CMP2090M

OBJECT-ORIENTED PROGRAMMING ASSIGNMENT REPORT

Luke Powell, [Student ID: POW16634291]

1. INTRODUCTION

This report will decompose the code of the assignment task which allows a user to take a reference image of Wally from a large cluttered scene of Wally and use the reference image as a template in order to find Wally on the large cluttered scene and highlight a box around him emphasising his location. The repot will discuss what programming features have been used in the code which finds Wally such as the structure of the classes involved, what methods and functions have been used as well as any additional C++ features. To add to this point, the Nearest-Neighbour search algorithm used within the programme will be broken down and explained in more detail. Once the code has been explained in a simpler form, the results of the compiled code will then be discussed. The ideal outcome from the programme is to compile and, in a user friendly way, show that Wally is being found and finally that Wally has been found and output to a .pgm file where the results of the programme can be viewed. The programme created efficiently runs in less than a minute, successfully finding Wally.

2. PROGRAMME STRUCTURE

In the program there are four source files. The first source file is the main.cpp file. This file is where everything is bought together and contains the code which interacts with the user. The functions within the main file read the .txt files in, put the image data values into matrices, run the sum of squared difference calculation, find a match for Wally and finally write the large image showing Wally's location to a .pgm file. The project also contains three more source files which are linked to a header file containing the classes. The classes include Base_Image, Large_Image and Ref_Image. The Large_Image and Ref_Image classes are derived classes of the Base_Image. The Base_Image class involves creating and defining the matrices within the program. Within the class the matrices are created, defined and copied with a pointer returning the area array which then adds the values into a new array. The class also contains the calculations behind the sum of squared difference. The sum of squared difference will be discussed further later in the report. Furthermore, there is also a Large_Image class. This class is involved with the large cluttered image. Within the class are two methods. The first method creates a matrix with a height and width of 1. The next method is the method which draws a black box around Wally on the large cluttered scene showing his location. The next class is the Ref_Image class which creates a matrix of size 49x36, copies the matrix and then gets the starting values for x, y and the SSD value and sets them to new values. Each class contains member functions, the initialisation of variables and data members. Moreover, each class has a specific task related to the programme. The Base_Image class deals with creating the matrices and the calculations behind the NNS algorithm. The Large_Image deals with drawing a black box in order to highlight the Wally image on the cluttered image. Finally, the Ref_Image class deals with matching the two images through get and set functions. It is the main.cpp file which brings the classes and files all together in order to completely work and successfully find Wally. Classes are important due to the fact that they allow code to be organised so that it is clear, easier to understand and runs more quickly. The addition of comments allows what is going on to be understood, making it easier to make any changes. A range of methods have been used to, once again, create a more organised project which was easy to navigate through. The programme also contains constructors and destructors. A constructor automatically makes an object usable whereas a destructor frees resources. Copy constructors have also been used within the program. Copy constructors construct a new object from an existing object of the same class. This is important because without copy constructors, new objects would have to be created and another objects value would then need to be assigned to the new object. This would waste time. A copy constructor saves time, makes the programme more efficient and ensures that the code looks more organised. Furthermore, pointers have also been used within the programme. Similarly, pointers help reduce the length of code which therefore makes it look more organised and easier to understand. This is also an advantage because is helps the programme to run faster, making it more efficient. Pointers allow variables defined outside a function to be accessed allowing the programme to be much more effective.

Class structure:

Parent class- `class Base_Image{`

Derived classes- `class Large_Image : public Base_Image {` `class Ref_Image : public Base_Image`

3. NNS Algorithm

The Nearest-Neighbour search algorithm which has been used is a sum of squared difference algorithm. Sum of squared difference is where a matrix containing the values for the cluttered image are taken away from the matrix where the values of the Wally image are contained. The difference for the x values between the Wally matrix and cluttered matrix is calculated followed by the difference between the two matrices for the y values. The difference is then squared, and the sum is found. The for-loop loops through the small image of Wally's 2D vector index and the if statement checks whether the small image is equal to 255 (greyscale value, binary is 1). This is important because if the image is equal to 255, then the sum of squared difference is calculated. The difference between the two images is found, each value is squared and then they are added together in order to obtain the SSD value.

```
for (int i = 0; i < Wally_X; i++)
        {
                for (int j = 0; j < Wally_Y; j++)
                {
                        if (Wally_Matrix[i] != 255 && Wally_Matrix[j] !=255)
                        {
                                Differencei = Wally_Matrix[i] - Cluttered_Matrix[i];
                                Differencej = Wally_Matrix[j] - Cluttered_Matrix[j];
                                DifferenceSum = Differencei + Differencej;
                                sum = DifferenceSum * DifferenceSum;
                                SSD = sum + SSD;
                        }
        }
        }
        return SSD;
```

