Advancde computer vision

高等電腦視覺

111618018 自動化碩一 吳祐毅

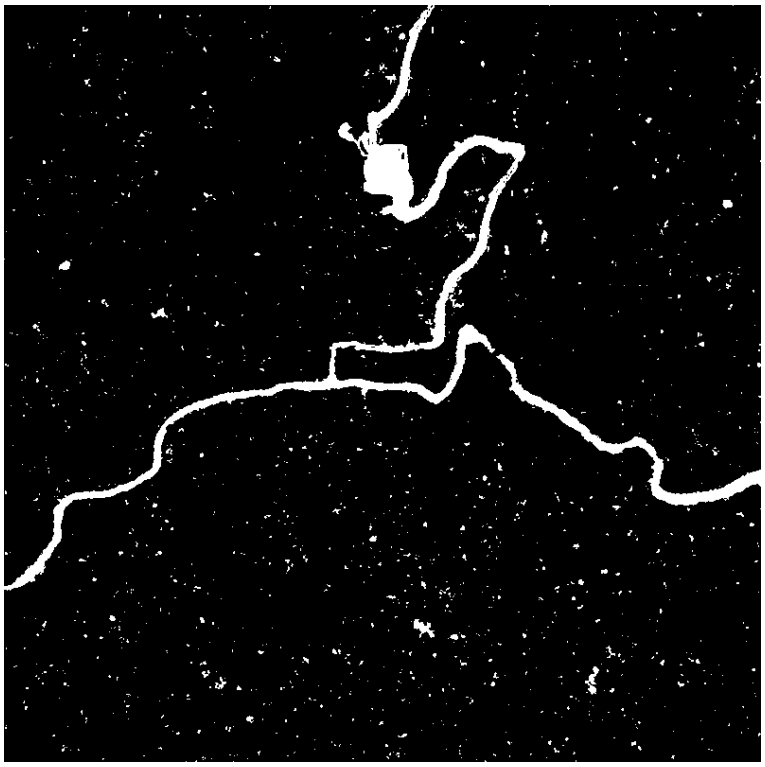
Homework 2

一、 C 語言影像處理(without other library)

1. Binarizing

先將 rgb 轉成灰階，再將灰階透過閥值判斷來決定該像素值為 0 或 255，在這裡閥值設定為 110。

```
int gray_num = 0;
for(int i = 0; i < height; i++){     //rgb > 灰階 > 二值化
    for(int j = 0; j < width; j++){
        gray_num = image_ori[(i * width + j) * 3 + 0] * 0.3 +
                   image_ori[(i * width + j) * 3 + 1] * 0.59 +
                   image_ori[(i * width + j) * 3 + 2] * 0.11;
        if(gray_num > 110){
            gray_num = 255;
        }else{
            gray_num = 0;
        }
        image_bin[(i * width + j) * 3 + 0] = gray_num;
        image_bin[(i * width + j) * 3 + 1] = gray_num;
        image_bin[(i * width + j) * 3 + 2] = gray_num;
    }
}
```



2. Morphology

先設定 kernel 核函數大小以及陣列裡每個值為 255，再將核函數去跟每個像素及該像素周圍去比較，在侵蝕的部分，如果比較的像素及周圍都有滿足核函數的像素值，則輸出為 255，若其中一項不滿足，則輸出為 0。

```
kernel_size_step = kernel_size3;
int kernel_step1[kernel_size_step][kernel_size_step];
for(int i = 0; i < kernel_size_step; i++){
    for(int j = 0; j < kernel_size_step; j++){
        kernel_step1[i][j] = 255;
    }
}
for(int i = 0; i < height; i++){
    for(int j = 0; j < width; j++){
        out_num = 255;
        for(int s = -1 * (kernel_size_step / 2); s < (kernel_size_step / 2) + 1; s++){
            for(int t = -1 * (kernel_size_step / 2); t < (kernel_size_step / 2) + 1; t++){
                if ((i + s) < 0 || (i + s) >= height || (j + t) < 0 || (j + t) >= width){ //邊緣判斷
                    compare_num = 255;
                }else{
                    compare_num = image_bin[((i + s) * width + (j + t)) * 3 + 0];
                }
                if(compare_num != kernel_step1[s + (kernel_size_step / 2)][t + (kernel_size_step / 2)]){
                    out_num = 0;
                }
            }
        }
        image_ero[(i * width + j) * 3 + 0] = out_num;
        image_ero[(i * width + j) * 3 + 1] = out_num;
        image_ero[(i * width + j) * 3 + 2] = out_num;
```

