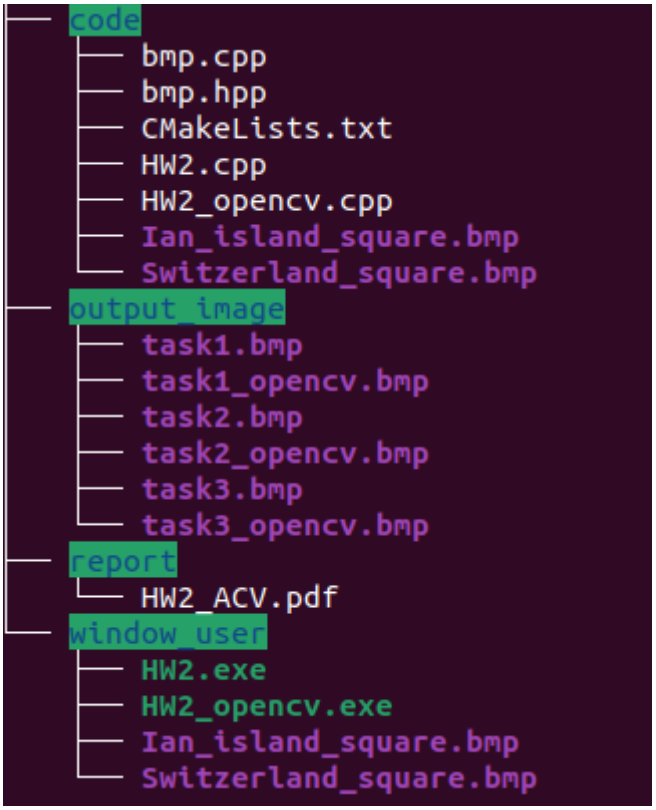
高等電腦視覺

作業#(2)

姓名:闕楷宸
學號:114318047
指導老師:黃正民

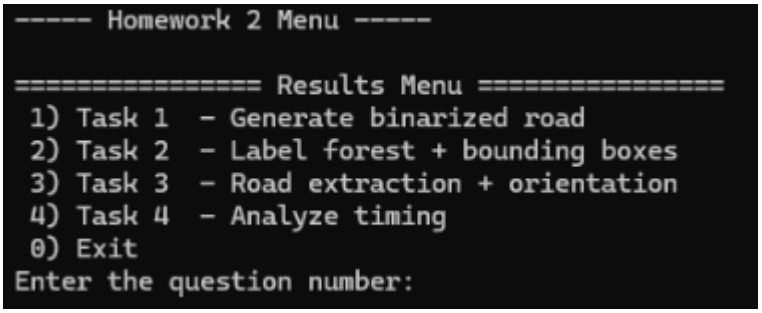
作業說明:

作業架構:



主程式HW2.cpp HW2_opencv.cpp

bmp.cpp、bmp.hpp:HW1使用的bmp讀寫



輸入0~4即可輸出結果

建置執行(window,linux)

Linux:

1.使用opencv 4.5.4(sudo apt install libopencv-dev)
2.建置與執行:

```
cd HW#2_114318047/code/
cmake -S . -B build
cmake --build build -j
./build/HW2
./build/HW2_opencv
```

Window:

```
cd window_user
./HW2.exe
./HW2_opencv.exe
```

Window 建置:

1:Cmake:

```
winget install Kitware.CMake
```

2:OpenCV for window:

```
C:\opencv\build\x64\vc16\bin(make sure DLLs are in this dir)
```

3:Confirm CMake config file exists:

```
C:\opencv\build\x64\vc16\lib\OpenCVConfig.cmake
```

4:Build and run

```
cd HW#2_114318047\code
```

5:Configure:

```
cmake -S . -B build -G "Visual Studio 17 2022" -A x64
-DOpenCV_DIR="C:\opencv\build\x64\vc16\lib"
```

