# 致理技術學院

## 資 訊 網 路 技 術 系
## 實 務 專 題 報 告

## 人臉辨識



指導老師：高楊達 教授

學生：周祐瑋(69334110)

陳建霖(69334123)

王志楷(69334127)

黃浩銘(69334137)



中華民國 97 年 1 月

# 致理技術學院

## 資 訊 網 路 技 術 系
## 實 務 專 題 報 告

# 人臉辨識

學生：周祐瑋(69334110)
陳建霖(69334123)
王志楷(69334127)
黃浩銘(69334137)

本成果報告書經審查及口試合格特此證明。

指導老師：_____

中華民國 97 年 1 月

# 專題研究授權書

本授權書所授權之專題研究為_____

_____

共_____人，在致理技術學院資訊網路技術系 _____學年度第_____學期完成資網實務專題。

專題名稱：_____

□同意　□不同意

本組同學共_____人，皆同意著作財產權之論文全文資料，授予教育部指定送繳之圖書館及本人畢業學校圖書館，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。

上述授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。上述同意與不同意之欄位若未勾選，該組同學皆同意視同授權。

指導教授姓名:

專題生簽名:　　學號:

(親筆正楷)　　(務必填寫)

中華民國　　年　　月　　日

# 誌　　謝

在致理四年中最充實的日子，就是三下到四上這一年的專題研究過程，這段時間組員們大家互相扶持、勉勵，雖然很辛苦但也充滿許多回憶。

最一開始，首先要先感謝我們專題指導老師高楊達教授，讓我們對於影像處理相關的程式沒有基礎的情況下，花了非常多的心力漸漸讓我們往更高的層次邁進，總是不辭辛勞，適時給予我們研究方向的啟發，提供寶貴的經驗，讓我們專題的研究成果更為完滿。

其次我們要感謝同在高楊達教授指導下另一組專題研究的同學們，在遇到困難時大家能相互支持與關心，彼此互相加油與砥礪。

最後，我們要感謝我們的家人在背後默默支持與關心，感謝他們一直以來對我們的付出。在此謹向所有幫助過我們的老師、同學與朋友，誠摯的感恩與祝福。

組員 周祐瑋 陳建霖 王志楷 黃浩銘 謹誌於

致理技術學院 資訊網路技術系中華民國九十七年一月

# 摘　　要

我們這組專題實務是為人臉辨識，首先由程式偵測膚色並規範出臉部位置，之後由人工手動將雙眼、嘴巴、鼻子等較明顯位置之五官定位，以利於找出人臉位置，最後再利用人臉之特徵進入(feature searching)資料庫作大量的圖像搜尋，進行分析辨識其身份。

關於人臉偵測的部份，我們只在專題報告中對相關技術加以介紹，因為我們是以人工的方式進行確認，所以沒有應用到相關的技術，影像輸入之後，我們將人臉上的雙眼、嘴唇手動點出，接著再把其點與點之間的距離記錄下來，雙眼之間的距離、嘴唇到眼睛的距離，就是我們用來人臉辨識的特徵資料，就可以藉由這些資料來判斷此人的身份。

**關鍵詞：**偵測膚色、五官定位、人臉特徵、資料庫搜尋、人臉辨識、判斷身份。

# 目　　錄

第一章 緒論..... 1

第一節　研究動機與目的..... 1

第二節　相關研究..... 2

第三節　人臉偵測..... 3

第四節　人臉辨識..... 4

第五節　專題架構..... 5

第二章 人臉辨識技術探討與理論..... 7

第一節　色彩模型..... 7

第二節　RGB..... 7

第三節　YCbCr..... 9

第四節　HSI..... 11

第五節　NCC..... 15

第六節　影像模式..... 15

第七節　BMP..... 16

第八節　JPEG..... 17

第九節　GIF..... 18

第十節　膚色偵測..... 18

第十一節　邊緣檢測..... 19

第三章　人臉辨識系統..... 24
第一節　人臉特徵擷取原理 ..... 24
第二節　相連區域標示法 ..... 25
第三節　眼睛特徵 ..... 28
第五節　人臉特徵相對位置 ..... 30
第六節　人臉特徵距離 ..... 30
第七節　應用公式 ..... 32
第八節　系統架構 ..... 35
第四章　操作說明..... 36
第一節　訓練畫面操作說明 ..... 36
第二節　辨識畫面操作說明 ..... 38
第五章　結論與未來展望..... 40
附錄一..... 45

# 圖　目　錄

圖 2- 1　RGB色彩模型中光的混和示意圖 ..... 8

圖 2- 2　RGB與YCbCr表示情形之影像 ..... 11

圖 2- 3　三角型色彩平面 ..... 12

圖 2- 4　圓形彩色平面 ..... 13

圖 2- 5　RGB轉HSI的示意圖 ..... 14

圖 2- 6　BMP點陣圖 ..... 16

圖 2- 7　JPEG轉換取樣示意圖 ..... 18

圖 3- 1　相連區域標示法(1) ..... 26

圖 3- 2　相連區域標示法(2) ..... 26

圖 3- 3　相連區域標示法(3) ..... 27

圖 3- 4　眼睛 ..... 28

圖 3- 5　嘴唇 ..... 29

圖 3- 6　臉部特徵中心點 ..... 31

圖 3- 7　臉部特徵位置間距 ..... 31

圖 3- 8　臉部特徵相對位置 ..... 32

圖 4- 1　訓練畫面(1) ..... 36

圖 4- 2　訓練畫面(2) ..... 37

圖 4- 3　訓練畫面(3) ..... 38

圖 4- 4　辨識畫面(1) ..... 39

圖 4- 5　辨識畫面(2) ..... 39

# 表　目　錄

表 1　相連之區域比較 ..... 27

# 第一章 緒論

## 第一節　研究動機與目的

目前市面上的數位照相機，許多都擁有著自動偵測臉部的功能，讓使用者在照相的時候不用擔心失焦的問題。而我們的專題主要也是針對人臉的部份，進行單人/多人臉部偵測的技術。我們利用程式對其圖像分析出人臉所在的位置，再針對其分析出的資料建立一個資料庫。最終再依程式所取得的臉部特徵資料，進入資料庫進行比對，進而完成人臉辨識。

好萊塢有許多的科幻電影、經常將類似的技術當作題材，而現在科技日新月異，許多的技術也不斷的研發出來，將來我們出門很有可能不必帶鑰匙、就醫也不需帶健保卡、更有可能連身份辨識方式也無需利用身份證來進行身份確認了，只要經由設置於門口的監視器輸入臉部影像，找出特徵後再經由程式來判斷其身分。而隨著科技日漸的進步，在機械設備普及以及技術的成熟下，有越來越多的人在臉型辨識這塊領域有著不凡的研究；以往臉型辨識往往受限於外在環境與影像清晰等等的問題而無法精確、完整的對其影像做出判斷，然而、在現今的科技及技

術之下，這些因素將不再是問題，如此一來便可以將不穩定的因素控制在最小，有助於研究，在未來要實現上述所說的那些情況，也不再是空想了。

現在臉型辨識這領域上面發展了許多的技術，其中最常見的是類神經網路辨識系統，其最大的優點是，不需要了解系統的數學模型為何，直接以神經網路取代系統的模型，一樣可以得到輸入與輸出之間的關係。因此，我們要先建立一個影像資料庫，藉由影像資料庫內的資訊，我們只要將影像輸入，便可以快速的來進行影像辨識，進而得知結果；而建立一個資料庫需要分成三個步驟：(1)擷取臉部特徵→(2)將臉部特徵轉換成參數並建立資料庫→(3)利用人臉辨識方法辨識。

利用這種生物認證(Biological Authentication)來作為身份的辨識方法，比傳統方式更為準確且方便，不必擔心被假冒或複製。若能利用這樣的方式來辨識身份，就無需隨身攜帶著大量的證件，使得身份的確認變得快速、可靠及便利。對於安全系統，罪犯比對等等的問題幫助甚大。

## 第二節　相關研究

人臉偵測（Face Detection）與人臉部辨識（Face recognition）之間的

差異在於，人臉偵測是輸入影像於程式中，找出影像中臉部的區域在什麼範圍，並利用臉部表面的特徵，例如：抓出眼、口、鼻的位置，測得距離、角度與面積；而人臉辨識則是，在人臉偵測之後，給予程式一張臉部影像，使用不同的演算法加以分類，並建立每個人的臉部特徵資料庫，使辨識系統判斷出影像中的人是誰。

## 第三節　人臉偵測

人臉偵測的部份我們提出以下二種方法來討論：

①邊緣（dege）

以邊緣來定義特徵，在最早是由 T. Sakai[9]等人所提出的邊緣描繪（edge representation）方法，其方法為將照片中人臉的特徵用線條描繪分析出人臉位置，另外 I. Craw[10]等人提出分級架構（hierarchical frame work）用以追蹤頭部的輪廓，利用曲線限制在有雜訊的邊緣時無法集中的性質，在頭部輪廓內利用邊緣特徵得到人臉部形狀、位置之間的關系，這是線跟隨器（line-follower）方式。

②灰階（gray-level）

我們將人臉影像灰階之後，可以在眼睛、嘴唇、眉毛等部位，找到

臉上較暗的部份，利用灰階值差距來定義門檻（threshold）定位出較暗的區域，取得人臉的特徵。

人臉偵測還不止這些方法，但本專題主要目的是進行人臉辨識，所以人臉偵測方面的相關研究就不在此綴述。

## 第四節　人臉辨識

人臉辨識相關的研究可以由以下三種方法發展：

一、擷取特徵（Feature-based）

經由擷取許多臉部特徵（例如：眼睛、眉毛、鼻子…等五官資料），運用其特徵之結合形成特徵向量，再使用測量法中的正交集合（orthogonal sets of measurements）來增加區分的能力。而所謂不同特徵的區分能力是經由單一特徵不斷的測試與整合後，在特徵向量與正交集合持續訓練後而產生的結果，以此結果找出最佳的區分方法，以做為日後辨識臉部等相關研究為參考之用。[11]

以擷取特徵做為基礎的方法成功的首要條件是，必需能夠完美的取得臉部特徵，但是到目前為止，還沒有一個方法是可以稱得上完美無缺的擷取方法。

二、類神經網路（Neural Network）

類神經網路可以將事先所收集到的樣本，經由訓練而學習到有決定性的規則（underlying rules）。類神經網路機率判斷法，其方法是結合了類神經網路與統計學而成，不但可以運用於人臉辨識，在生物辨識與認證系統下，更有好的成果。除此之外，類神經網路擁有了適應力強的特性。

三、主要構成要件分析基礎方法（Principal Component Analysis）

主要構成要件是分析產生特徵臉部（Eigenfaces）。這樣的方法使得人臉影像複雜度可大幅的下降，且能達到有著相當水準的辨識機率。但如果當人臉的尺寸變化時，將會降低其辨識機率。而特徵臉的方法是將原來人臉的影像空間，轉換為易於區分的特徵臉部空間，我們稱這些基本的向量為主要構成要件。

## 第五節　專題架構

此專題目一共有五個章節，第一章的緒論，主要在說明我們專題研究目的與動機和整個報告的架構。第二章是針對我們在人臉辨識技術上，所使用的技術與理論加以做探討介紹。第三章是介紹辨識系統臉部特徵的

擷取和辨識的條件，以提供參數資料庫的建立。第四章則是系統的介面和操作說明。第五章為本專題之結論與未來展望

# 第二章 人臉辨識技術探討與理論

## 第一節　色彩模型

色彩模型又稱為色彩空間或色彩系統，通常以三個座標軸來表示色彩，目的是為了利用某種標準來指定色彩。而各種模型間有著不同的特性與應用範圍，像是電腦是以 RGB 為顏色依據；YCbCr 則運用於數位視訊上；HSI 接近人們用來描述色彩的方式；NCC 可以解決因為光源亮度所造成的問題，以下就依這四種模型來加以說明及介紹。

## 第二節　RGB

人眼所見的各種色彩是因為光線有不同的波長所造成，經由實驗發現，人類肉眼對其中三種波長的感受特別強烈，只要適當的調整這三種光線的強度，就可以讓人眼可以感受到所有顏色。[5]

將 R、G、B 三種顏色光重疊的時候，每二種混合可以得到青綠色（cyan）、紫紅色（magenta）、黃（yellow）等中間色，所有的顏色合在一起就變成白色。如圖 2-1 所表示：

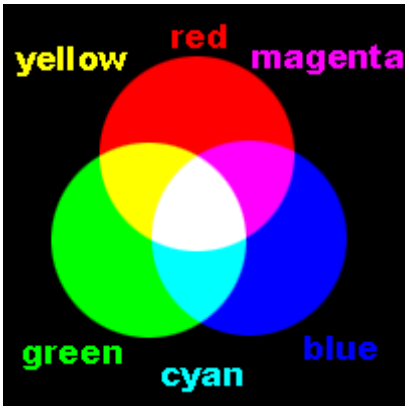



圖 2- 1　RGB 色彩模型中光的混和示意圖

RGB 就是紅(red)、綠(green)、藍(blue），這三種顏色稱為光的三原色。所有的彩色電視機、螢幕都具備產生這三種基本光線的發光裝置。因為這三種光線的混合幾乎可以呈現出所有人眼所能辨識的顏色，因此電腦裡頭就用 RGB 三個數值的大小來標示顏色，每個顏色用 8bit 來記錄，可以有 0~255，共 256 種亮度的變化，三種乘起來就有一千六百多萬種變化，這也是我們常聽到的 24 bit 全彩。[5]

