**機器視覺**

HW 2

資工三 109590049 林敬翰

1. **Convert the color image to a binary image.**

Code：

|  |
| --- |
| Mat ConvertToGray(Mat img) {  Mat gray = Mat::zeros(img.size(), CV\_8UC1);  for (int i = 0; i < img.rows; i++) {  for (int j = 0; j < img.cols; j++) {  Vec3b rgb = img.at<Vec3b>(i, j);  gray.at<uchar>(i, j) = 0.3 \* rgb[2] + 0.59 \* rgb[1] + 0.11 \* rgb[0];  }  }  return gray;  }  Mat ConvertToBinary(Mat img, int threshold) {  Mat grayimg = ConvertToGray(img);  Mat binary = Mat::zeros(grayimg.size(), CV\_8UC1);  for (int i = 0; i < grayimg.rows; i++) {  for (int j = 0; j < grayimg.cols; j++) {  uchar n = grayimg.at<uchar>(i, j);  if (n > threshold) {  binary.at<uchar>(i, j) = 0;  }  else {  binary.at<uchar>(i, j) = 1;  }  }  }  return binary;  }  Mat BinaryReConstruct(Mat img) {  Mat binary = Mat::zeros(img.size(), CV\_8UC1);  for (int i = 0; i < img.rows; i++) {  for (int j = 0; j < img.cols; j++) {  uchar n = img.at<uchar>(i, j);  if (n == 1) {  binary.at<uchar>(i, j) = 255;  }  else {  binary.at<uchar>(i, j) = 0;  }  }  }  return binary;  } |

將圖片轉成二值化沿用了上次HW1寫出來的灰階轉換、二值化轉換函式，但為了方便計算，將數值將二值化轉換函式的數值從255改成了1，後面輸出binary image時用了BinaryReConstruct函式再將數值轉為255來做圖片輸出。

1. **4-connected.**

|  |
| --- |
| Mat FourlabelConnect(Mat img, int threshold) {  Mat binary = ConvertToBinary(img, threshold);  Mat label=Mat::zeros(binary.size(),CV\_32SC1);  binary.convertTo(label, CV\_32SC1);  int counter = 1;  int propCount = 0;  int temp = 1;  for (int i = 1; i < binary.rows - 2; i++) {  int\* data = label.ptr<int>(i);  for (int j = 1; j < binary.cols - 2; j++) {  if (data[j] == 1) {  std::stack<std::pair<int, int>> neighbor;  neighbor.push(std::pair<int, int>(i, j));  ++counter;  while (!neighbor.empty()) {  std::pair<int, int> cur = neighbor.top();  int curX = cur.first;  int curY = cur.second;  label.at<int>(curX, curY) = counter;    neighbor.pop();  if (curY != 0) {  if (label.at<int>(curX, curY - 1) == 1) {  neighbor.push(std::pair<int, int>(curX, curY - 1));  temp++;  }  }  if (curY != binary.cols - 1) {  if (label.at<int>(curX, curY + 1) == 1) {  neighbor.push(std::pair<int, int>(curX, curY + 1));  temp++;  }  }  if(curX!=0){  if (label.at<int>(curX-1, curY) == 1) {;  neighbor.push(std::pair<int, int>(curX - 1, curY));  temp++;  }  }  if (curX != binary.rows - 1) {  if (label.at<int>(curX+1, curY) == 1) {  neighbor.push(std::pair<int, int>(curX + 1, curY));  temp++;  }  }  }  if (temp >= 100) {  propCount++;  }  temp = 0;  }  }  }  Mat colorLabel;  std::vector<Vec3b> colors;  for (int i = 0; i < label.rows \* label.cols; i++) {  colors.push\_back(Vec3b(rand() % 256, rand() % 256, rand() % 256));  }  colorLabel = Mat::zeros(label.size(), CV\_8UC3);  for (int i = 0; i < colorLabel.rows; i++) {  for (int j = 0; j < colorLabel.cols; j++) {  int labelValue = label.at<int>(i, j);  if (labelValue > 0) {  colorLabel.at<Vec3b>(i, j) = colors[labelValue - 1];  }  }  }  int num = counter - 1;  printf("count：%d\n", propCount);  return colorLabel;  } |

在4連通中，我使用了seed-filling演算法。用迴圈去跑每個pixel，如果該pixel是1且沒被標記過將這點儲存到一個stack中並給予個新的標籤，然後跑4連通的鄰近pixel並將鄰近的pixel是一的儲存到stack中直到該區域所有的1都被跑完。最後用亂數產生一個顏色映射表，將label中的值都替換成新的顏色。

1. **8-connected.**

|  |
| --- |
| Mat EightlabelConnect(Mat img, int threshold) {  Mat binary = ConvertToBinary(img, threshold);  Mat label = Mat::zeros(binary.size(), CV\_32SC1);  binary.convertTo(label, CV\_32SC1);  int counter = 1;  int propCount = 0;  int temp=1;  for (int i = 1; i < binary.rows - 2; i++) {  int\* data = label.ptr<int>(i);  for (int j = 1; j < binary.cols - 2; j++) {  if (data[j] == 1) {  std::stack<std::pair<int, int>> neighbor;  neighbor.push(std::pair<int, int>(i, j));  ++counter;  while (!neighbor.empty()) {  std::pair<int, int> cur = neighbor.top();  int curX = cur.first;  int curY = cur.second;  label.at<int>(curX, curY) = counter;  neighbor.pop();  for (int x = -1; x <= 1; x++) {  for (int y = -1; y <= 1; y++) {  if (x == 0 && y == 0) {  continue;  }  int x2 = curX + x;  int y2 = curY + y;  if (x2 >= 0 && x2 < binary.rows && y2 >= 0 && y2 < binary.cols) {  if (label.at<int>(x2, y2) == 1) {  neighbor.push(std::pair<int, int>(x2, y2));  temp++;  }  }  }  }  }  if (temp >= 100) {  propCount++;  }  temp = 0;  }  }  }  Mat colorLabel;  std::vector<Vec3b> colors;  for (int i = 0; i < label.rows \* label.cols; i++) {  colors.push\_back(Vec3b(rand() % 256, rand() % 256, rand() % 256));  }  colorLabel = Mat::zeros(label.size(), CV\_8UC3);  for (int i = 0; i < colorLabel.rows; i++) {  for (int j = 0; j < colorLabel.cols; j++) {  int labelValue = label.at<int>(i, j);  if (labelValue > 0) {  colorLabel.at<Vec3b>(i, j) = colors[labelValue - 1];  }  }  }  int num = counter - 1;  printf("count：%d\n", propCount);  return colorLabel;  } |

在8連通中，我一樣使用了seed-filling演算法。唯一的差別是在偵測鄰近pixel時變成偵測鄰近8格。所以我用了多兩層的迴圈來去跑附近(x±1, y±1)的範圍。

然而再計算有幾個物件時，由於轉換成二值化影像後還是會有一些小雜訊在，所以我只追蹤了大於100pixel的物件，剛好seed-filling演算法很好計算這個。

1. **Output**

轉換成二值化使用的threshold值：

1.png 100

2. png 232

3. png 90

一張含有 文字 的圖片

自動產生的描述4. png 230

數量輸出：

圖片輸出：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 4-connect | 8-connect | Binary |
| 1.png |  |  |  |
| 總數：39 | 總數：39 |  |
| 2.png |  |  |  |
| 總數：27 | 總數：27 |  |
| 3.png |  |  |  |
| 總數：9 | 總數：9 |  |
| 4.png |  |  |  |
| 總數：26 | 總數：23 |  |