4 
Dimg.^0.2 	Dimg.^0.2 

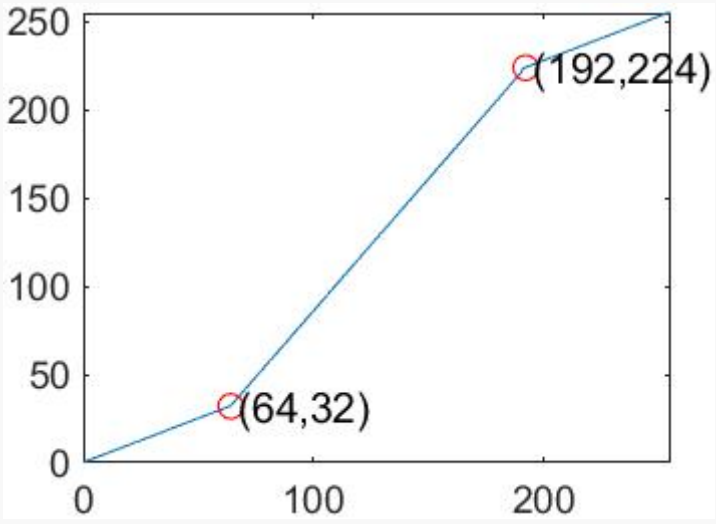
相關資料：[wiki Gamma correction]

查詢表格 Lookup Table

- 對 uint8 資料型態的圖片來說，轉換都是一個整數值對應到另一個整數值，可以把這樣的對應關係寫成表格，之後轉換強度值時就可以直接**查表**而不是**計算**，使用查詢表格可以節省計算量。。。
- 以**除以二**這個運算來寫的查詢表格如下

0	1	2	3	4	...	252	253	254	255
0	0	1	1	2	...	126	126	127	127

接下來是一個查詢表格的範例，加強一個亮度區間的對比度。

轉換圖	
原圖	
處理後	

`matlab` 的程式碼，注意最後的 `y(gimg+1)`，`y` 是查詢表格。

```
%-- example7 --%
%-- 製作查詢表格 y
```

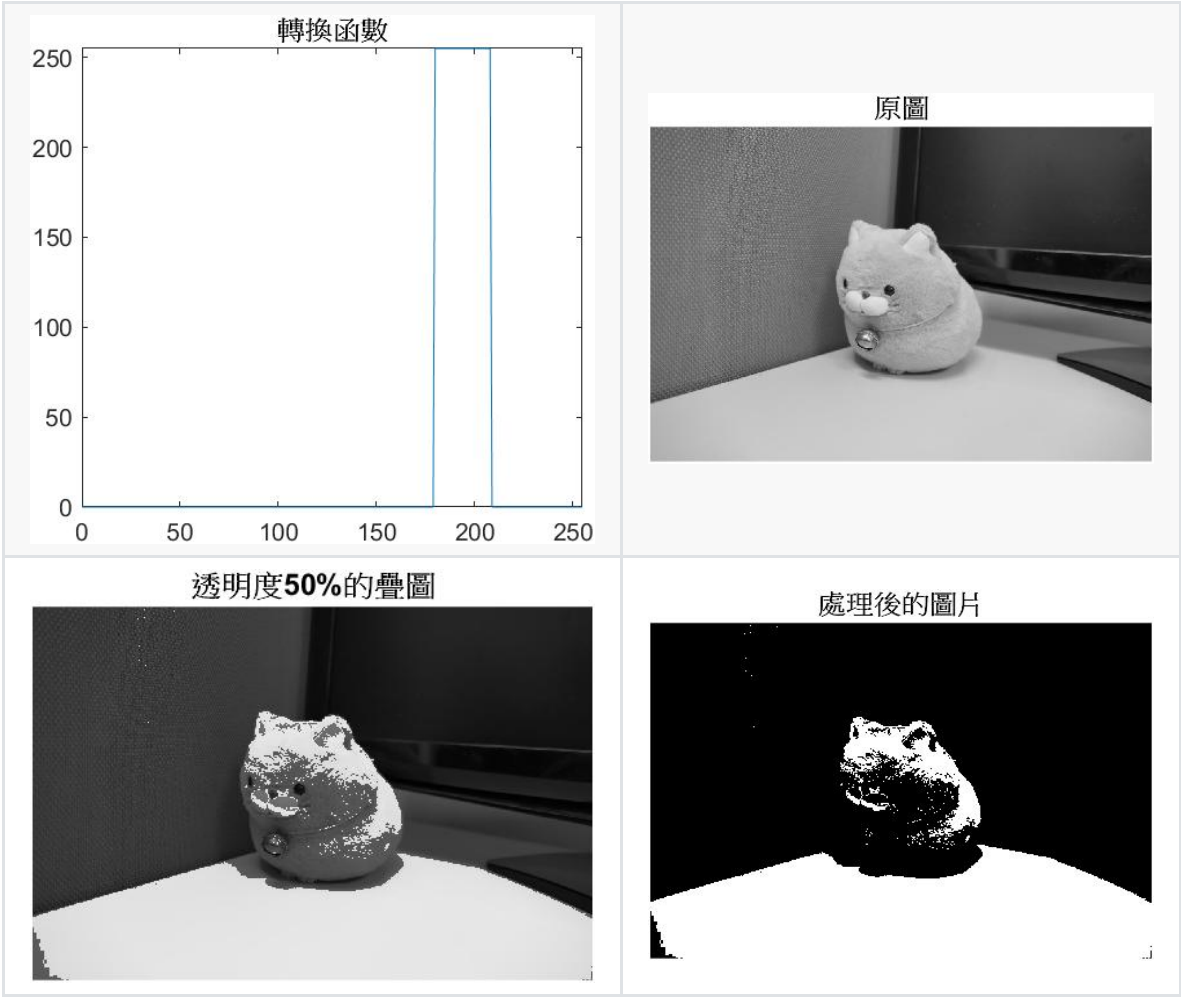
```
x = uint8(0:255);
y = x;
y(1:64) = y(1:64)/2;
y(192:255) = y(192:255)/2 + 128;
y(65:191) = uint8(double(y(65:191))/128*192 -64);

figure
%-- 畫線
subplot(2,2,1)
plot(x,y)
hold on
plot( 64,32 , 'ro' )
text( 67, 32 ,'(64,32)');
plot( 192,224 , 'ro' )
text( 195, 224 ,'(192,224)');
axis( [0,255,0,255])

%-- 原圖
subplot(2,2,2)
gimg = imread('demo2_gray.jpg');
imshow(gimg)

%-- 修改後的
subplot(2,2,4)
imshow( y(gimg+1) )
```

最後的 `gimg+1` 是因為 `matlab` 的索引值從 1 開始，可以看下面的表格(以**除以2**為例)

查詢表格的索引值 (Matlab)	1	2	3	4	5	...	253	254	255	256
原本強度	0	1	2	3	4	...	252	253	254	255
轉換後強度	0	0	1	1	2	...	126	126	127	127

強度切片-灰階

- 我們也可以只擷取特定段的強度值設成白色，其餘設成黑色。
- 擷取圖中灰階和其他差別較大的桌子

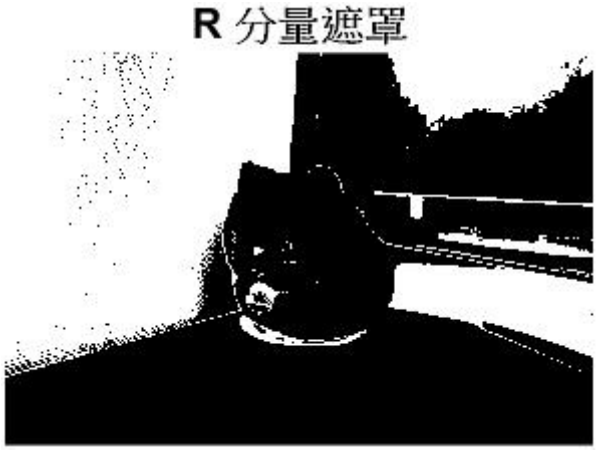
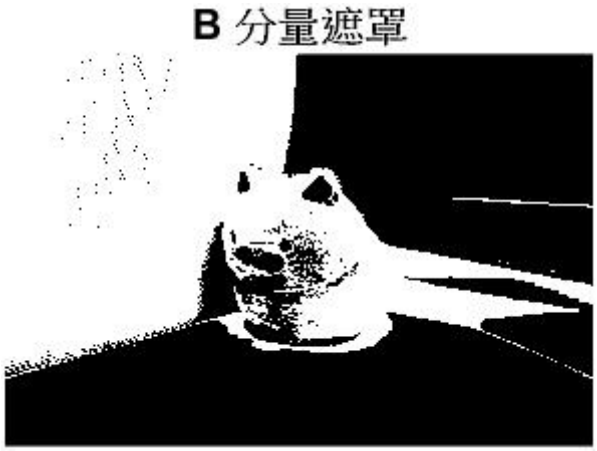



```
%-- example8 --%
%-- 製作查詢表格 y
x = uint8(0:255);
y = x;
y(1:180) = 0;
y(181:209) = 255;
y(210:end) = 0;

%-- 畫線
figure
subplot(2,2,1)
plot(x,y);
axis( [0,255,0,255] )
title('轉換函數')

%-- 讀圖
subplot(2,2,2)
gimg = imread('demo_gray.bmp');
imshow(gimg)
title('原圖')

%-- 處理後的圖片
tmp_gimg = y(gimg+1);

%-- 用透明度 50% 疊圖來看
subplot(2,2,3)
imshow( tmp_gimg.*0.5 + gimg.*0.5 )
title('透明度50%的疊圖')

```

```
subplot(2,2,4)
imshow(tmp_gimg)
title('處理後的圖片')

```

強度切片-彩色

- 從 RGB 色彩空間去擷取特定顏色，例子中擷取 R:[60:130], G:[100:180], B:[100:180]
- 相當於影像處理軟體中的魔法棒功能(選取顏色相近的區塊)，但是這個魔法棒沒有判斷邊界的功能。

..... 原圖	
R 分量遮罩	
G 分量遮罩	
B 分量遮罩	

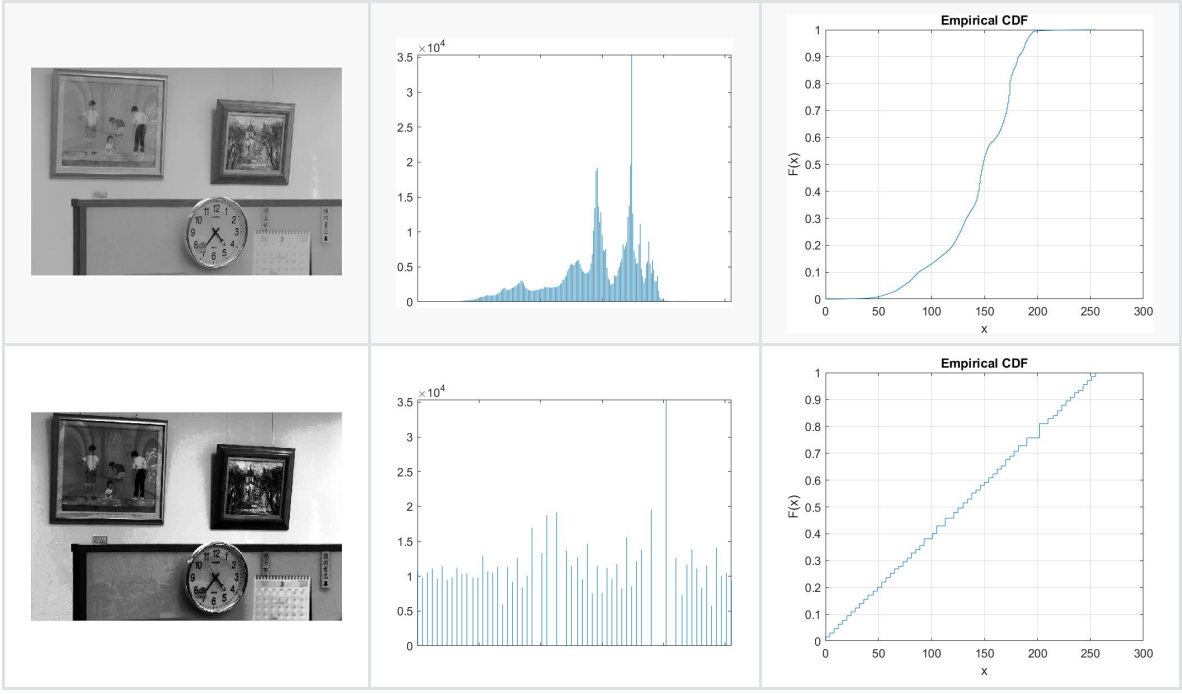
結果	

```
%-- example9 --%
%-- 製作查詢表格 Ry, Gy, By
x = uint8(0:255);

Ry = uint8(zeros(1,256));
Gy = Ry;
By = Ry;

Ry( 60:130 ) = 255;
Gy( 100:180 ) = 255;
By( 100:180 ) = 255;
cimg = imread('demo_color.bmp');
[h,w,d] = size(cimg);
mask = ones(h,w);

%-- R 分量
figure
subplot(2,2,1)
tmp = Ry( cimg(:,:,1) + 1 );

%-- 遮罩取交集
mask = mask&tmp;
imshow( tmp );
title('R 分量遮罩')

%-- G 分量
subplot(2,2,2)
tmp = Gy( cimg(:,:,2) + 1 );
mask = mask&tmp;
imshow( tmp );
title('G 分量遮罩')

%-- B 分量
subplot(2,2,3)
tmp = By( cimg(:,:,3) + 1 );
mask = mask&tmp;
imshow( tmp );
title('B 分量遮罩')

%-- 把遮罩轉成彩色圖片的格式
rgb_mask(:,:,1) = mask;
rgb_mask(:,:,2) = mask;
rgb_mask(:,:,3) = mask;

```

```
%-- 將遮罩用在原圖上
subplot(2,2,4)
cimg(~rgb_mask) = 0;
imshow(cimg)
```

直方圖等化

- 依照亮度出現的頻率，重新調整圖片的亮度分布。
- 使用 `matlab` 的 `histeq`




```
%-- example10 --%
gimg = imread('demo_gray.jpg');
figure
%-- 原圖
subplot(2,3,1)
imshow(gimg)

subplot(2,3,2)
imhist(gimg)
axis tight

subplot(2,3,3)
cdfplot(gimg(:))

%-- 做直方圖等化
ae_img = histeq(gimg);
subplot(2,3,4)
imshow(ae_img)

subplot(2,3,5)
imhist(ae_img)
axis tight

subplot(2,3,6)
```

```
25 cdfplot(ae_img(:))
```

相關資料：[wiki 直方圖等化]

對比度擴展、相同對比度調整

對比度的公式?

單閥值、雙閥值、假閥值、大津

- 將灰階圖轉成二值圖，高於門檻值(threshold)的都設為黑色，低於門檻值都設成白色。
- 直接應用在圖片上

鄰域強度轉換

- 卷積
- 相關性
- Matlab函數
- 強度轉換 (以不同的轉換函數 平移、延展、log、gamma...)
- 馬賽克、抗鋸齒？、
- 分段強度轉換
- 強度切片
- 直方圖等化
- 對比度擴展、相同對比度調整
- 單閥值、雙閥值、假閥值、大津

這章節與第四章的單點強度轉換不同的地方在於，新的像素值是由原本像素點附近像素的強度來決定。舉例來說，在一張圖片中，同樣一些具有強度 128 的像素，在第四章的單點強度轉換處理後會得到相同的結果，但在這章的鄰域強度轉換則會根據這個像素點周圍的像素強度而有所不同。

卷積(convolution)與相關性(correlation)

在鄰域強度轉換時，一個重點就是要如何將周圍的像素強度混合。這裡會介紹卷積運算與相關性運算。

卷積

在數學上，卷積(convolution) 是定義成這個樣子 $(f*g)(t)=\int_D f(\tau)g(t-\tau)d\tau$ 。

可以把它看成新函數 $(f*g)$ 某一點 (t) 的值，就是某一個範圍內 f 和 g 的乘積加總。

在二維的影像處理上，會用一個圖像以及一個遮罩來做卷積，以 3×3的來說，會定義成 $\sum_0^8 I(t)\times M(8-t)$，其中 t 的範圍 $[0,8]$ 是 3×3 的遮罩所蓋住的區域，從左至右，從上至下的編號。

*

以上例來看，就會是 $m_0\times p_8+m_1\times p_7+\ldots m_8\times p_0$ ，實際上的數字運算則會如下 (橘色方塊是兩個矩陣相乘對應的位置)。

* =

要注意的是做了卷積後，矩陣維度會變小。可以在最外圈補值(多加一圈數字，讓原本圖像維度變大)來讓處理後的矩陣維度維持不變。

另外一點則是，在圖像這裡的卷積要套上數學上的定義時，你要將 f 看作是那一小塊 3×3 區塊上的函數(若遮罩 g 是 5×5 那 f 就要看作是 5×5 區塊上的)，而非整張圖像的。

而這樣求出的結果實際上也只是 $f*g$ 這個函數的其中一點，舉例來說，上面結果的每一個方格都是由計算原本圖像中 3×3 區塊來得到，也就是 $f*g(t)=\sum_{\tau=0}^{8} f(\tau)\times g(t-\tau)$，但實際上只有計算 $t=8$ 的結果，$f*g(8)=\sum_{\tau=0}^{8} f(\tau)\times g(8-\tau)$，我們就是將這個值當作卷積的結果。

因為這樣子計算的緣故，原本卷積應該具有的性質(例如說結合律，對一張圖做兩次卷積不能先把兩個遮罩合成)就消失了，所以在影像處理上希望套用上數學上卷積所具有的性質時，要再次驗證性質還存不存在。

另外有一個類似結合律的性質，其中 $*$ 是卷積，而 $\times$ 是矩陣乘法。

$$(A * \begin{bmatrix}1\\2\\1\end{bmatrix}) * [1 \quad 2 \quad 1] = A * (\begin{bmatrix}1\\2\\1\end{bmatrix} \times [1 \quad 2 \quad 1]) = A * \begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$$

相關性

另外一個與卷積相似的運算是**相關性運算**，他的算法很直觀，同樣以 3×3 為例，他的計算是 $\sum_{0}^{8} I(t) \times M(t)$。假設有一個圖像 I 與遮罩 M ，你可以把 M 順(逆)時針旋轉 180° 得到另一個矩陣 N，然後 I 對 M 做卷積的結果正好會是 I 對 N 做相關性的結果，可以看下面的例子。

$*$ =

總結來說，卷積和相關性運算差別在就遮罩有沒有旋轉 180° ，而在遮罩旋轉後不會改變的情況下，兩者計算的結果更是完全一樣，很多情況下不會特別區分這兩種(把兩種都叫做卷積)。

在實作上要做卷積的話，可以先將遮罩旋轉 180° 然後實作簡單的相關性運算(雖然差異也就在寫迴圈時的下標不同而已)。

Matlab 函數

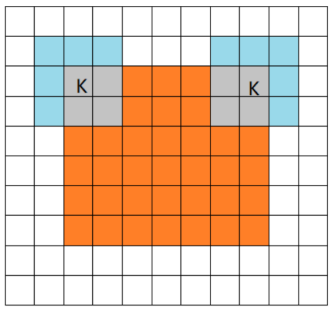
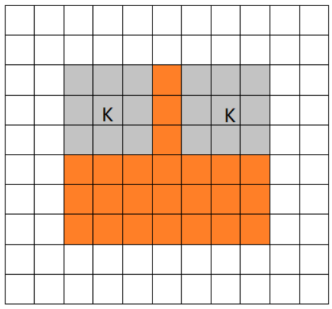
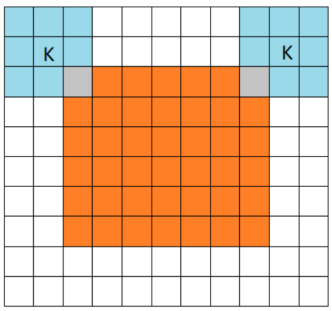
在 Matlab 中可以使用 `filter2` 和 `conv2` 來做相關性與卷積計算。

`filter2(mask,I)`

```
%-- example1 --%
gimg = imread('demo_gray.bmp');
gimg = im2double( gimg );

L = 21;
mask = ones(L);
mask = mask / (L*L);

ae = filter2( mask , gimg);
imshow(ae)
```

會發現邊緣出現了一圈黑影，因為 `filter2` 預設處理前後的照片大小會相同，會在外圍補 0 。

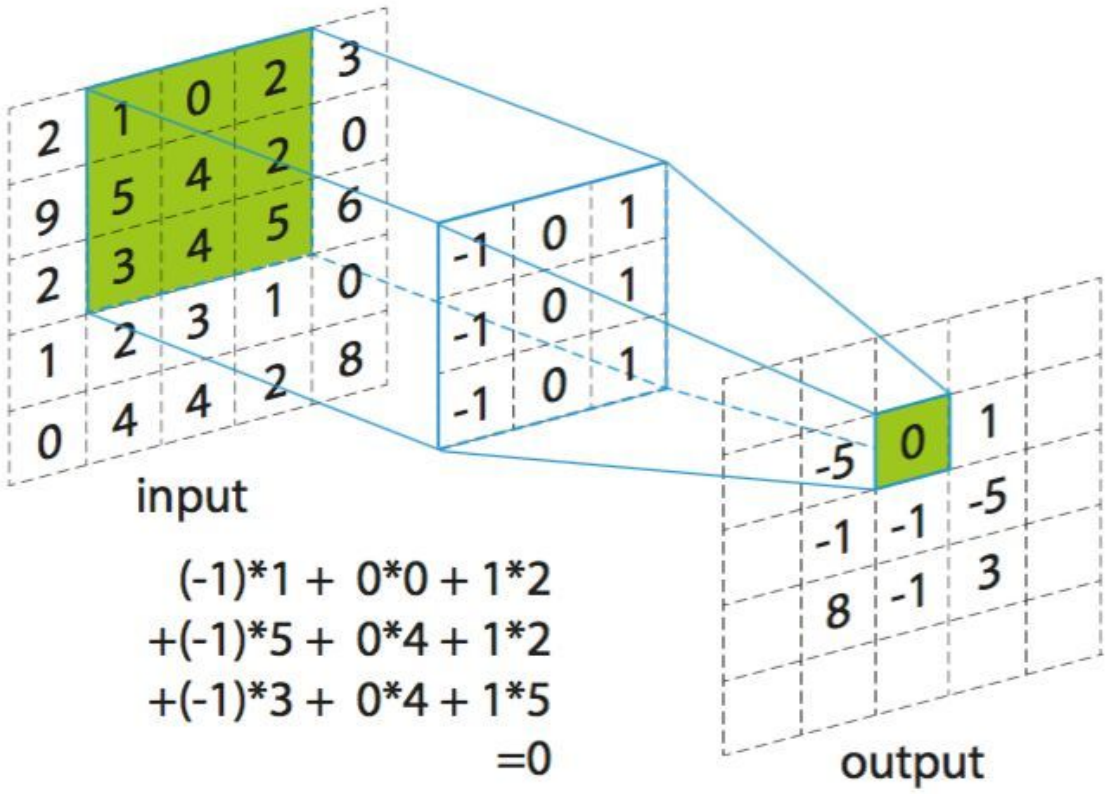
有三種設定邊緣的方式：`same`，`valid`，`full`，用法是 `filter2(M,I,'vaild')`

same(default)	valid	full
		

`conv2` 也是相同的用法，不過預設的參數是 `'full'`

線性濾波器

卷積(摺積、疊積、旋積、convolution)



在數學上，卷積(convolution) 是定義成這個樣子 $(f*g)(t) = \int_D f(\tau)g(t-\tau)d\tau$ 。

可以把它看成新函數 $(f*g)$ 某一點 (t) 的值，定義成某一個範圍內 f 和 g 的乘積加總。

舉例來說，如果今天 f 和 g 是下列這樣的數列。

f	1	5	3	2	7	3	5
g	6	5	1	3	0	2	0

那你就可以用這樣的方法算一種卷積 (設定一個積分的範圍)。

f	1	5	3	2	7	3	5
g	6	5	1	3	0	2	0
$f\ast g$	-	1*1 + 5*5 + 3*6	-	-	-	-	-
...							
$f\ast g$	-	44	28	13	13	6	-

附上 `c++` 計算的程式碼。

```
#include<iostream>
using namespace std;
int main(){
    int a[]= {1,5,3,2,7,3,5};
    int b[]= {6,5,1,3,0,2,0};
    int L = 7;
    for( int i=1;i<L-1;++i ){
        int s = 0;
        for(int j=-1;j<=1;j++){
            s += a[i+j]*b[i-j];
        }
        cout << s << " ";
    }
}
```

當然也可以有這種算法。

f	1	5	3	2	7	3	5
g	6	5	1	3	0	2	0
$f\ast g$	-	-	1*0 + 5*3 + 3*1 + 2*5 + 7*6	-	-	-	-
...							
$f\ast g$	-	-	70	38	18	-	-

或者是這樣的做法，固定其中一個的範圍。

f	1	5	3	2	7	3	5
g	6	5	1	-	-	-	-
$f\ast g$	-	1*1 + 5*5 + 3*6	5*1 + 3*5 + 2*6	-	-	-	-
...							
$f\ast g$	-	44	32	55	55	52	-

這些運算的目的都是把 f 的每一個範圍的值混合在一起，把周圍的資訊用一個值表達出來。

例如說局部平均，(改變數字就變成加權平均)

f	1	5	3	2	7	3	5
g	1/3	1/3	1/3	-	-	-	-
$f\ast g$	-	1*1/3 + 5*1/3 + 3*1/3	5*1/3 + 3*1/3 + 2*1/3	-	-	-	-
...							
$f\ast g$	-	3	3.3	4	4	5	-

例如說一階梯度(微分)， $f'(t) = f(t+1) - f(t)$

f	1	5	3	2	7	3	5
g	1	-1	-	-	-	-	-
$f\ast g$	-	1*(-1) + 5*1	-	-	-	-	-
...							
$f\ast g$	-	4	-2	-1	5	-4	2

例如說一階梯度(微分) (另一種版本)， $f'(t) = (f(t+1) - f(t-1))/2$

f	1	5	3	2	7	3	5
g	1/2	0	-1/2	-	-	-	-
$f\ast g$	-	1*(-1/2) + 5*0 + 3*1/2	-	-	-	-	-
...							
$f\ast g$	-	1	-1.5	2	0.5	-1	-

上列這些都是(離散版本)卷積的例子。

卷積也有一些好的性質 (要滿足前面的卷積定義)：

Matlab 實作

在 `matlab` 中可以使用 `imfilter` 來對圖像做卷積。

平均濾波(模糊)

使用 15*15 的平均遮罩來對圖像濾波。

Figure 2

File Edit View Insert Tools Desktop Window Help

```
%-- example1 --%
cimg = imread('demo_color.bmp');
gimg = rgb2gray(cimg);

mask = fspecial( 'average' , [15,15] );
img = imfilter( gimg , mask );

figure()
subplot(2,1,1)
imshow(gimg)
subplot(2,1,2)
imshow(img)
```

高斯濾波

使用高斯分布的遮罩來濾波。

Figure 1

File Edit View Insert Tools Desktop Window Help

```
%-- example2 --%
gimg = imread('gray_demo.jpg');

mask = fspecial( 'gaussian' , [15,15] , 1.2 );
img = imfilter( gimg , mask );

figure()
subplot(2,1,1)
imshow(gimg)
subplot(2,1,2)
imshow(img)
```

邊界濾波

在影像處理任務中，常常會需要捕捉物體的邊界，這裡會介紹找出邊界的方式。

這裡的邊界是指，兩個亮度相差超過一定程度的區塊交界。

所以找出邊界，就相當於計算梯度，並找出梯度絕對值夠大的地方。

實際有各式各樣找邊界的方式，例如說用不同的遮罩，或先做一些前處理(做模糊去雜訊等等)。

以 Sobel 遮罩 和 Prewitt 遮罩來說，

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

及

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Sobel 就比 Prewitt 能找出更多方向的邊界，Prewitt 則專注在找出水平和垂直的邊界。
在不同的任務場景，會選用不同的遮罩來處理。

而 Laplacian 遮罩則像是影像強化，他會計算一個像素與周圍像素的差距，那就可以用來模糊邊緣或是強化邊緣。

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Sobel

使用

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

遮罩來濾波 (找出垂直方向變化劇烈的部分)。

轉置

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

遮罩來濾波 (找出水平方向變化劇烈的部分)。

合在一起可以找出圖像的邊緣(輪廓)。

```
%-- example3 --%
gimg = imread('gray_demo.jpg');
gimg = rgb2gray(gimg);
gimg = im2double(gimg);

mask = fspecial( 'sobel'  );
img = imfilter( gimg , mask );

a = imfilter( gimg , mask );

b = imfilter( gimg , mask' );

th = 0.3;

img = abs(a)>th | abs(b)>th;

figure()
imshow(gimg)
figure()
imshow( img )

```

Prewitt

與 Sobel 相似，也可以使用

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

遮罩來找邊緣。

Roberts

也有

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

與

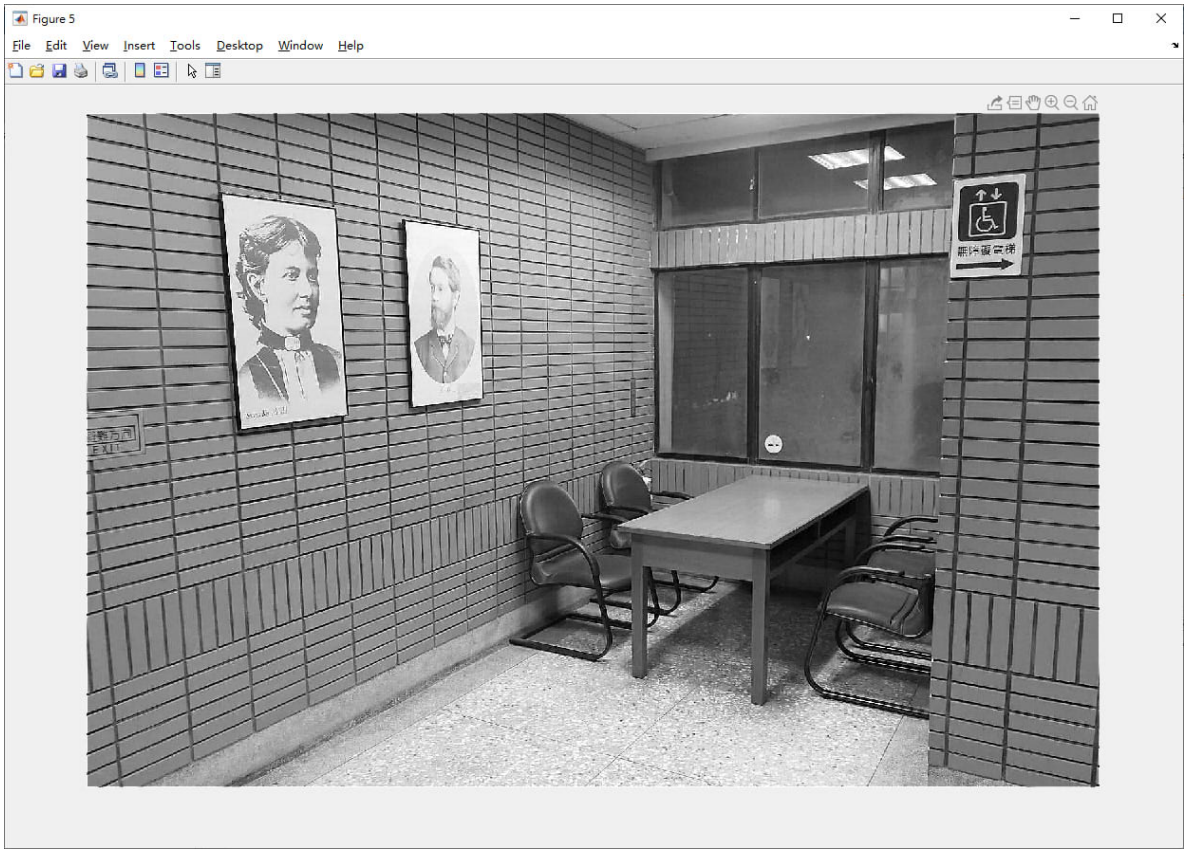
$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

來找邊界的方法。

Laplacian 濾波(影像強化)

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

可以用來(鈍化)強化邊緣(影像銳利化)。



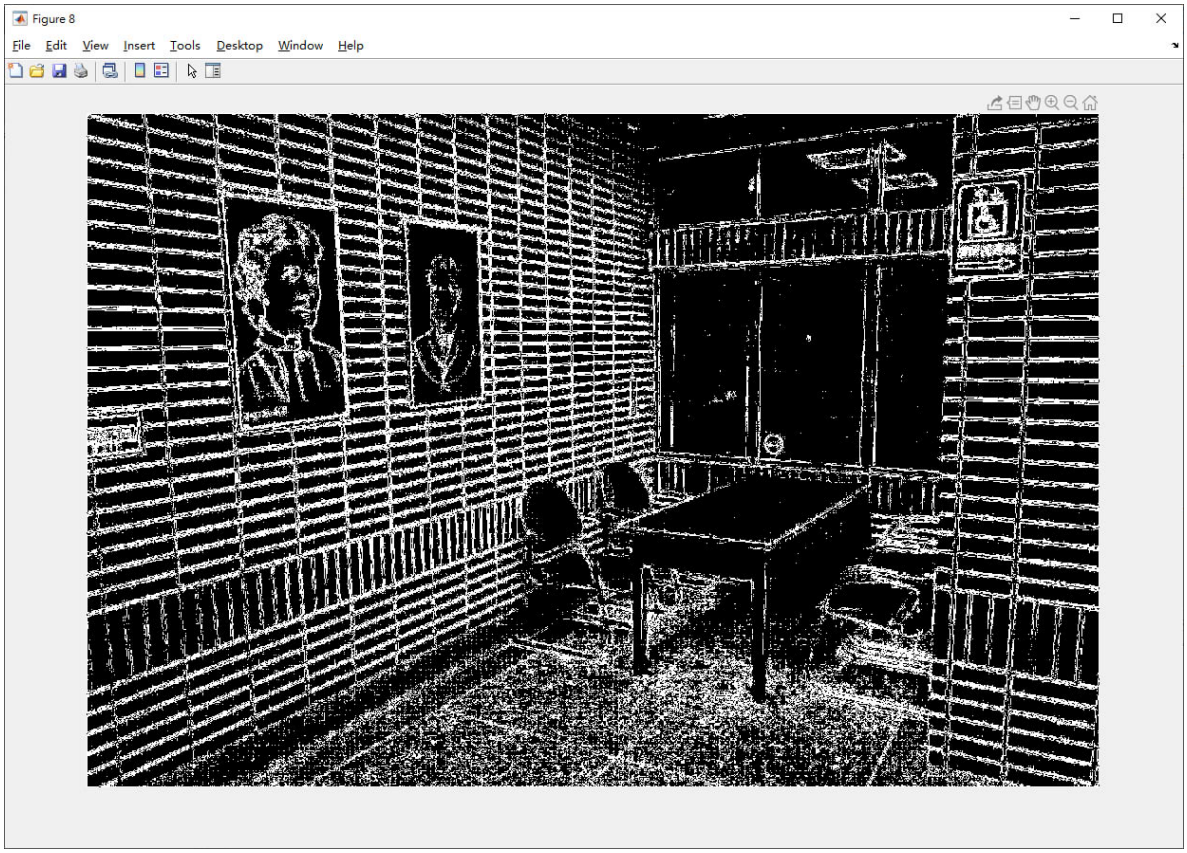
```
%-- example4 --%
gimg = imread('gray_demo.jpg');
gimg = rgb2gray(gimg);
gimg = im2double(gimg);

mask = fspecial( 'laplacian' , 0)
img = imfilter( gimg , mask );

figure()
imshow(gimg)
figure()
imshow( gimg - img )
```

其中 `fspecial( 'laplacian' , 0)` 的第二個參數介於 $[0, 1]$，調整不同的銳利化程度。

Laplacian 濾波(邊界擷取)

如同上面所說，Laplacian 是用來計算與周圍像素的差距，所以也可以直接拿這個值作為邊界的挑選基準。

但會有一個問題，如果 Laplacian mask 中心點的像素是雜訊(原本就和周圍相差很大的)，那這個像素的值就會遠比其它一般的值放大更多倍(所以對雜訊敏感)。

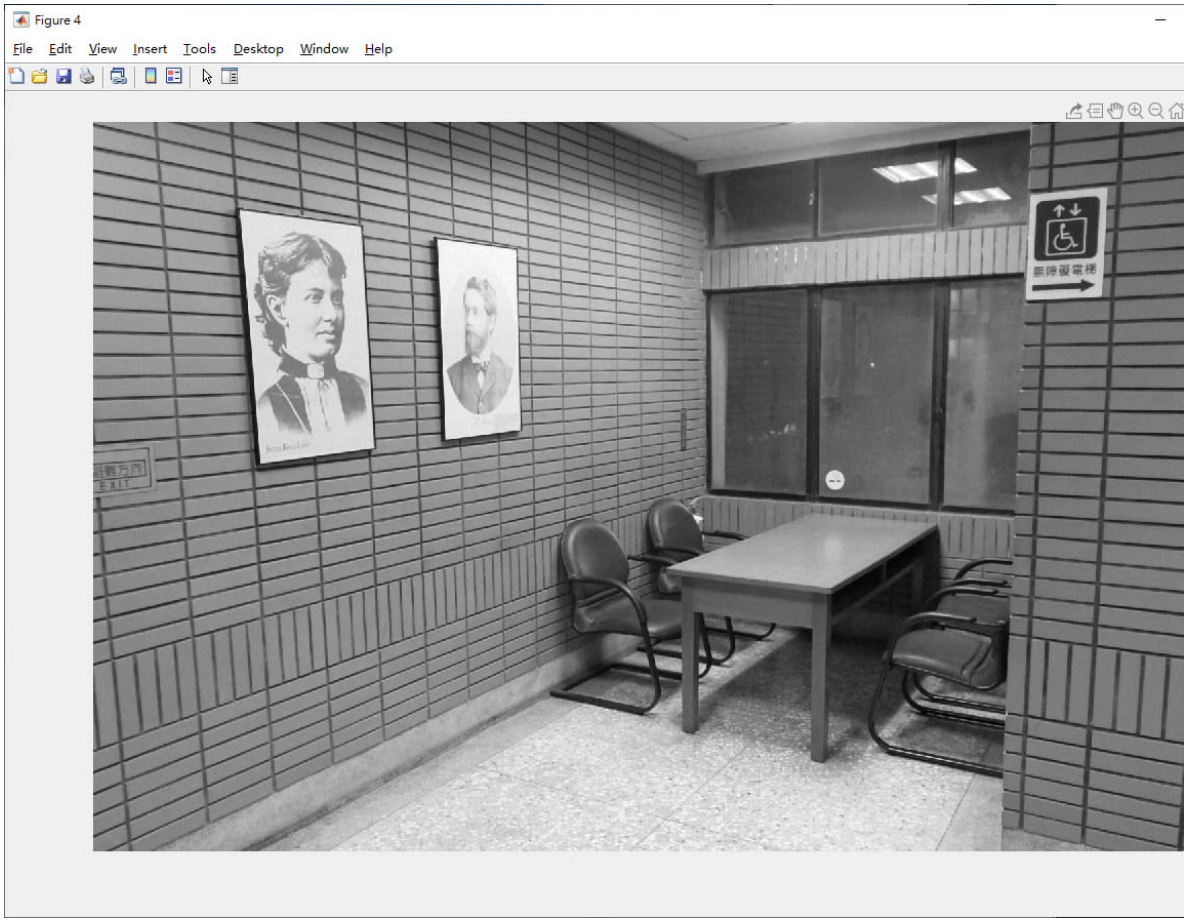


```
%-- example5 --%
gimg = imread('gray_demo.jpg');
gimg = rgb2gray(gimg);
gimg = im2double(gimg);

mask = fspecial( 'laplacian' , 0)
img = imfilter( gimg , mask );

figure()
imshow(gimg)
figure()
th = 0.05
imshow( abs(img)>th )
```

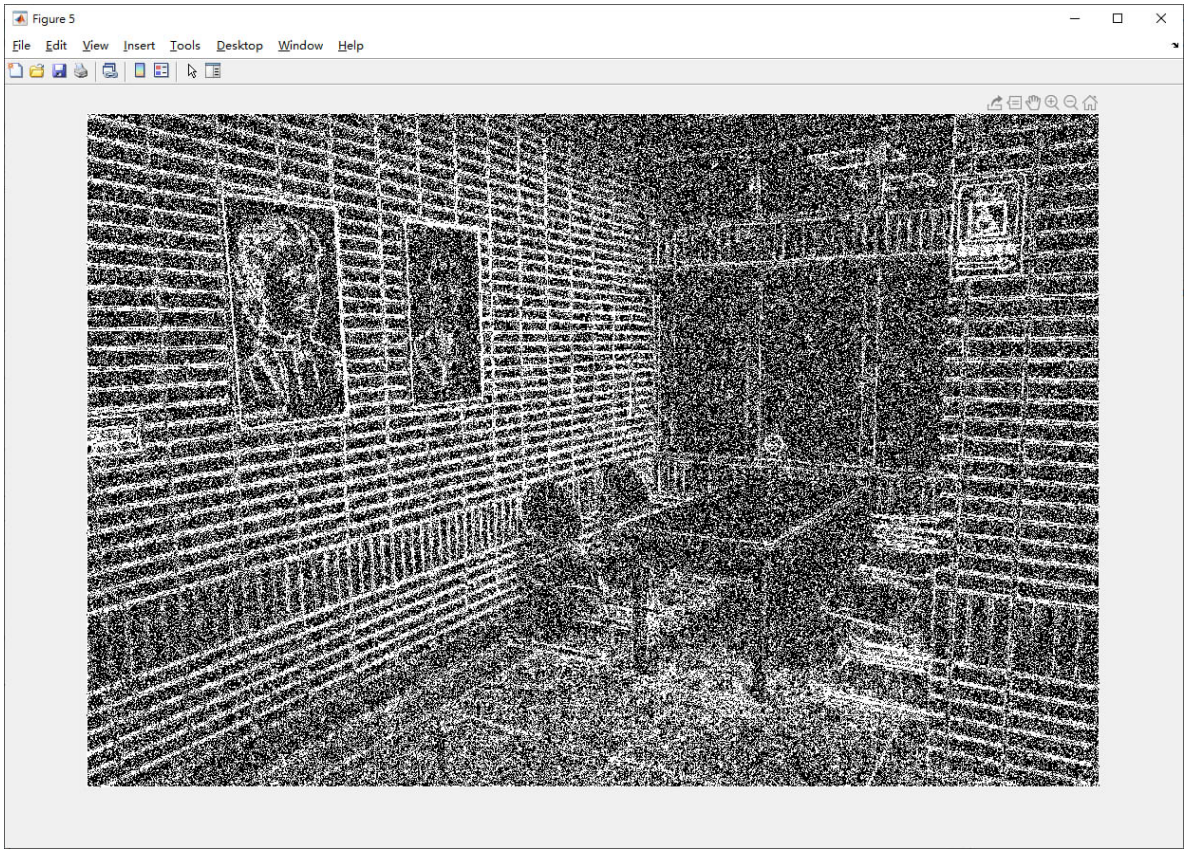
對比加入雜訊的。
下圖是添加了很微弱的高斯雜訊的圖片。

原圖：

加了雜訊 $(\mu, \sigma) \sim (0, 0.01)$ (圖像值介在 [0,1]) (imnoise 的參數是 σ^2)：



邊界處理後的

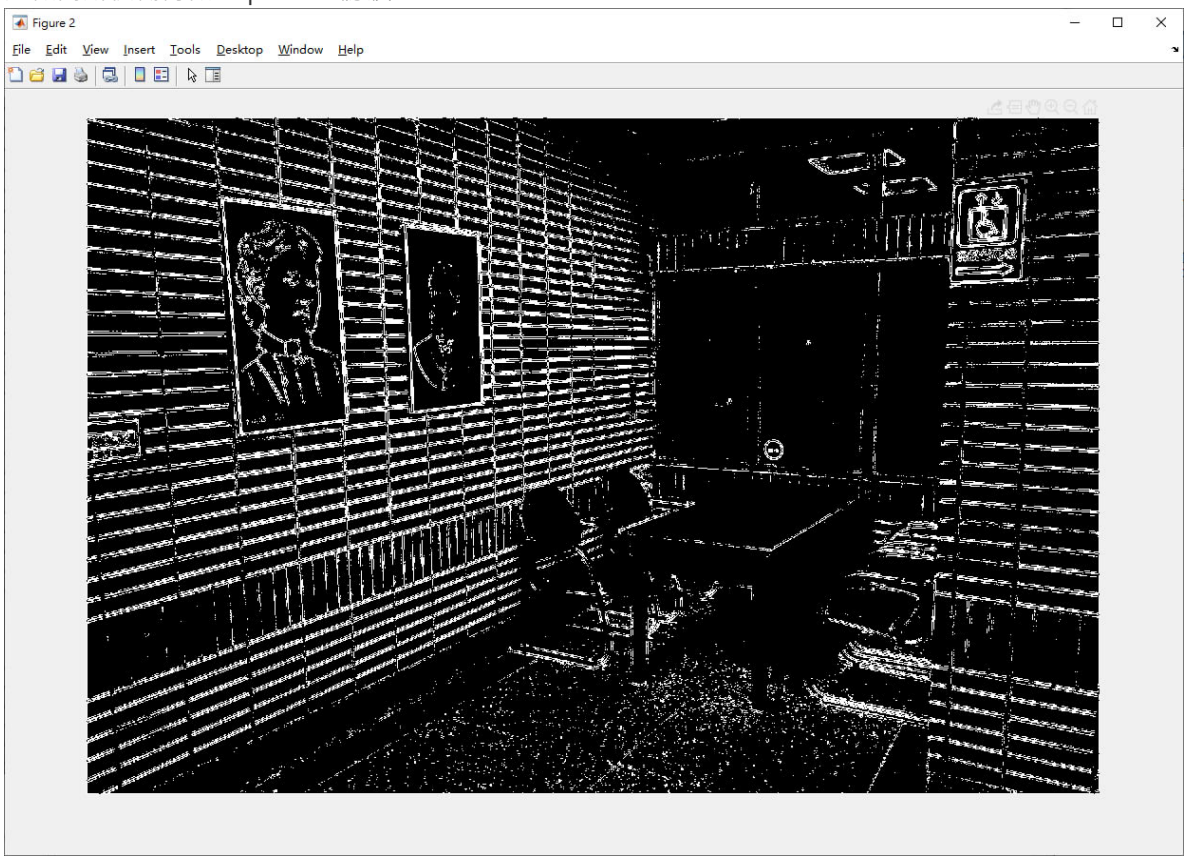



```
%-- example6 --%
gimg = imread('gray_demo.jpg');
gimg = rgb2gray(gimg);
gimg = im2double(gimg);

```

```
gimg = imnoise( gimg , 'gaussian' , 0 , 0.0001 );

mask = fspecial( 'laplacian' , 0)
img = imfilter( gimg , mask );

figure()
imshow(gimg)
figure()
th = 0.05
imshow( abs(img)>th )
```

所以一般拿 Laplacian 在做濾波時，通常會先做高斯模糊來去雜訊。

先做高斯模糊再做 Laplacian 濾波。

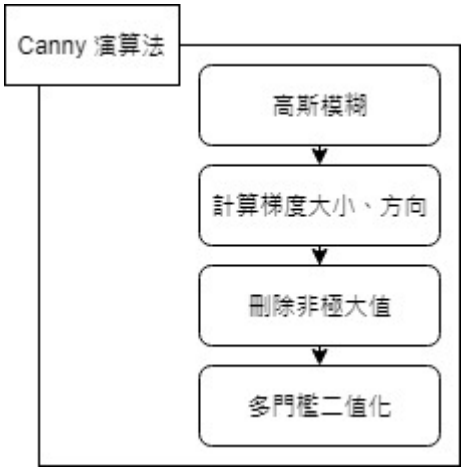



```
%-- example7 --%
gimg = imread('gray_demo.jpg');
gimg = rgb2gray(gimg);
gimg = im2double(gimg);

gimg = imnoise( gimg , 'gaussian' , 0 , 0.0001 );

mask = fspecial( 'gaussian' , [3,3] ,1.2)
gimg = imfilter( gimg , mask );

mask = fspecial( 'laplacian' , 0)
img = imfilter( gimg , mask );

figure()
imshow(gimg)
figure()
th = 0.05
imshow( abs(img)>th )
```

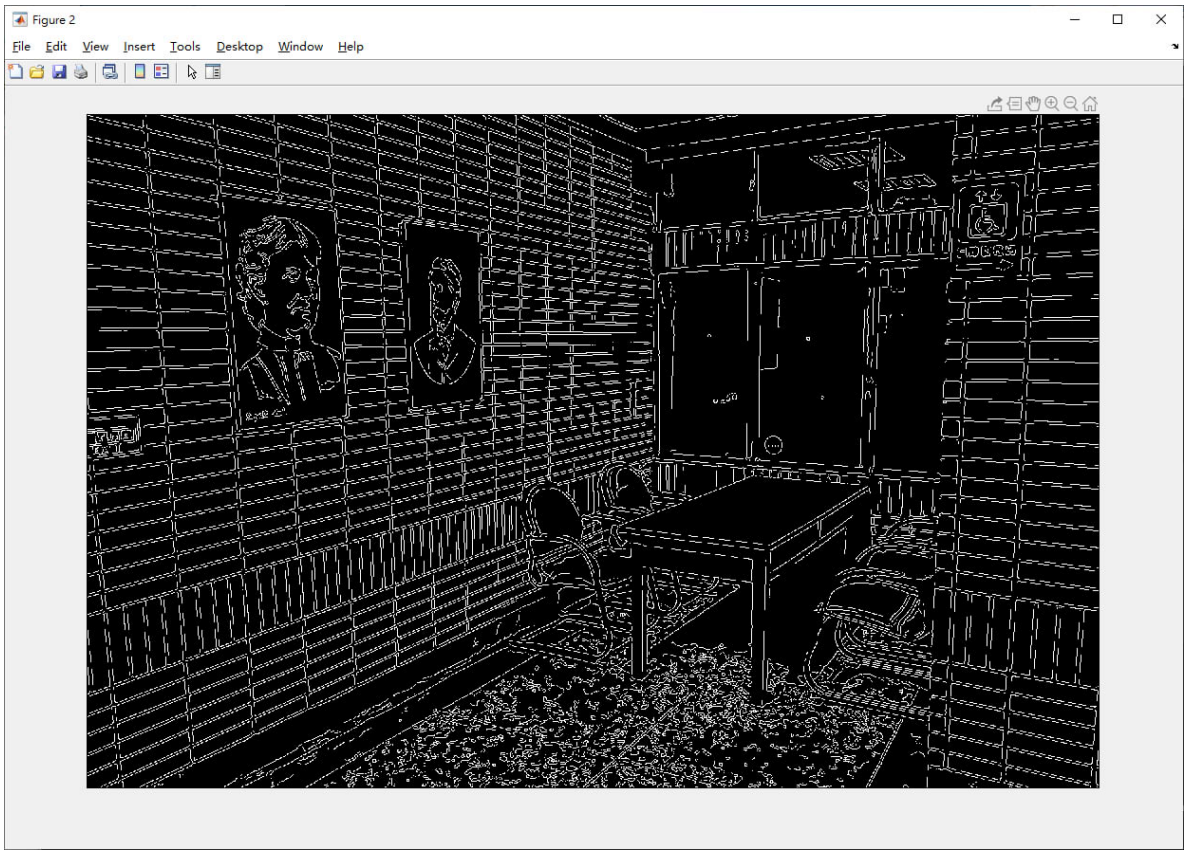
Canny

另外還有一些做更多前處理的邊界處理方法，例如 Canny。

有四個步驟



在 `matlab` 裡可以使用 `edge` 函數來完成。




```
%-- example8 --%
gimg = imread('gray_demo.jpg');
gimg = rgb2gray(gimg);

img = edge(gimg,'canny');

figure()
imshow(gimg)
figure()
imshow( img )
```

前面提到的 Sobel , Prewitt 等也可以使用 `edge(img,'sobel')` 來完成。

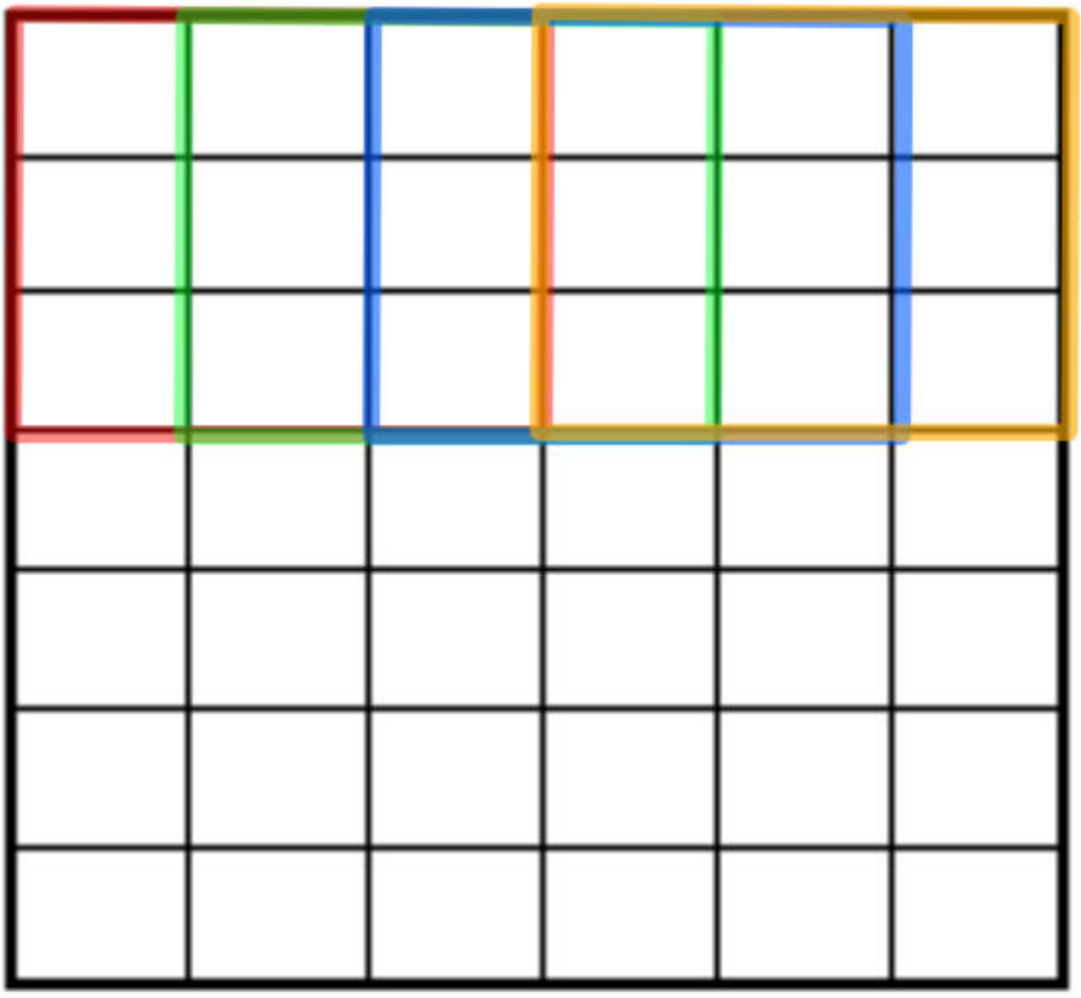
非線性濾波器

非線性濾波器

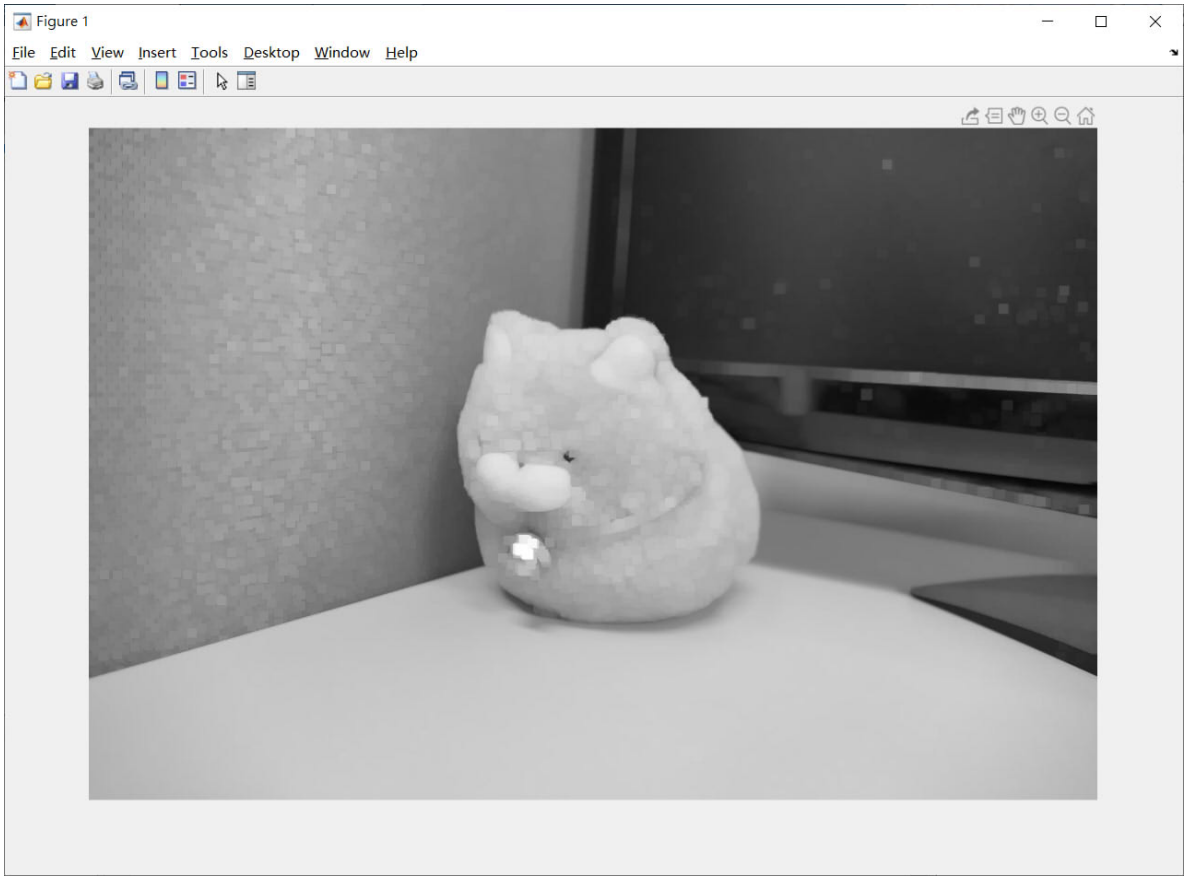
- 將遮罩下的元素依大小排列，並選擇特定順序的元素作為輸出。
- 在 Matlab 中使用 `nlfilter` 函數，速度慢
- 三個參數為：圖像矩陣、矩陣大小、呼叫的函數

```
%-- example1 --%
gimg = imread('demo_gray.bmp');

%-- 圖像矩陣、遮罩大小、使用的函數
img_max = nlfilter( gimg , [3,3] , 'max(x(:))' );
figure()
imshow(img_max)
```