而 RGB 表示的影像，有著些許的缺點，其缺點是 RGB 會因為光源的亮度強弱而有著深淺的變化，使得物體在相同顏色但不同亮度的環境下，影像所呈現出的顏色會略顯得不同，在作顏色的分割會有很大的誤差，因此必須採用光線強弱較不敏感的色彩空間（Color space)，這樣的方式對顏色做分割，例如正規化 RGB（Normalized RGB）可對光線的靈敏度減少，進而取代原先的 RGB。其正規化的公式為：

$$r = \frac{R}{R+G+B} \quad (1)$$

$$g = \frac{G}{R+G+B} \quad (2)$$

$$b = \frac{B}{R+G+B} \quad (3)$$

$$r+g+b=1 \quad (4)$$

我們只要把正規化所得到的 r、g 與 b，取代原先的 RGB space，就可以讓顏色對光線的靈敏度降低，在對於膚色做選取的作業時，就能夠把膚色範圍從影像中更精確的分離出來。

## 第三節　YCbCr

YCbCr 用於數位視訊中，Y 是指由彩色轉換成灰階影像，所得到的灰階值或稱為亮度值；Cb 與 Cr 則是儲存色彩資訊。Cb 為藍色成份與參考值的差距；Cr 為紅色成份與參考值的差距。

RGB 與 YCbCr 的關系如下：

$$\begin{aligned} Y &= 0.2990R + 0.5870G + 0.1140B \\ Cb &= -0.1687R - 0.3313G + 0.5000B + 128 \\ Cr &= 0.5000R - 0.4187G - 0.0813B + 128 \end{aligned} \quad (5)$$

$$\begin{aligned} R &= Y + 1.40200(Cr - 128) \\ G &= Y - 0.34414(Cb - 128) - 0.71414(Cr - 128) \\ B &= Y + 1.77200(Cb - 128) \end{aligned} \quad (6)$$

RGB 轉換為 YCbCr 色彩空間公式為：

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \quad (7)$$

轉換公式是依據人類眼睛對 R、G、B 三原色不同的敏感度而來，三種顏色的敏感度依序為；綠色（0.587）、紅色（0.299）、藍色（0.114）。數值愈大的顏色，其表示對人的眼睛較為敏感。圖 2-2 所示為以 RGB 與 YCbCr 表示情形：

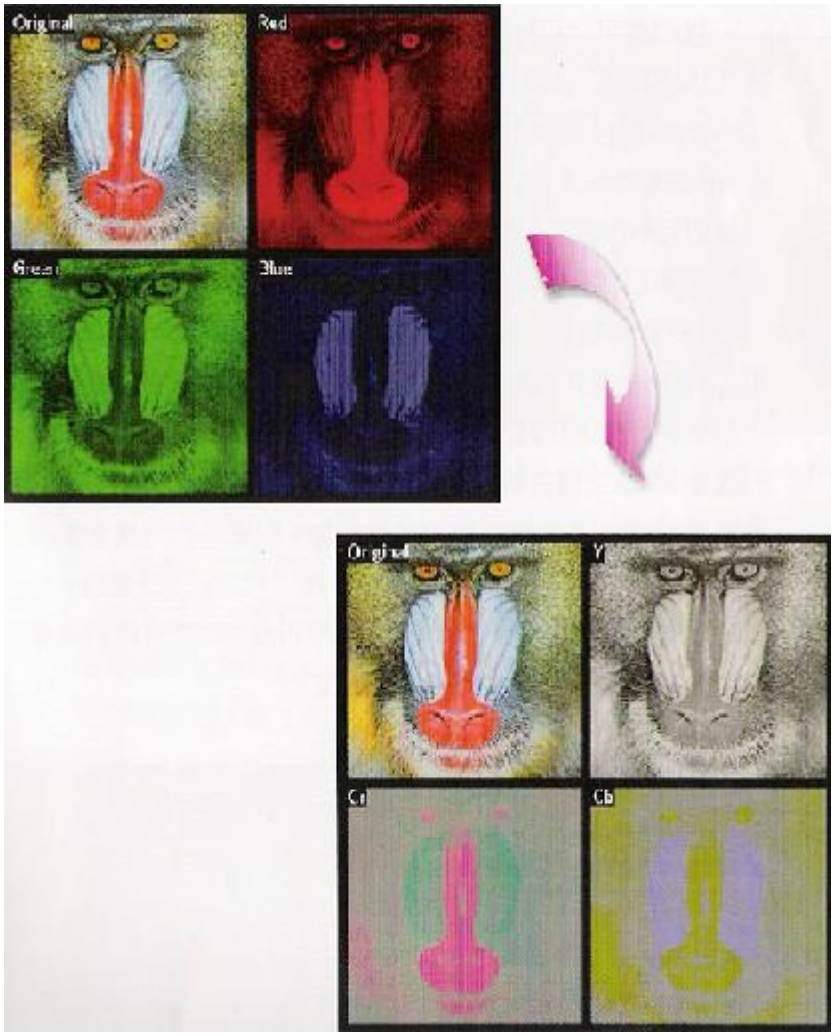



圖 2- 2　RGB 與 YCbCr 表示情形之影像

## 第四節　HSI

HSI 是根據人類對顏色的認知所建立的顏色模型，以色調、飽和度、亮度組合成各種顏色。

色調（Hue）：為物體傳導或反射的波長，其代表顏色本身，通常以0°~360°表示；飽和度（Saturation）：又稱色度（chromaticity），是指顏色的強度或純度，表示色彩的鮮艷與否。飽和度代表灰色與色調的比例，並以 0%(灰色)~100%(完全飽和)來衡量。強度軸的飽和度 0；亮度（Intensity）：為顏色的相對明暗度，以 0%(黑色)~100%(白色)的百分比來衡量。[4]

HSI 的優點是分離影像中色彩與亮度的部份,對顏色分割有較佳的效果，由於亮度被獨立出來，彼此之間的關聯性很小，故適合用來做影像前處理，所以很多的人臉偵測系統都使用 HSI 色彩空間，但由於電腦大部分是以 RGB 來表示，所以要把 RGB 轉換成 HSI。圖 2-3、圖 2-4 表示 HSI 三角型色彩平面與圓形彩色平面：

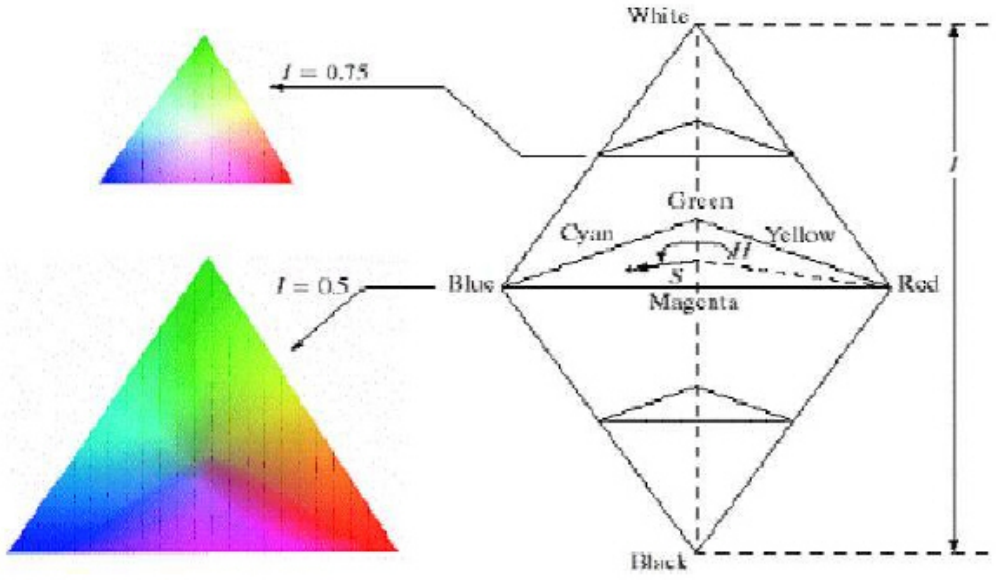



圖 2- 3　三角型色彩平面

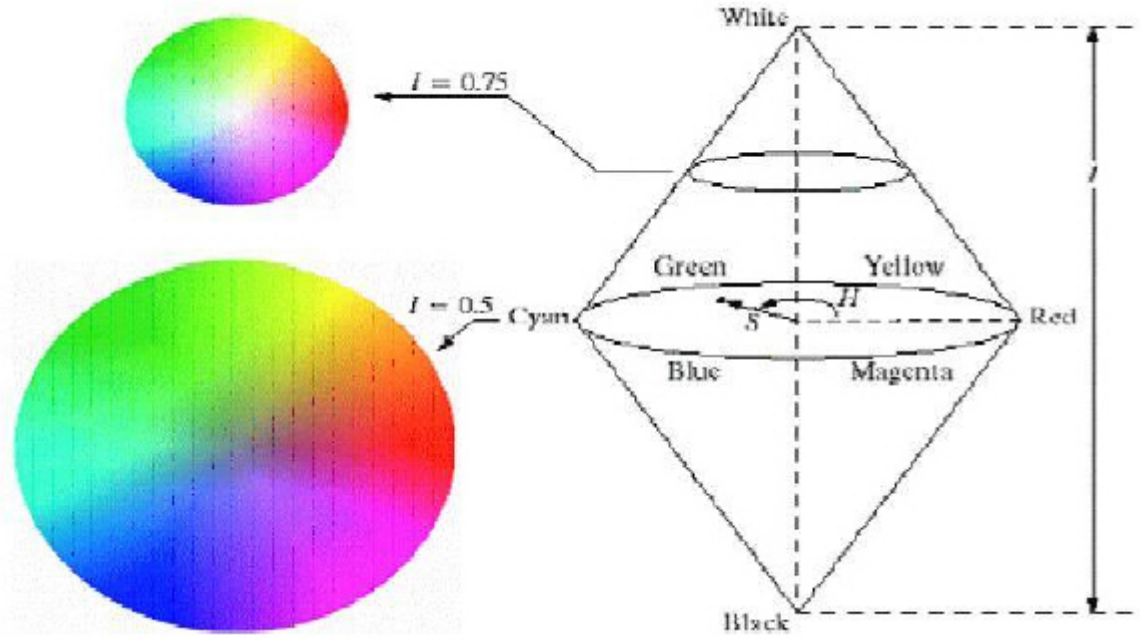



圖 2- 4　圓形彩色平面

RGB 色彩模型轉換到 HSI 色彩模型，由以下式子進行轉換。轉換公式為：

$$H = \begin{cases} \theta & if\ B \le G \\ 360 - \theta & if\ B > G \end{cases} \qquad (8)$$

其中

$$\theta = \cos^{-1}\left( \frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{\frac{1}{2}}} \right) \qquad (9)$$