在膨脹的部分，如果比較的像素及周圍有任一項滿足核函數的像素值，則輸出為 255，若全部都不滿足，則輸出為 0。

```
kernel_size_step = kernel_size7;
int kernel_step2[kernel_size_step][kernel_size_step];
for(int i = 0; i < kernel_size_step; i++){
    for(int j = 0; j < kernel_size_step; j++){
        kernel_step2[i][j] = 255;
    }
}
for(int i = 0; i < height; i++){
    for(int j = 0; j < width; j++){
        out_num = 0;
        for(int s = -1 * (kernel_size_step / 2); s < (kernel_size_step / 2) + 1; s++){
            for(int t = -1 * (kernel_size_step / 2); t < (kernel_size_step / 2) + 1; t++){
                if ((i + s) < 0 || (i + s) >= height || (j + t) < 0 || (j + t) >= width){ //邊緣判斷
                    compare_num = 0;
                }else{
                    compare_num = image_ero[((i + s) * width + (j + t)) * 3 + 0];
                }
                if(compare_num == kernel_step2[s + (kernel_size_step / 2)][t + (kernel_size_step / 2)]){
                    out_num = 255;
                }
            }
        }
        image_dil[(i * width + j) * 3 + 0] = out_num;
        image_dil[(i * width + j) * 3 + 1] = out_num;
        image_dil[(i * width + j) * 3 + 2] = out_num;
    }
```

形態學部分順序為透過二值化影像先侵蝕一次，再膨脹兩次，最後在侵蝕一次。

3. connected component

先建立一個與圖片大小相同的陣列，此陣列用來表示每個像素的標籤編號，由於是要判斷黑色區域的物件，所以將黑色區域標籤先設為 0，白色區域則設為-1。

```
for(int i = 0; i < height; i++){
    for(int j = 0; j < width; j++){
        if(image_mor[(i * width + j) * 3 + 0] == 0){
            label_array[i * width + j] = 0;
        }
        if(image_mor[(i * width + j) * 3 + 0] == 255){
            label_array[i * width + j] = -1;
        }
    }
}
int label_count = 0;
```

在連通元件算法的部分一共執行了 4 次，此 4 次執行的方向分別為左上往右下、右上往左下、左下往右上、右下往左上，第一次執行先判斷目標像素左邊及上面是否為黑色區域，若是的話則複製鄰近相同的標籤值，若不是黑色則給予新的標籤。

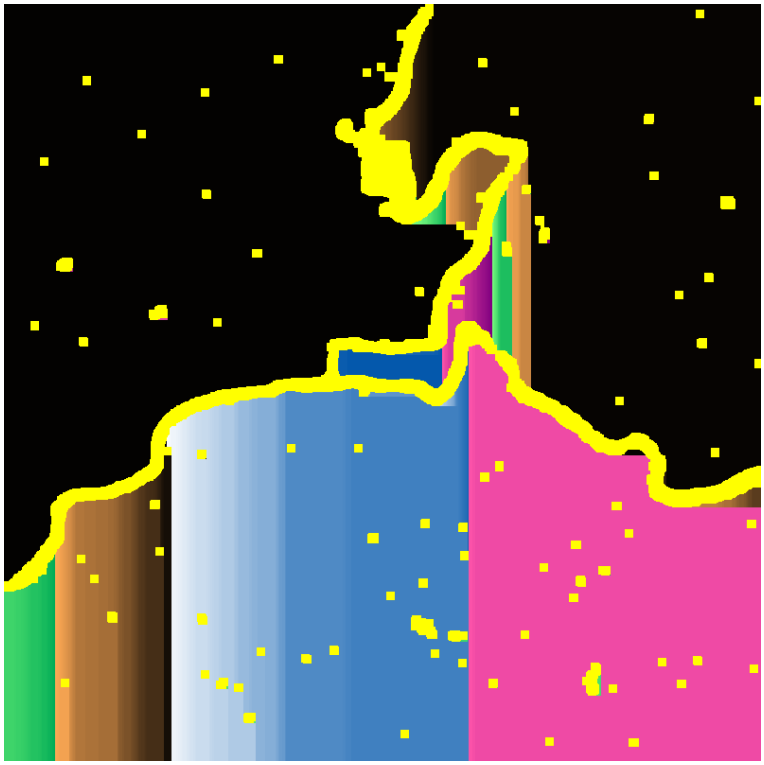
```
//第一次 左上往右下
for(int i = (height - 1); i >= 0; i--){
    for(int j = 0; j < width; j++){
        if(image_mor[(i * width + j) * 3 + 0] == 0){
            if((i + 1) < height && (j - 1) >= 0){
                if(image_mor[(i * width + (j - 1)) * 3 + 0] == 255 && image_mor[((i + 1) * width + j) * 3 + 0] == 255
                    label_count ++;
                    label_array[i * width + j] = label_count;
                }
                if(image_mor[((i + 1) * width + j) * 3 + 0] == 255 && image_mor[(i * width + (j - 1)) * 3 + 0] == 0){
                    label_array[i * width + j] = label_array[i * width + (j - 1)];
                }
                if(image_mor[((i + 1) * width + j) * 3 + 0] == 0 && image_mor[(i * width + (j - 1)) * 3 + 0] == 255){
                    label_array[i * width + j] = label_array[(i + 1) * width + j];
                }
                if(image_mor[((i + 1) * width + j) * 3 + 0] == 0 && image_mor[(i * width + (j - 1)) * 3 + 0] == 0){
                    if(label_array[(i + 1) * width + j] > label_array[i * width + (j - 1)]){
                        label_array[i * width + j] = label_array[i * width + (j - 1)];
                    }else{
                        label_array[i * width + j] = label_array[(i + 1) * width + j];
                    }
```

```
}else{
    if((i + 1) >= height && (j - 1) < 0){
        label_count ++;
        label_array[i * width + j] = label_count;
    }
    if((i + 1) >= height && (j - 1) >= 0){
        if(image_mor[(i * width + (j - 1)) * 3 + 0] == 0){
            label_array[i * width + j] = label_array[i * width + (j - 1)];
        }else{
            label_count ++;
            label_array[i * width + j] = label_count;
        }
    }
    if((i + 1) < height && (j - 1) < 0){
        if(image_mor[((i + 1) * width + j) * 3 + 0] == 0){
            label_array[i * width + j] = label_array[(i + 1) * width + j];
        }else{
            label_count ++;
            label_array[i * width + j] = label_count;
        }
    }
}
```