- 或者使用 `colfilt`，佔用比較多記憶體，但速度快很多
- 四個參數為：圖像矩陣、矩陣大小、濾波器的運作模式、呼叫的函數
- 關於濾波器的運作模式有 `'sliding'` 與 `'distinct'` 兩種，其中 `'sliding'` 的運作如下圖。

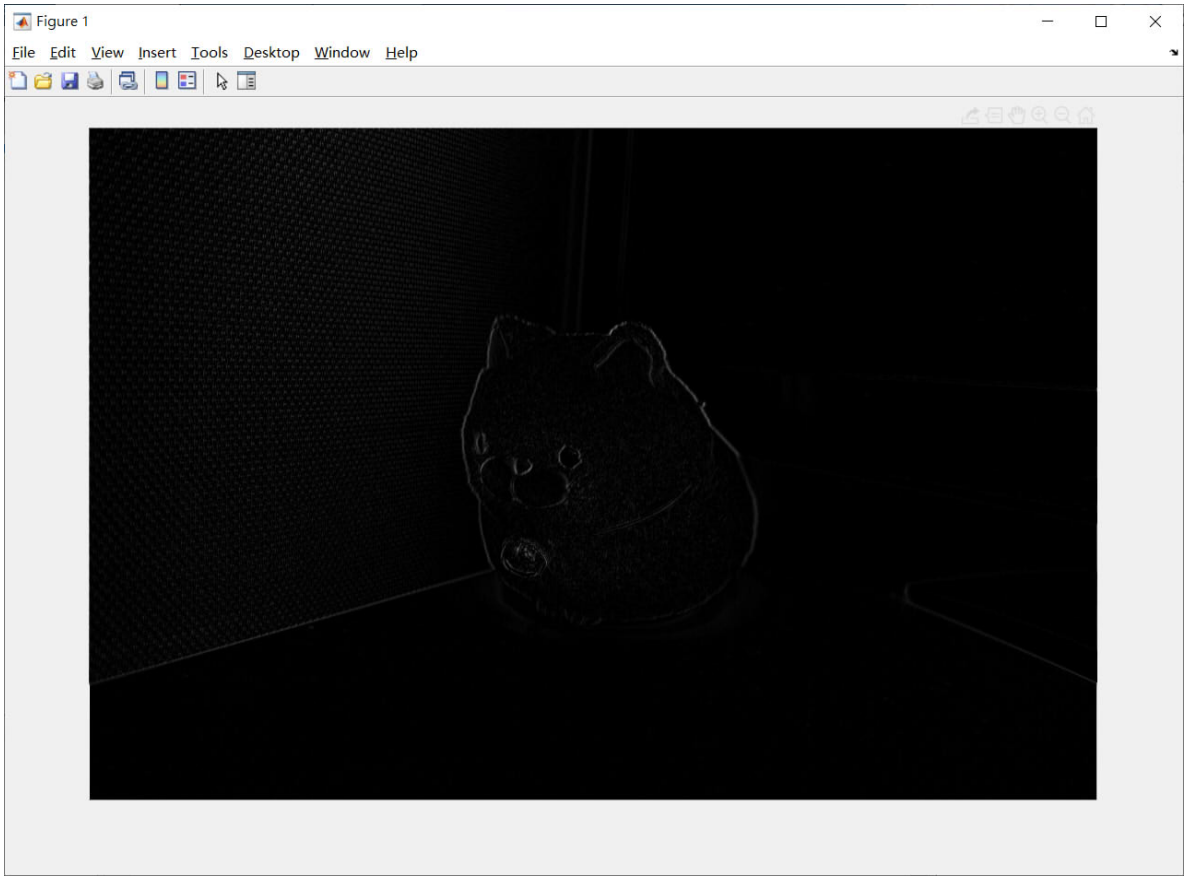


```
%-- example2 --%
gimg = imread('demo_gray.bmp');

%-- 圖像矩陣、遮罩大小、遮罩形式、使用的函數
img_max = colfilt( gimg , [10,10] , 'sliding' , @max );
figure()
imshow(img_max)

```



- 也可以使用自定義的函數

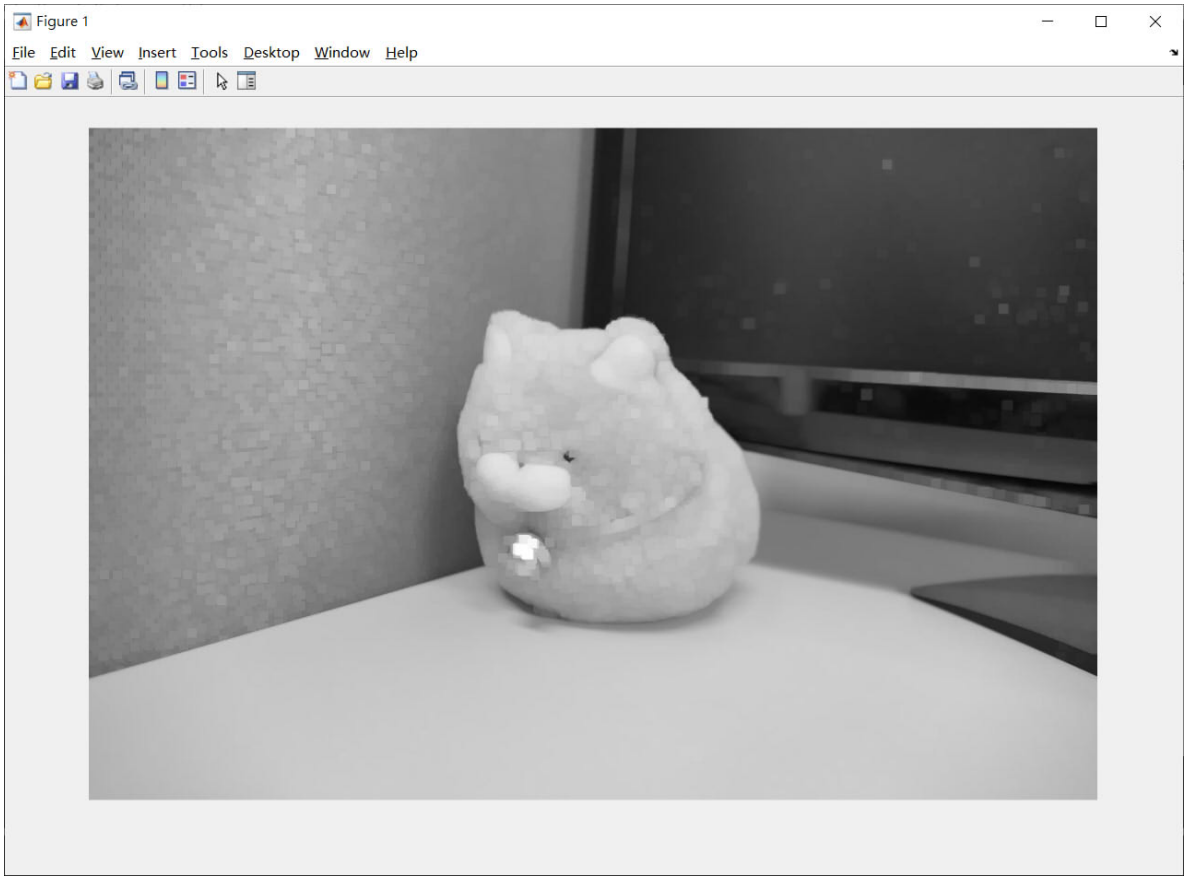
```
%-- example2_1 --%
gimg = imread('demo_gray.bmp');

%-- 圖像矩陣、遮罩大小、遮罩形式、使用的函數
img_max = colfilt( gimg , [3,3] , 'sliding' , @my_fun );
figure()
imshow(img_max)

function v = my_fun(A)
%-- colfilt 會將整個遮罩的元素排成一列，3*3 會排成 9*1，然後再傳入這個函數
%-- r = M*N , c 則會根據 colfilt 來演算法來決定大小
[r c] = size(A);
v = zeros(1, c);

for i = 1:c
    b = A(:,i);
    %-- 根據周遭顏色的相差程度，來轉換中心像素的亮度。
    v(i) = b(5)*( abs( mean(b(1:3)) - mean(b(6:9)) )/128  )  ;
end
v = uint8(v);
end
```



排序濾波器

- 可以從遮罩的元素中取出排序第 n 位的元素作為輸出。
- matlab 使用 `ordfilt2`
- 可以用來做 最大值、最小值、中值濾波

```
%-- example3 --%
gimg = imread('demo_gray.bmp');

%-- 圖像矩陣、選取的順序、權重遮罩
img_max = ordfilt2( gimg , 100 , ones(10,10) );
figure()
imshow(img_max)
```



- 中值濾波可以用於清除椒鹽雜訊
- 可以使用 `ordfilt2` 或是 `medfilt2` 來做
- `medfilt2(gimg,[3,3])`

```
%-- example4 --%
gimg = imread('demo_gray.bmp');
%-- 預設有 10% 的像素被修改
tmp = imnoise( gimg , 'salt & pepper' );
figure()
imshow(tmp)

%-- 圖像矩陣、選取的順序、權重遮罩
img_med = ordfilt2( gimg , 5 , ones(3,3) );
figure()
imshow(img_med)
```

Figure 1

File Edit View Insert Tools Desktop Window Help

Figure 2

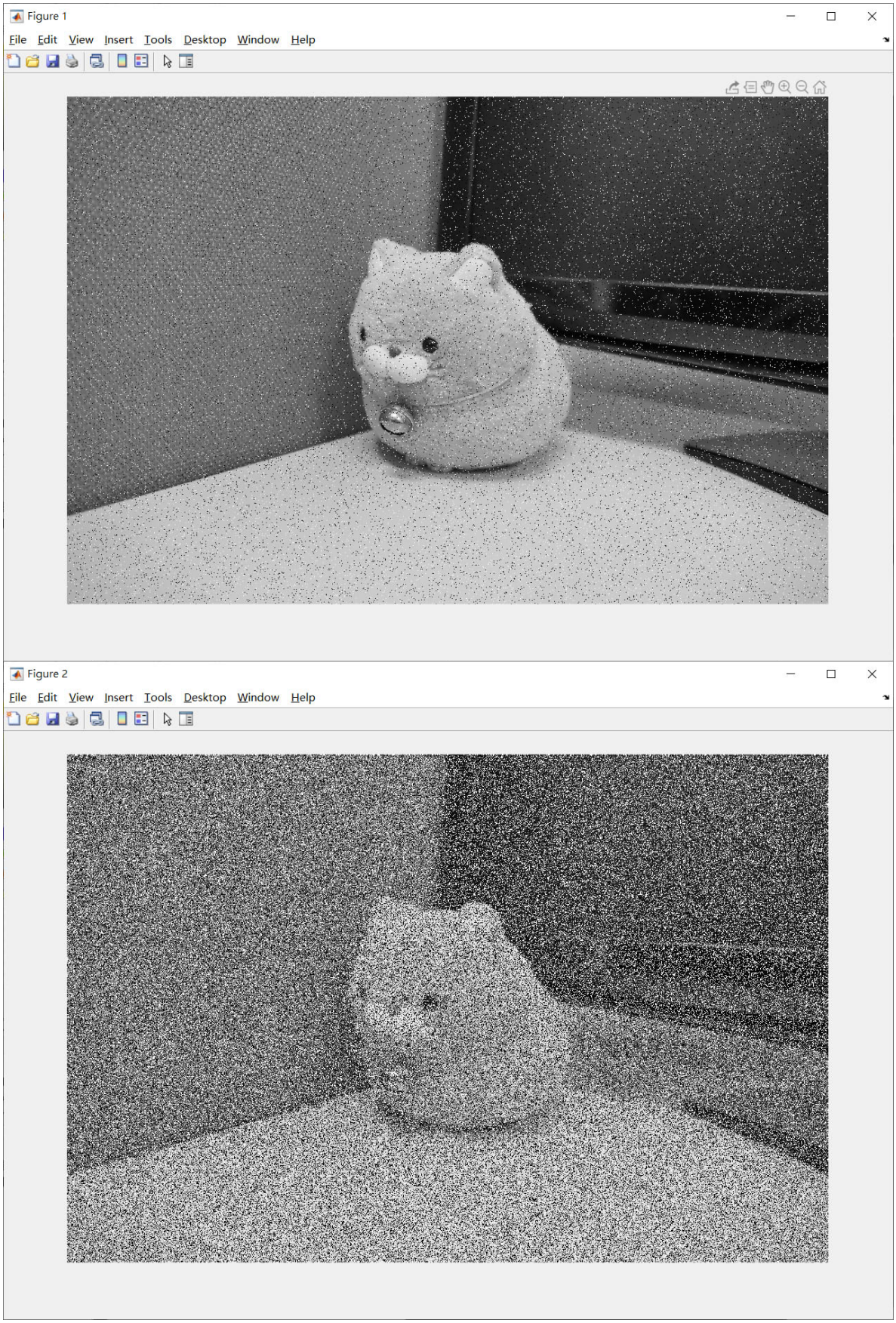
File Edit View Insert Tools Desktop Window Help

更多的雜訊 (全域)

椒鹽雜訊

- 隨機將像素點設成黑色或白色，使用內建函數 `imnoise` ，其中參數是代表有多少比例的像素點被修改。

```
%-- example1 --%
gimg = imread('demo_gray.bmp');
%-- 預設有 10% 的像素被修改
tmp = imnoise( gimg , 'salt & pepper' );
figure()
imshow(tmp)

%-- 加上參數
tmp = imnoise( gimg , 'salt & pepper' , 0.5);
figure()
imshow(tmp)
```

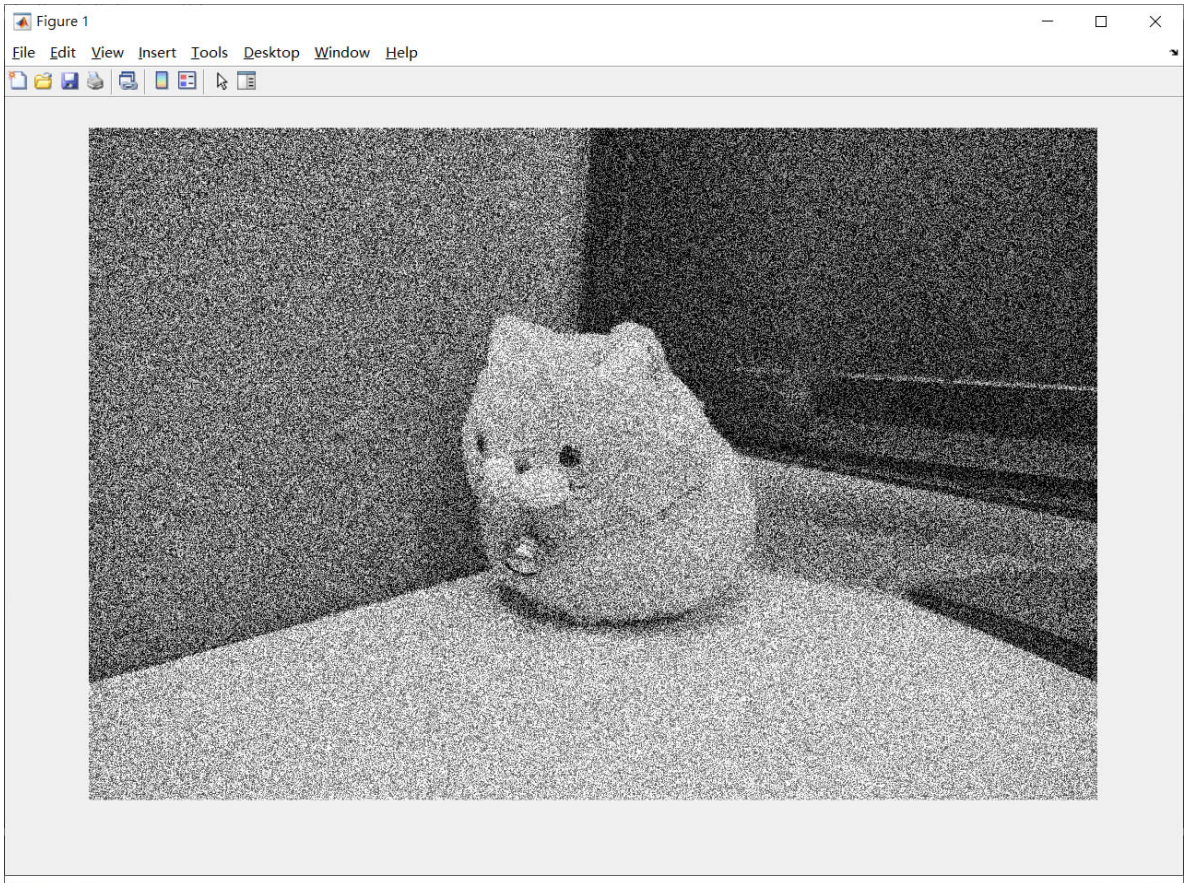


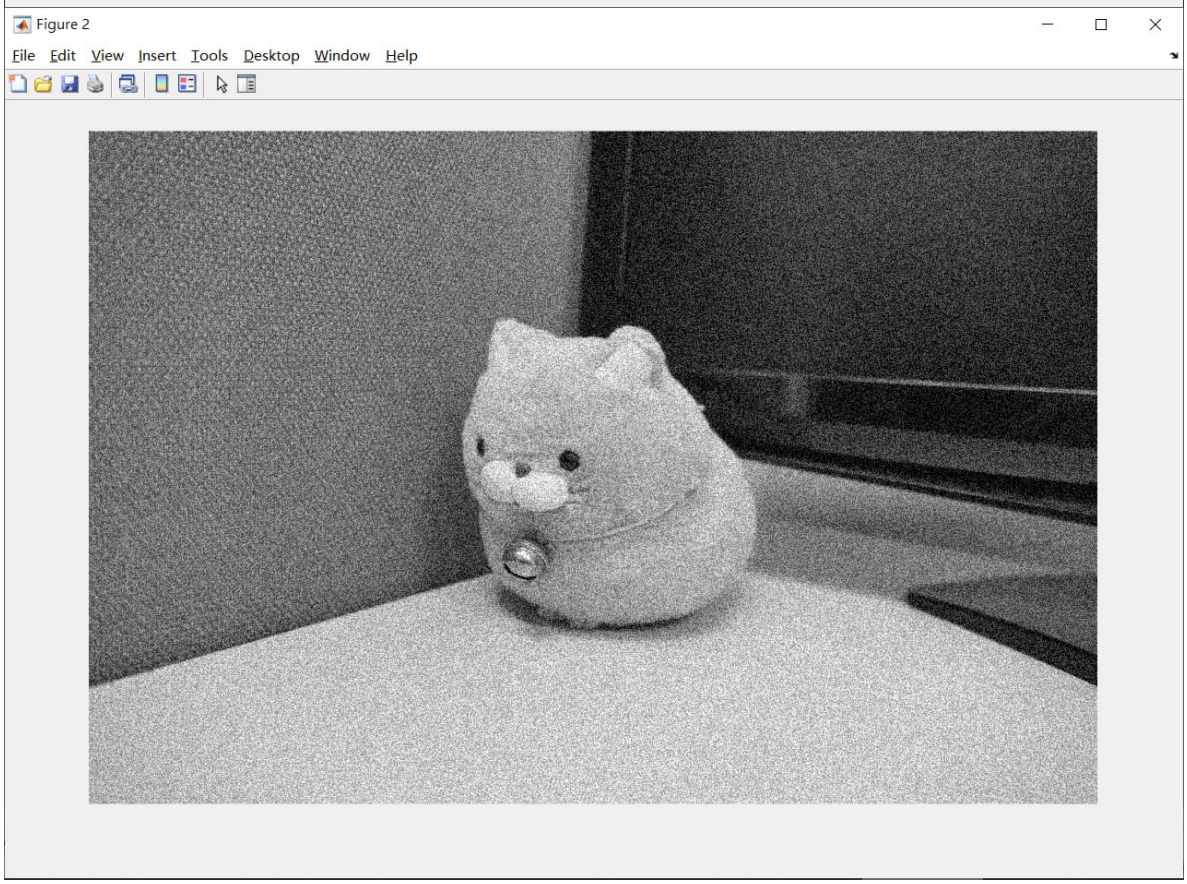
隨機雜訊

- 將整張圖片每個像素的亮度，隨機增加或減少一些，增加或減少的量呈均勻分布。

```
%-- example2 --%
gimg = imread('demo_gray.bmp');

```

```
%-- 轉換到 [0,1]
gimg = im2double(gimg);

%-- 生成在 [-0.5,0.5] uniform 分布的雜訊，
noise = rand( size(gimg) ) - 0.5;

%-- 添加上雜訊
tmp = gimg + noise;
figure()
imshow(tmp)

%-- 生成在 [-0.2,0.2] uniform 分布的雜訊，
noise = rand( size(gimg) )*0.4 - 0.2;

%-- 添加上雜訊
tmp = gimg + noise;
figure()
imshow(tmp)
```





高斯雜訊

- 將整張圖片每個像素的亮度，隨機增加或減少一些，增加或減少的量呈高斯分布。
- 多次獨立同分布的隨機雜訊疊加的結果會變成高斯雜訊 (因為獨立同分布均勻分布的相加會變成高斯分布)。
- 這裡的範例的 uniform 分布雜訊的範圍 [a,b] 有特別計算過，所以看起來兩張圖會很相似。

```
%-- example3 --%
gimg = imread('demo_gray.bmp');

%-- 轉換到 [0,1]
gimg = im2double(gimg);

%-- 使用內建的函數 imnoise 預設 N~(0,0.01)
tmp = imnoise( gimg , 'gaussian' );
figure()
imshow(tmp)

figure()
%-- 將雜訊造成的差距描繪出來
subplot(1,2,1)
imhist( tmp-gimg )

%-- 向右平移來顯示全部
subplot(1,2,2)
imhist( tmp-gimg + 0.5 )

%-- 重複疊加 10 次隨機雜訊
tmp = gimg;
for i=1:10
    %-- 生成在 [-0.05,0.05] uniform 分布的雜訊，
    noise = rand( size(gimg) )*0.1 - 0.05;

    %-- 添加上雜訊
    tmp = tmp + noise;
end
figure()
imshow(tmp)

figure()
%-- 將雜訊造成的差距描繪出來
subplot(1,2,1)
imhist( tmp-gimg )

%-- 向右平移來顯示全部
subplot(1,2,2)
imhist( tmp-gimg + 0.5 )
```

Figure 1

File Edit View Insert Tools Desktop Window Help

Figure 2

File Edit View Insert Tools Desktop Window Help

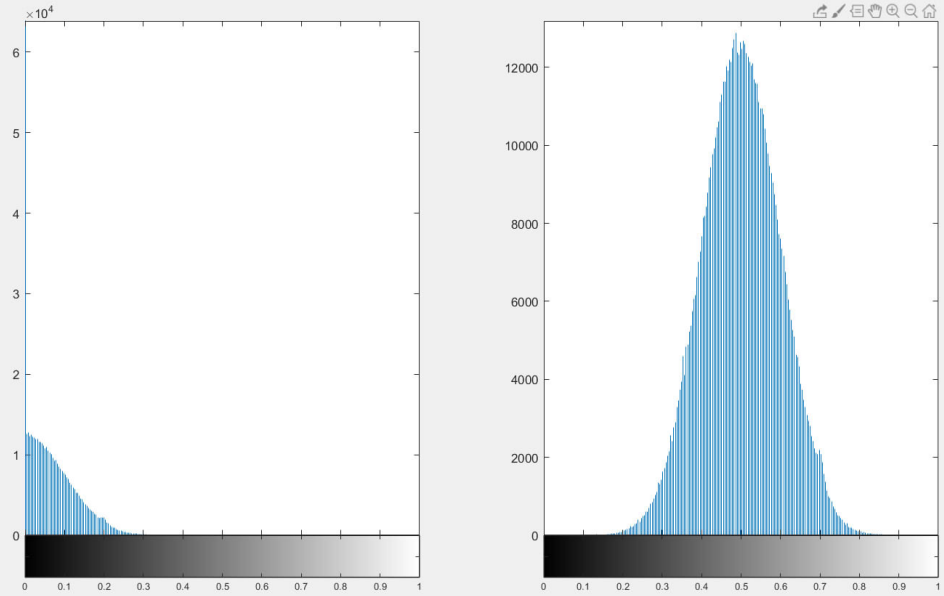
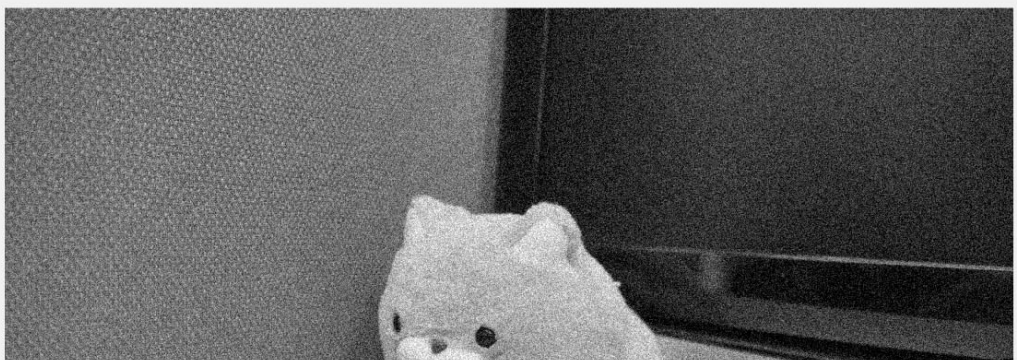
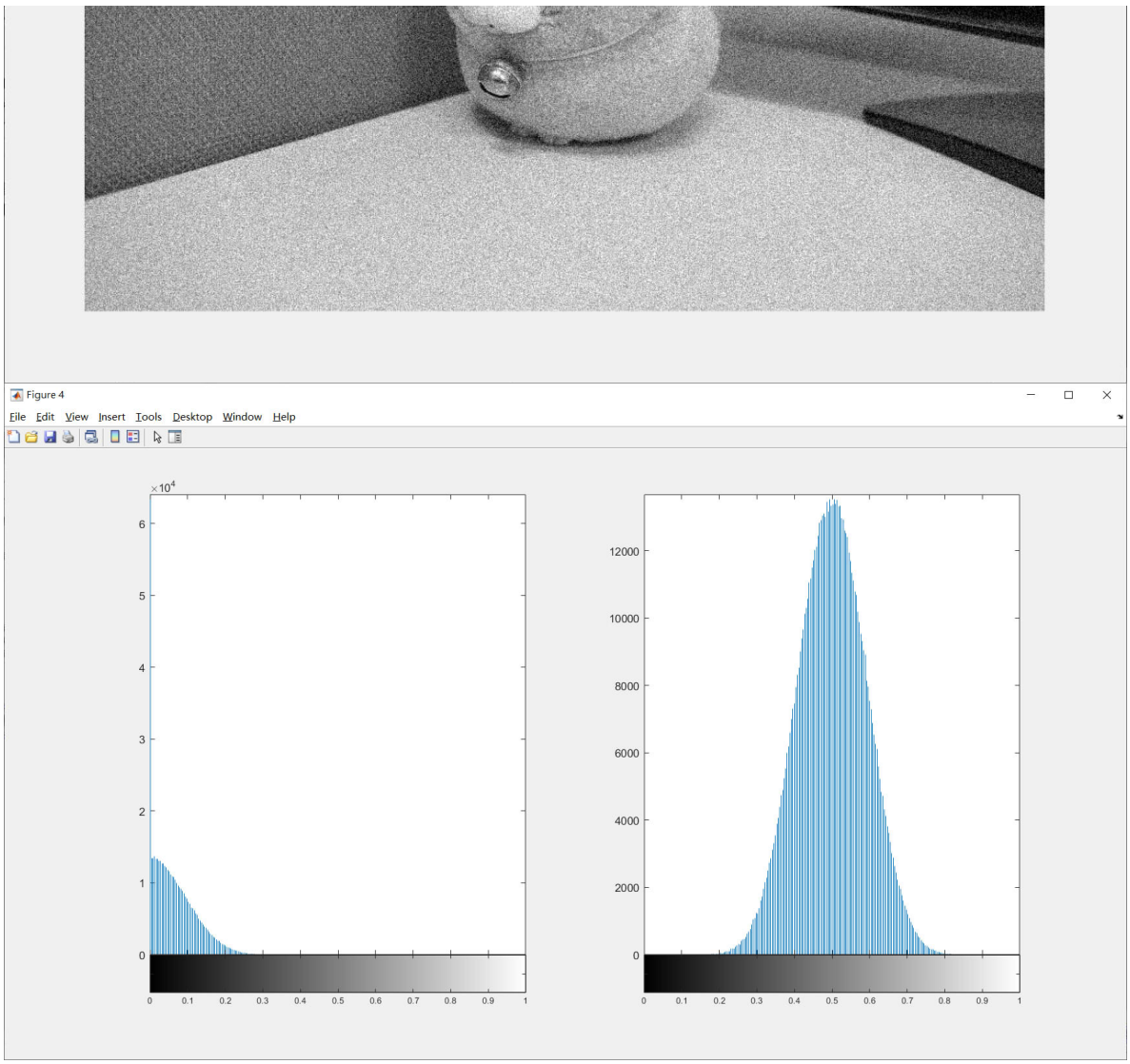


Figure 3

File Edit View Insert Tools Desktop Window Help





影像平均

- 如果有多張圖片，且圖片的雜訊都是獨立同分布的，將這些圖片取平均，可以降低雜訊的標準差，但雜訊的偏移(平均值)無法降低。
- 下圖分別是疊加1張, 10張, 100張的結果，可以觀察黑色螢幕的部分，會發現雜訊的斑點變淡消失。

```
%-- example4 --%
gimg = imread('demo_gray.bmp');

%-- 轉換到 [0,1]
gimg = im2double(gimg);

%-- 使用內建的函數 imnoise 設 N~(0,2 , 0.01)
tmp = imnoise( gimg , 'gaussian' , 0.2 , 0.01);
figure()
imshow(tmp)

%-- 假設疊加 10 張圖片
n = 10;
%-- 生成畫布
img = zeros( size(gimg) );
for i=1:n
    tmp = imnoise( gimg , 'gaussian' , 0.2 , 0.01);
    img = img + tmp/n;
```

```
end
figure()
imshow(img)

%-- 假設疊加 10 張圖片
n = 100;
%-- 生成畫布
img = zeros( size(gimg) );
for i=1:n
    tmp = imnoise( gimg , 'gaussian' , 0.2 , 0.01);
    img = img + tmp/n;
end
figure()
imshow(img)
```

Figure 1

File Edit View Insert Tools Desktop Window Help

Figure 2

File Edit View Insert Tools Desktop Window Help

Figure 3

File Edit View Insert Tools Desktop Window Help

- 連續變化的圖片...

白雜訊

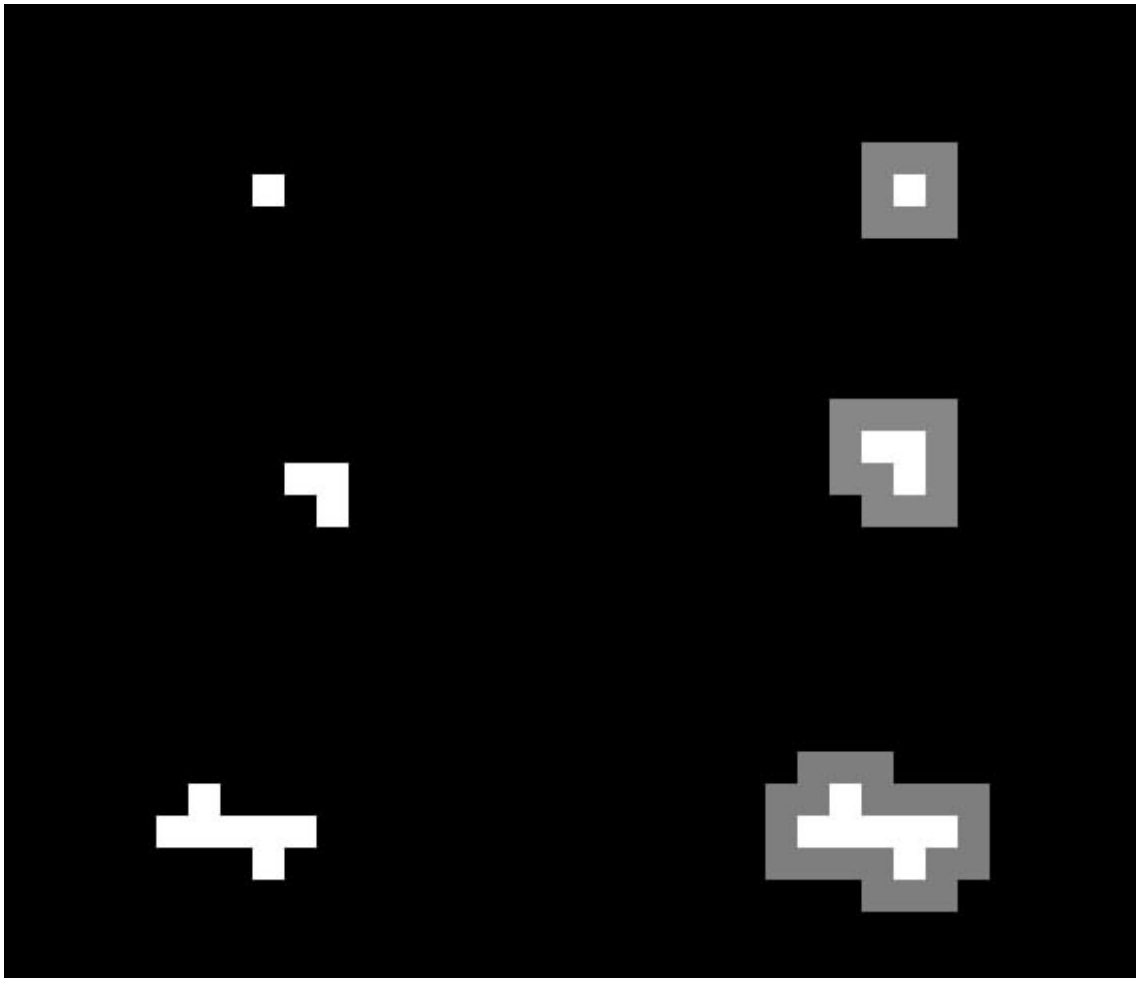
形態學

- 膨脹
- 侵蝕
- 結構元素
- 斷開(open)
- 閉合(close)
- 邊界抽取
- 註解
- 型態交離轉換(Hit-or-Miss transform) (找出特定形狀)
- 區域填充
- 底帽變換
- 頂帽變換

- 膨脹、侵蝕
- 結構元素
- 斷開、閉合
- 邊界抽取
- 型態交離轉換 (找出特定形狀)
- 頂帽、底帽轉換 (光照不均校正)
- 區域填充、連通區塊選取 (以及車牌偵測時使用的那個)
- Region growing
- 細線化、厚化、骨架化、剪除、重建
- 符號上的表示

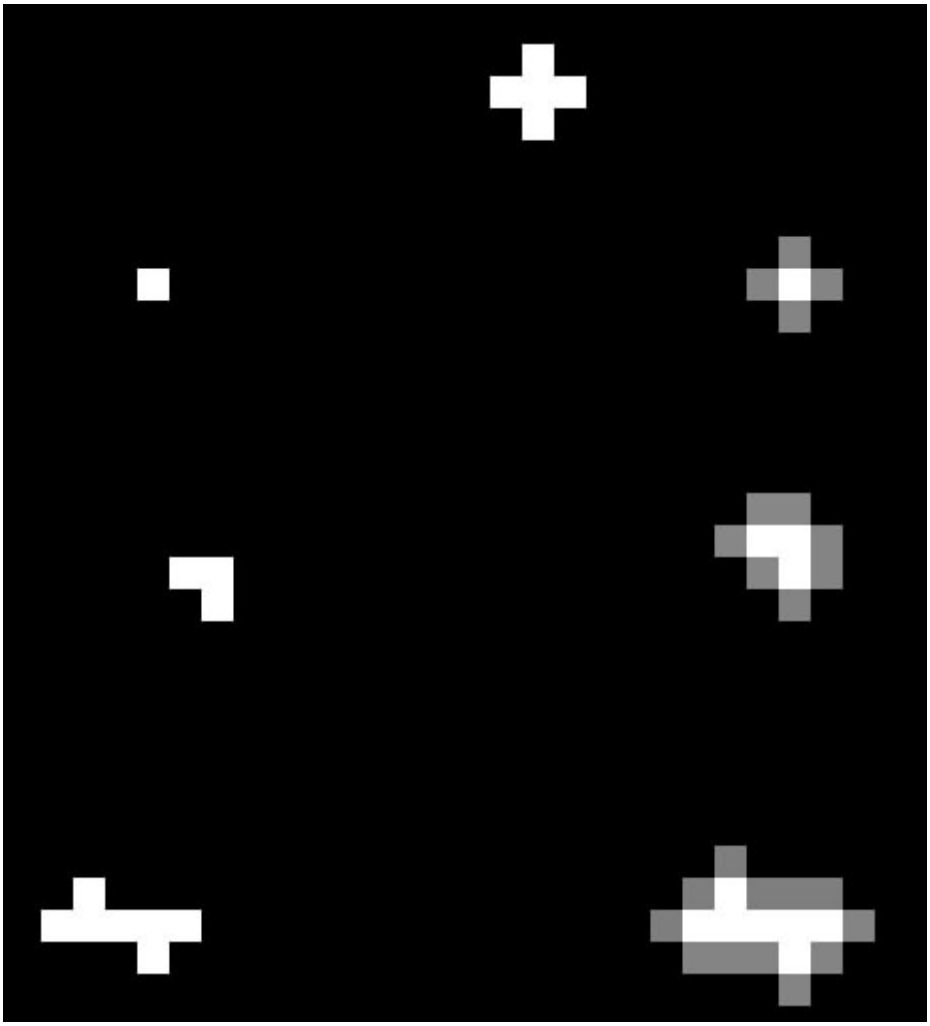
膨脹

影像上的膨脹(dilate)，最簡單的方式是應用在二值影像上。
他會將原本影像套用一個遮罩，根據遮罩的形狀，生成每一點的值。
如果今天遮罩是 3*3 的正方形的話，結果如下：

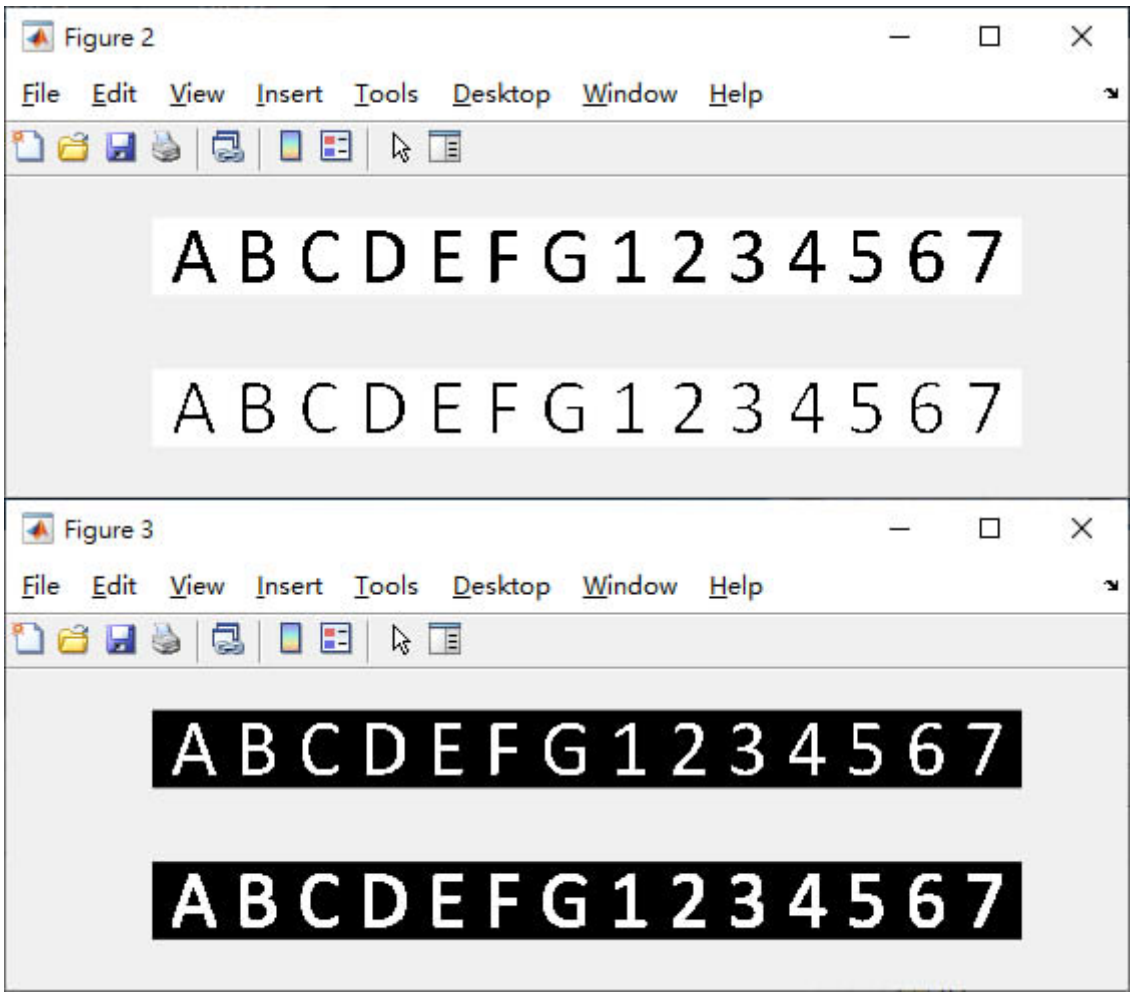


這個遮罩的作用和 3*3的最大值遮罩作用一樣。

或是遮罩是十字型：



他的做法是，如果圖上的某個位置有值，那就會用遮罩在那個位置蓋上去。(就像拿一支更粗的筆再描一次)



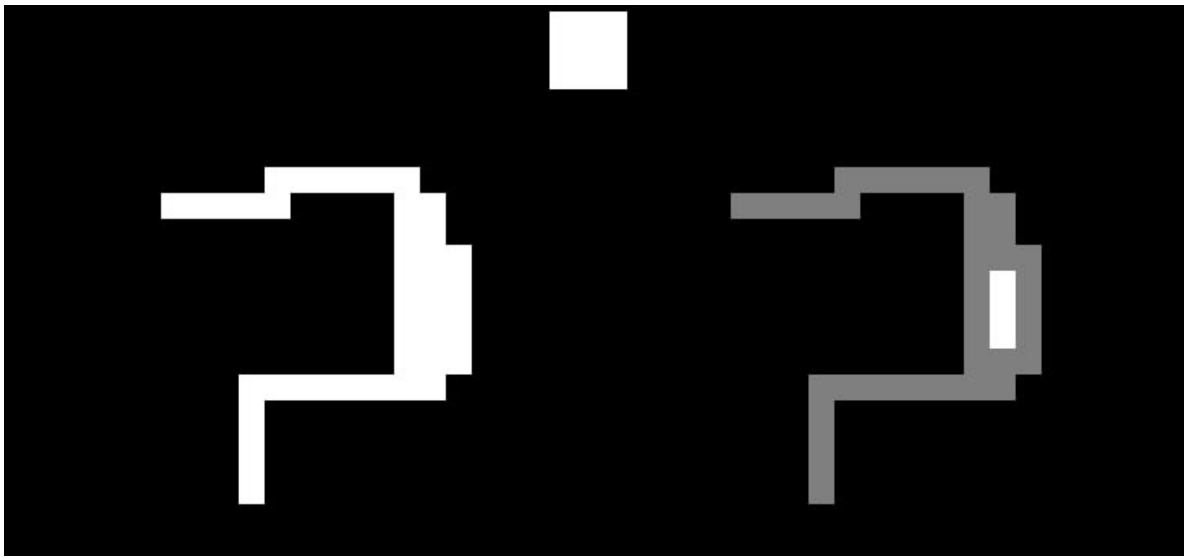
符號寫成 $ A \oplus B $ ，其中 $ A $ 是原本圖像，$ B $ 是遮罩。

```
%-- example1 --%
img = imread('example1.jpg');
img = im2bw( img );

mask = ones(3);

dimg = imdilate( img , mask );

figure()
subplot(2,1,1)
imshow( img )
subplot(2,1,2)
imshow( dimg )

dimg = imdilate( 1-img , mask );

figure()
subplot(2,1,1)
imshow( 1-img )
subplot(2,1,2)
imshow( dimg )
```

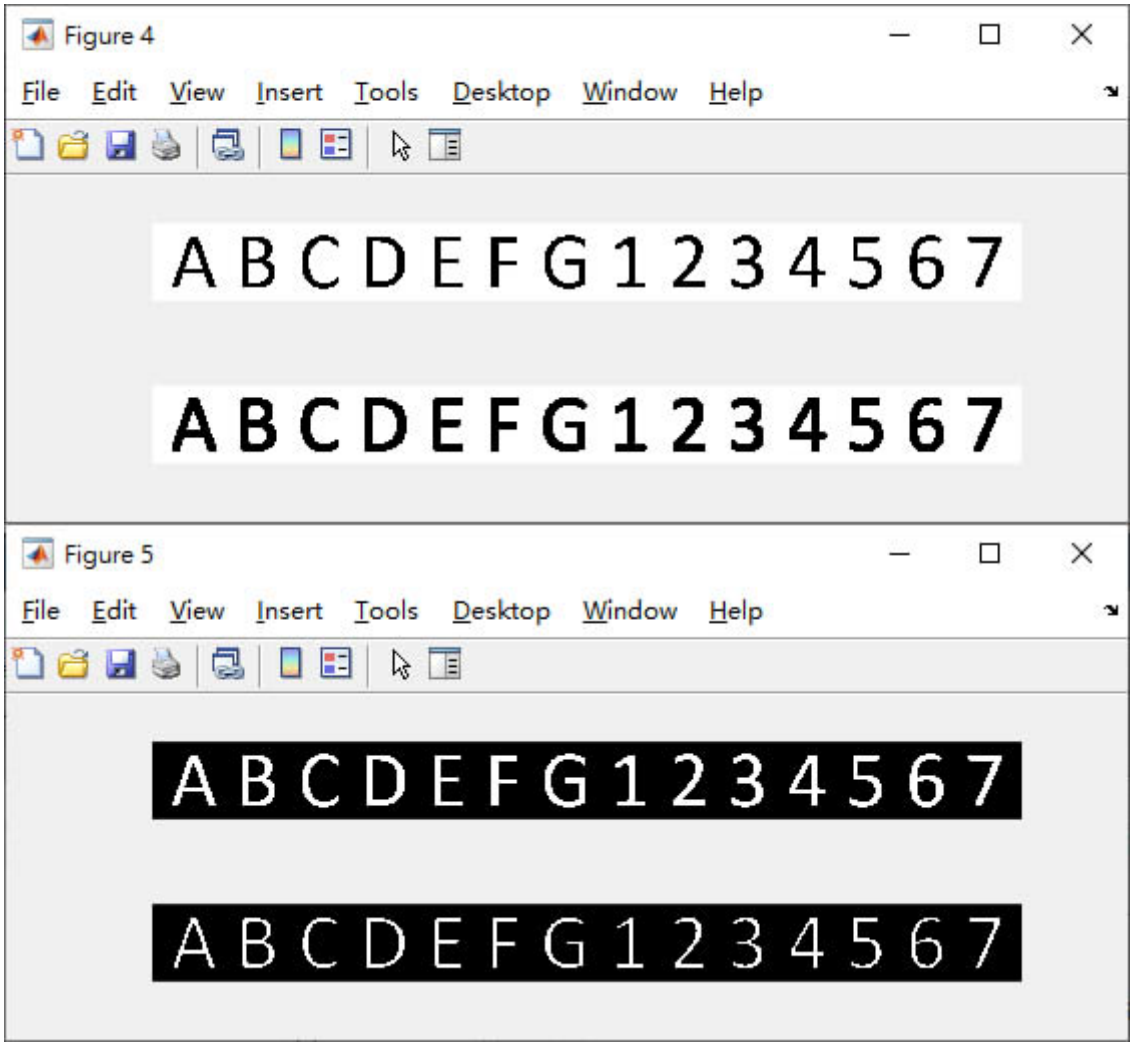
侵蝕

影像上的侵蝕(erode)，最簡單的方式是應用在二值影像上。

他的做法是，看看遮罩在圖像上哪個位置放得下，如果放得下就把值加上去。



這個遮罩的作用和 3*3的最小值遮罩作用一樣。



符號寫成 $ A \ominus B $ ，其中 $ A $ 是原本圖像，$ B $ 是遮罩。

```
%-- example2 --%
img = imread('example1.jpg');
img = im2bw( img );

mask = ones(3);

dimg = imdilate( img , mask );
```

```
8
9  figure()
10 subplot(2,1,1)
11 imshow( img )
12 subplot(2,1,2)
13 imshow( dimg )
14
15 dimg = imdilate( 1-img , mask );
16
17 figure()
18 subplot(2,1,1)
19 imshow( 1-img )
20 subplot(2,1,2)
21 imshow( dimg )
```

結構元素

結構元素就是剛剛提到的膨脹侵蝕中的遮罩(程式碼中的 mask)，使用不同的遮罩可以達到不同的效果。

遮罩並不侷限於 3*3，也可以 5*5 或是更大的，甚至是不同形狀的

可以使用 matlab 的 `strel` 來生成，或是自己輸入一個矩陣。
`help strel` 有更多形狀和用法

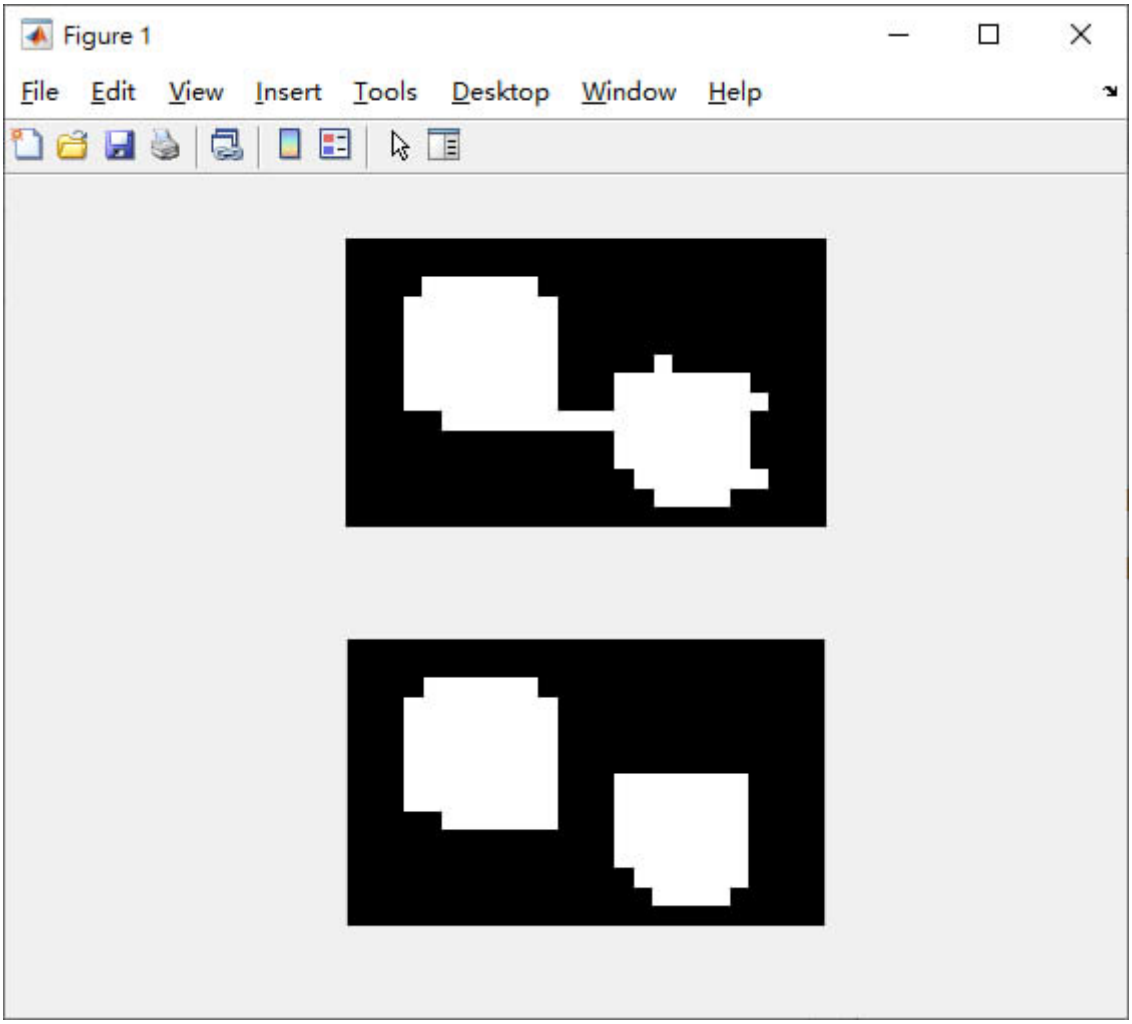
```
1 % 圓形
2 strel('disk', 11);
3
4 % 方型
5 strel('square', 11);
```

在灰階形態學(不單純作用在二值圖像上)時，遮罩值就可以不是整數值，也許會是呈現高斯分布(墨西哥帽)的數值分布，模擬顏色往外擴散的形式。

斷開(open)

先做一次侵蝕再做一次膨脹，可以清除圖案上細小的分支，連結。

定義為 $ A \circ B = (A \ominus B) \oplus B $

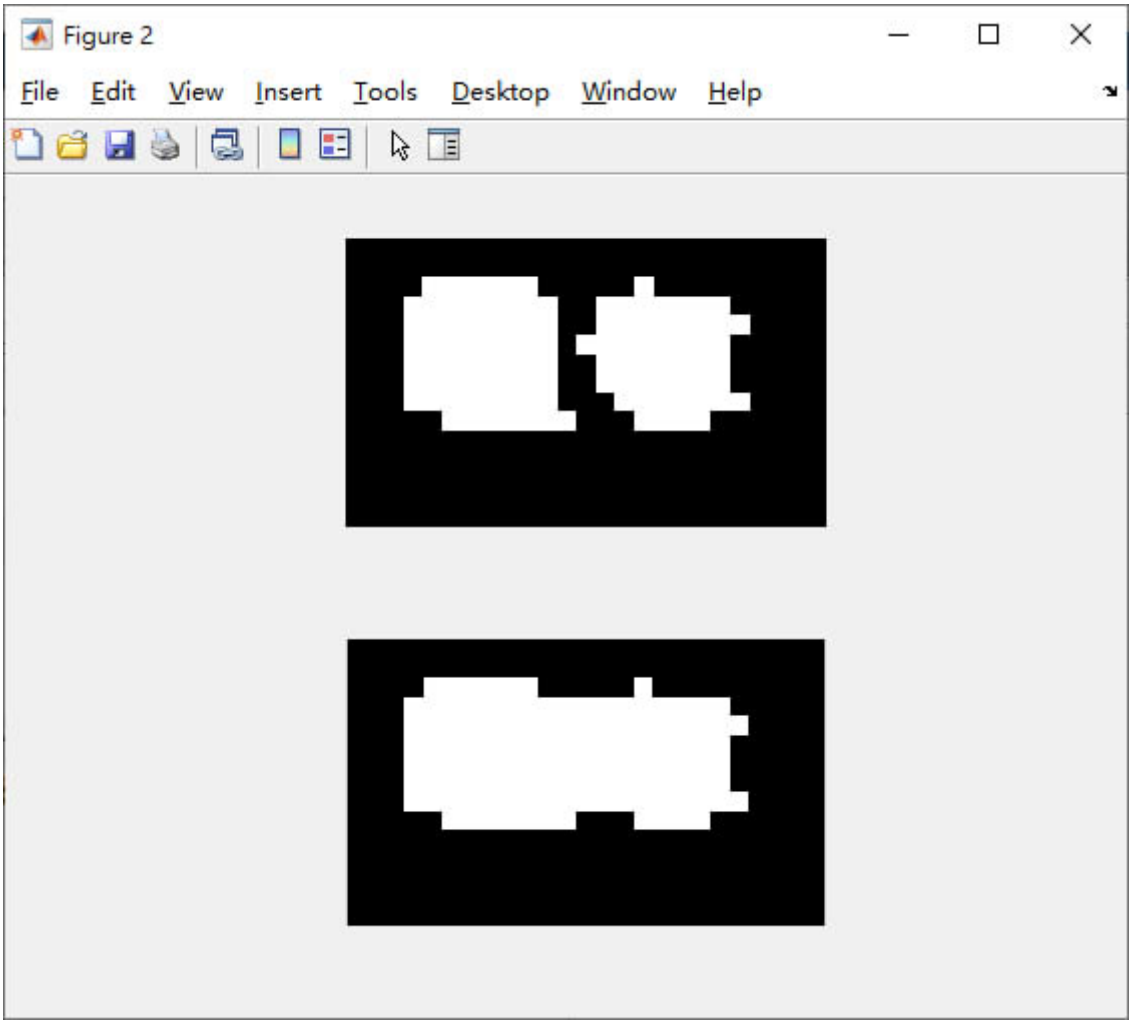



```
%-- example3 --%
img = imread('example7.bmp');
img = im2bw( img );

mask = ones(3);

eimg = imerode( img , mask );
dimg = imdilate( eimg , mask );

figure()
subplot(2,1,1)
imshow( img )
subplot(2,1,2)
imshow( dimg )
```

閉合(close)