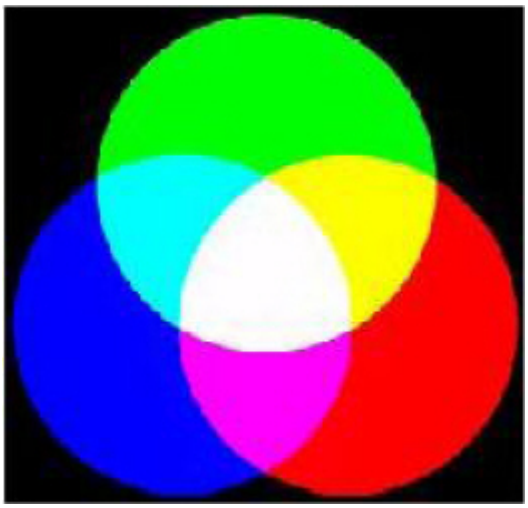
飽和度為：

$$S = 1 - \frac{3}{(R+G+B)}\left[\min(R,G,B)\right] \qquad (10)$$

亮度為：

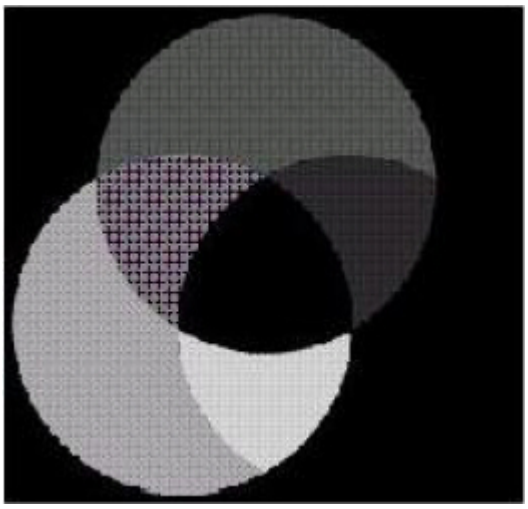
$$I = \frac{R+G+B}{3} \qquad (11)$$

色彩模型可由上述公式從 RGB 轉換到 HSI，也可以推導從 HSI 轉換回 RGB 色彩模型的公式。

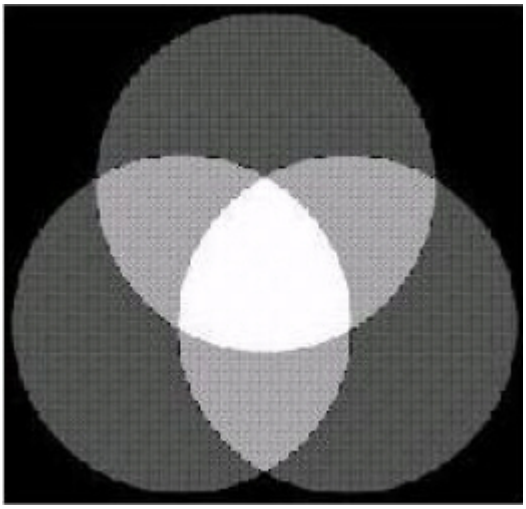


（i）RGB 組成之影像

（iii）飽和度



（ii）色調



（iv）亮度

圖 2- 5　RGB 轉 HSI 的示意圖

由上圖可看出 RGB 影像和所對應之 HSI 成份；(i)RGB 組成之影像，(ii)-(iv)各別表示影像的 H、S 與 I 的成分。

以上 RBG、YCbCr、HSI 三種是目前常使用於顏色分割的方法，我們這次專題選用的是 HSI 色彩空間做為顏色分割工具。

## 第五節　NCC

NCC（Normalized Color Coordinates）是解決 RGB 影像因為光源亮度的強弱，形成物體在相同色彩的地方呈現出深淺不同的顏色，其中使用的方法是將 R 與 G 做正規化，使 R 與 G 對亮度的靈敏度降低，因為 B 對亮度的靈敏度較小，所以可以略過，利用正規化得到 R 與 G 做膚色範圍門檻值選取，就可以把影像中的膚色分離出來。

## 第六節　影像模式

我們在處理影像資料時，一般最常用到的影像格式為 BMP 未經壓縮的影像檔格式，GIF 非破壞性壓縮，以及 JPEG 破壞性壓縮，其中 BMP 與 GIF 屬於點陣圖，是使用像素陣列表示的圖像，像素的色彩以 RGB 組合或灰度值表示，JPEG 則是向量圖形，用點、直線或多邊形等數學方程的幾何圖元表示圖像，以下對這三種影像模式來介紹。

## 第七節　BMP

BMP 為點陣圖(BitMap)的簡寫，是微軟(Microsoft)作業系統內部使用的點陣圖格式，BMP 格式一般沒有經過壓縮處理，所以通常比相同圖檔的壓縮圖像大很多，較不適合在網際網路或其他低速有容量限制的媒介上傳輸。

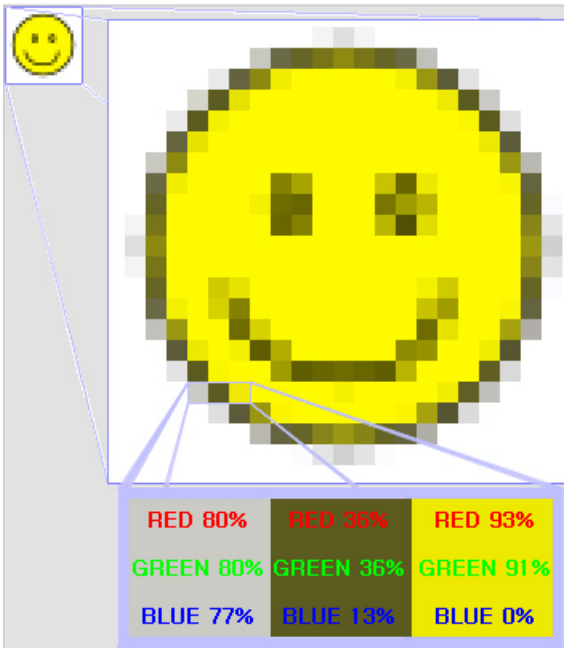



圖 2- 6　BMP 點陣圖

依照顏色深度不同，圖像上一個像素可以用一個或多個位元組表式，由 n/8 所確定（n 是位深度，1 位元組包含 8 個數據），例如：2（1 位）、16（4 位）、256（8 位）、65536（16 位）、1670 萬（24 位）種顏色。[6]

$$\text{BMP 檔案大小} \approx 54 + 4 \cdot 2^n + \frac{width \cdot height \cdot n}{8} \qquad (12)$$

54 代表文件頭，$4 \cdot 2^n$ 是彩色調色板的大小，但這是一個近似值，一個特定圖像並不一定會使用所有的顏色，典型的 BMP 檔案格式使用 RGB 色彩模型，每種顏色都是由不同強度(0~255)的紅色、綠色和藍色組成。

## 第八節　JPEG

JPEG 由國際標準組織（International Organization for Standardization ，簡稱 ISO）和國際電話電報諮詢委員會（International Telegraph and Telephone Consultative Committee ，簡稱 CCITT）所建立的一個數位影像壓縮標準，是一種針對相片影像而廣泛使用的一種失真壓縮標準方法，JPEG 採用可失真（Lose）編碼的概念，利用數位餘弦轉換法（Discrete Cosine Transform，簡稱 DCT），原始影像由 RGB 轉換為 YUV 色彩空間，Y 代表一個像素的亮度，U 代表色調，V 則是飽和度，因為人類的眼睛在 Y 成份可以比 U 和 V 更加清楚，所以利用這點來進行縮減取樣或叫色度抽樣（chroma subsampling），JPGE 縮減取樣（Downsampling）比有 4:4:4（無縮減取樣），4:2:2（在水準方向 2 的倍數取一個），還有最普遍的 4:2:0（在水準和垂直方向 2 的倍數中取一個）。[6]圖 7 是將同一張圖片經過無縮減與高縮減取樣所呈現出的結果：

（a）無縮減取樣

（b）高縮減取樣

圖 2- 7　JPEG 轉換取樣示意圖

## 第九節　GIF

GIF（Graphics Interchange Format）是一種點陣圖圖形格式，以 8~256 種顏色的圖向，採用 LZW 壓縮演算編碼，基本的原理是把原始影像資料中重複的字串編成一個表，利用表上的索引值取代原始影像資料中的字串，由於索引值的體積比原始影像中的字串體積小，所以 GIF 能使影像有壓縮的效果。GIF 的優點是在一定程度上保證圖像質量的同時將體積變的很小，缺點是由於編法的限制，故其色彩支援只有 256 色。

## 第十節　膚色偵測

在不同時間及不同地點，其膚色會因光線的明亮關係影響到皮膚所呈現出來的反應，使得皮膚看起來有所不同，偵測的結果亦不同。

而數位影像處理大都利用數值矩陣的方式來處理圖形資料，矩陣中元素值的大小代表影像中所對應的圖點的亮度值，這些元素我們稱之為像素、點素、圖元或畫素（pixel 或是 pel 即 picture element 的縮寫）。

在此數位矩陣中之元素，其明亮度依量化值而訂定出不同之灰階層度(gray level)。一般而言，灰階為 2 的冪次方，如 $2^8$ 即表示 256 個灰階。

在進行膚色偵測，首先要定義膚色的範圍，這裡我們選擇 HIS 色彩空間，優點是分離影像中色彩與亮度的部份，降低顏色對亮度的依賴。這部分完成後，接著是雜訊的去除，因為影像經過膚色偵測處理後，還是會發現在人臉周圍，背景會有類似膚色的雜點，為避免影響後續處理動作，雜訊去除使用到侵蝕（Erosion）與膨脹（Dilation），參考[1]方法是對影像先做侵蝕處理再做膨脹處理為佳。

## 第十一節 邊緣檢測

一般而言，影像處理過程包括：基本運算處理、影像邊緣增強

(edge enhancement)、邊緣檢測(edge detection)與邊緣連結等技術。此專題中、我們所用的邊緣檢測方式是以灰階影像為基礎。因此，必須找出彩色照片各像素所對應的灰階值。

邊緣檢測是圖像處理和計算視覺的基本問題，其邊緣檢測的目的，是標示數字圖像中亮度變化明顯的點。圖像屬性中的顯著變化通常會反應出屬性的重要事件與其變化的過程結果。而這些結果包括了有（i）深度上的不連續（ii）表面方向的不連續（iii）物質屬性的變化（iv）場景照明的變化。[6]

灰階影像邊緣檢測及邊線強化的處理技術包括灰階滑移(sliding)、灰階擴展(stretch)、影像反轉、對比強化、對比均衡等方法。在強化圖形邊線，是指影像的每個像素作簡單的數學運算，這樣可以改變影像整體的明暗度以及修正圖形的特性。灰階滑移則是將像素原本的灰階度數值加減一個常數，而使得整個灰階度分布圖往左或往右滑動，除了飽和(灰階度數值為 0 或最大)的像素數目會有所更動外，其整體灰階度分佈的曲線形狀可以說是沒有太大的變動。

假設 G(x,y) 為原始影像之點 (x,y) 之灰階度數值

而 G'(x,y) 為運算後影像之點 (x,y) 之灰階度數值

則

$$G'(x,y) = G(x,y) \pm a \qquad (13)$$

其中 a 為一常數

注意對 8 位元影像而言：

$$\text{當 } G'(x,y) > 255 \text{ 令 } G'(x,y) = 255 \tag{14}$$

$$\text{當 } G'(x,y) < 0 \text{ 令 } G'(x,y) = 0 \tag{15}$$

而灰階擴展是將影像灰階度數值全部加減一個常數、擴張運算則是將影像灰階度全部乘或除上某一個常數。經過濃度擴張運算後，我們可以觀察到、使用灰階度分佈長條圖來表示時，其間隔會變為原有的若干倍。而當乘上的常數大於 1，則影像度明亮，255 的灰階度數值像素的數目可能增加（以 8bit 的影像來看）。反之，若乘上的常數小於 1，則影像度偏暗。

$$G'(x,y) = G(x,y) \cdot a \tag{16}$$

其中 a 為一常數

對比強化則是增進影像的對比及動態範圍特性。如果能夠找出一個影像灰階度數值集結的地方，我們將可以令此集結外的最大灰階度數值與最小灰階度數值為對比強化運算的上下限，而將此影像的灰階

度伸展開來，使用 0 到 255 全部的灰階度資訊，就可以方便人眼的觀察。與擴張運算相比較，此方法可以使得灰階度數值飽和（灰階度數值超過 255）的像素數目減少，又可以避免整張影像灰階度數值偏高的情形。

$$G'(x,y)=\frac{G(x,y)-y_1}{y_2-y_1}\cdot 255 \qquad (17)$$

所謂反轉運算即是將影像中的亮處變暗，暗處變亮，造成了類似照片正片、負片的效果，在經過反轉運算後的影像，又稱為負片影像（Negative Image），因為人的眼睛敏感曲線在明亮處易呈現飽和狀態，亦無法分辨出明亮部份的細部結構，此時我們若將整個影像以反轉運算之方式進行轉換，則明亮的部份會轉換為較暗，此時再來觀察，自然比較容易辨別一些微細的差別。

對 8 位元影像而言，其反轉運算為

$$G'(x,y)=255-G(x,y) \qquad (18)$$

人臉影像則可經由上述所說明的各種技術加以處理，使其臉部特徵更為明顯，以利後續的邊線處理及特徵的擷取。

邊緣檢測常用的濾波方式為低通及高通 2 種，低通濾波器中常見的均化遮罩。是指將遮罩中所有灰階度數值加總後求其平均，然後寫入對應的像素中，因此又稱之為移動平均濾波器，這樣的方式可以將影像處理後多餘的雜訊去除。

這樣的方式，乍看之下好像和像素合併運算（Pixelize）一樣，但仔細想想，很容易發現兩者之間還是有所不同的。像素合併運算是將求出的平均值當作對應遮罩中所有像素灰階度數值，如果是 3×3 的遮罩，其出來的結果是 9 個像素灰階度數值大小都一樣；而低通濾波則是將求出之平均值寫入遮罩中所對應的像素，所以一次只處理一個像素的值，旁邊的像素則是在下一次的遮罩移動時再另行作處理。因此出來的結果，灰階度數值和周圍的各像素灰階度數值可能不一樣，也可能一樣。低通濾波通常會產生影像模糊的效果。[12]

而高通濾波是屬於在數位信號中的遮罩運算。如果我們把前中後的數值乘上若干倍的權重，然後加總成為新對應點的值，那麼這樣的行為通常和低通濾波有關，相反的，我們如將前後的值乘上權重後再相減，拿來當成新對應點的值，那樣大致上都是屬於高通濾波的方式。高通濾波的作用恰巧與低通濾波作用完全相反，其增強了影像高頻特性。