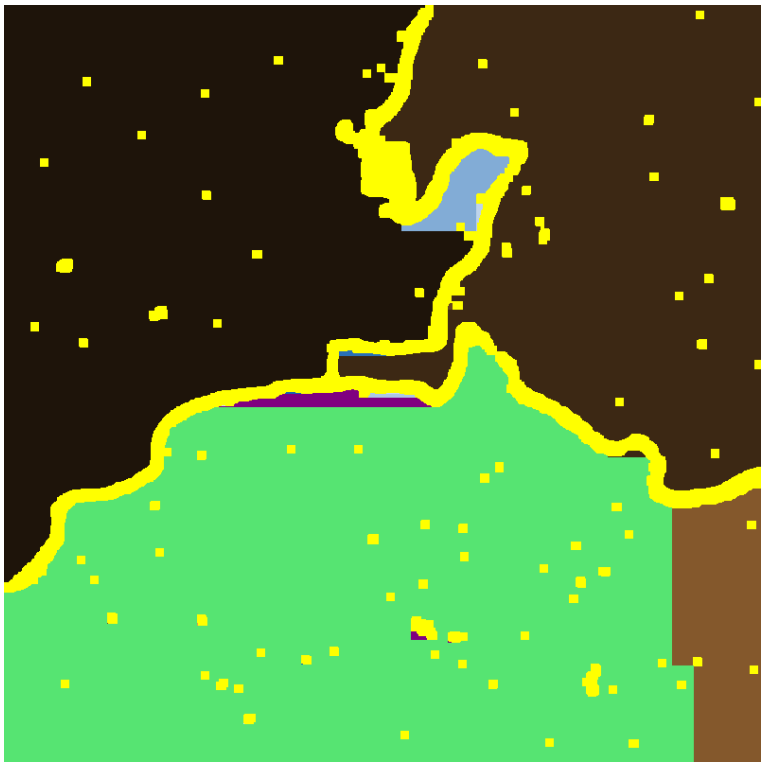
分別執行四次連通元件算法後對不同標籤的區域填不同顏色並統計標簽數。

```
first label count:372
second label count:17
third label count:5
fourth label count:4
```

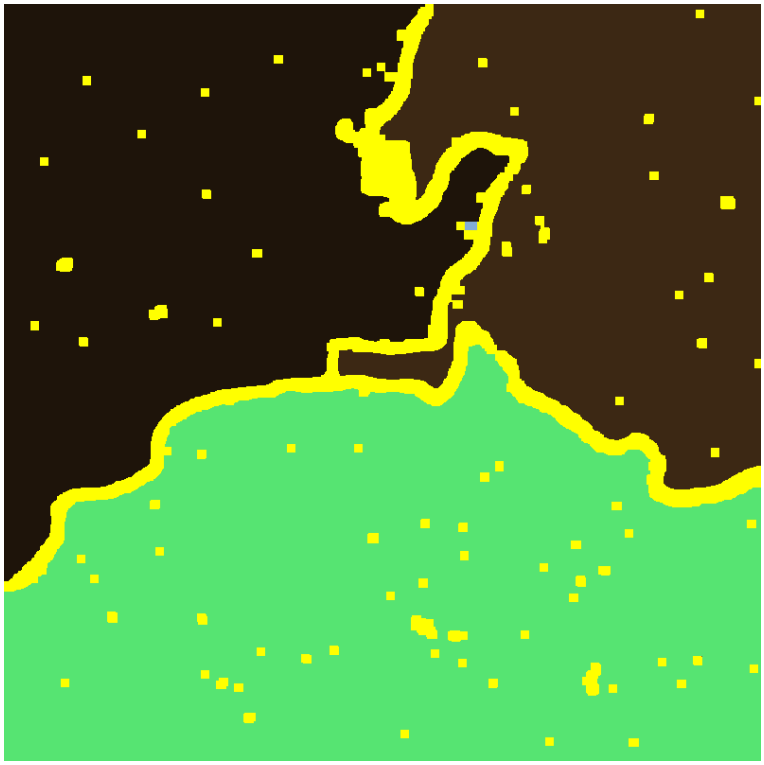
第一次執行:



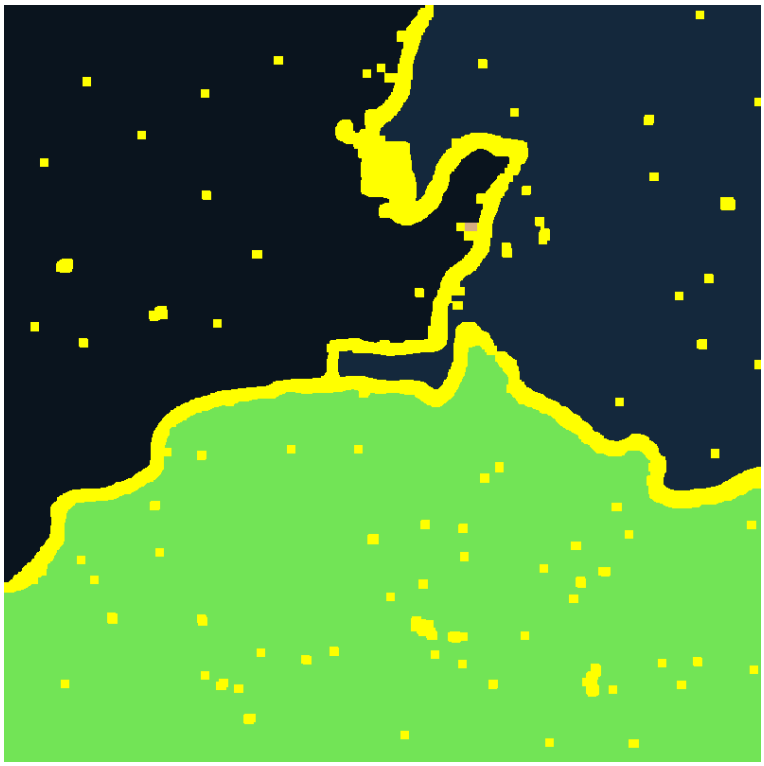
第二次執行:



第三次執行:



第四次執行:



由於標籤編號在執行完 4 次連通元件算法後可能會有標籤編號有較大的值出現，所以需要對剩下的標籤重新定義編號。

```
first renew label: 1 original label: 1
first renew label: 2 original label: 2
first renew label: 3 original label: 47
first renew label: 4 original label: 165
```

接著再列出等校標籤表，將連接區域但不同標籤值列出，方式為對每一個像素的標籤值與鄰近上下左右像素的標籤值進行比較，若有不同的標籤值則該標籤為等校標籤。

```
The label equivalence relationships:
1: 3
2:
3: 1
4:
```

```
record = 0;
for(int i = (height - 1); i >= 0; i--){
    for(int j = 0; j < width; j++){
        if(image_mor[(i * width + j) * 3 + 0] == 0){
            if((i + 1) < height && (i - 1) >= 0 && (j + 1) < width && (j - 1) >= 0){
                //上面
                if((image_mor[((i + 1) * width + j) * 3 + 0] == 0) && (label_array[i * width + j] != label_array[(i + 1) * width + j])){
                    record = 0;
                    for(int k = 0; k < label_count; k++){
                        if(label_array[(i + 1) * width + j] == equ_label[label_array[i * width + j] - 1][k]){
                            record = 1;
                        }
                        if(equ_label[label_array[i * width + j] - 1][k] == 0 && record == 0){
                            equ_label[label_array[i * width + j] - 1][k] = label_array[(i + 1) * width + j];
                            record = 1;
                        }
                    }
                }
                //左邊
                if(image_mor[(i * width + (j - 1)) * 3 + 0] == 0 && label_array[i * width + j] != label_array[i * width + (j - 1)]){
                    record = 0;
                    for(int k = 0; k < label_count; k++){
                        if(label_array[i * width + (j - 1)] == equ_label[label_array[i * width + j] - 1][k]){
                            record = 1;
                        }
                        if(equ_label[label_array[i * width + j] - 1][k] == 0 && record == 0){
                            equ_label[label_array[i * width + j] - 1][k] = label_array[i * width + (j - 1)];
                            record = 1;
                        }
```

```
//右邊
if(image_mor[(i * width + (j + 1)) * 3 + 0] == 0 && label_array[i * width + j] != label_array[i * width + (j + 1)]){
    record = 0;
    for(int k = 0; k < label_count; k++){
        if(label_array[i * width + (j + 1)] == equ_label[label_array[i * width + j] - 1][k]){
            record = 1;
        }
        if(equ_label[label_array[i * width + j] - 1][k] == 0 && record == 0){
            equ_label[label_array[i * width + j] - 1][k] = label_array[i * width + (j + 1)];
            record = 1;
        }
    }
}
//下面
if(image_mor[((i - 1) * width + j) * 3 + 0] == 0 && label_array[i * width + j] != label_array[(i - 1) * width + j]){
    record = 0;
    for(int k = 0; k < label_count; k++){
        if(label_array[(i - 1) * width + j] == equ_label[label_array[i * width + j] - 1][k]){
            record = 1;
        }
        if(equ_label[label_array[i * width + j] - 1][k] == 0 && record == 0){
            equ_label[label_array[i * width + j] - 1][k] = label_array[(i - 1) * width + j];
            record = 1;
        }
    }
}
```

```
}else{
    if((i + 1) >= height && (j - 1) >= 0){ //上面邊界
        if(image_mor[(i * width + (j - 1)) * 3 + 0] == 0 && label_array[i * width + j] != label_array[i * width + (j - 1)]){
            record = 0;
            for(int k = 0; k < label_count; k++){
                if(label_array[i * width + (j - 1)] == equ_label[label_array[i * width + j] - 1][k]){
                    record = 1;
                }
                if(equ_label[label_array[i * width + j] - 1][k] == 0 && record == 0){
                    equ_label[label_array[i * width + j] - 1][k] = label_array[i * width + (j - 1)];
                    record = 1;
                }
            }
        }
    }
    if((i + 1) < height && (j - 1) < 0){  //左邊邊界
        if((image_mor[((i + 1) * width + j) * 3 + 0] == 0) && (label_array[i * width + j] != label_array[(i + 1) * width + j])){
            record = 0;
            for(int k = 0; k < label_count; k++){
                if(label_array[(i + 1) * width + j] == equ_label[label_array[i * width + j] - 1][k]){
                    record = 1;
                }
                if(equ_label[label_array[i * width + j] - 1][k] == 0 && record == 0){
                    equ_label[label_array[i * width + j] - 1][k] = label_array[(i + 1) * width + j];
                    record = 1;
                }
            }
```