先做一次膨脹再做一次侵蝕，可以將圖案靠近的區塊連結在一起。

定義為 $ A \bullet B = (A \oplus B) \ominus B $

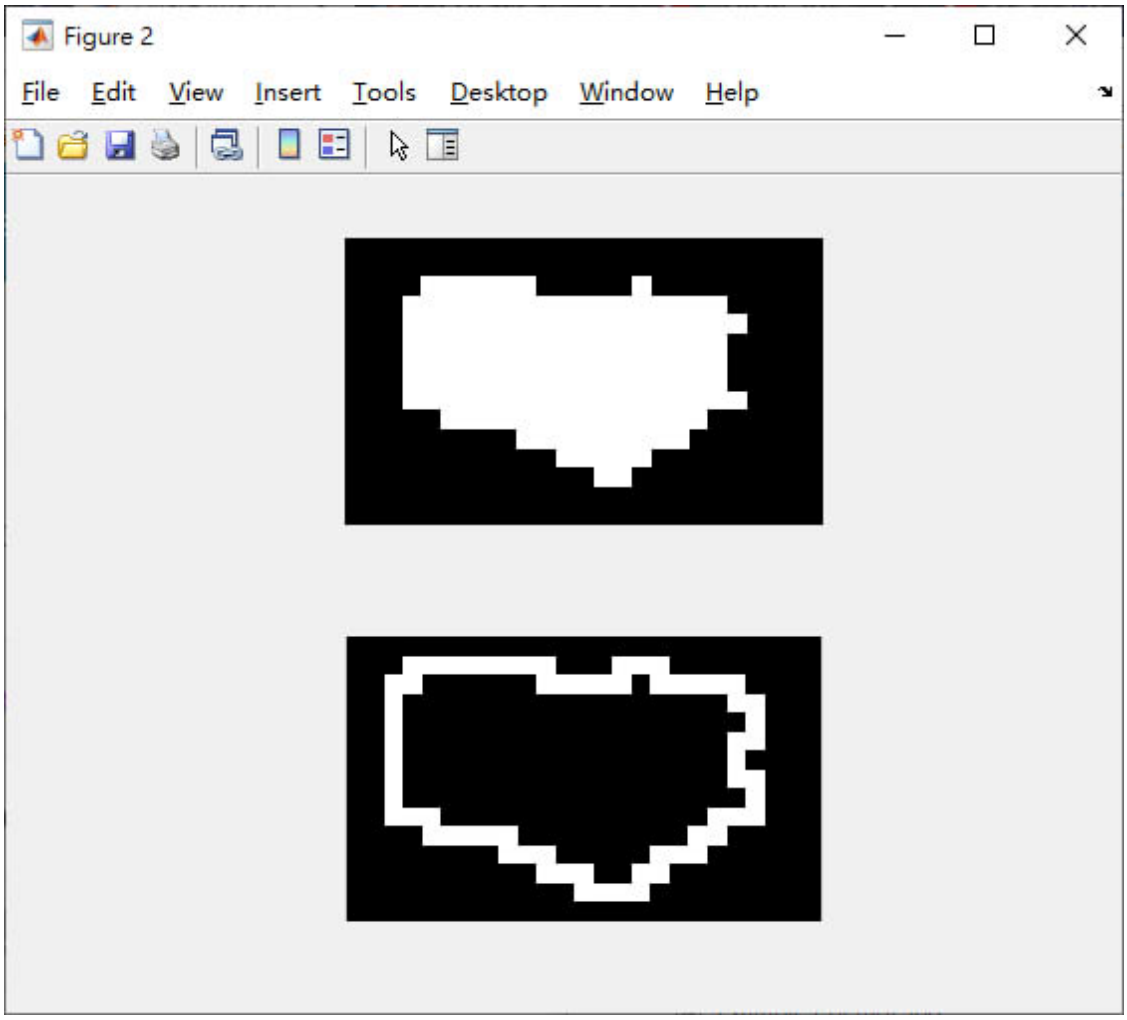



```
%-- example4 --%
img = imread('example8.bmp');
img = im2bw( img );

mask = ones(3);

dimg = imdilate( img , mask );
eimg = imerode( dimg , mask );

figure()
subplot(2,1,1)
imshow( img )
subplot(2,1,2)
imshow( eimg )
```

邊界抽取

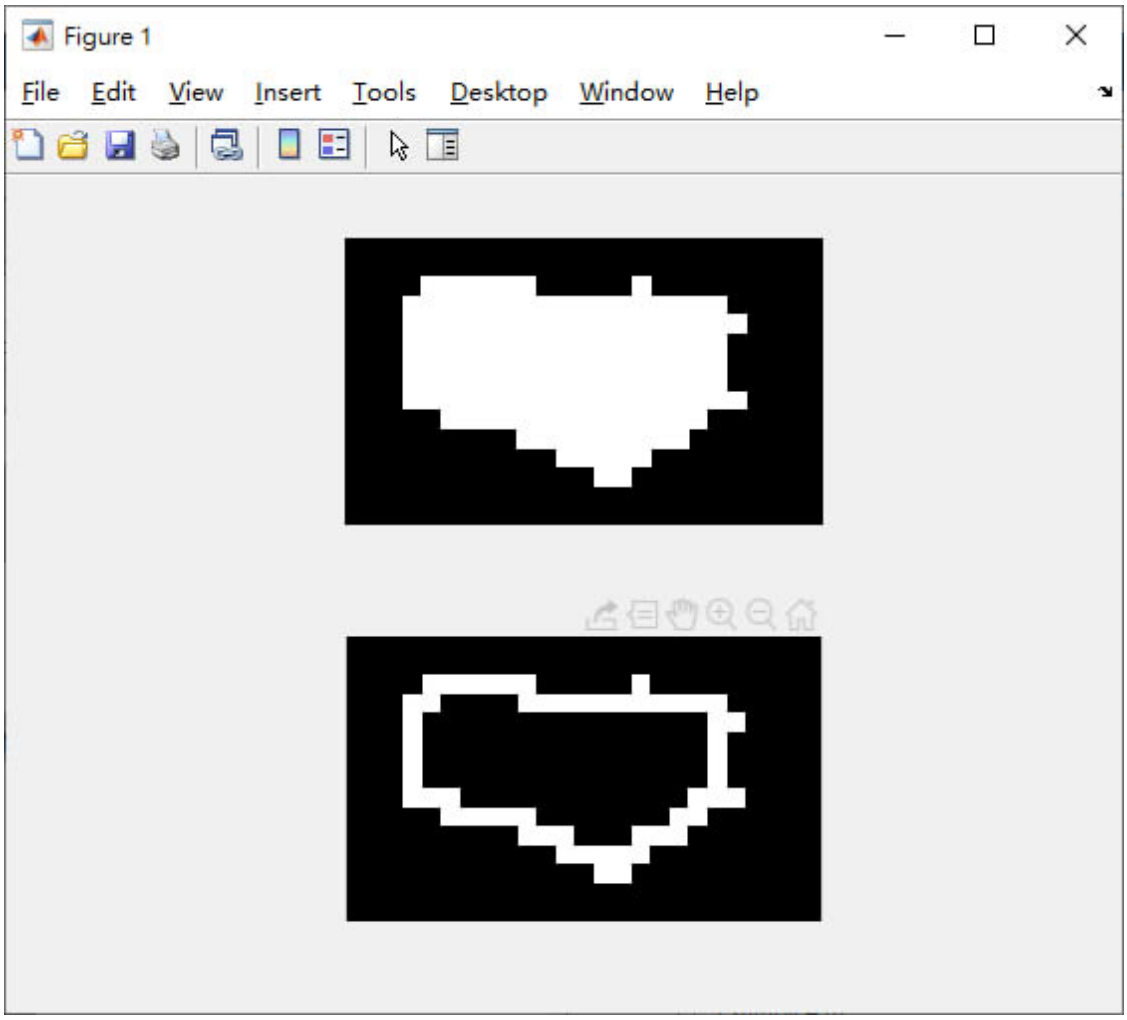
可以透過侵蝕膨脹來得到圖像的邊界。
又分為外邊界(原本圖案範圍外)與內邊界(原本圖案範圍內)。

- 外邊界為膨脹圖案減去原本圖案。




```
%-- example4 --%
img = imread('example9.bmp');
img = im2bw( img );

mask = ones(3);

dimg = imdilate( img , mask );

figure()
subplot(2,1,1)
imshow( img )
subplot(2,1,2)
imshow( dimg-img )
```

- 內邊界為原本圖案減去侵蝕圖案。




```
%-- example4 --%
img = imread('example9.bmp');
img = im2bw( img );

mask = ones(3);

eimg = imerode( img , mask );

figure()
subplot(2,1,1)
imshow( img )
subplot(2,1,2)
imshow( img-eimg )
```

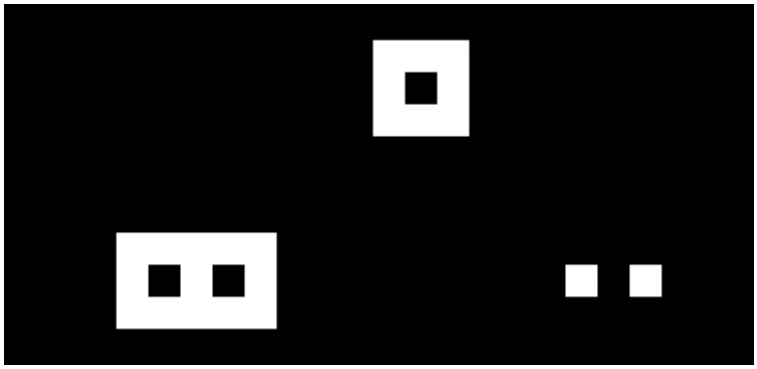
註解

膨脹、侵蝕並不互為逆運算 (因為斷開、閉合並不總是 identity)
實際上它們的關係是：

$ \overline{ A \ominus B } = \overline{A} \oplus \hat{B} $
$ \overline{ A \oplus B } = \overline{A} \ominus \hat{B} $

膨脹侵蝕也不是線性運算，因為他們可以看做是最大值、最小值遮罩的特別版本，也就是說並不能把一系列的膨脹、侵蝕的合成成單一運算。

另外，侵蝕、膨脹的結構元素並不一定要包含中心點的位置。
也就是說侵蝕之後的圖案並不一定是原本的子集。
以下面例子來說，(以回字作為結構元素)侵蝕後的結果和原本的圖案完全沒有交集。

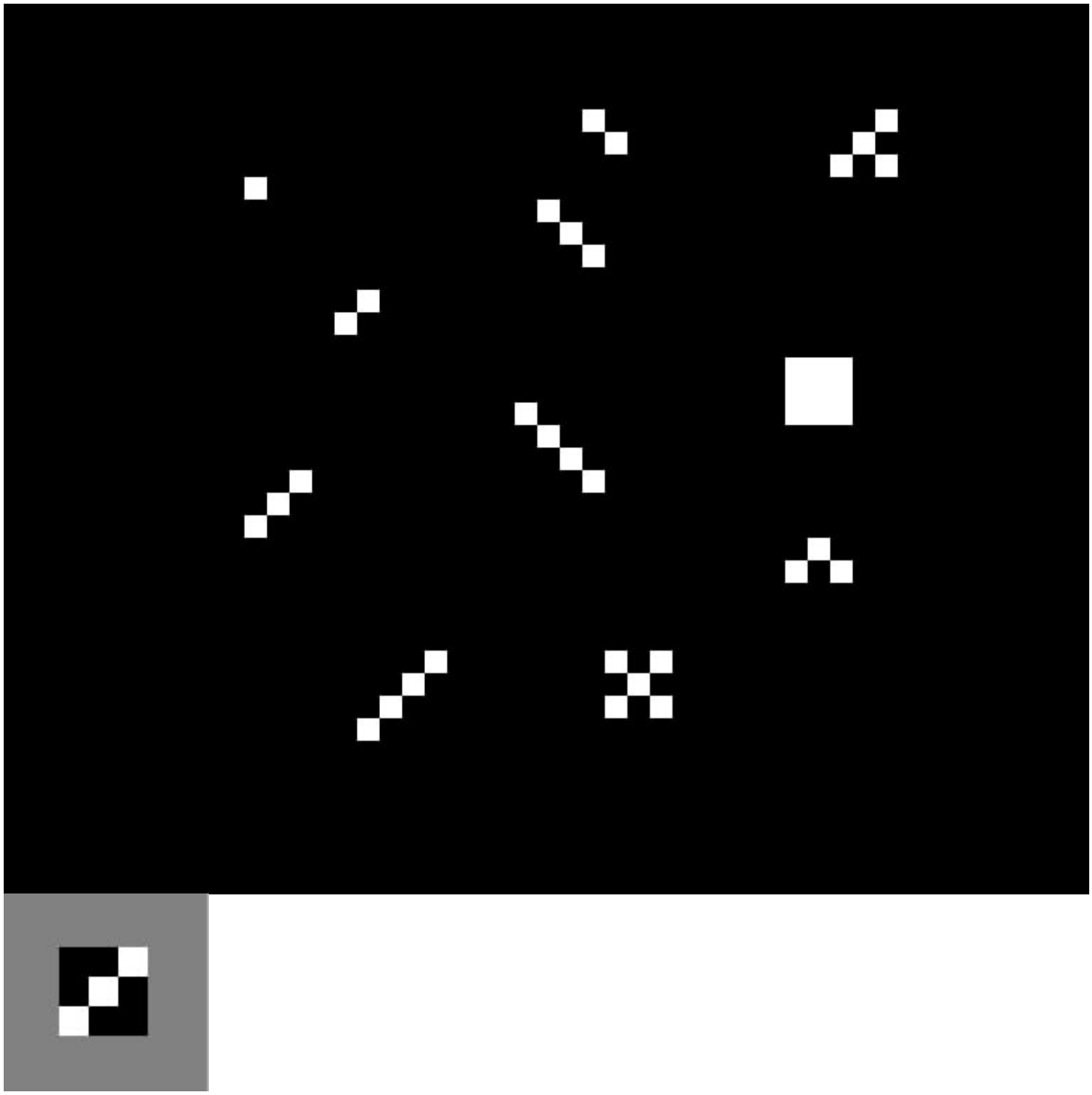


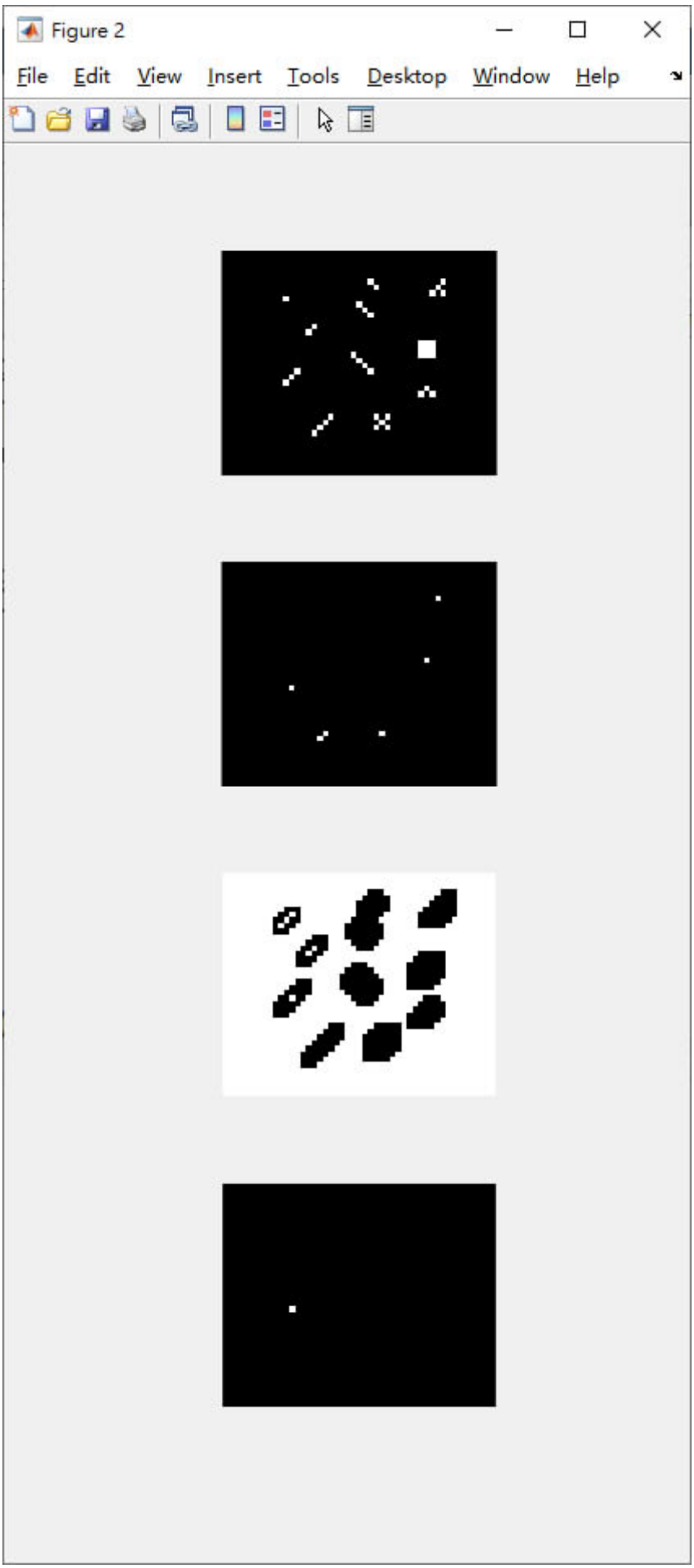
另外，閉合、斷開運算是等幕的(作用多次等於作用一次)。

型態交離轉換(Hit-or-Miss transform) (找出特定形狀)

0. 先決定一個結構元素 C
1. 透過 C 侵蝕圖片 A 找出可能的位置 A_1
2. 透過 C 的外邊界侵蝕圖片補集 $ \overline{A} $ 找出可能的位置 A_2
3. 交集 $ A_1 $ 與 $ A_2 $ 就能找出形狀為 C 的圖案位置

以下例來說，我從下列這張圖片中找到某個 3*3 的圖案。





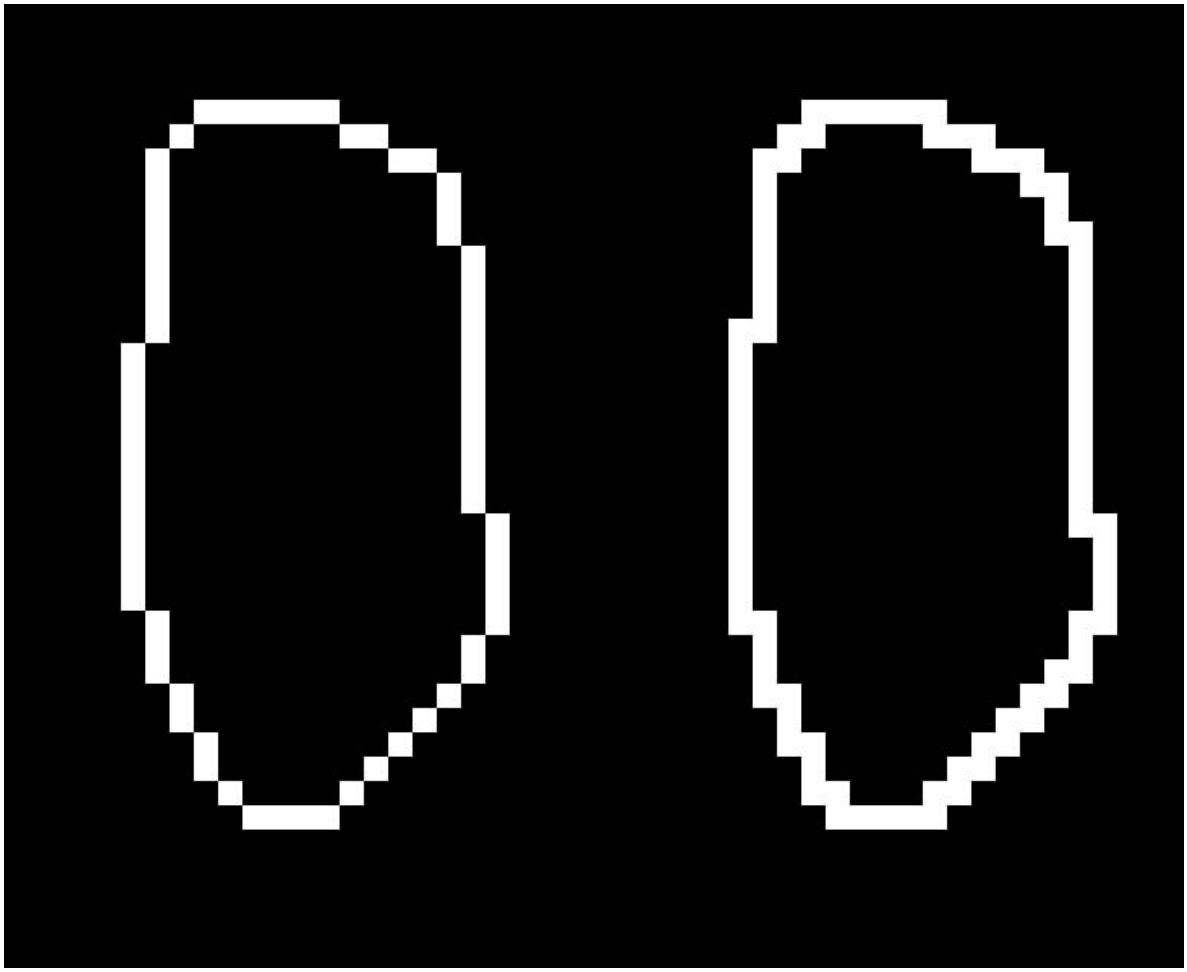

```
%-- example7 --%
img = imread('example12.jpg');
img = im2bw( img );
```

```
4
5  %-- struct element C
6  C = [ 0 0 1 ; 0 1 0 ; 1 0 0  ];
7
8  %-- outter edge
9  Z = zeros(size(C)+2);
10 Z(2:end-1 , 2:end-1 ) = C;
11 Z = imdilate(Z , ones(3) ) - Z;
12
13 %-- A
14 A1 = imerode( img , C );
15 A2 = imerode( 1-img , Z );
16
17 figure()
18 subplot(4,1,1)
19 imshow( img )
20 subplot(4,1,2)
21 imshow( A1 )
22 subplot(4,1,3)
23 imshow( A2 )
24 subplot(4,1,4)
25 imshow( A1 & A2 )
```

區域填充

區域填充用來將一個點生長到整個被邊線包圍的區域。
首先必須先有一個封閉的曲線(4-連通 或 8-連通)。

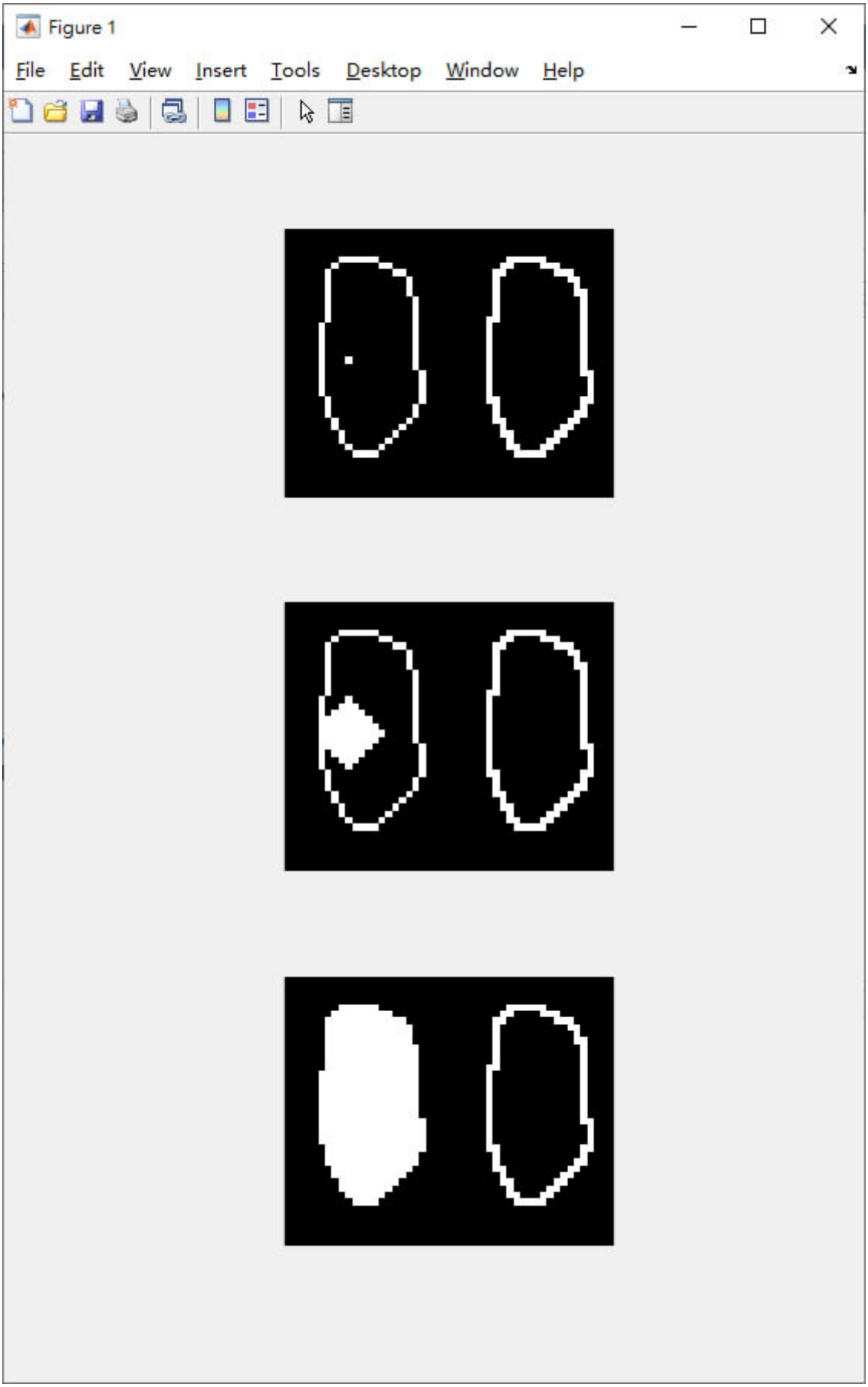
關於 4-連通 與 8-連通，簡易的判別方法是，4-連通只能走上下左右， 8-連通可以走斜方向，看能不能從裡面區域走到外面區域。



填充的過程使用膨脹運算，先挑選一個內部點，並不斷使用膨脹運算直到圖案不再發生改變。

$ P_n = (P_{n-1} \oplus C) \cap \overline{A} , P_0 = \{p\} $

4-連通的區域填充 C 只能選用十字，8-連通則可以選用正方形。
在 4-連通的區域使用正方形會超出邊界。




```
%-- example8 --%
img = imread('example16.jpg');
img = im2bw( img );

P = zeros( size(img) );
P(20,10) = 1;
```

```
C = [ 0 1 0 ; 1 1 1 ; 0 1 0];

figure()
subplot(3,1,1)
%-- P0
imshow(P | img)

for i = 1:5
    P = imdilate( P , C ) & ~img;
end

subplot(3,1,2)
imshow(P | img)

isChange = true;
while isChange
    tmp = P;
    P = imdilate( P , C ) & ~img;
    isChange = 1 - min(   P(:) == tmp(:)    );
end

subplot(3,1,3)
imshow(P | img)
```

底帽變換

$ A - (A \circ C) $

頂帽變換

$ (A \bullet C) - A $

- (光照不均校正)
- Region growing
- 細線化、厚化、骨架化、剪除、重建
- 符號上的表示

影像編碼

章節重點

- 無失真壓縮
    - Huffman 編碼
        - 熵
    - 行程長度編碼
        - Gray 碼
- 失真壓縮

---

0. 文件的 markdown 源碼
1. 範例圖片檔 打包下載
2. 範例程式碼 打包下載

無失真壓縮

行程長度編碼(run-length encoding)

行程長度編碼，或簡稱 RLE ，這個編碼方式是使用"可變長度的碼來取代**連續重複出現**的資料"，舉例而言，有類似下面的資料。

```
AAAABBBCCCCCCDDDDDEEEEFFFFF
```

我們可以把它重新編碼成

```
4A3B6C5D4E5F
```

比較編碼前後，可以將原本長度27的字串壓縮成長度12，並達到了 12/27 ≈ 44% 。

而這個壓縮方法也可以應用到影像上面，實際上這個壓縮方法可以應用在任何資料上，在不同類型的資料上也可能會有各種變型。

影像上的 RLE

我們先考慮一張影像，這張影像每個像素有 2^3=8 種顏色。 (一般灰階圖有 2^8=256 種 ，彩色圖則有 2^{24} 種)。

同時可以稱這種可以表達的範圍為"位元深度 3"。只要查看圖片的詳細資料通常就會看到，附圖是一張彩色影像的詳細資料。

首先我們把這張位元深度 3 的影像包含的像素列出來。它的數值可能會類似下列所示：

$$\begin{bmatrix} 0 & 3 & 2 & 1 \\ 1 & 4 & 6 & 2 \\ 0 & 0 & 4 & 3 \\ 3 & 7 & 7 & 0 \end{bmatrix}$$

第一步，我們先將所有數字串聯起來。

```
0321
3462
3343
6770
```

第二步，則是依據下表將所有數字轉換為 0 和 1。

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

獲得下列編碼(空格實際上都不需要，下列是為了方面觀察)

```
000 011 010 001
011 100 110 010
011 011 100 011
110 111 111 000
```

第三步，將數字重新分割成三個位元平面，分別取每個3位數字的第1位、第2位及第3位。

第一個位元平面

```
0 0 0 0
0 1 1 0
0 0 1 0
1 1 1 0
```

第二個位元平面

```
0 1 1 0
1 0 1 1
1 1 0 1
1 1 1 0
```

第三個位元平面

```
0 1 0 1
1 0 0 0
1 1 0 1
0 1 1 0
```

第四步，則是"從0開始"計算每個"連續 0 或 1 的長度"來編碼。以上例來說(如果第一個出現的就是 1，要加 0 在最前頭)，就會編碼成

```
4 121 211 31

121 112 211 31

1111 13 211 121
```

到這邊為止，就算是對這個 3 位元深度的圖像做了 RLE 編碼。

壓縮後佔用的空間為(假設編碼後的數字用 2 位元來儲存)， (9+11+12)*2 = 64 位元

而原先占用的空間為 4*4*3 = 48 位元，壓縮之後大小反而放大了。

另外在第四步也有提出別種編碼方式，以兩個數字一組，第一個數字為 1 出現的位置，第二個數字為連續 1 的長度。以這種編碼的話，上面的資料會被編碼為

```
0 22 31 13

22 1132 1241 13

2141 11 1241 22
```

另外也有在轉換時，使用別種轉換表(Gray碼、格雷碼)。
規則為： (從二進制來轉換) (其中 n 代表轉換碼的第 n 位)
$G(n) = B(n+1) + B(n)$

二進位	Gray碼
100	..0
100	.10
100	110

0	1	2	3	4	5	6	7
000	001	011	010	110	111	101	100

```
%-- example1 --%
gimg = imread('demo_gray.bmp');

%--  Gray 碼
num = uint8( 0 );
for i = 0:255
    num = uint8( i );
    gray( i+1 ) = bitxor( num , bitshift( num , -1 )  );
end

tmpImg = gimg;

%-- 把像素排成一列
L = prod( size(gimg) );
tmpImg = reshape( gimg , 1 , L );

%-- 先轉換為 gray 碼
tmpImg = gray( tmpImg + 1 );

outMean = [];
outLen = [];
%-- 切割位元平面
for i = 1:8
    bitArr = mod(tmpImg,2);
    tmpImg = tmpImg/2;
    out = RLE( bitArr );
    outMean( end+1 , : ) = mean(out);
    outLen( end+1 ) = length( out );  %-- 紀錄編碼後的長度
end

totalLen = sum( outLen(1:end) );
fprintf( 'After coding ... Need %d uint16\n' , totalLen  );
```

```
33 fprintf( 'Before coding ... Need %d uint8\n' , L );
```

霍夫曼編碼

根據字母出現頻率來做可變長度的編碼。舉例來說，原本定義 0-3 四個數字，可能會需要用到兩個位元。

0	1	2	3
00	01	10	11

但可以根據文本中字母出現的頻率重新定義編碼。以 `0001113120100001222` 來說。

- 先統計每個字出現的次數。

0	1	2	3
8	6	4	1

- 不斷地將次數最少的兩個合併

0	1	2	3
8	6	5	

0	1	2	3
8	6	5	
	11		

0	1	2	3
8	6	5	
	11		
19			

- 編碼

0	1	2	3
8 [0]	6	5	
	11 [1]		
19			

0	1	2	3
8 [0]	6 [10]	5 [11]	
	11 [1]		
19			

<table>
<tr><td>0 [0]</td><td>1 [10]</td><td>2 [110]</td><td>3 [111]</td></tr>
<tr><td rowspan="2">8 [0]</td><td>6 [10]</td><td colspan="2">5 [11]</td></tr>
<tr><td colspan="3">11 [1]</td></tr>
<tr><td colspan="4">19</td></tr>
</table>

- 最後獲得的編碼就是

0	1	2	3
[0]	[10]	[110]	[111]

- 原本的 `0001113120100001222` 有 19 字元，需要 38 位元來編碼。
- 改成 Huffman 編碼後，變成 (1*8 + 2*6 + 3*4 + 3*1) = 35 位元。
- 壓縮率為 35/38 = 0.92
- 使用 Matlab 實作

```
%-- example2 --%

gimg = imread('demo_gray.bmp');

L = prod( size( gimg ) ); %-- 計算總 pixel 數量

P = [];
%-- 統計每種亮度的出現次數
for i = 0:255
    P( i+1 , 1 ) = sum( gimg(:) == i ) /  L ;
    P( i+1 , 2 ) = i+1;
end

%-- 複製一份
cP = P( : , 1 );

%-- 建立霍夫曼樹
tree = {};
for i = 1:length(P)
    tree{i} = i ;
end

while length( P(:,1) ) > 1
    %-- 找出機率最低的兩個，合併
    P = sortrows(P);

    %-- 將兩個機率合併，存入第二個
    P( 2 , 1 ) = P( 1 , 1 ) + P( 2 , 1 );

    %-- 紀錄根節點
    root = P( 2 , 2 );

    %-- 建樹
    tree{ P(2,2) } = { tree{ P( 1,2 ) }  , tree{ P(2,2) } };

    %-- 合併後的機率就去掉
    P( 1 , : ) = [];
end
```

```
39
40 %-- 取得建好的霍夫曼樹
41 tree = tree{root};
42
43 %-- 根據霍夫曼樹求出每個字的編碼
44 out = coding( tree , [] );
```

- 印出編碼後的結果

```
1  %-- 印出所有字的編碼，並計算加權長度
2  len = 0;
3  for i=1:256
4      fprintf( "%03d coding to ... " ,  out{i}{1} );
5      len = len + cP( out{i}{1} ) * length( out{i}{2} );
6      for j = 1:length( out{i}{2} )
7          fprintf( out{i}{2}(j) );
8      end
9      fprintf("\n");
10 end
11 len
12
```

```
1  function out = coding(  tree , str  )
2      out = {};
3      if length( tree ) == 1
4          out{ end+1 } = { tree , str } ; %-- 記錄編碼
5      else
6          R = coding( tree{1} , [str , "0"] ); %-- 往左節點分叉
7          L = coding( tree{2} , [str , "1"] ); %-- 往右節點分叉
8
9          for i = 1 : length(R)
10             out(end+1) = R(i); %-- 記錄右節點底下的所有編碼
11         end
12
13         for i = 1 : length(L)
14             out(end+1) = L(i); %-- 記錄左節點底下的所有編碼
15         end
16     end
17 end
```

- 編碼後的加權長度為 7.2886 位元
- 壓縮率為 7.2866/8 = 0.91

熵

熵的概念最初是從物理熱力學引出，用來估計一段訊息的隨機程度。在影像壓縮裡，熵可以是作是壓縮程度的理論最小值。

$$H(I) = -\sum_{d \in I} P(d) log_2(P(d))$$

其中 I 代表圖像，d 代表圖像中的一個一個亮度 (如: 0, 152, 255 , 67 ...)，$P(d)$ 代表 d 這個強度在整張圖的出現機率，$log_2(P(d))$ 則是對這個機率取對數。

以剛才用作 Huffman 編碼的 `0001113120100001222` 字串為例。可以計算出熵為

0	1	2	3
0.42	0.31	0.21	0.05

$-[0.42 * (-1.25) + 0.31 * (-1.68) + 0.21 * (-2.25) + 0.05 * (-4.32)]$
$= -[-0.525 - 0.52 - 0.472 - 0.216] = 1.733$

理論壓縮率(假設原本是以兩位元編碼每一個字)會是 1.733/2 = 0.866

- 用 Matlab 來計算熵 (要注意 log2 裡面不能放 0)

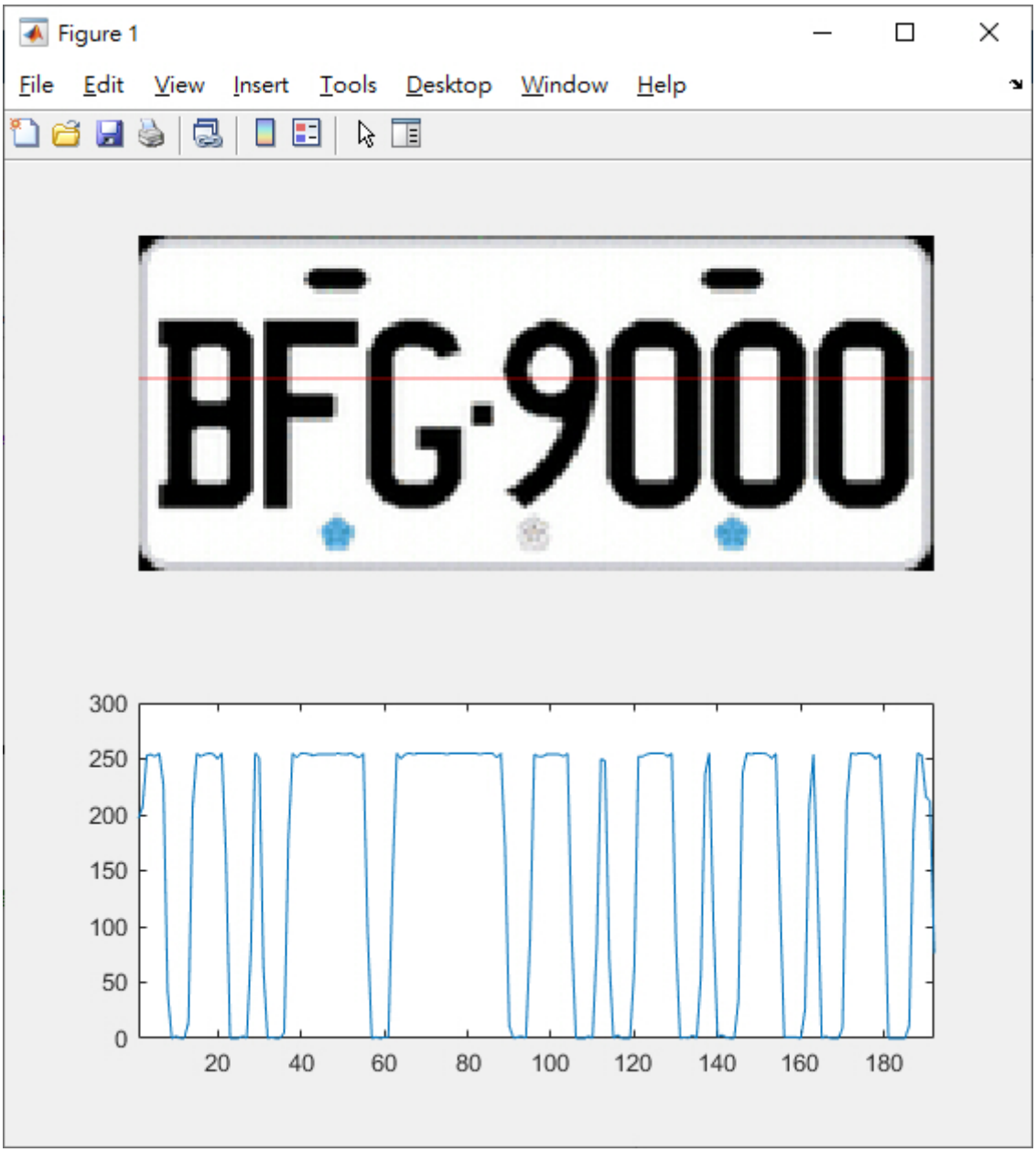
```
%-- example3 --%
gimg = imread('demo_gray.bmp');

L = prod( size( gimg ) ); %-- 計算總 pixel 數量

P = [];
%-- 統計每種亮度的出現次數
for i = 0:255
    P( i+1 ) = sum( gimg(:) == i ) /  L ;
end

entropy = 0;
for i=1:256
    if P(i) ~= 0
        entropy = entropy - ( P(i)*log2(P(i)) );
    end
end

entropy
```

- 理論壓縮率(假設原本是以八位元編碼每一個字)會是 7.247/8 = 0.905

車牌特徵

從車牌圖片來看，可以發現亮度值不斷上下跳躍。
(圖片上的紅線代表用來繪製亮度折線圖的像素)

所以我們假設車牌是亮度會急遽變化的區塊。然後設計一套流程(演算法)，來找出這樣的區塊。



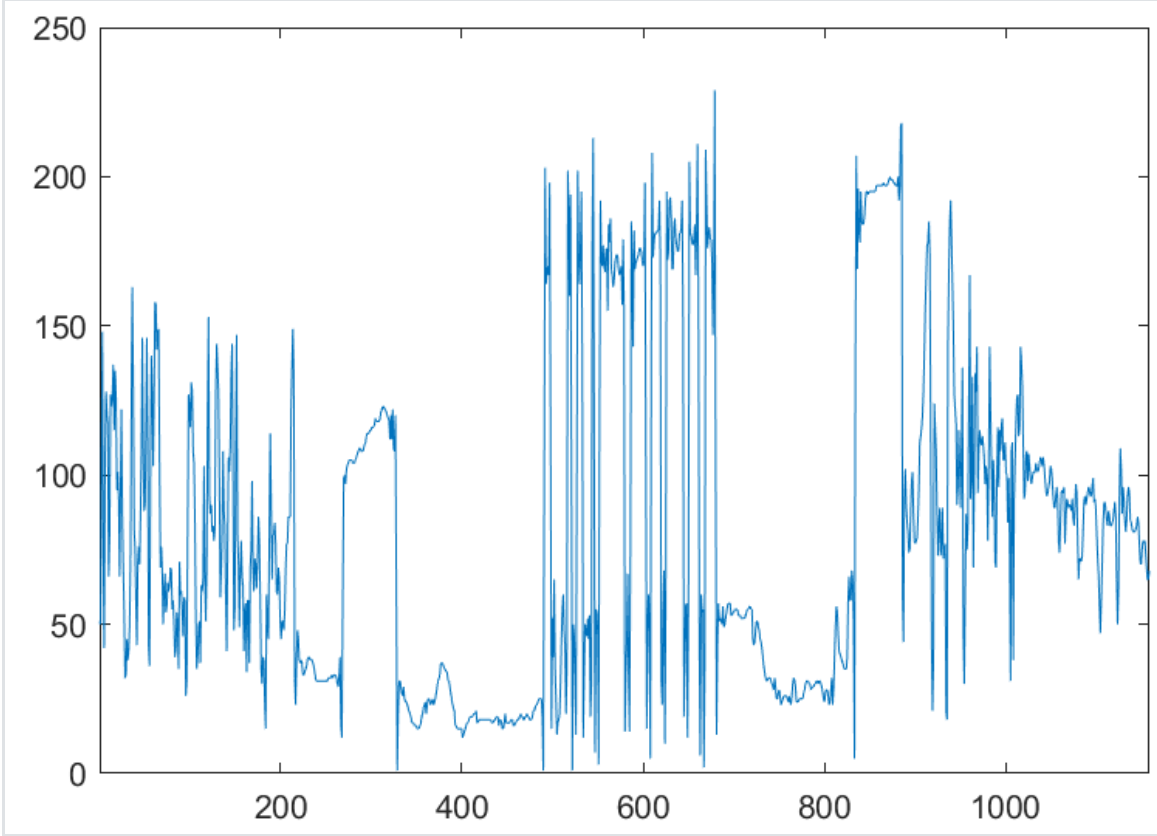
可以透過調整範例程式碼的 `h` 來看看不同高度的折線圖。

```
%%%%----  example1  ----%%%%

cimg = imread('../image/licences_demo1.jpg');   % 讀圖片
gimg = rgb2gray( cimg );                                    % 灰階化

%%%%----  畫出折線圖  ----%%%%

h = 35;                  % 設定擷取的高度

```

```
figure()
subplot(2,1,2)
plot( gimg(h,:) )      % 畫折線圖
set(gca, 'xLimSpec', 'tight'); % 設定圖軸貼齊

%%%%----  在原本的圖片上描上紅線  ----%%%%

%-- 做出遮罩
mask = zeros( size(cimg( h , : , : )) );
mask = uint8( mask );
mask( 1 , : , 1 ) = 255;     % 設定成紅色的

%-- 調和原本的顏色和遮罩的顏色
alpha = 0.3;
cimg( h , : , : ) = (cimg( h , : , : )*(1-alpha) ) + mask*alpha ;

subplot(2,1,1)
imshow( cimg )
```

複雜環境中的車牌特徵

觀察車牌在整體圖片中亮度的變化，會發現在車牌部分亮度變化快速，而且變化的幅度很大。



可以透過調整範例程式碼的 `h` 來看看不同高度的折線圖。
(在跑程式時要注意可能會因為螢幕解析度不足導致 1px 的線不會出現，稍微放大縮小圖片就會出現。)

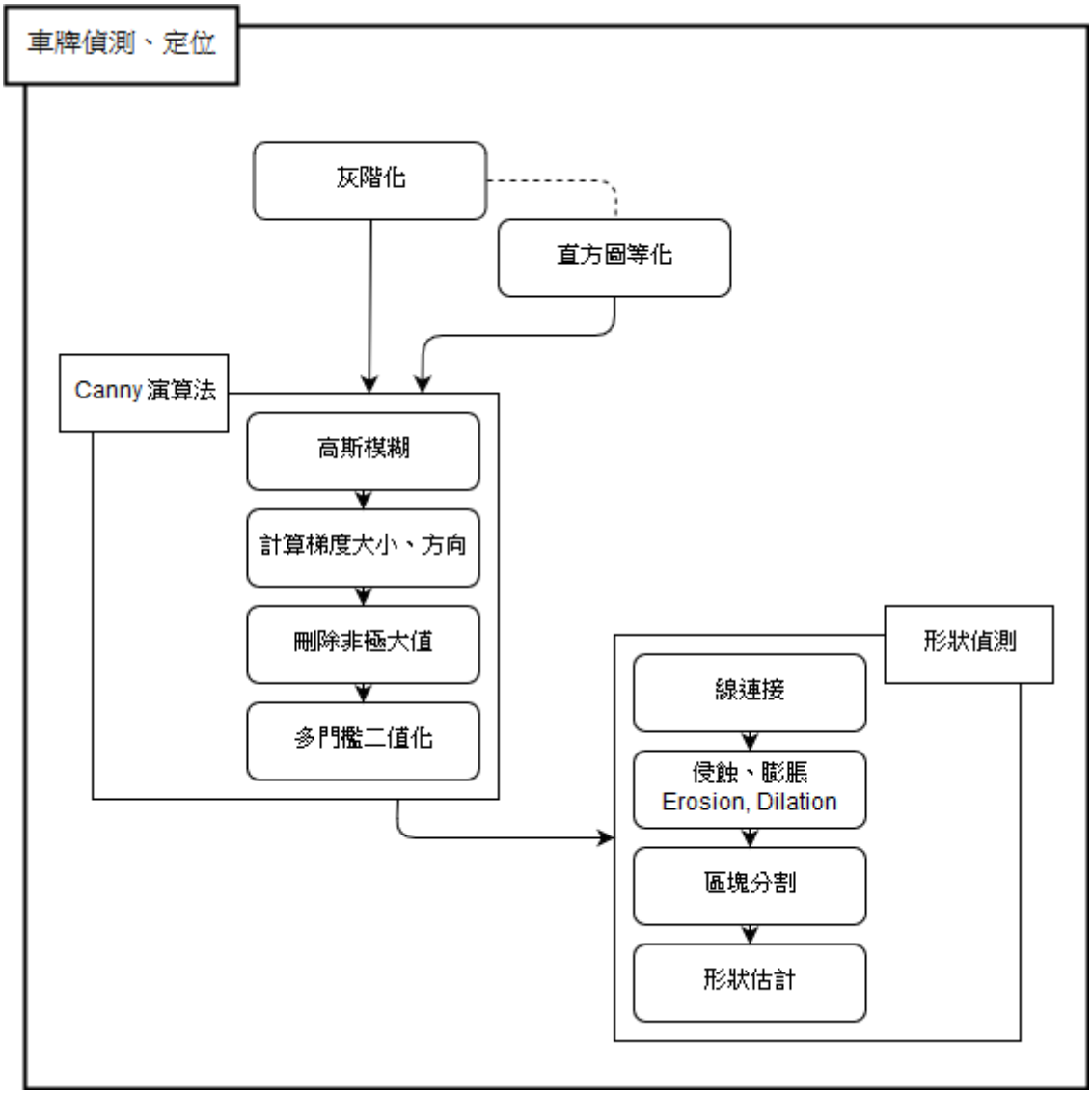
```
%%%%----  example2  ----%%%%

cimg = imread('../image/car_demo1.jpg');    % 讀圖片
gimg = rgb2gray( cimg );                              % 灰階化

```

```
%%%%----  畫出折線圖  ----%%%%

h = 692;                    % 設定擷取的高度

figure()
subplot(2,1,2)
plot( gimg(h,:) )      % 畫折線圖
set(gca, 'xLimSpec', 'tight'); % 設定圖軸貼齊

%%%%----  在原本的圖片上描上紅線  ----%%%%

%-- 做出遮罩
mask = zeros( size(cimg( h , : , : )) );
mask = uint8( mask );
mask( 1 , : , 1 ) = 255;     % 設定成紅色的

%-- 調和原本的顏色和遮罩的顏色
alpha = 0.7;
cimg( h , : , : ) = (cimg( h , : , : )*(1-alpha) ) + mask*alpha ;

subplot(2,1,1)
imshow( cimg )

```

因為從亮度圖上來觀察，會發現車牌與周遭場景不同的亮度分布，所以以這點來發展一套捕捉車牌的方法。

以此特徵發展的方法

- 前處理
  - 一開始先使用灰階化降低計算量、去除顏色差異。
  - 再做直方圖等化，這個方法可以增加影像的對比度，減少亮度不足對這套方法的影響。
- 邊界擷取
  - 使用 canny 演算法來求取影像的邊界(也就是梯度變化大的地方)，
    其中 canny 可以消除**粗邊界** (使用 sobel 就會求比較粗的邊界，在不同的任務中會有不同的影響)。
- 形狀偵測
  - 線連接：將兩個足夠近的 pixel 連接，讓車牌部分形成一整個黑色區塊。
  - 侵蝕膨脹：填補車牌黑色區塊的縫隙，並去除車牌部分與環境的區塊連通性。
  - 區塊分割：根據連通性，將圖片分割成數個區塊
  - 形狀估計：計算每個區塊的形狀，是否接近平行四邊形。

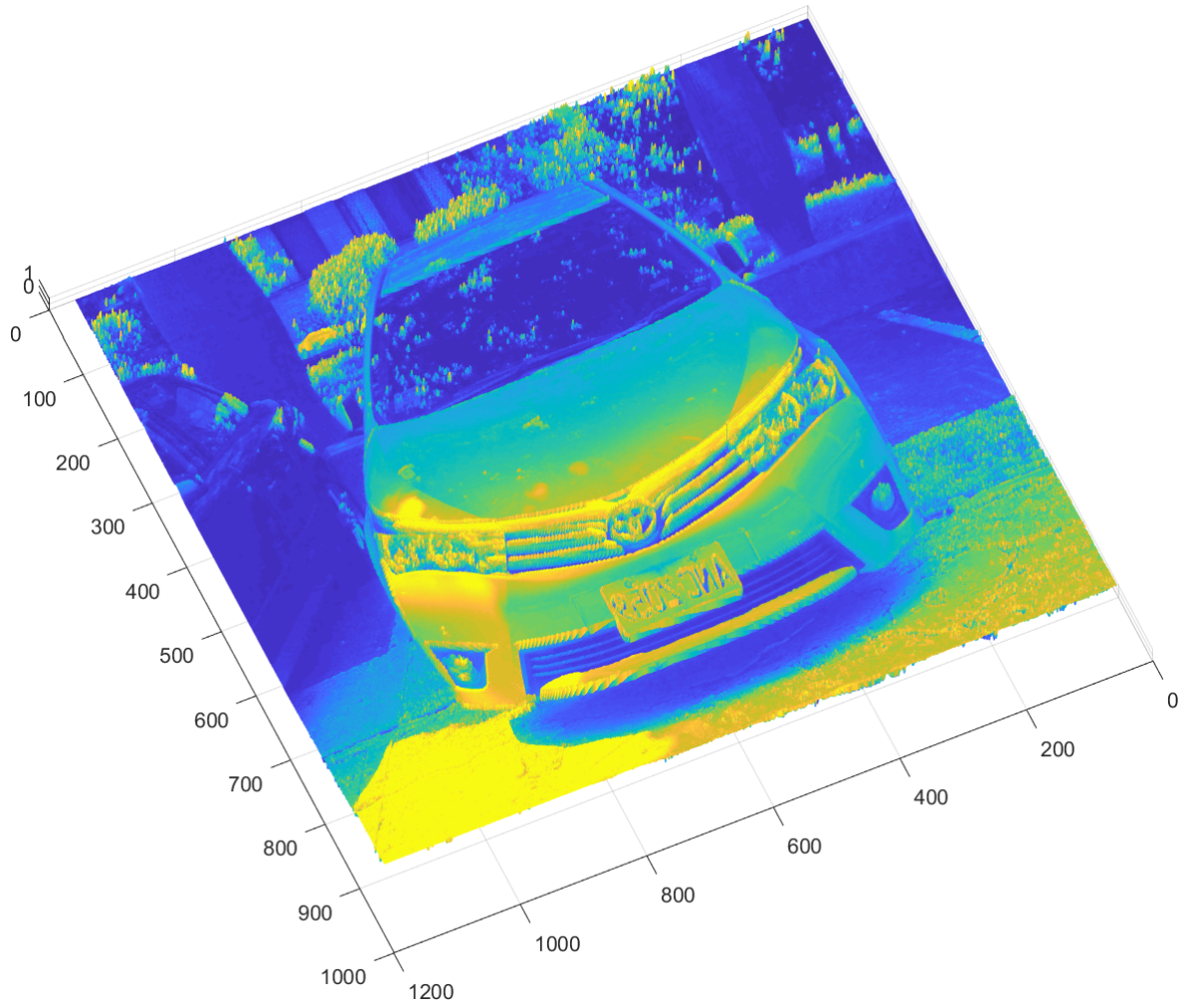


其他觀察的方法

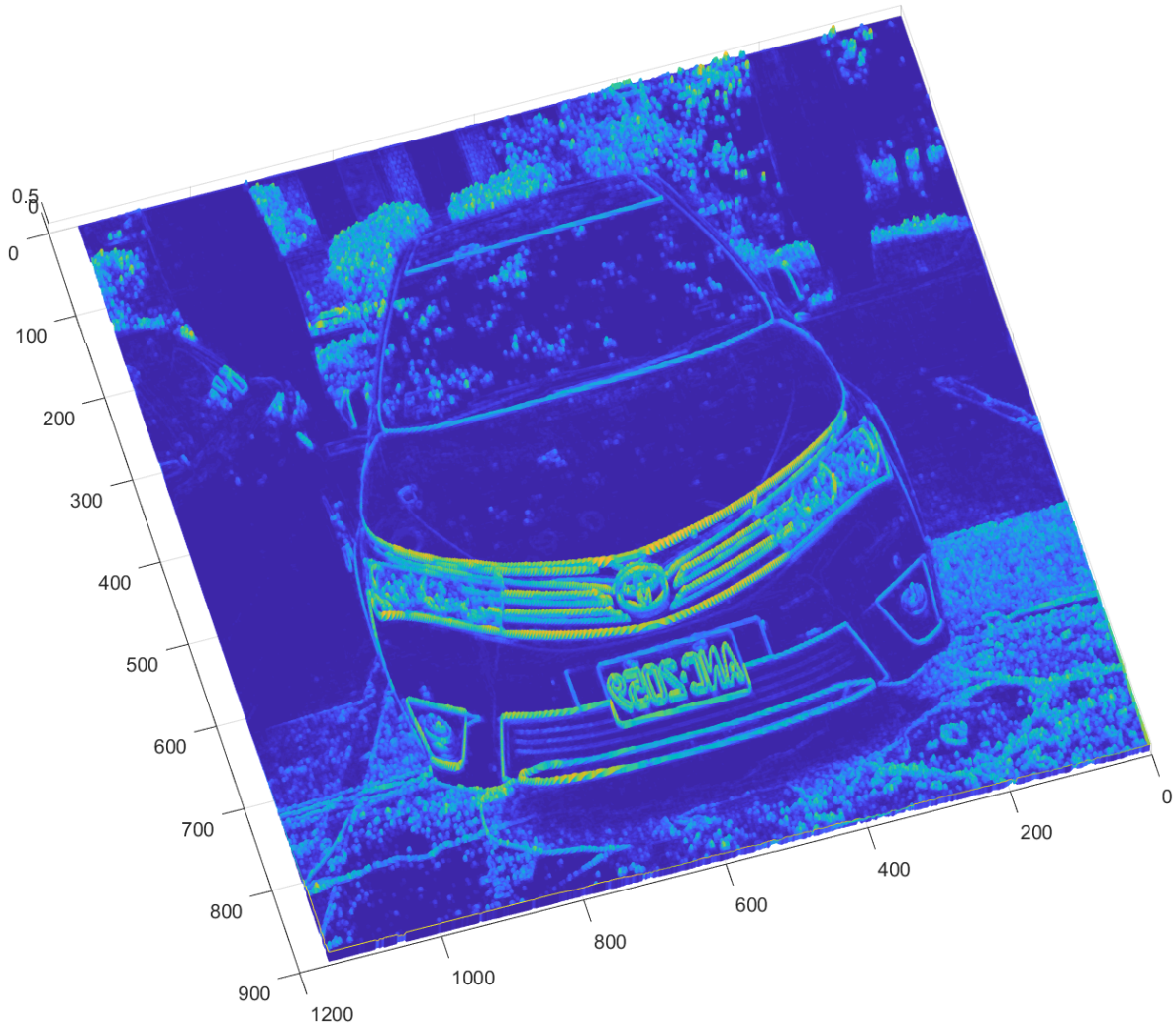
除了直接從原本的(未經處理)的圖片觀察有甚麼特徵。
(如果發現只有車牌獨有的特徵，那就可以發展一套方法來捕捉)
也可以先將圖片做一些處理再來觀察。

例如說使用 `mesh` 來觀察圖片，圖片就會變成許多**山峰**、**山谷**的樣子；
計算局部標準差，可以將**高原**的部分去除掉；
套用最大值濾波，會形成小的高原，對應於形態學中的膨脹；
套用最小值濾波，則會造出盆地，對應於形態學中的侵蝕 ...