# 第三章　人臉辨識系統

每個人在這是世上都是獨一無二的，因此，人與人之間的辨識就可靠著臉部的特徵來辨識，如眼睛、鼻子、嘴型、臉型等等。透過臉部的特徵我們可以快速且輕易的分辨出對方，對電腦而言也是一樣，電腦在分辨時，也是依據這些特徵來做辨識，因此，辨識的特徵必須是清晰且明確的，而且必須將這些特徵標點出來方便電腦分析；所以必須將輸入的圖形正規化，並且將其轉換成可識別之參數，以利於電腦分析及其判斷。

## 第一節　人臉特徵擷取原理

進行人臉辨識的第一步驟，就是要先行建立一個完整的資料庫，因此，需要將輸入的所有臉部特徵取樣，並且將其臉部特徵清楚的標示出來之後才能夠進行比對。

因此，為了建立一個資料庫，必須先將人臉的特徵擷取出來，方可將這些特徵轉換成參數。本專題所採用的方式即是以人工的方式將輸入的臉部圖形取樣，將眼睛及嘴唇這些臉部特徵，藉由人工方式取出並以相連區域標示法找出中心點以找出特徵的相對位置，接著對圖

形裡的特徵描繪並將對象的特徵位置找出來，再依形狀、間距及角度為依據轉換成參數，建構資料庫以利提供比較。

在辨識方面，由於取出的臉部特徵不一定每次取在相同位置，且因影像圖片明暗及清晰度受到當時照相機焦距及光源方向和亮度等等的影響，因此即使相同的人但因照相的取景有些許的不同，各個臉部特徵器官的形狀取得也不盡相同，所以必須藉由點選的方式，點選出臉部特徵位置，再藉由電腦用相連區域比較法找出中心點。

## 第二節　相連區域標示法

所謂相連區域標示法是將一數位影像分割成若干區域，而這些相連區域是根據空間上相鄰位置且顏色值相似才會連成一個封閉區域，當我們用人工點選臉部特徵時，即以點下的那一點作為原點，此時會將原點放進暫存器裡以堆疊的方式暫存（如圖 3-1）當與相連的區域做比較時會將原點取出，原點會與相連的區域做相互的比較，其顏色值要是相近，就將其視為同一個物件，再將比較後的相連區域座標存進暫存器裡（如圖 3-2），要是數值差異較大，便視為相異的物件而忽視不去理會，以此方式將存放在暫存器的座標點一一取出，相繼與相鄰區域做比較，最後將相同的物件集結起來便可形成一個圖形；

藉由此方式找出圖形之後便可自動找出外框(如圖 3-3),再經由電腦運算處理計算出中心位置,如此一來便可以輕易的找出中心點,也不用擔心點選時無法正確的點出中心位置,此方式最大的好處就在於只需點出臉部特徵位置,便可經由電腦計算出中心點。

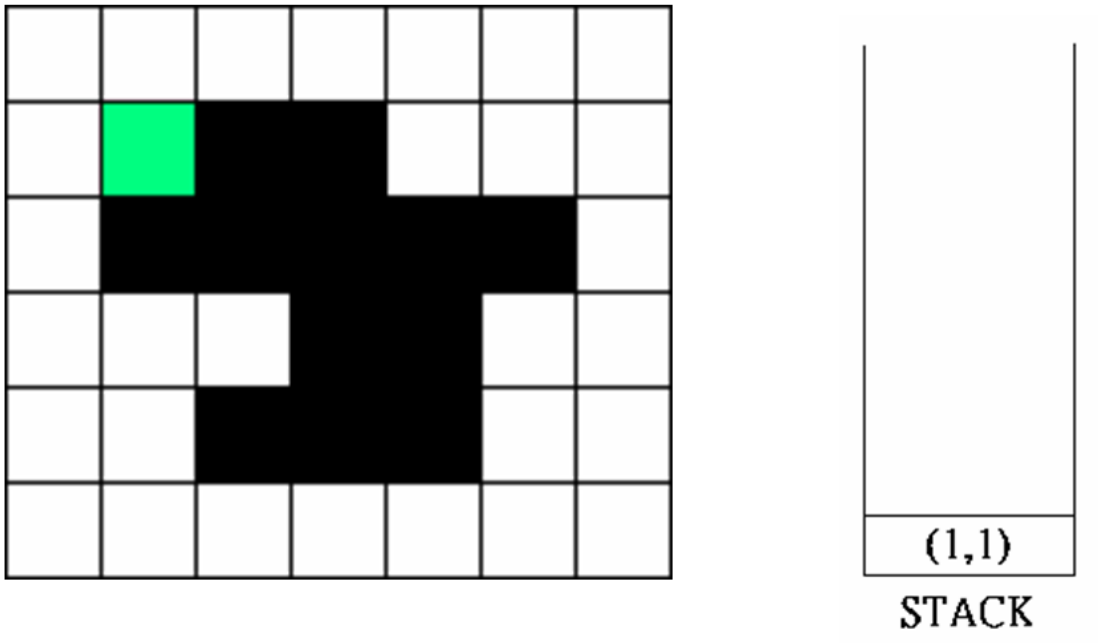



圖 3- 1　相連區域標示法(1)

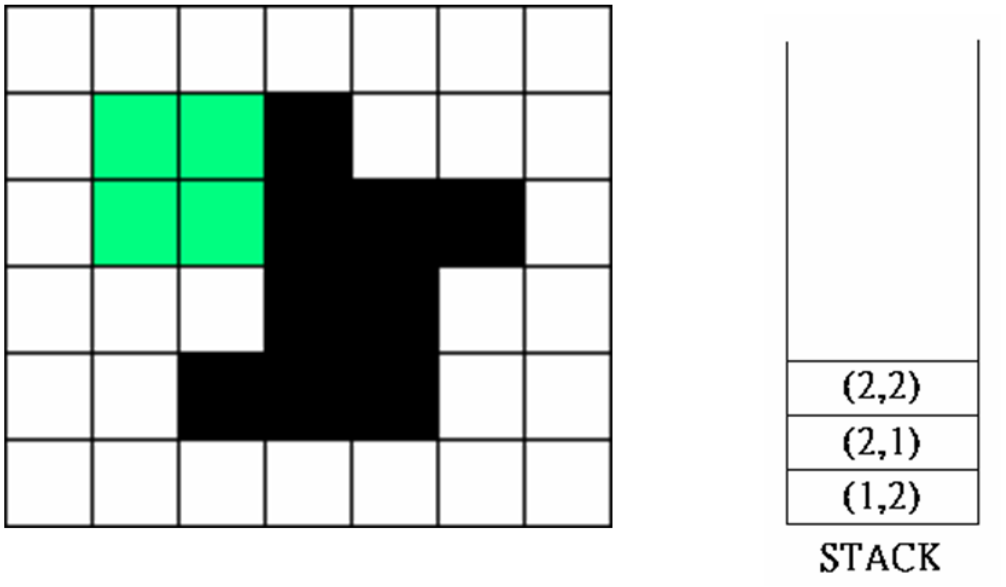



圖 3- 2　相連區域標示法(2)

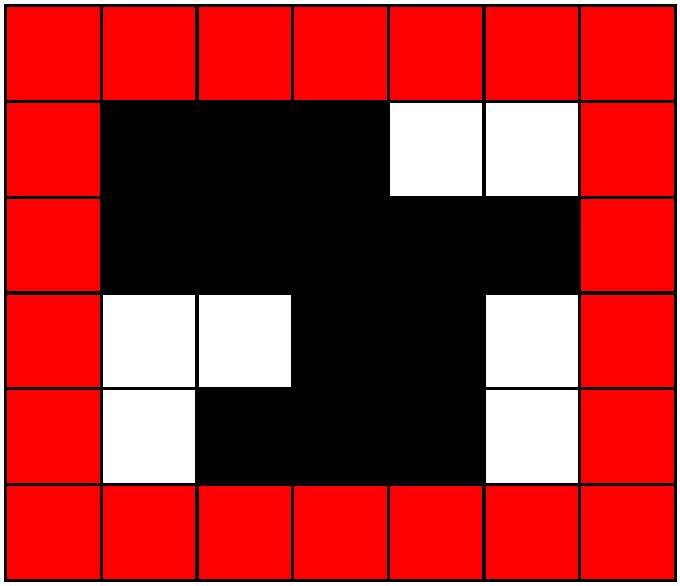


圖 3- 3　相連區域標示法(3)

**Min　(X,Y) ◊ (1,1)……起始座標**

**Max　(X,Y) ◊ (4,5)……結束座標**

**Min　(X,Y) ◊ (0,0) ……起始座標**

**Max (X,Y) ◊ (5,6) ……結束座標**

判斷區域相連標示之作法如下：

以（X，Y）為原點之初始值，開始與相連之區域比較，

abs($H_{(X,Y)} - H_{(X-1,Y-1)}$)＜T（T 為門檻值）；x-i,y-j,　i,j $\in \{-1,...,+1\}$

表 1　相連之區域比較

| | | |
|---|---|---|
| （X-1，Y+1） | （X，Y+1） | （X+1，Y+1） |
| （X-1，Y） | （X，Y） | （X+1，Y） |
| （X-1，Y-1） | （X，Y-1） | （X+1，Y-1） |

## 第三節　眼睛特徵

檢測眼睛時，由於影像圖片的明暗及影像圖片清晰度的關係容易產生誤差，因此，須先將眼睛的影像用相連區域比較法尋找出相同的影像區域去除雜訊，如此一來，便能取得較為完整且清晰的眼睛。使用相連區域比較法最大的目的是藉由原點跟周圍相互比較，直至將整個眼睛的輪廓大致都找出來，之後藉由計算找出中心點。由於以人工的方式找出的中心點，並不能確保其精確度，只要有些許的誤差即可能影響到之後的辨識，因此，為了確保辨識的準確度我們必須藉由相連區域比較法幫我們找出眼睛的中心點，以這種方式來找出臉部特徵中心點最大的好處，就是可以將誤差降低到最小，也可以將辨識流程標準化，不會因為操作人員不同，就產生辨識的誤差，也可以將操作簡單化，不需做複雜的指令或輸入許多的數值，只需點眼睛即可快速的找出中心點。

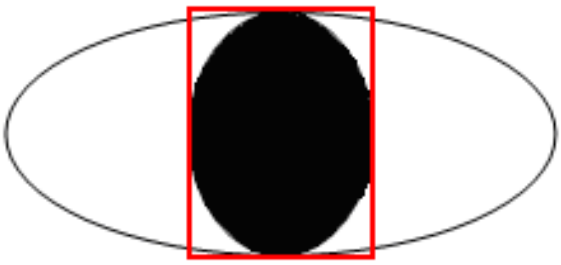


圖 3- 4　眼睛

## 第四節　嘴唇特徵

檢測嘴唇時，由於影像圖片的明暗及影像圖片清晰度的關係容易產生誤差，因此，須先將嘴唇的影像用相連區域比較法尋找出相同的影像區域，如此一來，便能取得較為完整且清晰的嘴唇。使用相連區域比較法最大的目的是藉由原點跟周圍相互比較，直至將整個嘴唇的輪廓大致都找出來，之後藉由計算找出中心點。此方式不會因為人為的疏失或操作的失誤，大大的降低辨識的準確度，而且也使的操作更為簡單方便，減少失誤的發生，也因為操作簡單化，不需做複雜的指令或輸入許多的數值，只需些許的時間便能找出嘴唇的中心點。

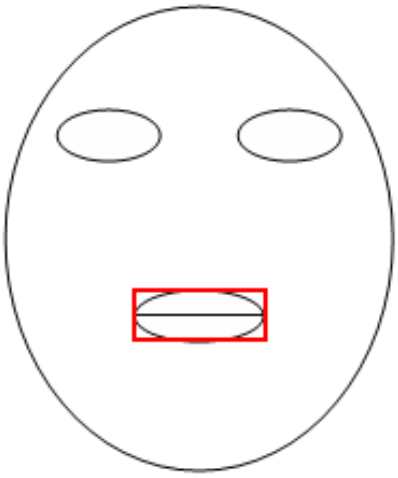


圖 3- 5　嘴唇

## 第五節　人臉特徵相對位置

人的臉不會因情緒及外在環境的關係而產生變化，例如：高興時嘴唇會上揚、生氣時眼睛會睜大以及遇到強光時會瞇著眼睛，這些變化都會影響到臉部特徵的形狀，這些情況對於人臉辨識都會產生極大的影響。因此，為了確保辨識的精準度，必須加上臉部器官的相對位置，因為不管臉部的表情如何的變化，臉部器官的相對位置永遠是不會改變的，會改變的只有形狀，所以我們利用這個原理將臉部器官之間相對位置的距離以及連線所形成之夾角，做為我們辨識的重要依據。

## 第六節　人臉特徵距離

臉部器官的相對位置有兩項，分別是兩眼之間的距離、眼睛對嘴唇，因此為了取出正確的相對距離，必須先在人臉器官上標記出相對位置的點，以確保每次在取相對位置時臉部器官之間的相對距離，不至於出現太大的誤差，相對位置如圖 3-6 所示