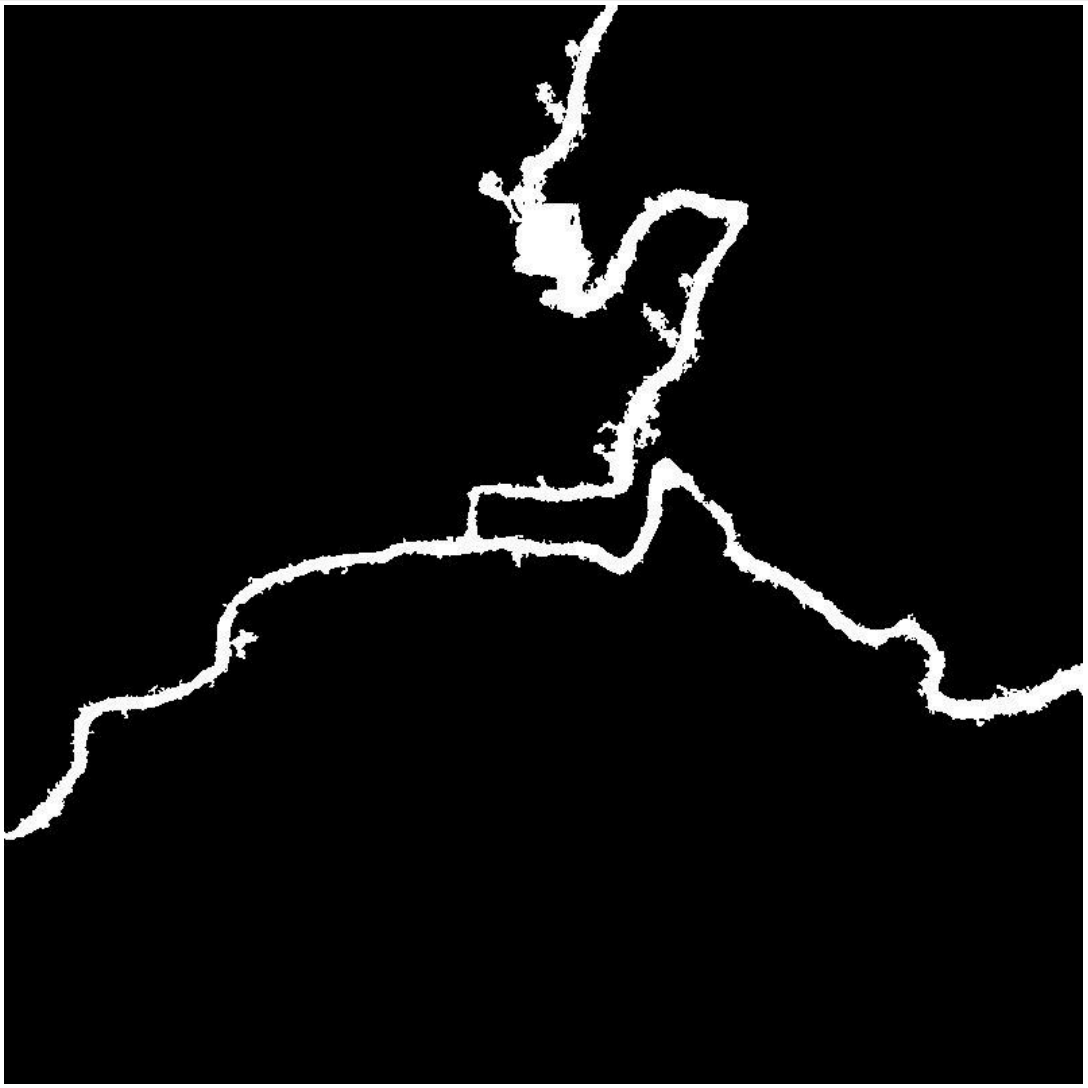
6:Produce exe:

```
cmake --build .\build --config Release
```

7:Run exe

```
.\build\Release\HW2.exe
.\build\Release\HW2_opencv.exe
```

Task1: Generate a binarized image of road



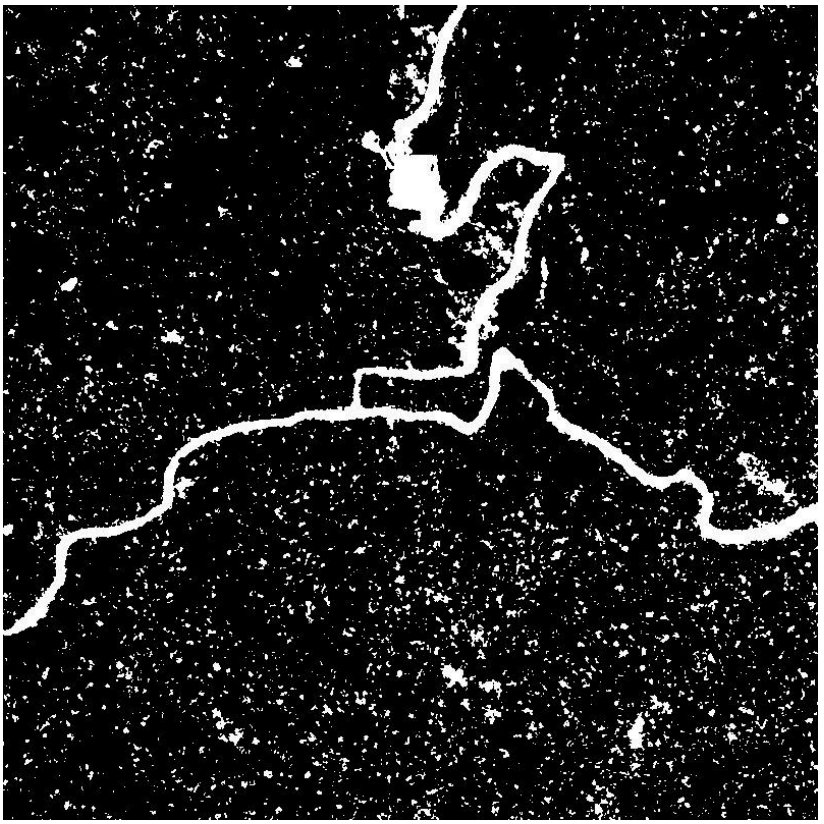
task1.bmp(./HW2)、**task1_opencv.bmp**(./HW2_opencv)

Discussion(task1 C++)