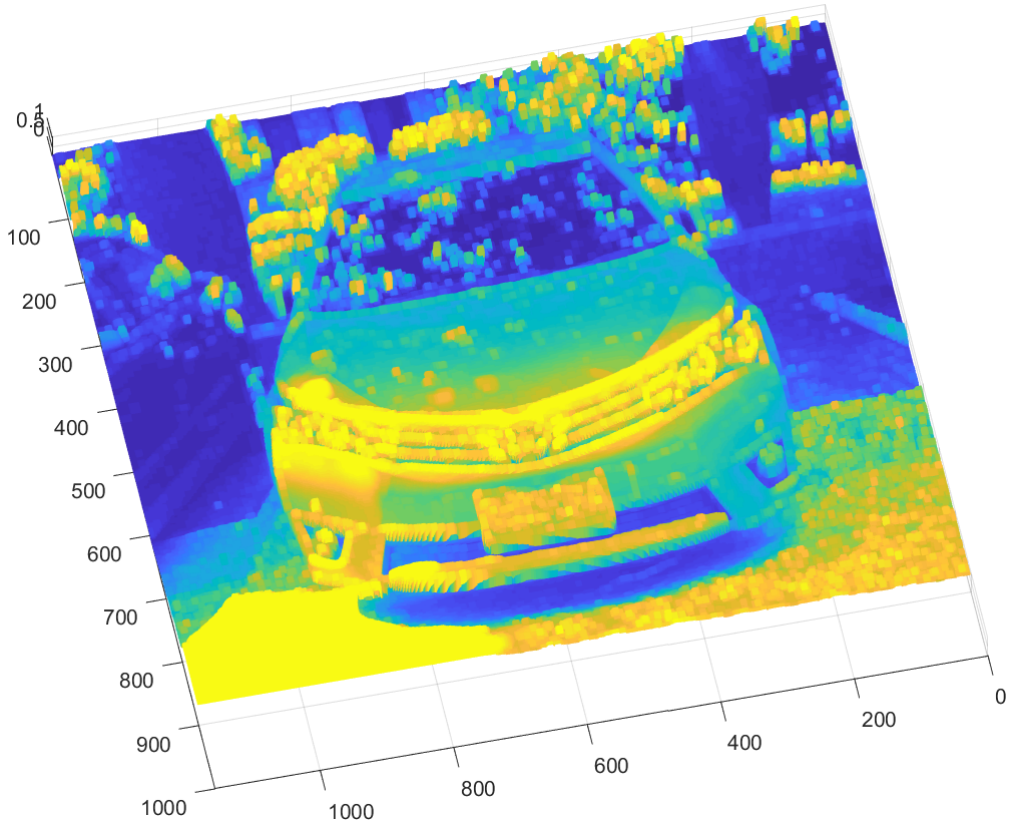
使用 `mesh` 可以用 3D 的圖像來顯示。(像地理課上那種地形、山脈模型)


1
0
0
100
200
300
400
500
600
700
800
900
1000
1200
0
200
400
600
800
1000

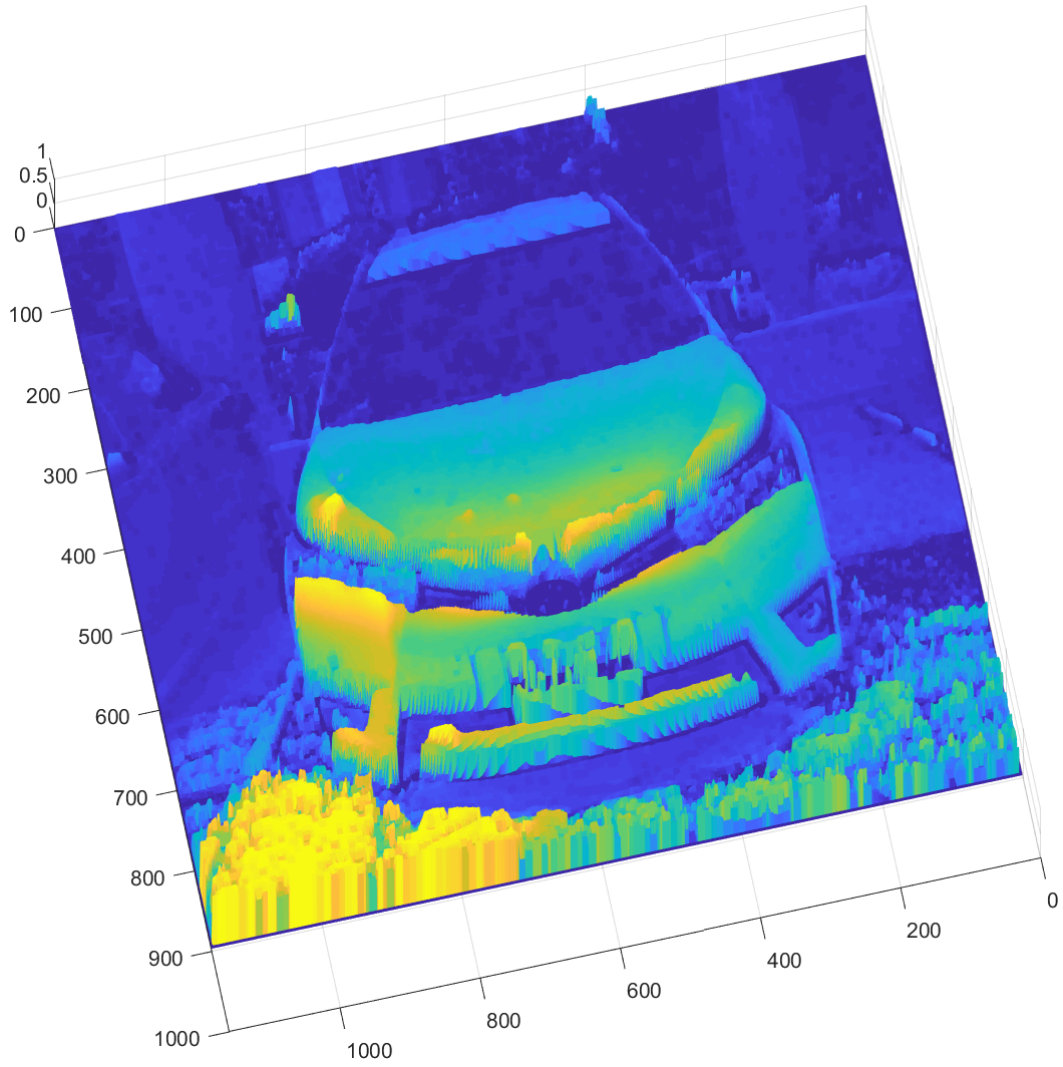
計算每個小區塊的 `std` 再顯示。



套用最大值遮罩



套用最小值遮罩


1
0.5
0
0
100
200
300
400
500
600
700
800
900
1000
1000
800
600
400
200
0

灰階化

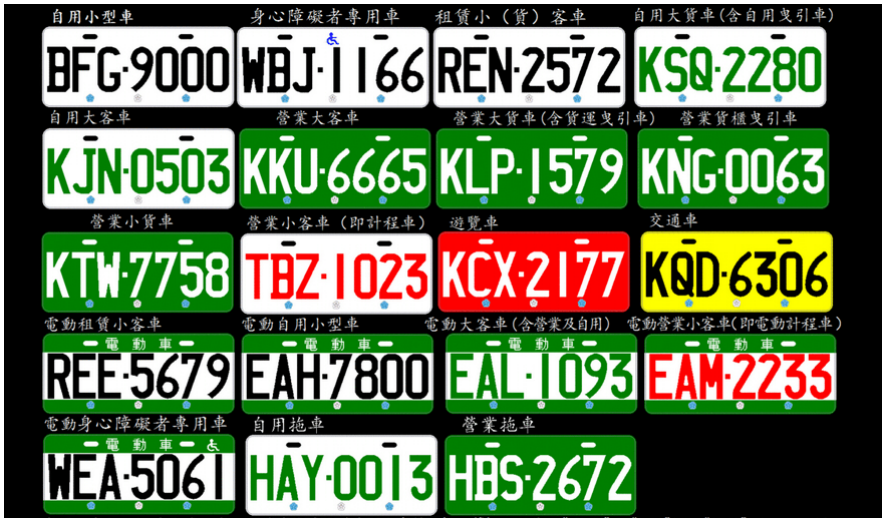
首先會對處理的影像做灰階化。

優點：

0. 後續處理的資料量變少(計算量變少)。
1. 以(感覺上)比較自然的方式計算梯度。
   (因為人眼所看到的**邊緣**是由 **RBG混合**之後呈現的**邊緣**，所以計算**平均值**的梯度**感覺上比較自然**)
2. 忽略顏色差異。

缺點：

0. 顏色資訊消失。(如果捕捉的是紅色車牌，因為整面的紅色在一般場景中並不常見，那直接透過顏色來定位或許會比接下來一系列處理要好。)



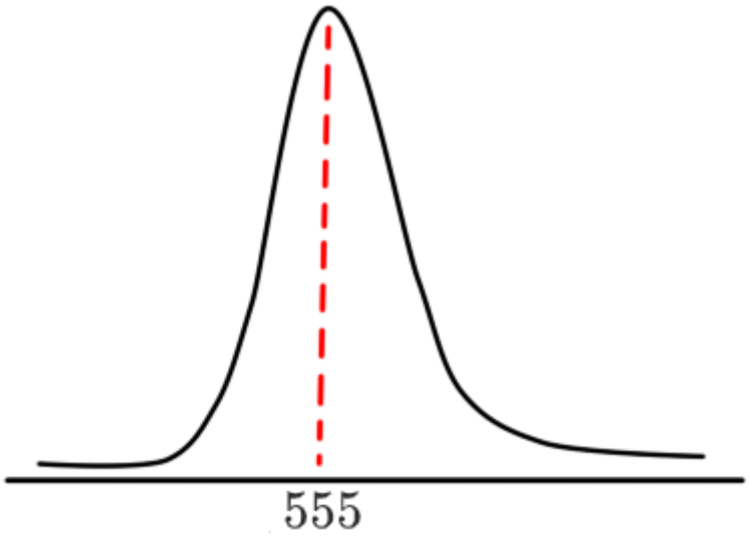
在 `matlab` 可以使用 `rgb2gray` 來求得灰階圖片，其中它的計算如 `0.2989 * R + 0.5870 * G + 0.1140 * B` ，係數是根據肉眼對顏色的敏感度決定出來的。
從係數裡可以看到肉眼**對綠色比較敏感**，關於這裡敏感粗略的解釋：

> 進行一個實驗，假設有一個綠色光源以及一個藍色光源，慢慢的把它們的能量增加，那會先注意到綠色光源。

關於灰階的加權數值 (上)

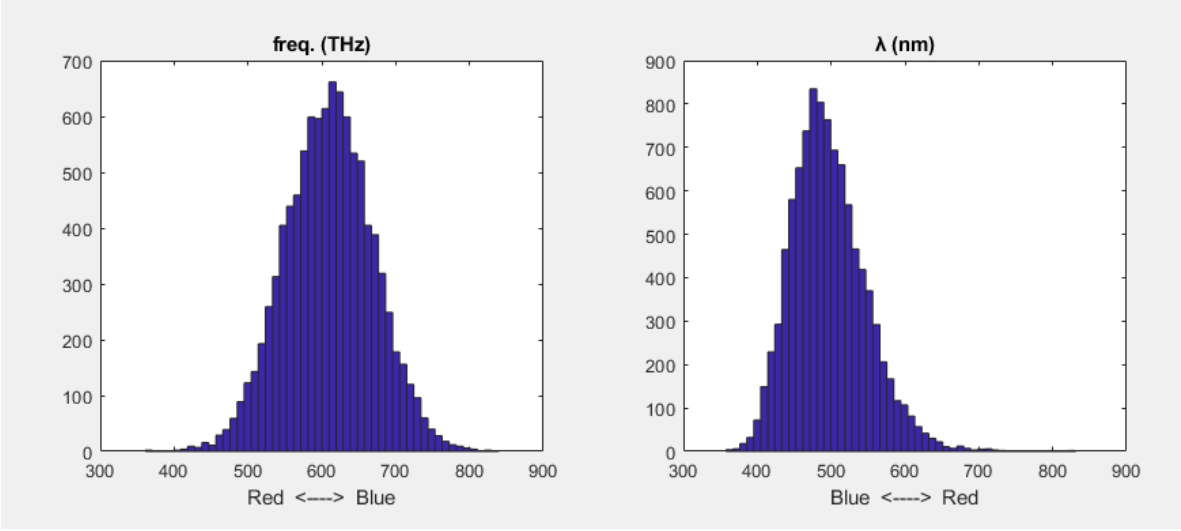
根據 wiki 上對可見光的描述。
一般人眼可感受的波長範圍是 $390 \sim 700$ nm；
對應頻率為 $430 \sim 790$ THz；
視力正常時對 555 nm 的波長最敏感。

在搭配剛剛提到 `matlab` 中灰階所使用的加權數值是 $[\,0.2989\,,\,0.5870\,,\,0.1140\,]$ 。
也許你可以把人眼對於可見光的敏感程度畫成一個類似常態分布，但左邊比較陡峭的圖，類似下面這樣。
模擬在綠光最敏感、紅光(長波長)次之、藍光(短波長)最弱。



如果你假設人眼對頻率的敏感度呈現某種分布，那你可以透過 `matlab` 來模擬，看看結果和認知上是不是一樣。

一開始我假設人眼對頻率呈現常態分布，(因為頻率和能量呈正比)
再轉換到波長上來看，(頻率和波長呈反比)
就發現原本常態分佈的圖變成左邊比較陡峭，跟實驗結果相似 (可以在透過平移 freq. 部分的圖讓 λ 的高峰落在 555 nm)。
所以這個假設還算合理，你可以把人眼對頻率的敏感度看作常態分佈，而那個特殊的加權值 $[\,0.2989\,,\,0.5870\,,\,0.1140\,]$ 也並不是那麼不自然。



模擬的範例程式碼

```
%%%%----  example4  ----%%%%
c = 300000;
figure()

%%%%----  if frequency is normal  ----%%%%
h1 = 430;
h2 = 790;
sigma = (h2-h1)/6;
mu = (h2-h1)/2 + h1;

N = 10000;
res1 = randn(N,1);
res1 = res1 * sigma  + mu;
res2 = c ./ res1;
```

```
15
16  subplot(2,2,1)
17  hist( res1 , 50 );
18  title('freq. (THz)')
19  xlabel('Red  <---->  Blue')
20
21  subplot(2,2,2)
22  hist(res2,50)
23  title('λ (nm)')
24  xlabel('Blue  <---->  Red')
25
26  %%%%----  if wavelength is normal  ----%%%%
27  w1 = 390;
28  w2 = 700;
29  sigma = (w2-w1)/6;
30  mu = (w2-w1)/2 + w1;
31
32  N = 10000;
33  res1 = randn(N,1);
34  res1 = res1 * sigma  + mu;
35  res2 = c ./ res1;
36
37  subplot(2,2,3)
38  hist( res2 , 50 );
39  title('freq. (THz)')
40  xlabel('Red  <---->  Blue')
41
42  subplot(2,2,4)
43  hist(res1,50)
44  title('λ (nm)')
45  xlabel('Blue  <---->  Red')
```

關於灰階的加權數值 (下)

灰階的加權數值可以用 `matlab` 那組 $[0.30, 0.59, 0.11]$ ，也可以單純相加除以三 $[1/3, 1/3, 1/3]$ ，或是其他的加權比例，反正只要加起來等於 1 ，最後得到的都是一張**灰階圖**。

但實際上，不同的數值算出來的結果當然會不同，具體來說到底會產生那些差異？

這裡以 `matlab` 的 `rgb2gray` 和 **相加除以三** 為例： (後續簡稱 A 和 B)

原 *I*	
A	

B	

從處理後的圖片上看不出甚麼差別 (也許有些人會覺得上面的比較亮一點)。

那這時候就有些方法可以用了，這裡從最簡單的兩張圖相減開始。

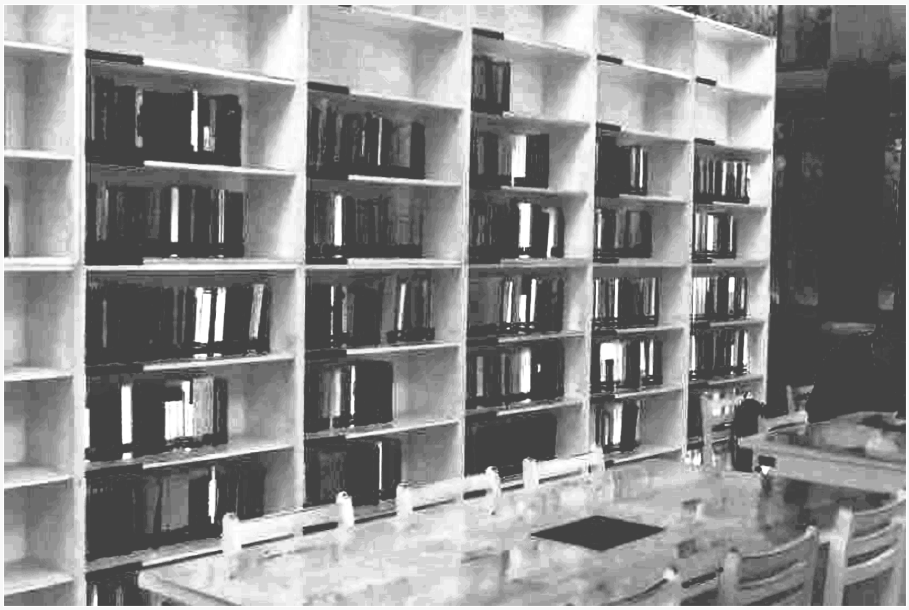
$A-B$	
$B-A$	

第一張圖可以略微看到有些東西特別亮，第二張圖則看不太出甚麼來。

這時候就要就是要把小範圍的亮度映射到大範圍的亮度。而映射的方法有很多，線性映射、指數型映射、或是基於統計的直方圖等化方法。每個方法都有各自的優缺點，如果只是單純拿來肉眼觀察，可以都拿來做做看，挑一個比較看得出差別的。

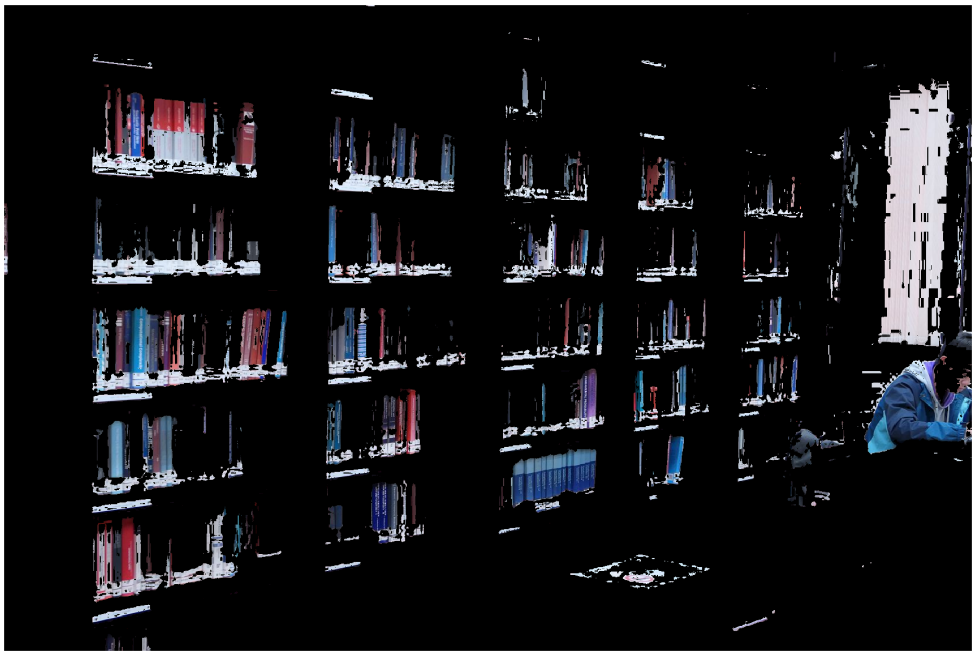
唯一要注意的就是，如果原本的圖像標準差非常小(可能所有數值都集中在 $[0, 5]$ 之間)，那做了轉換之後得到的結果可能會沒有意義，因為那一點點的差距可能是雜訊(浮點數取整、圖片儲存方式、壓縮)造成的，這就像是把一小段鋸齒狀的雜訊放大來看，發現變化很劇烈，但其實變化的幅度非常小。

在這裡我挑了直方圖等化來調整亮度，因為這個方法可以自適應 (不用人為設定參數)。
使用 `matlab` 的函數 `histeq()` 得到下列這樣的結果。

`histeq(a-b)` `h1`	
`histeq(b-a)` `h2`	

接著再把圖像疊在原本的影像上，觀察看看那些部分被**強調**或是**弱化**了。
可以單純使用 `.*` 來看，因為數值已經轉到 $[0, 1]$ 了，但有時候變化太連續，會看不出來有甚麼差別；
所以我這裡會先把 `h1` 的值做二值化，再做 `.*`

在做二值化時，可以用 `matlab` 中的 `imbinarize` 預設會套用 Otsu's 演算法求得閥值進行二值化。
(Otsu's 也是自適應的方法，不用人為調配參數)
另外也有 `graythresh` 可以套用 Otsu's 演算法單純求閥值而不對圖像做處理。

`I.*bh1`	
`I.*bh2`	

最後得到的結果，是**紅**、**藍**會被弱化，而**黃**、**綠**會被強化。

再回去看原本的加權數值： $[0.30, 0.59, 0.11]$ v.s. $[1/3, 1/3, 1/3]$
紅、**藍**弱化以及**綠**強化的結果很自然，
而黃色 (紅色+綠色)，因為綠色強化的幅度遠大於紅色弱化的幅度，所以黃色在使用 `rgb2gray` 時會比**相加除以三** 顯得更亮。

灰階圖的其他應用

深度圖

直方圖等化

如果整體亮度都聚集在一個小範圍內(標準差很小)，那車牌與周圍環境的亮度差距也會變小(注意亮度的刻度只有到90)。我們可以透過直方圖等化來修正這個問題。

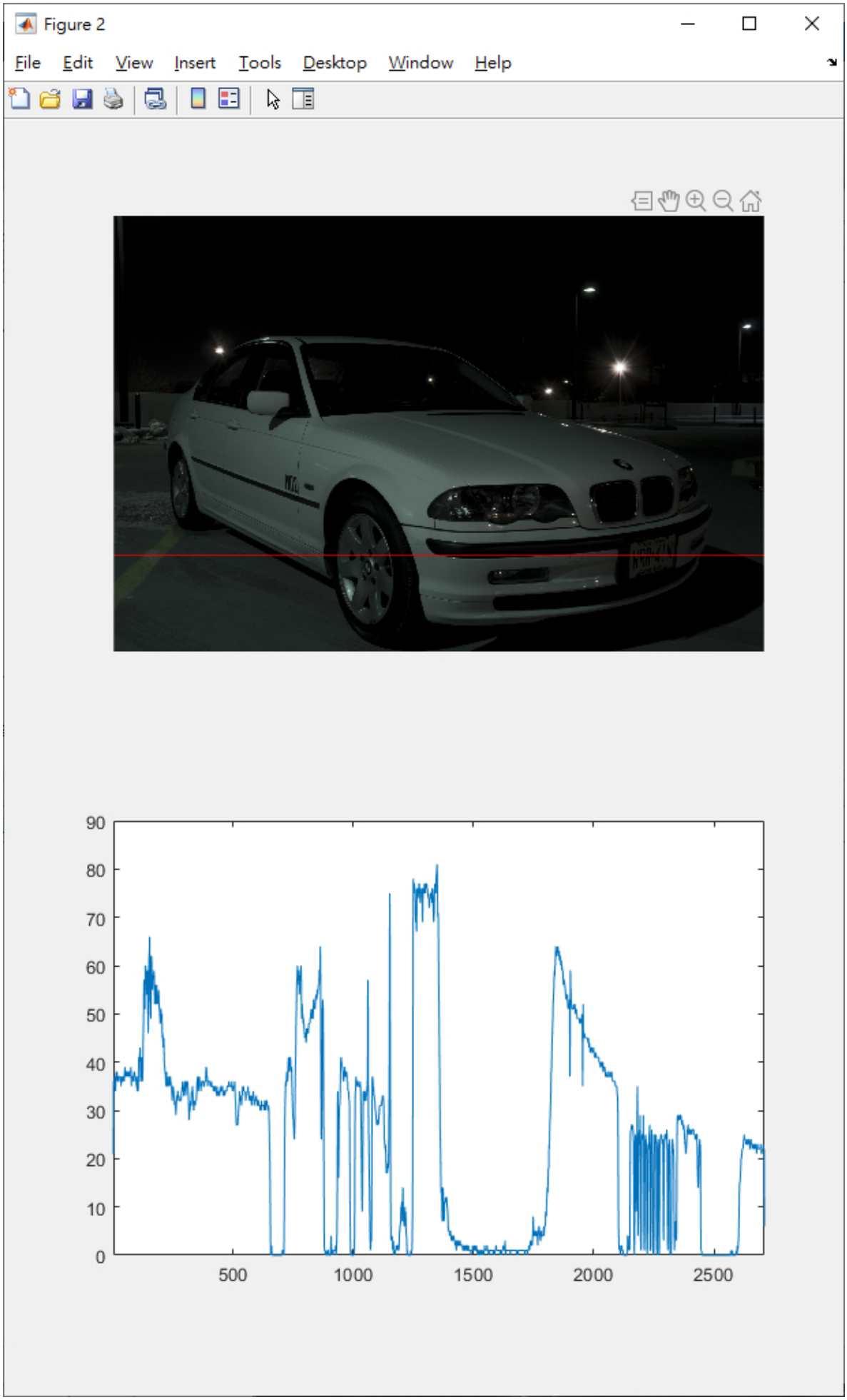


Figure 2
File Edit View Insert Tools Desktop Window Help
90
80
70
60
50
40
30
20
10
0
500
1000
1500
2000
2500

- 使用 `histeq`

```
%-- process2
fileName = 'car_demo2_gray';
gimg = imread( [fileName '.jpg'] );

eqimg = histeq( gimg );

figure()
imshow( eqimg );

figure()
imhist(eqimg)

imwrite( eqimg , [fileName '_eq.jpg'] )
```

Figure 5

File Edit View Insert Tools Desktop Window Help

Figure 6

File Edit View Insert Tools Desktop Window Help

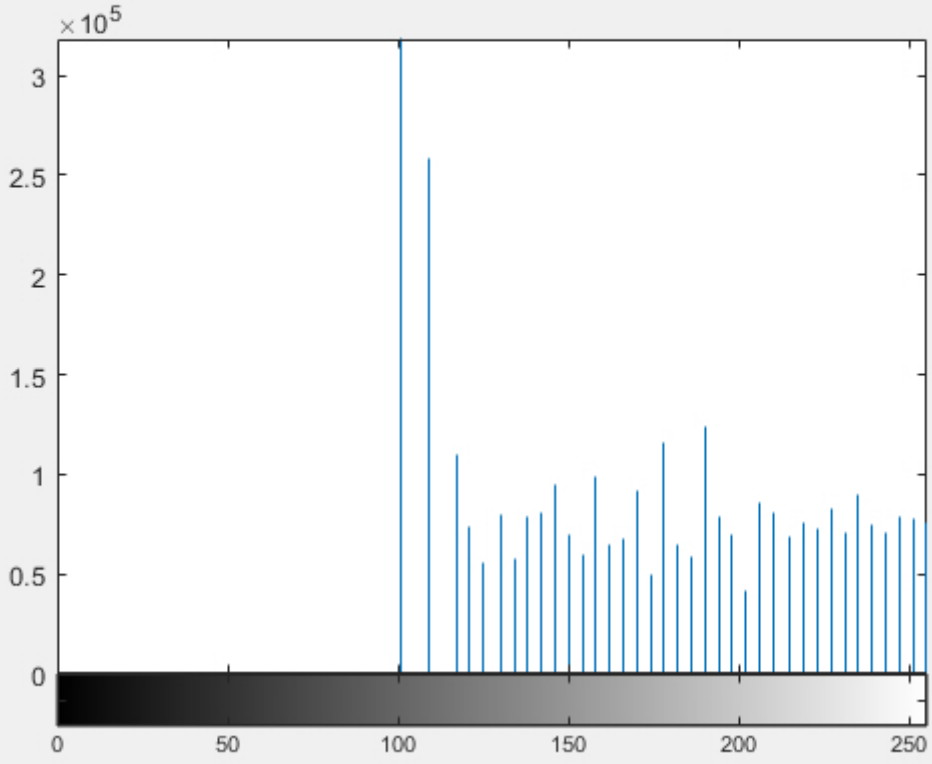
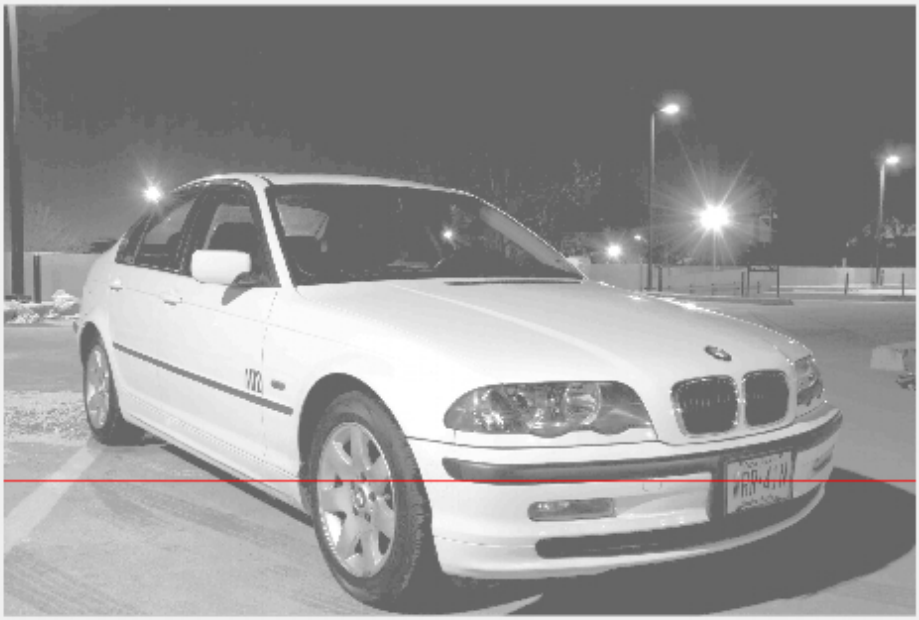
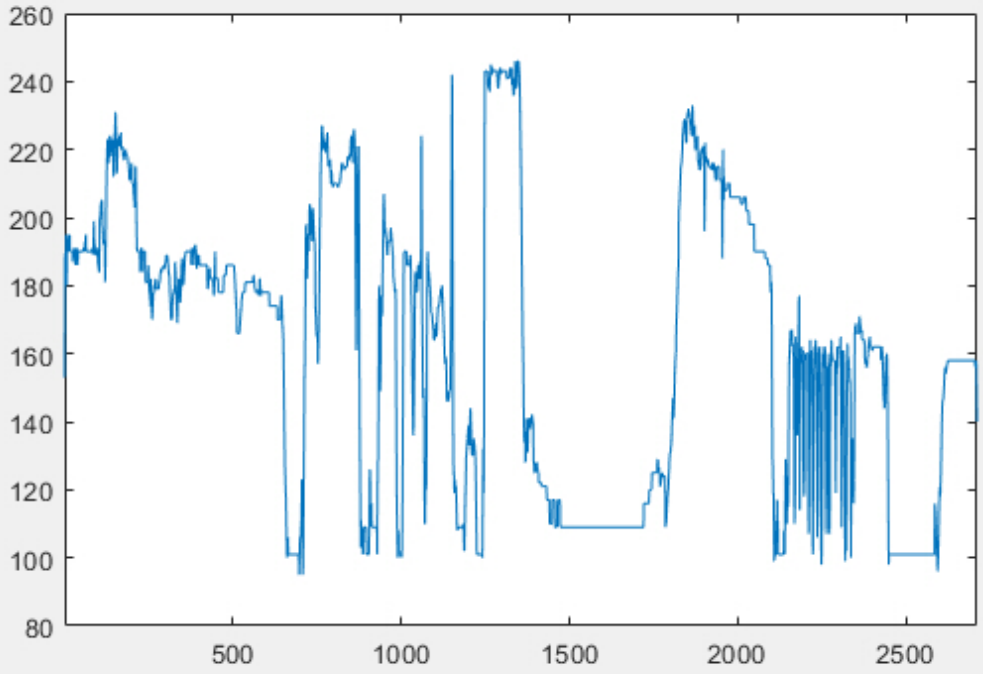


Figure 5

File Edit View Insert Tools Desktop Window Help



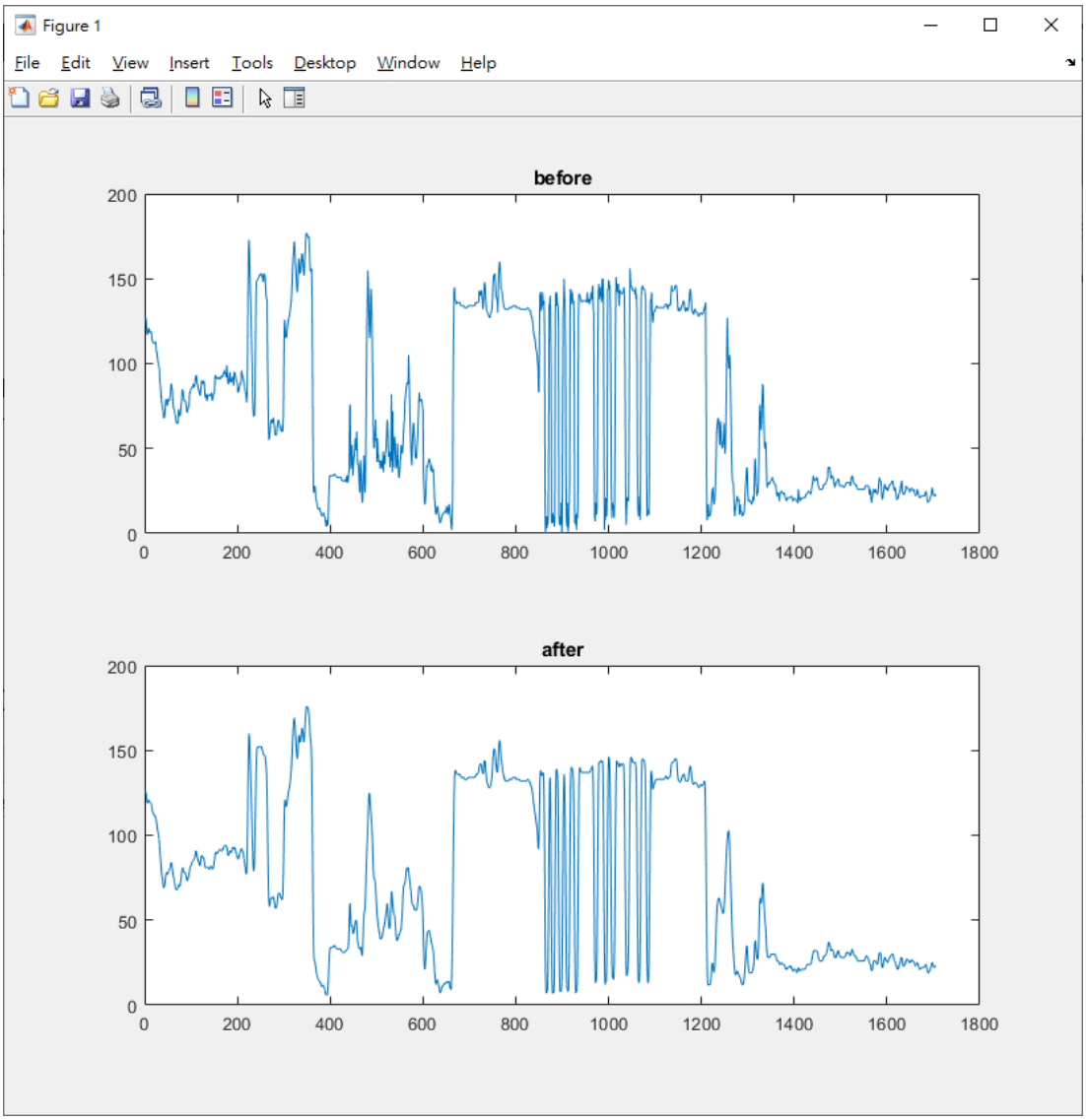

260
240
220
200
180
160
140
120
100
80
500
1000
1500
2000
2500

模糊 (Canny演算法)

接下來會先對圖片做模糊處理，這會消滅變動小的變化。(仔細觀察會發現鋸齒消失了)

這裡會有兩個參數需要選擇，**遮罩大小**與**遮罩類型**，這邊使用的是高斯遮罩，σ 設為 1.2，遮罩大小則設為 $[5,5]$

- 使用 `gausFilter` 和 `imfilter` (詳見....章)




```
%-- process3
fileName = 'car_demo1_gray';

gimg = imread( [fileName '.jpg'] );

h = 591;

%-- 高斯模糊
sigma = 1.2;
gausFilter = fspecial( 'gaussian' , [5,5] , sigma );
blur_gimg = imfilter( gimg , gausFilter , 'replicate');

figure()
subplot(2,1,1)
```

```
plot( gimg(h,:) )
title('before')

subplot(2,1,2)
plot( blur_gimg(h,:) )
title('after')

imwrite( blur_gimg , [ fileName '_blur.jpg' ]  );
```

選擇 Gaussian 遮罩的原因是因為參數會比較好選擇(大部分參數都會有好的效果)。只要取一個適當的 $\sigma = 1.2$ ，不論遮罩大小為何效果都不會偏差太多。
但如果選擇的是平均遮罩，那效果會嚴重的被**遮罩大小**改變。

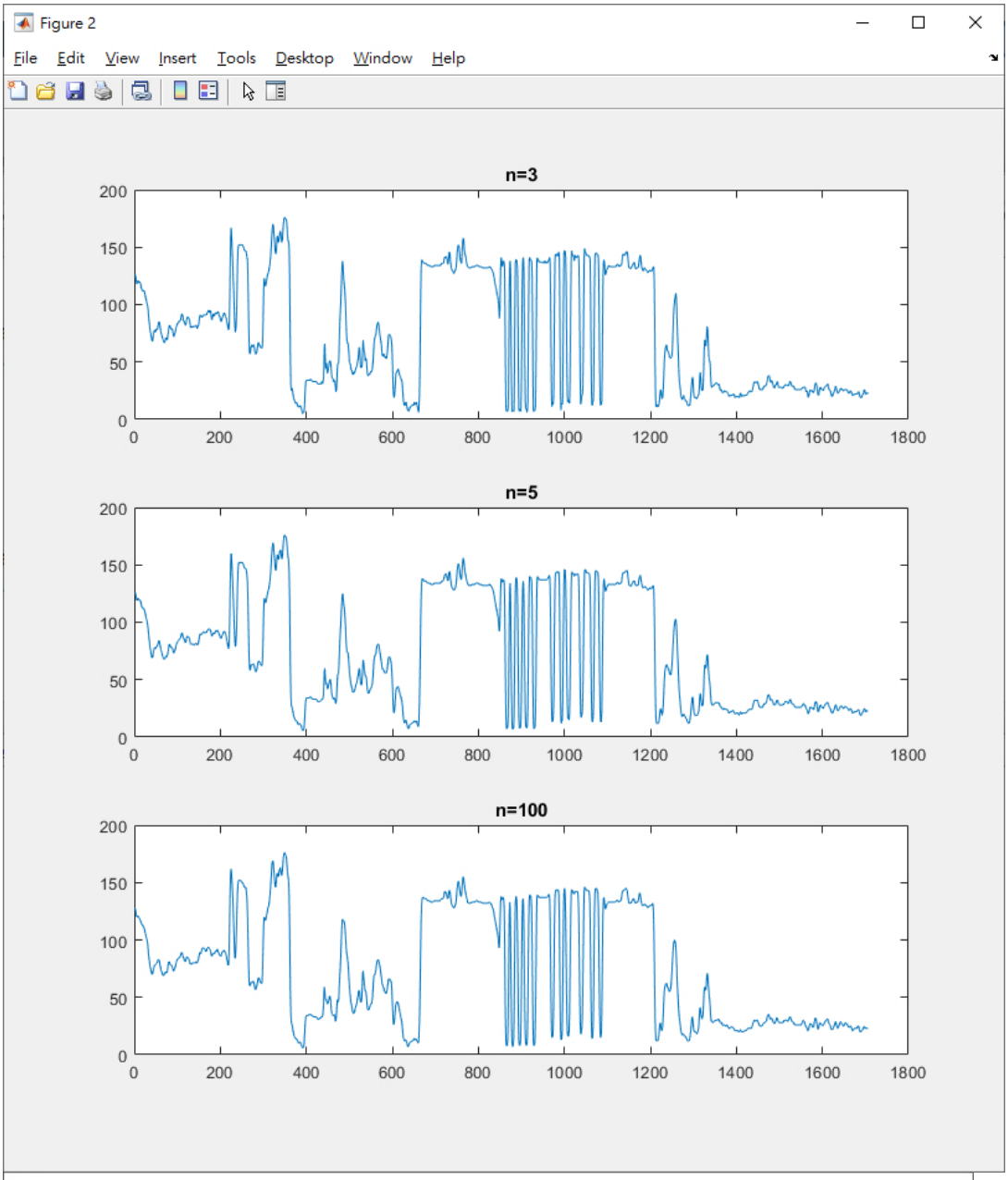


Figure 2
File Edit View Insert Tools Desktop Window Help
n=3
n=5
n=100

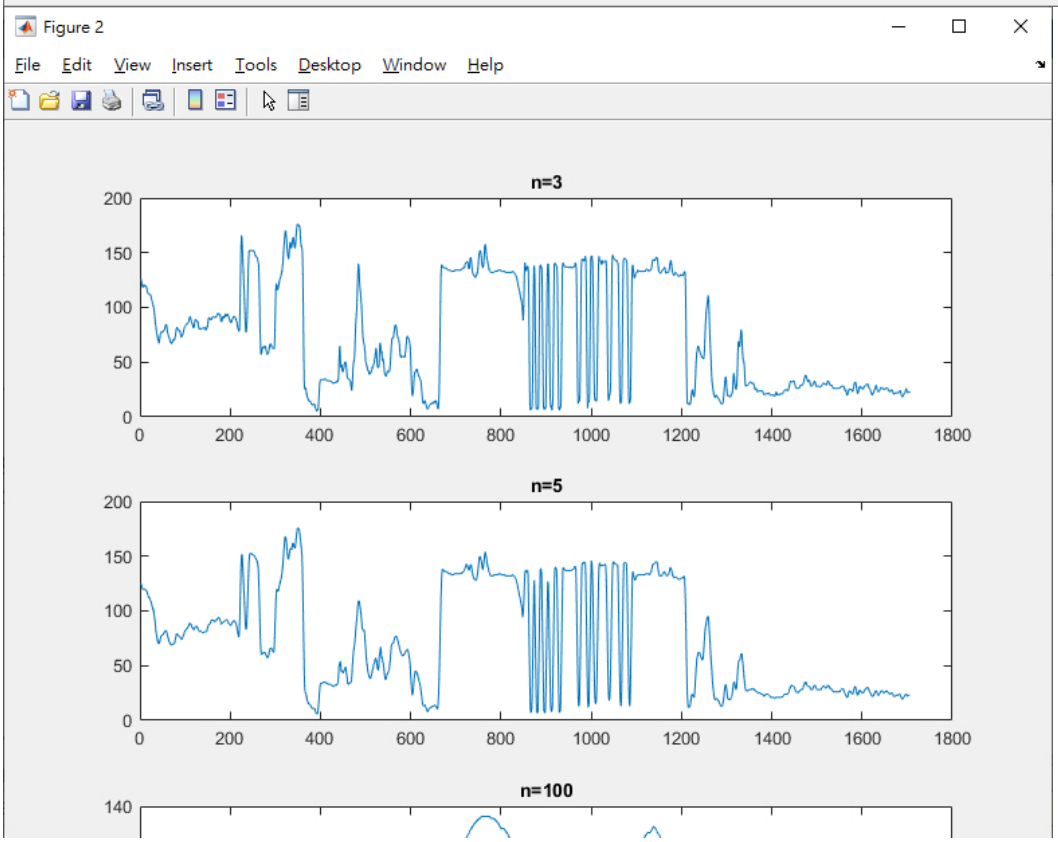
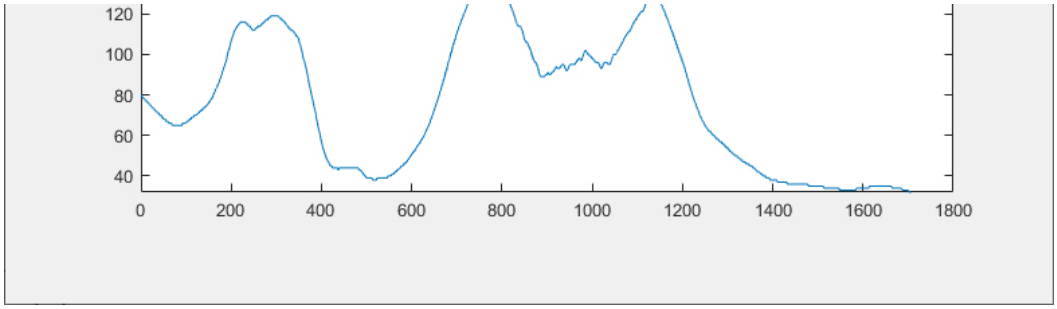


Figure 2
File Edit View Insert Tools Desktop Window Help
n=3
n=5
n=100

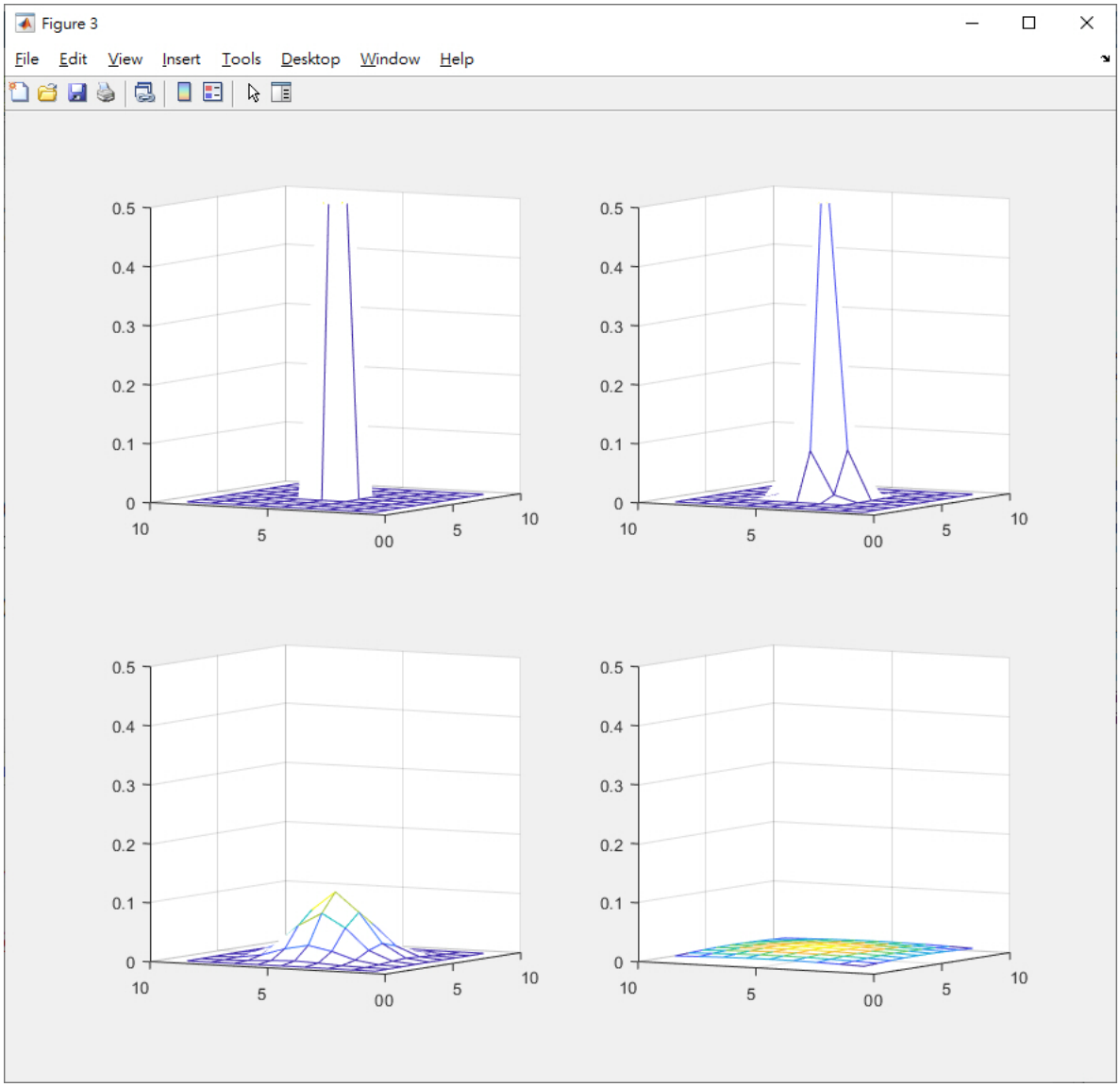


這是因為高斯函數的定法在遠離中心點的時候會快速減少，底下附上生成 Gaussian 遮罩的程式碼，實際使用時用 Matlab 內建的比較方便。

$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}((x-\mu)/\sigma)^2}$

```
%-- Gauss
function mask = Gauss( n , sigma )

mu = 0.5 + n/2;

for x=1:n
    g(x) = (1/ (sigma*sqrt(2*pi)  ) )*exp( -0.5*( (x-mu)/sigma )^2   );
end

%-- 使總和為 1
g = (1/sum(g))*g;

mask = g'*g;
end
```

另一方面，高斯遮罩在 σ 極大的情況下相等於平均遮罩，σ 極小則相當於不取遮罩。




```
%-- example4
n = 9;
sigmas = [ 0.1 , 0.5 , 1.2 , 5 ];

figure()
%-- 視角參數
az = -60;
el = 5;

for i=1:4
    sigma = sigmas(i);
    gausFilter = fspecial( 'gaussian' , [n,n] , sigma );
    subplot( 2,2,i )
    %-- 繪圖
    mesh( gausFilter );
    %-- 設定圖軸
    zlim([ 0 0.5])
    %-- 設定視角
    view(az,el);
end
```

計算梯度、梯度方向 (Canny演算法)

遮罩使用的是 Sobel 遮罩。 (詳見第 ... 章)
可以透過 mesh 來感受一下大小

```
%-- process4
fileName = 'car_demo1_gray_blur';

gimg = imread( [fileName '.jpg'] );

xMask = [ -1 0 1 ; -2 0 2 ; -1 0 1 ];
yMask = [ -1 -2 -1 ; 0 0 0 ; 1 2 1 ];

%-- 轉成 double 之後計算可能會有負數
gimg = im2double( gimg );
gx_img = imfilter( gimg , xMask , 'replicate');
gy_img = imfilter( gimg , yMask , 'replicate');

%-- 計算梯度大小
gNorm = sqrt( gx_img.^2 + gy_img.^2 );

%-- 範圍是 [ -pi/2 , pi/2 ]
gDirection = atan(gy_img ./ gx_img);

%-- 處理 x 方向梯度為 0 的
gDirection( gx_img==0 ) = pi/2;
```

刪除非極大值 (Canny演算法)

根據梯度方向來判斷當前值是不是區域極大值，不是的話設成非邊界點，這可以讓重疊邊界消失(因為只會保留區域極大值)。

```
%-- process5

load( 'grad.mat' )

%-- 方向分類
%-- 1: 90度
%-- 2: 45度
%-- 3: 0度
%-- 4: -45度
gDirType = ones(size(gDirection));      % 90
gDirType( gDirection > 67.5 ) = 1;      % (67.5,90]
gDirType( gDirection <= 67.5 ) = 2;     % ( ... , 67.5]
gDirType( gDirection <= 22.5 ) = 3;     % ( ... , 22.5]
gDirType( gDirection <= -22.5 ) = 4;    % ( ... , -22.5]
gDirType( gDirection <= -67.5 ) = 1;    % ( ... , -67.5]

gDirection = gDirType;

%-- 複製一份
gNormCopy = gNorm;

%-- 1 4 7
%-- 2 5 8
%-- 3 6 9
%-- 每個方向要比較的對象
masks = [ 4,6 ; 7,3 ; 2,8 ; 1,9 ];
```

```
27
28  for dir=1:4
29      for i=1:2
30          mask = zeros( 3,3 );
31          mask( 2,2 ) = 1;
32          mask( masks( dir , i ) ) = -cosd(11.25);  %-- 需要修正
33          isBigger = imfilter( gNorm , mask , 'replicate');
34
35          %-- 找到對應方向的非極大點
36          notMax = isBigger < 0;
37
38          %-- 拿比較對象那部分計算的結果
39          notMax = notMax & gDirType==dir;
40
41          gNormCopy(  notMax ) = 0;
42      end
43  end
```

多門檻二值化 (Canny演算法)

最後一步則是設立兩個門檻值，稱為上門檻與下門檻。
當梯度值大於上門檻時設為邊界點，梯度值低於下門檻時設為非邊界點。
當梯度值介於兩者之間且周圍有邊界點，設為邊界點；否則設為非邊界點。

線連接

這裡要使用車牌是亮度**頻繁變化**這個特徵。
只要兩個白色 pixel 的**距離夠短**，那就會是頻繁變化的部分。

這裡就要將兩個夠近的白色 pixel 挑出來，將中間的部分也填成白色；
預期這樣處理後，車牌部分就會整個被填成白色。

原圖	
canny	

線連接	

使用 `matlab` 實作，其中 `line` 是自己定義的函數。

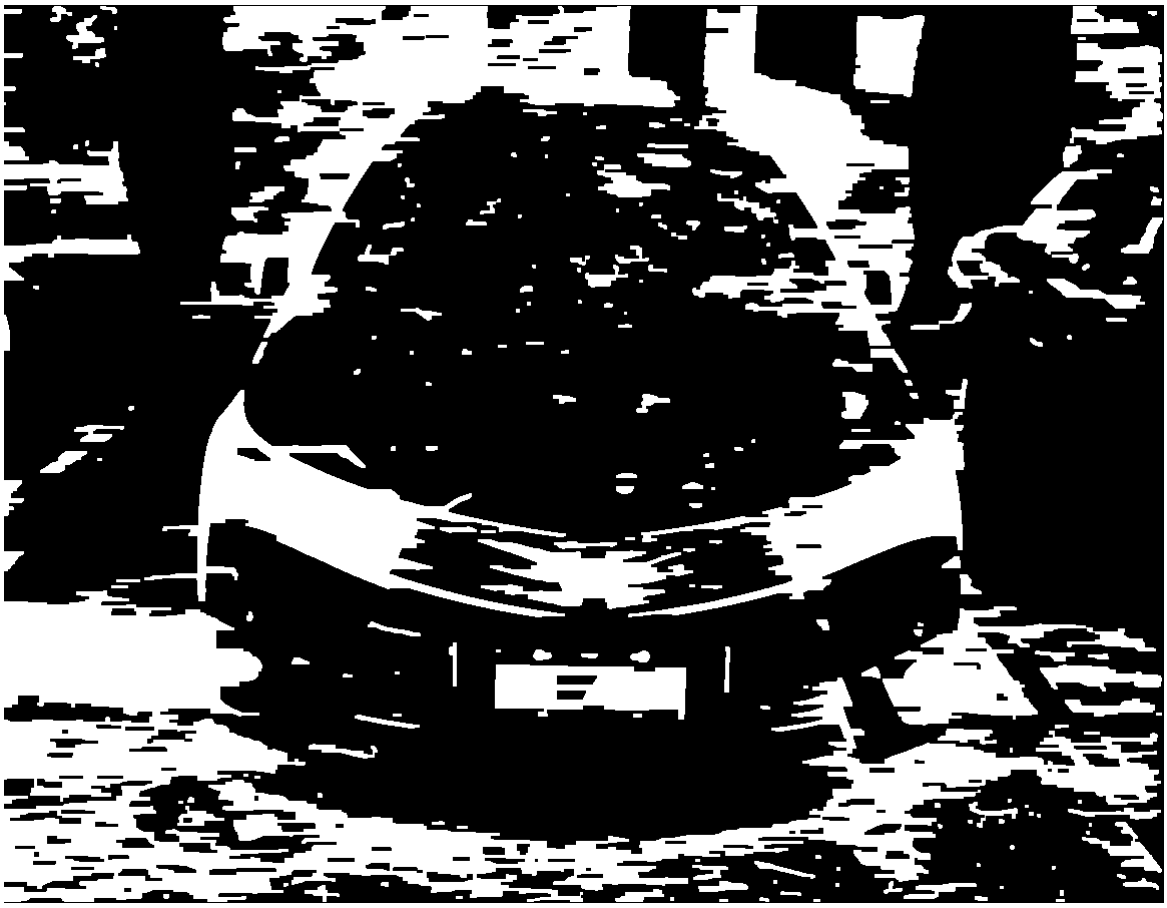
```matlab
%%%%----  example10  ----%%%%

cimg = imread('../image/car_demo1.jpg');   % 讀圖片
gimg = rgb2gray( cimg );                                % 灰階化

eimg = edge( gimg , 'canny' );      % canny 邊界

figure()
imshow(eimg);
imwrite( eimg , 'edge.png' );       % 存檔

figure()
Limg = line( eimg , 20 );       % 線連接
imshow(Limg)
imwrite( Limg , 'line.png' );

% 線連接的函數
function res = line( eimg , L )
[h,w] = size(eimg);

% 跑過所有的 pixel
for y=1:h
    start = 1;  % 記錄白色 pixel 出現的地方
    step = 0;   % 記錄到目前經過了幾個黑色 pixel
    for x=1:w
        if eimg(y,x)        % 走到了白色 pixel
            if step <= L   % 經過的黑色 pixel 小於門檻值
                eimg( y , start : x-1 ) = 1;
            end
            step = 0;       % 歸零步數
            start = x+1;   % 記錄位置
        end
```

```
33
34          if ~eimg(y,x)
35              step = step+1;
36          end
37      end
38
39      res = eimg;
40  end
41
42  end
```