最後再重新定義一次所有的標籤值，最後可以得到三個森林區域，其標籤分別為 1、2、3。

```
Equivalent Labels after:
1 2 4
finial total label: 3
second renew label: 1 original label: 1
second renew label: 2 original label: 2
second renew label: 3 original label: 4
```

4. property analysis

統計三個區域的面積以及計算質心。

```
label1 area: 189343
label1 cen_x: 197
label1 cen_y: 567
label2 area: 145206
label2 cen_x: 639
label2 cen_y: 577
label3 area: 261792
label3 cen_x: 407
label3 cen_y: 172
```

```
int cal_label[label_count][3]; //計算質心面積
for(int i = 0; i < label_count; i++){
    for(int j = 0; j < 3; j++){
        cal_label[i][j] = 0;
    }
}
int area, cen_x_total, cen_y_total;
for(int k = 0; k < label_count; k++){
    area = cen_x_total = cen_y_total = 0;
    for(int i = (height - 1); i >= 0; i--){
        for(int j = 0; j < width; j++){
            if(label_array[i * width + j] == label_number[k]){
                cal_label[k][0] ++;
                cen_y_total += i;
                cen_x_total += j;
            }
        }
    }
    cal_label[k][1] = cen_y_total / cal_label[k][0];
    cal_label[k][2] = cen_x_total / cal_label[k][0];
}

for(int i = 0; i < label_count; i++){
    printf("label%d area: %d\n", i + 1, cal_label[i][0]);
    printf("label%d cen_x: %d\n", i + 1, cal_label[i][2]);
    printf("label%d cen_y: %d\n", i + 1, cal_label[i][1]);
}
```

5. drawing

先找出三個區域的範圍，並且透過更改像素的 rgb 來畫出三個區域的邊界，以及在找出質心位置後將質心位置透過 3*3 大小黑色區域畫出。

```
int min_left = 0;
int max_right = 0;
int min_under = 0;
int max_top = 0;
for(int k = 0; k < label_count; k++){
    min_left = 799;
    max_right = 0;
    min_under = 799;
    max_top = 0;
    for(int i = (height - 1); i >= 0; i--){
        for(int j = 0; j < width; j++){
            if(label_array[i * width + j] == label_number[k]){
                if(i > max_top){
                    max_top = i;
                }
                if(i < min_under){
                    min_under = i;
                }
                if(j > max_right){
                    max_right = j;
                }
                if(j < min_left){
                    min_left = j;
                }
            }
        }
    }
```