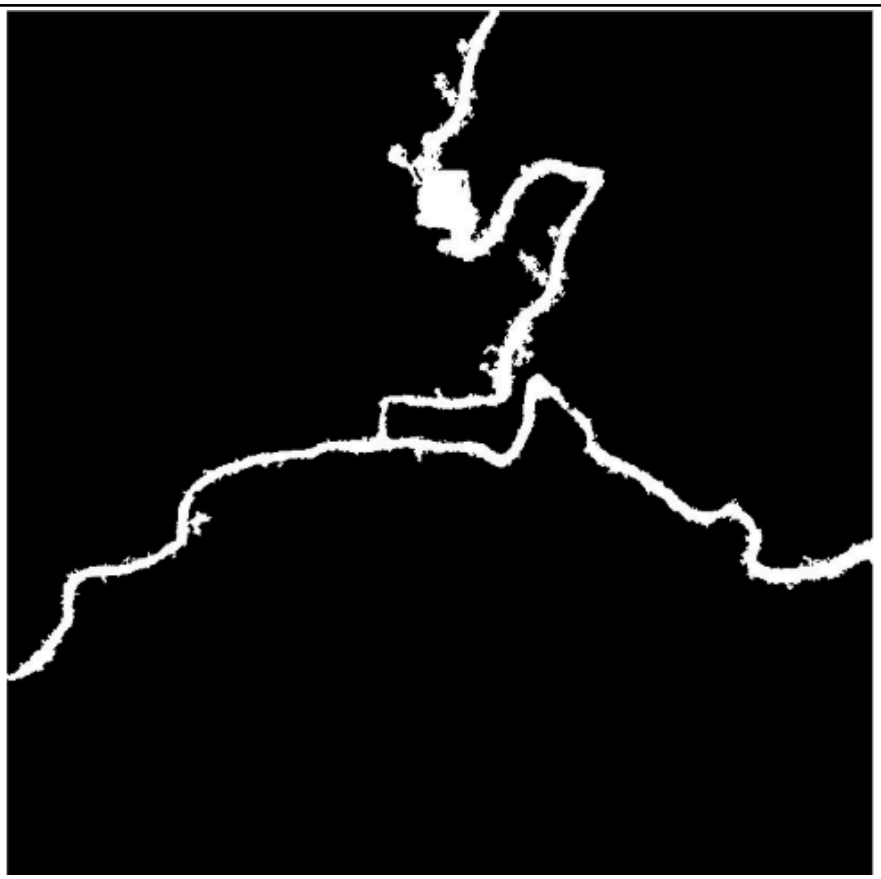
此題須將Ian_island_square.bmp轉成binarized image，我使用average intensity

```
int average_intensity = ((int)px[0] + (int)px[1] + (int)px[2]) / 3;
```

去做thresholding，並且使用BFS 尋找connect component透過面積去filter掉一些過小的component，下圖為未進行component filter的結果(許多小點)



Binarized image(未進行component filtering)



Binarized image(After component filtering)

並且將常用的功能寫成subfunction，比如說bfs、isWhite、setBlack，分別是使用BFS配合4-neighbor 遍歷圖像找尋component，isWhite則是檢查pixel是否為白、setBlack則是將comnponent設為黑色(用於component filtering)，下圖為bfs:

```
static void bfs(const bmp::BMPImage& img, int rowSize, int
width, int height,
                std::vector<int>& visited, int sr, int sc,
                std::vector<std::pair<int, int>>& compPixels,
                bool targetWhite)
{
    // 4-neighbors
    const int dr[4] = {-1, 1, 0, 0};
    const int dc[4] = { 0, 0, -1, 1};

    std::queue<std::pair<int, int>> q;
    q.push({sr, sc});
    // occupy starting pixel
    visited[sr * width + sc] = 1;

    while (!q.empty()) {
        auto [r, c] = q.front();
```

```
        q.pop();
        compPixels.push_back({r, c});

        for (int k = 0; k < 4; ++k) {
            int nr = r + dr[k];
            int nc = c + dc[k];
            if (nr < 0 || nr >= height || nc < 0 || nc >= width)
                continue;

            int id = nr * width + nc;
            if (visited[id]) continue;

            const uint8_t* np = &img.data[nr * rowSize + nc * 3];
            bool isWhite = (np[0] == 255 && np[1] == 255 && np[2]
== 255);
            bool isBlack = (np[0] == 0 && np[1] == 0 && np[2] ==
0);

            // Choose behavior based on targetWhite flag
            if ((targetWhite && isWhite) || (!targetWhite &&
isBlack)) {
                visited[id] = 1;
                q.push({nr, nc});
            }
        }
    }
}
```

並且再透過component的面積大小決定是否被filter掉(MIN_AREA:900)

```
if ((int)compPixels.size() < MIN_AREA) {
    for (std::pair<int, int>& p : compPixels) {
        setBlack(img, rowSize, p.first, p.second);
    }
}
```

Discussion(task1 OpenCV)

