



2021-2022

## **Communication and Mini-Project**

LEBANESE UNIVERSITY

FACULTY OF ENGINEERING – 3<sup>rd</sup> BRANCH

Specialty: Electrical and Electronics Engineering

4<sup>th</sup> year of engineering

Done by:

**Mohamad Ousseily & Karim Sleiman**

---

### **Determination of Blood group using Image processing.**

Under the supervision of:

**Dr. Youssef HARKOUS**

## **Acknowledgements**

Firstly, we would like to thank the director of the faculty of engineering Dr. Hussein Shreim, the Dean of our faculty Dr. Rafic Younes, our supervisor and head of the department of electrical engineering Dr Youssef Harkous. We are deeply thankful for them for giving us this opportunity to build up our knowledge and experience in the midst of the situation we live in. We also are very thankful for the presence of the members of the jury.

Mohamad and Karim.

## Table of Contents:

1. List of figures.....	3
2. General Introduction.....	4
3. Chapter 1: Development.....	5
3.1 Introduction .....	5
3.2 Biological approach of blood types.....	6
3.3 Image processing.....	8
3.3.1 Pre-Processing techniques: Red Green Blue (RGB) and grayscale.....	8
3.3.2 Thresholding .....	9
3.3.3 Morphological operations .....	13
3.4 Conclusion.....	19
4. Chapter 2: Results.....	20
5. General Conclusion .....	27
6. References .....	28
7. Appendix .....	29

## 1. List of figures

Figure 1: Red Blood Cell Compatibility table .....	6
Figure 2: Agglutination tests & results .....	7
Figure 3: Three dimension representation of RGB color image.....	8
Figure 4: RGB image .....	9
Figure 5: Grayscale image using rgb2grayscale.....	9
Figure 6: Multi-level & single-level thresholding .....	10
Figure 7: Thresholding techniques .....	11
Figure 8: Otsu's Binarization compared to global thresholding.....	12
Figure 9: Visualization of Otsu's algorithm on two weights. ....	13
Figure 10: Euclidean 3-space .....	14
Figure 11: Simple Dilation operation of given H & I matrices. ....	15
Figure 12: example of dilation technique on an input image (upper left) by the octagonal structure (lower right).....	15
Figure 13: Visual representation of the erosion example .....	16
Figure 14: Erosion example on a given image input (upper left) .....	17
Figure 15: Image I, composed of set (object) A and background. Structuring element, B. Translations of B while being contained in A. (A is shown dark for clarity). Opening of A by B. ....	18
Figure 16: Image I, composed of set (object) A, and background. Structuring element B. Translations of B such that B does not overlap any part of A. (A is shown dark for clarity). Closing of A by B.....	18
Figure 17: Image processing general steps.....	19
Figure 18: The GUI used .....	21
Figure 19: When the user press « input Image” button. ....	21
Figure 20: The user chooses an image. ....	22
Figure 21: Transformation of RGB image to grayscale level.....	22
Figure 22: Threshold applied.....	23
Figure 23: Erosion operation applied .....	24
Figure 24: Segmentation is applied .....	25
Figure 25: Result of applying the tests .....	26
Figure 26: Quantification button is pressed before input image .....	26

## **2. General Introduction**

Blood tests are crucial in all medical domains. A common method is used, a sample of a certain patient is taken, and a number of tests are done on this sample. From the results, we can obtain the blood type of the patient. The time of the test is important for a faster procedure of the tests. Statistics show that there are many deaths of patients that needed blood transfused to them fast. A normal blood test takes 15 minutes to 1 hour in order for it to be accurate. Other problems could also occur in the normal blood group test, one of them is human error. In a huge hospital, human error is inevitable. Image processing is a group of methods that are used by softwares. These at their turns will try to make the human's work for them. Image processing is very wide; we plan to use its concepts to make a software that will read the blood type of a certain person. However, why do we need to have an automatic system for this simple procedure? We look forward to engineer a system that does the readings of a blood group, from an input blood image. For that we will start, firstly by explaining, in detail, how the human blood works and how the normal blood test is made. We will then encounter the different types of image processing techniques that are used in this project. We will explain every one of these techniques and their varieties. Secondly, we will try to implement the image processing techniques on an input blood image. This implementation will be done using MATLAB. We will explain every step we took to be able to find results.

### **3. Chapter 1: Development**

#### **3.1 Introduction**

Blood group tests are widely made in all medical domains. Image processing techniques are used here to try to solve these problems. It is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is also a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Here the output should be the blood type of a certain obtained blood image. In image processing, two types of methods are generally used. There is analogue and digital processing. Analogue image processing is used for hard copies, like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing uses computers. It helps manipulate data and analyze them. Three general phases exist in an image processing technique: pre-processing, enhancement and display, and information extraction. In this part, we will explain the scientific problem we face firstly, and then we will explain the steps of the image processing algorithms starting with pre-processing techniques, thresholding, morphological operations and lastly quantification.

### 3.2 Biological approach of blood types

The determination of blood is important in almost all medical procedures, from blood transfusion simply, to transplantation medicine and many operations that are more complex. Two systems coexist in the blood groups. First, the ABO group system, this system is responsible for the differentiation of the patients' blood group: A, B, AB, or O blood group. Second, The Rhesus (Rh) System that associates the patient with a negative blood group or a positive one. It should be established that a patient can't receive any type of blood from another patient. Therefore, the following rules should be followed to avoid incompatibility: The negative Rhesus can give the positive Rhesus blood group, The O blood group can give all the other groups, and A blood cannot give B blood groups (and vis-versa). On the red blood cell, there exists in many situations extracellular membranes which associates with the ABO group, a red blood cell can either have an A antigen on its surface, a B antigens, both antigens or neither, these are the A, B, AB and O blood groups respectively. In fact, from this red blood cell's microscopic representation we can deduce a table (*figure 1*) of compatibility of all possible donor/recipient situations we can encounter.





































Recipient	Donor							
	O-	O+	A-	A+	B-	B+	AB-	AB+
O-	✓	✗	✗	✗	✗	✗	✗	✗
O+	✓	✓	✗	✗	✗	✗	✗	✗
A-	✓	✗	✓	✗	✗	✗	✗	✗
A+	✓	✓	✓	✓	✗	✗	✗	✗
B-	✓	✗	✗	✗	✓	✗	✗	✗
B+	✓	✓	✗	✗	✓	✓	✗	✗
AB-	✓	✗	✓	✗	✓	✗	✓	✗
AB+	✓	✓	✓	✓	✓	✓	✓	✓

*Figure 1: Red Blood Cell Compatibility table*

It should also be noted that these tests are very important, in a case of error; the patient might get in a very bad condition, and sometimes be the cause of death. With that being

said, we need a trustworthy system that detects the blood type of a certain patient. The normal blood test, which is being used in many hospitals, is very simple. The blood test consists of a chemical reaction test, if this reaction so called agglutination occurs, then the test is positive. Therefore, three tests are made, Anti – A blood group test, Anti – B blood group test and Rh – chromosome test.

These tests can determine the presence of Antigens A, B and Rh respectively. The chemical reaction is seen with a microscope, or it must be visualized by a specialist. In that order, the exact blood group of the patient is determined in the labs.

Anti-A	Anti-B	Anti-D	Control	Blood Type
				O-pos
				O-neg
				A-pos
				A-neg
				B-pos
				B-neg
				AB-pos
				AB-neg
				Not valid

*Figure 2: Agglutination tests & results*



### 3.3 Image processing

#### 3.3.1 Pre-Processing techniques: Red Green Blue (RGB) and grayscale

Pre-processing techniques requires more knowledge about color image. A pixel color in colored image is a combination of three colors Red, Green and Blue (RGB). The RGB color values are represented in three dimensions XYZ.

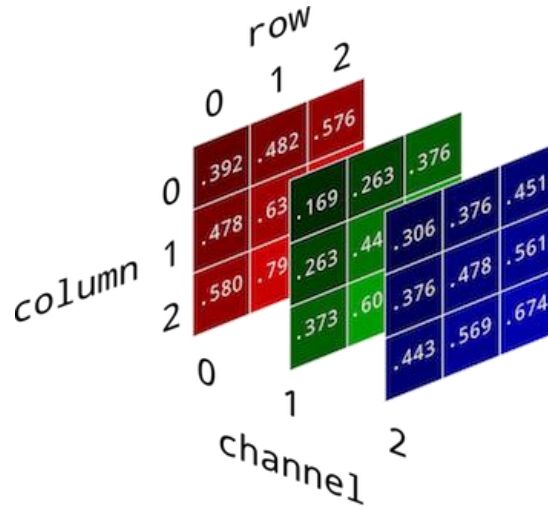


Figure 3: Three dimension representation of RGB color image

The quality of a color image depends on the number of bits a digital device could support. The basic color image represented by 8 bit, the high color image represented using 16 bits, the true color image represented by 24 bit, and the deep color image is represented by 32 bit. The number of bits decides the maximum number of different colors supported by the digital device. If each Red, Green, and Blue occupies 8 bit then the combination of RGB occupies 24 bit and supports 16,777,216 different colors. The 24 bit represents the color of a pixel in the color image. The grayscale image has represented by luminance using 8 bits value [8].