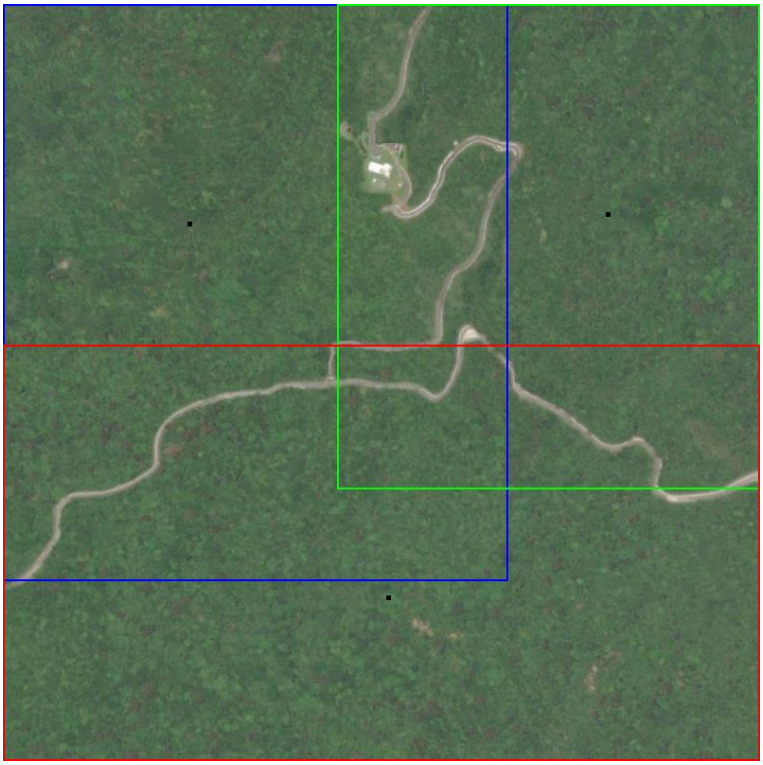
```
    label_edge[k][0] = min_left;
    label_edge[k][1] = max_right;
    label_edge[k][2] = min_under;
    label_edge[k][3] = max_top;
}

for(int i = 0; i < label_count; i++){
    printf("label%d:", i + 1);
    for(int j = 0; j < 4; j++){
        printf("%d ", label_edge[i][j]);
    }
    printf("\n");
}

int color_number[3][3] = {{255, 0, 0}, {0, 255, 0}, {0, 0, 255}};
```

```
for(int k = 0; k < label_count; k++){
    //下面
    for(int i = label_edge[k][2]; i <= label_edge[k][2] + 1; i++){
        for(int j = label_edge[k][0]; j <= label_edge[k][1]; j++){
            image_ori[(i * width + j) * 3 + 0] = color_number[k][0];
            image_ori[(i * width + j) * 3 + 1] = color_number[k][1];
            image_ori[(i * width + j) * 3 + 2] = color_number[k][2];
        }
    }
    //上面
    for(int i = label_edge[k][3] - 1; i <= label_edge[k][3]; i++){
        for(int j = label_edge[k][0]; j <= label_edge[k][1]; j++){
            image_ori[(i * width + j) * 3 + 0] = color_number[k][0];
            image_ori[(i * width + j) * 3 + 1] = color_number[k][1];
            image_ori[(i * width + j) * 3 + 2] = color_number[k][2];
        }
    }
    //左邊
    for(int i = label_edge[k][2]; i <= label_edge[k][3]; i++){
        for(int j = label_edge[k][0]; j <= label_edge[k][0] + 1; j++){
            image_ori[(i * width + j) * 3 + 0] = color_number[k][0];
            image_ori[(i * width + j) * 3 + 1] = color_number[k][1];
            image_ori[(i * width + j) * 3 + 2] = color_number[k][2];
        }
    }
    //右邊
    for(int i = label_edge[k][2]; i <= label_edge[k][3]; i++){
        for(int j = label_edge[k][1] - 1; j <= label_edge[k][1]; j++){
            image_ori[(i * width + j) * 3 + 0] = color_number[k][0];
            image_ori[(i * width + j) * 3 + 1] = color_number[k][1];
            image_ori[(i * width + j) * 3 + 2] = color_number[k][2];
```

```
for(int k = 0; k < label_count; k++){
    for(int i = -2; i < 3; i++){
        for(int j = -2; j < 3; j++){
            image_ori[((cal_label[k][1] + i) * width + (cal_label[k][2] + j)) * 3 + 0] = 0;
            image_ori[((cal_label[k][1] + i) * width + (cal_label[k][2] + j)) * 3 + 1] = 0;
            image_ori[((cal_label[k][1] + i) * width + (cal_label[k][2] + j)) * 3 + 2] = 0;
        }
    }
}
```



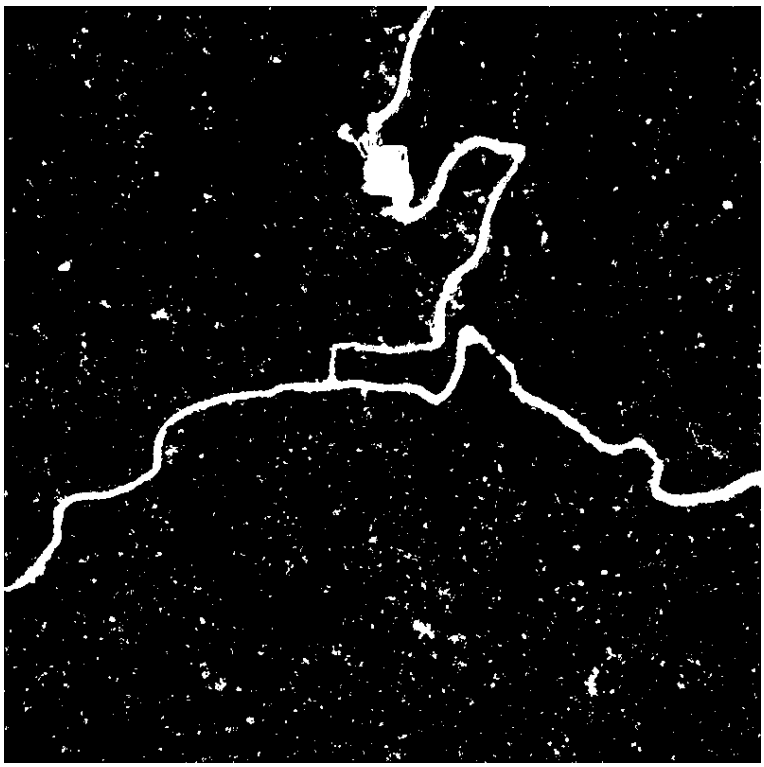
二、 OpenCV 影像處理

1. Binarizing

先將彩色影像透過 cvtColor 轉成灰階圖，再透過 threshold 轉二值化。

```
//binarizing
Mat img_gray;
Mat img_bin;
img.copyTo(img_gray); //複製影像
cvtColor(img, img_gray, COLOR_BGR2GRAY);
img_gray.copyTo(img_bin); //複製影像
threshold(img_gray, img_bin, 110, 255.0, 0);
imwrite("img_bin.bmp", im
imshow("img_bin", img_bin);
double t_bin = (double)getTickCount();
cout << "binarizing time: " << (t_bin - t_start) / (getTickFrequency()) * 1000 << "(ms)" << endl;
```

