Communauté
UNIVERSITÉ Grenoble Alpes

Digital analysis of fingerprints

Castells Thibault, Çoba Xhenis, Hernández Mayra,
Pantović Anja

January 18, 2019

Contents

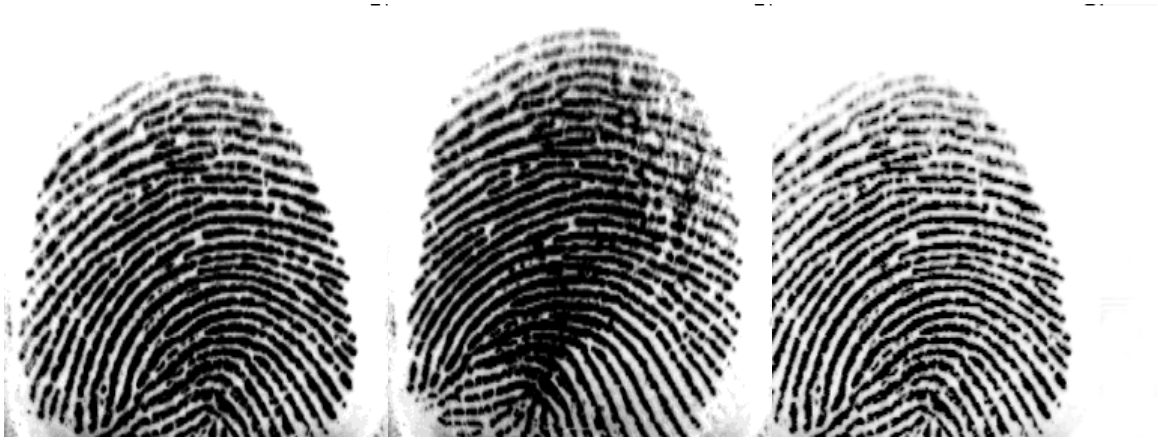
MSIAM

Abstract

To be written in the end..

Introduction
To be written in the end..

The images that we are using to test our work are given below.



(a) *Clean finger* (b) *Warp finger* (c) *Translated x-axis finger*

Figure 1: Test images.

1 Image Loading, Saving and Pixels Manipulation

1.1 Pixel Managing

1.1.1 Description

In this and the proceeding sections we will work with gray-scaled images, thus we will be using one channel. We consider the top left corner of the image to be the origin and we take intensity value to be an integer between 0 and 255.

To work on the image related tasks, we will use CImage library which we find extremely straightforward to learn and use and which is versatile enough to deal with a large number of image types and can deal with any type of pixel values.

As our first task, we will search for the minimum and the maximum in pixel intensity values by simply going through all pixel intensity values. Furthermore, we will cast all pixels values into floating numbers which will be a useful tool in further computations.

By changing some pixel values from the original image, we will create a new one with additional white and black rectangles.

1.1.2 Results

Firstly, we have created a new class called "Image" to enable us to create image related methods efficiently. This class contains its height and width as attributes, and an instance of the class "CImg<unsigned char>".

To search for the minimum and maximum we have implemented a commonly used algorithm shown in the methods: "maxIntensity()" and "minIntensity()". For finding the minimum, we start by it being the highest number pixel intensity value can take, meaning min = 255, and then iterating through all the pixels we update min value whenever we encounter a smaller pixel intensity value. Maximum value is found analogously.

By simple analysis we can see that 0 intensity value corresponds to black while 255 represents white pixels.

In order to insert black and white rectangles into an image we have created the method "drawRect" with five input parameters: the first two of them fix the top left corner while third and fourth give the height and width of the rectangle. The last parameter gives the possibility of choosing the intensity value of the pixels of the whole rectangle.

Furthermore, we implemented 3 methods to perform a symmetry transform of an image. These 3 methods differ from one another by the axis along which we operate the symmetry: x-axis, y-axis and main diagonal axis. According to us, this pixel swapping operation is not a rotation. We can easily observe this through the images below.

(a) Symmetry along x-axis (b) Symmetry along y-axis (c) Symmetry along diagonal-axis

Figure 2: 3 symmetry methods applied to *clean finger*.

1.2 Pressure Simulation

1.2.1 Description

In this section we will be working on the simulation of the pressure variation of the finger knowing its center of pressure. To do this, we will perform a pixel operation on the original image as follows:

$$g(x, y) = c(x, y)f(x, y), \tag{1}$$

where $c(x, y) \in [0, 1]$ is the weight coefficient, $g(x, y)$ is the resulting image and $f(x, y)$ corresponds to the pixel intensity value of the original image at every point (x, y). The aim of the weight function is to decrease pixel intensity as the distance of the center of pressure increases. We will be searching for a function $c(x, y)$ as previously described that gives the best representation of what we have to deal with in practice.

We will be implementing both isotropic and anisotropic cases. Function is said to be **isotropic** if it does not depend on the direction, meaning that it will decrease with the same speed depending only on the distance from the center, whereas an **anisotropic** function is dependent on the direction in which it is observed.

1.2.2 Results

Since we are searching for a function $c(r)$ that vanishes as $r \to \infty$, we found easier and more intuitive reversing corresponding intensity values of white and black, meaning setting white to 0 and black to 255. Furthermore, we have considered normalized euclidean distance.

We posed the new framework in the following way:

$$g(x, y) = 255 - c(x, y)(255 - f(x, y)). \tag{2}$$

The center of pressure is to be chosen as an input parameter. It is represented by a white square in our output images.