The luminance of a pixel value of a grayscale image ranges from 0 to 255. The conversion of a color image into a grayscale image is converting the RGB values (24 bit) into grayscale value (8 bit). There are many ways to convert from a color image to grayscale.

An example of transforming an image from RGB to grayscale is a MATLAB function named:

`rgb2gray(RGB)` [15]; it converts RGB values to grayscale values by forming a weighted sum of the R, G, and B components:

$$G_{ray-bit} = 0.2989 * R + 0.5870 * G + 0.1140 * B$$



*Figure 4: RGB image*



*Figure 5: Grayscale image using rgb2grayscale*

### 3.3.2 **Thresholding**

Also called binarization, image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast. It is crucial in image processing techniques because it separates two main parts of an image.

Thresholding could be used on a single level, in other words, it could be done according to a single threshold  $T$ . It could also be used on multilevel thresholding; multiple thresholds  $T_i$  are set to obtain an output of multiple parts. These parts are usually differentiated by color.

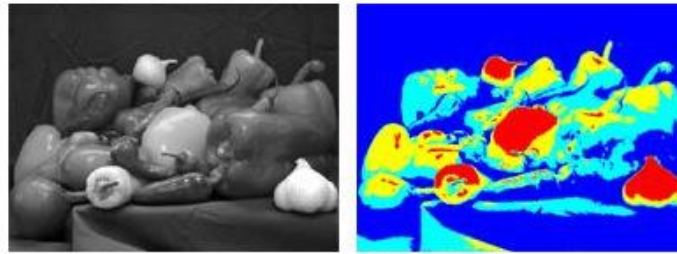


Image thresholding using multi-level thresholding



Image thresholding using a set level

*Figure 6: Multi-level & single-level thresholding*

Since this image processing technique is important, it has many algorithms. Many methods exist; these are categorized into how the threshold is chosen and into how this threshold is used to finalize the process.

The threshold value can be automatically determined using any of these methods:

- *Histogram shape-based methods*: where the peak values of this histogram are used to calculate the threshold value. In a very simple way, this value could be the peak value of the histogram.
- *Clustering-based methods*: where the gray-level samples are clustered in two parts as background and foreground generally, it could also be clustered into two other groups. The clustering techniques are many, one of them, which is widely used, is the K-means algorithm.
- *Entropy-based methods*: result in algorithms that use the entropy of the foreground and background regions, the cross-entropy between the original and binarized image, etc.

- *Other methods:* There are other ways to find the threshold value, using object shape methods. Spatial methods that require higher-order probability distribution. And other local methods, which need for each certain number of pixel, a threshold value.

Once the threshold value is obtained, different algorithms can be applied to obtain a binarized image. From Binary threshold, to truncate and to zero thresholding techniques as seen in the (figure 7).

Binary	$\text{dst}(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
Inverted Binary	$\text{dst}(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$
Truncated	$\text{dst}(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$
To Zero	$\text{dst}(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
To Zero Inverted	$\text{dst}(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$

*Figure 7: Thresholding techniques*

These methods are simple and easily implemented and used. However, they can produce errors when too much noise is present. This is why we sometimes need automatic thresholding. It is a great way extract useful information encoded into pixels while minimizing background noise.

This is accomplished by utilizing a feedback loop to optimize the threshold value before converting the original grayscale image to binary. The idea is to separate the image into two parts: the background and foreground, using the following algorithm.

Select initial threshold value, typically the mean 8-bit value of the original image.

- Divide the original image into two portions;
- Pixel values that are less than or equal to the threshold; background
- Pixel values greater than the threshold; foreground
- Find the average mean values of the two new images
- Calculate the new threshold by averaging the two means.

This is a general algorithm for the majority of thresholding techniques, but one of the most famous method is called Otsu's Binarization. This technique uses Gaussian filtered image inputs and not original noisy images. Thus, it can create its own histogram, and then deduce the binarized image.

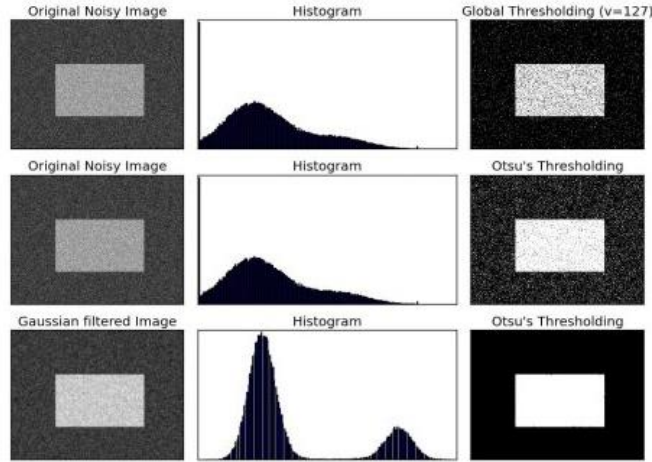


Figure 8: Otsu's Binarization compared to global thresholding.

In global thresholding, we used an arbitrary chosen value as a threshold. In contrast, Otsu's method avoids having to choose a value and determines it automatically. Consider an image with only two distinct image values (bimodal image), where the histogram would only consist of two peaks. A good threshold would be in the middle of those two values.

Similarly, Otsu's method determines an optimal global threshold value from the image histogram. This section demonstrates a Python implementation of Otsu's binarization to show how it actually works. Since we are working with bimodal images, Otsu's algorithm tries to find a threshold value ( $t$ ) which minimizes the weighted within-class variance given by the relation:

$$\sigma_{\omega}^2(t) = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)], \text{ where:}$$

- $\omega_0(t)$  and  $\omega_1(t)$  are two classes weights, set from the start, or computed from a histogram of intensity levels.
- $\mu_0$  and  $\mu_1$  are the respective means of each weight.

The algorithm then goes as follows:

1. Compute histogram and probabilities of each intensity level.
2. Set up initial  $\omega_0(0)$  and  $\omega_1(0)$  accordingly.
3. Step through all possible thresholds.
  1. Update  $\omega_0(t)$  and  $\omega_1(t)$ ,  $\mu_0$  and  $\mu_1$ .
  2. Compute  $\sigma_\omega^2(t)$ . Repeat step 3.
4. Desired threshold corresponds to the maximum of  $\sigma_\omega^2(t)$ .

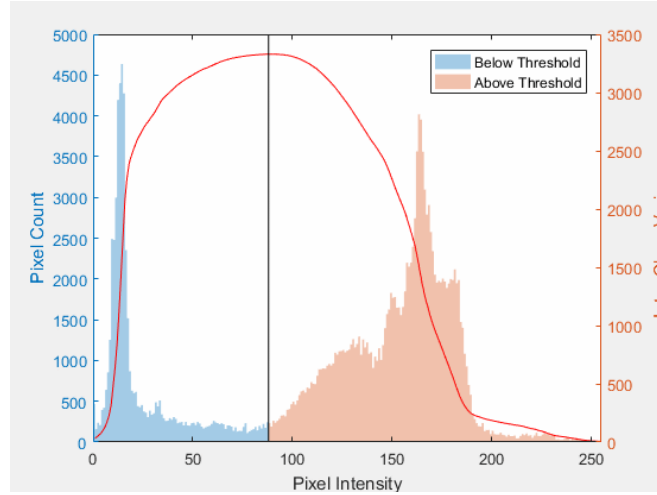


Figure 9: Visualization of Otsu's algorithm on two weights.

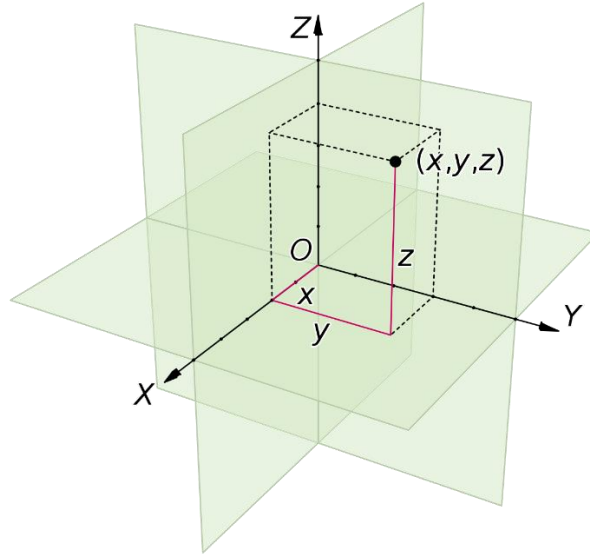
It actually finds a value of  $t$ , which lies in between two peaks such that variances to both classes are minimal.