- 眼睛部份：取兩眼的中心點分別為 A 和 B
- 嘴唇部分：取嘴唇的中心點 C

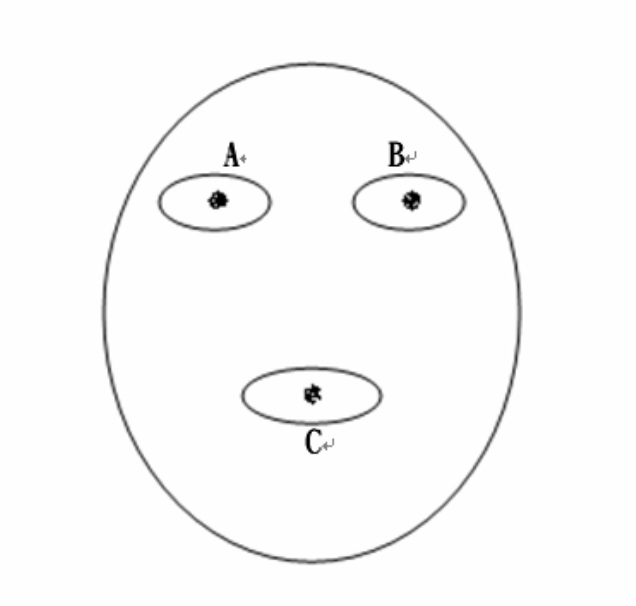



圖 3- 6　臉部特徵中心點

依據 A、B、C 這三個點，我們可以畫出雙眼及嘴唇的連接線段，如圖 3-7 所示：

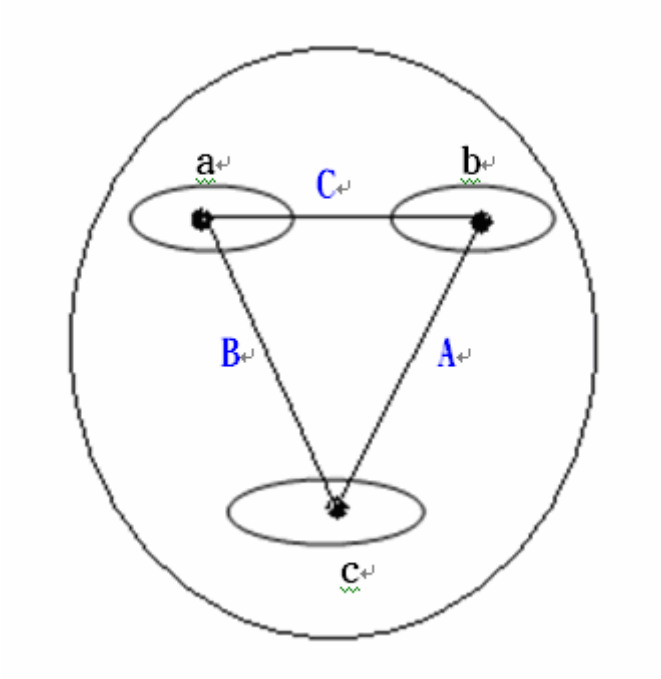



圖 3- 7　臉部特徵位置間距

在△ＡＢＣ中，若ＢＣ線段＝a，ＡＣ線段＝b，ＡＢ線段＝c

則有餘弦定理如下：

$a^2=b^2+c^2-2bc\ \cos A$　或　$\cos A=(b^2+c^2-a^2)/2bc$

$b^2=c^2+a^2-2ca\ \cos B$　　$\cos B=(c^2+a^2-b^2)/2ca$

$c^2=a^2+b^2-2ab\ \cos C$　　$\cos C=(a^2+b^2-c^2)/2ab$

第七節　應用公式

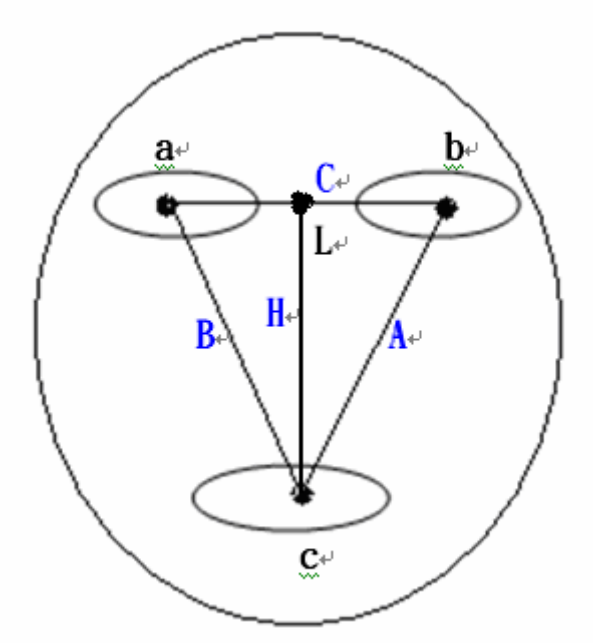
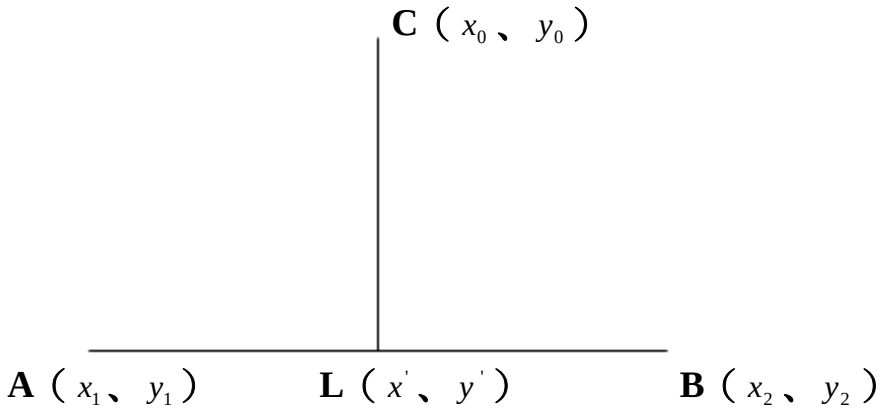



圖 3- 8　臉部特徵相對位置




$\overline{AB}$：

$$y'' = ax + by \qquad (1)$$

$\overline{CL}$：

$$y'' = \frac{1}{a}x'' + b'' \qquad (2)$$

（將$x_0$、$y_0$代入）：

$$y_0'' = \frac{1}{-a}x_0'' + b''$$

求得$b'$

$$y = ax + b_1$$

$$y'' = \frac{-1}{a}x'' + b_2$$

將點L（$x'$、$y'$）代入，解聯立方程式，求x’及y’

$$y' = ax' + b_1$$

$$y' = \frac{-1}{a}x' + b_2$$

$$0 = (a + \frac{1}{a})x' + (b_2 - b_1)$$

$$x' = \frac{b_2 - b_1}{a + \frac{1}{a}}$$

求 $y'$ ：

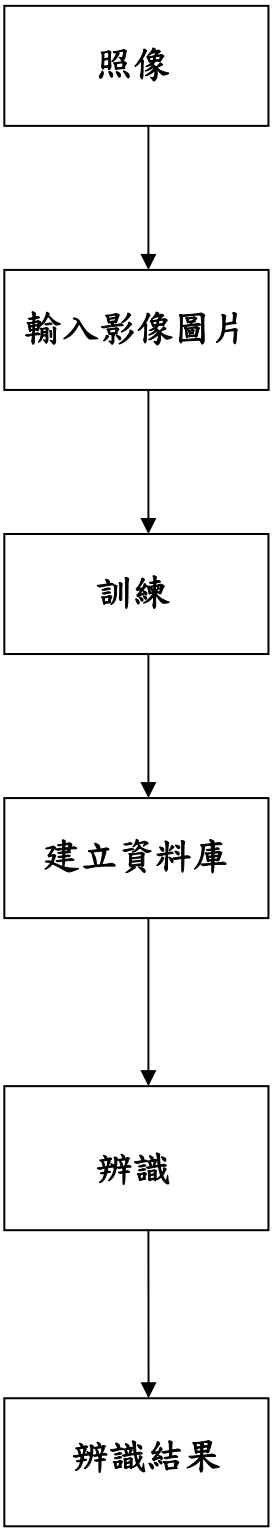
$$2ay' + y' = 2ab_2 + b_1$$

$$y'(2a+1) = 2ab_2 + b_1$$

$$y' = \frac{2ab_2 + b_1}{2a+1}$$

故 $\overline{CL} = \sqrt{(x_0 - x')^2 + (y_0 - y_1')^2}$ ，即圖 3-8 的 H

## 第八節　系統架構



# 第四章　操作說明

當我們將收集來的圖片資料，欲進行辨識時，我們須先行建立一個資料庫，才能對之後輸入的圖片進行比對；因此，須先將圖片放進訓練模式下進行分析比較，進而得到我們所需要的數據及資料，之後再到辨識模式下進行比對，比對完成後就可以得知辨識的結果了。

## 第一節　訓練畫面操作說明

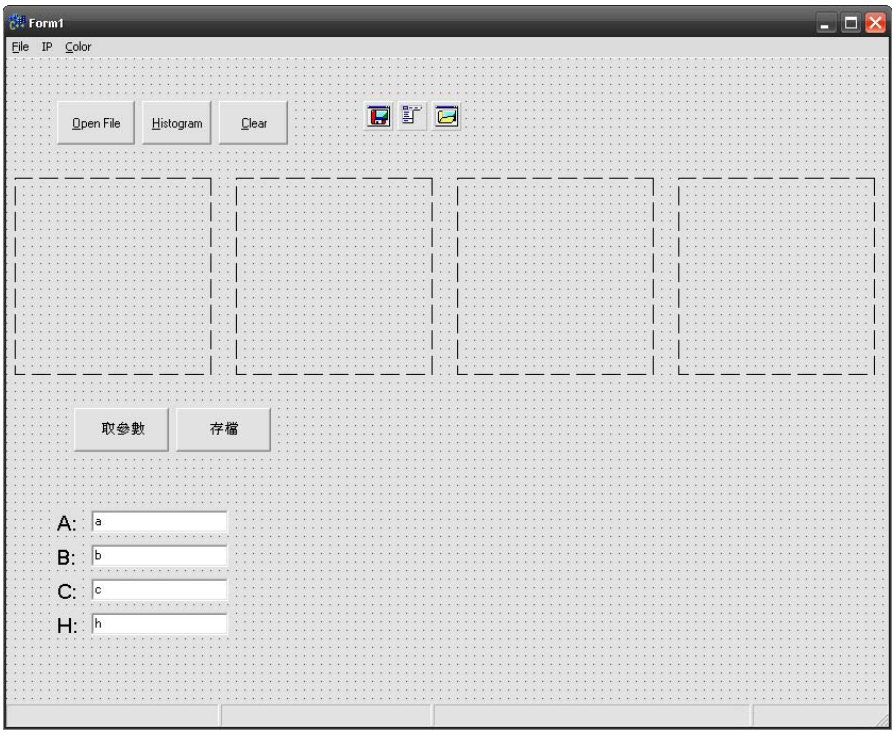



圖 4- 1　訓練畫面(1)

一開起畫面時(如圖 4-1)我們需將準備好的圖片輸入進去，因

此，先按 Open File 鈕，進入圖片所存放的位置，將需要的圖片點選起來，點選起來的圖片就會出現在紅色框框所標示的地方的位置上(如圖 4-2)，等到圖片都選取完就會顯示圖像在畫面上(如圖 4-3)，之後再按 取參數 鈕來取出線段 A、線段 B、線段 C 及線段 H 的數值，而取出的數值會出現在畫面上(如圖 4-3 紅色框框所標示的地方)，此時即是顯示成功取出數值，A、B、C 之數值是經由相連區域比較法找出中心點後的線段距離，而 H 之數值則為 A、B、C 的投影向量；之後便可以按 存檔 鈕來存檔製作資料庫，倘若選取圖片過程中點選錯誤圖片或是已完成圖片參數製作及可按下 Clear 鈕清除圖片。


圖 4- 2　訓練畫面(2)