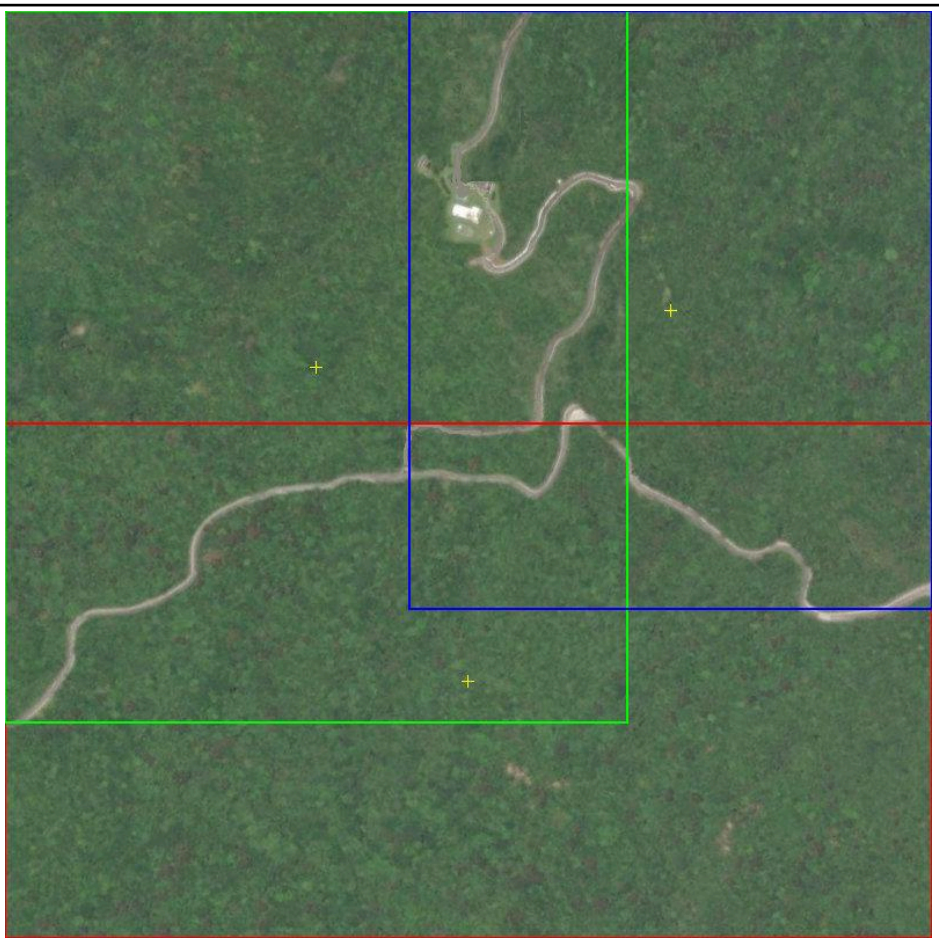
OpenCV則是將BGR bmp轉成grayscale(COLOR_BGR2GRAY)，並且同樣使用thresholding進行filter，並使用OpenCV提供的connectedComponentsWithStats計算connected component如下：

```
int number_of_connected_components =
connectedComponentsWithStats(mask, labels, stats, centroids, 4);
```

並且再進行面積濾波，如C++程式一樣。

Task2: Label the forest regions with 4-connected neighbor

Label each connected component with R/G/B bounding box in output image,and compute the centroid of each regions and print the data on output image or command window.



task2.bmp(./HW2)、**task2_opencv.bmp**(./HW2_opencv)

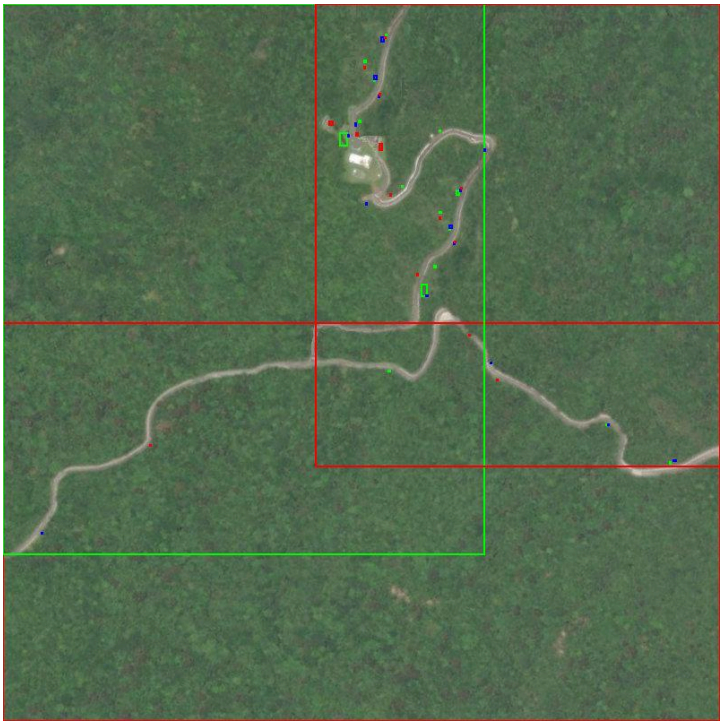
```
Enter the question number: 2
Box 1: Centroid=(399,221), Area=355200
Box 2: Centroid=(268,492), Area=329718
Box 3: Centroid=(574,541), Area=232716
```

Discussion (Task2 C++)