It should be noted that the methods we set here are used in global or local thresholding, the difference is that global thresholding takes the whole image, whereas the local thresholding takes separate parts of the same image to work with, each of them is used in its own beneficial domain.

### 3.3.3 Morphological operations

Mathematical morphology provides an approach to the processing of digital images, which is based on shape. Appropriately used, mathematical morphological operations tend to simplify image data preserving their essential shape characteristics and eliminating irrelevancies. The language of mathematical morphology is that of set theory. Sets in mathematical morphology represent the shapes, which are manifested on binary or gray tone images. The set of all the black pixels in a black and white image, (a binary image) constitutes a complete description of the binary image. Sets can be represented in Euclidean spaces.





*Figure 10: Euclidean 3-space*

Sets in Euclidean 2-space denote foreground regions in binary images. Sets in Euclidean 3-space may denote time varying binary imagery or static gray scale imagery as well as binary solids. Sets in higher dimensional spaces may incorporate additional image information, like color, or multiple perspective imagery.

We begin the discussion of morphology by studying two operations: dilation and erosion. These operations are fundamental to morphological processing [14].

**Dilation** is the morphological transformation, which combines two sets using vector addition of set elements.

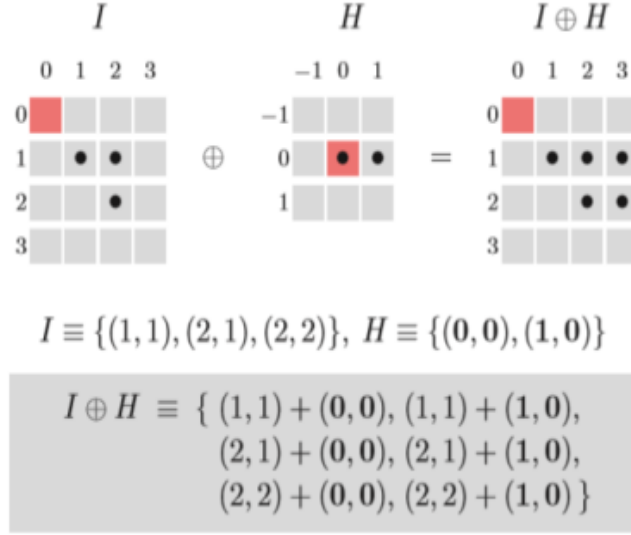
If  $A$  and  $B$  are sets in  $N$ -space ( $E^N$ ) with elements  $a$  and  $b$ , respectively,  $a = (a_1, \dots, a_N)$  and  $b = (b_1, \dots, b_N)$  being  $N$ -tuples of element coordinates, then the dilation of  $A$  by  $B$  is the set of all possible vector sums of pairs of elements, one coming from  $A$  and one coming from  $B$ .

Definition 1: Let  $A$  and  $B$  be subsets of  $E^N$ . The dilation of  $A$  by  $B$  is denoted by  $A \oplus B$  and is defined by

$$A \oplus B = \{c \in E^N \mid c = a + b \text{ for some } a \in A \text{ and } b \in B\}.$$

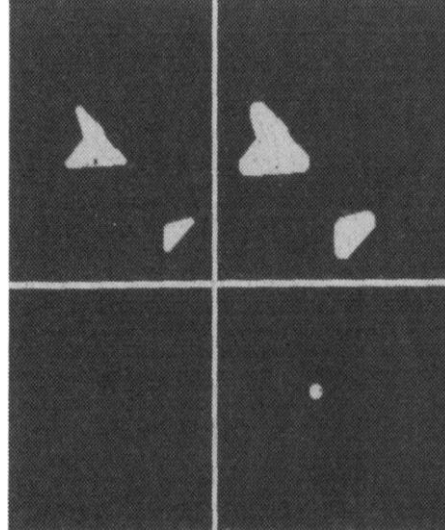
Dilation by small squares is a neighborhood operation easily implemented by adjacency connected array architectures (grids). This image processing technique is also called: fill, expand or grow technique.

We can see in the example that illustrates an instance of the dilation operation:



*Figure 11: Simple Dilation operation of given  $H$  &  $I$  matrices.*

We can also check this other example which clearly shows why the dilation is called a “grow” or an “expand” technique. In the following (*figure 11*), the upper left shows the input image consisting of two objects.



*Figure 12: example of dilation technique on an input image (upper left) by the octagonal structure (lower right)*

The lower right shows the octagonal structuring element. The upper right shows the input image dilated by the octagonal structuring element. In fact, the octagonal structuring was



used to expand the input image by the size of its radius. That shows how dilation is an addition operation.

**Erosion** is the morphological dual to dilation. It is the morphological transformation, which combines two sets using the vector subtraction of set elements.

If A and B are sets in Euclidean N-space, then the erosion of A by B is the set of all elements  $x$  for which  $x + b \in A$  for every  $b \in B$ .

Definition 2: The erosion of A by B is denoted by  $A \ominus B$  and is defined by

$$A \ominus B = \{x \in E^N \mid x + b \in A \text{ for every } b \in B\}.$$

Erosion is sometimes also called shrinking or reducing operation, which is the opposite of dilation in logic. It should not be taken as the exact opposite of dilation in the image processing world. Example: This illustrates an instance of erosion. Let us take A and B such as:

$$A = \{(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (3, 1), (4, 1), (5, 1), \}$$

$$B = \{(0, 0), (0, 1)\}$$

$$A \ominus B = \{(1, 0), (1, 1), (1, 2), (1, 3), (1, 4)\}$$

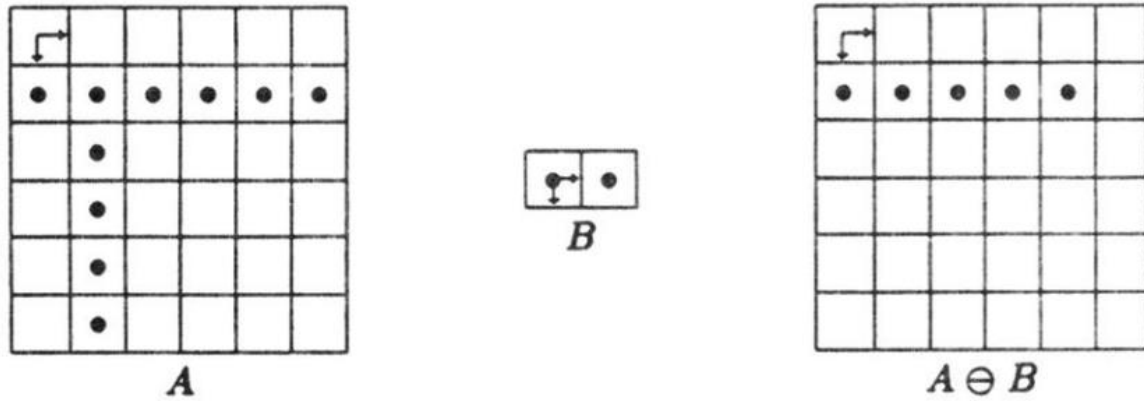


Figure 13: Visual representation of the erosion example

Since erosion can remove elements from a given input and change the shape of the main image, we can use it to remove unwanted elements in general. This following example shows how:

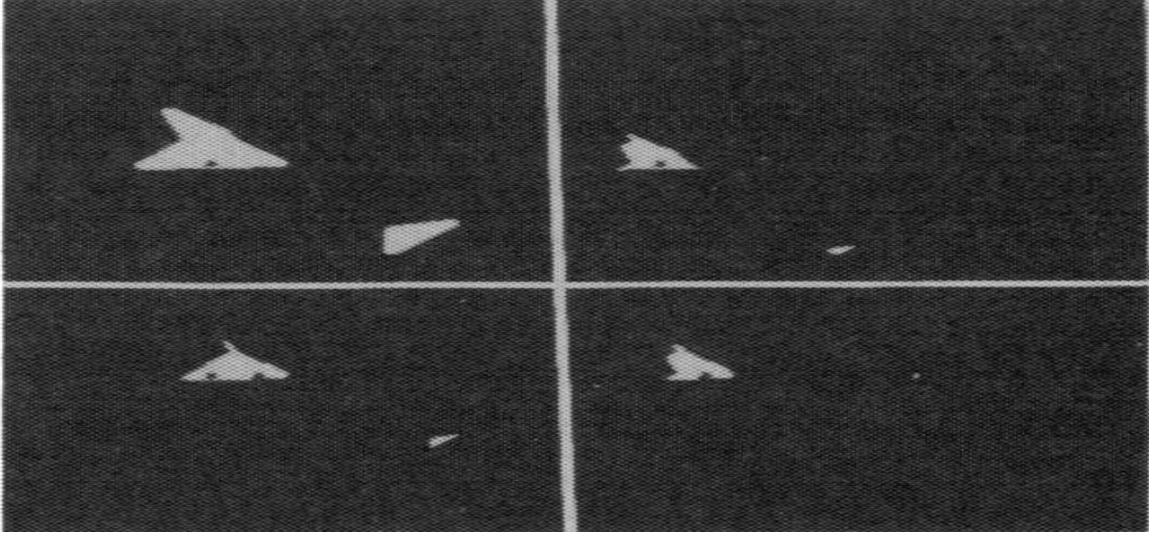


Figure 14: Erosion example on a given image input (upper left)

The upper left shows the input image consisting of two blobs. The upper right shows the input image eroded by the structuring element  $\{(0, 0), (-14, 0)\}$ . The lower left shows the input image eroded by the structuring element  $f(0, 0) (0, -14)\}$ . The lower right shows the input image eroded by the structuring element  $I(0, 0), (0, -14), (-14, 0)\}$ .

Erosion can also represent a difference of elements A & B, thus definition 2 becomes:

$$A \ominus B = \{x \in E^N \mid \text{for every } b \in B, \text{there exists an } a \in A \text{ such that } x = a - b \}.$$

In practice, dilations and erosions are usually employed in pairs, either dilation of an image followed by the erosion of the dilated result, or image erosion followed by dilation. In either case, the result of iteratively applied dilations and erosions is an elimination of specific image detail smaller than the structuring element without the global geometric distortion of unsuppressed features. The most important combinations of dilation and erosion are opening and closing. Opening generally smooths the contour of an object. Closing also tends to smooth sections of contours, but, as opposed to opening, it generally eliminates small holes, and fills gaps in the contour.

The opening of set A by structuring element B, denoted by:  $A \circ B$ , is defined:

$$A \circ B = (A \ominus B) \oplus B$$

Thus, the opening A by B is the erosion of A by B, followed by a dilation of the result by B. Similarly, the closing of set A by structuring element B, denoted  $A \bullet B$ , is defined as

$$A \bullet B = (A \oplus B) \ominus B$$

Which says that the closing of A by B is simply the dilation of A by B, followed by erosion of the result by B.

In (figure14 & 15), we can see the examples of opening and closing respectively, in a very simplified yet clear way.

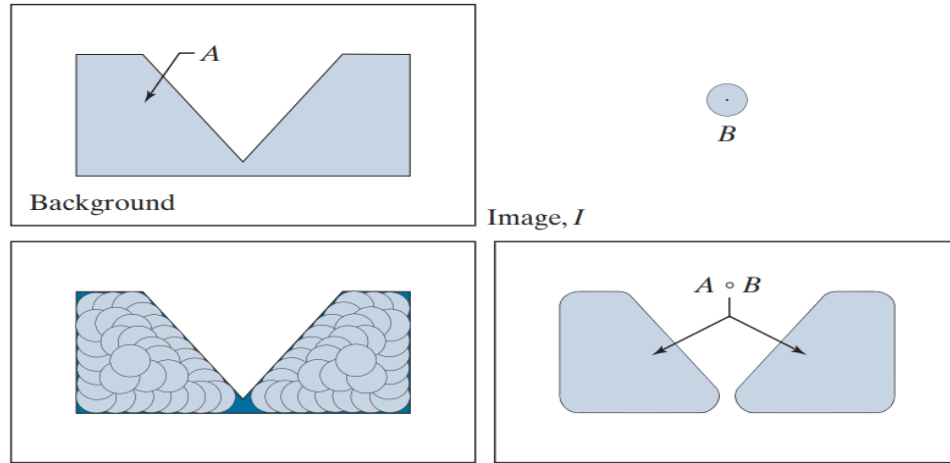


Figure 15: Image  $I$ , composed of set (object)  $A$  and background. Structuring element,  $B$ . Translations of  $B$  while being contained in  $A$ . ( $A$  is shown dark for clarity). Opening of  $A$  by  $B$ .

Opening an image with a disk-structuring element smooths the contour, breaks narrow isthmuses, and eliminates small islands and sharp peaks or capes.

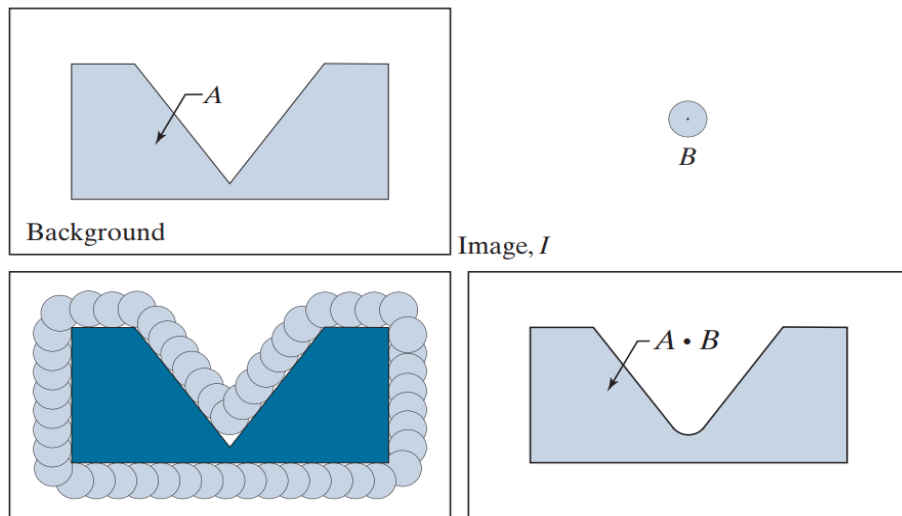
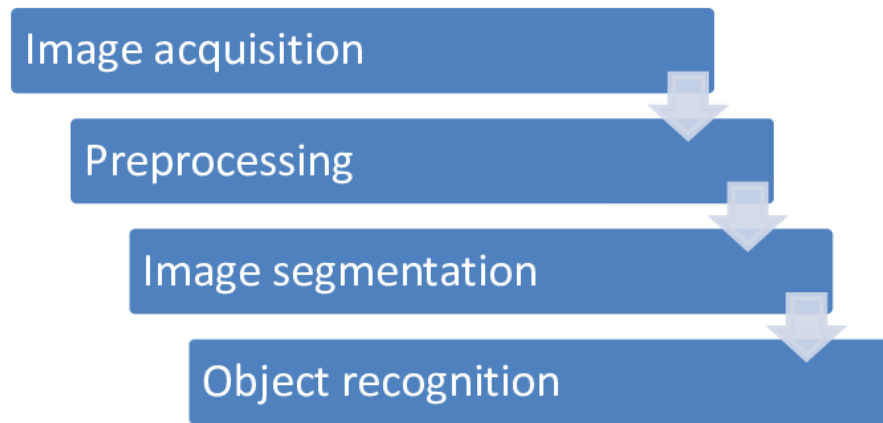


Figure 16: Image  $I$ , composed of set (object)  $A$ , and background. Structuring element  $B$ . Translations of  $B$  such that  $B$  does not overlap any part of  $A$ . ( $A$  is shown dark for clarity). Closing of  $A$  by  $B$ .

Closing an image with a disk-structuring element smooths the contours, fuses narrow breaks and long thin gulfs, eliminates small holes, and fills gaps on the contours [3].

### 3.4 Conclusion

After seeing the problem, we face in blood testing, we chose to try solving them using image processing techniques. These techniques are based on pre-processing techniques, which prepares an input image to grayscale or to a system that is simpler to work with. We also saw the thresholding techniques, how they can form a pre-processed image into a background/foreground formation. We explained the different morphological operations. These operations are mainly dilation and erosion, by using them in the correct implementation we can apply opening and closing algorithms on certain elements.



*Figure 17: Image processing general steps*

Since image processing need these steps to work, we will try to implement these algorithms in the next chapter in order to find results to our problem.

