國立雲林科技大學

106學年度第二學期

**數位影像處理**

第二次平時作業

班 級：四電子三A

學 號：B10400001

學 生：郭鎧碩

授課教師：藍呂興 老師

繳交日期：107年05月14日

[一、 Rotation 2](#_Toc514095469)

[1. 題目說明 2](#_Toc514095470)

[2. 本程式之演算法 2](#_Toc514095471)

[3. 製作方法 3](#_Toc514095472)

[4. 函數程式碼 3](#_Toc514095473)

[5. 結果圖與原圖比較 4](#_Toc514095474)

[二、 Canny edge detector 6](#_Toc514095475)

[**1.** 題目說明 6](#_Toc514095476)

[**2.** 本程式的演算法 6](#_Toc514095477)

[3. 製作方法 6](#_Toc514095478)

[4. 函數程式碼(函數寫在mask.cpp) 9](#_Toc514095479)

[5. 結果圖與原圖比較 10](#_Toc514095480)

[三、 Hough transform for line detection 12](#_Toc514095481)

[**1.** 題目說明 12](#_Toc514095482)

[**2.** 本程式的演算法 12](#_Toc514095483)

[**3.** 製作方式 13](#_Toc514095484)

[**4.** 函數程式碼(函數寫在mask.c) 14](#_Toc514095485)

[5. 結果圖與原圖比較 15](#_Toc514095486)

[四、 心得 16](#_Toc514095487)

1. Rotation
2. 題目說明

我們利用下方方程式反矩陣來算出對應之原圖的座標，再利用內插法計算其內插點之數值。

且因為題目要求必須以圖片中心做旋轉，所以需要事先對圖做平移再計算，在後面會有更詳細的介紹與說明。

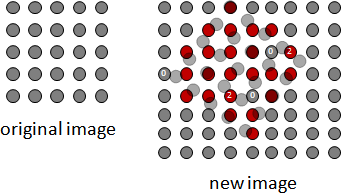
1. 本程式之演算法



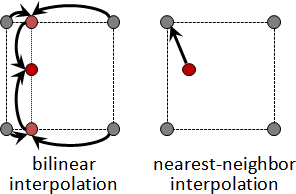
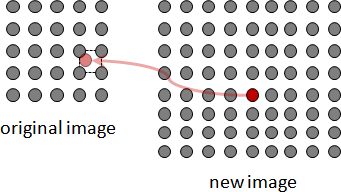
1. 製作方法

**先了解正向處理跟反向處理的差別**

正向處理（直接法）：窮舉原圖片每個像素的座標，一一變換，得到新座標。新座標四捨五入，讓像素座標是整數。然而，有些地方密集重疊、有些地方稀疏留白，無法構成圖片。這種方式行不通。



逆向處理（試誤法）：窮舉新圖片每個像素的座標，一一逆向變換，得到原座標。實施[二維內插](http://www.csie.ntnu.edu.tw/~u91029/Interpolation.html)，例如[近鄰內插（恰為四捨五入）](http://en.wikipedia.org/wiki/Nearest-neighbor_interpolation)、[雙線性內插](http://en.wikipedia.org/wiki/Bilinear_interpolation)，求得理想的像素數值。



從上述可以了解逆向處理較正向處理好。

**原點在哪裡**

需要注意的是程式碼會以圖 (0, 0) 作為定點轉換，如果需要從中心點旋轉或是任點轉需要轉換一下座標。

假如現在要以 (10, 10) 為中心旋轉，那麼在旋轉前你需要先把for迴圈扣調這個範圍，也就是從 (j, i) 要從負號 -10 開始跑，經過旋轉公式計算出來之後還需要將原本的 10 加回來，這樣就可以獲得正確的位置了。