This part of code goes through the cluttered scene using the values for the image of Wally to try and find the similarity between the cluttered image and the Wally image. The first for loop deals with the x values and the second with the y values before finding the similarity between the two images.

```
for (int i = 0; i < Cluttered_X - Wally_X; i = i + Area_X)
{
        for (int j = 0; j < Cluttered_Y - Wally_Y; j = j + Area_Y)
        {
                Area_Vec = (*Cluttered_Scene).getArea(i, j, Wally_X, Wally_Y);
```

Following this, the sum of square difference is calculated by the function coding the sum of squared difference calculation from above, leading to object variables being set to the start values for x and y and the sum of squared difference value.

```
                if(i == 0 && j == 0)
                {
```

```
            (*Vec_Object).setX_Start(i);
            (*Vec_Object).setY_Start(j);
            (*Vec_Object).setSSD(SSD);
         }
```

An if statement then checks to see whether the sum of squared difference value is smaller than the SSD value which has previously been stored. If it is then then SSD value replaces the previously stored object, if not the loop continues. This is important because the smallest SSD value is the best match for Wally.

```
        if (SSD < (*Vec_Object).getSSD())
        {
            (*Vec_Object).setSSD(SSD);
            (*Vec_Object).setX_Start(i);
            (*Vec_Object).setY_Start(j);
        }
    }
}
```

The final part of the algorithm takes the smallest value which has just been calculated and updates the cluttered scene data. It does this because this is where Wally has been calculated to be. The Highlight_Image function is then called and overloaded with the information on Wally's location. This function draws a black box around where Wally has been calculated to be:

```
Large_Image_Data = (*Cluttered_Scene).Highlight_Image((*Vec_Object).getX_Start(),
(*Vec_Object).getY_Start(), Large_Image_Data, Wally_X, Wally_Y, Cluttered_Y);
```

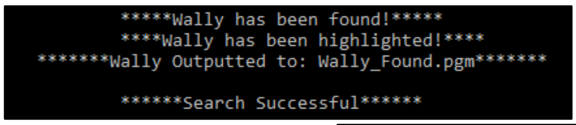
Overall, the algorithm runs very efficiently and smoothly. The pixels are checked individually as well which means that the programme should take longer to run than if each value was to be checked in a different way. The nearest-neighbour search algorithm is easy to implement whilst efficiently calculating the SSD within the program which is important because we want to program to run as quickly as possible without any errors.

4. Results

The desired outcome of the programme is to find Wally within the large cluttered image. This is important because if the output of the code is a box around Wally on the large image then this therefore shows that the code has run successfully and does what is required of it.

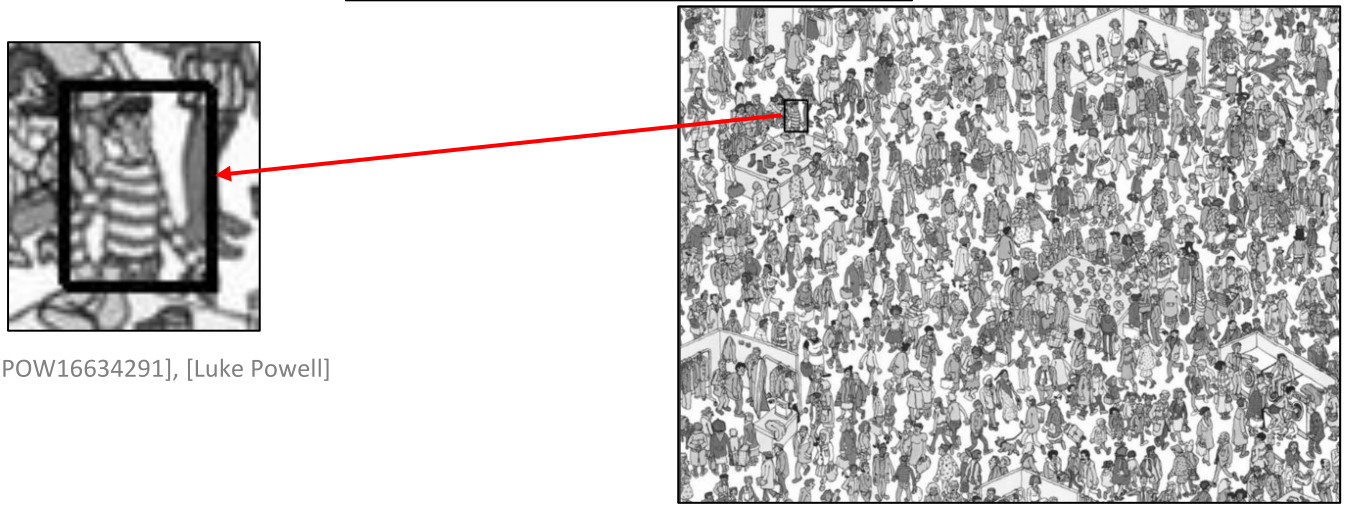
4.1 Best matching

The best match is of course Wally being found. If Wally has been found, then the code clearly indicates this and informs the user that the results have been output to a .pgm file called 'Wally_Found.pgm' This tells the user that the programme has successfully run, encouraging the user to open the file which will show them where Wally is within the large cluttered scene.



```
         *****Wally has been found!*****
          ****Wally has been highlighted!****
*******Wally Outputted to: Wally_Found.pgm*******

         ******Search Successful******
```

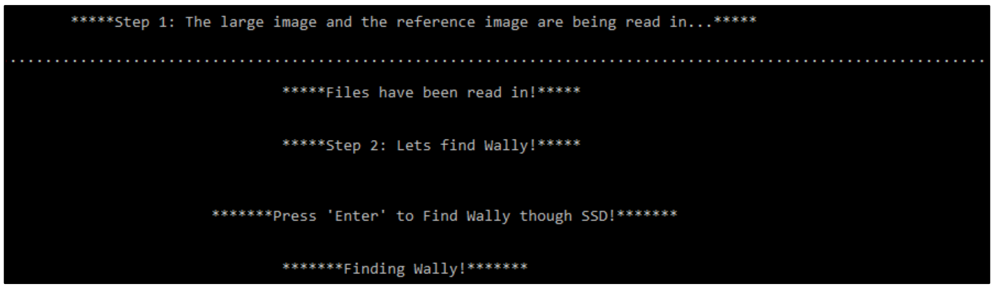


[POW16634291], [Luke Powell]

Above is the .pgm file which successfully shows that the programme has output a black box around Wally within the cluttered image. This is important because it shows that the programme successfully works. The best match of Wally which was found was successful due to the fact that it found Wally exactly though using nearest-neighbour search and more particularly, a sum of squared difference algorithm.

5. Discussion & Conclusion

Overall, the programme was a success due to the fact that the output of the code is what was expected of it. Linking back to the introduction, the purpose of the code was explained. This was to take a small reference image of Wally and match it against a large cluttered image in order to find the location of Wally in said large image. This is important because the code does just that. The output of the code is exactly what was require of it. A task which was present was how to make the programme user friendly when the output is in a separate external file. To solve this problem, it was important to set a goal of building a programme with a user friendly interface which keeps the user informed about what the programme is doing whilst running. I believe that this task was successfully solved. This is because it is important that the user is involved in some way. The user is not directly involved with the main task of the programme which is for the algorithm to find Wally, so it was crucial to maintain attention through the implementation of a loading bar to show that the programme was working and reading in the .txt files as well as prompts to keep the user active whilst the programme is running and regular updates about what the program is actually doing at certain points. This is vital as it shows the user that something is happening.



```
     *****Step 1: The large image and the reference image are being read in...*****

..........................................................................................

                        *****Files have been read in!*****

                        *****Step 2: Lets find Wally!*****


                  *******Press 'Enter' to Find Wally though SSD!*******

                        *******Finding Wally!*******
```

On the other hand, something which worked well was the nearest neighbour search algorithm. The sum of squared difference algorithm correctly calculated the difference between two matrices (one being the small image of Wally and the other being a constantly changing matrix of the large image). The algorithm successfully squared the sum before working out whether it was less that the previously derived sum. This concluded with the smallest sum being found therefore finding Wally. The efficient algorithm along with the use of classes breaking the project up into multiple source files and a header file allowed the compilation and running of the programme to be smooth and not take to long. The programme runs successfully in under a minute, checking each pixel separately. Overall, I believe that the programme was a success as the main task at hand was to reference a small image of Wally to a large cluttered image, find his location in the large image and show this through highlighting Wally which has successfully been completed by the programme.