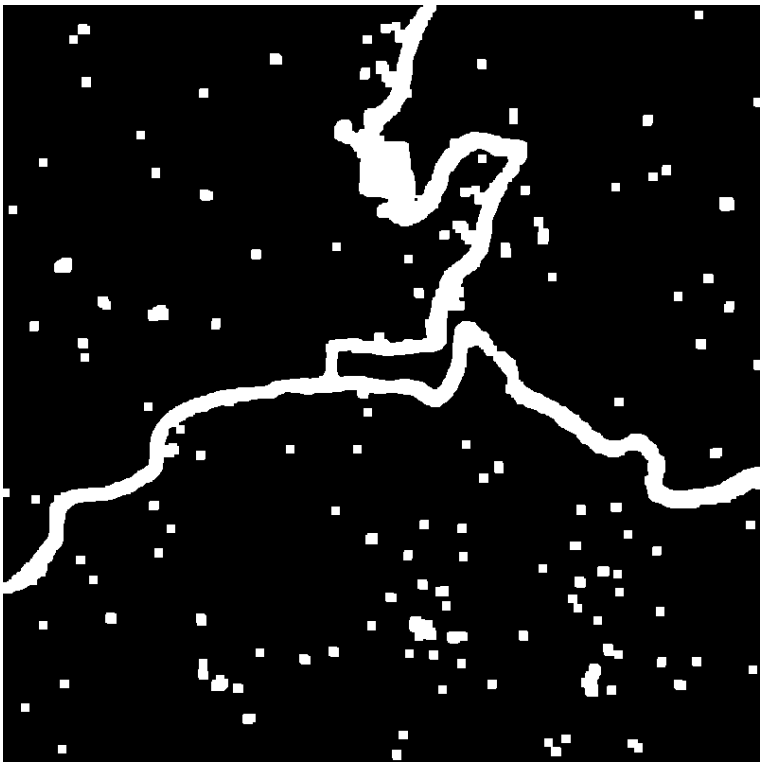
cv::Mat img_bin



2. Morphology

使用 erode 和 dilate 來對二值化影像進行侵蝕和膨脹。

```
//morphology
Mat img_mor;
img_bin.copyTo(img_mor); //複製影像
Mat kernel_size3 = getStructuringElement(MORPH_RECT, Size(3, 3));
Mat kernel_size5 = getStructuringElement(MORPH_RECT, Size(5, 5));
Mat kernel_size7 = getStructuringElement(MORPH_RECT, Size(7, 7));
Mat kernel_size9 = getStructuringElement(MORPH_RECT, Size(9, 9));
erode(img_bin, img_mor, kernel_size3, Point(-1, -1), 1);
dilate(img_mor, img_mor, kernel_size7, Point(-1, -1), 1);
dilate(img_mor, img_mor, kernel_size5, Point(-1, -1), 1);
erode(img_mor, img_mor, kernel_size3, Point(-1, -1), 1);
imwrite("img_mor.bmp", img_mor);
imshow("img_mor", img_mor);
double t_mor = (double)getTickCount();
cout << "morphology time: " << (t_mor - t_bin) / (getTickFrequency()) * 1000 << "(ms)" << endl;
```



3. connected component

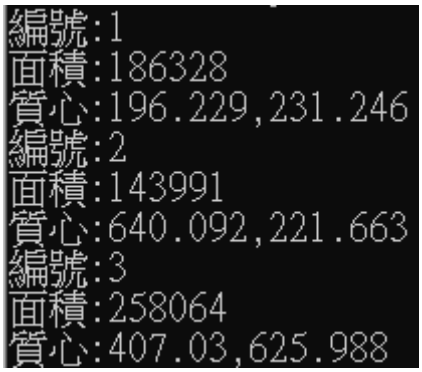
使用 connectedComponentsWithStats 找出連通元件的數量，並且也能得出各區域的面積、質心。

```
//connected component
Mat img_label, stats, centroids, img_color;
int area, left, right, top, under;
for (int i = 0; i < img.rows; i++) {
    for (int j = 0; j < img.cols; j++) {
        int pv = img_mor.at<uchar>(i, j);
        img_mor.at<uchar>(i, j) = 255 - pv;
    }
}
int i, nccomps = connectedComponentsWithStats(img_mor, img_label, stats, centroids);
double t_con = (double)getTickCount();
cout << "connected component time: " << (t_con - t_mor) / (getTickFrequency()) * 1000 << "(ms)" << endl;
```

4. property analysis

用面積大小超過 100 來去除小區域的面積，並且顯示剩下的區域其編號、面積大小、及質心位置。

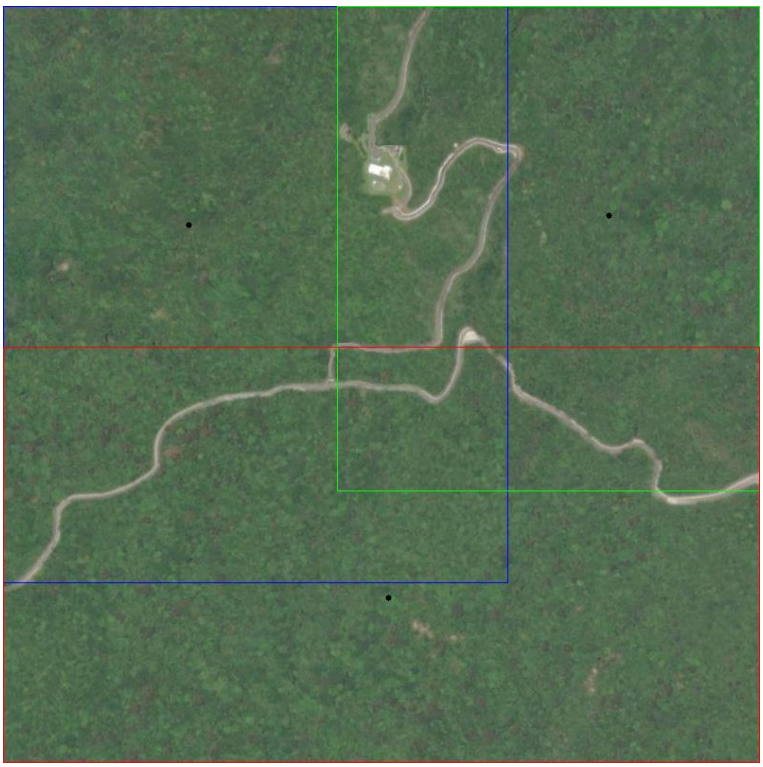
```
//property analysis
int label_index = -1;
for (int i = 1; i < nccomps; i++) {
    if (stats.at<int>(i, CC_STAT_AREA) >= 100) {
        label_index++;
        cout << "編號:" << label_index + 1 << endl;
        cout << "面積:" << stats.at<int>(i, CC_STAT_AREA) << endl;
        cout << "質心:" << centroids.at<double>(i, 0) << "," << centroids.at<double>(i, 1) << endl;
    }
}
double t_pro = (double)getTickCount();
cout << "property analysis time: " << (t_pro - t_con) / (getTickFrequency()) * 1000 << "(ms)" << endl;
```



5. drawing

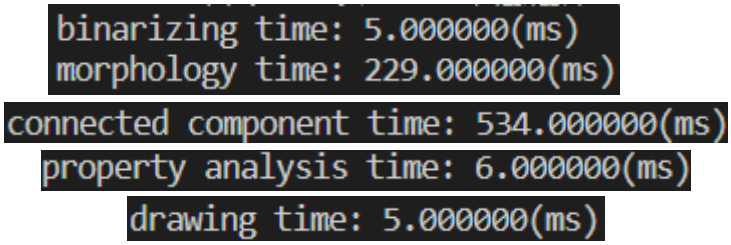
最後為畫出每一塊森林的區域以及標示質心的位置。

```
//drawing
vector<Rect> label_edge(3);
vector<Scalar> label_color(3);
label_color[0] = Scalar(255, 0, 0);
label_color[1] = Scalar(0, 255, 0);
label_color[2] = Scalar(0, 0, 255);
label_index = -1;
for (int i = 1; i < nccomps; i++) {
    if (stats.at<int>(i, CC_STAT_AREA) >= 100) {
        label_index++;
        label_edge[label_index] = Rect(stats.at<int>(i, CC_STAT_LEFT),
                                       stats.at<int>(i, CC_STAT_TOP),
                                       stats.at<int>(i, CC_STAT_WIDTH),
                                       stats.at<int>(i, CC_STAT_HEIGHT));
        cv::rectangle(img, label_edge[label_index], label_color[label_index], 1, LINE_8, 0);
        cv::circle(img, Point(centroids.at<double>(i, 0), centroids.at<double>(i, 1)), 3, Scalar(0, 0, 0));
    }
}
imwrite("img_label.bmp", img);
imshow("label", img);
double t_dra = (double)getTickCount();
cout << "drawing time: " << (t_dra - t_pro) / (getTickFrequency()) * 1000 << "(ms)" << endl;
```

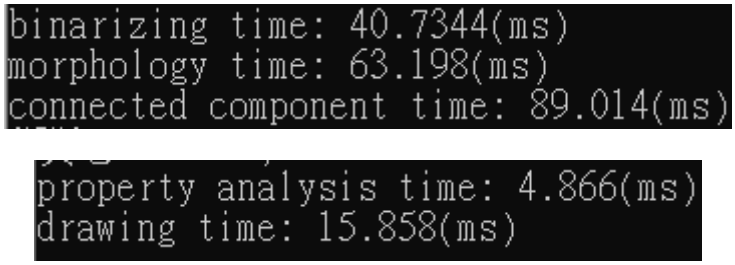


三、 時間比較

C program:



OpenCV:



Time(ms)	binarizing	morphology	connected component	property analysis	drawing
C program	5.00	229.00	534.00	6.00	5.00
OpenCV	40.73	63.20	89.01	4.87	15.86