ENSP YAOUNDE, DEPARTMENT OF COMPUTER ENGINEERING

Project Report: Image Processing

Yuhala Peterson Jr. Dohbila

17 janvier 2018

1 Preamble

This mini-report presents the work done in the course of our Image Processing Class Project, which required us to implement the different image processing algorithms seen during our Image Processing Course.

The objective of this document is to present all the steps necessary for the proper testing of all the algorithms, which were implemented in the C++ Programming Language.

The Netbeans IDE was used for programming and for the sake of simplicity, all the images used during this project are in `.pgm` format.

2 General Structure of the C++ Project

The project is made up of four(4) principal files :

- Two(2) **header files** : `Functions.h` and `Image.h`
- Two(2) **.cpp files** : `Image.cpp` and `main.cpp`

The `Image.cpp` file contains the class definition of our **Image Class**. All the image processing algorithms are applied on **Image Objects**. The `Image.h` file is the associated header file

The `Functions.h` header file contains the prototypes of most of the algorithms implemented in this project.

The `main.cpp` file contains the definitions of all the function prototypes defined in the `Functions.h` header file. It also contains the `main` function, which is the entry point to our program.

3 User Manual

In this section we will present all the steps necessary for testing the different algorithms.

The main project directory is the `ImageProcessing` folder contained in the file `Yuhala_Peterson_Jr.zip` submitted.

It is important to note that this is a `Netbeans Project`. As such, using Netbeans IDE would ease the testing of the program. Nevertheless, if you do not have Netbeans installed, you could use the linux command line to compile the program, assuming you have g++ compiler installed.

A total of 26 algorithms were implemented. These algorithms are grouped as such :

- **Logic Operations** (6)
  - — Image Binarization based on Otso's thresholding algorithm
  - — Logic NOT
  - — Logic AND
  - — Logic NAND
  - — Logic OR
  - — Logic XOR
- **Mathematical Operations** (3)
  - — Image Addition
  - — Image Subtraction
  - — Image Multiplication
- **Image Histogram, Luminance, and Contrast Enhancement algorithms** (6)
  - — Linear Contrast Enhancement
  - — Linear Contrast Enhancement with Saturation
  - — Image Histogram
  - — Histogram Equilization
  - — Contrast Calculation
  - — Luminance Calculation
- **Convolution and Filters** (9)
  - — Gauss Filter
  - — Smoothing Filter
  - — Image Sharpening
  - — Robert Filter

  - Prewitt Filter
  - Sobel Filter
  - Edge Detection
  - Laplacien Convolution
  - Image Mirror
- **Interpolation** (1)
  - Nearest Neighbour Interpolation
- **Morphological Operations** (1)
  - Image Erosion

An implementation of each algorithm exists in the main function (commented); To implement a given algorithm, uncomment the line corresponding to the given algorithm. All the images used are located in the `ImageProcessing/images/`.

3.1 How to use the program

Let's suppose we want to **binarize** the image `lena.pgm` in the folder : `ImageProcessing/images/`.

1. Using a text editor or an IDE, open the `main.cpp` file. In the **main function**, uncomment the line corresponding to image binarization (otsuBinarize). ie

   ```
   Image imageIn = readImage("images/lena.pgm");

   Image imageOut=imageIn.otsuBinarize();
   ```

   Leave the other algorithm implementations commented, so as not to produce the wrong `imageOut` object. Save the changes to the `main.cpp` file.
2. If you are using the linux command line, open a linux terminal in the folder : `ImageProcessing` and type the following command to compile and run the program :

   ```
   sudo g++ main.cpp Image.cpp -o app && ./app
   ```
   If you are using Netbeans IDE (or another IDE), simply run the project.
3. If the above command runs without any errors (ignore the warnings), then the image has been binarized and written to the file :
   `ImageProcessing/output.pgm`
   You can open this output file to view the output and make sure its coherent.

Lets suppose now that we want to do **image addition**. ie We want to add the images : `lena.pgm` and `aya.pgm`
The only difference here is we have two input image files.

1. In the **main function**, add a second input image by uncommenting the appropriate line. ie
```
Image imageIn = readImage("images/lena.pgm");
Image imageIn2 = readImage("images/aya.pgm");
Image imageOut = addition(imageIn, imageIn2);
```
2. Save the changes made to the `main.cpp` file and in the linux terminal, run the program :
```
sudo g++ main.cpp Image.cpp -o app && ./app
```
3. If there are no errors, the two images will be added (pixel wise) and the result written to the file :
`ImageProcessing/output.pgm`
Open the file to view the output and check if its coherent.

These same steps should be repeated accordingly when testing the various algorithms listed.

3.2 Troubleshooting errors

- Most **segmentation faults** are caused by problems at the level of reading an image. In case you encounter a segmentation fault, recheck the path to the input image and ensure that it exists.

4 Conclusion

In terms of productivity, a good number of algorithms seen in class were implemented, and the project helped us to master various concepts on image processing.

However, some difficulties were encountered at the level of image segmentation, image interpolation, and image morphological operations. Nevertheless, enough research was done on the above concepts and we learnt as much.