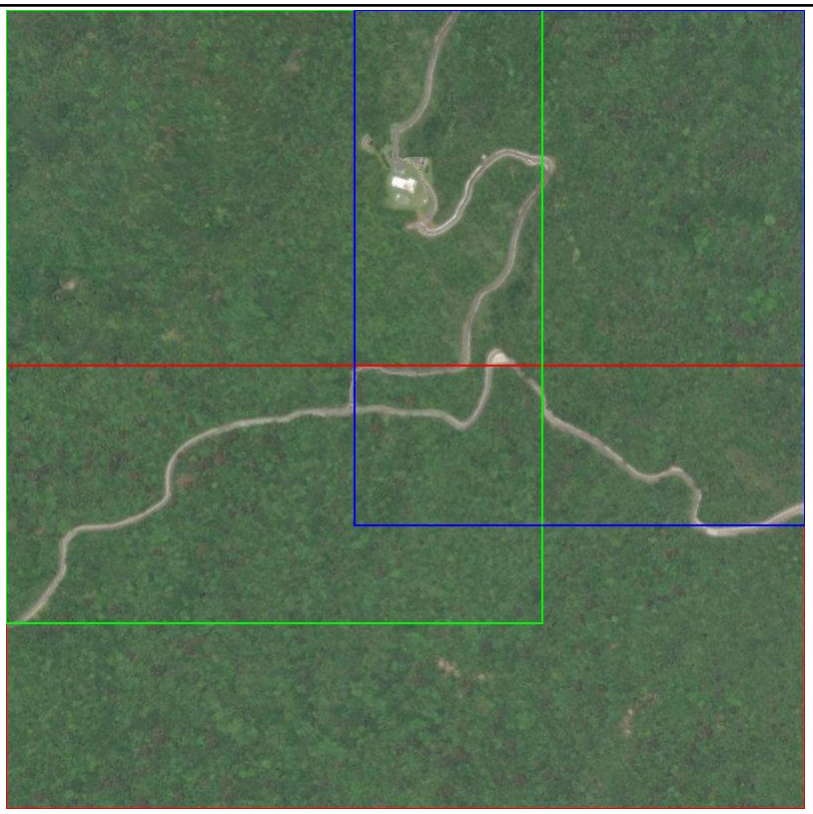
此題使用前題所得到的tas1.bmp作為mask，來找尋mask中的黑色區塊進行標注，方法為借助先前使用的bfs找尋component，並檢查是否為黑，如果是，進行標注，使用tuple存入component的最小及最大row、col值，如下:

```
int minR = height, maxR = -1, minC = width, maxC = -1;
for (const auto& p : compPixels) {
    if (p.first < minR) minR = p.first;
    if (p.first > maxR) maxR = p.first;
    if (p.second < minC) minC = p.second;
    if (p.second > maxC) maxC = p.second;
}

// Only keep large enough black regions
if ((int)compPixels.size() >= MIN_FOREST_AREA)
// Add one new bounding box into vector.
    boxes.emplace_back(minR, minC, maxR, maxC);
```

下圖為未進行component filtering的結果，因此使用面積過濾:



下圖為使用component filtering的結果，一些過小的森林會被過濾

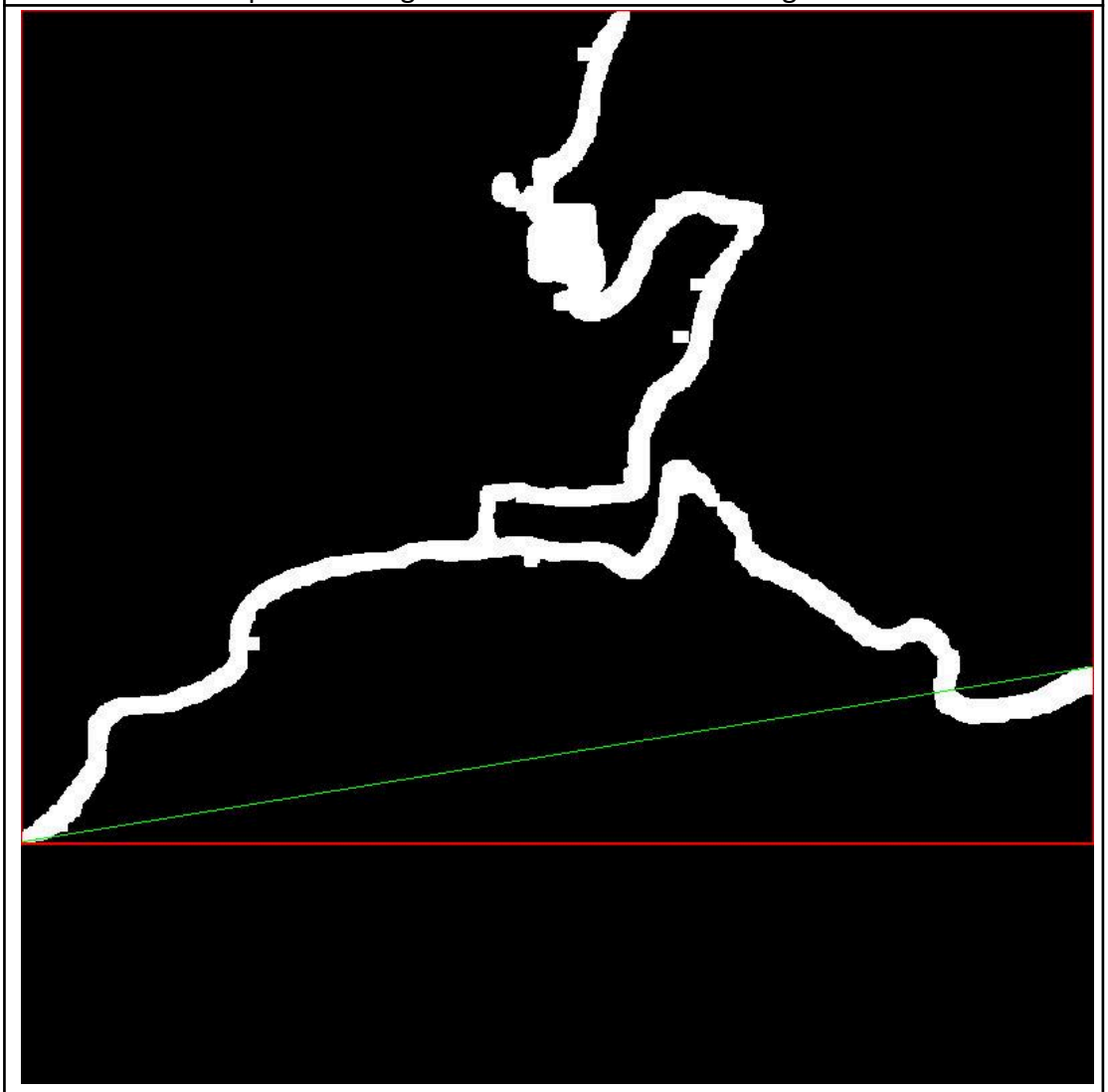


Discussion (Task2 OpenCV)