## **4. Chapter 2: Results**

### **4.1 Introduction**

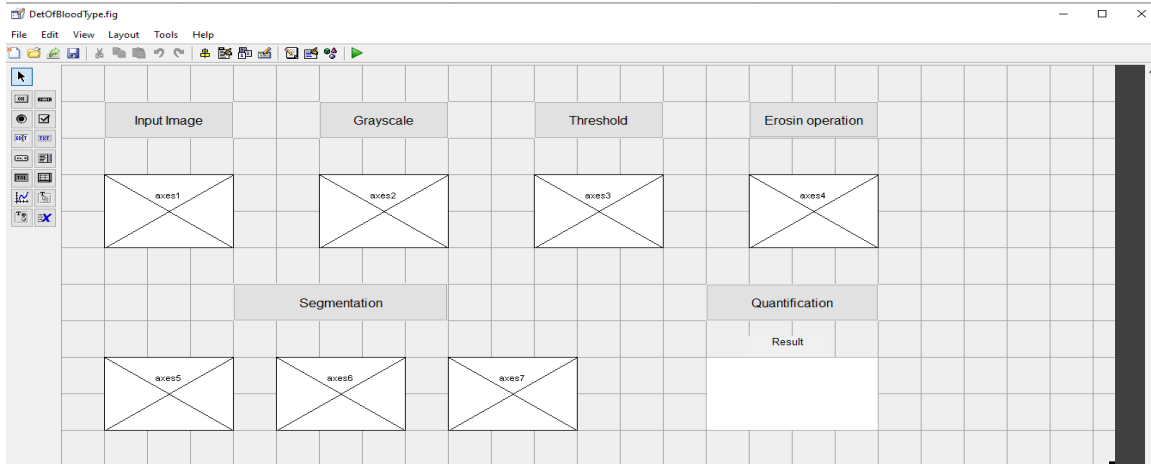
The project is done using MATLAB R2021a; the user will obtain the results after doing some steps that will be discussed below using a GUI. The steps to be discussed are obtaining the image as input, transform it to grayscale level, perform Otsu's thresholding, apply morphological operations, then apply image segmentation, and finally do a quantification to obtain the result. We are working on images of blood that went through the first process of blood testing, three tests on the same blood's patient were made and a picture was taken of the results (Anti-A test, Anti-B test and Anti-Rh test). We will see later that agglutination could occur, which the program should be able to identify.

All these steps will be implemented as buttons in the GUI, and the result of each step will be displayed as an image. The detailed MATLAB code can be found in the (*APPENDIX*).

## 4.2 Results and Discussion

### Sep 1: Building the GUI

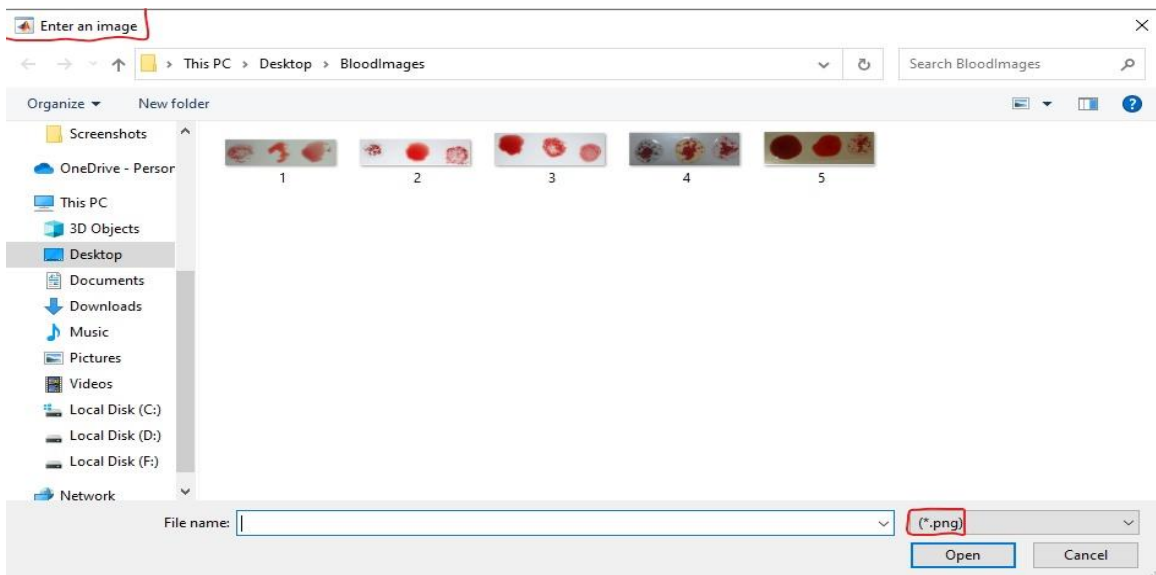
Using “guide” command in MATLAB a GUI window will open, where we will drag and drop buttons to be programmed later, axes to put the output of the function executed by each button, and text to display the final result.



*Figure 18: The GUI used*

### Step 2: input image and resizing

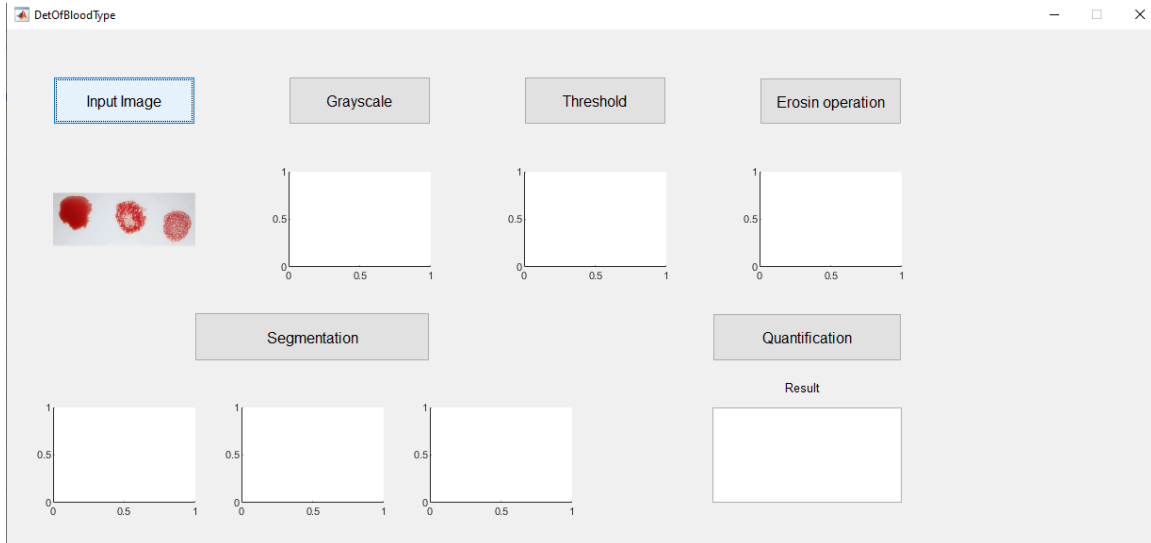
The user first will press on the button “input Image”, then a folder containing the blood test images of type ‘PNG’ will open.



*Figure 19: When the user press « input Image” button.*

After choosing the image, it will be resized to 320 x 120 by using:

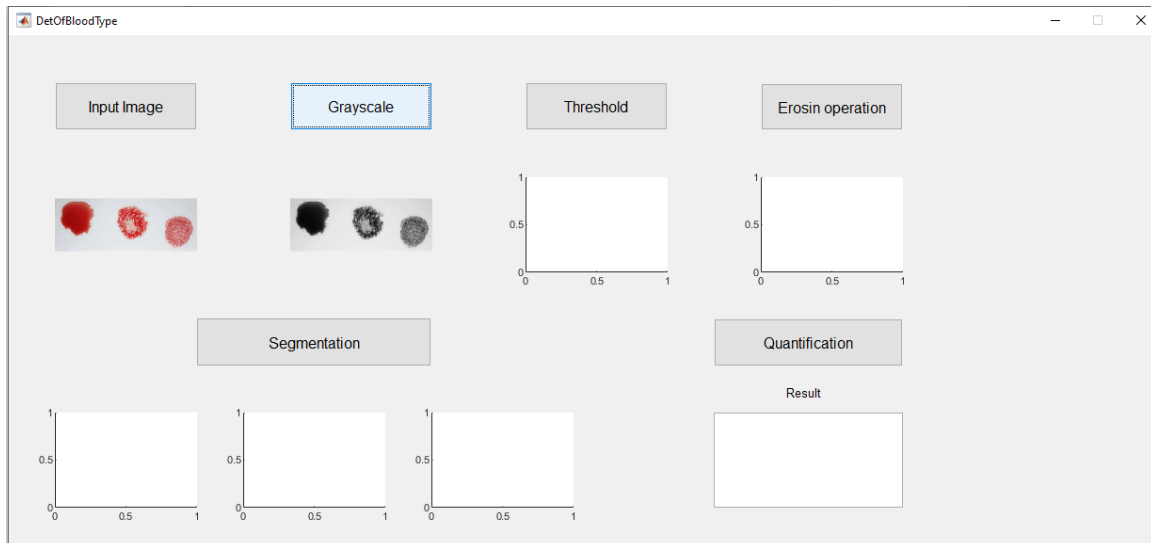
```
imresize(image, [columns rows]);
```



*Figure 20: The user chooses an image.*

### **Step 3: Transform the RGB image to grayscale**

In this step, we chose a simple algorithm. To do this transformation we will do a color plane extraction, in which we will extract each time a color from the RGB image starting by extracting the red then green then blue. This extraction is based on the most dominant color in the color image.