First, we took into account isotropic functions.

When searching for those fulfilling the previously described conditions we came up with the following:

- $c(r) = \frac{1}{1+r}$
- $c(r) = e^{-r}$.

When implementing these functions we realized that we need them to decrease more rapidly, hence we introduced additional parameters which tune these functions. By trial and error, experimenting with different tuning coefficients, the following parameters satisfied our desired result the most:

$$c(r) = (e^{-r})^{10} \tag{3}$$

$$c(r) = \frac{1}{1 + r^{15}} \tag{4}$$

$$c(r) = \frac{1}{1 + (\frac{r}{40})^2}. \tag{5}$$

(a) Equation (3) (b) Equation (4) (c) Equation (5)

Figure 3: Isotropic functions applied to *clean˙finger*

We came to the realization that our functions decreased rather uniformly, while we aimed for a function which will preserve as much intensity as possible around the center and vanish more rapidly as we go further. This lead us to the Gaussian function:

$$c(r) = e^{-\alpha r^2}. \tag{6}$$

(a) $\alpha = 0.002$ (b) $\alpha = 0.0002$

Figure 4: Gaussian isotropic function applied to *clean finger*.

Nevertheless, when observing examples of what we get in practice we find that the pixel intensity transform seems anisotropic, following an elliptic shape. Thus, we have adjusted the functions to change depending on

the direction from the center of pressure. To obtain this, we decided to make distance function dependent on the direction, mimicking an ellipse of width a and height b :

$$dist(x, y; x_{center}, y_{center}) = \sqrt{(\frac{x - x_{center}}{a})^2 + (\frac{y - y_{center}}{b})^2}.$$

Gaussian function gave us the best approximation results of what we could encounter in practice. Changing the parameter α we can adjust the vanishing speed as given in Figure 5.

(a) $\alpha = 0.025$ (b) $\alpha = 0.07$ (c) $\alpha = 0.3$

Figure 5: Gaussian anisotropic function applied to *clean finger*.

1.2.3 Conclusion

Coming soon...

2 Geometrical Warps

2.1 Description

Image warping is the process through which the pixel coordinates (x, y) of an image are transformed according to a motion model.

We will present two widely used motions in this section: *translation* and *rotation*.

The equation to be used for *translation* is:

$$f(x, y; p) = f(x, y, a, b) = (x + a, y + b) \tag{7}$$

and for *rotation* we will use the equation:

$$g(x, y; p) = g(x, y, \theta, x_0, y_0) =$$

$$= \cos(\theta)(x_1 - x_0) + \sin(\theta)(y_1 - y_0) + x_0 - \sin(\theta)(x_1 - x_0) + \cos(\theta)(y_1 - y_0) + y_0$$

Given the pixel intensity values of the initial image, we will be able to visualize the transformed image using these motions.

2.2 Results

For the first approach to this task, *translation*, we implemented a method which takes as parameters two integers, a and b. These parameters move each coordinate of the pixels of the image along the x-axis and y-axis respectively.

The second method, *rotation*, takes as parameters the angle of rotation and the coordinates of its center. While implementing the code for this method we had difficulties when the angle of rotation differed from $\pi/2$.

Figure 6: Translation applied to *clean finger*, with parameters (40,44)

Figure 7: Rotation applied to *clean finger*, with a $\pi/4$ angle.

This was due to the loss of information we encountered when casting the computed coordinates, which is illustrated by the image below:

We came up with two effective solutions to overcome the problem of the loss of information.

The first solution we proposed is to use the bilinear interpolation after applying rotation. The *bilinear_interpolation* function allows us to do that by using the nearest pixels holding preserved information.

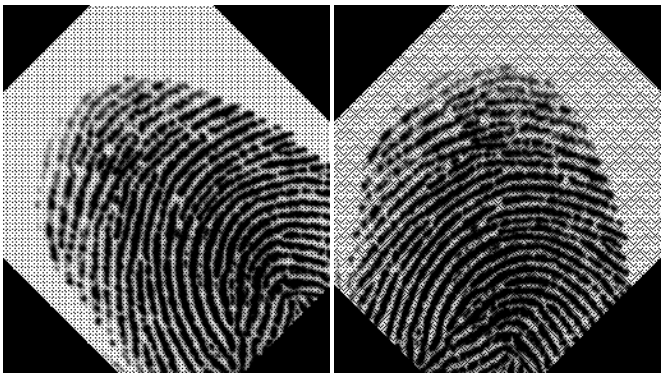
Figure 8 shows the final output.

Figure 8: Rotation and filling of the lost information applied to *clean finger*.

The second solution we proposed is to begin from the coordinates in the rotated picture. First, we computed the exact theoretical coordinates before the rotation, as floats, using the *inverse_rotation* function. Once we had these coordinates, we could predict the pixel intensity value at this position by applying the bilinear interpolation algorithm on the nearest pixels of the initial picture.

Figure 9: Rotation using inverse rotation and bilinear interpolation applied to *clean finger*.

To ensure our algorithm is correct, we implemented the method *inverse rotation*. The function we used in this case is the inverse function of rotation g^{-1}. When applying this method to the rotated image, we would get the initial image.



(a) rotation (b) inverse rotation

Figure 10: Rotation and inverse rotation of a $\pi/4$ angle applied to *clean finger* with the first method.

References