在opencv的程式中，我使用IMREAD_GRAYSCALE轉為灰階圖像，並且使用threshold()將task1.bmp(mask)進行反轉，因為opencv使用的connectedComponentsWithStats函式必須使用白色作為target進行connected component的遍歷(BFS)，

Task3: Use the morphology algorithms to reserve the road

Report the length and orientation of the longest axis



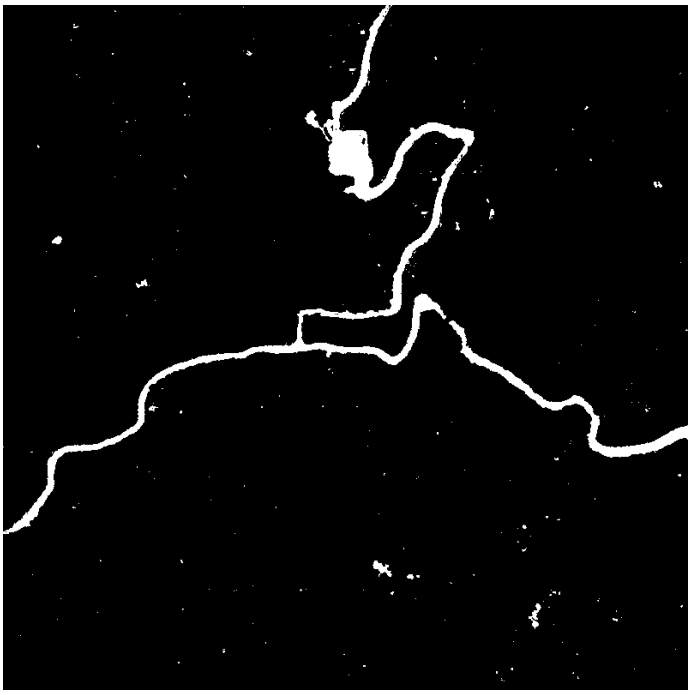
Task3.bmp

```
Enter the question number: 3
Component 1: Area = 33095, Bounding Box = [(179, 0), (799, 799)]
Longest axis length: 807.533
Orientation (degrees): 9.26403
```

Discussion (Task3 C++)

此題較複雜，我先使用先前使用過的二值化方法(使用average intensity將圖像二值化後)，此為Stage1，如下:

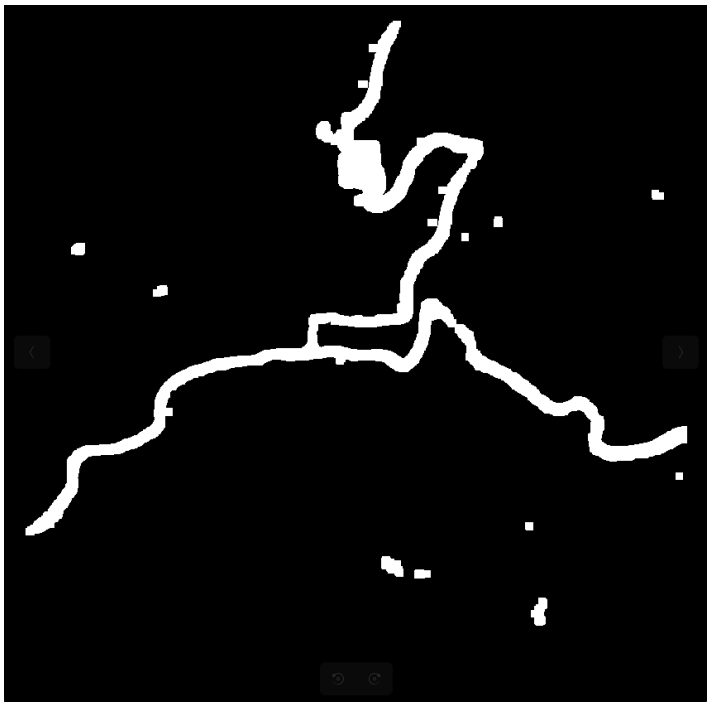
```cpp
const int intensity_threshold = 110;
for (int r = 0; r < height; ++r) {
    for (int c = 0; c < width; ++c) {
        uint8_t* px = &img.data[r * rowSize + c * 3];
        int average_intensity = ((int)px[0] + (int)px[1] + (int)px[2]) / 3;

        if (average_intensity < intensity_threshold) {
            px[0] = 0; px[1] = 0; px[2] = 0; // Set to black
        } else {
            px[0] = 255; px[1] = 255; px[2] = 255; // Set to white
        }
    }
}
```



Stage1 結果

Stage2為morphology，根據二值化的結果，我們可以知道需要對圖像進行一次Erosion(消除白色點)，並做dilation(填補破碎的路)，並使用subfunction將常用的erosion、dilation寫好並呼叫，這裡在做完Opening完後，又進行一次dilation，以補上路面，下圖為使用一次Opening跟一次dilation的過程:

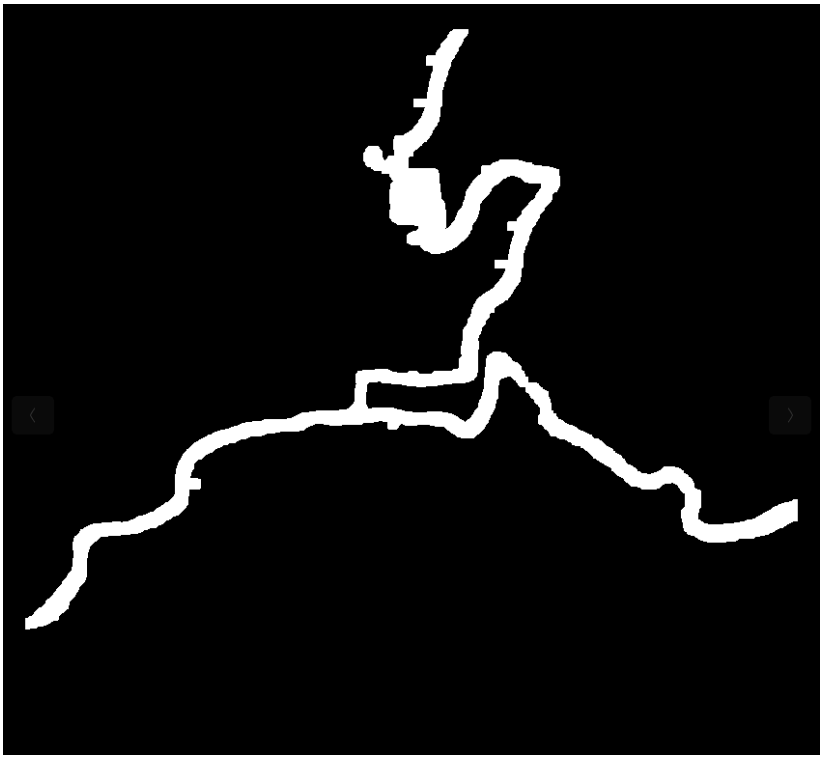
```cpp
// Do Opening
bmp::BMPImage eroded = img; // Copy for erosion

// Erosion
apply_erosion(img, eroded, kernel_size);

bmp::BMPImage dilated = eroded; // Copy for first dilation

// call dilate function
apply_dilatation(eroded, dilated, kernel_size);

// one more Dilation to restore road width
bmp::BMPImage temp = dilated; // Copy for second dilation

// Dilation
apply_dilatation(dilated, temp, kernel_size+4);

dilated = temp; // Update dilated image
```



Stage2 結果

根據stage2的結果我們可以發現細點減少很多，但是仍存在，並且面積放大，但是路面已成功填起來，因此在stage3(connected component)的時候，會檢查component的面積是否過小，並進行濾除。

```cpp
const int MIN_ROAD_AREA = 2000; // Minimum area for road components
std::vector<int> visited(width * height);

for (int sr = 0; sr < height; ++sr) {
    for (int sc = 0; sc < width; ++sc) {

        // If not visited
        if (!visited[sr * width + sc]) {
            uint8_t* px = &dilated.data[sr * rowSize + sc * 3];
            if (px[0] == 255 && px[1] == 255 && px[2] == 255) { // white pixel
                std::vector<std::pair<int, int>> compPixels;

                // Perform BFS to find all connected white pixels
                bfs(dilated, rowSize, width, height, visited, sr, sc, compPixels, true);

                // If component area is less than MIN_ROAD_AREA, set all its pixels to black
                if ((int)compPixels.size() < MIN_ROAD_AREA) {
                    for (std::pair<int, int>& p : compPixels) {
                        setBlack(dilated, rowSize, p.first, p.second);
                    }
                }
            }
        }
    }
}
```