*Figure 21: Transformation of RGB image to grayscale level.*

As a result, we will get a grayscale image with dominance of bright pixels if the extracted color is dominant in the original image and darker image otherwise, and by counting the number of dark pixel with value for example lower than 50. Then the image having the most number of dark pixel will be used as grayscale image in the next steps.

#### **Step 4: Apply Thresholding**

The image is transformed to binary using global threshold algorithm of Otsu is, such that the foreground is represented with white pixels and the background with black pixels.

We will start the algorithm by obtaining the histogram of the image, then the sum of all pixels, after that the probability of each value in the histogram is calculated.

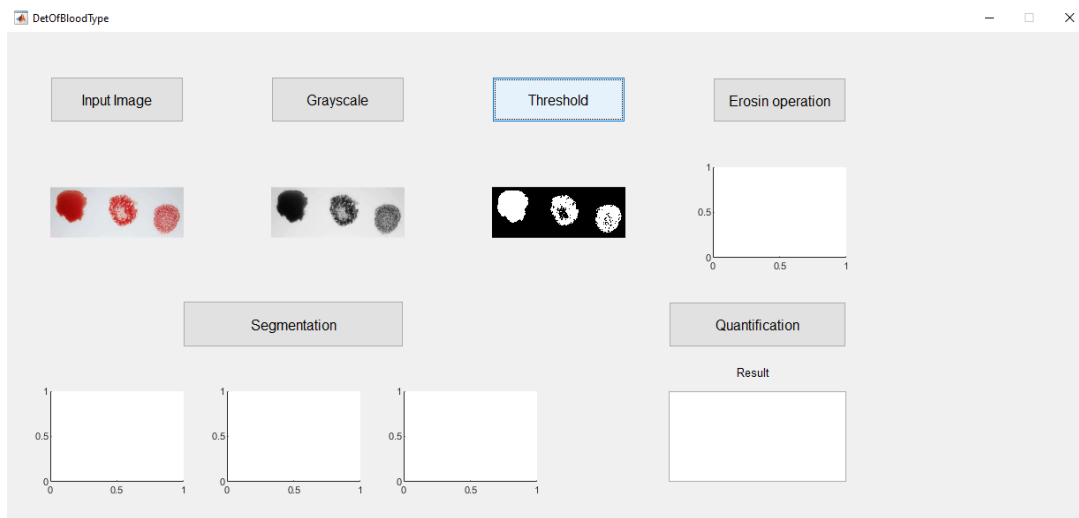
Then for a variable named k chosen between pixel's values 50 and 250, the image is divided into two clusters, and then the weight of each cluster is calculated by summing the probabilities of the first element of the histogram to the element of value k for the first cluster, and by summing the others for the second cluster.

Then the mean of each cluster is calculated: the sum of dot products between each value and its probability divided by the weight of that cluster.

Then we will search the variance by applying the formula:

$$\text{Var}(k) = w_b * w_f * ((U_b - U_f)^2);$$

Then we will get the position L for which the variance is maximum and then the threshold value will be L-1 because the positions in array var(K) starts from 0.

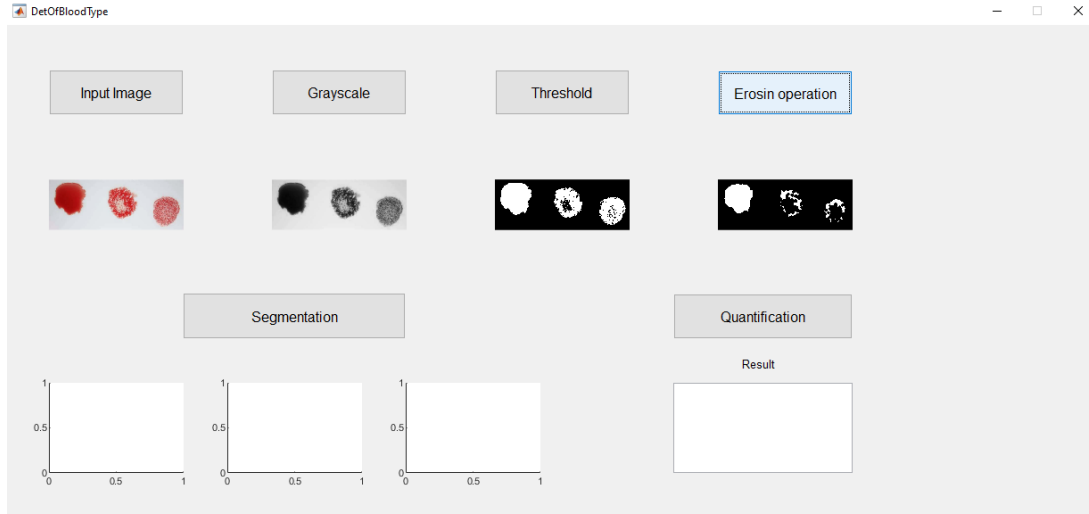


*Figure 22: Threshold applied.*



### **Step 5: apply morphological operation**

In order to eliminate the noise pixels, and to divide the image that represent agglutination to parts to let the quantification in the next steps easier the erosion is applied with a disk of radius equals 4 pixels.



*Figure 23: Erosion operation applied*

### **Step 6: Image segmentation**

The goal is to detect the agglutination, so the image is segmented into three fragments each one represents a test. Again, the tests are respectively Anti-A, Anti-B then Anti Rh.

In order to crop each part, we must precise a square in the image, which is represented by two rows and two columns, so the algorithm is developed as follow:

To obtain the first column, a counter will count the number of pixels in each column until it obtains two pixels, which for example will be noted  $c$ . So the first column  $c1$  is chosen by taking  $c-1$  to make sure to crop the full image, so for that we used two for loop in which in each iteration the value of the counter will be returned to zero.

To obtain the second column, same as above the loop will start from  $c1$  and will stop until the number of pixels obtained in a column named  $c$  is equal to zero. Then the next column named  $c2$  will be obtained such that:  $c2 = c + 1$ . To make sure that there is no black column in between the same test due to erosion operation, a condition will take place confirming that the distance between  $c1$  and  $c2$  must be large enough.

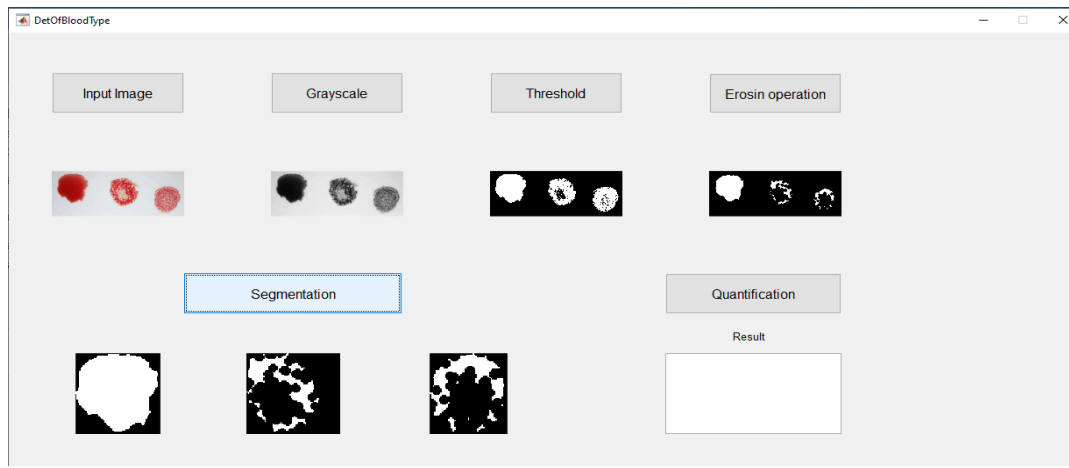
To obtain the first row, we will count the number of pixels in each row from  $c1$  to  $c2$ , when we obtain a number of pixels higher or equal to 2 in a row named  $r$ , the first row named  $r1$  is obtained by  $r1=r-1$ .