斷開、閉合

在做完線連接後，要將影像上脆弱 (細) 的連接去除掉。
直接套用型態學上的斷開，使用 `matlab` 的函數 `imopen`

接著再套用閉合 `imclose`，填補區域內的裂縫。



到這裡為止，車牌應該要從場景中獨立出來 (和其他區塊沒有連接)。

區域分割

再來需要抓出影像中每一個區塊，這裡可以透過 `bwconncomp`。

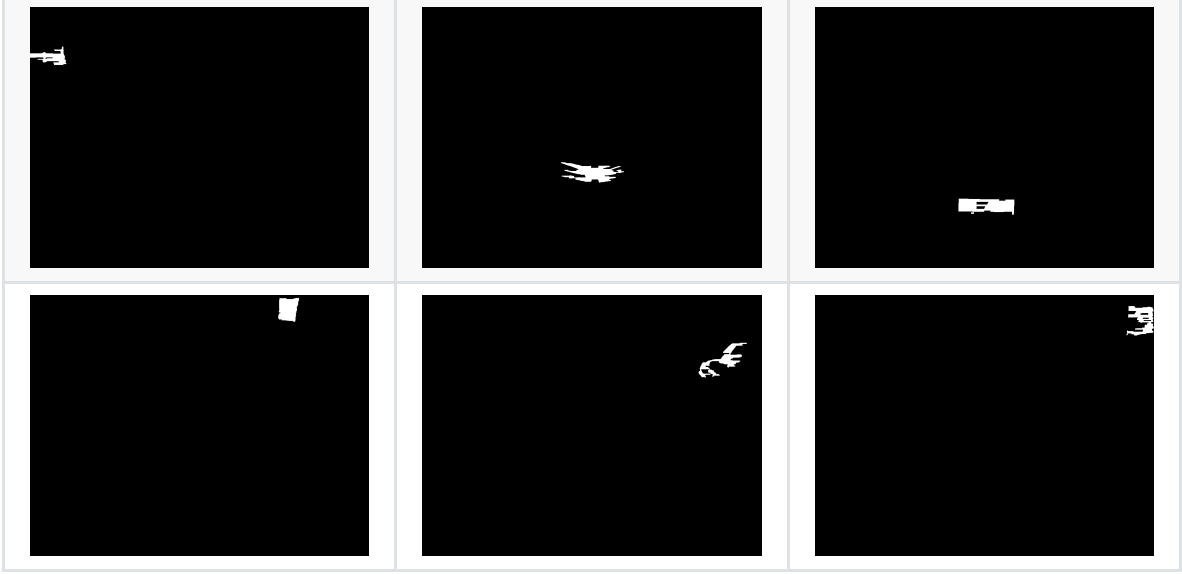
```
res = bwconncomp( cimg )
res =

  struct with fields:

    Connectivity: 8
       ImageSize: [896 1158]
      NumObjects: 248
    PixelIdxList: {1×248 cell}
```

處理後會有一些資訊，其中 `connectivity` 是分割時使用的連通方式，可以另外設定成 4-連通等等。
`NumObjects` 是總共分割出來的數量。
`PixelIdxList` 則是每個區塊的像素的索引值。(2,150) 的 pixel 索引值就會是 $2 \times W + 150$

使用 `res.PixelIdxList{1}` 來獲得索引值。(下圖左上角有一個白色區域)

```
res = bwconncomp( cimg );

img = zeros( size(cimg) );
img( res.PixelIdxList{1} ) = 1;
imshow( img );
```

最後則是設定區塊大小的上下界來篩選車牌候選人。



```
%%%%----  example12  ----%%%%

cimg = imread('../image/close.png');   % 讀圖片

res = bwconncomp( cimg );
```

```
% 設定區塊大小
inf = 3000;
sup = 10000;

cand = {};
cc = 1;
for i=1: res.NumObjects
    num = length( res.PixelIdxList{i} );
    if  num > inf  && num < sup
        cand{cc} = res.PixelIdxList{i};
        cc = cc + 1;
    end
end

cc = cc - 1;

% 全部存檔
for i=1:cc
    img = zeros( size(cimg) );
    img( cand{i} ) = 1;
    imwrite( img ,  sprintf('%02d.png' , i)  )
    imshow( img )
end
```

形狀估計

再抓出所有可能的區塊後，最後就是要判斷區塊的形狀是不是長方形。

這裡使用 `regionprops` 中的 `Extent` 功能，他是計算 $\frac{\text{區域面積}}{\text{包含區域的最小矩形面積}}$，其中這個矩形是可以旋轉的。

計算出來值高的話，那區域就會接近矩形。

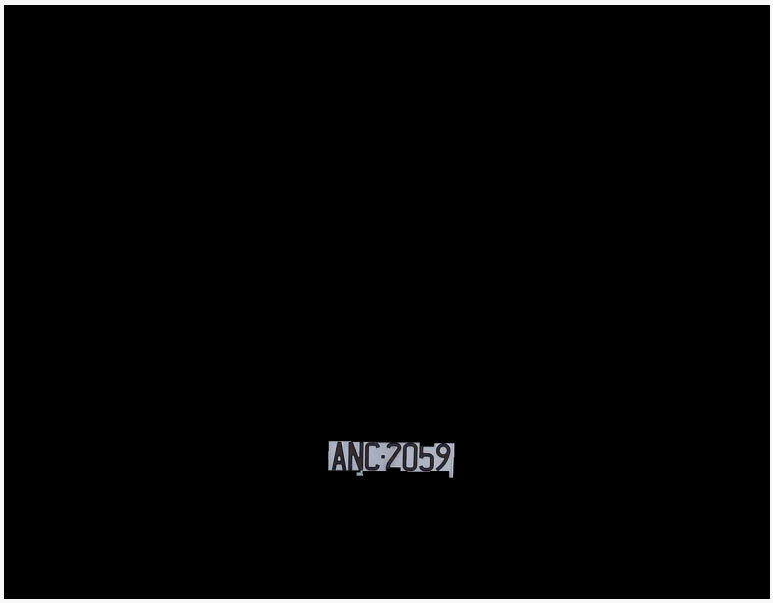
做這個之前可以先做一次 `imfill` 用來把區域內的空隙填起來。

```
%%%%----  example13  ----%%%%

cc = 6;
for i=1:cc
    eimg = imread(  sprintf( '../image/%02d.png' , i ) );   % 讀圖片
    eimg = boolean(eimg);
    eimg = imfill( eimg , 'hole' );
    ext = regionprops( eimg , 'Extent' );

    % 顯示每個求得的比例
    msg = sprintf( '%02d : %.4f' , i , ext.Extent );
    disp( msg );
end
```

得到的結果：

```
>> ex13
01 : 0.4492
02 : 0.4685
03 : 0.7776
04 : 0.7994
05 : 0.2694
06 : 0.5742
```

最後設定一個閥值，例如說 0.7 ，大於這個的就視作車牌輸出結果。

03 : 0.7776	
04 : 0.7994	

實作的程式碼

```
%%%%----  example14  ----%%%%

cimg = imread( '../image/car_demo1.jpg' );
cimg = im2double( cimg );

cc = 6;
bound = 0.7;
out_cc = 1;
```

```
for i=1:cc
    eimg = imread(  sprintf( '../image/%02d.png' , i ) );   % 讀圖片
    eimg = boolean(eimg);
    eimg = imfill( eimg , 'hole' ); % 填洞
    ext = regionprops( eimg , 'Extent' ); % 計算 extent

    % 尋找大於邊界值的
    if ext.Extent > bound
        figure()
        % 遮罩處理
        img = cimg .* eimg;
        imshow( img );
        imwrite( img , sprintf('f%02d.png' , out_cc) );
        out_cc = out_cc + 1;
    end
end
```

總結

到目前為止，已經可以從一張車子的圖片，抓取出有可能的車牌位置。
但可以發現，仍然會有誤報(抓錯)的情況，在雨天、光照不均、車牌歪斜的情況更容易造成錯誤，而要改進這些問題可能就要加入更多的判斷式，或是調整這個演算法。

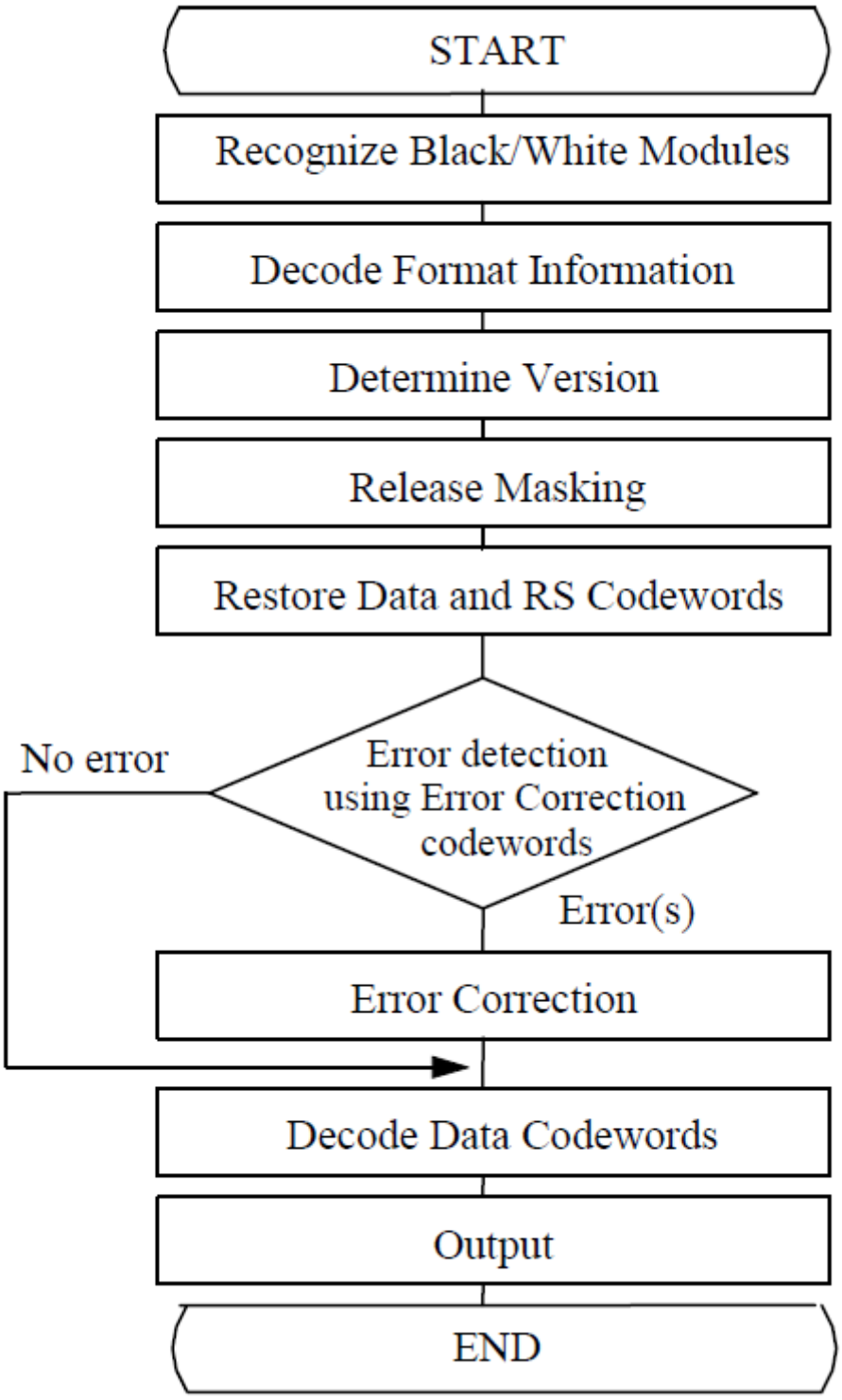
QR Code 圖片解碼

用手機拍照圖片如下：

- 不能太過歪斜、對比要足夠
- 旋轉沒關係
- 畫面中最好只有 QR code

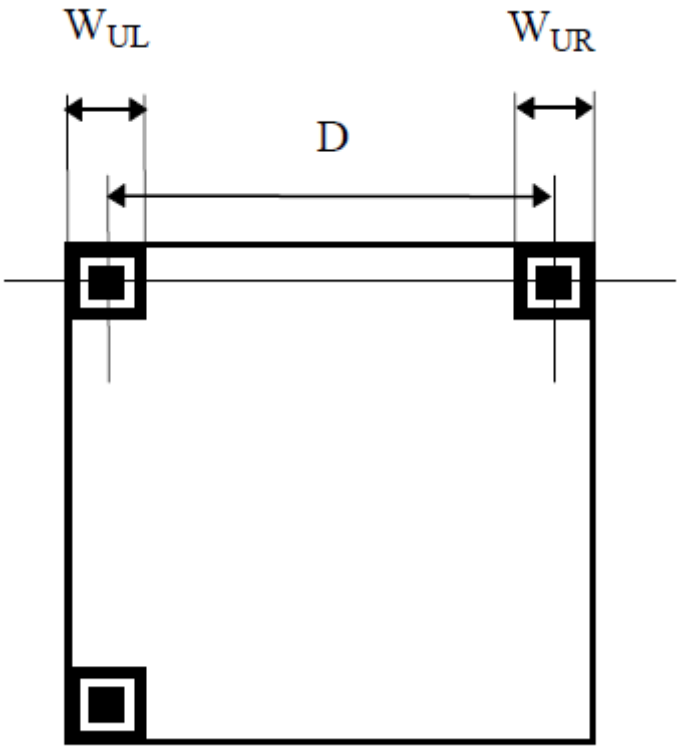
demo image 1	demo image2
	

大略流程

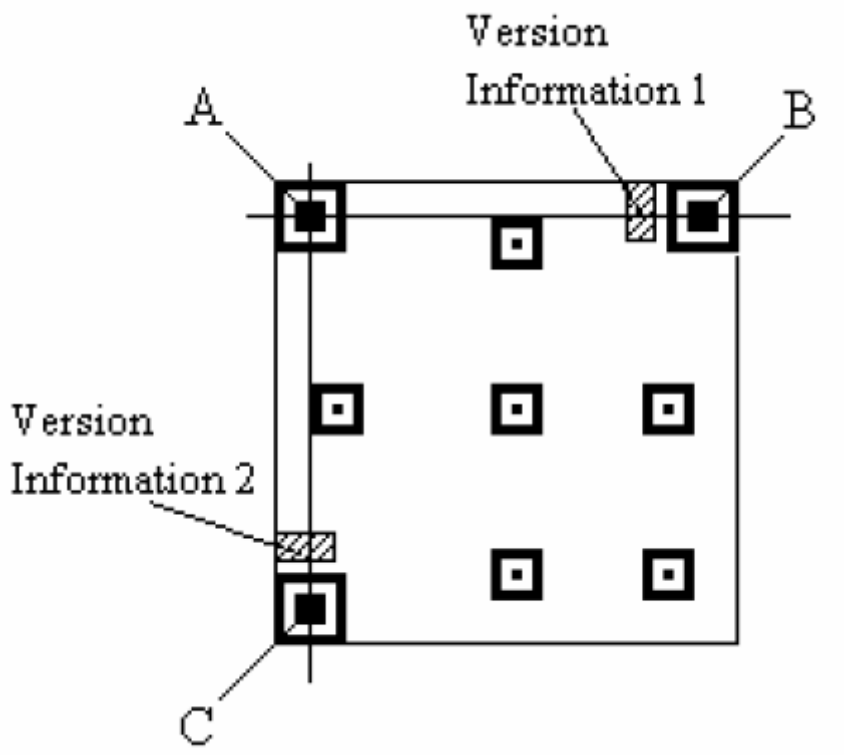


詳細流程

1. 決定 global threshold。(規格書使用 (最亮-最暗)/2)
2. 定位 finder pattern
    1. 逐線掃描(沿 row 與 column)，尋找 1:1:3:1:1 的圖案
    2. 找出 3 個 finder pattern，如果只找到 1 個，沿 micro QR 解碼流程
3. 根據 finder pattern 計算旋轉角度並校正
4. 計算 D, W_{UL}, W_{UR}

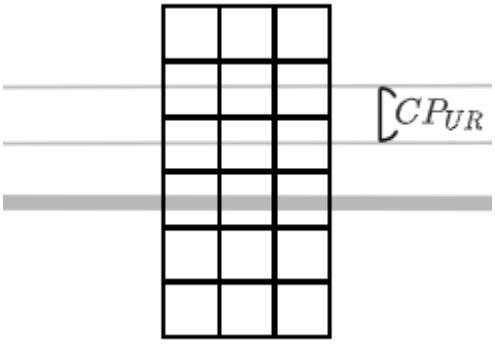


5. 計算估計黑/白小塊寬度的 $X = (W_{UL} + W_{UR})/14$
6. 計算(高)估計的版本號碼 $V = [D/X - 10]/4$
    - 版本號和大小的關係 $D = 17 + 4 \times V - 7 = 10 + 4 \times V$
7. 如果 $V \leq 6$，認定為沒有版本資訊區塊；$V \geq 7$，解碼版本資訊區塊
    1. 計算右上的黑/白小塊大小 $CP_{UR} = W_{UR}/7$
    2. 找基準線 $\overline{AB}, \overline{AC}$



    3. 用基準線 $\overline{AB}$ 和 CP_{UR} 生成取樣格 (sample grid)
        - 和 $\overline{AB}$ 垂直生成垂直線

- 和 $\overline{AB}$ 平行生成平行線
- 每條垂直線、水平線相距 CP_{UR}



4. 從取樣格中心點得到資料，並做錯誤糾正。如果錯誤超過容量上限，則取左下的版本資訊區塊再次解碼。

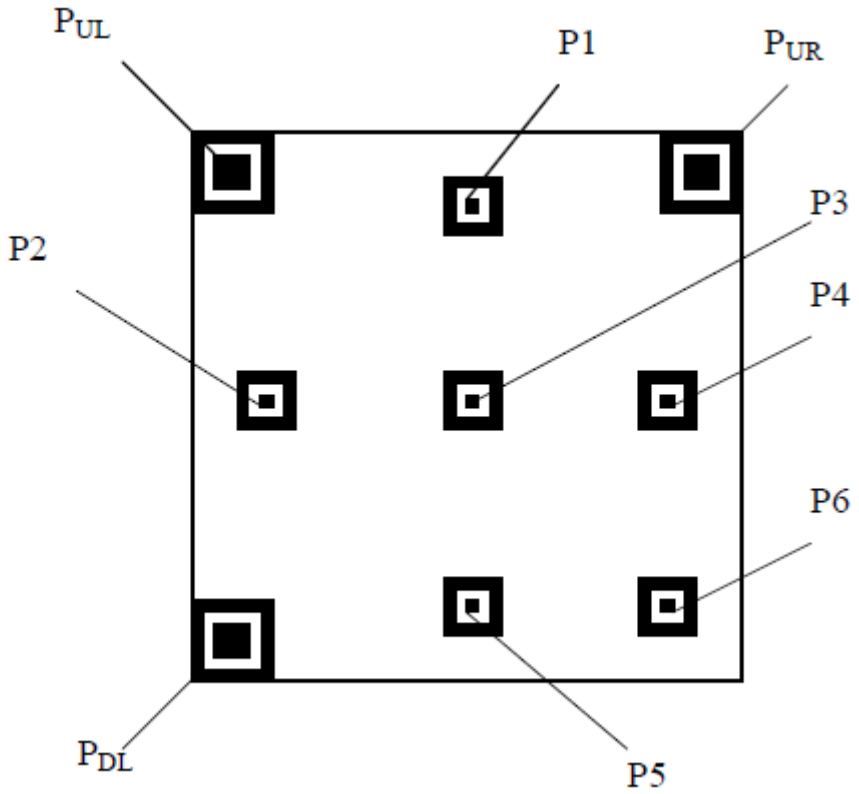
8.

- 沒有 alignment pattern 的 QR code (version 1)
  - 找到 timing pattern，並用他來生成整個 QR code 的取樣格

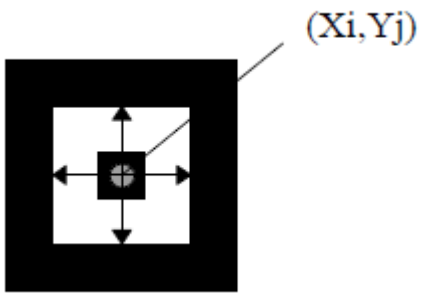
- 有 alignment pattern 的 QR code
  - 找到 timing pattern，生成上(左)方 6 格的格線，及下(右)方的格線

8.1. 計算 $CP_{UL} = W_{UL}/7$

8.2. 根據 find pattern 的位置及規格書標準，(使用 finder pattern 中心和 CP_{UL}) 計算最外圈的 alignment pattern $P1, P2$ 的估計位置



8.3. 在估計位置附近透過掃描白色區塊的邊界，尋找 alignment pattern 中心



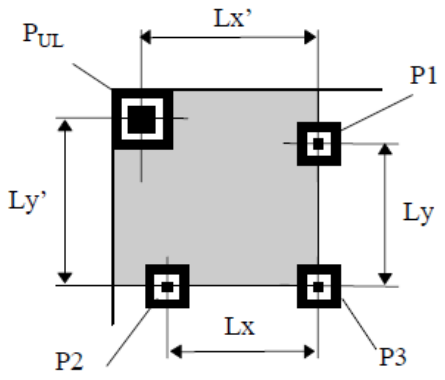
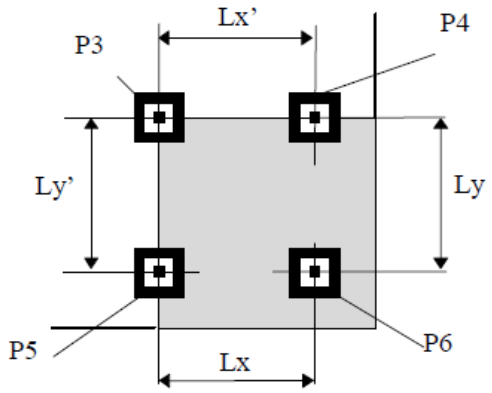
8.4. 與 (2) 相同的方法計算 $P3$ 的估計位置

8.5. 與 (3) 相同的方法找出 $P3$ 的中心

8.6. 計算 $L_x =$ ($P3$ 與 $P2$ 中心的距離)
計算 $L_y =$ ($P3$ 與 $P1$ 中心的距離)

根據規格書的標準，找出 $AP =$ (兩個 alignment pattern 中心的距離)
計算 $CP_x = L_x / AP$
計算 $CP_y = L_y / AP$

計算 $CP'_x = L'_x /$ (alignment pattern 中心到 finder pattern 中心的距離)
計算 $CP'_y = L'_y /$ (alignment pattern 中心到 finder pattern 中心的距離)

左上 (Step6)	右下 (Step8)
	

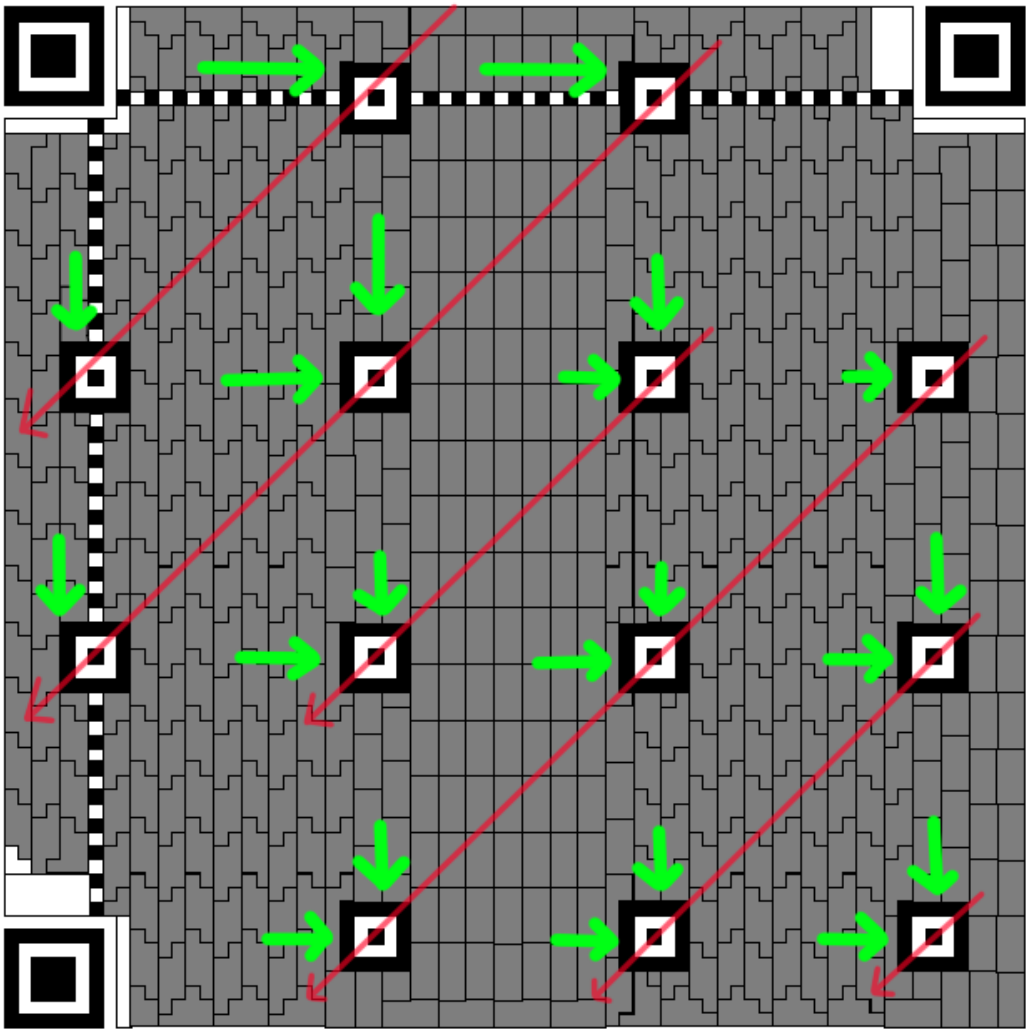
8.7. 並依 CP_x, CP_y, CP'_x, CP'_y 生成取樣格

- 如果區塊在 QR code 上緣，取樣格要包含上面
- 如果區塊在 QR code 下緣，取樣格要包含下面
- 如果區塊在 QR code 左緣，取樣格要包含左邊
- 如果區塊在 QR code 右緣，取樣格要包含右邊

8.8. 使用左上、上方、左方的 alignment pattern 來找出下一個 alignment pattern。

1. 使用 (2) 估計位置
2. 使用 (3) 計算實際位置

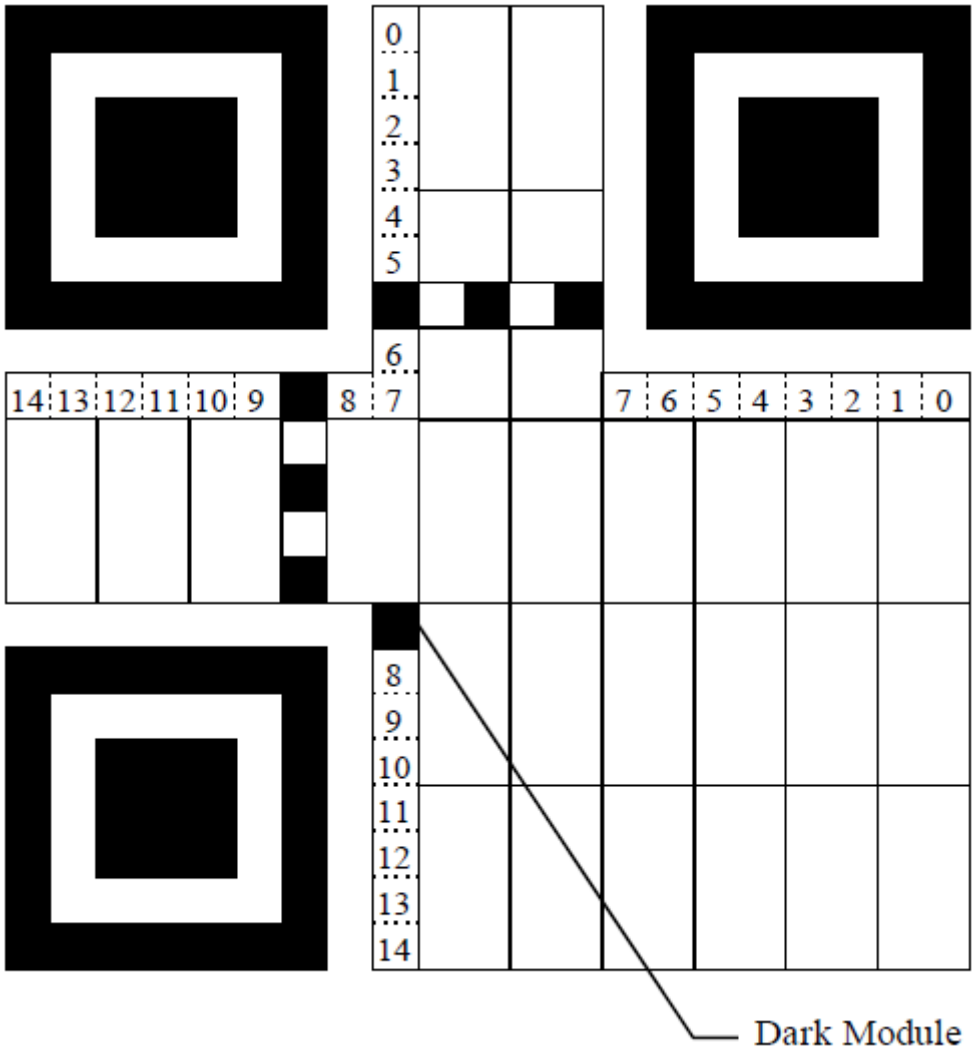
3. 使用 (6) 來做出取樣格



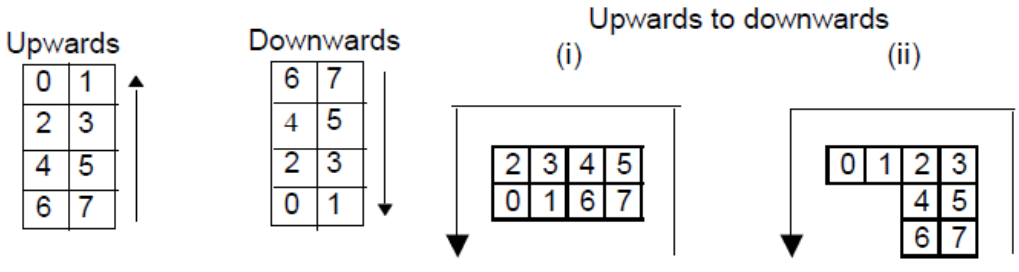
Version 14

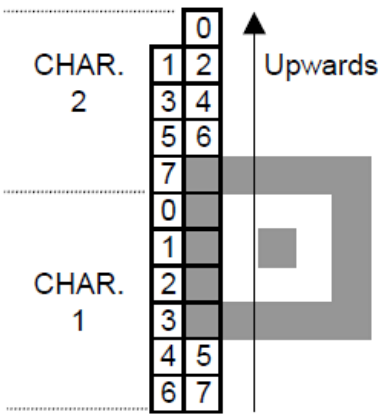
9. 對每一個格線交叉點取樣 3×3 pixels ，並根據 global threshold 決定是黑或白。
10. 解碼左上的 format 區塊，得到 XOR 遮罩。如果錯誤超過容量上限，再解碼左下、右上的 format 區塊。
11. 如果兩塊 format 區塊都無法正確解碼，把 format 區塊的 bit string 順序反向後再解碼看看。

如果解碼成功，將 (9) 獲得的取樣資料行列 (column,row) 互換再繼續解碼。

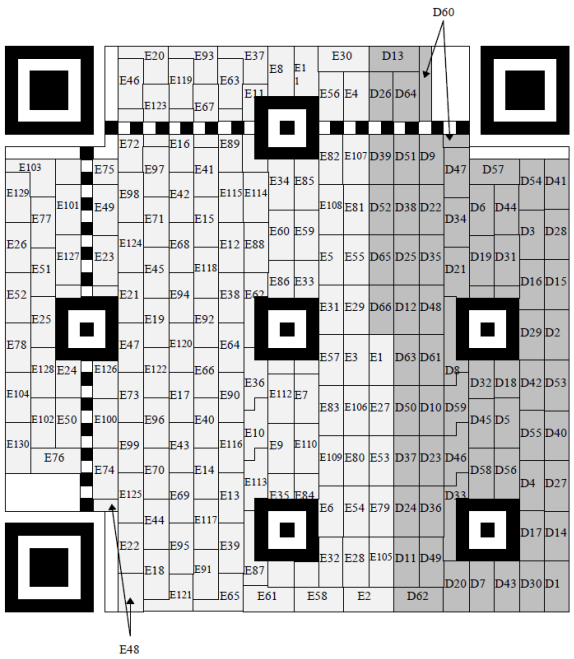


12. 根據 format 資料對對取樣資料做 XOR 遮罩
13. 根據規格書上 6.7.3 的擺放順序還原出每個 codeword





14. 依(版本號碼、容錯率) Table.9 及 規格書 6.6 方式 把 codeword 按組重新排列。

QR code	codeword 分組
	D1 - D13 Data Block 1 D14 - D26 Data Block 2 D27 - D39 Data Block 3 D40 - D52 Data Block 4 D53 - D66 Data Block 5 E1 - E26 EC Block 1 E27 - E52 EC Block 2 E53 - E78 EC Block 3 E79 - E104 EC Block 4 E105 - E130 EC Block 5

15. 對重排後的 codeword 做錯誤糾正。
16. 對錯誤糾正後的 codeword 解碼。
17. 根據 mode indicator (資料類型) 、character count indicators (資料字數) 來把 codeword 分段。
    1. 將資料分割成多個的 QR code，mode indicator 會多標註資料編號。
18. 根據資料類型來解碼，並根據資料編號串起資料。

Part 1 (定位圖案)

1. 決定 global threshold。(規格書使用 (最亮-最暗)/2)

將圖片二值化，規格書中是使用圖片中的(最亮值 - 最暗值)/2 做為 global threshold ，不過使用其他方法(Ostu, local threshold...)也可以。

原圖	二值化
	

註：使用比較複雜的二值化、去雜訊方法可以讓 QR code 偵測程式適應更多的環境，不過要求使用者正常地拍攝 QR code 圖案也是一種方法。

```
%-- f01_binarize.m
%-- read image
img = imread('image1.jpg');
img = rgb2gray(img);
img = im2double(img);

%-- compute global threshold
img_max = max(img(:));
img_min = min(img(:));
th = (img_max - img_min)/2;

%-- imbinarize
b_img = imbinarize(img,th);
imwrite(b_img,'01_binarize.png');
```

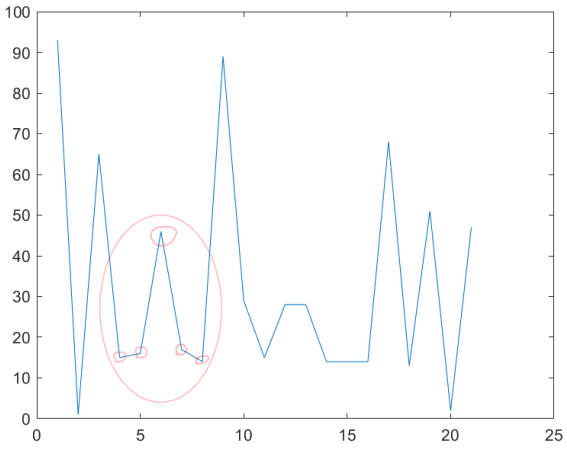
2. 定位 finder pattern

2.1. 逐線掃描(沿 row 與 column)，尋找 1:1:3:1:1 的圖案

掃描每一個 row 與 column，並計算**連續出現**的黑/白塊數量，看看有沒有 1:1:3:1:1 的區塊。 (容忍誤差 0.5)

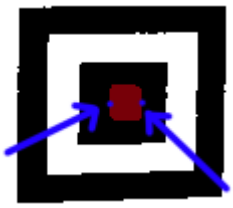
下圖為掃描其中一個 row，和對應的**連續出現數量**，y 軸是出現次數，x 軸則是第幾次黑白交替。

圖中畫圈的部分就要找的 1:1:3:1:1 區塊。(實際數值為 $15, 16, 46, 17, 14$)

其中一條線	對應的
	

演算法：

1. 給定一個 row/column，計算黑/白轉折點的位置。
2. 根據轉折點的位置，計算每一段的長度；並計算比例是否符合。
3. 將**符合部分的中心**記錄下來。(或紀錄外圍，事後再計算中心。)



4. column 找到的 mask 與 row 找到的 mask 做 AND 區塊的中心點即為 finder pattern 中心。

row 上找符合比例的區塊中心 (灰色區塊)	column 上找符合比例的區塊中心 (紅色區塊)	AND
		

註：如果出現不正確的小區塊(大小明顯不對)，可以嘗試用 `imopen` 來處理掉小區塊。

程式碼：

1. 給定一個 row/column，計算黑/白轉折點的位置。

```
function pt = find_turning_point(line)

pt_cc = 1;
prev = line(1);
L = length(line);

for i=2:L
    %-- if black/white change
    if line(i) ~= prev
        pt( pt_cc ) = i;
        pt_cc = pt_cc + 1;
        prev = line(i);
    end
end

if prev == line(L)
    pt( pt_cc ) = L;
end

end
```

2. 根據轉折點的位置，計算每一段的長度；並計算比例是否符合。

```
function pos = finder_position( turning_pt )
pos = [];

pt_dis = turning_pt(2:end) - turning_pt(1:end-1);

L = size( pt_dis , 2 );
tol = 0.5;
base = [ 1 , 1 , 3 , 1 , 1 ];
upper_bound = base+tol;
lower_bound = base-tol;

for i=1:L-4
    pattern = pt_dis( i:i+4 );
    pattern = pattern./pattern(1);
    upper = sum(pattern<upper_bound);

    if upper == 5
        lower = sum(pattern>lower_bound);
        if lower == 5
            pos = [ pos ; turning_pt(i) , turning_pt(i+5) ];
        end
    end
end
```

3. 將**符合部分的中心**記錄下來。(或紀錄外圍，事後再計算中心。)

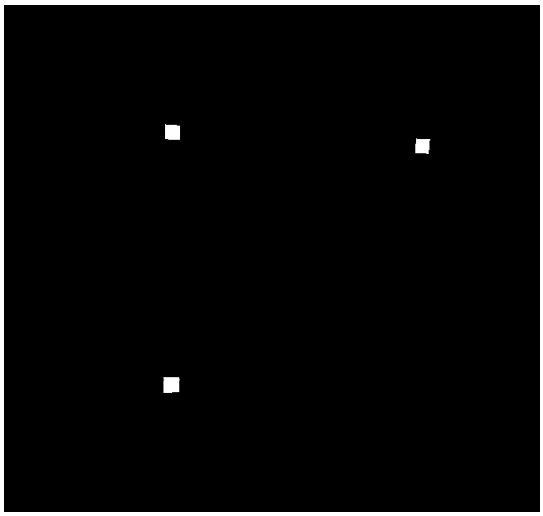
```
function mask = get_mask( img )
[h,w] = size(img);
mask = logical(zeros(h,w));

for y=1:h
    turning_pt = find_turning_point( img(y,:) );
    pos = finder_position(turning_pt);
```

```
    [r,c] = size(pos);
    if r>0
        for j=1:r
            s_pt = pos(j,1);
            e_pt = pos(j,2);
            dis = floor((e_pt - s_pt)/7);
            s_pt = s_pt + 3*dis;
            e_pt = e_pt - 3*dis;
            mask(y,s_pt:e_pt) = 1;
        end
    end
end
```

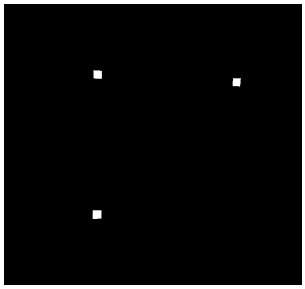
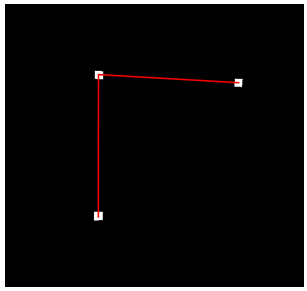
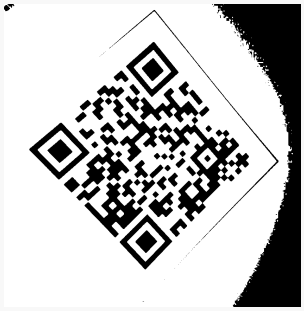
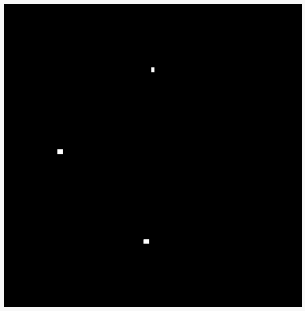
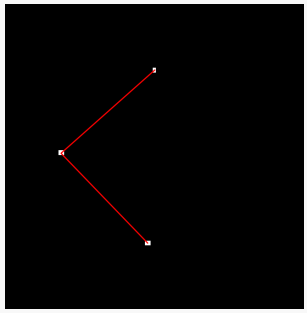
4. column 找到的 mask 與 row 找到的 mask 做 AND 區塊的中心點即為 finder pattern 中心。

```
%-- f02_detect.m
%-- read image
img = imread('01_binarize.png');
img = im2double(img);

%-- get mask
[h,w] = size(img);
row_mask = get_mask(img);
col_mask = (get_mask(img'))';
mask = row_mask & col_mask;
imwrite( mask , '02_finder.png' )

%-- for display
g_img = zeros( h , w , 3 );
img( mask ) = img( mask )*0.5;
R = img;
R( mask ) = R( mask ) + 0.5;

g_img(:,:,1) = R;
g_img(:,:,2) = img;
g_img(:,:,3) = img;

imshow(g_img)
```

2.2. 找出 3 個 finder pattern，如果只找到 1 個，沿 micro QR 解碼流程

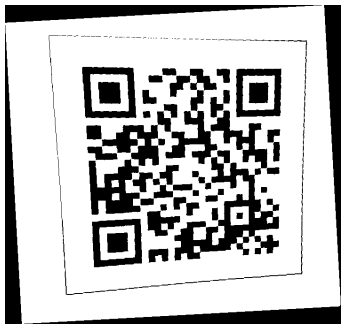
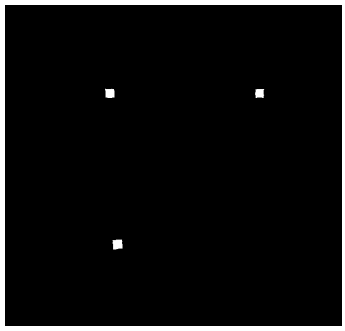
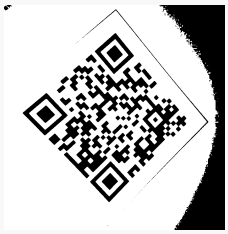
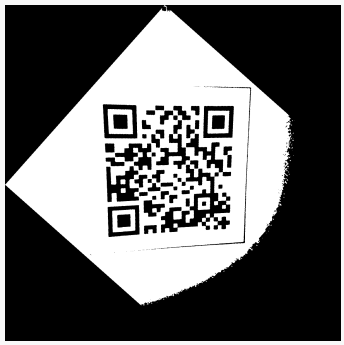
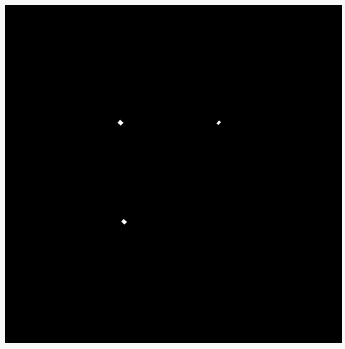
二值化	得到 finder 中心
	

3. 根據 finder pattern 計算旋轉角度並校正

	二值化結果	求得的 mask	角度示意圖
case1			
case2			

演算法：

1. 找到 mask 區塊的中心點 (使用 `bwlabel` 來得到連通區塊，)
2. 計算角度，找到直角
3. 找到整個直角三角形應該轉幾度

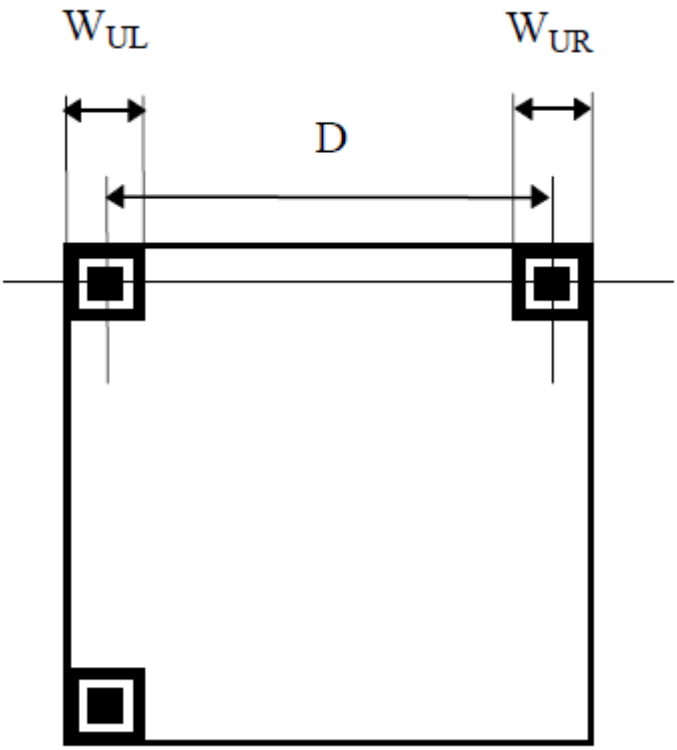
	二值化結果	旋轉結果	旋轉 finder
case1			
case2			

```m
%-- f03_region.m
img = imread('01_binarize.png');
mask = imread('02_finder.png');

%---- algorithm part1
%-- label each connected region
mask_idx = bwlabel( mask );

%-- region amount
region_amount = max(mask_idx(:));
if region_amount ~= 3
    disp('region amount should be 3');
end

%-- find region center
pt = [];
for i=1:3
    [y,x] = find(mask_idx == i);
    mx = mean(x);
    my = mean(y);
    pt = [ pt ; mx , my ];
end

%---- algorithm part2
%-- find each angle
v1 = pt(2,:) - pt(3,:);
v2 = pt(1,:) - pt(3,:);
v3 = pt(1,:) - pt(2,:);

v1 = v1 ./ norm(v1);
v2 = v2 ./ norm(v2);
v3 = v3 ./ norm(v3);

```

```
a1 = acos(dot(v2,v3));
a2 = acos(dot(v1,v3));
a3 = acos(dot(v1,v2));

%-- find right angle (70~110 degree)
idx = 1;
if abs(a1-pi/2) < pi/9
    u = pt(2,:) - pt(1,:);
    v = pt(3,:) - pt(1,:);
    idx = 1;
elseif abs(a2-pi/2) < pi/9
    u = pt(1,:) - pt(2,:);
    v = pt(3,:) - pt(2,:);
    idx = 2;
else
    u = pt(1,:) - pt(3,:);
    v = pt(2,:) - pt(3,:);
    idx = 3;
end

%---- algorithm part3
%-- the image's y-axis is up-to-down
d1 = atan2( -u(2),u(1) );
d2 = atan2( -v(2),v(1) );

%-- compute the angle from v to u (clockwise)
inc = mod(d2-d1+2*pi,2*pi);
if inc > pi/2
    %-- d2 before d1 (clockwise)
    rotate_theta = -d1;
else
    %-- d2 behind d1 (clockwise)
    rotate_theta = -d2;
end

%-- rotate image
img = imrotate( img , rotate_theta*180/pi );
mask = imrotate( mask , rotate_theta*180/pi );

imwrite( img , '03_binarize.png' );
imwrite( mask , '03_finder.png' );
```

Part 2 (版本資訊)

4. 計算 D, W_{UL}, W_{UR}



演算法：

1. 由上一步的 finder 找到 finder 中心。
2. 根據 finder 中心找到 D。
3. 根據 finder 中心的 row 計算 W_{UL}, W_{UR}

程式碼：

```
function [D,W_UL,W_UR] = f04_cal_length(img,mask)

%-- label each connected region
mask_idx = bwlabel( mask );

%-- find region center
pt = [];
for i=1:3
    [y,x] = find(mask_idx == i);
    mx = mean(x);
    my = mean(y);
    pt = [ pt ; mx , my ];
end

%-- find UL & UR index
max_y = max(pt(:,2));
max_x = max(pt(:,1));

UR_idx = find( pt(:,1) == max_x );
```

```
DL_idx = find( pt(:,2) == max_y );
UL_idx = 1+2+3 - DL_idx - UR_idx;

UR = pt(UR_idx,:);
UL = pt(UL_idx,:);

%-- compute D
D = UR(1) - UL(1);

%-- find W_UL & W_UR
target_y = floor(UR(2));
turning_pt = find_turning_point( img(target_y,:) );
pos = finder_position(turning_pt);

W_UL = pos(1,2) - pos(1,1);
W_UR = pos(2,2) - pos(2,1);
end
```

5. 計算估計黑/白小塊寬度的 $X = (W_{UL} + W_{UR})/14$

```
X = (W_UL+W_UR)/14;
```

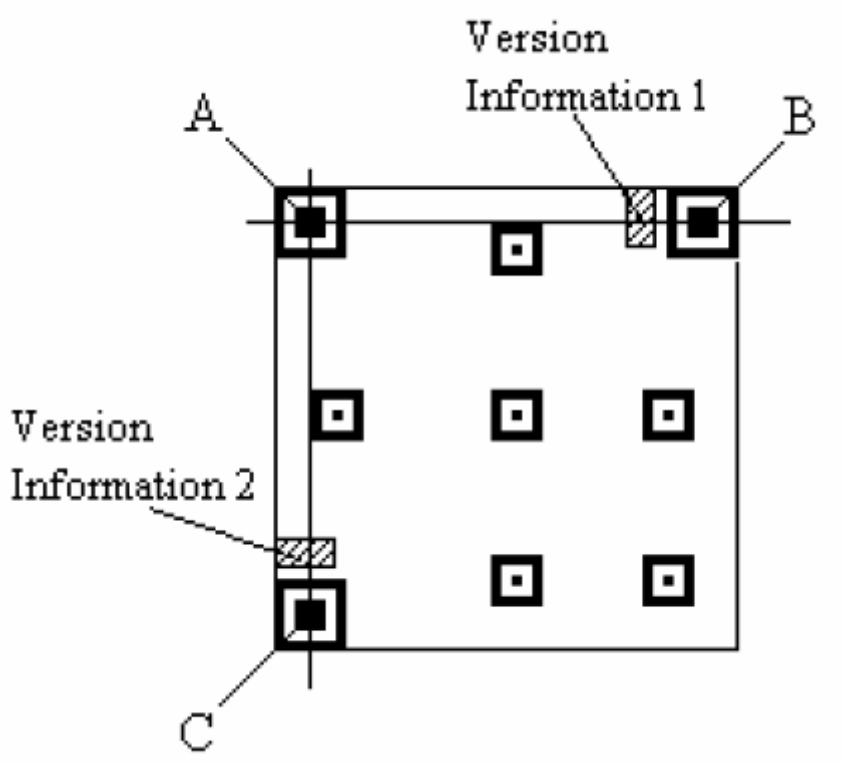
6. 計算估計的版本號碼 $V = [D/X - 10]/4$

- 版本號和大小的關係 $D = 17 + 4 \times V - 7 = 10 + 4 \times V$

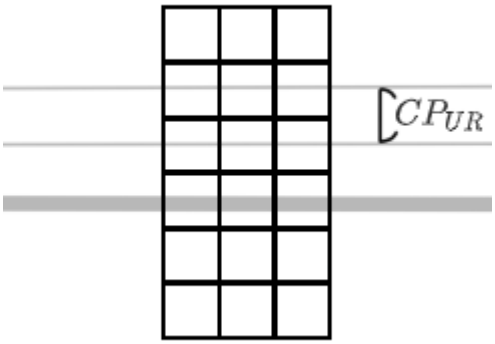
```
V = (D/X-10)/4;
```

1. 如果 $V \leq 6$，認定為沒有版本資訊區塊；$V \geq 7$，解碼版本資訊區塊
    1. 計算右上的黑/白小塊大小 $CP_{UR} = W_{UR}/7$

2. 找基準線 $\overline{AB}, \overline{AC}$



3. 用基準線 $\overline{AB}$ 和 CP_{UR} 生成取樣格 (sample grid)
    - 和 $\overline{AB}$ 垂直生成垂直線
    - 和 $\overline{AB}$ 平行生成平行線
    - 每條垂直線、水平線相距 CP_{UR}



4. 從取樣格中心點得到資料，並做錯誤糾正。如果錯誤超過容量上限，則取左下的版本資訊區塊再次解碼。

Part 3 (讀取資料)

8. 讀取資料

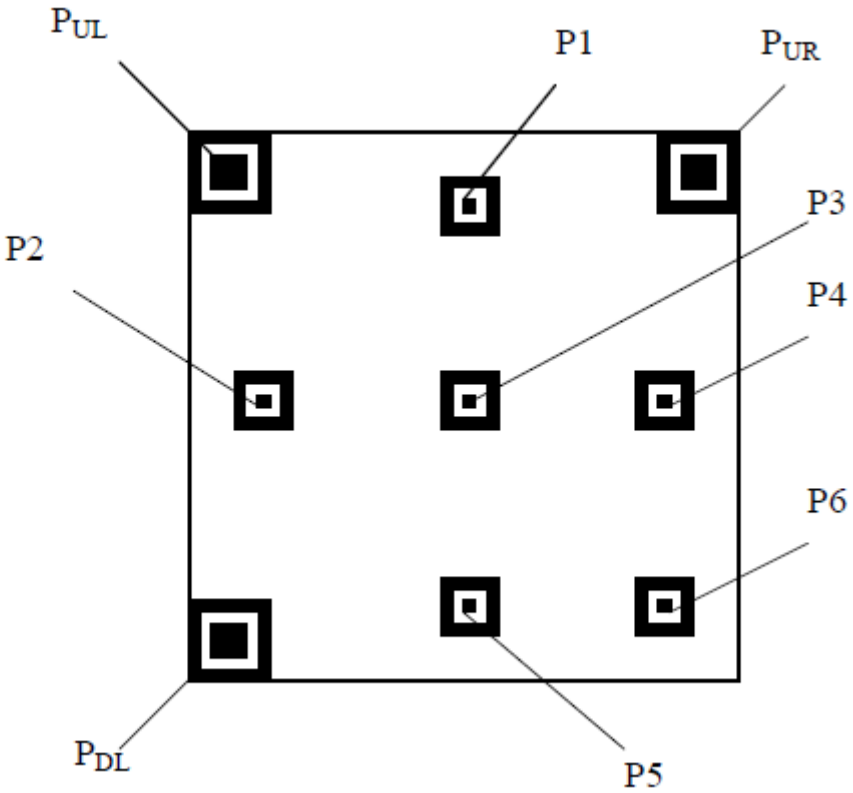
- 沒有 alignment pattern 的 QR code (version 1)
  - 找到 timing pattern，並用他來生成整個 QR code 的取樣格
- 有 alignment pattern 的 QR code
  - 找到 timing pattern，生成上(左)方 6 格的格線，及下(右)方的格線

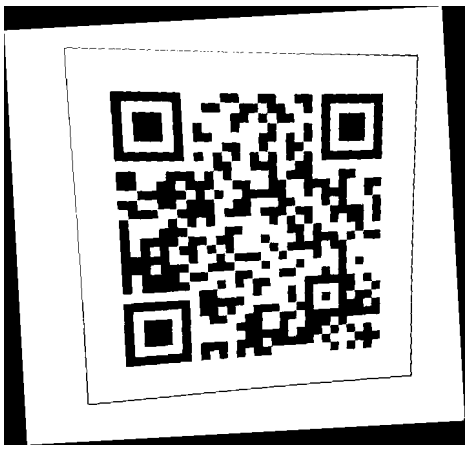
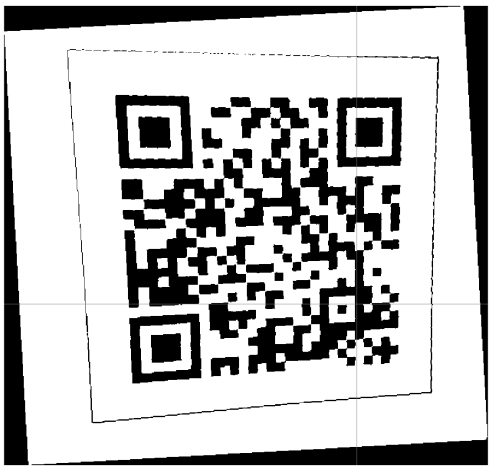
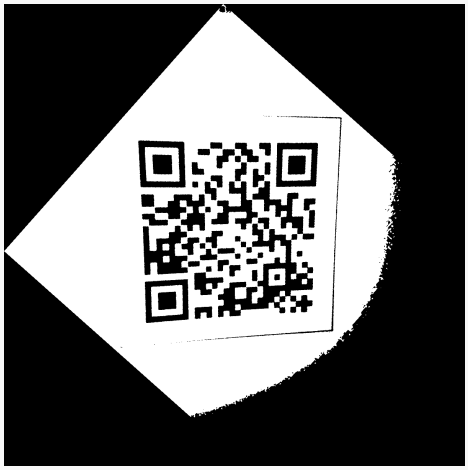
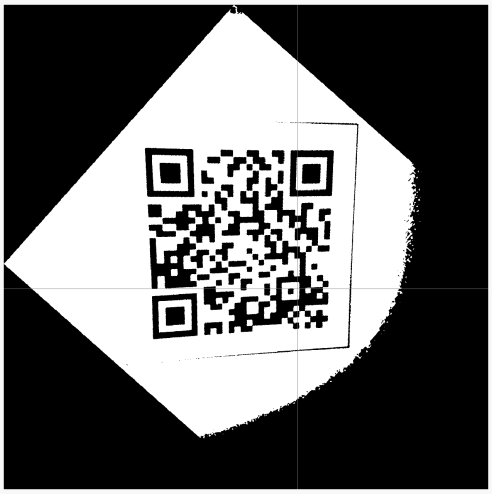
8.1. 計算 $CP_{UL} = W_{UL}/7$

```
CP_UL = W_UL/7;
```