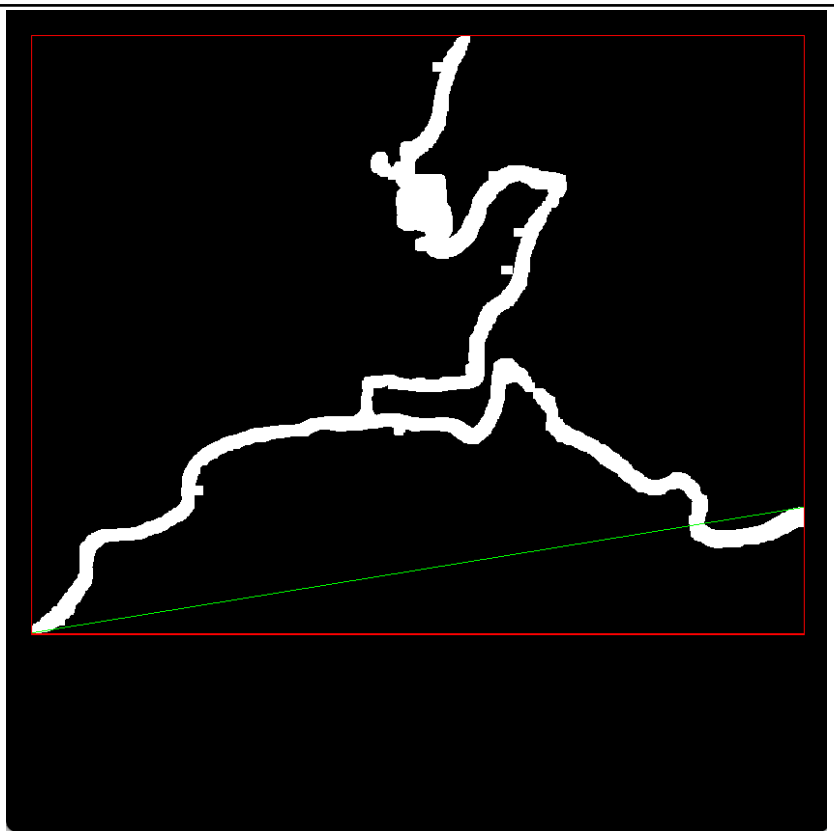


根據Stage3(connected component)的結果，我們已發現路面在完整的情況下，細點也成功消除，透過component的面積進行filter，接下來則是Stage4(propery analysis)，判斷路面的位置，並交給Stage5進行標注

```
Bounding Box 1: [(178, 0), (799, 799)]
Area: 35355
Corners: Top-Left=(178, 0), Top-Right=(178, 799), Bottom-Left=(799, 0), Bottom-R
ight=(799, 799)
Component 1: Area = 35355, Bounding Box = [(178, 0), (799, 799)]
```

```cpp
for (int sr = 0; sr < height; ++sr) {
    for (int sc = 0; sc < width; ++sc) {

        // If not visited
        if (!visited[sr * width + sc]) {
            uint8_t* px = &dilated.data[sr * rowSize + sc * 3];
            if (px[0] == 255 && px[1] == 255 && px[2] == 255) { // white pixel
                std::vector<std::pair<int, int>> compPixels;
                bfs(dilated, rowSize, width, height, visited, sr, sc, compPixels, true);

                // Find bounding box
                int minR = height, maxR = -1, minC = width, maxC = -1;
                for (const auto& p : compPixels) {
                    if (p.first < minR) minR = p.first;
                    if (p.first > maxR) maxR = p.first;
                    if (p.second < minC) minC = p.second;
                    if (p.second > maxC) maxC = p.second;
                }

                // Store bounding box and area
                boundingBoxes.emplace_back(minR, minC, maxR, maxC);
                componentAreas.push_back((int)compPixels.size());
            }
        }
    }
```

根據Stage4(property analysis)的結果，我們可以發現只有一個component，並且透過先前相似的方法進行bfs找尋component，並找尋Bbox，存入四個頂點的位址，並交給Stage5(draw Bbox)處理



Discussion(task3: OpenCV)

此題使用OpenCV就簡單很多，比如說可以使用OpenCV提供的Thresholding function: **threshold(gray, binary, 110, 255, THRESH_BINARY);**，即可將pixel值透過intensity:110進行二值化，如果大於110及轉成255，並且也有使用

```
Mat kernel = getStructuringElement(MORPH_RECT, Size(3, 3));
```

可以生成3x3 structing element，並使用以下兩個function即可做到Opening以及dilation的效果:

```
morphologyEx(binary, opened, MORPH_OPEN, kernel);
dilate(opened, dilated, getStructuringElement(MORPH_RECT,
Size(7, 7)));
```

當然也還有**int number_of_connected_components = connectedComponentsWithStats(dilated, labels, stats, centroids, 4);** 這個好用的函式能夠計算component

Task4: Analyze and print the computational time of your program

```
Task 3 Timing Information:
 Stage 1 (Binarization): 1064 us
 Stage 2 (Morphological Operations): 36987 us
 Stage 3 (Connected Component Analysis): 1852 us
 Stage 4 (Property Analysis): 1030 us
 Stage 5 (Bounding Box Drawing): 5 us
 Total Time: 40941 us

Time Complexity Analysis:
 Stage 1 (Binarization): O(H * W)
 Stage 2 (Morphological Operations): O(H * W * K^2),
 Stage 3 (Connected Component Analysis): O(H * W)
 Stage 4 (Property Analysis): O(H * W)
 Stage 5 (Bounding Box Drawing): O(H * W)
 Overall Time Complexity: O(H * W * K^2), as morphol
```

上圖為未使用OpenCV的分析結果

```
Enter choice: 4
Longest axis length: 809.994
Orientation (deg): 9.45069

Timing (ms):
 Stage1 Binarization: 41 µs
 Stage2 Morphology:   1348 µs
 Stage3 Components:   17968 µs
 Stage4 Analysis:     272 µs
 Stage5 BoundingBox:  28 µs
 Total:               19660 µs
Task3 complete: saved task3_opencv.bmp
```

上圖為使用OpenCV的分析結果

Discussion (task4)

Task4使用task3()的function，計算各個stage所需時間，以下是比較:

Time(us)	Binarizing	morphology	Connected component	Property analysis	Drawing
C++ program	1064	36987	1852	1030	5
OpenCV	41	1348	17968	272	28

從上表可以發OpenCV在大部份的Stage都贏過C++ 程式，但是比如像是connected component卻大大贏過使用OpenCV的程式，這是因為OpenCV中的

connectedComponentsWithStats() 雖然很方便，但是其內部做的事情比使用C++來得多，比如說他需要標注所有pixels，並且還計算各個component的width、heigh、area等等，並計算中心點，最後並Allocate 三個matrices，分別是labels、stats、centroids，存入上述所計算的結果，但是我自己使用的C++程式只須traverse一次，針對白色pixel，並且使用queue進行處理