To obtain the second row, the same as obtaining the second column and the first row, by taking in consideration a good distance between the two rows.

After that, the original binary image named *s* for example will be cropped and the first segment named *crop1* will be obtained by the MATLAB syntax:

```
crop1 = s([r1 : r2] [c1 : c2]);
```

Furthermore, the same is done to the other segments.



*Figure 24: Segmentation is applied*

### **Step 7: Quantification**

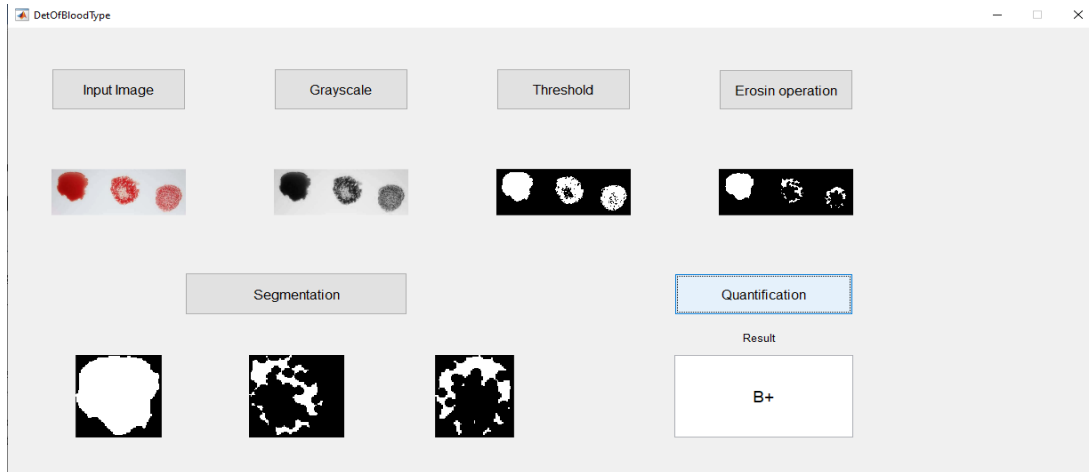
This is the final step in which we will obtain the blood type. The goal is to detect the agglutination, so on each image two tests are performed and it will be sufficient if one of them is true. The first test is to count the number of pieces in each image by using a function in MATLAB `bwlabel(image)` which one its return type is the number of connected objects found in the image, if that number is greater or equal to three then the agglutination is detected. The second test is to count the percentage of white pixels in each image and if that percentage is less than 60%, so agglutination occurs.

For example, in the image below:

The first image in the segmentation step has 1 piece and 68.9% of white pixels.

The second image has 9 pieces and 18.91% of white pixels.

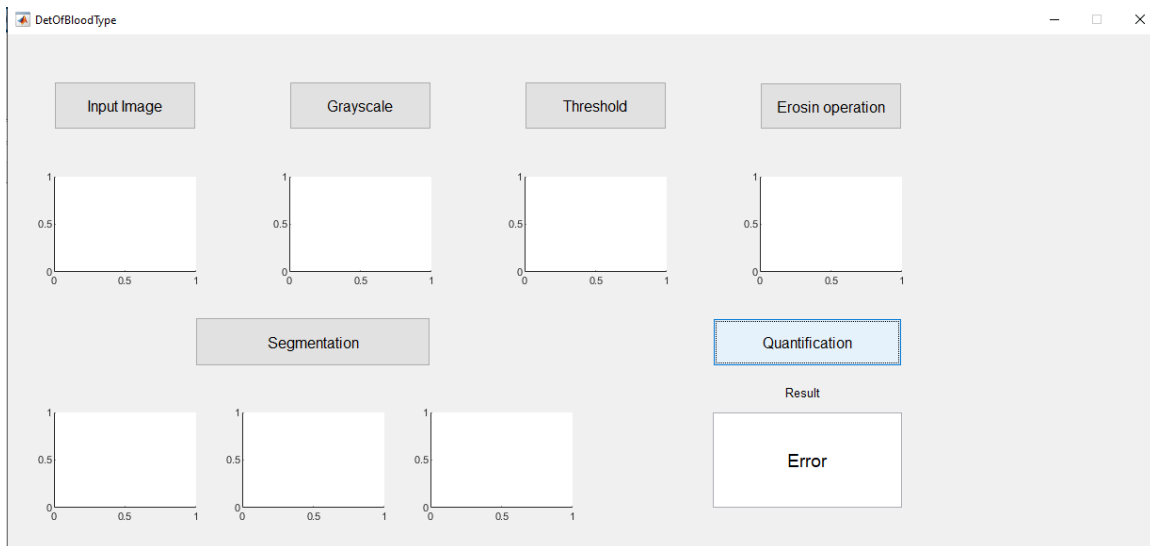
The third image has 7 pieces and 23.43% of white pixels.



*Figure 25: Result of applying the tests*

### **If the quantification button is pressed before other steps**

An error message will be displayed.



*Figure 26: Quantification button is pressed before input image*

## **4.3 Conclusion**

To conclude, we started by preparing the colored image to grayscale in order for the threshold to be simpler. A binarized image is then segmented after the erosion process. Finally, we implemented a quantification algorithm that reads the pixels in number and percentage, which gives us numbers to work with, these numbers deduce the blood group of the patient's blood. Nevertheless, this is one example of a blood test, many other

examples of other blood tests can be made using the same MATLAB code, and obviously gives us the right blood group. These blood images inputs can be found at the end of the (*APPENDIX*).

## **5. General Conclusion**

In the end, we can say that we arrived to a good program that finds the blood type of an input blood image. We started by explaining the concepts that we needed to use in the code. These concepts, which were mainly image processing techniques, were strategically used in Chapter 2. We used the logic of how blood tests work; we adapted the pre-processing technique that was the best visually. Otsu's method, erosion and segmentation were used accordingly. From that, quantification was simple and we arrived to results. All these were explained in Chapter 1 in details. The issue was to be able to implement the problem we face on MATLAB. With that being said, we can say that this system can, in our opinion, be used in medical domains for two main reasons. The system could make much less error than humans could, this will be mostly shown in crowded hospitals for example. Second, it would be faster and easier to save the data directly for each patient, which saves time for the medical staff to do assignments that are more important.

## 6. **References**

- [1] Jadwiga Rogowska, “Handbook of Medical Image Processing and Analysis” (Second Edition), 2009
- [2] E., Umbaugh, Scott, “Digital Image Processing and Analysis with MATLAB and CVIPtools”, Third Edition (third ed.), 2017.
- [3] Rafael C. Gonzalez, Richard E. Woods, “Digital Image Processing”, (Fourth Edition), 2018.
- [4] Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". IEEE Trans. Sys. Man. Cyber. 9 (1): 62–66.
- [5] Ankita Dalvi, Hanu Kumar Pulipaka, “Determination of Blood group using Image Processing”, International Journal of Scientific & Engineering Research Volume 9, Issue 3, March-2018.
- [6] Ana Ferraz, “Automatic System For Determination of Blood Types Using Image Processing Techniques”, A Physical, Volume 172, Issue 1, December 2011.
- [7] G. Ravindran, T. Joby Titus, M. Pravin, P. Pandiyan, “Determination and Classification of Blood Types using Image Processing Techniques”, International Journal of Computer Applications (0975 – 8887) Volume 157 – No 1, January 2017.
- [8] C. Saravanan, “Color Image to Grayscale Image Conversion”, Second International Conference on Computer Engineering and Applications, 2010.
- [9] Parthima Guruprasad, Kushal S Mahalingpur, Manjesh.T.N, “OVERVIEW OF DIFFERENT THRESHOLDING METHODS IN IMAGE PROCESSING”, TEQIP Sponsored 3rd National Conference on ETACC, June 2020.
- [10] Ashutosh Kumar Chaubey, “Comparision of The Local and Global Thresholding Methods in Image Segmentation”, World Journal of Research and Review (WJRR), Volume-2, Issue-1, Pages 01-04, January 2016.
- [11] Rafael C. Gonzalez, “Digital Image Processing”, by doxygen 1.8.13, 2022.
- [12] Emrys Kirkman, “Blood groups”, Published by Elsevier Ltd, ANAESTHESIA AND INTENSIVE CARE MEDICINE 8:5, 2007.
- [13] Hossein Adibi, Nader Khalesi, Hamid Ravaghi, Mahdi Jafari, and Ali Reza Jeddian, “Root-Cause Analysis of a Potentially Sentinel Transfusion Event: Lessons for Improvement of Patient Safety”, Acta Medica Iranica, 2012;50(9): 624-631, © Tehran University of Medical Sciences, 2012.
- [14] Robert M. Haralick, Stanley R. Sternberg, Xinhua Zhuang, “Image Analysis Using Mathematical Morphology”, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-9, NO. 4, JULY 1987
- [15] Mathworks.com