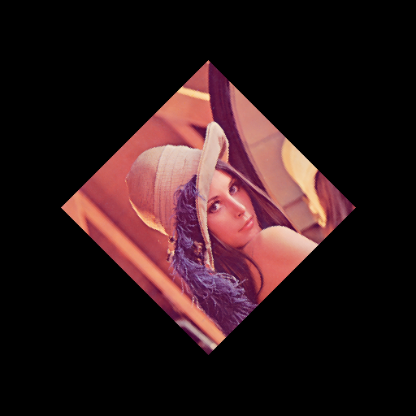
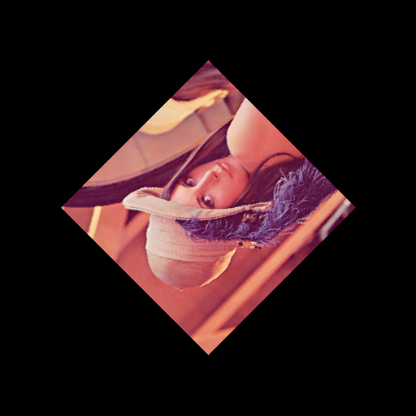
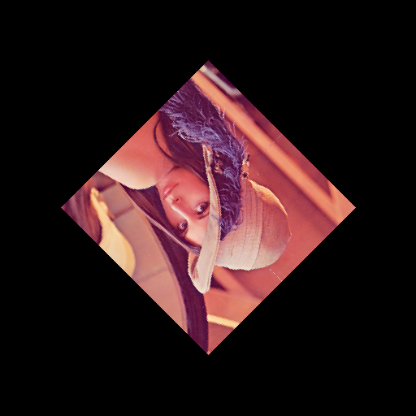
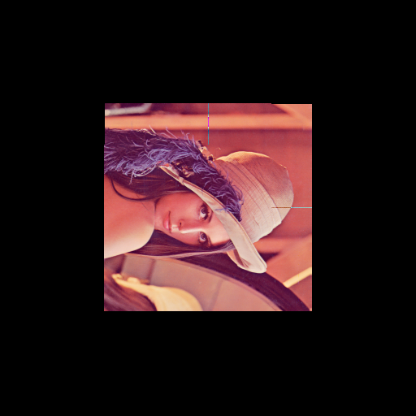
此外獲得正確位置之後還有一個問題就是旋轉後的圖比旋轉前的圖還要大，勢必會計算出超出原圖範圍的點，跑一個if過濾負號與超出原圖長寬的點即可。

1. 函數程式碼

Rotate (輸出矩陣，輸入矩陣，圖長，圖高，旋轉角度)

|  |
| --- |
| void Rotate(int r[1024][1024], int R[1024][1024], int& width, int& height, float angle) {  int i, j;  double x, y;  int p1, p2;  int half\_width = width / 2; //中心點座標 = 長and高 / 2;  int half\_height = height/ 2;  int after\_i, after\_j;  for (i = 0; i <= width; i++)  for (j = 0; j <= height; j++) {  after\_i = i - half\_width; //先對圖片做平移，使中心點在原點  after\_j = j - half\_height;  x = after\_i \* cos(angle\*M\_PI / 180) + after\_j \* sin(angle\*M\_PI / 180) + half\_width; //記得加一半長與高  y = after\_i \* sin(angle\*M\_PI / 180)\*(-1) + after\_j \* cos(angle\*M\_PI / 180) + half\_height;  if (x>0 && x<width && y>0 && y<height) { //判斷是否對應到之原圖有像素點  p1 = R[int(x)][int(y)] \* abs((int) y - int(y + 1)) +  R[int(x)][int(y + 1)] \* abs((int) y - int(y));  p2 = R[int(x + 1)][int(y)] \* abs((int) y - int(y + 1)) +  R[int(x + 1)][int(y + 1)] \* abs((int) y - int(y));  r[i][j] = p1 \* abs((int) x - int(x + 1)) + p2 \* abs((int) x - int(x));  }  }  } |

1. 結果圖與原圖比較



1. Canny edge detector
   1. 題目說明

邊緣檢測的目的是標識圖像中亮度變化明顯的點。邊緣檢測方法首先計算邊緣強度，通常用一階導數表示，例如梯度。但Canny引入了非最大抑制概念，它是說邊緣定義為在梯度方向具有最大梯度值的點。這點使他是最常用的邊緣檢測方法

* 1. 本程式的演算法



* 1. 製作方法

Gaussian filter

原圖可能有雜訊，所以先對原圖做高斯濾波

|  |
| --- |
| void Gaussian\_LPF(int r[1024][1024], int R[1024][1024], int& width, int& height, float sigma) {  int i, j;  float Gaussian\_kernel[9],total=0;  int count=0;  //高斯矩陣  for (i = 0; i < 3;i++) {  for (j = 0;j < 3;j++) {  Gaussian\_kernel[count] = exp((-pow(abs(i - 1), 2) - pow(abs(j - 1), 2)) / (2 \* sigma \* sigma));  total += Gaussian\_kernel[count];  count++;  }  }  for (count = 0; count < 9;count++) {  Gaussian\_kernel[count] /= total;  }  convolution(r, R, Gaussian\_kernel, width, height);  } |

第二是找出各個點的梯度，只要對圖做sobel即可得到

|  |
| --- |
| float Gx[] = { -1, 0, 1,  -2, 0, 2, //x方向  -1, 0, 1 };  convolution(after\_x, r, Gx, width, height);  float Gy[] = { 1, 2, 1,  0, 0, 0, //y方向  -1,-2,-1 };  convolution(after\_y, r, Gy, width, height);  for(int x=1;x<width;x++)  for (int y = 1;y < height;y++)  r[x][y] = hypot(after\_x[x][y], after\_y[x][y]); |

剛剛X,Y的值，用atan()算出角度，再根據此角度找出方向上最大值

|  |
| --- |
| for(int x=0;x < width;x++)  for (int y = 0;y < height;y++) {  float dir = (float)(fmod(atan2(after\_y[x][y], after\_x[x][y]) + M\_PI,M\_PI) / M\_PI) \* 8;  if (((dir <= 1 || dir > 7) && r[x][y] > r[x][y + 1] && r[x][y] > r[x][y - 1]) ||  ((dir > 1 && dir <= 3) && r[x][y] > r[x - 1][y + 1] && r[x][y] > r[x + 1][y - 1]) ||  ((dir > 3 && dir <= 5) && r[x][y] > r[x + 1][y] && r[x][y] > r[x - 1][y]) ||  ((dir > 5 && dir <= 7) && r[x][y] > r[x + 1][y + 1] && r[x][y] > r[x - 1][y - 1]))  nms[x][y] = r[x][y];  else  nms[x][y] = 0;  } |

最後再透過閘值來畫軌跡

|  |
| --- |
| for (int x = 0;x<width;x++)  for (int y = 0;y < height;y++) {  if (nms[x][y] >= tmax && r[x][y] == 0) { // trace edges  r[x][y] = MAX\_BRIGHTNESS;  int nedges = 1;  edges[0] = c;  do {  nedges--;  int t = edges[nedges];  int nbs[8]; //// neighbours  nbs[0] = t - width; // nn  nbs[1] = t + width; // ss  nbs[2] = t + 1; // ww  nbs[3] = t - 1; // ee  nbs[4] = nbs[0] + 1; // nw  nbs[5] = nbs[0] - 1; // ne  nbs[6] = nbs[1] + 1; // sw  nbs[7] = nbs[1] - 1; // se  for (int i = -1; i < 2; i++)  for (int j = -1; j < 2; j++)  if (i == 0 && j == 0)  continue;  else  if (nms[x + i][y + j] >= tmin && r[x + i][y + j] == 0) {  r[x + i][y + j] = MAX\_BRIGHTNESS;  edges[nedges] = nbs[3\*(i+1)+(j+1)];  nedges++;  }  } while (nedges > 0);  }  c++;  } |

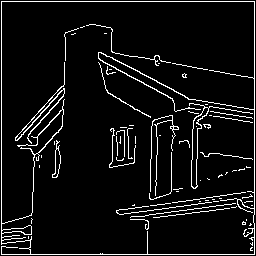
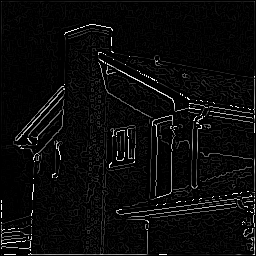
* 1. 函數程式碼(函數寫在mask.cpp)

Rotate (輸出矩陣，輸入矩陣，圖長，圖高，最小閘值，最大閘值，高斯濾波的變數)