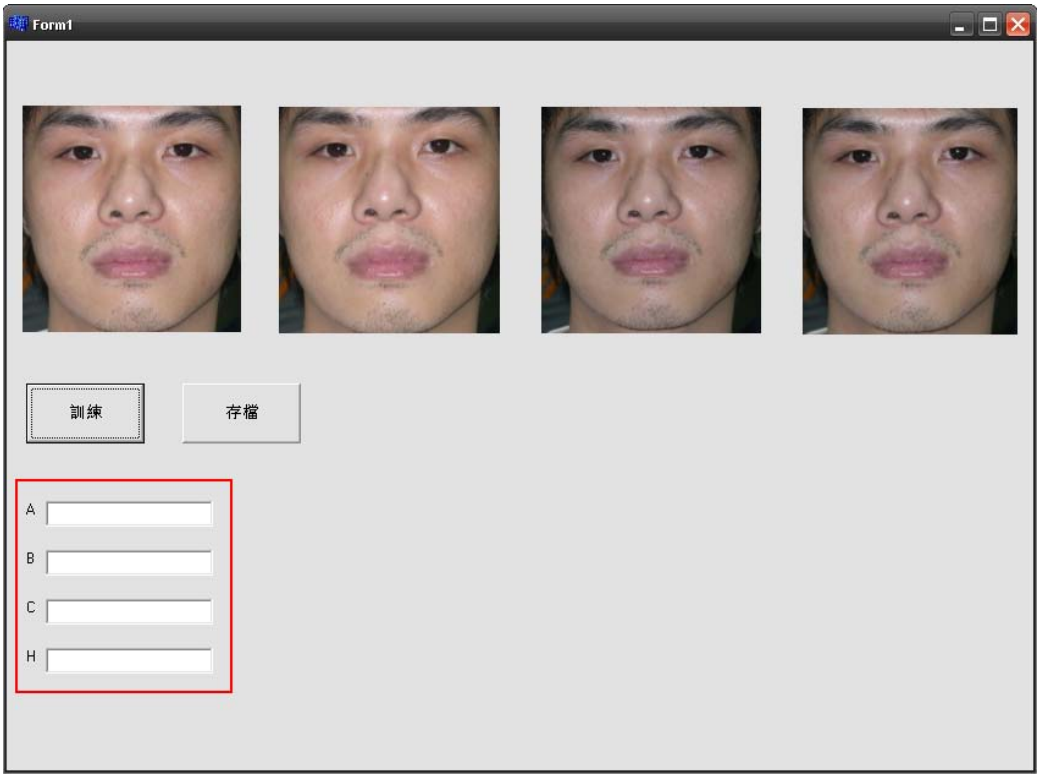



圖 4- 3　訓練畫面(3)

## 第二節　辨識畫面操作說明

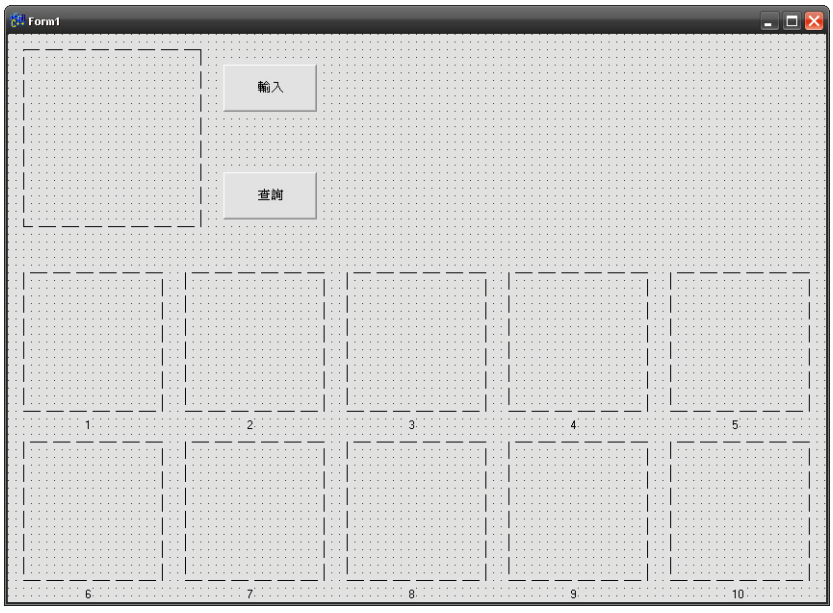


圖 4- 4　辨識畫面(1)

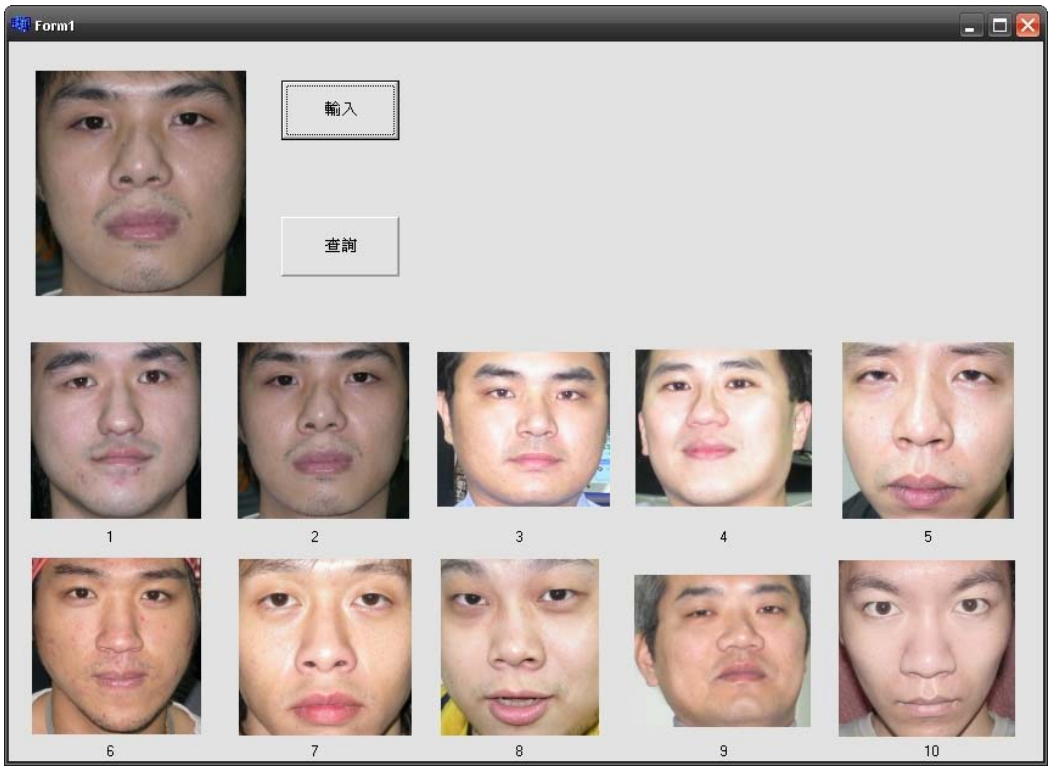



圖 4- 5　辨識畫面(2)

進入辨識畫面時（如圖 4-4）按下 

鈕，點選欲辨識的圖片，之後欲辨識的圖像便會出現在左上角,接下來在按下 

鈕進行資料比對；由於先前在訓練模式下進行過參數資料的擷取，因此，會將訓練模式下完成擷取的資料圖像與欲辨識的圖片藉由參數資料比對，最後會將資料庫裡較為相似的圖像由小到大依序排列出來，於畫面的下方顯示出來（如圖 4-5）。

# 第五章　結論與未來展望

本研究專題是以影像圖片來做人臉辨識，在人臉辨識的部份我們利用雙眼與嘴唇的臉部特徵來辨識，這裡我們是使用相連域標示法，找出眼睛、嘴唇的中心點，依此確認臉部雙眼與嘴唇的距離，定義其相對位置，因此，在辨識之前必須先進行特徵資料庫的建立，經由特徵點距離參數，透過程式訓練介面，擷取參數存入資料庫，只要參數樣本充足，對辨識結果愈加精確。

影像圖片的部份，因為目前影像辨識所採用的技術的關係，我們系統只能單純的處理正面的影像，其他側面、或是拍攝時臉部有偏轉，會影響到特徵取得的參數，進而會讓我們辨識的結果產生誤差，所以程式採用辯識者的影像圖片，頭髮不能遮到額頭、眼睛，以免造成判斷錯誤，特別需要注意在建立臉部特徵檔案的時候，臉部表情、拍攝的角度、相機的亮度，甚至是使用相機都必須盡量一致，盡量減少照片在拍攝時的誤差，台灣近視的人數比例將近全人口數的一半，有可能擷取檔案照片時，大多數使用者都有帶眼鏡，戴眼鏡的人在拍照時，眼鏡鏡架的顏色、眼鏡鏡片的顏色、鏡片反射的光線，都有可能造成照片的誤差，因此，人臉偵測時我們採用人工的方式來點出雙眼、唇部的位置，藉此將誤差

減至最小。

雖然此系統對於辨識時有較多的條件限制，但是操作的介面卻是極具人性化，最大的優點便是操作簡單方便、容易上手，不管是任何人都可以操作，對於喜好上網的人來說，此系統可以快速又簡單的在網路上搜尋辨識人物，不必學習或使用複雜的操作介面來辨識人物。

在人臉偵測的部份是採用人工的方式來點出雙眼、唇部的位置，將來這部份可以改善為程式來判定。在人臉辨識的部份可以再增加辨識的特徵點作為改進，現在只有利用辨識者的眼睛到嘴唇的距離作為判斷依據，以後程式除了現有的條件之外，鼻子的大小、長度，鼻子與其他臉部特徵之間的相對位置、膚色，也可以列入判斷條件之一，讓辨識作業可以更加的精確。

# 參考文獻

[1] 陳繼棠，”結合膚色區域分割與主要成份分析於多人臉部辨識，”國立臺灣海洋大學機械與機電工程學系碩士學位論文，民國 95 年

[2] 林晁立，”以臉部器官形狀、寬度、相對位置從事人臉影像辨識，”私立東海大學資訊科學研究所碩士論文，民國 89 年

[3] 黃泰祥，”具備人臉追蹤與辨識功能的一個智慧型數位監視系統，”私立中原大學電子工程學系碩士學位論文，民國 93 年

[4] 江雅雯，”值基於色彩及紋理分析之鈔票偵測系統，”私立中原大學資訊工程學系碩士學位論文，民國 95 年

[5] 林明毅，李慶長，呂韶宜，”YCbCr 色彩空間之資訊隱藏應用系統研究，”國立臺北商業技術學院資訊管理系

[6] 維基百科，http://zh.wikipedia.org/

[7] 張學仁，”行政院新聞局視聽資料數位化之研究，”

[8] M. Soriano, B Martinkauppi, S. Huovinen, and M. Laaksonen, “Using the Skin Locus to Cope with Changing Illumination Conditions in Color-Based Face Tracking,” *Proc. IEEE Nordic Signal Processing Symposium,pp. 383-386,2000.*

[9] T. Sakai, M. Nagao,and T. Kanade, “Computer Analysis and Classification of Photographs of Human Faces,” *Proc. First USA-Japan Computer Conference,pp.2-7,1972.*

[10] I. Craw, H. Ellis, and J. R. Lishman, “Automatic Extraction of Face Feature,” *Pattern Recognition Lett ,pp.183-187, 1987.*