## 7. Appendix

### **MATLAB code:**

```
function varargout = DetOfBloodType(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn',   @DetOfBloodType_OpeningFcn, ...
                  'gui_OutputFcn',    @DetOfBloodType_OutputFcn, ...
                  'gui_LayoutFcn',    [], ...
                  'gui_Callback',     []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before DetOfBloodType is made visible.
function DetOfBloodType_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.output = hObject;

guidata(hObject, handles);

function varargout = DetOfBloodType_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

% --- Executes on button press in InputImage.
function InputImage_Callback(hObject, eventdata, handles)
%Clear all axes.
cla(handles.axes1,'reset');
cla(handles.axes2,'reset');
cla(handles.axes3,'reset');
cla(handles.axes4,'reset');
cla(handles.axes5,'reset');
cla(handles.axes6,'reset');
cla(handles.axes7,'reset');
set(handles.Result,'String','');
[file,path,indx] = uigetfile('.png','Enter an
image','C:\Users\Mohamad\Desktop\BloodImages');
s=append(path,file);
```

```

x=imread(s);
x=imresize(x,[120 320]);
axes(handles.axes1);
imshow(x);

% --- Executes on button press in ToGrayscale.
function ToGrayscale_Callback(hObject, eventdata, handles)
x=getimage(handles.axes1);
axes(handles.axes2);
i1=0;
i2=0;
i3=0;
xR=x(:,:,1);%red color extraction.
[rows,cols]=size(xR);

for c=1:cols
    for r=1:rows
        pixval = xR(r,c);
        if(pixval<50)
            i1=i1+1;
        end
    end
end
xG=x(:,:,2);%green color extraction.
for c=1:cols
    for r=1:rows
        pixval = xG(r,c);
        if(pixval<50)
            i2=i2+1;
        end
    end
end
xB=x(:,:,3);%blue color extraction.
for c=1:cols
    for r=1:rows
        pixval = xB(r,c);
        if(pixval<50)
            i3=i3+1;
        end
    end
end
I=[i1,i2,i3];
m=max(I);%get the darker image.
if m==i1
    imshow(xR);
else
    if m==i2
        imshow(xG);
    else
        imshow(xB);
    end
end

% --- Executes on button press in Threshold.
function Threshold_Callback(hObject, eventdata, handles)
z=getimage(handles.axes2);

```

```

axes(handles.axes3);
H=imhist(z);
TP=sum(H);
P=H/TP;
%Otsu Threshold
for k= 50:250
    wb=sum(P(1:k));
    wf=sum(P(k+1 : end));
    Ub=dot(0:k-1,P(1:k))/wb;
    Uf=dot(k:255,P(k+1:256))/wf;
    Var(k)=wb*wf*((Ub-Uf)^2);
end
Max_Variance=max(Var);
L=find(Var==Max_Variance);
Threshold=L-1;
imshow(z<Threshold);

% --- Executes on button press in DoErosion.
function DoErosion_Callback(hObject, eventdata, handles)
y=getimage(handles.axes3);
axes(handles.axes4);
se=strel('disk',4);
y=imerode(y,se);
imshow(y);

% --- Executes on button press in Segmentation.
function Segmentation_Callback(hObject, eventdata, handles)
s=getimage(handles.axes4);
axes(handles.axes5);
[rows,cols]=size(s);

%First crop.
count1=0;
for c=1:cols
    for r=1:rows
        if(s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1>2)
        c1=c-1;
        break;
    end
    count1=0;
end

x=c1+1;
count1=0;
for c=x:cols
    for r=1:rows
        if(s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1==0)
        c2=c+1;

```



```

        if(c2>c1+60)
            break;
        end
    end
    count1=0;
end

count1=0;
for r=1:rows
    for c=c1:c2
        if(s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1>2)
        r1=r-1;
        break;
    end
end

count1=0;
x=r1+1;
for r=x:rows
    for c=c1:c2
        if(s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1==0)
        r2=r+1;
        if(r2>r1+50)
            break;
        end
    end
    count1=0;
end
crop1=s([r1:r2],[c1:c2]);
imshow(crop1);

%Second crop
axes(handles.axes6);
count1=0;
for c=c2:cols
    for r=1:rows
        if(s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1>2)
        c3=c-1;
        break;
    end
    count1=0;
end

x=c3+1;

```

```

count1=0;
for c=x:cols
    for r=1:rows
        if (s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1==0)
        c4=c+1;
        if(c4>c3+70)
            break;
        end
    end
    count1=0;
end
count1=0;

for r=1:rows
    for c=c3:c4
        if (s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1>2)
        r3=r-1;
        break;
    end
end

count1=0;
x=r3+1;
for r=x:rows
    for c=c3:c4
        if (s(r,c)==1)
            count1=count1+1;
        end
    end
    if(count1==0)
        r4=r+1;
        if(r4>r3+40)
            break;
        end
    end
    count1=0;
end
crop2=s([r3:r4],[c3:c4]);
imshow(crop2);

%Third crop
axes(handles.axes7);
count1=0;
for c=c4:cols
    for r=1:rows
        if (s(r,c)==1)
            count1=count1+1;
        end
    end
end

```

```

        end
        if(count1>2)
            c5=c-1;
            break;
        end
        count1=0;
    end

    x=c5+1;
    count1=0;
    for c=x:cols
        for r=1:rows
            if(s(r,c)==1)
                count1=count1+1;
            end
        end
        if(count1==0)
            c6=c+1;
            if(c6>c5+40)
                break;
            end
        end
        count1=0;
    end
    count1=0;

    for r=1:rows
        for c=c5:c6
            if(s(r,c)==1)
                count1=count1+1;
            end
        end
        if(count1>2)
            r5=r-1;
            break;
        end
    end

    count1=0;
    x=r5+1;
    for r=x:rows
        for c=c5:c6
            if(s(r,c)==1)
                count1=count1+1;
            end
        end
        if(count1==0)
            r6=r+1;
            if(r6>r5+40)
                break;
            end
        end
        count1=0;
    end
    crop3=s([r5:r6],[c5:c6]);
    imshow(crop3);

```

```

% --- Executes on button press in GetResult.
function GetResult_Callback(hObject, eventdata, handles)
I1=getimage(handles.axes5);
[L,n1]=bwlabel(I1);
I2=getimage(handles.axes6);
[L,n2]=bwlabel(I2);
I3=getimage(handles.axes7);
[L,n3]=bwlabel(I3);
s1=sum(I1(:));%nb of white pixels
np1=numel(I1);%nb of pixels
p1=s1/np1;%percentage of white pixels in the first image
s2=sum(I2(:));
np2=numel(I2);
p2=s2/np2;
s3=sum(I3(:));
np3=numel(I3);
p3=s3/np3;

if(n1==0) || (n2==0) || (n3==0)
    set(handles.Result,'string','Error');
elseif((n1>=3) && (n2<3) && (n3<3)) || ((p1 <0.6) && (p2>0.6) &&
(p3>0.6))
    set(handles.Result,'string','A-');
elseif((n1<3) && (n2>=3) && (n3>=3)) || ((p1>0.6) && (p2<0.6) &&
(p3<0.6))
    set(handles.Result,'string','B+');
elseif((n1<3) && (n2>=3) && (n3<3)) || ((p1>0.6) && (p2<0.6) &&
(p3>0.6))
    set(handles.Result,'string','B-');
elseif((n1>=3) && (n2>=3) && (n3>=3)) || ((p1<0.6) && (p2<0.6)
&& (p3<0.6))
    set(handles.Result,'string','AB+');
elseif((n1>=3) && (n2>=3) && (n3<3)) || ((p1<0.6) && (p2<0.6)
&& (p3>0.6))
    set(handles.Result,'string','AB-');
elseif((n1<3) && (n2<3) && (n3>=3)) || ((p1>0.6) && (p2>0.6)
&& (p3<0.6))
    set(handles.Result,'string','O+');
elseif((n1<3) && (n2<3) && (n3<3)) || ((p1>0.6) && (p2>0.6) && (p3>0.6))
    set(handles.Result,'string','O-');
elseif((n1>=3) && (n2<3) && (n3>3)) || ((p1<0.6) && (p2>0.6) &&
(p3<0.6))
    set(handles.Result,'string','A+');
end

function Result_Callback(hObject, eventdata, handles)

function Result_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

Images to test:

