

02132 ASSIGNMENT 1 REPORT

SOFTWARE IMPLEMENTATION OF A CELL DETECTION AND COUNTING ALGORITHM IN C

Group: 22

Mikkel Arn Andersen s224187

Niclas Juul Schæffer s224744

Rasmus Kronborg Finnemann Wiuff s163977

github.com/rwiuff/02132Assignment1 

October 1st

1 WORK DISTRIBUTION

Explain here who has done what, for both implementation and report.

Table 1: Work distribution on the project

Name	Implementation tasks	Report tasks
Mikkel Arn Andersen	Erosion, Detection optimisation	
Niclas Juul Schæffer	Detection, Detection optimisation	
Rasmus Wiuff	Program structure, Detection, Memory Optimisation	Sections 2, 3 and 5.1

2 DESIGN

2.1. DATASTRUCTURES

There are two kinds of information needed in the program. Incrementers of various sorts for counting and keeping checks on processes. These are mainly of type `int` as these behave neatly for integer counting, even though they have a larger drain on memory. For storing the image there exists two arrays: One for the original 3-channel image (provided by `cbmp.c`) as well as a flattened grey scale array: `unsigned char tmp_image[BMP_WIDTH][BMP_HEIGHT]`. This is practical as only one conversion is needed to produce a grey scale array, as opposed to convert back and forth before saving. The original image array is only edited when a location is marked with a cross. The temporary array is used in all other calculations and changes at every step in the process. Another good reason for using array is the fact that C always pass them by-reference to functions, meaning no copying is needed when mutating the original arrays.

2.2. PROGRAM STRUCTURE

Firstly the image needed to be converted into an array with the provided function. Another function deals with flattening the array by averaging the 3 channels into one. Hereafter the program follows the provided algorithmic structure. A function applies a binary threshold onto the array. A do-while loop checks if any white pixels are left. As long as that is not the case a round of erosion is made using a separate function. Hereafter a detection function does the following:

1. Iterate over pixels
2. Use a function to check if there is a valid cell
3. Increment a counter and draw a cross on the original image array using another function
4. Erase the captured cell area (set intensity to nought)
5. Print information about the location to the console

Then the while loop repeats until no pixels are left. *Explain here what the design process was. Explain how you structured your code (e.g., divide functionality into functions, decide the functions prototypes, etc.). Explain how you decide to represent and store data (e.g., what representation, what buffers to use, etc.). Motivate the design decision you made. Lastly the main method prints information about runtime, counted cells, etc. Table 2 shows functions and functionality.*

Table 2: Functions and their division of labour

Step	Function	Effect
0	read_bitmap	Imports bitmap as a three dimensional array
1	Greyscaling	Populates two dimensional array with greyscale image
2	Binary threshold	Applies a binary threshold on the greyscale image
3a	Pixel check	Check for white pixels
3b	Erosion	Erodes image using erosion element
3c	Detection	Run detection routine
3c1	Draw	Draw cross on original array
3c3	Erase	Erase area with detected cell
4	write_bitmap	Exports image array as bitmap

3 IMPLEMENTATION

Briefly discuss the implementation in C of your design. Explain how you have exploited the C language in the context of embedded system to implement the algorithm. You can include some code snippets if these are relevant to explain certain aspects of the implementation.

4 OPTIMIZATIONS AND ENHANCEMENTS

4.1. OTSU'S METHOD

Otsu's method was extensively looked into in order to get an automated optimal intensity for the binary threshold. Following was discovered:

- Otsu's method works best if two distinct classes of intensity exists, i.e. two distinct peaks in a intensity histogram over the image.
- Most pictures in the set does not contain two distinct peaks.
- Existing implementations where tried to find optimal intensities ([EBI package for R](#), [OpenCV \(using Python\)](#)). These tools pointed to intensities way higher than empirically studied values (115-150 against the provided 90 or tested 80-110).

A way of overcomming the problem could be through local thresholding, where ranges of intensities are left out, however time was already wasted on this rather time consuming endeavour.

Explain here the optimizations and enhancements you have implemented in order to improve cell detection rate, execution time, memory use, and/or other algorithm characteristics you considered relevant. Explain what was the motivation (thinking-process) behind the optimizations and enhancements you implemented.

5 TEST AND ANALYSIS

5.1. PROOF OF CONCEPT VERSION

This section will discuss the non-optimised version of the program.

5.1.1. Functionality tests Table 3 shows an overviev over detection on the various images.

5.1.2. Execution time analysis

5.1.3. Memory use analysis *Report here the results from the test and analysis you have carried out according to the assignment instructions. You need to at least address the following: functionality tests, execution time analysis, memory use analysis. For each optimization/enhancements you implement, you need to perform tests to prove its validity. If you have implemented optimization/enhancements which do not give the expected benefits, describe why it does not work. Remember to discuss the results from the test and analysis you have carried out, do not just present them, but explain and argue their meaning.*

Table 3: Functionality test on provided images. ✓ indicates 300 cells accounted for. ✗ indicates otherwise.

	Easy	Medium	Hard	Impossible
1	✓	✗	✗	✗
2	✗	✗	✗	✗
3	✓	✗	✗	✗
4	✗	✗	✗	✗
5	✗	✗	✗	✗
6	✗	✗	✗	N/A
7	✗	✗	✗	N/A
8	✓	✗	✗	N/A
9	✗	✗	✗	N/A
10	✗	✗	✗	N/A

6 REFERENCES

List here the references that you have used (if any) It can be articles or websites where you have found inspiration and understanding.