8.2. 計算最外圈的 alignment pattern $P1, P2$ 的估計位置

根據 find pattern 的位置及規格書標準，使用 finder pattern 中心和 CP_{UL} 來計算。



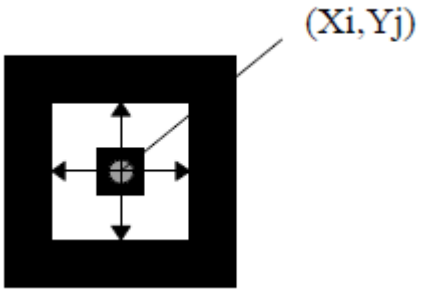
	二值化結果	估計位置 (交錯點)
case1		
case2		

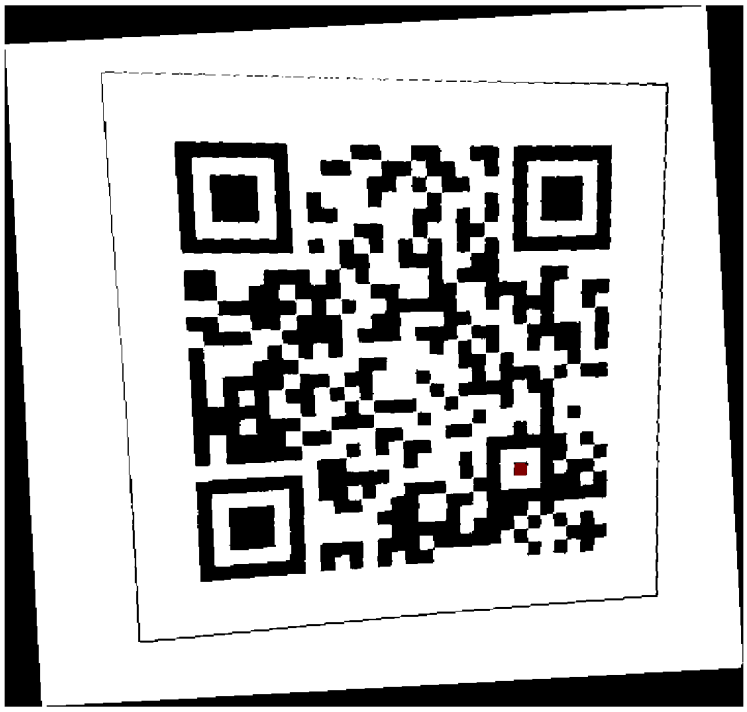
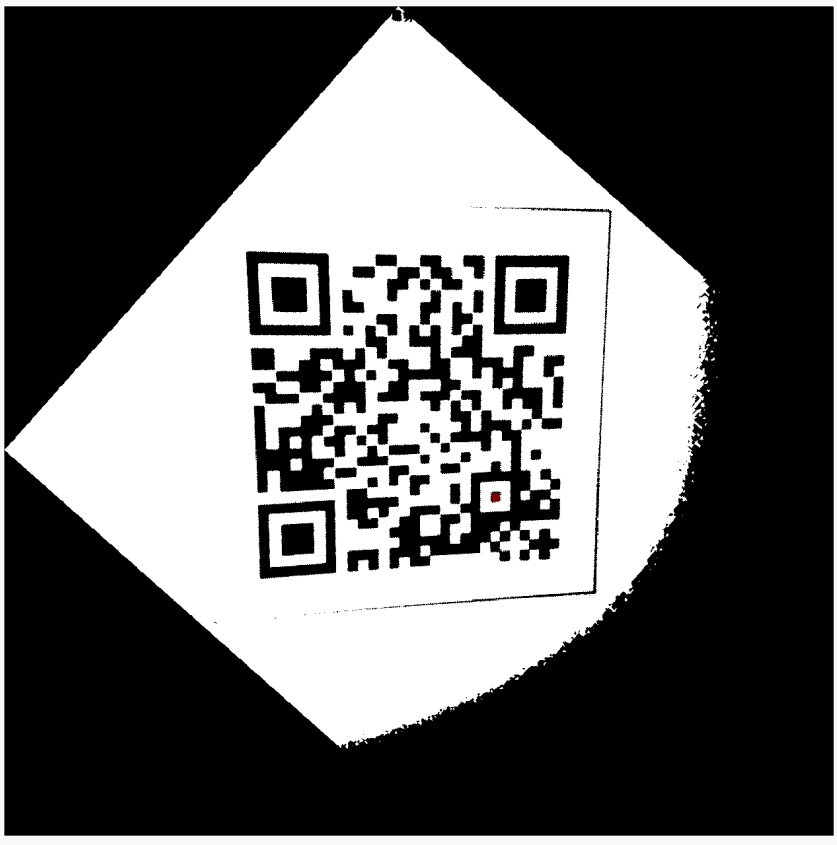
程式碼：(需要再修改才能適用給各種版本的 QR code)

```
%-- step 8.2
est_align_pos = version_position(V);
align_amount = size(est_align_pos,1);

mask = imread('03_finder.png');
[P_UL,P_UR,P_DL] = CP(mask);

% estiate the outter alignment pattern (V3)
P1 = est_align_pos(1,:);
P1 = P1 - 3;
P1(1) = P1(1)*CP_UL + P_DL(1);
P1(2) = P1(2)*CP_UR + P_UR(2);
P1 = floor(P1);
```

8.3. 在估計位置附近透過掃描白色區塊的邊界，尋找 alignment pattern 中心

與尋找 finder pattern 相同作法，找出 1:1:1 的圖案。



	尋找後的結果 (注意 alignment pattern 中心)
case1	
case2	

```
%-- step 8.3
row_mask = get_align_mask(img,P1(2),2*floor(X));
col_mask = (get_align_mask(img',P1(1),2*floor(X)))';
mask = row_mask & col_mask;
%-- remove noise
mask = imopen( mask , strel('rect',[5,5]) );

%-- for display
%[h,w] = size(img);
%g_img = zeros( h , w , 3 );
%img( mask ) = img( mask )*0.5;
%R = img;
%R( mask ) = R( mask ) + 0.5;

%g_img(:,:,1) = R;
%g_img(:,:,2) = img;
%g_img(:,:,3) = img;
%figure()
%imshow(g_img)
```

8.4. 與 8.2 相同的方法計算 $P3$ 的估計位置

8.5. 與 8.3 相同的方法找出 $P3$ 的中心

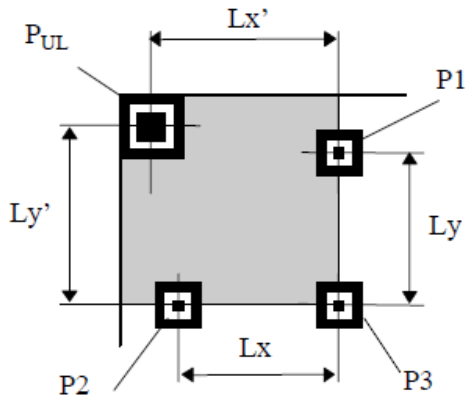
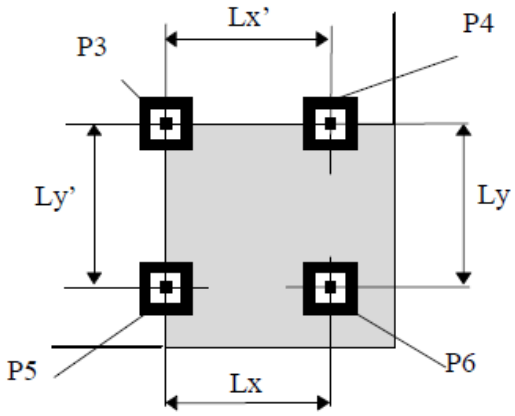
8.6. 計算區塊的相關參數

計算 $L_x =$ ($P3$ 與 $P2$ 中心的距離)
計算 $L_y =$ ($P3$ 與 $P1$ 中心的距離)

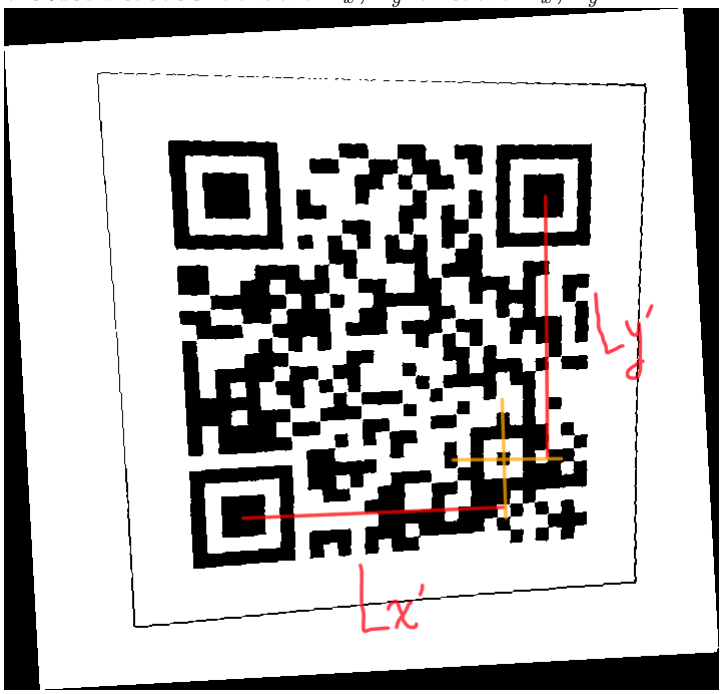
根據規格書的標準，找出 $AP =$ (兩個 alignment pattern 中心的距離)
計算 $CP_x = L_x / AP$
計算 $CP_y = L_y / AP$

當在周圍有 finder pattern 時，計算：
計算 $CP'_x = L'_x /$ (alignment pattern 中心到 finder pattern 中心的距離)
計算 $CP'_y = L'_y /$ (alignment pattern 中心到 finder pattern 中心的距離)

註：把 $CP_x, CP_y, \cdots$ 當作向量 (x, y) 來算會比單純算出一個值要好。

左上 (Step6)	右下 (Step8)
	

在我們使用的例子中要計算 L'_x, L'_y 不用計算 L_x, L_y



程式碼：

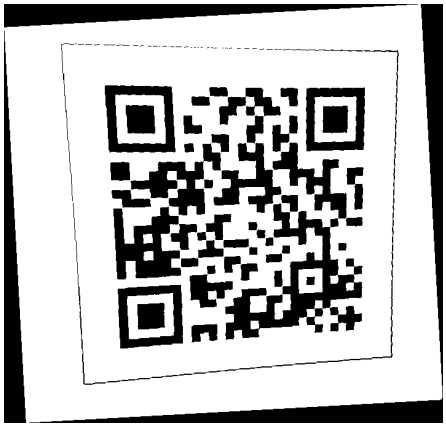
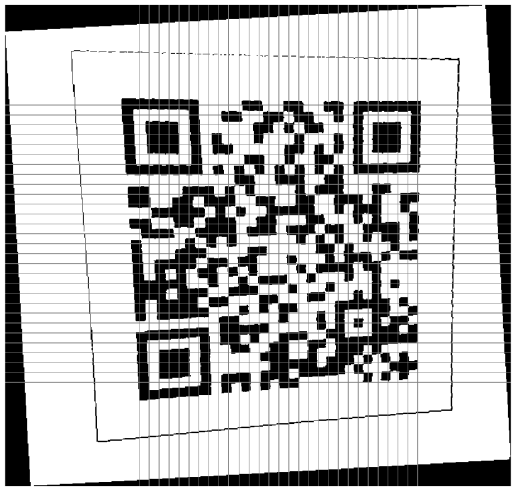
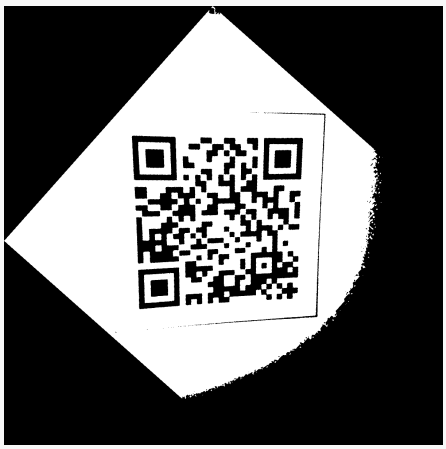
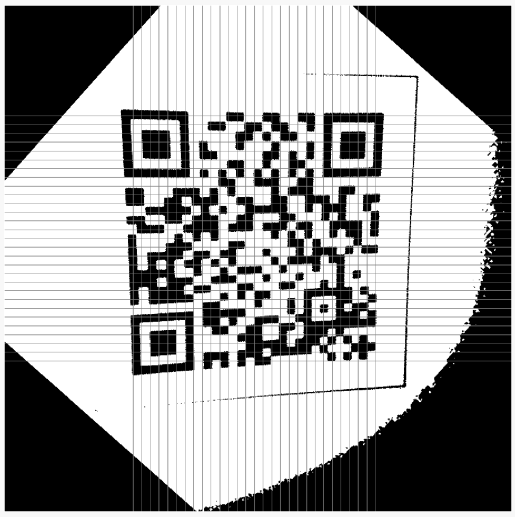
```
%-- step 8.6 (V3)
mask_idx = bwlabel(mask);
[y,x] = find(mask_idx == 1);
P1 = [ mean(x) , mean(y) ];

Lx1 = norm(P_DL-P1);
Ly1 = norm(P_UR-P1);

CPx1 = Lx1/(est_align_pos(1,1)-3);
CPy1 = Ly1/(est_align_pos(1,2)-3);
```

8.7. 並依 CP_x, CP_y, CP'_x, CP'_y 生成取樣格

- 如果區塊在 QR code 上緣，取樣格要包含上面
- 如果區塊在 QR code 下緣，取樣格要包含下面
- 如果區塊在 QR code 左緣，取樣格要包含左邊

- 如果區塊在 QR code 右緣，取樣格要包含右邊

在我們的例子(V3)中，取樣格剛好要包含整個 QR code。

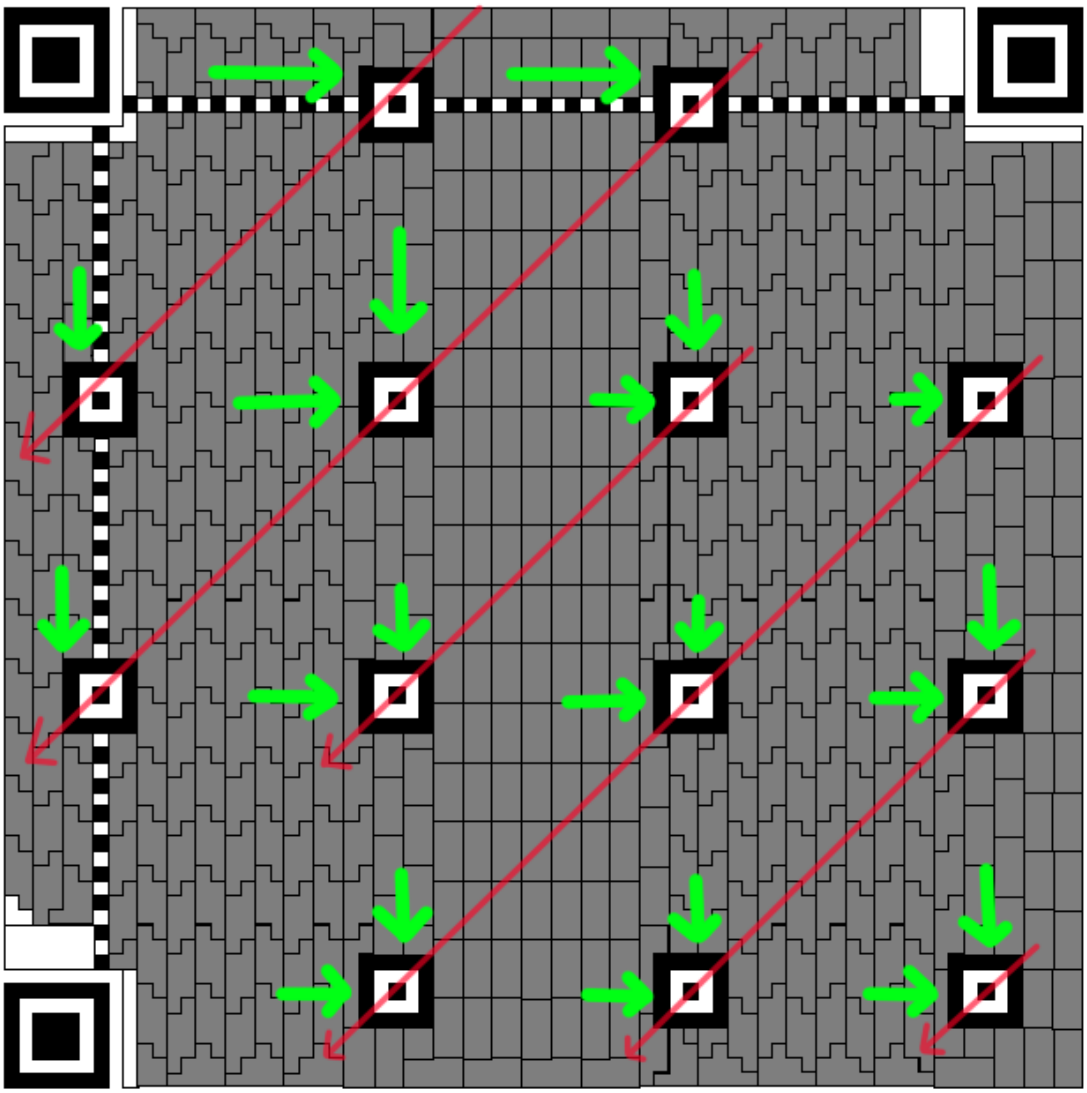
	二值化圖	取樣格 (有可能因為解析度關係會看不到 1px 的線，線是等距的)
case1		
case2		

程式碼：

```
%-- step 8.7 (V3)
V = round(V);
W = 17 + 4*V;
X = CPx1*( (1:W) - 23 ) + P1(1);
Y = CPy1*( (1:W) - 23 ) + P1(2);

X = floor(X);
Y = floor(Y);

%-- for display
img( Y , : ) = 0.5;
img( : , X ) = 0.5;
imshow(img);
```

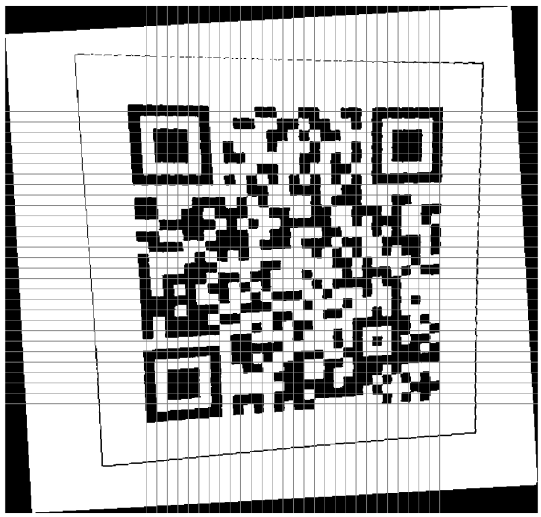
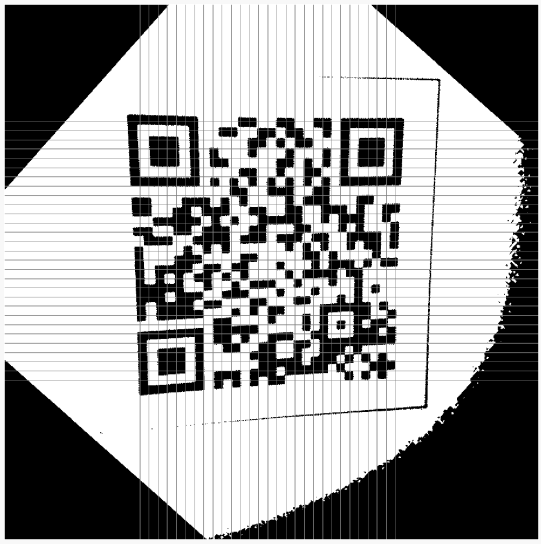
8.8. 使用左上、正上、正左的 alignment pattern 來找出下一個 alignment pattern

1. 使用 8.2 的方法估計位置
2. 使用 8.3 的方法計算實際位置
3. 使用 8.6 的方法來做出取樣格



Version 14

9. 對每一個格線交叉點取樣 3×3 pixels 來決定是黑或白。

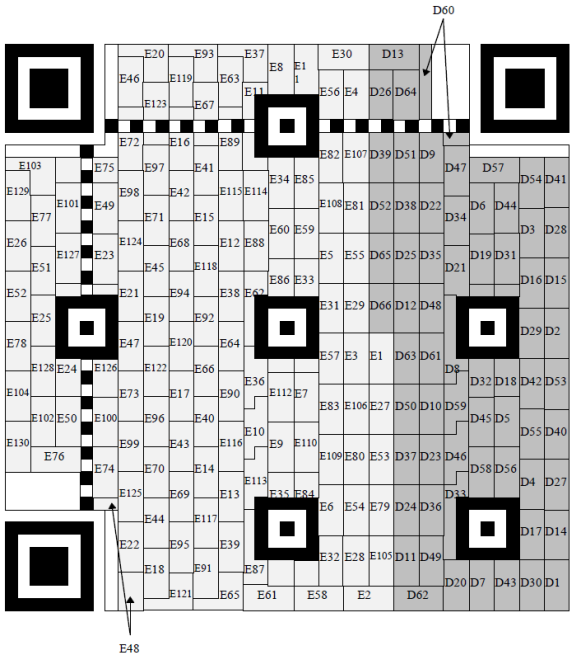
	取樣格	以 3×3 (多數決)來決定是黑/白
case1		
case2		

```
%-- step 9 (V3)
result = ones(29,29);
ycc = 1;
for y=Y
    xcc = 1;
    for x=X
        sample = img( y-1:y+1 , x-1:x+1 );
        result(ycc,xcc) = mean(sample(:)) > 0.5;
        xcc = xcc + 1;
    end
    ycc = ycc + 1;
end
imshow(result)
```

Part 4 (解碼資料)

5. 解碼左上的 format 區塊，得到 XOR 遮罩。如果錯誤超過容量上限，再解碼左下、右上的 format 區塊。
6. 如果兩塊 format 區塊都無法正確解碼，把 format 區塊的 bit string 順序反向後再解碼看看。

   如果解碼成功，將 (9) 獲得的取樣資料行列 (column,row) 互換再繼續解碼。
    p09
7. 根據 format 資料對對取樣資料做 XOR 遮罩
8. 根據規格書上 6.7.3 的擺放順序還原出每個 codeword
    p10
9. 依(版本號碼、容錯率) Table.9 及 規格書 6.6 方式 把 codeword 按組重新排列。

QR code	codeword 分組
	D1 - D13 Data Block 1 D14 - D26 Data Block 2 D27 - D39 Data Block 3 D40 - D52 Data Block 4 D53 - D66 Data Block 5 E1 - E26 EC Block 1 E27 - E52 EC Block 2 E53 - E78 EC Block 3 E79 - E104 EC Block 4 E105 - E130 EC Block 5

10. 對重排後的 codeword 做錯誤糾正。
11. 對錯誤糾正後的 codeword 解碼。
12. 根據 mode indicator (資料類型) 、character count indicators (資料字數) 來把 codeword 分段。
    1. 將資料分割成多個的 QR code，mode indicator 會多標註資料編號。
13. 根據資料類型來解碼，並根據資料編號串起資料。

RS Codes

簡介

RS codes (Reed-solomon codes) 是一種錯誤糾正碼，透過額外傳遞的資料來修正錯誤。

RS Codes -- wiki

Finite Field

在 RS 編碼中，我們使用的不是一般的實數系統，而是在 Finite Field 上做加減乘除，Finite Field 的概念在代數的課程會提到，即有限個元素構成的 field (體)。

探究原因是使用 Finite Field 時，計算過程中的所有數值都可以用有限的數字來表示，而有限的數字則可以編成固定長度的編碼這一好處。

但缺點則是乘法與一般乘法不同，為求速度，通常需要先建立乘法表。

參考：Residue#Finite Field -- 演算法筆記

簡單的例子

這裡有幾個簡單的例子。

- $F_2 \cong \mathbb{Z}_2$

```
+ | 0 1       · | 0 1
--+----       --+----
0 | 0 1       0 | 0 0
1 | 1 0       1 | 0 1
```

- $F_3 \cong \mathbb{Z}_3$

```
+ | 0 1 2       · | 0 1 2
--+------       --+------
0 | 0 1 2       0 | 0 0 0
1 | 1 2 0       1 | 0 1 2
2 | 2 0 1       2 | 0 2 1
```

- $F_4 \cong V_4 \cong \mathbb{Z}_2 \times \mathbb{Z}_2$

```
+ | 0 1 A B       · | 0 1 A B
--+--------       --+--------
0 | 0 1 A B       0 | 0 0 0 0
1 | 1 0 B A       1 | 0 1 A B
A | A B 0 1       A | 0 A B 1
B | B A 1 0       B | 0 B 1 A
```

相關的定義

這裡利用 Primitive polynomial 來建立一個 $GF(p^m)$

首先有幾個定義：

- primitive element
  若一個元素是 $GF(p^m)$ 的乘法群的生成元，則稱為 primitive element。
  $\alpha \text{ is a primitive element } \implies \forall x \neq 0 \in GF(p^m), x = \alpha^i, \text{for some } i$
- Irreducible polynomial (不可拆多項式)
  假設一個多項式 $f(x)$ 除了 1 和本身外沒有其他因式，則 $f(x)$ 稱為 irreducible polynomial。
- Primitive polynomial (本質多項式)
  $f(x) \in GF(p)[X]$，滿足

  1. $\deg f = m$
  2. $f(x)$ 不可拆
  3. $f(\alpha) = 0$

  則 $f(x)$ 是 $GF(p^m)$ 中相對於 α 的 primitive polynomial。

建立 Finite Field

1. 再來我們就可以挑選一個 $GF(2^8)$ 的 primitive polynomial $p(x) = 1 + x^2 + x^3 + x^4 + x^8$

   註：選 2^8 的原因是因為 QR code 中是以 8 個黑/白塊為一個單位來編碼，理論上可以選 $2^{10}, 2^{12}, \cdots$ 等等其他數字。
2. 加法定為多項式上的加法 ($\mathbb{F}_2[X]$ 上的加法)
3. 乘法定為多項式上的乘法 (mod $p(x)$)

得到 $GF(2^8)$ 裡面每個元素

數字形式	位元形式	多項式形式	α^i 形式
0	0000 0000	0	0
1	0000 0001	1	α^0
2	0000 0010	x	α^1
4	0000 0100	x^2	α^2
8	0000 1000	x^3	α^3
16	0001 0000	x^4	α^4
32	0010 0000	x^5	α^5
64	0100 0000	x^6	α^6
128	1000 0000	x^7	α^7
29	0001 1101	$x^4+x^3+x^2+1$	α^8
58	0011 1010	$x^5+x^4+x^3+x$	α^9
⋮	⋮	⋮	⋮
142	1000 1110	$x^7+x^3+x^2+x$	α^{254}
(重複) 1	0000 0001	1	α^{255}

編碼

當你在使用 RS 編碼時，要注意的事情有兩個：

- 使用的字母表
- RS 編碼的格式

字母表

當在做編碼時，一個重要的點是要確定編碼使用的字母表，字母表是編碼中可以出現的符號集合。

例如說一個字母 a，使用不同字母表來表示(編碼)

符號	2 進位	10 進位	16 進位	Base64
a	0110 0001	97	61	YQ==

2 進位的字母表是 $\{0,1\}$

10 進位的字母表是 $\{0,1,2,3,\cdots,8,9\}$

16 進位的字母表是 $\{0,1,2,3,\cdots,8,9,A,B,C,D,E,F\}$

常見的編碼 Base64 的字母表則是 $\{A,B,\cdots,Y,Z,a,b,\cdots,y,z,0,1,2,\cdots,9,\cdots\}$ 共 64 個字元

碼字(codeword)、符號(symbol)

在編碼系統裡，常常會用碼字或是符號來代表每一個編碼的單位。

2 進制的 a 就由 8 個符號(symbol) 或稱碼字(codeword) 組成。

10 進制的 a 就由 2 個碼字組成。

16 進制的 a 就由 2 個碼字組成。

而在 QR code 中，8 個黑/白小塊一起當作是一個碼字，他的字母表是 $\{0, 1, \alpha, \alpha^2, \cdots, \alpha^{254}\} = GF(2^8)$，由 256 種碼字組成 (每種碼字應一種圖案)。

RS 編碼的格式 (Table 9)

在 RS 編碼中，通常會有兩個 (c, k) 或三個 (c, k, r) 組成一組的數字，例如說 (255,223) 或者 (255,223,16)，這代表他總共會編碼出 255 個碼字 (codeword) ，其中 223 個碼字是資料碼、32 (16*2) 個碼字是校正碼。

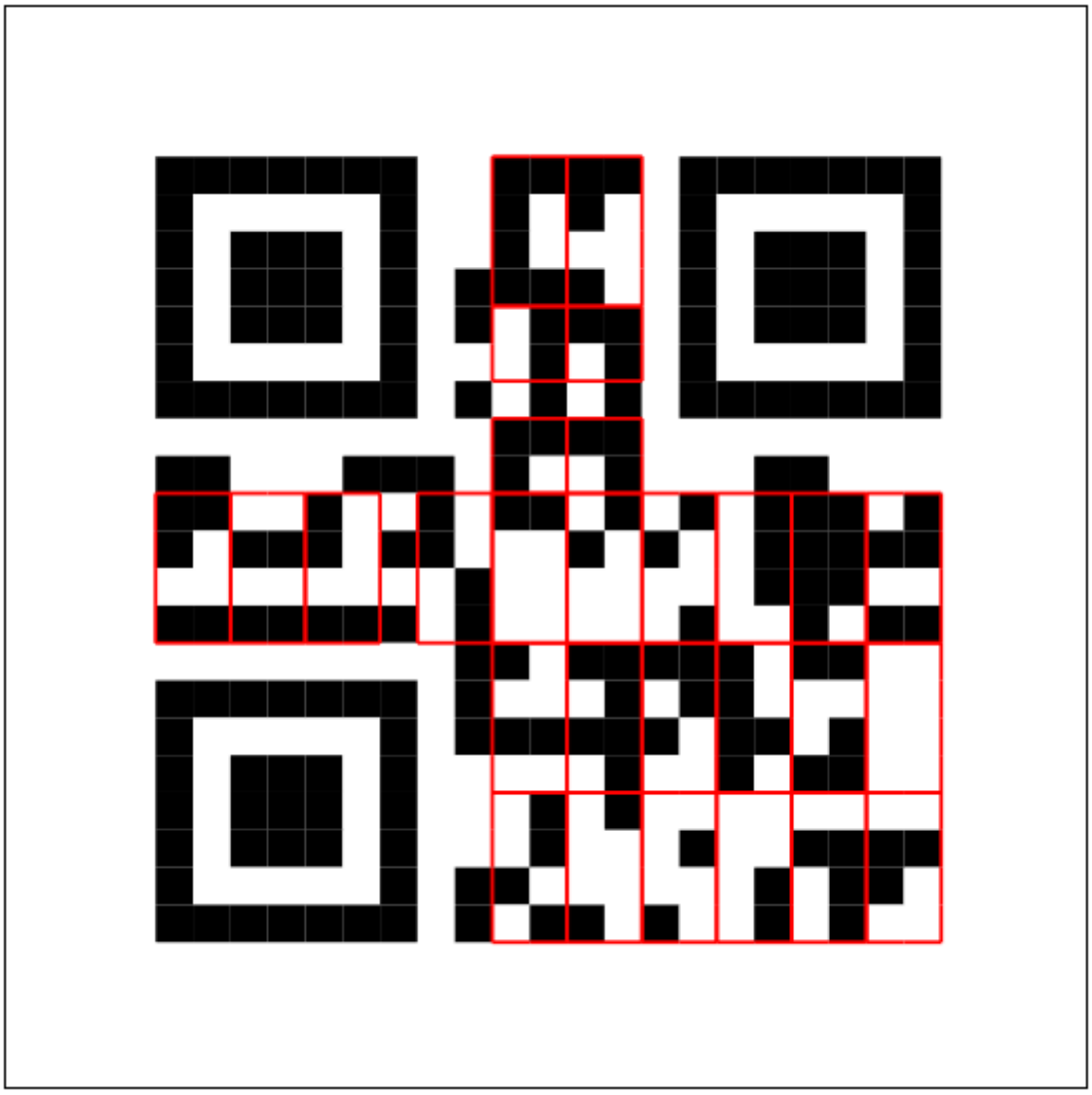
數字 16 代表這 255 個碼字中，最多可以有 16 個碼字發生錯誤，如果預先知道哪幾個碼字有錯，則最多可以有 32 個碼字發生錯誤。

QR code 組成

在開始編碼前，我們先回頭來看 QR 的組成：

在 QR code 中，一個資料(校正)區塊代表 8 個黑/白小塊。

整個 QR code 扣除功能圖案部分，就是由許多這種資料(校正)區塊組成。



QR 中的每個區塊(8小塊)，對應 RS 編碼中的資料(校正)區塊，上圖就會使用 (26,16,5) 的 RS 編碼。(總共有 26 個區塊(8小塊)，其中有一部份是 4 小塊一組，那要和下面接著 4 小塊的一起看。)

RS 編碼 in GF(256)

在這裡我們以 $(10, 5, 1)$ 格式的 RS 編碼為例。

1. 根據格式，可以有 5 個資料碼，這裡以 $(148, 80, 113, 154, 141)$ 為例，對應 $(\alpha^{38}, \alpha^{54}, \alpha^{94}, \alpha^{146}, \alpha^{197})$
2. 資料轉換 $d(x) = \alpha^{38} + \alpha^{54}x + \alpha^{94}x^2 + \alpha^{146}x^3 + \alpha^{197}x^4$
3. 查表 (Table A.1) 得到對應的 generator polynomials $g(x) = x^5 + \alpha^{113}x^4 + \alpha^{164}x^3 + \alpha^{166}x^2 + \alpha^{119}x + \alpha^{10}$
4. 計算餘式 $r(x) = d(x)x^5 \mod g(x)$
5. 最後得到編碼後的結果 $C(x) = d(x)x^5 + r(x)$

- 其中的$d(x)x^5$ 乘上的 x^5 是根據 $10 - 5 = 5$ 得到
- 會有一個性質 $C(x) \mod g(x) = 0$ 當傳輸沒有錯誤時。

RS 編碼 in GF(2)

因為 $GF(256)$ 上的計算過於複雜，所以這裡計算 $GF(2)$ 的版本 (相當於 QR code 中的每一個 黑/白 小塊就是一個資料區塊)。

1. 資料以 `10100` 為例
2. 資料轉換 $d(x) = x^4 + x^2$
3. 對應的 generator polynomials
   $g(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) = (x - 1)^5$
   $= (x^2 - 2x + 1)^2(x - 1) = (x^2 + 1)^2(x - 1)$
   $= (x^4 + 2x^2 + 1)(x - 1) = (x^4 + 1)(x - 1)$
   $= x^5 - x^4 + x - 1 = x^5 + x^4 + x + 1$
4. 計算餘式 $r(x) = [d(x)x^5 \mod g(x)] = x^2 + 1$

```
  x pow ->   9 8 7 6 5 4 3 2 1
             -------------------
d(x)x^5 ->   1 0 1 0 0 0 0 0 0
g(x)    ->   1 1 0 0 1 1
             -------------------
               1 1 0 1 1 0 0 0
               1 1 0 0 1 1
             -------------------
r(x)   ->            1 0 1 0 0
```

   - 這裡在 $GF(2)$ 內，加(減)法相當於 XOR，若是在 $GF(256)$ 時，就必須在多項式上(或說轉成二進制上)做加法再轉換回來。
     例如說 $\alpha^2 + \alpha^3 = 4 + 8 = (0000\ 0100) + (0000\ 1000) = (0000\ 1100) = 12 = \alpha^{27}$
5. 最後得到編碼後的結果 $C(x) = d(x)x^5 + r(x) = x^9 + x^7 + x^4 + x^2$
   對應 `10100 10100`

RS 編碼 in GF(4)

數字形式	位元形式	多項式形式	α^i 形式
0	00	0	0
1	01	1	α^0
2	10	x	α^1
3	11	$x + 1$	α^2

1. 資料以 `01230` 為例
2. 資料轉換 $d(x) = x^3 + \alpha x^2 + \alpha^2 x$
3. 對應的 generator polynomials

$$
\begin{aligned}
g(x) &= (x-\alpha^0)(x-\alpha^1)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) \\
&= (x-1)(x-\alpha)(x-\alpha^2)(x-1)(x-\alpha) \\
&= (x^2+(-1-\alpha)x+\alpha)(x-\alpha^2)(x-1)(x-\alpha) \\
&= (x^2+\alpha^2 x+\alpha)(x-\alpha^2)(x-1)(x-\alpha) \\
&= (x^3+\alpha^2x^2+\alpha x-\alpha^2x^2-\alpha^4x-\alpha^3)(x-1)(x-\alpha) \\
&= (x^3-1)(x^2+\alpha^2x+\alpha) \\
&= (x^5+\alpha^2x^4+\alpha x^3-x^2-\alpha^2x-\alpha) \\
&= (x^5+\alpha^2x^4+\alpha x^3+x^2+\alpha^2x+\alpha)
\end{aligned}
$$

4. 計算餘式 $r(x) = [d(x)x^5 \mod g(x)] = \alpha x^4 + x^3 + \alpha x^2 + \alpha^2 x + \alpha^2$

```
  x pow ->   9 8 7 6 5 4 3 2 1
             ------------------
d(x)x^5 ->   0 1 2 3 0 0 0 0 0
g(x)    ->   1 3 2 1 3 2
             ------------------
               2 0 2 3 2 0 0 0
               2 1 3 2 1 3
             ------------------
                 1 1 1 3 3 0 0
                 1 3 2 1 3 2
             ------------------
                   2 3 2 0 2 0
                   2 1 3 2 1 3
             ------------------
r(x)   ->            2 1 2 3 3
```

5. 最後得到編碼後的結果
   $C(x) = d(x)x^5 + r(x) = x^8 + \alpha x^7 + \alpha^2 x^6 + \alpha x^4 + x^3 + \alpha x^2 + \alpha^2 x + \alpha^2$
   對應 `01230 21233`

`01231 21233`

1.

$\left\{\begin{array}{l}
S_1 = R(1) = r_0 + r_1 + r_2 + \cdots + r_{9} \\
S_2 = R(\alpha) = r_0 + r_1\alpha + r_2\alpha^2 + \cdots + r_{9}\alpha^{9}
\end{array}\right. $

$\left\{\begin{array}{l}
S_1 = R(1) = 1 \\
S_2 = R(\alpha) = 1\cdot \alpha^4 = \alpha
\end{array}\right. $

2.

$\left\{\begin{array}{l}
S_1\sigma_1 - S_2 = 0
\end{array}\right.$

$\left\{\begin{array}{l}
\sigma_1 - \alpha = 0
\end{array}\right.$

$\implies \sigma(x) = (1+\beta_1 x) = \sigma_0 + \sigma_1x = 1 + \alpha x$

3.

$\sigma(\alpha^2) = 0$

$\implies x = -\beta_i^{-1} = \beta_i^{-1} \implies \beta_i = x^{-1} = \alpha$

$\alpha = \alpha^1 = \alpha^4 = \alpha^7 = \alpha^{10}$

GF 的大小要大過編碼長度？

4.

解碼

延續前面的符號，不過這裡使用 $(26,16,4)$ RS 格式 (依據規格書附錄範例)。

符號定義

- $R = (r_0,r_1,r_2,\cdots,r_{25})$：接收到的訊息
  $R(x) = r_0 + r_1x + r_2x^2 + \cdots + r_{25}x^{25}$

- $C=(c_0,c_1,c_2,\cdots,c_{25})$：正確的訊息
  $C(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{25}x^{25}$

- $E=(e_0,e_1,e_2,\cdots,e_{25})$：傳輸過程造成的錯誤
  $E(x) = e_0 + e_1x + e_2x^2 + \cdots + e_{25}x^{25}$

$\left\{\begin{array}{l}
R(x) = C(x) + E(x) && && \cdots(1)\\
R(x)= (c_0 \bigoplus e_0) + (c_1\bigoplus e_1)x + (c_2\bigoplus e_2)x^2 + \cdots + (c_{25}\bigoplus e_{25})x^{25} && && \cdots(2)
\end{array}\right.$

$(1),(2)$ 中的 $+$ 是 $GF(2^8)[X]$ 中的加法，一般的多項式加法。

(2) 中的 $\bigoplus$ 是 $GF(2^8) $ 中的加法(同減法)，是把兩個元素換成二進制後做 XOR (XOR 同時也是 $\mathbb{Z}_2$ 中的加(減)法)。

RS 解碼 in GF(256)

因為解碼相對於編碼複雜許多，所以分成了四個步驟來做：

1. 計算徵狀值 (Syndromes)
2. 決定錯誤位置多項式 $\sigma(x)$
3. 找出錯誤位置 (即解出 $\sigma(x)$ 的根)
4. 計算錯誤位置上的錯誤值

1. 計算徵狀值 (Syndromes)

根據前面的定法，我們有

$\left\{
\begin{array}{l}
C(x) = d(x)x^{n-k} + r(x) \\
R(x) = C(x) + E(x)
\end{array}
\right.$

其中 $C(x)$ 代入 $\alpha^0,\alpha^1,\cdots,\alpha^7$ 都會是 0

- 因為 $d(x)x^{n-k} = g(x)\times Q(x) + r(x) = g(x)\times Q(x) - r(x)$
  (轉成多項式來看就可以發現，加(減)法相同)
  所以 $C(x) = g(x)\times Q(x)$
- 另一點則是 generator polynomials 的定法 $g(x) := (x-\alpha^0)(x-\alpha^1)(x-\alpha^2)\cdots(x-\alpha^6)(x-\alpha^7)$

所以有 $R(\alpha^i)=C(\alpha^i)+E(\alpha^i) = E(\alpha^i),\quad \forall\> 0\leq i \leq 7$

然後就可以計算徵狀值 (syndromes)

$\left\{\begin{array}{l}
S_1 = R(1) = r_0 + r_1 + r_2 + \cdots + r_{25} \\
S_2 = R(\alpha) = r_0 + r_1\alpha + r_2\alpha^2 + \cdots + r_{25}\alpha^{25} \\
\cdots \\
S_8 = R(\alpha^7) = r_0 + r_1\alpha^7 + r_2\alpha^{14} + \cdots + r_{25}\alpha^{175}
\end{array}\right. $

- 會有幾條要看 QR code Table.9 的 $c-k-p$
- $(c,k,r) = (26,16,4)$ 的 $p=2$，有 8 條

同時我們知道：

$\left\{\begin{array}{l}
S_1 = e_{j_1}\alpha^{j_1} + e_{j_2}\alpha^{j_2} + \cdots + e_{j_v}\alpha^{j_v} \\
S_2 = e_{j_1}\alpha^{2j_1} + e_{j_2}\alpha^{2j_2} + \cdots + e_{j_v}\alpha^{2j_v} \\
\cdots \\
S_8 = e_{j_1}\alpha^{8j_1} + e_{j_2}\alpha^{8j_2} + \cdots + e_{j_v}\alpha^{8j_v} \\
\end{array}\right.$

$j_1,\cdots,j_v$ 代表 v 個出錯的位置 (可能是 $1,2,4,5$ 之類)

簡化符號：$\delta_i := e_{j_i}$ 及 $\beta_i := \alpha^{j_i}$

$\left\{\begin{array}{l}
S_1 = \delta_1\beta_1 + \delta_2\beta_2 + \cdots + \delta_v\beta_v \\
S_2 = \delta_1\beta_1^2 + \delta_2\beta_2^2 + \cdots + \delta_v\beta_v^2 \\
\cdots \\

S_8 = \delta_1\beta_1^8 + \delta_2\beta_2^8 + \cdots + \delta_v\beta_v^8 \\
\end{array}\right. \qquad \cdots(1)$

再定義 $\sigma(x) = (1+\beta_1 x)(1+\beta_2 x)\cdots(1+\beta_v x) = \sigma_0 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_v x^v$

$\left\{\begin{array}{l}
\sigma_0 = 1 \\
\sigma_1 = (\beta_1+\beta_2+\cdots+\beta_v) \\
\cdots \\
\sigma_v = \beta_1\beta_2\cdots\beta_v \\
\end{array}\right.\qquad\cdots(2) $

由 (1), (2) 轉換?得到

$\left\{\begin{array}{l} S_1\sigma_4 - S_2\sigma_3 + S_3\sigma_2 - S_4\sigma_1 + S_5 = 0 \\ S_2\sigma_4 - S_3\sigma_3 + S_4\sigma_2 - S_5\sigma_1 + S_6 = 0 \\ S_3\sigma_4 - S_4\sigma_3 + S_5\sigma_2 - S_6\sigma_1 + S_7 = 0 \\ S_4\sigma_4 - S_5\sigma_3 + S_6\sigma_2 - S_7\sigma_1 + S_8 = 0 \\ \end{array}\right.$

$ S_{i+v} + \sigma_1S_{i+v-1} + \cdots + \sigma_vS_i = 0, i=1,\cdots,2t-v $

---

2. 決定錯誤位置多項式 $\sigma(x)$

$\left\{\begin{array}{l}
S_1\sigma_4 - S_2\sigma_3 + S_3\sigma_2 - S_4\sigma_1 + S_5 = 0 \\
S_2\sigma_4 - S_3\sigma_3 + S_4\sigma_2 - S_5\sigma_1 + S_6 = 0 \\
S_3\sigma_4 - S_4\sigma_3 + S_5\sigma_2 - S_6\sigma_1 + S_7 = 0 \\
S_4\sigma_4 - S_5\sigma_3 + S_6\sigma_2 - S_7\sigma_1 + S_8 = 0 \\
\end{array}\right.$

4 個未知數、4 條方程式，可以解出 $\sigma_1,\sigma_2,\sigma_3,\sigma_4$

- 在 $(26,16,4) $ 格式下 ，如果 $v>4$，就會解不出錯誤位置。

3. 找出錯誤位置 (即解出 $\sigma(x)$ 的根)

把 $GF(256)$ 中所有元素一一代入，即可找到 $\sigma(x)$ 所有的根。

同時因為 $\sigma(x)$ 的形式，找到根相當於找到 x 使得 $1+\beta_i x = 0$$\implies x = -\beta_i^{-1} = \beta_i^{-1} \implies \beta_i = x^{-1}$

找到 β_i 相當於找到 α^{j_i}，再查表 (看 α 的幾次會等於 $ \beta_i$) 就可以知道 j_i。

4. 計算錯誤位置上的錯誤值

知道 β_i 後，再回頭計算這個聯立方程組。

$\left\{\begin{array}{l}
S_1 = \delta_1\beta_1 + \delta_2\beta_2 + \cdots + \delta_v\beta_v \\
S_2 = \delta_1\beta_1^2 + \delta_2\beta_2^2 + \cdots + \delta_v\beta_v^2 \\
\cdots \\
S_8 = \delta_1\beta_1^8 + \delta_2\beta_2^8 + \cdots + \delta_v\beta_v^8 \\
\end{array}\right. \qquad \cdots(1)$

v 個未知數，8 條方程式，只要 $v\leq 8$ 就能解出來。

- 所以如果知道錯誤位置，最多可以找到 8 個錯誤。
  完全不知道的情況就只能找到 4 個錯誤。

RS 解碼 in GF(2)

我們以前面編碼出的結果來解碼。

資料碼	編碼結果	錯誤 case 1	錯誤 case 2
$\texttt{10100}$	$\texttt{10100 10100}$	$\texttt{10100 1}{\color{red}\texttt{1}}\texttt{100}$	$\texttt{1010}{\color{red}\texttt{1 }}1{\color{red}\texttt{1}}\texttt{10}{\color{red}\texttt{1}}$

1. 計算徵狀值 (Syndromes)
2. 決定錯誤位置多項式 $\sigma(x)$
3. 找出錯誤位置 (即解出 $\sigma(x)$ 的根)
4. 計算錯誤位置上的錯誤值

case 1

$(c,k,r) = (10,5,1)$ ，$p=3$ 的 RS code

$\texttt{10100 1}{\color{red}\texttt{1}}\texttt{100}$ 錯誤在第 6 個位置 (從 0 開始算)

1. 計算徵狀值

- 看有幾條 $c-k-p = 10-5-3 = 2$

$\left\{\begin{array}{l}
S_1 = R(1) = r_0 + r_1 + r_2 + \cdots + r_{9} \\
S_2 = R(\alpha) = r_0 + r_1\alpha + r_2\alpha^2 + \cdots + r_{9}\alpha^{9}
\end{array}\right. $

在 $GF(2)$ 中，元素只有 $\{0, 1\} $，$\alpha=1$

$\left\{\begin{array}{l}
S_1 = 1+0+1+0+0+1+1+1+0+0 = 5 = 1 \\
S_2 = 1 + 1\cdot \alpha^2 + 1\cdot \alpha^5 + 1\cdot \alpha^6 + 1\cdot \alpha^7 = 1 + 1 + 1 + 1 + 1 = 5 = 1
\end{array}\right.$

2. 決定錯誤位置多項式 $\sigma(x)$

$\left\{\begin{array}{l}
S_1\sigma_1 - S_2 = 0
\end{array}\right.$

$\implies \sigma_1 - 1 = 0 \implies \sigma_1 = 1$

$\implies \sigma(x) = (1+\beta_1 x) = \sigma_0 + \sigma_1x = 1 + x$

3. 找出錯誤位置 (即解出 $\sigma(x)$ 的根)

代入所有元素 $\{0, 1\}$

$\left\{\begin{array}{l}
\sigma(0) = 1 \\
\sigma(1) = 1 + 1 = 2 = 0
\end{array}\right.$

$\implies 1$ 是 $\sigma(x)$ 的一個根

$\implies \beta_1 = 1$

4. 計算錯誤位置上的錯誤值

參考：

[2012] 里德所羅門碼之運算分析.pdf

[1960] POLYNOMIAL CODES OVER CERTAIN FINITE FIELDS.pdf

Convolution_RS.pdf

[2004] 可重複規劃之里德所羅門解碼器設計.pdf

BCH Codes

BCH Codes -- wiki

In QR Code

在 QR code 中，Format Information 是用 BCH 編碼。

但在 QR code 的 BCH 解碼方式是尋找距離最近的(或說錯誤最少的)答案，而不是(像 RS Codes)精準的解出每一個錯誤位置與值。

漢明距離 (Hamming distance)

漢明距離是計算兩個**等長**字符串**對應位置**的**不同字符**的數量。

舉例來說：

`1011` 和 `1011` 距離是 0

`1011` 和 `1001` 距離是 1

`1011` 和 `1101` 距離是 2

漢明距離 -- wiki

BCH Codes 編碼

在 QR codes 中，使用 (15,5) 的 BCH Codes。(總長度為 15，資料長度 5)

固定使用 generator function $g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$ (`10100110111`)

以容錯率 M 、XOR遮罩編號 5，做為資料 `00101`

編碼過程

1. 資料轉換 $d(x) = x^2 + 1$
2. 乘上 x^{15-5} ：$d(x)x^{10} = x^{12} + x^{10}$
3. 計算餘式 $r(x) = [d(x)x^{10} \mod g(x)] = x^7 + x^6 + x^4 + x^3 + x^2$

```
  x pow  ->   12 11 10 9 8 7 6 5 4 3 2 1 0
             -------------------------------
d(x)x^10 ->    1  0  1 0 0 0 0 0 0 0 0 0 0
g(x)     ->    1  0  1 0 0 1 1 0 1 1 1
             -------------------------------
r(x)     ->                1 1 0 1 1 1 0 0
```

4. 得到編碼結果 $C(x) = d(x)x^{10} + r(x) = x^{12} + x^{10} + x^7 + x^6 + x^4 + x^3 + x^2$

`00101 00110 11100`

做 XOR 遮罩 (防止出現太多連續黑/白)

固定使用 $X =$`10101 00000 10010`

與編碼結果 C 做 XOR

```
    00101 00110 11100
XOR 10101 00000 10010
-----------------------
    10000 00110 01110
```

最後放置在 QR code 上的圖案就是 `10000 00100 01110`

BCH Codes 的性質

- 在 BCH Codes 中，有某些 generator function 可以讓生成的碼之間的漢明距離特別大。

在 (15,5) 的 BCH 碼以及 $g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$ 的情況，生成的碼之間的距離最小是 7 ，用比對漢明距離的方式，只要錯 3 個以內都可以還原回來。

比對表

解碼過程

Golay Codes

Golay Codes -- wiki

In QR Code

在 QR code 中，version Information 是用 Golay 編碼。

與 BCH Codes 相似，他也是尋找距離相近的答案，而不是(像 RS Codes)精準的解出每一個錯誤位置與值。

在 (18,6) 的 Golay 碼以及 $g(x) = x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^5 + x^2 + 1$ 的情況，生成的碼之間的距離最小是 8 ，用比對漢明距離的方式，只要錯 3 個以內都可以還原回來。

Golay Codes 編碼

在 QR codes 中，使用 (18,6) 的 Golay Codes。(總長度為 18，資料長度 6)

固定使用 generator function $g(x) = x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^5 + x^2 + 1$ （`1111100100101`）

以版本 7 為例，資料為 `000111`

編碼過程

1. 資料轉換 $d(x) = x^2 + x + 1$
2. 乘上 x^{18-6} ：$d(x)x^{12} = x^{14} + x^{13} + x^{12}$
3. 計算餘式 $r(x) = [d(x)x^{12} \mod g(x)] = x^{11} + x^{10} + x^7 + x^4 + x^2$

```
  x pow  ->   14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
             ---------------------------------
d(x)x^12 ->    1  1  1  0  0 0 0 0 0 0 0 0 0 0 0
g(x)     ->    1  1  1  1  1 0 0 1 0 0 1 0 1
             ---------------------------------
r(x)     ->             1  1 0 0 1 0 0 1 0 1 0 0
```

4. 得到編碼結果 $C(x) = d(x)x^{12} + r(x) = x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^7 + x^4 + x^2$

   `00011 11100 10010 100`

比對表

解碼過程

浮水印的需求

在現代的通訊系統中，有許多資料傳輸都是以加密形式進行 (如: https 就會在傳輸封包前，先將資料加密)，這些加密的目的都是：**即使攻擊者知道東西被加密，也無法破解**，這些都是屬於**資訊加密**的範疇。

就像是「把東西鎖在箱子」裡，但是也有一些需求是「把東西給別人看，但別人拿不走」，以及「檢查東西的完整性」。在影像浮水印的章節裡，這些東西就是影像；添加浮水印，就能完成這兩個需求。

對於「把東西給別人看，但別人拿不走」，就加入一個難以移除 (強健型) 的浮水印。
對於「檢查東西的完整性」，就加入一個容易被破壞 (易碎型) 的浮水印。

浮水印的分類

- 依視覺感官
  - 可見型：肉眼可見的浮水印。最常見的可見型浮水印，是將 Logo 以一定的透明度然後覆蓋在其他圖片上。
  - 不可見型：肉眼不可見。
- 依強韌程度
  - 易碎型：只要圖片稍作修改就會被破壞，會具有偵測錯誤的能力，甚至有搭配易碎浮水印修復影像的演算法。
  - 強健型：不論對圖片作甚麼修改都不容易去除掉，可以用作版權保護、追蹤。
- 依嵌入方式
  - 空間域：在空間域上作處理的浮水印，容易處理、容量大、易被破壞。
  - 頻率域：先將圖片轉換到頻率域再作處理，容量小，但不易被破壞。
- 依取出方式
  - 非盲目型：需要原始影像、浮水印、金鑰來萃取出浮水印。
  - 半盲目型：需要浮水印和金鑰來萃取浮水印。
  - 盲目型：需要金鑰就可以萃取浮水印。

註：這些分類都是獨立的，可以有一個不可見、易碎、空間域轉換的盲目型浮水印；也可以有一個不可見、強韌、頻率域轉換的半盲目型浮水印。

浮水印嵌入系統

- 兩個步驟
  - 嵌入演算法 (藏入浮水印的過程)
  - 取出演算法 (取出浮水印的過程)
  - (?) 偵測演算法 (可能會從取出演算法細分出來)
- 三種資料
  - 浮水印 (要隱藏的資料)
  - 原始圖案 (資料載體，用來把浮水印隱藏進去)
  - 偽裝圖案 (資料載體，已經藏好浮水印)
  - (?) 金鑰 (對於要公開**取出演算法**的系統會有)

可見型浮水印

一般常見的浮水印嵌入方法：$A' = A \cdot (1 - (1 - B) \cdot \alpha)$；$A$ 是原圖、B 是浮水印、α 是浮水印的透明程度`。

原圖：

浮水印：

B	$1 - B$	$(1 - B) \cdot \alpha$	$1 - (1 - B) \cdot \alpha$
			

添加浮水印 (右下方) 後的圖片：

```
%-- ex.1
%-- 讀圖檔、轉到 [0,1]
cimg = imread('b271.jpg');
cimg = im2double(cimg);
```

```
[ch , cw , d] = size(cimg);

%-- 讀圖檔、轉到 [0,1]
mask = imread('mark.png');
mask = im2double(mask);
[mh , mw , d] = size(mask);

%--  把白色當成背景，把非白色的部分調整亮度(透明度)
mask = 1 - (1 - mask) * 0.4;

%-- 離邊邊距離 100 pixels
padding_left = 100;
padding_bottom = 100;

%-- 框出要覆蓋的範圍
x_start = cw - padding_left - mw;
x_end = cw - padding_left - 1;

y_start = ch - padding_bottom - mh;
y_end = ch - padding_bottom - 1;

A = cimg( y_start : y_end , x_start : x_end , : );
%-- 用相乘重新設定亮度
cimg( y_start : y_end , x_start : x_end , : ) = A.*mask;
imshow(cimg)
```

不可見型浮水印

不可見型浮水型指的是**不容易發現**，通常要藏入越多資料或是要增強浮水印的強健性，就越容易看出圖片被修改。對於這種不可見浮水印，通常有兩個指標衡量改變的程度。

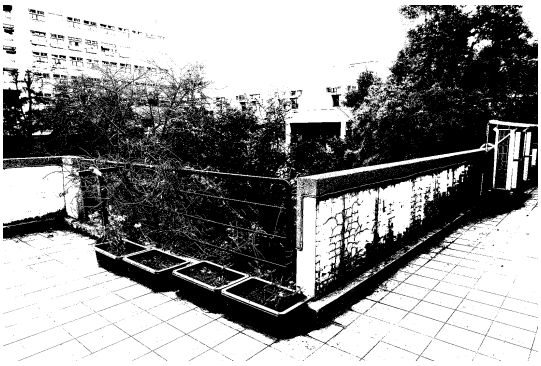
- 一個是 MSE (Mean Square Error)
  $MSE = \frac{1}{MN} \sum\sum [I(x,y) - K(x,y)]^2$
- 以及 PSNR (Peak signal-to-noise ratio)
  $PSNR = 20 \cdot \log_{10}(\frac{MAX_I}{\sqrt{MSE}})$
  MAX_I 是圖片最大值，通常就是 255；一般而言 PSNR 大於 40 就可以接受的。

LSB 浮水印 (Least Significant Bit)

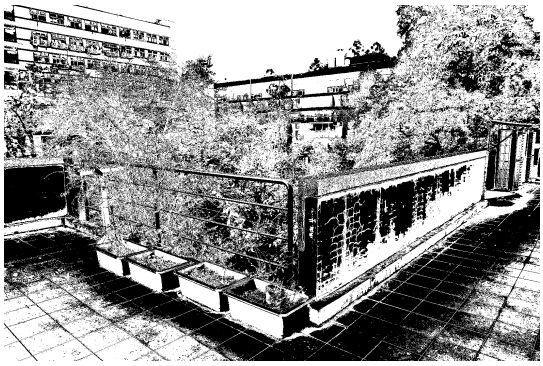
LSB 浮水印是使用圖片中每個像素的最低位元來嵌入浮水印。

從圖片的位元切片 (只取其中一個平面) 可以看到，會發現在第 5 位元以後就幾乎看不清楚是甚麼東西了。
(為了展示，這邊都有將亮度做拉伸成 0 和 255)