|  |
| --- |
| void canny\_edge\_detection(int r[1024][1024], int R[1024][1024], int& width, int& height, int tmin, int tmax, float sigma) {    int after\_x[1024][1024] = { 0 }, after\_y[1024][1024] = { 0 };  int nms[1024][1024] = { 0 };  //做高斯低通濾波  Gaussian\_LPF(r, R, width, height, sigma);  //save\_bmp((char\*)"picture\\house\_step1.bmp", r, r, r);  //尋找圖像中的亮度梯度 對圖做sobel  float Gx[] = { -1, 0, 1,  -2, 0, 2,  -1, 0, 1 };  convolution(after\_x, r, Gx, width, height);  float Gy[] = { 1, 2, 1,  0, 0, 0,  -1,-2,-1 };  convolution(after\_y, r, Gy, width, height);  for(int x=1;x<width;x++)  for (int y = 1;y < height;y++)  r[x][y] = hypot(after\_x[x][y], after\_y[x][y]);  //save\_bmp((char\*)"picture\\house\_step2.bmp", r, r, r);    // Non-maximum suppression  for(int x=0;x < width;x++)  for (int y = 0;y < height;y++) {  float dir = (float)(fmod(atan2(after\_y[x][y], after\_x[x][y]) + M\_PI,M\_PI) / M\_PI) \* 8; //atan2() return -pi~pi ,so add pi  if (((dir <= 1 || dir > 7) && r[x][y] > r[x][y + 1] && r[x][y] > r[x][y - 1]) ||  ((dir > 1 && dir <= 3) && r[x][y] > r[x - 1][y + 1] && r[x][y] > r[x + 1][y - 1]) || ((dir > 3 && dir <= 5) && r[x][y] > r[x + 1][y] && r[x][y] > r[x - 1][y]) ||  ((dir > 5 && dir <= 7) && r[x][y] > r[x + 1][y + 1] && r[x][y] > r[x - 1][y - 1]))  nms[x][y] = r[x][y];  else  nms[x][y] = 0;  }  //save\_bmp((char\*)"picture\\house\_step3.bmp", nms, nms, nms);  //歸零  for (int x = 0;x < width;x++)  for (int y = 0;y < height;y++)  r[x][y] = 0;  int edges[1024 \* 1024] = { 0 };  int c = 1;  //在圖像中跟蹤邊緣  for (int x = 0;x<width;x++)  for (int y = 0;y < height;y++) {  if (nms[x][y] >= tmax && r[x][y] == 0) { // trace edges  r[x][y] = MAX\_BRIGHTNESS;  int nedges = 1;  edges[0] = c;  do {  nedges--;  int t = edges[nedges];  int nbs[8]; //// neighbours  nbs[0] = t - width; // nn  nbs[1] = t + width; // ss  nbs[2] = t + 1; // ww  nbs[3] = t - 1; // ee  nbs[4] = nbs[0] + 1; // nw  nbs[5] = nbs[0] - 1; // ne  nbs[6] = nbs[1] + 1; // sw  nbs[7] = nbs[1] - 1; // se  for (int i = -1; i < 2; i++)  for (int j = -1; j < 2; j++)  if (i == 0 && j == 0)  continue;  else  if (nms[x + i][y + j] >= tmin && r[x + i][y + j] == 0) {  r[x + i][y + j] = MAX\_BRIGHTNESS;  edges[nedges] = nbs[3\*(i+1)+(j+1)];  nedges++;  }  } while (nedges > 0);  }  c++;  }  save\_bmp((char\*)"picture\\camera\_step4.bmp", r, r, r);  } |

* 1. 結果圖與原圖比較

Tmin=50, Tmax=80, sigma=1



原圖

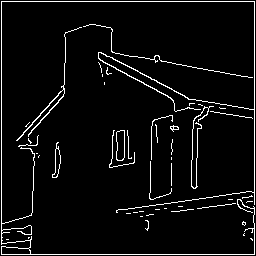
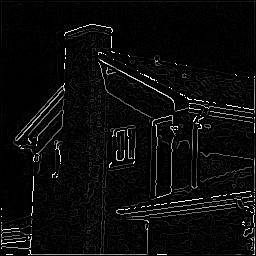
高斯濾波後

sobel

非最大值抑制

結果圖

Tmin=100, Tmax=150, sigma=1



可以看出在修改閘值後 有些邊緣線就沒辦法顯示出來。

1. Hough transform for line detection
   1. 題目說明

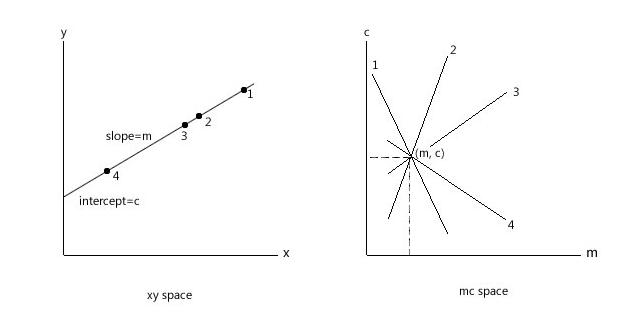
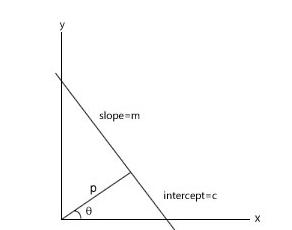
霍夫變換是用來辨別找出物件中的特徵，例如：線條。他的**演算法**流程大致如下，給定一個物件、要辨別的形狀的種類，演算法會在參數空間(parameter space)中執行投票來決定物體的形狀， 而這是由累加空間(accumulator space)裡的局部最大值(local maximum)來決定。

* 1. 本程式的演算法



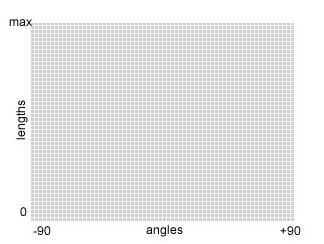
* 1. 製作方式

以理論而言，在圖像中檢測直線的問題，其實質是找到構成直線的所有的像素點。那麼問題就是從找到直線，變成找到符合y=mx+c(左下圖)的所有(x,y)的點的問題。但若是這樣m，c有無窮大的問題，所以改用右下的方式

1. 將θ角在-90度到90度的范圍裡，劃分為很多區間，對所有的像素點(x,y)在所有θ角的時候，求出ρ.從而累加ρ值出現的次數。高於某個閾值的ρ就是一個直線。

2. 這個過程就類似於如下一個二維的表格，橫坐標就是θ角，ρ就是到直線的最短距離。



透過下方的程式碼，我們可僅走一次圖片並取得各特徵點在各角度的值，並累加至Accumulator矩陣中。

|  |
| --- |
| for (i = 2; i < width - 2; i++)  for (j = 2; j < height- 2; j++) {  if (R[i][j] == 255) {  for (angle = -90; angle < 91; angle++) {  angle\_new = angle \* M\_PI / 180;  s = round(i \* cos(angle\_new) + j \* sin(angle\_new));  Accumulator[s][angle+90]++;  }  }  else  continue;  }  } |

最後透過這程式碼，尋找Accumulator中符合我們所要的s，angle並顯示在圖中。

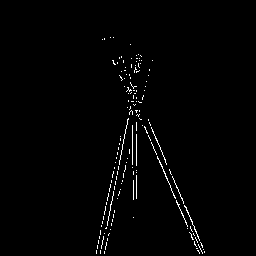
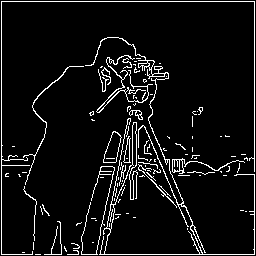
|  |
| --- |
| for (s = 0; s < 800; s++) {  for (angle = -90; angle < 91; angle++) {  if (Accumulator[s][angle+90] > 80) {  angle\_new = angle \* M\_PI / 180;  for (i = 2; i < width-2; i++) {  for (j = 2; j < height-2; j++)  if ( (s == round(i \* cos(angle\_new) + j \* sin(angle\_new))) && R[i][j]==255)  r[i][j] = 255;  }  }  else  continue;  }  } |

* 1. 函數程式碼(函數寫在mask.c)

Hough\_Line\_Transform (輸出矩陣，輸入矩陣，圖長，圖高)

|  |
| --- |
| void Hough\_Line\_Transform(int r[1024][1024], int R[1024][1024], int& width, int& height) {  int Accumulator[800][181] = { 0 };  int s,i,j;  int angle;  double angle\_new;  for (i = 2; i < width - 2; i++)  for (j = 2; j < height- 2; j++) {  if (R[i][j] == 255) {  for (angle = -90; angle < 91; angle++) {  angle\_new = angle \* M\_PI / 180;  s = round(i \* cos(angle\_new) + j \* sin(angle\_new));  Accumulator[s][angle+90]++;  }  }  else  continue;  }  for (s = 0; s < 800; s++) {  for (angle = -90; angle < 91; angle++) {  if (Accumulator[s][angle+90] > 80) {  angle\_new = angle \* M\_PI / 180;  for (i = 2; i < width-2; i++) {  for (j = 2; j < height-2; j++)  if ( (s == round(i \* cos(angle\_new) + j \* sin(angle\_new))) && R[i][j]==255)  r[i][j] = 255;  }  }  else  continue;  }  }  } |

* 1. 結果圖與原圖比較



原圖

Canny

Hough line detector

1. 心得

旋轉圖片一開始不知道如何做出在中心點做旋轉，幸好後來在網路上有找到如何在圖中某個點做旋轉的演算法，才成功做了出來。旋轉後的圖在色彩上面有些許改變，我想可能是因為彩色圖由RGB所構成，做內插法時有四捨五入的影響才會如此。

在Canny edge detector中我有寫convolution()來讓主程式比較有可觀性，一二三步驟其實都還滿容易的，最後一個步驟畫軌跡比較困難些，有想到兩種方法，一個需要比較多的時間透過重複尋找大於最小閘值的點，另一個則是建一個矩陣只需要掃描一次圖片即可。

Hough transform只要搞懂如果很多點在同一直線上，他們在某個r，某個角度會是一樣這個觀念就很容易做出來，加上canny的結果圖我個人還滿意的，所以偵測直線就很輕鬆。