[11] 林群雄，吳建樺，” 使用多類支持向量機混合三角偵測做人臉辨識，” 國立台北大學資訊管理研究所

[12] “產學互動遠距教學訓練之實驗計

畫”http://cslin.auto.fcu.edu.tw/cvision/imagepro/index.html,
逢甲大學

# 附錄一

```
#include <vcl.h>
#pragma hdrstop
#include "Otsu.h"
#include "Color.h"
#include "Header.h"
#include "Bmp.h"
#include <math.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
     : TForm(Owner)
{
   obj = NULL;
   bOpen = false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
     if(OpenPictureDialog1->Execute()){

Image1->Picture->Bitmap->LoadFromFile(OpenPictureDialog1->FileName);
          bOpen = true;
     }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{/*
```

```
    計算 Histogram
    */
  long his[256],max;
  int i,x,y;
  Byte *ptr;


  for(i=0;i<256;i++)
    his[i] = 0;

     for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
     {
        ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
        for ( x = 0; x < Image1->Picture->Bitmap->Width; x++)
           his[ptr[x]]++;
     }

     max = his[0];
     for(i=1;i<256;i++)
           max = (max < his[i]) ? his[i] : max ;

     int h,w;
     h = Image1->Picture->Bitmap->Height;

     for(i=0;i<256;i++){
        Image2->Canvas->MoveTo(10+i,h-10);
        Image2->Canvas->LineTo(10+i,h-10-(his[i]*200)/max);
     }


  //利用 Chart 物件
  Series1->Clear();
  for(int i=0;i<256;i++)
        Series1->AddXY(i,his[i],"",clTeeColor);

}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm1::BitBtn3Click(TObject *Sender)
{
      int h,w;
      h = Image1->Picture->Bitmap->Height;
      Image2->Canvas->Pen->Color = clWhite;

      for(int i=0;i<256;i++){
         Image2->Canvas->MoveTo(10+i,h-10);
         Image2->Canvas->LineTo(10+i,0);
      }
      Image2->Canvas->Pen->Color = clBlack;
      Series1->Clear();
      Series2->Clear();
      Series3->Clear();
      Series4->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Histogram1Click(TObject *Sender)
{
    BitBtn1Click(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Exit1Click(TObject *Sender)
{
    exit(0);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Binary1Click(TObject *Sender)
{
    int x,y;
    Byte *ptr;
      for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
      {
```

```
            ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
            for ( x = 0; x < Image1->Picture->Bitmap->Width; x++)
               ptr[x] = (ptr[x] > 128 ) ? 255 : 0;
        }
        Image1->Refresh();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Open1Click(TObject *Sender)
{
    BitBtn2Click(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::nSobelClick(TObject *Sender)
{
    int ySobel[3][3] = {{-1,0,1},
                        {-2,0,2},
                        {-1,0,1}};
    int xSobel[3][3] = {{-1,-2,-1},
                        {0,0,0},
                        {1,2,1}};
    Byte *ptr = NULL;
    Byte *Img = NULL;
    Byte *xResult = NULL;
    Byte *yResult = NULL;

    int i,j,x,y,H,W,iTemp1,iTemp2;

    H = Image1->Picture->Bitmap->Height ;
    W = Image1->Picture->Bitmap->Width;

    Img = new Byte[H*W];
    xResult = new Byte[H*W];
    yResult = new Byte[H*W];
    for(i=0;i<H*W;i++){
        Img[i] = xResult[i] = yResult[i] = 0;
```

```
}

//將 image copy 到一維陣列

for(i=0;i< H; i++){
    ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
    for(j=0;j<W;j++)
        Img[i*W+j] = ptr[j];
}//end of for-i

for(i=1;i<H-1;i++){
    for(j=1;j<W-1;j++){
        iTemp1 = iTemp2 = 0;

        //開始執行 Sobel

        for(y=-1;y<=1;y++)
            for(x=-1;x<=1;x++){
                iTemp1 += Img[(i+y)*W+(j+x)] * xSobel[x+1][y+1];
                iTemp2 += Img[(i+y)*W+(j+x)] * ySobel[x+1][y+1];
            }//end of x-for
          xResult[i*W+j] = abs(iTemp1);
          yResult[i*W+j] = abs(iTemp2);
      }//end of for-j
}//end of for-i

//將結果放回 Image1 裡

for(i=0;i<H;i++){
    ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
    for(j=0;j<W;j++){
        iTemp1 = (xResult[i*W+j] <= 60) ? 255 : 0;
        iTemp2 = (yResult[i*W+j] <= 60) ? 255 : 0;
        ptr[j] = ((iTemp1 || iTemp2) == 0 ) ? 0 : 255;
    }//end of for-j
}//end of for-i
Image1->Refresh();
delete [] Img;
```

```
    delete [] xResult;
    delete [] yResult;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Smoothing1Click(TObject *Sender)
{
    int Mask[3][3] = {{1,1,1},
                      {1,1,1},
                      {1,1,1}};

    Byte *ptr = NULL;
    Byte *Img = NULL;
    Byte *Result = NULL;

    int i,j,x,y,H,W,iTemp1;

    H = Image1->Picture->Bitmap->Height ;
    W = Image1->Picture->Bitmap->Width;

    Img = new Byte[H*W];
    Result = new Byte[H*W];
    for(i=0;i<H*W;i++){
        Img[i] = Result[i]= 0;
    }


    //將 image copy 到一維陣列

    for(i=0;i< H; i++){
        ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
        for(j=0;j<W;j++)
            Img[i*W+j] = ptr[j];
    }//end of for-i

    for(i=1;i<H-1;i++){
        for(j=1;j<W-1;j++){
            iTemp1 = 0;
```

```
            //開始執行 Smoothing

            for(y=-1;y<=1;y++)
                for(x=-1;x<=1;x++){
                    iTemp1 += Img[(i+y)*W+(j+x)] * Mask[x+1][y+1];
                }//end of x-for
              Result[i*W+j] = (iTemp1/9);
          }//end of for-j
    }//end of for-i


    //將結果放回 Image1 裡

    for(i=0;i<H;i++){
        ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
        for(j=0;j<W;j++){
            ptr[j] = Result[i*W+j];
        }//end of for-j
    }//end of for-i
    Image1->Refresh();
    delete [] Img;
    delete [] Result;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::nSharpingClick(TObject *Sender)
{
    int Mask[3][3] = {{0,-1,0},
                      {-1,4,-1},
                      {0,-1,0}};

    Byte *ptr = NULL;
    Byte *Img = NULL;
    Byte *Result = NULL;

    int i,j,x,y,H,W,iTemp1;

    H = Image1->Picture->Bitmap->Height ;
```

```
W = Image1->Picture->Bitmap->Width;

Img = new Byte[H*W];
Result = new Byte[H*W];
for(i=0;i<H*W;i++){
    Img[i] = Result[i]= 0;
}


//將 image copy 到一維陣列

for(i=0;i< H; i++){
    ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
    for(j=0;j<W;j++)
        Img[i*W+j] = ptr[j];
}//end of for-i

for(i=1;i<H-1;i++){
    for(j=1;j<W-1;j++){
        iTemp1 = 0;

        //開始執行 Smoothing

        for(y=-1;y<=1;y++)
            for(x=-1;x<=1;x++){
                iTemp1 += Img[(i+y)*W+(j+x)] * Mask[x+1][y+1];
            }//end of x-for
          Result[i*W+j] = iTemp1;
       }//end of for-j
}//end of for-i


//將結果放回 Image1 裡

for(i=0;i<H;i++){
    ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
    for(j=0;j<W;j++){
        ptr[j] = ptr[j] + Result[i*W+j];
    }//end of for-j
}//end of for-i
```

```
    Image1->Refresh();
    delete [] Img;
    delete [] Result;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Save1Click(TObject *Sender)
{
    if(SavePictureDialog1->Execute())
        Image1->Picture->SaveToFile(SavePictureDialog1->FileName);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::EquationClick(TObject *Sender)
{
/*
    計算 Histogram
    */
  long his[256],max;
  int i,x,y,a,b;
  Byte *ptr;
  double dProb[256],dAcc = 0.0;


  for(i=0;i<256;i++)
    his[i] = 0;

     for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
     {
        ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
        for ( x = 0; x < Image1->Picture->Bitmap->Width; x++)
           his[ptr[x]]++;
     }

    for(i=0;i<256;i++)
        dProb[i] =
```

```
(double)his[i]/(Image1->Picture->Bitmap->Height*Image1->Picture->Bitmap
->Width);

    dAcc = 0.0;
    i = 0;
    while(dAcc<0.05){
        dAcc += dProb[i];
        i++;
    }
    a = i;
    dAcc = 0;
    i = 0;
    while(dAcc<0.95){
        dAcc += dProb[i];
        i++;
    }
    b = i;

     for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
     {
        ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
        for ( x = 0; x < Image1->Picture->Bitmap->Width; x++){
           if(ptr[x] < a) ptr[x] = 0;
           else if(ptr[x] >b ) ptr[x] = 255;
           else ptr[x] = (double)(ptr[x]-a)*255/(b-a);

        }
     }
    Image1->Refresh();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::HistEquationClick(TObject *Sender)
{
/*
    計算 Histogram
```

```
  */
  long his[256],max;
  int i,x,y,a,b;
  Byte *ptr;
  double dProb[256],dAcProb[256]= {0.0};


  for(i=0;i<256;i++)
   his[i] = 0;

    for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
    {
       ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
       for ( x = 0; x < Image1->Picture->Bitmap->Width; x++)
          his[ptr[x]]++;
    }

   for(i=0;i<256;i++)
       dProb[i] =
(double)his[i]/(Image1->Picture->Bitmap->Height*Image1->Picture->Bitmap
->Width);


   //累進機率

   dAcProb[0] = dProb[0];
   for(i=1;i<256;i++)
       dAcProb[i] = dProb[i] + dAcProb[i-1];


    for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
    {
       ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
       for ( x = 0; x < Image1->Picture->Bitmap->Width; x++){
          ptr[x] = 255 * dAcProb[ptr[x]];
       }
    }
   Image1->Refresh();
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::getRGB1Click(TObject *Sender)
{
/*
    計算 Histogram
    */
  struct RGB{
    unsigned char B;
    unsigned char G;
    unsigned char R;
  };
  long Rhis[256]={0},Ghis[256]={0},Bhis[256]={0};
  int i,x,y;
  RGB *ptrRGB;



    for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
    {
      ptrRGB = (RGB *)Image1->Picture->Bitmap->ScanLine[y];
      for ( x = 0; x < Image1->Picture->Bitmap->Width; x++){
        Rhis[ptrRGB[x].R]++;
        Ghis[ptrRGB[x].G]++;
        Bhis[ptrRGB[x].B]++;
      }
    }

  //利用 Chart 物件
  Series2->Clear();
  Series3->Clear();
  Series4->Clear();
  Series2->SeriesColor = clRed;
```

```
   Series3->SeriesColor = clGreen;
   Series4->SeriesColor = clBlue;
   for(int i=0;i<256;i++){
         Series2->AddXY(i,Rhis[i],"",clTeeColor);
         Series3->AddXY(i,Ghis[i],"",clTeeColor);
         Series4->AddXY(i,Bhis[i],"",clTeeColor);
   }
}
//---------------------------------------------------------------------------
/*
    此方法過濾掉紅色部分
*/
void __fastcall TForm1::RFilter1Click(TObject *Sender)
{
   struct RGB{
     unsigned char B;
     unsigned char G;
     unsigned char R;
   };
   int x,y;
   RGB *ptrRGB;

   for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
   {
      ptrRGB = (RGB *)Image1->Picture->Bitmap->ScanLine[y];
      for ( x = 0; x < Image1->Picture->Bitmap->Width; x++){
         ptrRGB[x].R = 0;
      }
   }
   Image1->Refresh();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ToGray1Click(TObject *Sender)
{
   struct RGB{
```

```
    unsigned char B;
    unsigned char G;
    unsigned char R;
  };
  int x,y;
  RGB *ptrRGB;

  for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
  {
     ptrRGB = (RGB *)Image1->Picture->Bitmap->ScanLine[y];
     for ( x = 0; x < Image1->Picture->Bitmap->Width; x++){
        ptrRGB[x].R = ptrRGB[x].G = ptrRGB[x].B =
(ptrRGB[x].R+ptrRGB[x].G+ptrRGB[x].B)/3.0;
     }
  }
  Image1->Refresh();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::getFlower1Click(TObject *Sender)
{
  struct RGB{
    unsigned char B;
    unsigned char G;
    unsigned char R;
  };
  int x,y;
  RGB *ptrRGB;

  for (y = 0; y < Image1->Picture->Bitmap->Height; y++)
  {
     ptrRGB = (RGB *)Image1->Picture->Bitmap->ScanLine[y];
     for ( x = 0; x < Image1->Picture->Bitmap->Width; x++){
        if(!((ptrRGB[x].R >= 200 &&  ptrRGB[x].R <=252) &&
(ptrRGB[x].G >= 175 &&  ptrRGB[x].G <=211)  && (ptrRGB[x].B >=
158 &&  ptrRGB[x].B <=188)))
              ptrRGB[x].R = ptrRGB[x].G = ptrRGB[x].B = 255;
```

```
      }
   }
   Image1->Refresh();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CC1Click(TObject *Sender)
{
   int i,j,iW,iH,iIndex=0;
   Byte **Img,*ptr;
   TStack *tStack = new TStack;
   TPoint *tPoint,*tLTop ,*tRBottom ;


   //複製到二維陣列

   iW = Image1->Picture->Bitmap->Width;
   iH = Image1->Picture->Bitmap->Height;
   Img = new Byte *[iH];
   for(i=0;i<iH;i++)
     Img[i] = new Byte[iW];
   for(i=0;i<iH;i++){
     ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
     for(j=0;j<iW;j++)
          Img[i][j] = (ptr[j] == 0) ? 1: 0;    //black --> 1, white --> 0
   }
   bool bFlag = false,bModified=true;
   tLTop = new TPoint(0,0);
   tRBottom = new TPoint(0,0);
   while(bModified){
          bModified = false;

          //先找到第一點黑點

          for(i=0;i<iH;i++){
            for(j=0;j<iW;j++){
                 if(Img[i][j] == 1){
                      Img[i][j] = 0;    //mark to white
                      bModified = true;
```

```
            bFlag = true;
            break;
        }
    }//end of for-j
    if(bModified)
        break;
}
if(bModified){
    tStack->Push(new TPoint(j,i));
    tLTop->x = j;
    tLTop->y = i;
    tRBottom->x = j;
    tRBottom->y = i;
}
else{ //找不到黑點了


     break;
}
//start the connected component
tPoint = (TPoint *)tStack->Pop();
bFlag = false;
while(!bFlag && bModified){
  for(i=tPoint->y-1;i<=tPoint->y+1;i++)
        for(j=tPoint->x-1;j<=tPoint->x+1;j++){
            if(i>=0 && i<iH && j>=0 && j<iW && Img[i][j] == 1){
                tStack->Push(new TPoint(j,i));
                Img[i][j] = 0;
                tLTop->x = (tLTop->x > j) ? j :    tLTop->x;
                tLTop->y = (tLTop->y > i) ? i :    tLTop->y;
                tRBottom->x = (tRBottom->x < j) ? j :    tRBottom->x;
                tRBottom->y = (tRBottom->y < i) ? i :    tRBottom->y;
                bModified = true;
            }//end of if
        }//end of for

    if(tStack->Count() > 0){
```

```
            tPoint = (TPoint *)tStack->Pop();
          }else
            bFlag = true;

        }//end of while
        //tStack->Free();

        //畫圖

        Image1->Canvas->Pen->Color = clRed;
        Image1->Canvas->MoveTo(tLTop->x,tLTop->y);
        Image1->Canvas->LineTo(tRBottom->x,tLTop->y);
        Image1->Canvas->LineTo(tRBottom->x,tRBottom->y);
        Image1->Canvas->LineTo(tLTop->x,tRBottom->y);
        Image1->Canvas->LineTo(tLTop->x,tLTop->y);

Image1->Canvas->TextOutA(tLTop->x-10,tLTop->y-10,IntToStr(iIndex+1));
        iIndex++;

   }//end of while-bModified
   //clear memory
   for(i=0;i<iH;i++)
     delete [] Img[i];
   delete [] Img;
   delete tLTop;
   delete tRBottom;
   tStack->Free();
}
//---------------------------------------------------------------------------
/*

   就先用 8 鄰居中，只要有一個方向非黑色，就將該點浸蝕

*/
void __fastcall TForm1::Erosion1Click(TObject *Sender)
{
   int x,y,h,w,i,j;
   TList *tList = new TList();
   TPoint *tP;
```

```
Byte **Img,*ptr;
bool bBreak;
h = Image1->Picture->Bitmap->Height;
w = Image1->Picture->Bitmap->Width;
Img = new Byte *[h];
for(i=0;i<h;i++)
  Img[i] = new Byte[w];

for(i=0;i<h;i++){
  ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
  for(j=0;j<w;j++)
       Img[i][j] = (ptr[j] == 0) ? 1: 0;    //black --> 1, white --> 0
}
for(y=1;y<h-1;y++){
  for(x=1;x<w-1;x++){
       if(Img[y][x] == 1) { //find the black point
            bBreak = false;
            for(i=y-1;i<=y+1;i++){
                 for(j=x-1;j<=x+1;j++){
                      if(Img[i][j] == 0){
                           tP = new TPoint(x,y);
                           tList->Add(tP);
                           bBreak = true;
                           break;
                      }//end of if
                 }//end of for-j
                 if(bBreak) break;
            }//end of for-i
       }//end of if Img[y][x] == 1
  }//end of for-x
}//end of for-y

//erosion
for(i=0;i<tList->Count;i++){
  tP = (TPoint *)tList->Items[i];
  Img[tP->y][tP->x] = 0;
}//end of for-i
```

```
    //copy to Image1
    for(y=0;y<h;y++){
      ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
      for(x=0;x<w;x++){
           ptr[x] = (Img[y][x] == 1) ? 0 : 255;
      }
    }
    Image1->Refresh();
    for(i=0;i<h;i++)
      delete [] Img[i];
    delete [] Img;
    tList->Clear();
}
//---------------------------------------------------------------------------
/*
    就先用 8 鄰居中，只要有一個方向非黑色，就將該點膨脹
*/
void __fastcall TForm1::Dilation1Click(TObject *Sender)
{
    int x,y,h,w,i,j;
    TList *tList = new TList();
    TPoint *tP;
    Byte **Img,*ptr;
    bool bBreak;
    h = Image1->Picture->Bitmap->Height;
    w = Image1->Picture->Bitmap->Width;
    Img = new Byte *[h];
    for(i=0;i<h;i++)
      Img[i] = new Byte[w];

    for(i=0;i<h;i++){
      ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
      for(j=0;j<w;j++)
           Img[i][j] = (ptr[j] == 0) ? 1: 0;    //black --> 1, white --> 0
    }
```

```
for(y=1;y<h-1;y++){
  for(x=1;x<w-1;x++){
       if(Img[y][x] == 0) { //find the white point
            bBreak = false;
            for(i=y-1;i<=y+1;i++){
                 for(j=x-1;j<=x+1;j++){
                      if(Img[i][j] == 1){
                           tP = new TPoint(x,y);
                           tList->Add(tP);
                           bBreak = true;
                           break;
                      }//end of if
                 }//end of for-j
                 if(bBreak) break;
            }//end of for-i
       }//end of if Img[y][x] == 1
  }//end of for-x
}//end of for-y

//dilation
for(i=0;i<tList->Count;i++){
  tP = (TPoint *)tList->Items[i];
  Img[tP->y][tP->x] = 1;
}//end of for-i

//copy to Image1
for(y=0;y<h;y++){
  ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
  for(x=0;x<w;x++){
       ptr[x] = (Img[y][x] == 1) ? 0 : 255;
  }
}
Image1->Refresh();
for(i=0;i<h;i++)
  delete [] Img[i];
delete [] Img;
tList->Clear();
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Ostu1Click(TObject *Sender)
{
     Otsu *obj = new Otsu();
     int **Img,h,w,i,j,x,y;
     Byte *ptr;
     h = Image1->Picture->Bitmap->Height;
     w = Image1->Picture->Bitmap->Width;
   Img = new int *[h];
       for(i=0;i<h;i++)
         Img[i] = new int[w];
   for(i=0;i<h;i++){
     ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[i];
     for(j=0;j<3*w;j+=3)
          Img[i][j/3] = ptr[j];
   }

   obj->Otsu_do(Img,h,w);
   //copy to Image1
   for(y=0;y<h;y++){
     ptr = (Byte *)Image1->Picture->Bitmap->ScanLine[y];
     for(x=0;x<3*w;x+=3){
          ptr[x] = Img[y][x];
          ptr[x+1] = Img[y][x];
          ptr[x+2] = Img[y][x];
     }
   }

   Image1->Refresh();
   for(i=0;i<h;i++)
     delete [] Img[i];
   delete [] Img;
     delete obj;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Openning1Click(TObject *Sender)
{
     Erosion1Click(Sender);
     Dilation1Click(Sender);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ToHSI1Click(TObject *Sender)
{
/*
     struct RGB{
          unsigned char B;
          unsigned char G;
          unsigned char R;
    };
*/
     obj = new ColorProcessing();
     RGB *ptr;
     int x,y,w,h,i;
     int iW,iH;
     float s;
     iH = Image1->Picture->Bitmap->Height;
     iW = Image1->Picture->Bitmap->Width;

     hsiArray = new HSI *[iH];
     for(y=0;y<iH;y++)
          hsiArray[y] = new HSI[iW];

     for(y=0;y<iH;y++){
          ptr = (RGB *)Image1->Picture->Bitmap->ScanLine[y];
          for(x=0;x<iW;x++){
               obj->getHSI(ptr[x].R,ptr[x].G,ptr[x].B,h,s,i);
               hsiArray[y][x].H = h;
               hsiArray[y][x].S = s;
               hsiArray[y][x].I = i;
          }//end of for-x
```

```
    }//end of for-y

}
//---------------------------------------------------------------------------


__fastcall TForm1::~TForm1()
{
/*
      int h =    Image1->Picture->Bitmap->Height;
      for(int i=0;i<h;i++)
         delete [] hsiArray[i];

      delete [] hsiArray;
*/
}


void TForm1::HSICC(int X, int Y)
{
   int i,j,iW,iH,iIndex=0;
   Byte **Img;
   RGB *ptr;
   TStack *tStack = new TStack;
   TPoint *tPoint,*tLTop ,*tRBottom ;


   int iPixFormat = Image1->Picture->Bitmap->PixelFormat;    //enum
{pfDevice, pf1bit, pf4bit, pf8bit, pf15bit, pf16bit, pf24bit, pf32bit,
pfCustom};

   //複製到二維陣列

   iW = Image1->Picture->Bitmap->Width;
   iH = Image1->Picture->Bitmap->Height;

   Image2->Picture->Bitmap->Width = Image1->Picture->Bitmap->Width;
   Image2->Picture->Bitmap->Height = Image1->Picture->Bitmap->Height;
```

```
    Image2->Picture->Bitmap->Assign(Image1->Picture->Bitmap);
/*
    Img = new Byte *[iH];
    for(i=0;i<iH;i++)
      Img[i] = new Byte[3*iW];

    for(i=0;i<iH;i++){
       ptr = (RGB *)Image1->Picture->Bitmap->ScanLine[i];
       for(j=0;j<iW;j++){
          Img[i][j] = ptr[j].R;
          Img[i][j+1] = ptr[j].G;
          Img[i][j+2] = ptr[j].R;
       }
    }
*/
    if(obj == NULL)
          obj = new ColorProcessing();
    int _h,_i;
    float _s;
    ptr = (RGB *)Image2->Picture->Bitmap->ScanLine[Y];
    obj->getHSI(ptr[X].R,ptr[X].G,ptr[X].B,_h,_s,_i);

    bool bFlag = false,bModified;
    tLTop = new TPoint(0,0);
    tRBottom = new TPoint(0,0);
    bModified=true ;
    while(bModified){
          bModified = false;

          //先找到第一點黑點

          ptr[X].R = ptr[X].G = ptr[X].B = 0;
          bModified = true;
          bFlag = true;


          if(bModified){
             tStack->Push(new TPoint(X,Y));
```

```
            tLTop->x = X;
            tLTop->y = Y;
            tRBottom->x = X;
            tRBottom->y = Y;
          }
          else{ //找不到黑點了


             break;
          }
          //start the connected component
          tPoint = (TPoint *)tStack->Pop();
          bFlag = false;
          while(!bFlag && bModified){
           for(i=tPoint->y-1;i<=tPoint->y+1;i++)
               for(j=tPoint->x-1;j<=tPoint->x+1;j++){
                   //if(i>=0 && i<iH && j>=0 && j<iW && HDist(i,j,_i)){
                   if(i>=0 && i<iH && j>=0 && j<iW &&
HDist(i,j,_h,_s,_i)){
                       tStack->Push(new TPoint(j,i));
                       ptr = (RGB *)Image2->Picture->Bitmap->ScanLine[i];
                       ptr[j].R = ptr[j].G = ptr[j].B = 0;
                       tLTop->x = (tLTop->x > j) ? j :    tLTop->x;
                       tLTop->y = (tLTop->y > i) ? i :    tLTop->y;
                       tRBottom->x = (tRBottom->x < j) ? j :    tRBottom->x;
                       tRBottom->y = (tRBottom->y < i) ? i :    tRBottom->y;
                       bModified = true;
                   }//end of if
               }//end of for

            if(tStack->Count() > 0){
               tPoint = (TPoint *)tStack->Pop();
            }else{
               bFlag = true;
               bModified = false;
            }//end of else
          }//end of while
```

```
        //tStack->Free();
        //畫圖
        Image1->Canvas->Pen->Color = clRed;
        Image1->Canvas->MoveTo(tLTop->x,tLTop->y);
        Image1->Canvas->LineTo(tRBottom->x,tLTop->y);
        Image1->Canvas->LineTo(tRBottom->x,tRBottom->y);
        Image1->Canvas->LineTo(tLTop->x,tRBottom->y);
        Image1->Canvas->LineTo(tLTop->x,tLTop->y);
   }//end of while-bModified
   //clear memory
/*
   for(i=0;i<iH;i++)
     delete [] Img[i];
   delete [] Img;
*/
   delete tLTop;
   delete tRBottom;
   tStack->Free();
}
void __fastcall TForm1::Image1MouseDown(TObject *Sender,
        TMouseButton Button, TShiftState Shift, int X, int Y)
{
   HSICC(X,Y);
}
//---------------------------------------------------------------------------


bool TForm1::HDist(int y, int x, int _h, float _s, int _i)
{
        int h,i;
        float s;
        bool bReturn = false;
        RGB *ptr;
        ptr = (RGB *)Image2->Picture->Bitmap->ScanLine[y];
        obj->getHSI(ptr[x].R,ptr[x].G,ptr[x].B,h,s,i);
        float fDist = 0.0;
```

```
        fDist = pow(h-_h,2) + pow(s- _s,2) + pow(i-_i,2);
        fDist = sqrt(fDist);
        if(!CheckBox1->Checked){
           if(abs(i-_i) < 20) //It's better for eye
              bReturn = true;
           else
              bReturn = false;
        }else{
           if(abs(h-_h) < 5 )
              bReturn = true;
           else
              bReturn = false;
        }
        return bReturn;
}
void __fastcall TForm1::Image1MouseMove(TObject *Sender, TShiftState
Shift,
          int X, int Y)
{
     int w,h;
     w = Image1->Picture->Bitmap->Width;
     h = Image1->Picture->Bitmap->Height;
     if(bOpen && (X > 0) && (X < w) && (Y >0) && (Y < h) ){
        int h,i;
        float s;
        RGB *ptr;
        ptr = (RGB *)Image1->Picture->Bitmap->ScanLine[Y];

        obj->getHSI(ptr[X].R,ptr[X].G,ptr[X].B,h,s,i);
        StatusBar1->Panels->Items[0]->Text =
"X:"+IntToStr(X)+"Y:"+IntToStr(Y);
        StatusBar1->Panels->Items[1]->Text = "RGB:" + IntToStr(ptr[X].R)+
"," + IntToStr(ptr[X].G) +    "," + IntToStr(ptr[X].B);
        StatusBar1->Panels->Items[2]->Text = "HSI:" + IntToStr(h) + "," +
FloatToStr(s) + "," + IntToStr(i);
     }
}
```