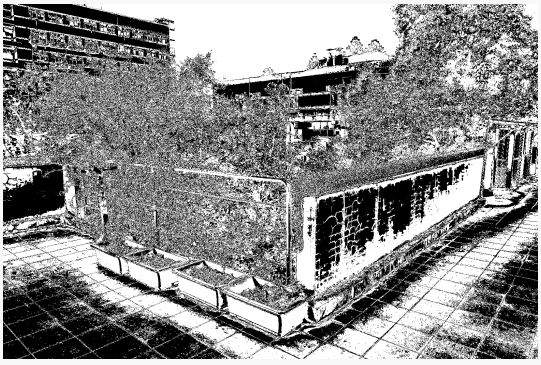
第0位元



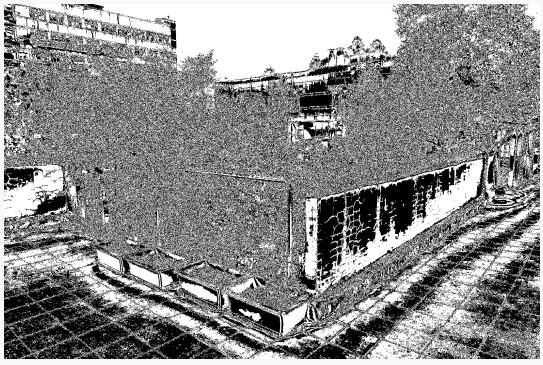
第1位元



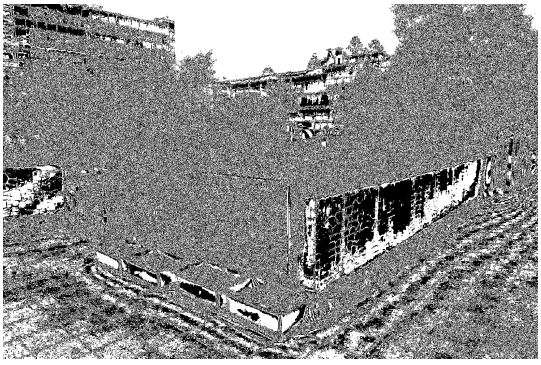
第2位元



第3位元

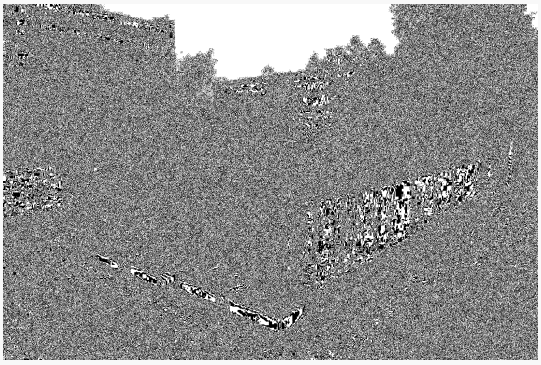


第4位元

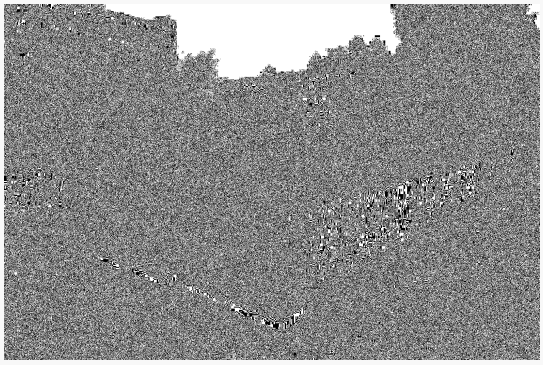


第5位元

第6位元



第7位元



另外一個角度是從一個一個平面去掉來看，沒有取到的平面通通設成 0；這裡會發現只取 0~3 位元平面的結果就和取了 0~7 平面的結果差不多。

0位元平面

0,1位元平面

0~2位元平面

0~3位元平面

0~4位元平面

0~5位元平面

0~6位元平面

0~7位元平面

透過這樣的觀察，可以發現

1. 在圖像複雜的部份可以取越少平面，單調的地方則要取越多。
2. 大概取 0~3 平面，共 4 個位元平面就差不多可以代表一張圖片了。

所以這個浮水印的做法就是將資訊嵌入在圖像的最後幾個位元。

使用二值影像嵌入灰階影像 - 1

一開始，我們先嘗試將二值影像嵌入一張灰階影像。

策略：挑選最後一個**位元平面歸零**，在最後一個**位元平面加上浮水印的 0 或 1 值**。

添加浮水印： 

添加前	添加後
	

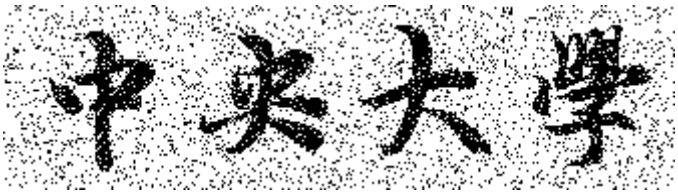
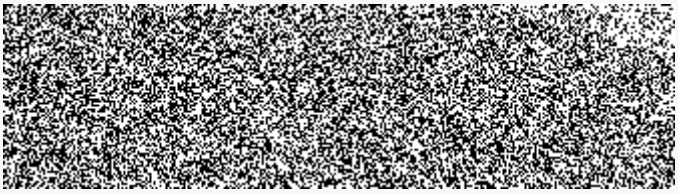
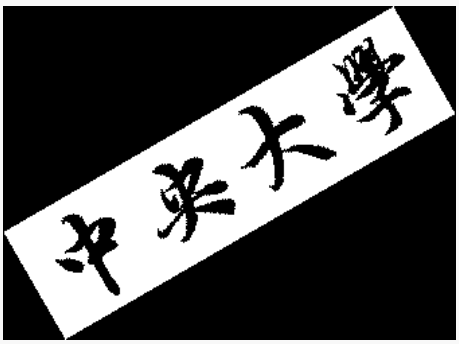
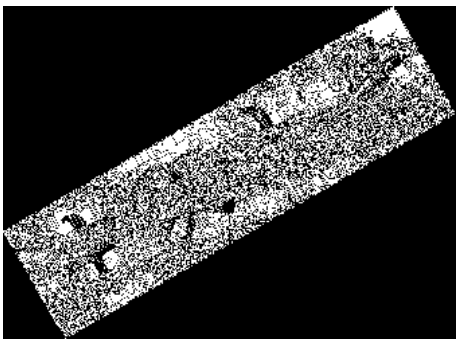
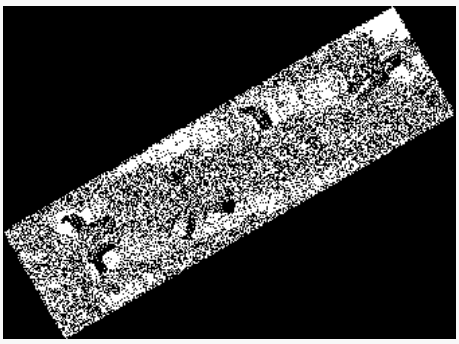
因為是不可見浮水印，所以肉眼不容易看到。(要注意實作時要儲存成 `.png` 不然會經過 `jpg` 壓縮導致浮水印消失。)
可以計算 PSNR = 50.9133 (通常 40 以上是可以接受的)

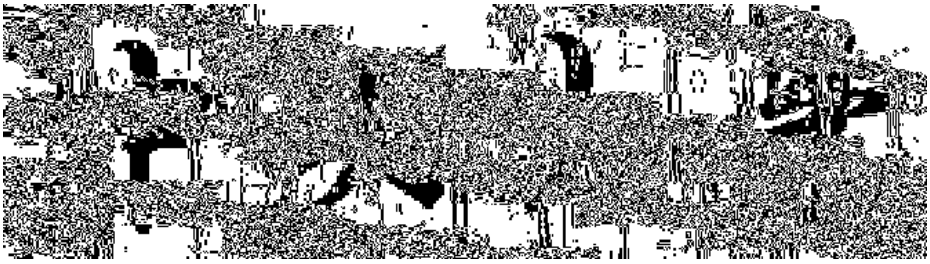
```
%-- ex.4
%-- 讀圖檔、轉到灰階
cimg = imread('b032.jpg');
gimg = rgb2gray( cimg );

%-- 複製一份 gimg
A = gimg;
%-- 歸零最後一個位元平面
A = A - bitand(gimg,1);

%-- 讀浮水印圖檔
mask = imread('mask.png');
[ mh , mw , d ] = size(mask);

%-- 範圍左上角
range_x = 1:mw;
range_y = 1:mh;
A( range_y , range_x ) = A( range_y , range_x ) + uint8(mask);
imwrite( A , 'lsb1.png' );

%-- 計算 MSE
mse = mean(mean( (gimg - A).^2 ))
psnr = 20*log10( 255 / sqrt(mse) )
```

添加完後可以在萃取出來


```
%-- ex.5
%-- 讀圖檔
gimg = imread('lsb1.png');
imshow( bitand( gimg , 1 )*128  );
```

接著就可以添加各種雜訊攻擊。

雜訊	結果
20% 椒鹽雜訊	
高斯雜訊	
裁切 (如果切到的話)	
旋轉(最近鄰內插)	
旋轉(雙線性內插)	
旋轉(雙三次內插)	
放大(最近鄰內插)	

雜訊	結果
縮小(最近鄰內插)	
放大(雙三次內插))	
縮小(雙三次內插))	
3×3平均模糊	
jpg 壓縮	

從上面看到可以發現，會更改到像素值的，對於 LSB 的影響都很大。

對於平均遮罩，統計一下模糊前後的差距，發現平均在 0 附近，標準差則有 16 左右；也就是說在第三位以後的數字 `000x xxxx` 在模糊後都會隨便亂跳，除非把資訊藏到前三個位元裡面才有可能克服這點。

對於裁切我們則可以想把辦法資料藏在圖片的各個地方，避免一個部分被裁切掉導致浮水印消失。

使用二值影像嵌入灰階影像 - 對抗裁切

策略：把浮水印藏在圖片的各個角落，克服裁切的影響。

這裡使用一個亂數產生器來生成藏資料的位置，而這個亂數產生器會需要一個**種子** (相同種子會生成相同的亂數序列) ，這個**種子**同時也是這個浮水印演算法的**金鑰** (即使知道整個浮水印演算法，也必須要有**金鑰**才有辦法偵測、取出浮水印)。

這裡一樣展示一下嵌入前後的圖片，不過因為是嵌入在最後一個位元，所以看不出差別。

嵌入前	嵌入後
	

```
%-- ex.8
%-- 讀圖檔、轉到灰階
cimg = imread('b032.jpg');
gimg = rgb2gray( cimg );
[ gh , gw , d ] = size(gimg);
A = gimg;
%-- 讀浮水印圖檔
mask = imread('mask.png');
[ mh , mw , d ] = size(mask);

%-- 設定亂數種子
seed = 7852;
rng(seed);

for y=1:mh
    for x=1:mw
        yy = randi(gh);
        xx = randi(gw);
        %-- 歸零選中位置最後一個位元
        A( yy , xx ) = A( yy , xx ) - bitand( A( yy , xx ) , 1 );
        A( yy , xx ) = A( yy , xx ) + uint8(mask(y,x));
    end
end

imwrite( A , 'lsb2.png' );
```

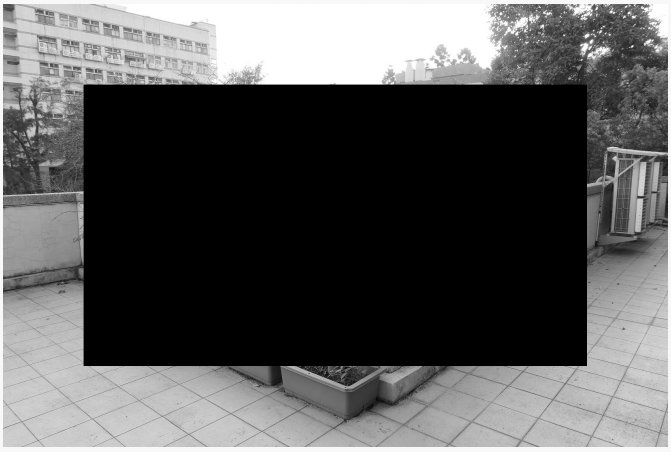
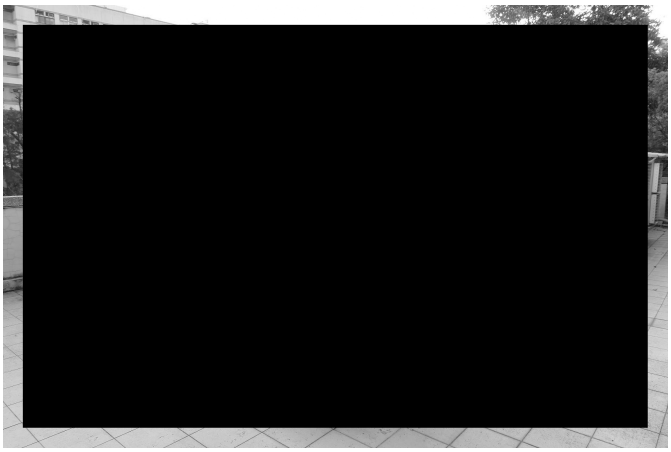
(未經過裁切) 取出結果： 

因為有可能會重複填到同一個位置，所以產生圖中的雜點。
(發生碰撞的機率近似是 $p(n) \sim 1 - 1/\exp(n^2/(2N))$，$n$ 是浮水印的 pixels 數量，N 是原圖片的 pixels 數量，取自生日問題wiki)
(這裡可以更進一步算出會有多少比例的浮水印出現再重複位置)

```
%-- ex.9
%-- 讀圖檔
gimg = imread('lsb2.png');

```

```
%-- 讀浮水印圖檔
mask = imread('mask.png');
%-- 圖片資訊不需要，但需要尺寸資訊
[ mh , mw , d ] = size(mask);

%-- 儲存擷取出的資料
B = zeros( mh , mw );

%-- 亂數種子
seed = 7852;
rng(seed);

for y=1:mh
    for x=1:mw
        yy = randi(gh);
        xx = randi(gw);
        %-- 挑選最後一個位元
        B(y,x) = bitand( A( yy , xx ) , 1 );
    end
end

imwrite( B , 'p21.png' )
```

經過不同位置、程度的裁切再取出

使用二值影像嵌入灰階影像 - 避免碰撞、多浮水印

由上面的實驗可以發現，使用 RNG (亂數產生器) 來挑選填入的位置可以大幅減少裁切造成的影響。
下一步，我們要避免碰撞，同時添加五次同樣的浮水印到圖片中，增加浮水印的抗裁剪的強度。

要避免碰撞，就要讓可以藏入資訊的地方變多， 這裡我要使用每個 pixel 的所有位元平面來藏，這樣就有原本 8 倍的容量。但這樣一旦修改到靠前面的位元平面，就會大大影響圖片，這裡就要引入一個失真補償的方法。

失真補償

失真補償是當藏入資訊後，把改變的數值盡量調整回來的方法。
它的做法是生成兩個值，然後從裡面挑比較好的一個。

1. 對於一個要藏入資料的 pixel `1010 1100` 分成三個部分 `A` `X` `B`。
    - `A` 是藏匿位置前的所有位元。
    - `X` 是藏匿位置的位元。
    - `B` 是藏匿位置後的所有位元。
2. 假設要藏在第 3 個位置 (從左邊、從 0 開始計算)，那可能會長這樣。
    - `A` = `101`
    - `X` = `0`
    - `B` = `1100`
3. `X` = `0` 而要藏入的是 `X'` = `1`，就要計算：
    - `A` `X` `0000` -1 = `101` `0` `0000` -1 = `100` `1` `1111`
    - `A` `X` `1111` $+1$ = `101` `0` `1111` $+1$ = `101` `1` `0000`
4. `X` = `1` 而要藏入的是 `X'` = `0`，就要計算：
    - `A` `X` `0000` -1 = `101` `1` `0000` -1 = `100` `0` `1111`
    - `A` `X` `1111` $+1$ = `101` `1` `1111` $+1$ = `110` `0` `0000`

註：如果 `X'` = `X` 就不用算了

總結來說，就是把 `B` 部分擺 `0` 然後減 1 ；以及把 `B` 部分擺 `1` 然後加 1 ，再看誰和原本的值比較接近。
上面的過程就是計算 `A` `X` `B` 這個值的上邊界和下邊界。

舉例來說，原本數值是`1010 1110` = 174
想要藏入第 2 個平面 (`X` 的位置) `10x0 1110`

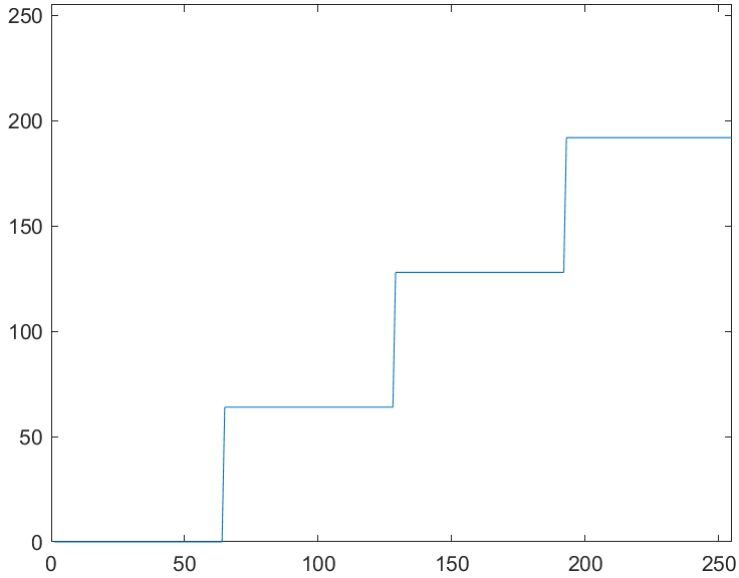
1. 如果要藏入 `1` ，值不變，那就甚麼都不用修改。
2. 如果要藏入 `0` ：
    - `10` `1` `0 0000` -1 = `10` `0` `1 1111` = 159
    - `10` `1` `1 1111` $+1$ = `11` `0` `0 0000` = 192
3. 挑選比較接近的 159 作為結果。

VQ based 浮水印

在介紹 VQ based 浮水印之前，首先要先了解 VQ 壓縮。

VQ 壓縮就像是取高斯函數一樣，將原本無數多的值，縮減到只有寥寥幾個，並用更少的位元數量來代表。

例如說你可以把原本: $2^8 = 256$ 種數值轉成只有 $2^2 = 4$ 種 (**色帶處理**)，那就壓縮了 4 倍。
原本需要儲存 $8 \times \text{pixels}_{\text{數量}}$ (bits)，壓縮後則儲存 $2 \times \text{pixels}_{\text{數量}} + \text{編碼字數量} \times 8$ (bits)



影像格式中的 GIF 就是用這種方式來儲存彩色 (只保存 256 種顏色)。

浮水印

VQ based 浮水印是將影像透過 VQ 壓縮，生成編碼簿 CB，並將影像編碼為**索引值**(下例的範圍是 $\{0, 1, \ldots, 255\}$)，再將浮水印(下例的範圍是 $\{0, 1\}$) 8 位一組後與索引值做 Xor，得到一組驗證資訊。

之後再透過 CB 對修改後的圖像編碼出索引值，與驗證資訊 Xor 得到浮水印。

注意：浮水印並沒有鑲嵌到圖像之中！而是透過浮水印與圖像生成一組驗證資訊。(實際應用會需要第三公正方來維護這些驗證資訊，證明誰先誰後。)

生成驗證資訊

1. 生成編碼簿 CB
2. 將影像編碼成**索引值** I
3. 使用亂數產生器從 I 中挑選 T 個位置，每個位置的數值與浮水印 M 做 Xor 得到驗證金鑰 K

驗證浮水印

1. 根據 CB 將影像編碼成索引值 I'
2. 使用相同亂數種子的亂數產生器從 I' 中挑選 T 個位置，每個位置的數值與 驗證金鑰 K 做 Xor 即可得到浮水印影像。

實作程式碼：

- 編碼 VQ_encode.m
- 解碼 VQ_decode.m
- 生成編碼簿 gen_CB.m
- 根據編碼簿計算索引值 cal_idx.m

實作結果

	圖片	浮水印
原圖		
jpg壓縮與裁切		

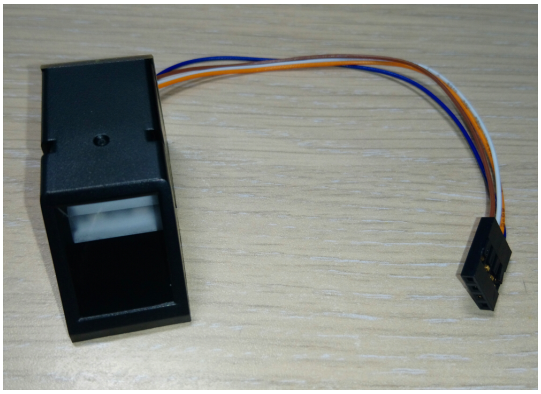
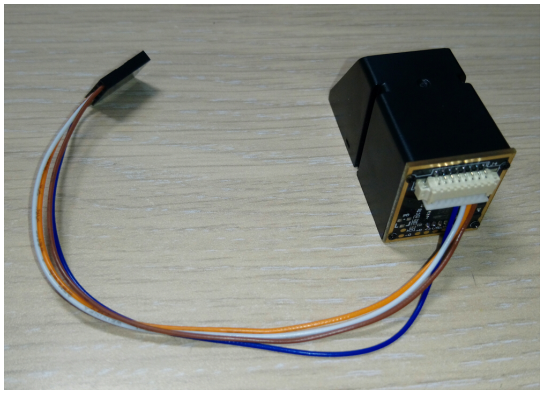
參考資料

挑戰影像處理 [數位浮水印技術], 潘正祥 張真誠 林詠章 著作

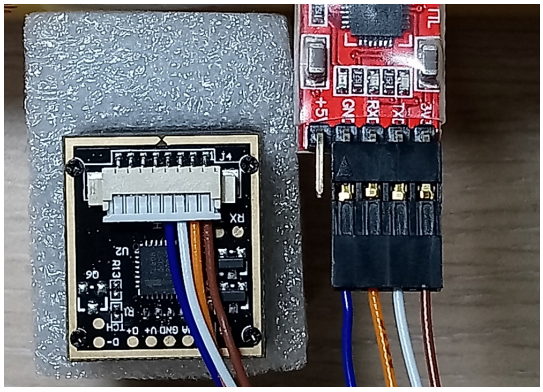
指紋擷取設備使用方式

硬體介紹

這裡使用市面上最容易取得的 JM-101B (AS608晶片) 指紋擷取模組來取得指紋照片。

正面	反面
	

並搭配 CP2102 (USB 轉 TTL 傳輸線) 連接上電腦來操作。

CP2102	與 AS608 連接
	

驅動程式

根據 USB 轉 TTL 傳輸線的型號不同，會需要裝不同的驅動程式。

- CP201x系列: CP210x_Universal_Windows_Driver.zip
- PL2303: pl2303_332102_64.zip

硬體操作指令集

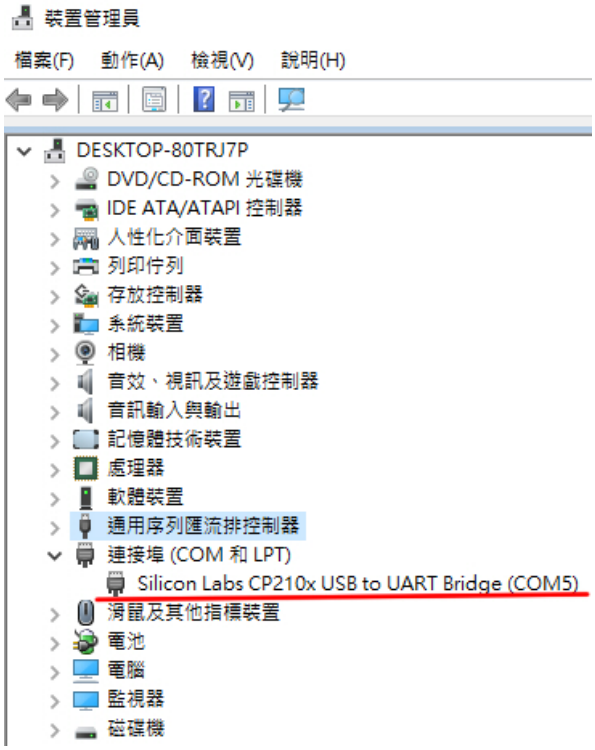
要和硬體裝置通訊需要遵守硬體開發商給的操作手冊。

通常上面會記載操作指令(通常是一連串16進制數字)。

AS608 晶片的操作手冊: [AS608通訊手冊.pdf]

將硬體連接上電腦

在連接硬體並裝好驅動程式後，就可以在電腦的"裝置管理員"中看到裝置了。

對左下的"開始圖案"按右鍵	裝置管理員介面
	

連接的裝置上顯示 "Silicon Labs CP210x USB to UART Bridge (COM5)"，
其中的 UART 是通訊協議，另外也有 SPI、I2C、USB 通訊協議等，
COM5 則是連接埠的編號，這個在撰寫程式時會用到，要記下來。

Python 程式撰寫

在硬體連接完畢，也確認出現在裝置管理員上之後，就可以開始撰寫程式控制這個指紋辨識模組了。

在 Python 中，可以使用 `pySerial` 這個函式庫來與連接上電腦的硬體溝通。

安裝指令: `pip install pyserial`

基礎操作

```python
import serial  # 引用pySerial模組

COM = "COM5"         # 根據在裝置管理員看到的序號
baudrate = 9600*12   # 根據硬體設定 (這代表傳輸速度 bit/s)

# timeout 是等待時間，如果接收訊息時，裝置超過 0.5 秒沒有回傳就中斷接收。
dev = serial.Serial( COM , baudrate , timeout=0.5 )
```

透過 `serial.Serial(...)` 就可以連接上裝置，並開始發送指令給裝置。

上面設定的 baudrate 會隨著硬體設定而變，AS608 晶片支援 9600~921600 bps；可以透過指令設定 baudrate，但要注意電腦到底有沒有支援設定的 baudrate。

如果設置了太高的 baudrate 而電腦、USB to TTL 模組沒有支援，就必須找別的硬體設備才有辦法再控制晶片。

這裡設定 115200 絕大多數硬體都可以運作的數值。

根據指令集編寫程式

這裡以第 1 個指令 `PS_GetImage` 來示範。

根據指令書上的資料 .....

---

(1) 錄入圖像 PS_GetImage

- 功能說明：探測手指，探測到後錄入指紋圖像存於 ImageBuffer。返回確認碼表示：錄入成功、無手指等。
- 輸入參數：none
- 返回參數：確認字
- 指令代碼：01H
- 指令封包格式

2 bytes	4 bytes	1 byte	2 bytes	1 byte	2 bytes
封包頭	晶片地址	封包標誌	封包長度	指令碼	校驗和
0xEF01	xxxx	01H	03H	01H	05H

- 回應封包格式

2 bytes	4 bytes	1 byte	2 bytes	1 byte	2 bytes
封包頭	晶片地址	封包標誌	封包長度	確認碼	校驗和
0xEF01	xxxx	07H	03H	xxH	sum

- 備註：
  - 確認碼=00H 表示錄入成功
  - 確認碼=01H 表示收包有錯
  - 確認碼=02H 表示感測器上無手指
  - 確認碼=03H 表示錄入不成功

- sum 指校驗和

這裡的"指令封包"就是程式需要傳送給設備的資料；而"回應封包"就是設備會回傳給程式的資料。

在指令中要注意的是指令的數值和大小。

- 以封包長度為例：2 bytes 與 03H，這代表實際的資料是 0x0003
- 以指令碼為例：1 byte 與 01H，這代表實際的資料是 0x01

其中的晶片地址預設是 0xFFFF FFFF。

結合這些資訊就可以得到 PS_GetImage 的指令封包(分隔只是方便觀看，實際上資料沒有)：

0xEF01 FFFF FFFF 01 0003 01 0005

寫在 python 要寫成

```
command = b'\xEF\x01\xFF\xFF\xFF\xFF\x01\x00\x03\x01\x00\x05'
```

或是先寫成 16 進制的字串，再透過 `bytearray.fromhex` 來轉換

```
a = 'EF01FFFFFFFF0100030010005'
command = bytearray.fromhex(a)
```

最後透過 `dev.write(command)` 來發送指令，`dev.read(12)` 來接收指令 (其中的 12 是因為回應封包長度是 12)。

```
dev.write(command)
dev.read(12)
# 得到回應 b'\xef\x01\xff\xff\xff\xff\x07\x00\x03\x02\x00\x0c'
```

回應封包也會根據指令集上的格式。

`b'\xef\x01\xff\xff\xff\xff\x07\x00\x03\x02\x00\x0c'`

其中的第 10 個 byte `\x02` 代表感應器上無手指。

看需要甚麼功能，根據指令書上的格式傳送、接收封包就可以控制硬體。

程式碼整合

筆者根據指令書完成了

- (1) PS_GetImage : 控制裝置擷取影像
- (10) PS_UpImage : 從裝置獲得指紋圖像
- (14) PS_WriteReg : 寫入晶片的暫存器 (斷電也不會重置的設定)
- (15) PS_ReadSysPara : 讀取晶片的系統參數

檔案: demo.py

`.py` 中 `AS608` 是撰寫的 `class`，可以在其他 `.py` 中透過 `from demo import AS608` 來使用。

使用範例:

```
if __name__ == '__main__':
    # 設定鮑率
    COM = "COM5"
    baudrate = 9600

    dev = AS608( COM , baudrate*12 )

    while True:
        if dev.PS_GetImage(): # 感測到手指並擷取圖像
            time.sleep(1)
            dev.PS_GetImage() # 等待 1 秒再次擷取 (因為可能感測到時並沒有把手指放好)
            img_data = dev.PS_UpImage() # 將擷取影像存入電腦
    del dev
```

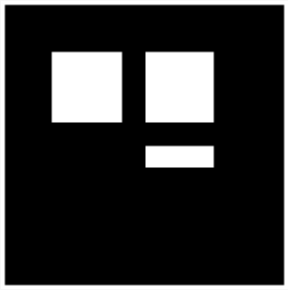
透過使用 PS_GetImage 和 PS_UpImage 就獲得實際的指紋圖像(256*288 pixels)來做後續處理。

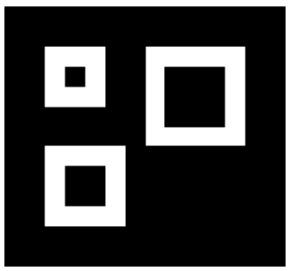
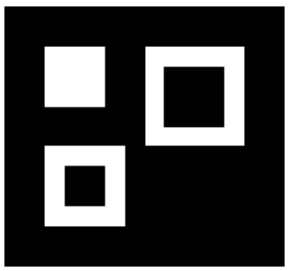
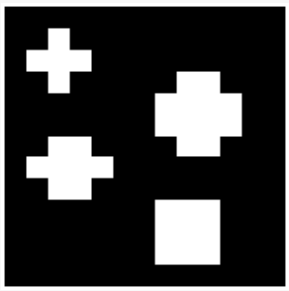
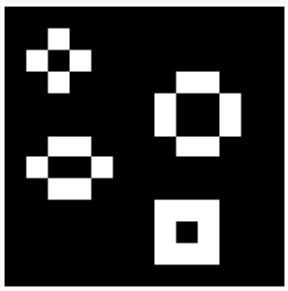
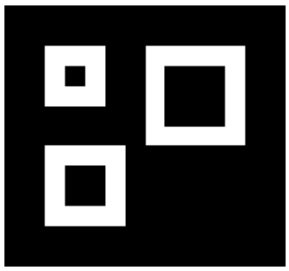
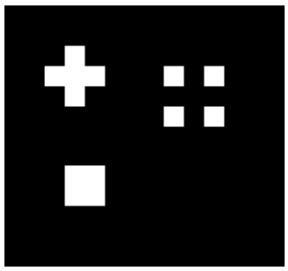
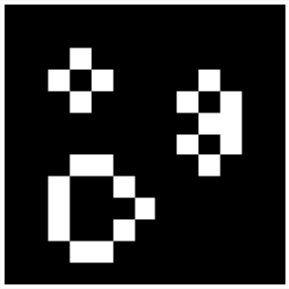
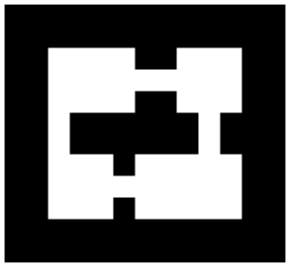
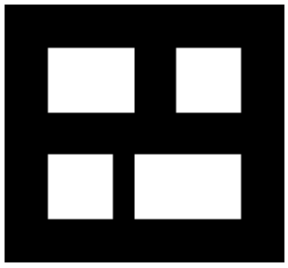
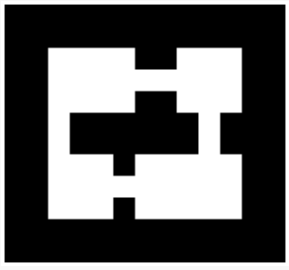
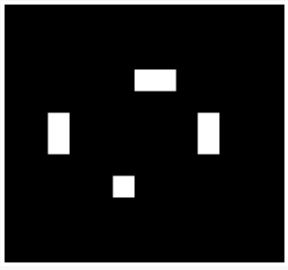
相關的論文

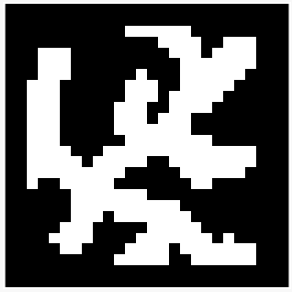
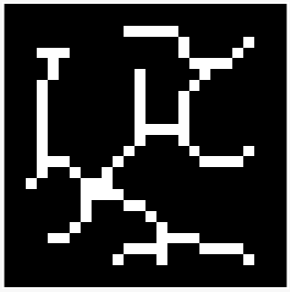
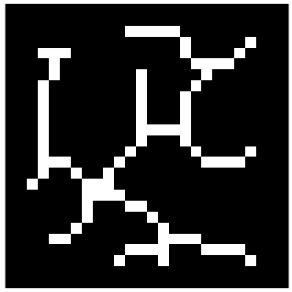
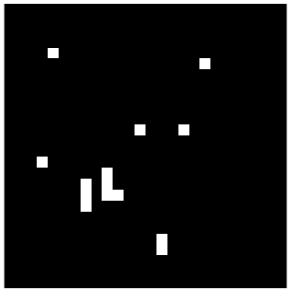
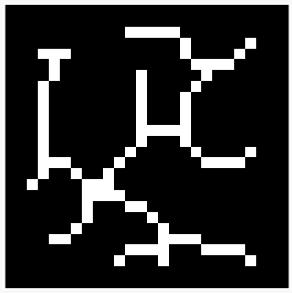
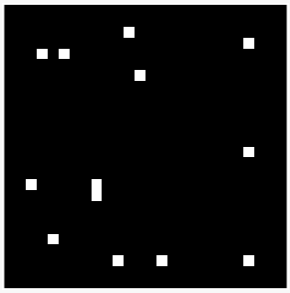
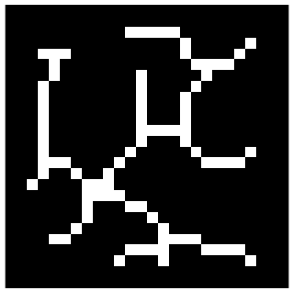
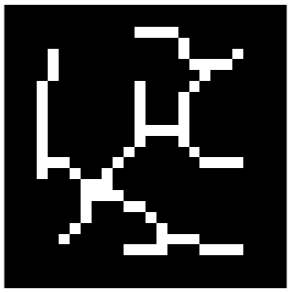
- 2004比對指紋影像之人機介面.pdf
- 2010具重疊指紋及汙點雜訊之指紋圖像辨識方法探討.pdf

相關的 `matlab` 形態學函數。

```
ae = bwmorph( img , operation )
ae = bwmorph( img , operation , n )
```

operation	description	處理前	處理後
'dilate'	膨脹		
'erode'	侵蝕		
'close'	閉合		
'bothat'	底帽轉換 (閉合減去原圖)		
'bridge'	連接兩個分隔 (1px) 的連通區域		
'clean'	去除孤立點		

operation	description	處理前	處理後
'fill'	將被 1 包圍的 0 補成 1		
'remove'	移除		
'majority'	補成周圍區域佔多數的數值。(中間值遮罩)		
'diag'	把 4 連通補成 8 連通		
'open'	斷開		
'tophat'	頂帽轉換 (原圖減去斷開)		

operation	description	處理前	處理後
'hbreak'	去除 H 型連接		
'skel'	骨架化 設定參數 n 決定程度 設定 inf 變成 1px 的骨架		
'branchpoints'	找到圖片骨架化(1px 寬)後的分支點		
'endpoints'	找到圖片骨架化的末端		
'spur'	去除末端的小分岔		
'thicken'	加粗 和膨脹類似，但不會讓線連在一起 設定參數 n 決定加粗程度		

operation	description	處理前	處理後
'thin'	細化 和侵蝕類似，但不會讓線消失 設定參數 n 決定細化程度		

可以參考 `help bwmorph`

指紋分類

指紋奇異點是指指紋中具有特別紋路的區域，例如說螺旋區域 (core) 與三角區域 (delta)，一旦我們可以抓出這些區域，就能依此作為兩張指紋圖片比對的定位點。

在 1823 年，德國 Breslau 大學的教授 Johannes E. Purkinje 將指紋進行分類 (分為 9 類)；之後的分類大多以此為基礎再發展下去。

下圖為 FBI 所用的指紋分類系統，分為三大類八小類

- 弧形 (Arch)：簡單弧型 (Plain Arch) 、帳篷弧型 (Tented Arch)
- 箕形 (Loop)：正箕型 (Ulnar Loop)、反箕型 (Radial Loop)
- 斗形 (Whorl)：雙箕型 (Double Loop Whorl)、簡單斗型 (Plain Whorl)、囊型 (Central Pocket Loop Whorl)、雜型 (Accidental Whorl)

FINGERPRINT PATTERNS AND CLASSIFICATIONS

Plain Arch

In plain arches the ridges enter on one side of the impression and flow or tend to flow out the other side with a rise or wave in the center.

Tented Arch

Tented arches are similar to plain arches with the exception that the ridges in the center form a definite angle; or one or more ridges at the center form an upthrust; or they approach the loop type of pattern, possessing two of the basic characteristics of the loop, but lacking the third.

Ulnar Loop

Ulnar loops are those types of pattern in which the loops flow in the direction of the little fingers.

The above pattern would be an ulnar pattern if on the right hand, and a radial pattern if on the left hand. The above pattern is also sometimes called a right slant loop, regardless of which hand it appears on.

Radial Loop

Radial loops are those types of pattern in which the loops flow in the direction of the thumbs.

The above pattern would be a radial pattern if on the right hand, and an ulnar pattern if on the left hand.

The above pattern is also sometimes called a left slant loop, regardless of which hand it appears on.

Double Loop Whorl

The double loop whorl consists of two separate loop formations, with two separate and distinct sets of shoulders and two deltas.

Plain Whorl

A plain whorl has two deltas and at least one ridge making a complete circuit, which may be spiral, oval, or any variant of the circle. An imaginary line drawn between the two deltas must touch or cross at least one of the recurving ridges within the pattern area.

Central Pocket Loop Whorl

The central pocket loop whorl consists of one or more recurving ridges, or an obstruction at a right angle to the inner line of flow, with two deltas between which an imaginary line would cut or touch no recurving ridge within the pattern area. The inner line of flow of a central pocket loop whorl is determined by drawing an imaginary line between the inner delta and the center of the innermost recurve or looping ridge.

Accidental Whorl

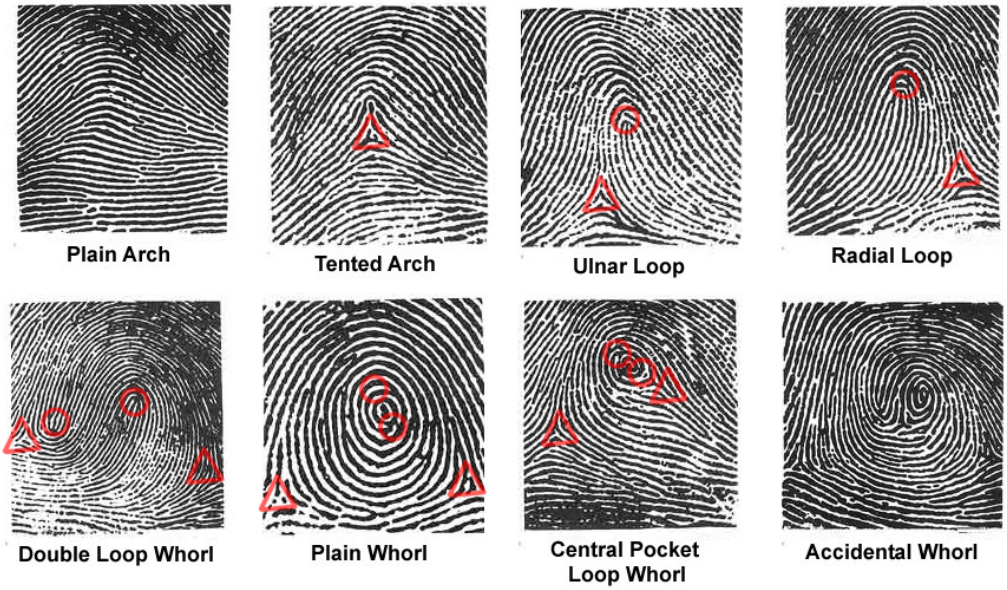
The accidental whorl is a pattern with two or more deltas, and a combination of two or more different types of patterns exclusive of the plain arch. This classification also includes those exceedingly unusual patterns which may not be placed by definition into any other classes.

Above fingerprint images from *The Science of Fingerprints - Classification and Uses*, by the FBI Identification Division, 1957.

- 上述的三大類是根據指紋中的螺旋區域 (core) 與三角區域來分 (delta)。

類別	Core	Delta
圖案		

- 以上面的圖案來說 (Central Pocket Loop Whorl 要再查查)



尋找奇異點

介紹完基本的指紋分類後，我們要從圖片中萃取出 core 與 delta。

具體的步驟如下：

1. 讀取圖片
2. 計算梯度場
3. 計算區域梯度
4. 計算區域方向場 (指紋流向場)
5. 計算 poincare index

1. 讀取圖片

這裡使用 9 張經過強化處理後的指紋圖片

讀取圖片

```
%---- load image
%file_name = 'test.png';
img = imread( file_name );
img = im2double(img);
```

2. 計算梯度場

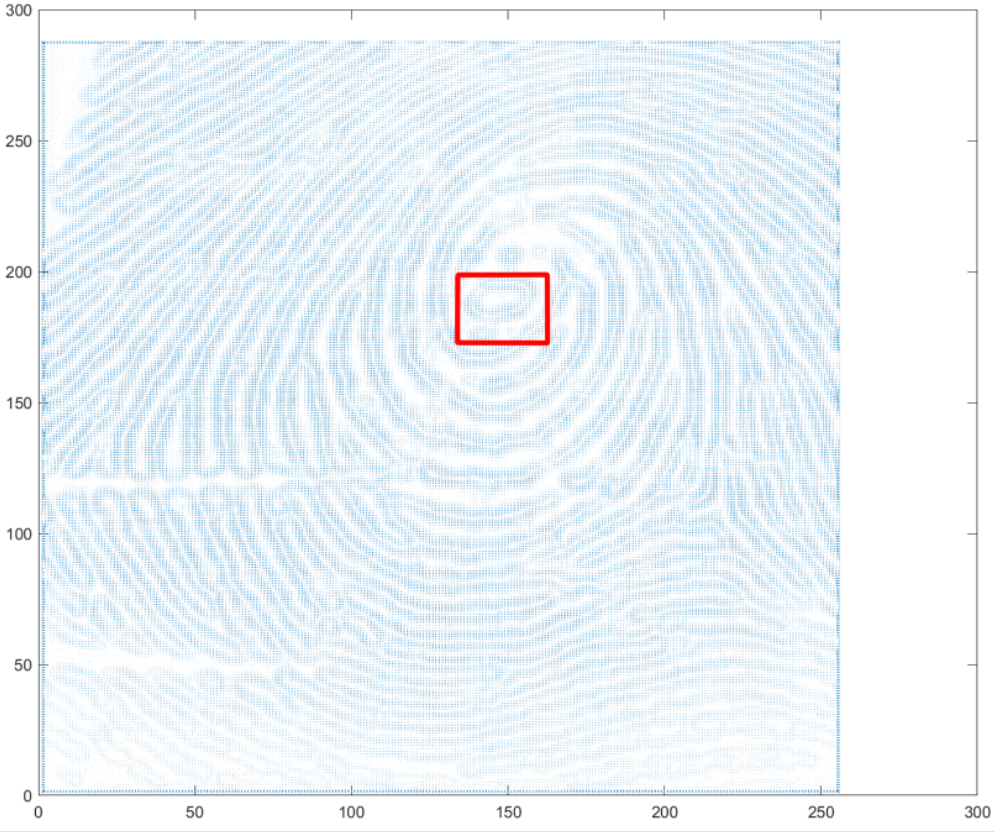
梯度場這裡使用 sobel mask 來計算

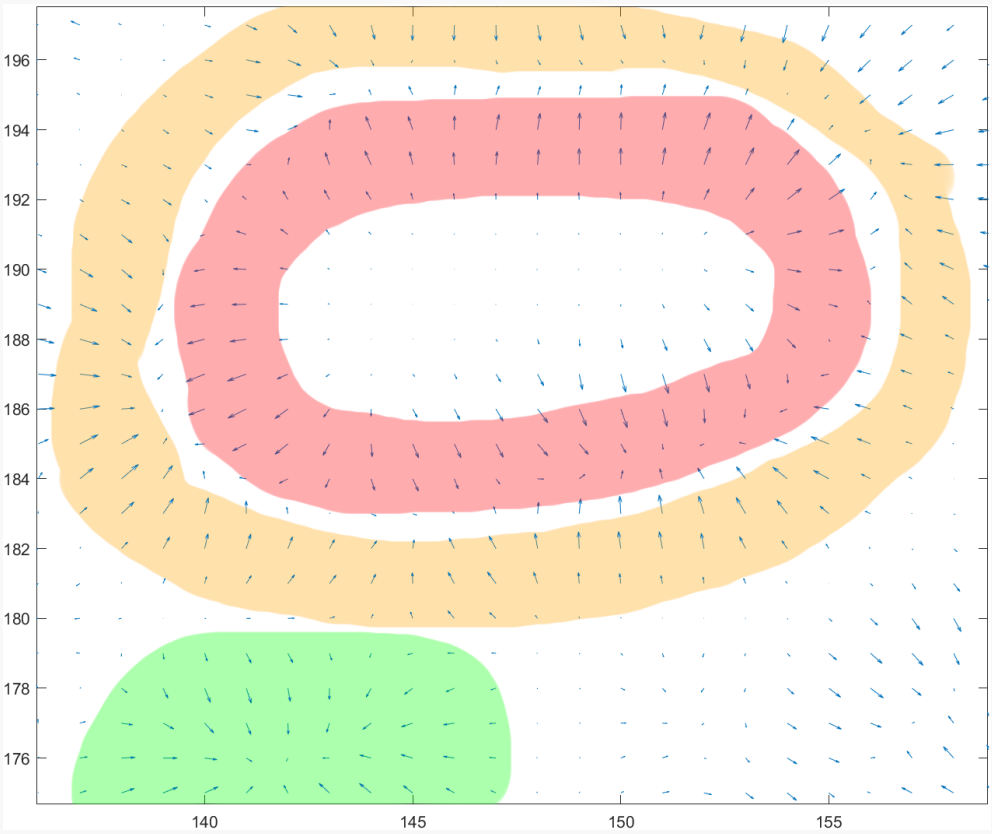
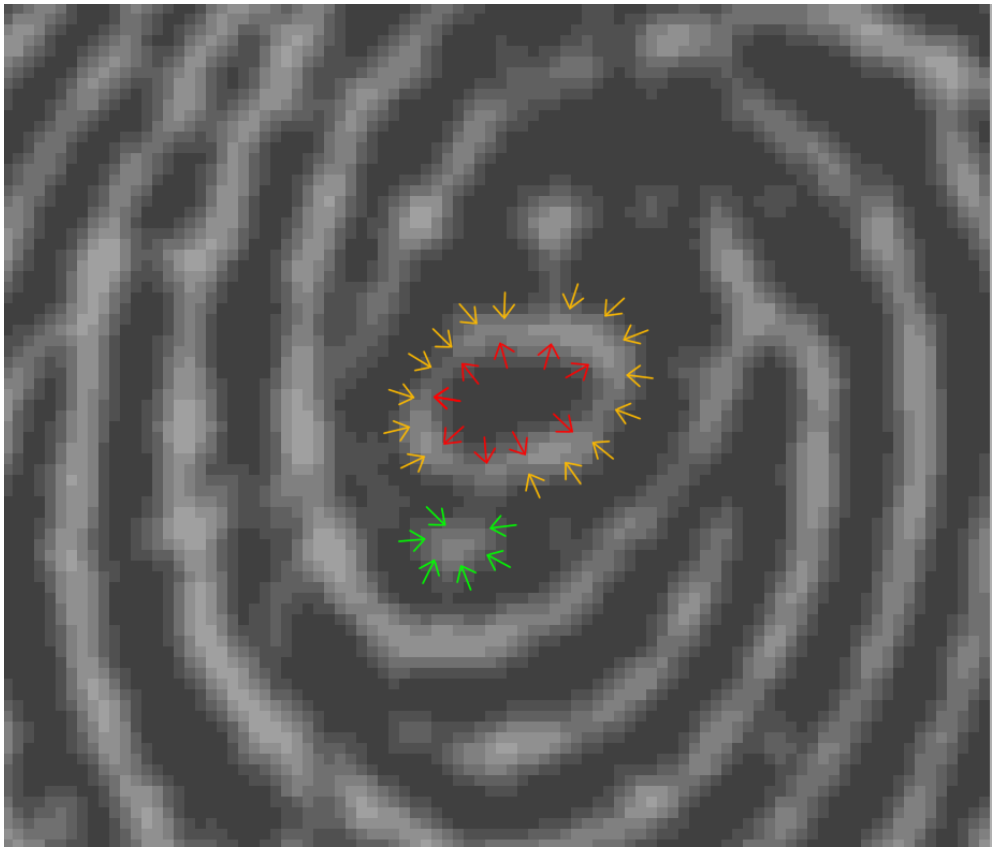
```
%---- compute the gradient field
sobel_y = [ -1 -2 -1 ; 0 0 0 ; 1 2 1 ];
sobel_x = sobel_y';

gx = filter2( sobel_x , img );
gy = filter2( sobel_y , img );
```

以及顯示向量場的程式碼

```
[ h , w ] = size( img );
[X,Y] = meshgrid( 1:w , 1:h );  % 如果改成 h:-1:1 可以讓梯度場和圖片方向一致
figure
quiver( X , Y , gx , gy )
```

計算出來的結果如下：

原圖	
梯度場	 300 250 200 150 100 50 0 0 50 100 150 200 250 300

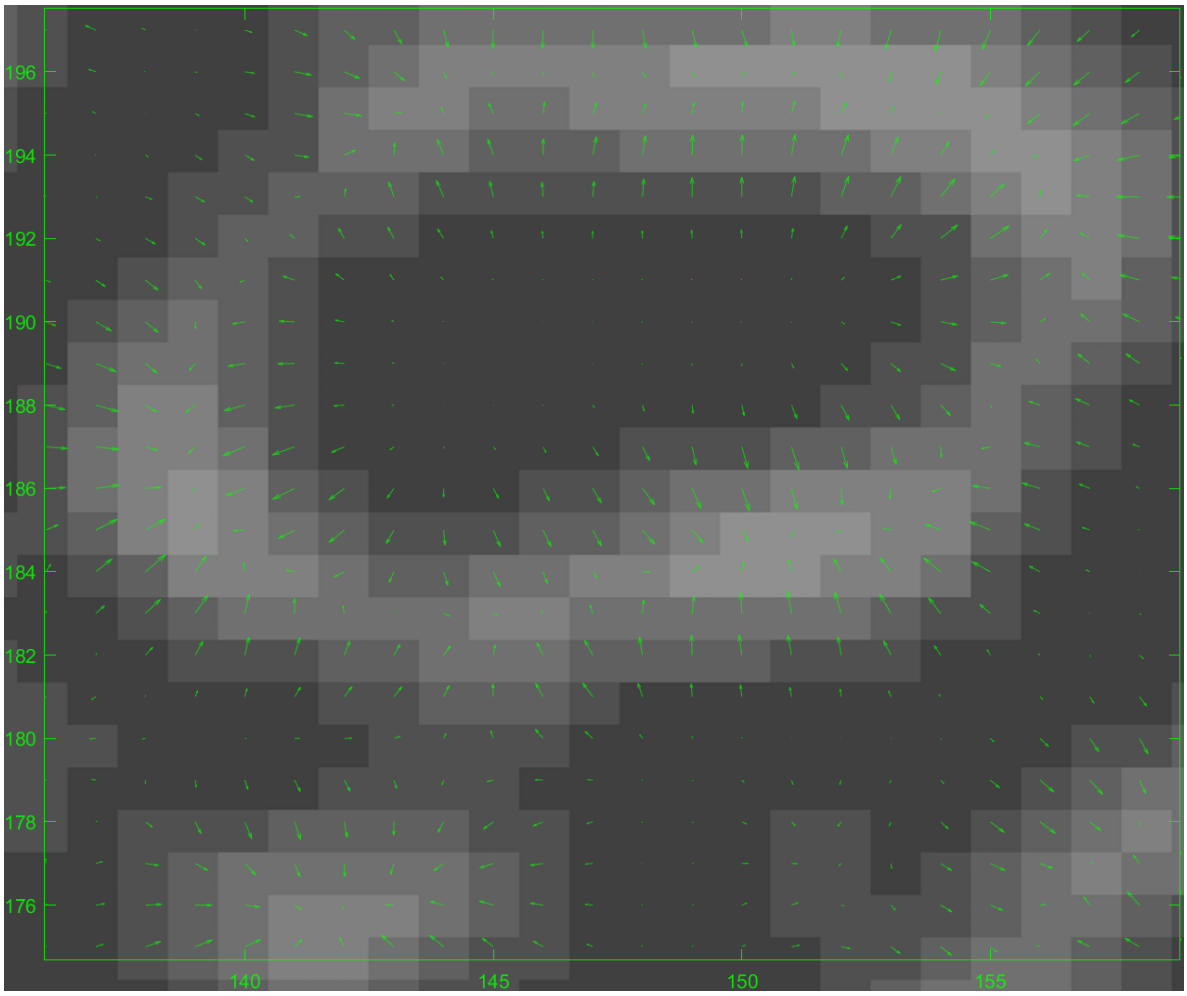
局部放大 (顏色對應下圖)	
對應圖片(已經上下翻轉)	

這裡會觀察到兩件事情：

- 畫梯度場時，梯度場和原圖片是上下顛倒的。(因為梯度場圖的 y 軸是由下往上、圖片則是由上往下)
- 梯度方向會指向山脊(白色)的部分

從「梯度方向指向山脊(白色)」這件事來看，只要將梯度方向轉 90° ，我們就可以獲得紋路的方向。

不過實際上會發現，在梯度場中，很多區塊是沒有箭頭的，也就是說很多區塊是沒有梯度(梯度太小)的。

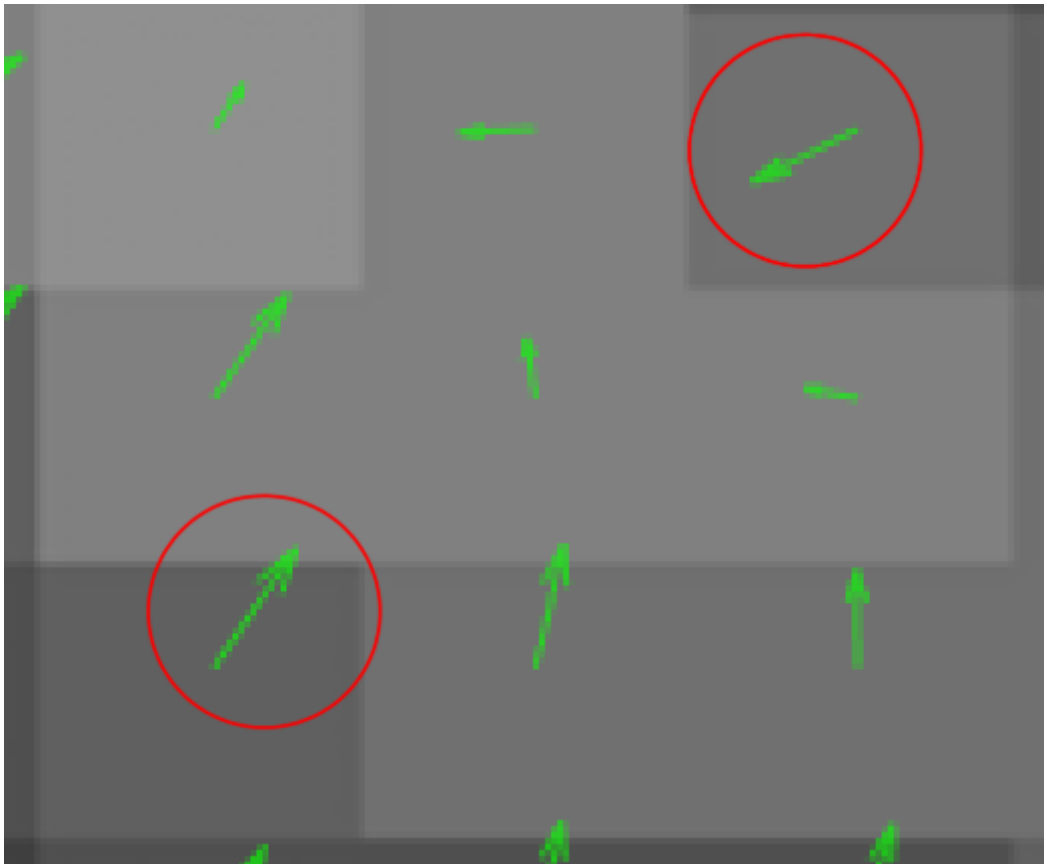


所以實際上在計算時，會使用區域梯度來做為方向場，有兩個用途

- 降低雜訊
- 讓沒有梯度的區塊跟隨附近的梯度。

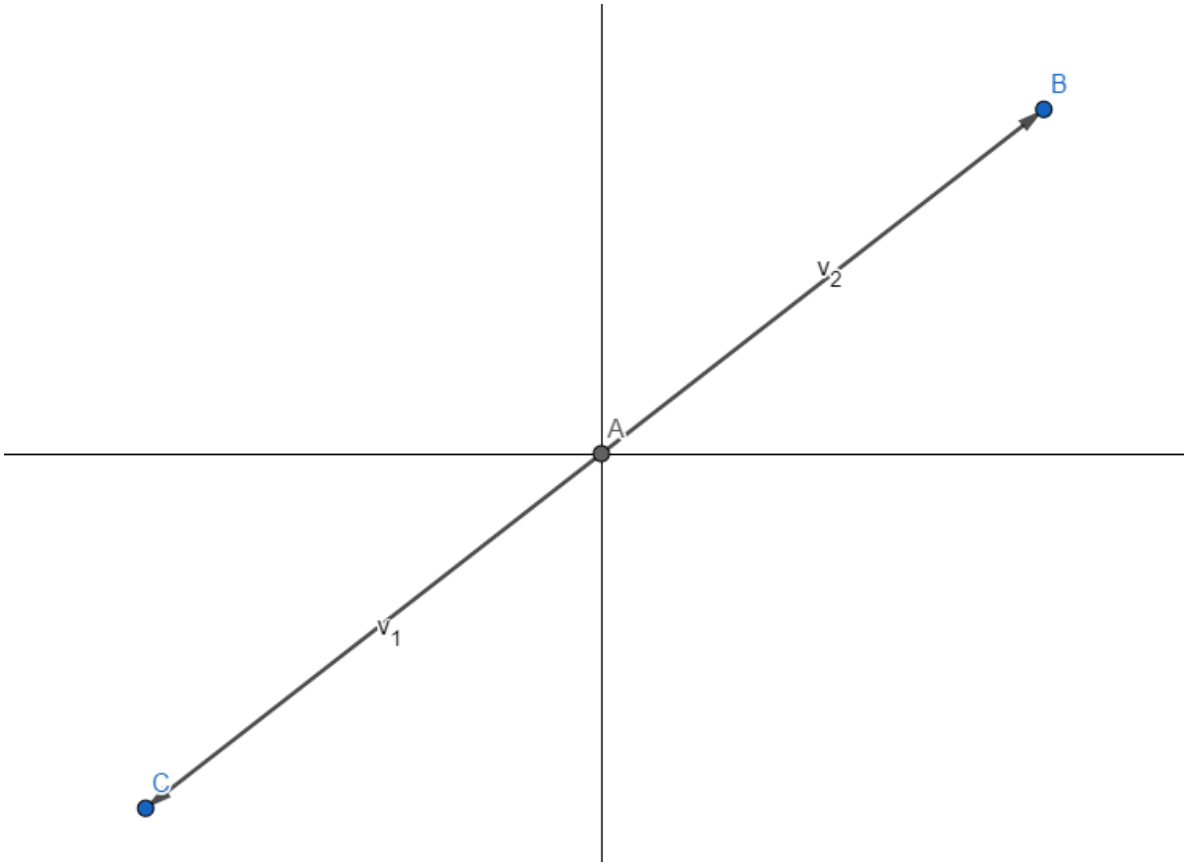
2.1. 計算區域梯度

在計算區域梯度時，有一件事情要注意：「兩個梯度可能會相消」，如下圖的情況。
原本是計算九宮格內的梯度平均，但如果發生下面這種兩個梯度方向相反的情況，那平均值反而會非常接近 0。



我們希望的是這兩個向量不是**相消**而是**相加**起來，這樣再轉 90° 就會是紋路方向。
這裡有一些數學推導

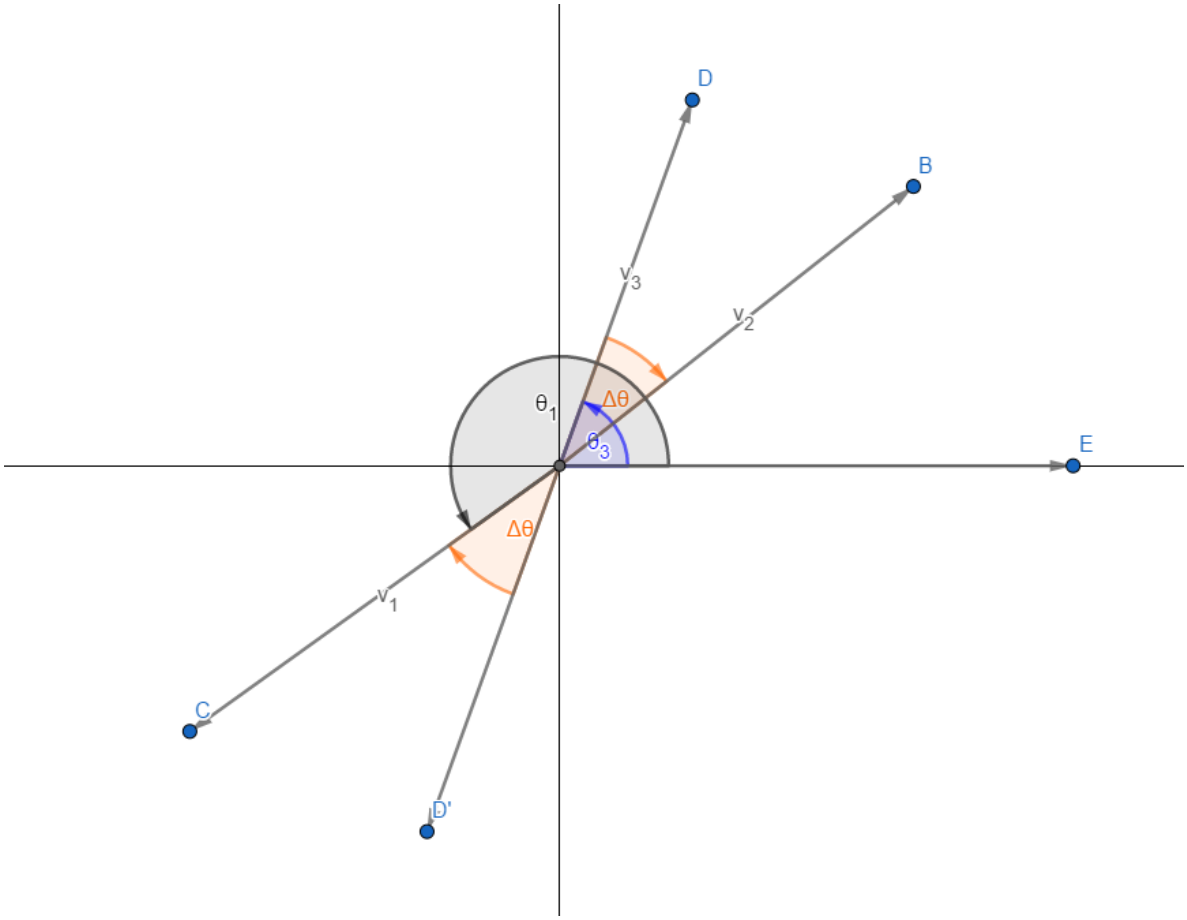
- 我們直接考慮最簡單的情況：



首先有 v_1, v_2，對應長度 l_1, l_2 、角度 θ_1, θ_2，其中 $\theta_1 = \theta_2 + \pi$ (這裡角度單位是**弧度**)

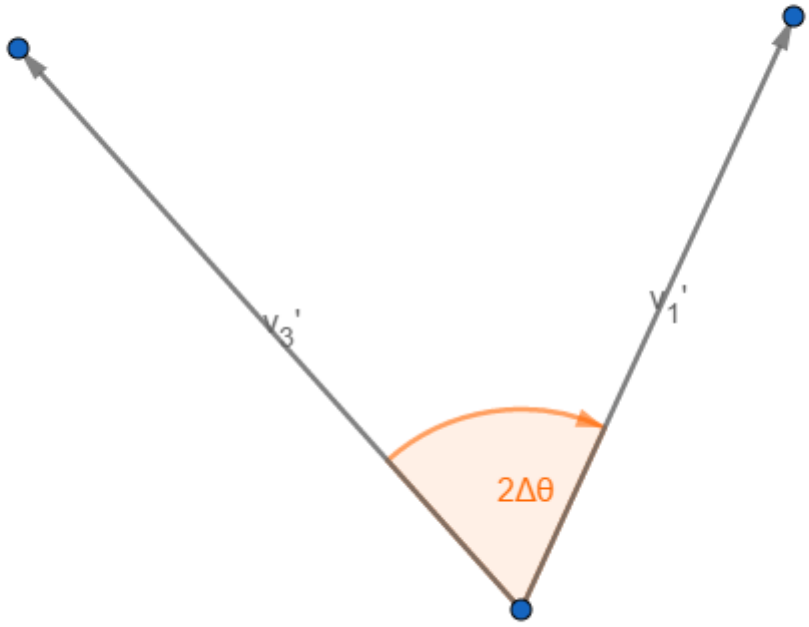
1. 先轉換到極座標：$\begin{cases} v_1 = (l_1, \theta_1) = (l_1, \theta_2 + \pi) \\ v_2 = (l_2, \theta_2) \end{cases}$
2. 接著把「角度加倍再 mod 2π」得到：$\begin{cases} v_1' = (l_1, 2\theta_1) = (l_1, 2\theta_2) \\ v_2' = (l_2, 2\theta_2) \end{cases}$
3. 再直接相加兩個向量得到：$v_1' + v_2' = (l_1 + l_2, 2\theta_2)$
4. 最後再把角度減半得到：$v_s = (l_1 + l_2, \theta_1)$

- 我們用這個方法來將兩個方向相反的向量加在一起
- 再來考慮兩個向量有稍微偏差的情況



首先有 v_1, v_3，對應長度 l_1, l_3 、角度 θ_1, θ_3，其中 $\theta_1 = \theta_3 + \pi + \Delta\theta$ (圖中的 $\Delta\theta$ 也許是 $-\pi/6 = -30^\circ$)

1. 先轉換到極座標：$\begin{cases} v_1 = (l_1, \theta_1) = (l_1, \theta_3 + \pi + \Delta\theta) \\ v_3 = (l_3, \theta_3) \end{cases}$
2. 接著把「角度加倍再 mod 2π」得到：$\begin{cases} v_1' = (l_1, 2\theta_1) = (l_1, 2\theta_3 + 2\Delta\theta) \\ v_3' = (l_3, 2\theta_3) \end{cases}$
3. 最後得到兩個向量 v_1', v_3' 如下



4. 最後再將兩個向量相加，也可以避免掉互相抵消掉的情況。

- 實際上的算法如下
    1. 向量角度加倍
    2. 向量相加
    3. 最後把角度減半

- 向量角度加倍

$$\begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} g_\rho \cos g_\varphi \\ g_\rho \sin g_\varphi \end{bmatrix}$$

$$\begin{bmatrix} g'_x \\ g'_y \end{bmatrix} = \begin{bmatrix} g_\rho^2 \cos 2g_\varphi \\ g_\rho^2 \sin 2g_\varphi \end{bmatrix} = \begin{bmatrix} g_\rho^2(\cos^2 g_\varphi - \sin^2 g_\varphi) \\ g_\rho^2(2 \sin g_\varphi \cos g_\varphi) \end{bmatrix}$$

對應程式碼

```
% double angle for addition
pre_Gx = gx.^2 - gy.^2;
pre_Gy = 2 * gx .* gy;
```

- 向量相加 (這裡用 35*35 的鄰近區塊)

```
w = 35;
Gx = filter2( ones(w) , pre_Gx );
Gy = filter2( ones(w) , pre_Gy );
```

- 計算角度 & 角度減半

為了讓算出來的角度在 $[-\frac{\pi}{2}, \frac{\pi}{2}]$，所以下面分了 case1 (第一、四象限)、case2 (第二象限)、case3 (第三象限)。

```
case1 = Gx >= 0;
case2 = Gx < 0 & Gy >= 0;
case3 = Gx < 0 & Gy < 0;

pre_theta = atan( Gy./Gx );
pre_theta(case2) = pre_theta(case2) + pi;
pre_theta(case3) = pre_theta(case3) - pi;

% compute the region gradient
theta = 0.5 * pre_theta;
```

計算區域方向場 (指紋方向場)

- 計算指紋方向場時，直接把梯度方向場轉 90° 就完成了。

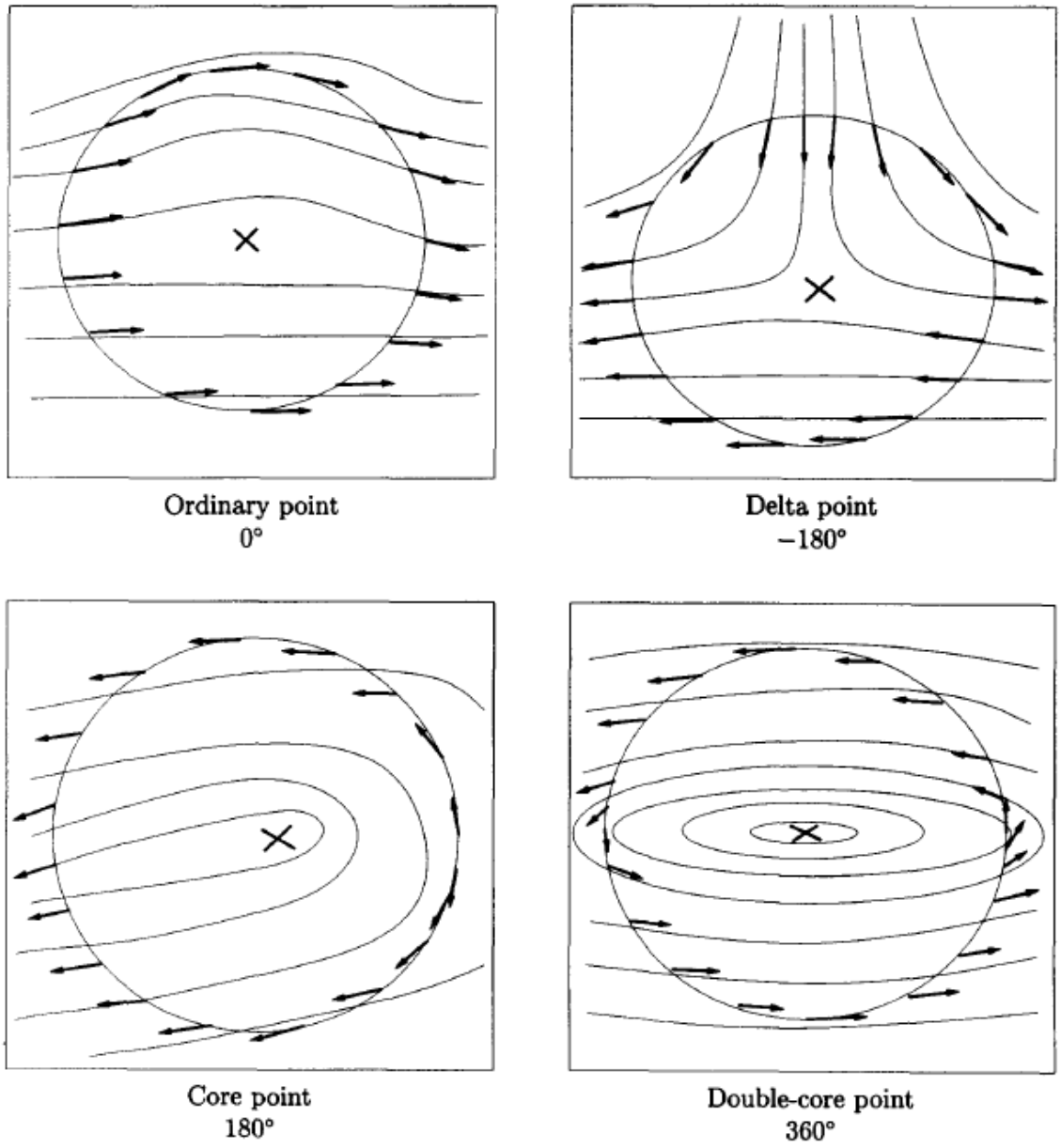
  為了保持角度在 $[-\frac{\pi}{2}, \frac{\pi}{2}]$，所以也分 case1, case2。

```
% compute the region direction
t_case1 = theta <= 0;
t_case2 = ~t_case1;
theta( t_case1 ) = theta( t_case1 ) + 0.5*pi;
theta( t_case2 ) = theta( t_case2 ) - 0.5*pi;
```

計算 poincare index

最後要找出 core 和 delta，這裡使用的方法是計算 poincare index。

poincare index 可以用來計算一個區塊內向量方向的變化，如下面的圖片，他會計算一個**區塊邊界**上的方向。
在影像處理上，區塊邊界就會變成在 3*3 區塊的外圈或是 5*5 區塊的外圈。



圖片: FINGERPRINT CLASSIFICATION, 1995 , KALLE KARU and ANIL K. JAIN.

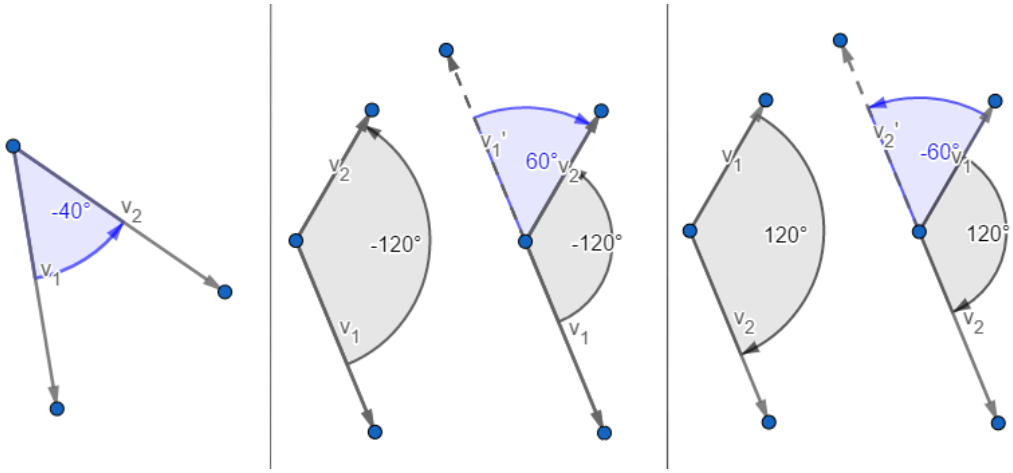
poincare index 的實際計算方法：

順時針的挑選每個邊界上的向量，計算兩個向量的角度差，在限制角度在 $[-\frac{\pi}{2}, \frac{\pi}{2}]$ 下，有 3 種 case：

case 1：兩個角度差 $\leq \frac{\pi}{2}$

case 2：兩個角度差 $< -\frac{\pi}{2}$, 取 $\pi + \Delta\theta$

case 3：兩個角度差 $> \frac{\pi}{2}$, 取 $-\pi + \Delta\theta$

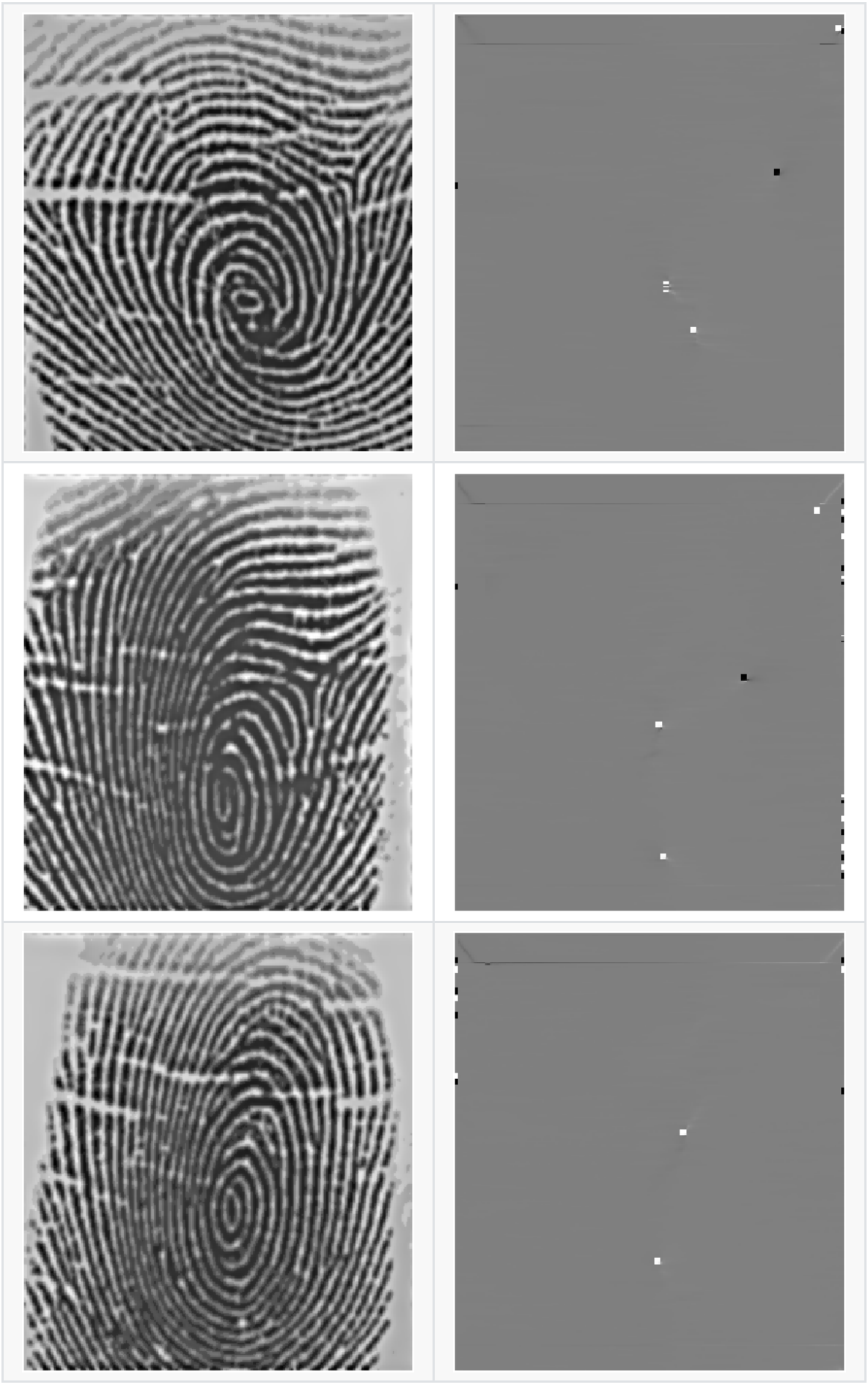


在 Matlab 中計算 poincare index

```
% compute the index
I = colfilt( theta , [5,5] , 'sliding' , @angle_sum );

% show image
figure()
imshow( (I+pi)/(2*pi)  )
```

```
function s = angle_sum( M )

[r c] = size(M);
s = zeros(1, c);
# 順時針方向
pos = [ 1:5:21 , 22:25 , 20:-5:5 , 4:-1:2 ];
len = numel(pos)-1;

for i = 1:c
    a = M(:,i);

    # 分 case 計算
    for j=1:len
        diff = a(pos(j+1)) - a(pos(j));
        if abs(diff) <= pi/2
            ;
        elseif diff < -pi/2
            diff = pi + diff ;
        elseif diff > pi/2
            diff = -pi + diff;
        end
        s(i) = s(i) + diff;
    end

end

end
```

在指紋方向場上計算 poincare index 得到，白點是 π，黑點是 $-\pi$，灰色部分是 0 (經過 `(I+pi)/(2*pi)` 轉換)



參考資料：

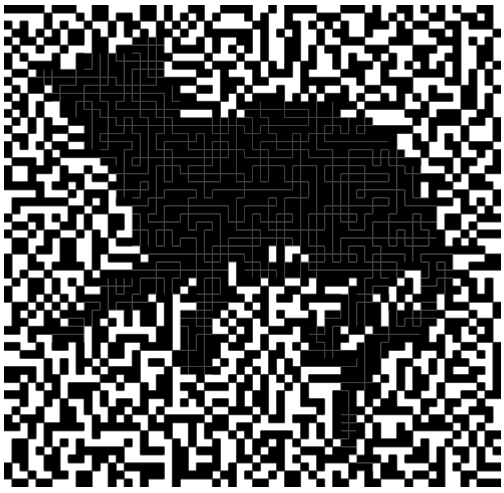
- [1995] FINGERPRINT CLASSIFICATION.pdf
- [2002] Systematic Methods for the Computation of the Directional Fields and Singular Points of Fingerprints.pdf

視覺密碼介紹

傳統密碼學是加密者透過金鑰，再使用特定演算法來將資訊加密；而解密者也需要一把金鑰，並使用特定演算法來解密。

而視覺密碼的加密則是生成許多張圖片，透過"疊合圖片"來解密。

下圖就是一個例子，可以操作看看下方的左右移、疊合、分開的按鈕。



往左移 往右移 疊合 分開

密碼生成方法

1994 年 Noar 和 Shamir 提出使用"像素擴展"的方法來產生視覺密碼。

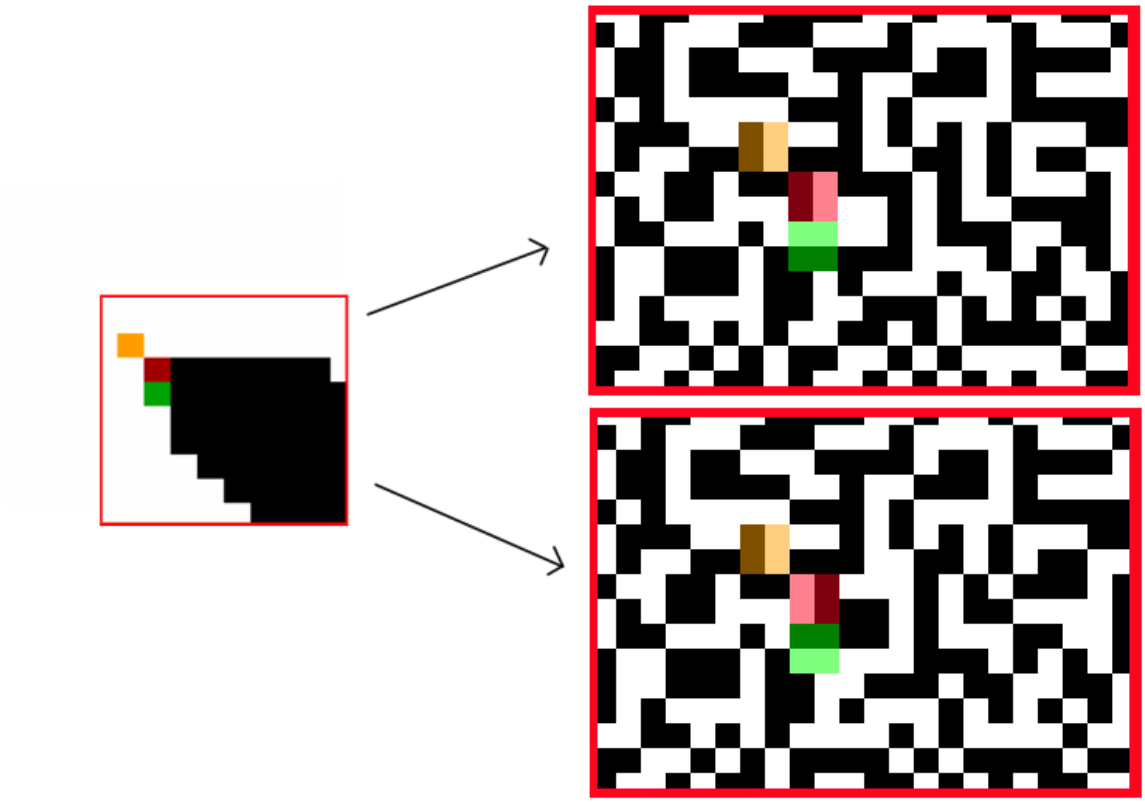
具體做法如下：

1. 產生兩張長寬皆為原圖兩倍的空白子圖 A 與空白子圖 B
2. 針對每一個像素去判斷顏色，
   如果是白色，從白色組的遮罩隨機挑選 1 組出來，作為子圖 A 與子圖 B 的像素；
   如果是黑色，從黑色組的遮罩隨機挑選 1 組出來，作為子圖 A 與子圖 B 的像素。

原圖	■						□					
子圖 A	◨	◧	◪	◩	◪	◧	◨	◧	◪	◩	◪	◧
子圖 B	◧	◨	◩	◪	◧	◪	◨	◧	◪	◩	◪	◧
疊合結果			■				◨	◧	◪	◩	◪	◧

黑色像素就全黑，白色像素就變成半黑半白(遠看就像是灰色)。

把處理過程局部放大來看：(橘色是白點；紅、綠色是黑點)



具體的 `matlab` 程式碼 crypo_binary.m (在處理過程中要**特別注意**存檔要存成 `.png` 來避免 `jpg` 的壓縮模糊)

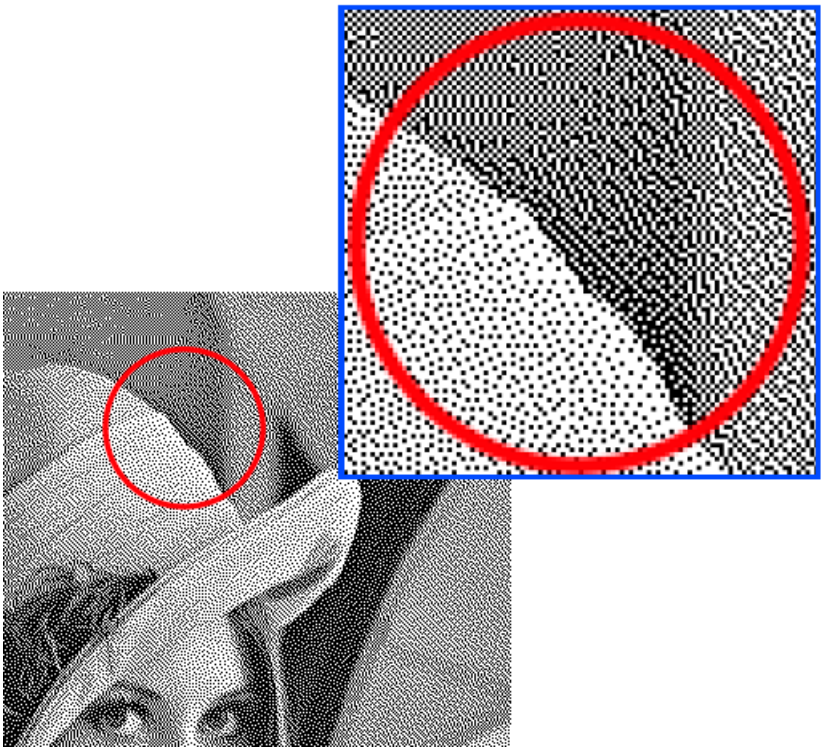
```
close all;clear all;
% 讀入圖片
img = imread('cat4.png');
img = imbinarize(img, 'adaptive'); % 二值化

[h,w] = size(img);

% 生成空白子圖
pA = zeros( 2*h , 2*w );
pB = zeros( 2*h , 2*w );

% 定義 mask
white = zeros( 12 , 2 , 2 );
white(1,:,:) = [ 1 0 ; 1 0 ];
white(2,:,:) = [ 0 1 ; 0 1 ];
white(3,:,:) = [ 1 1 ; 0 0 ];
white(4,:,:) = [ 0 0 ; 1 1 ];
white(5,:,:) = [ 1 0 ; 0 1 ];
white(6,:,:) = [ 0 1 ; 1 0 ];
for i=1:6
    white(i+6,:,:) = white(i,:,:);
end

black = zeros( 12 , 2 , 2 );
black(1,:,:) = [ 1 0 ; 1 0 ];
black(2,:,:) = [ 0 1 ; 0 1 ];
black(3,:,:) = [ 1 1 ; 0 0 ];
black(4,:,:) = [ 0 0 ; 1 1 ];
black(5,:,:) = [ 1 0 ; 0 1 ];
black(6,:,:) = [ 0 1 ; 1 0 ];

```

```
black(7,:,:) = black(2,:,:);
black(8,:,:) = black(1,:,:);
black(9,:,:) = black(4,:,:);
black(10,:,:) = black(3,:,:);
black(11,:,:) = black(6,:,:);
black(12,:,:) = black(5,:,:);

% 迭代每個像素
for y=1:h
    for x=1:w
        % 隨機挑一組來放
        s = randi(6);

        yy = (y-1)*2 + 1;
        xx = (x-1)*2 + 1;

        if img(y,x) == 1 % 白色情況
            pA( yy:yy+1 , xx:xx+1 ) = white(s,:,:);
            pB( yy:yy+1 , xx:xx+1 ) = white(s+6,:,:);
        else % 黑色情況
            pA( yy:yy+1 , xx:xx+1 ) = black(s,:,:);
            pB( yy:yy+1 , xx:xx+1 ) = black(s+6,:,:);
        end
    end
end

imwrite(pA,'cat4A.png');
imwrite(pB,'cat4B.png');
```

應用於灰階圖

因為上面方法是使用於二值圖上，所以要應用在灰階圖還需要多一步處理。

將灰階圖轉成二值圖的技術稱做**半色調化**。



這裡介紹最簡單的誤差擴散法。

誤差擴散法

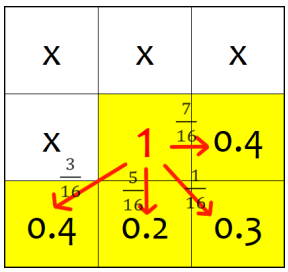
誤差擴散法是將圖片的數值重設為 0 和 1 的方法。演算法如下：

1. 決定閥值 T
2. 如果像素值大於 T，就將數值重設成 1；否則設為 0
3. 將重設前後的誤差以特定比例加 (或減) 到周圍的像素上
4. 換到下一個像素，回到步驟 2

從圖像上來說是下面這樣 (注意黃色區塊的數字變化)

原圖	(閥值 < 0.5 的情況) 把增加的量由附近的像素負擔	(閥值 > 0.5 的情況) 把減少的量由加給附近的像素
0.5 0.8 0.3 0.8 0.1 0.4 0.4 0.2 0.3	1 → 0.5 0.3 0.6 0 0.4 0.4 0.2 0.3	0 → 1 0.3 0.9 0.3 0.4 0.4 0.2 0.3

在實作中我是使用 Floyd–Steinberg 提出的權重：$\frac{7}{16}, \frac{5}{16}, \frac{3}{16}, \frac{1}{16}$



在碰到邊界時，我是採取**不傳那個方向**的方式。(當然也可以碰到邊界時就改變權重，讓整體數值不會有遺失)

誤差擴散法實作範例

原圖	誤差擴散法
	

`matlab` 程式碼 halftone.m , demo_halftone.m (在處理過程中要**特別注意**存檔要存成 `.png` 來避免 `jpg` 的壓縮模糊)

```matlab
function bimg = halftone(img)
[h,w] = size(img);

bimg = zeros(h,w);
% 用 Ostu's method 取閥值
th = graythresh(img);

for y=1:h
    for x=1:w
        if img(y,x) > th
            bimg(y,x) = 1;
        else
            bimg(y,x) = 0;
        end
        err = img(y,x) - bimg(y,x);

        if x+1<=w
            img(y,x+1) = img(y,x+1) + err*7/16;
            if y+1<=h
                img(y+1,x) = img(y+1,x) + err*1/16;
            end
        end
        if y+1 <= h
            if x-1 >= 1
                img(y+1,x-1) = img(y+1,x-1) + err*3/16;
            end
```

```
            img(y+1,x) = img(y+1,x) + err*5/16;
        end
    end
end

end
```

```
img = imread('lena.png');
img = im2double(img);
gimg = rgb2gray(img);
bimg = halftone(gimg);

imwrite( bimg , 'lena_binary.png' );
```

最後再將二值圖經過視覺密碼的處理。

往左移 往右移 疊合 分開

更進階的視覺密碼

一般視覺密碼會將圖片分拆成兩張"雜亂的"子圖，但有方法可以讓分離的子圖也具有某種圖案。

其中的關鍵在於"色階調整"以及"使用偽裝影像作為誤差擴散的基準"。

這裡的方法使用三張圖，分別為機密影像 C、偽裝影像 A 及 偽裝影像 B。

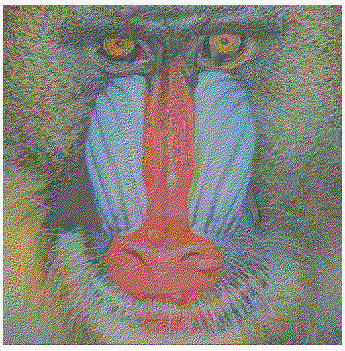
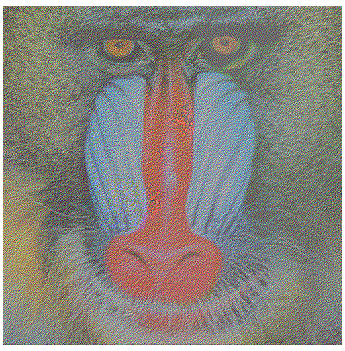
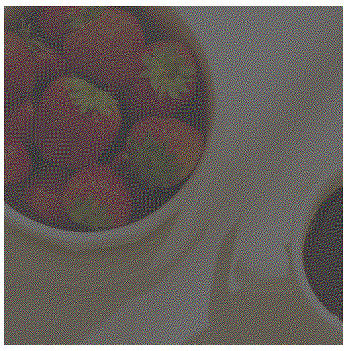
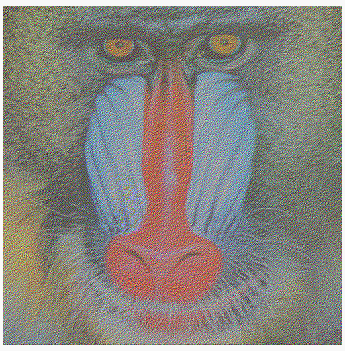
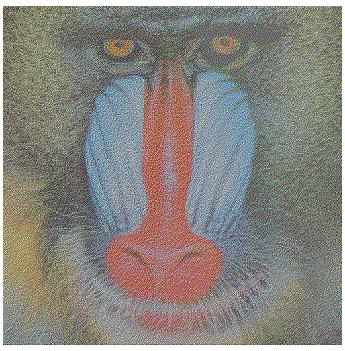
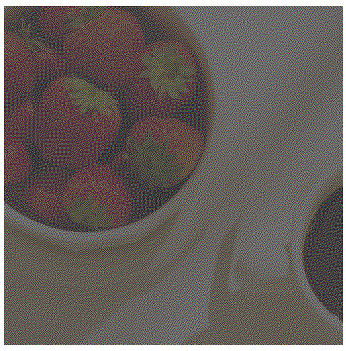
演算法

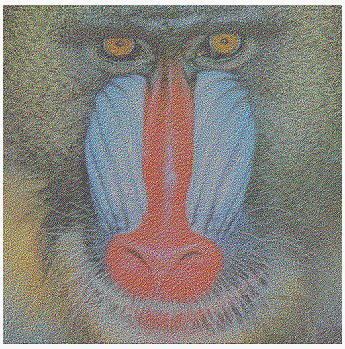
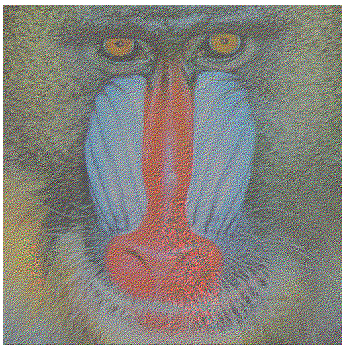
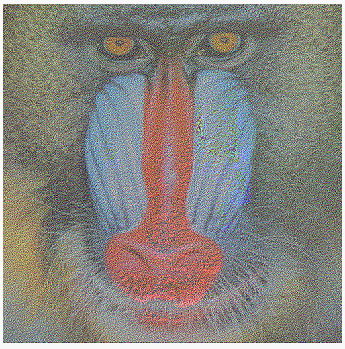
1. 對圖 A, B, C 做色階調整
    - 對於圖 A, B ，將數值 $[0, 1]$ 線性映射到 $[CT_1, CT_2]$ 得到 A', B' ，定義門檻值 $T = (CT_1 + CT_2)/2$。
    - 對於圖 C ， 將數值 $[0, 1]$ 線性映射到 $[ST_1, ST_2]$ ，得到 C' 。
2. 對圖 C' 做一般的誤差擴散，得到 bC。
3. 遞迴每個像素
    - 如果 bC 為白點，則 A', B' 皆設為白點 (**錯誤來源 1)
    - 如果 bC 為黑點
        - 如果 A', B' 皆為白點(大於門檻值 T)，隨機挑一個設為黑點 (**錯誤來源 2)
        - 否則 A', B' 依照是否大於門檻值 T 來設定
    - 在**每次**像素設定完成後，進行誤差擴散。

誤差擴散的**關鍵**程式碼：

這部分的程式碼如下：

```
if bC(y,x) == 1
    %-- 結果必須是白色
    bA(y,x) = 1;
    bB(y,x) = 1;
else
    %-- black
    %-- white -> 1
    %-- black -> 0
    whiteA = A(y,x) > T;
    whiteB = B(y,x) > T;

    if whiteA & whiteB
        %-- 都是白色，但結果必須是黑色
        if rand() > 0.5
            bA(y,x) = 1;
            bB(y,x) = 0;
        else
            bA(y,x) = 0;
            bB(y,x) = 1;
        end
    else
        %-- 本來應該的顏色
        bA(y,x) = whiteA;
        bB(y,x) = whiteB;
    end
end
```

子圖 A	子圖 B	bC
		
		
		
		
		

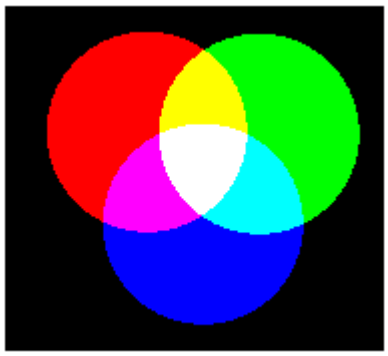
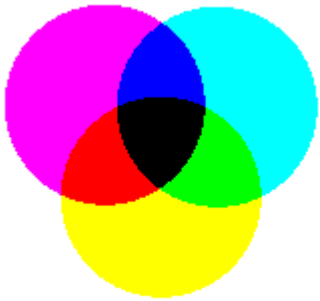
子圖 A	子圖 B	bC
		
		
		

可以 photoshop 或其他影像處理軟體中使用**正片疊底**來驗證。

濾色(加法混合)：一般的 RGB 相加。(模擬色光混合)

正片疊底(減法混合)：先將 RGB 轉成 CMY 後相加，再轉回 RGB。(模擬顏料混合)

左為濾色、右為正片疊底。

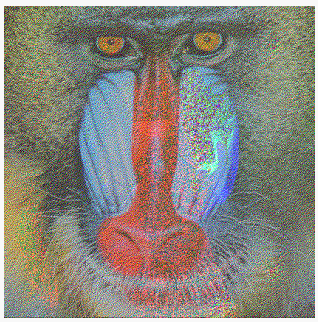
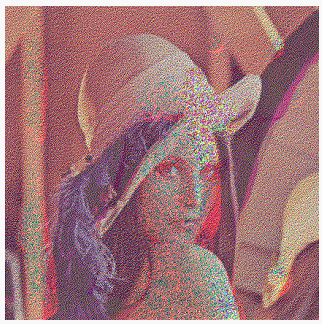



困難點

在處理這個問題時，會碰到的問題是**色階調整**，也就是參數 $[CT_1, CT_2, ST_1, ST_2]$ 設定上的困難。

失敗的**色階調整**會出現兩個問題：

1. 顯示出圖 C 的輪廓
2. 圖 A, B 產生無規則的雜訊區塊

	圖 A	圖 B	圖 C
產生輪廓			
雜訊區塊			

仔細觀察規則

> - 如果 bC 為白點，則 A', B' 皆設為白點 (**錯誤來源 1)
> - 如果 bC 為黑點
> - 如果 A', B' 皆為白點(大於門檻值 T)，隨機挑一個設為黑點 (**錯誤來源 2)
> - 否則 A', B' 依照是否大於門檻值 T 來設定
> - 在**每次**像素設定完成後，進行誤差擴散。

會發現錯誤來源有兩個：

1. 如果 bC 為白點，則 A', B' 皆設為白點
2. 如果 bC 為黑點，且 A', B' 皆為白點(大於門檻值 T)，隨機挑一個設為黑點

產生輪廓來自於錯誤來源 1，可以透過調低 C 的亮度解決。

雜訊區塊來自於錯誤來源 2，可以透過壓縮 $[CT_1, CT_2]$ 或 $[ST_1, ST_2]$ 來解決。

完整程式碼 (使用 gem.m 來產生圖片)： color_halftone.m , halftone.m , gen.m

參考資料

[2004] 像素不擴展之彩色視覺密碼技術.pdf

[2008] 基於視覺密碼學之影像浮水印技術.pdf

[2016] 基於視覺密碼之彩色機密隱藏機制.pdf

CSS 3D 旋轉 (使用 html,css 實作)

實作結果 (旋轉圖片與超連結)

實作程式碼

整個實作分成兩個檔案編寫 `demo.html` 及 `demo.css`

- demo.html
- demo.css

`demo.html` 定義頁面上的結構

```
<!DOCTYPE html>
<html>
    <head>
        <!-- 透過 link 來引入 demo.css -->
        <link rel="stylesheet" href="demo.css">
    </head>
    <body>
        <div class="camera"> <!-- 相機 -->
            <div class="space"> <!-- 立體空間 -->
                <div class="horizontal box1"> <!-- 第一個平面 -->
                    <a target="_blank"
href="https://www.usnews.com/education/best-global-universities/search?
country=taiwan&region=asia"> <!-- 建立超連結 -->
                        <img src="test1.jpg"> <!-- 建立圖片 -->
                    </a>
                </div>
                <div class="horizontal box2"> <!-- 第二個平面 -->
                    <a target="_blank" href="https://www.youtube.com/watch?
v=d3X6JpUc8Ew">
                        <img src="test2.jpg">
                    </a>
                </div>
                <div class="horizontal box3"> <!-- 第三個平面 -->
                    <a target="_blank"
href="http://lit.ncu.edu.tw/shownew.asp?NC_No=2&N_ID=118">
                        <img src="test3.jpg">
                    </a>
                </div>
                <div class="horizontal box4"> <!-- 第四個平面 -->
                    <a target="_blank" href="https://ncu.edu.tw/rd/">
                        <img src="test4.jpg">
                    </a>
                </div>
            </div>
        </div>
    </body>
</html>
```

`demo.css` 定義 `html` 頁面上元素的顯示方式

- 相機設定：`.camera` 會指定到具有 `class="camera"` 的元素

```
.camera{ /* 設定相機 */
```

```
    /* XY平面大小 */
    width:840px;
    height:300px;

    /* 相機相對於 XY 平面的位置 */
    /* center center 相當於  x:50% y:50%*/
    perspective-origin:center center;

    /* 相機距離 XY 平面的距離 */
    /* -moz- 和 -webkit- 是為了在不同瀏覽器也可以正常顯示 */
    perspective:1500px;
    -moz-perspective:1500px;
    -webkit-perspective:1500px;
}
```

- 立體空間設定：`.space` 會指定到具有 `class="space"` 的元素

```
.space{ /* 建立 XY 平面 */
    /* 設定 relative 使得底下的元素會依它為座標起點 */
    position:relative;

    /* XY 平面的大小 */
    width:100%;
    height:100%;

    /* XY 平面標示線 */
    /* border:1px dashed #000; */

    /* 設定以 3D 成像 */
    transform-style:preserve-3d;

    /* 旋轉中心設定 */
    transform-origin:center center -210px;
}
```

- 立體空間中的元素設定
  - `.space div` 會指定到具有 `class="space"` 的元素**底下的** `div` 元素
  - `.space div.horizontal img` 會指定到
    具有 `class="space"` 的元素**底下的**
    具有 `class="horizontal"` 的 `div` 元素**底下的**
    `img` 元素

```css
.space div{
    /* 使立體空間中的物體的位置不會相互干擾 */
    position:absolute;
}

.space div.horizontal img{
    /* 設定圖片大小 */
    width: 100%;
    height: 100%;
}
```

- 設定立體空間中的每個物件的位置
    - 第一個物件

```css
.box1{ /* 正面 */
    transform-origin:center center; /* 預設值，以 左右置中、上下置中 */
    transform:translateX(50%) translateY(50%);
}
```

`transform-origin` 會設定物體移動、旋轉的中心。

`translateX`, `translateY` 會依據物體的大小來決定移動多少。
這裡物體大小是：寬:420px, 高:150px
結合起來是：
將**物體中心** (`transform-origin:center center;`)
向右移動 210px (`translateX(50%)`)
向下移動 75px (`translateY(50%)`)

(demo) 將滑鼠移動虛線框內

    - 第二個物件

```css
.box2{ /* 右側 */
    transform-origin:left center; /* 以左邊界中心為移動、旋轉中心 */
    transform:translateX(150%) translateY(50%) rotateY(90deg);
}
```

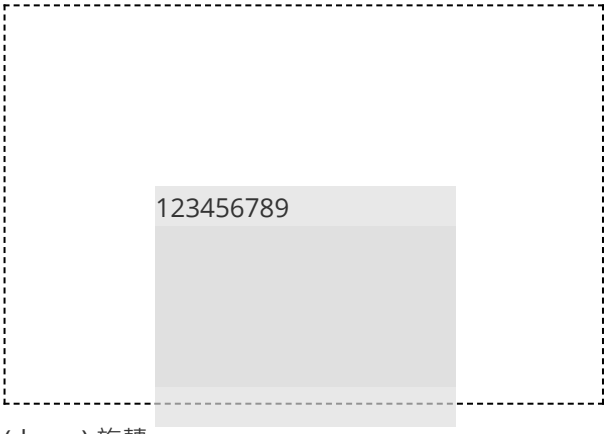
(demo) 移動

(demo) 透視下的旋轉 `rotateY(90deg)`

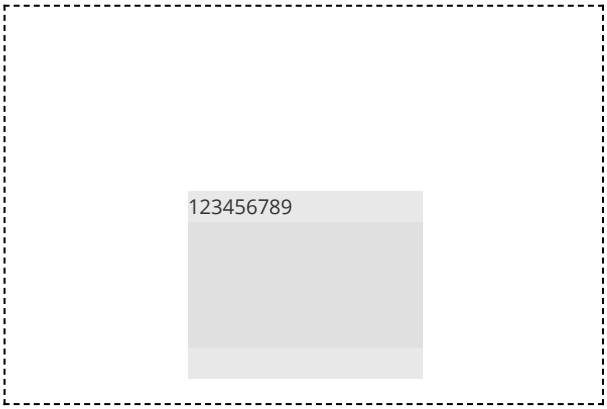
- 第三個物件

```
.box3{ /* 後側 */
    background:rgba(0,255,0,.2);
    transform-origin:center center; /* 預設值，以 左右置中、上下置中 */
    transform:translateX(50%) translateY(50%) translateZ(-420px)
rotateY(180deg);
}
```

(demo) 移動: `translateZ(-420px)`



(demo) 旋轉



- 第四個物件

```
.box4{ /* 左側 */
    background:rgba(255,0,255,.2);
    transform-origin:right center; /* 以右邊界中心為移動、旋轉中心 */
    transform:translateX(-50%) translateY(50%) rotateY(-90deg);
}
```

(demo) 移動

123456789

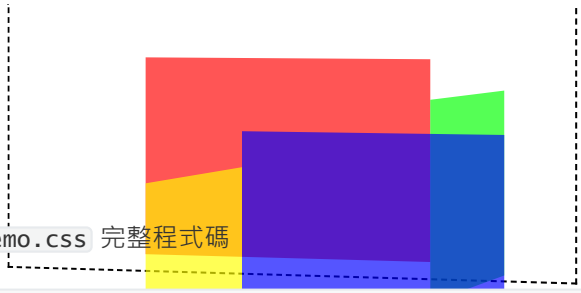
(demo) 透視下旋轉 `rotateY(-90deg)`

123456789

- 設定旋轉動畫

```
.space{
    /* 設定旋轉動畫 */
    animation:s 10s linear infinite;
    -webkit-animation:s 10s linear infinite;
}
/* 設定旋轉動畫 */
@keyframes s{
    0%{
        transform:rotateY(0);
    }
    100%{
        transform:rotateY(-359.9deg);
    }
}
```

  - `animation` 後面依序接 `動畫名稱` `動畫時長` `循環次數`
  - `@keyframe` 則可以設定每個時間百分比，動畫要進展到甚麼程度。(相當於設定關鍵影格)



demo.css 完整程式碼

```css
.camera{ /* 設定相機 */
    /* XY平面大小 */
    width:840px;
    height:300px;

    /* 相機相對於 XY 平面的位置 */
    /* center center 相當於  x:50% y:50%*/
    perspective-origin:center center;

    /* 相機距離 XY 平面的距離 */
    /* -moz- 和 -webkit- 是為了在不同瀏覽器也可以正常顯示 */
    perspective:1500px;
    -moz-perspective:1500px;
    -webkit-perspective:1500px;
}

.space{ /* 建立 XY 平面 */
    /* 設定 relative 使得底下的元素會依它為座標起點 */
    position:relative;

    /* XY 平面的大小 */
    width:100%;
    height:100%;

    /* XY 平面標示線 */
    /* border:1px dashed #000; */

    /* 設定以 3D 成像 */
    transform-style:preserve-3d;

    /* 旋轉中心設定 */
    transform-origin:center center -280px;
}

.space div{
    /* 投影空間中的物體的位置不會相互干擾 */
    position:absolute;
}

.space div.horizontal{ /* 水平方向的 */
    /* 為了圖片所設定的寬高 */
    width: 420px;
    height: 150px;
}

.space div.horizontal img{
    /* 設定圖片大小 */
    width: 100%;
    height: 100%;
```

```
}

.box1{ /* 正面 */
    transform-origin:center center; /* 預設值，以 左右置中、上下置中 */
    transform:translateX(50%) translateY(50%);
}

.box2{ /* 右側 */
    transform-origin:left center; /* 以左邊界中心為移動、旋轉中心 */
    transform:translateX(150%) translateY(50%) rotateY(90deg);
}

.box3{ /* 後側 */
    background:rgba(0,255,0,.2);
    transform-origin:center center; /* 預設值，以 左右置中、上下置中 */
    transform:translateX(50%) translateY(50%) translateZ(-420px)
rotateY(180deg);
}

.box4{ /* 左側 */
    background:rgba(255,0,255,.2);
    transform-origin:right center; /* 以右邊界中心為移動、旋轉中心 */
    transform:translateX(-50%) translateY(50%) rotateY(-90deg);
}

.space{
    /* 設定旋轉動畫 */
    animation:s 10s linear infinite;;
    -webkit-animation:s 10s linear infinite;
}

/* 設定旋轉動畫 */
@keyframes s{
    0%{
        transform:rotateY(0);
    }
    100%{
        transform:rotateY(-359.9deg);
    }
}
```

參考資料

- 玩轉 CSS 3D - 原理篇
- 玩轉 CSS 3D - 正四面體與正六面體

開發環境資訊

- 網站使用 laradock image 為基底，僅使用其中的 nginx 及 workspace
- 由使用者自行將 markdown 轉換為 html ，再透過 laravel 的模板系統套用側邊欄。
    - 實現即時修改，不用每次更新都要重新運行 gitbook
    - 可用的語法不會依賴於 gitbook
    - 可以自由的使用 JS 技術，不用擔心衝突。(每個頁面都是獨立的)
    - 不用架設網頁環境即可開始撰寫文件。

開發環境建立

1. 從 github clone 下來後，執行 `git submodule update --init --recursive` 重建 `laradock` 。
2. 到 `tool` 底下將 6 個檔案放到正確位置後，執行 `run.bat` 建立 docker container。
3. 到 volume 底下，執行 `enter.bat` 進入 workspace container
4. 進入 contanier 後，執行 `composer install` 及 `npm install` 即可完成建置。

```
imweb
    |---- tool
            |---- .env              (laravel 的設定檔，放 volume )
            |---- .envld            (laradock 的設定檔，放 laradock，要改名成 .env )
            |---- default.conf      (nginx 設定，放 lardock/nginx/sites )
            |---- key               (nginx 密碼，放 lardock/nginx/sites )
            |---- run.bat           (啟動網頁，放 laradock)
            |---- stop.bat          (停止網頁，放 laradock)
```

- 目錄結構

```
imweb
    |---- .git
    |---- .gitignore
    |---- .gitmodules
    |---- laradock
    |       |---- nginx
    |               |---- sites
    |                       |---- default.conf    (nginx 設定，用來設定密碼)
    |                       |---- key             (nginx 密碼)
    |---- volume
    |       |---- .env              (laravel 的環境設定檔)
    |       |---- public
    |       |       |---- project   (教學文件-專題)
    |       |       |---- dev_env   (開發環境設定)
    |       |       |---- topic     (教學文件-單一章節)
    |       |       |---- list      (網頁目錄)
    |       |
    |       |---- 其餘文件...
    |
    |---- tool
```

```
21              |---- .env              (laravel 的設定檔，放 volume )
22              |---- .envld            (laradock 的設定檔，放 laradock )
23              |---- default.conf      (nginx 設定，放 lardock/nginx/sites )
24              |---- key               (nginx 密碼，放 lardock/nginx/sites )
25              |---- run.bat           (啟動網頁，放 laradock)
26              |---- stop.bat          (停止網頁，放 laradock)
27              |---- rebuild.bat       (強制重建容器，放 laradock)
28              |---- enter.bat         (進入容器，放 volume)
29              |---- watch.sh          (生成 css，放 volume)
```

進階開發環境設定

1. 在 `docker-compose.yml` 中可以設定 port，預設使用 80 port。 `NGINX_HOST_HTTP_PORT=80`
2. 在 `docker-compose.yml` 中可以設定 docker container 的名稱。
3. 修改 `docker-compose.yml` 後，需要執行 `run.bat` 使設定生效。
4. 新增使用者：在 `nginx` 底下執行 `htpasswd -c key 使用者名稱`，之後會要求輸入兩次密碼。
    - `htpasswd` 需要在 `linux` 環境底下安裝，筆者會進到 nginx 的 container，然後安裝。
        1. `docker exec -it container名稱 bash`
        2. `apt-get update` `apt-get install mini-httpd -y`
        3. 切換到 `\etc\nginx\conf.d\`
        4. `htpasswd -c key 使用者名稱`

編寫備註

- 筆者使用 `Typora` 來編寫文件 (所見即所得的 `markdown` 編輯器)。
- 如果要使用 `Typora` 來編寫 JS 文件，透過外部引入(加上 `defer` 可以控制最後載入) 就可以避免與 markdown 語法衝突。