Implementing Iris Detection Algorithms in MATLAB

Xuexin Wei
Columbia University
Electrical Engineering
Email: xw2463@columbia.edu

Erik Su
Columbia University
Electrical Engineering
Email: es3573@columbia.edu

Bernard Nguyen
Columbia University
Electrical Engineering
Email: bqn2000@columbia.edu

Abstract

Biometrics is a growing field of interest as large amounts of data are being associated to an individual. As such, the means for accurately distinguishing between persons is increasingly needed for security. Similar to fingerprints, the human iris band contains a rich texture of information, but is even more distinct in comparison. This higher level of distinction allows iris detection methods to be less prone to fraud and exhibit very low rates of misidentification. In this report we explore the naive implementation of iris detection, the challenges associated, and a number of optimizations necessary to achieve suitable accuracy.

I. Introduction

In this project we aimed to achieve a naive implementation of iris detection. The process can be broadly split into segmentation of the iris, normalization, feature encoding, and finally template matching [1] [2]. Segmentation refers to the isolation of desired pixels of our input image and is implemented using filtering preprocessing steps and the Circle Hough Transform. After performing segmentation, the pixels are arranged in a "donut" fashion, suitable to be normalized from polar coordinates into a rectangular form for ease of feature extraction. Through 2D Gabor filtering, we end with binary templates that are used to compare across other iris templates. Once suitable templates are created, matching algorithms are used to quantify the results into proper detection thresholds. For this project we aimed to implement each of these steps and optimize parameters such that the matching algorithms resulted in high accuracy.

II. Technical Approach

A. Iris Segmentation

The first step in iris detection is segmentation of the iris band from the pupil and the sclera. To achieve this, the image is typically grayscaled and smoothed with a Gaussian filter as preprocessing steps for an edge detection algorithm. The circle Hough transform is then applied to the edge map in order to estimate the centers and radii of both the pupil and the iris boundaries. Once these values have been found, the pixels surrounding the iris band is then suppressed to be normalized for the feature extraction stage.

In achieving the preprocessing steps of the image, we used MATLAB's rgb2gray() function as well as imgaussfilt() function with a σ value of 6.5. This relatively large value of sigma would smooth our original image a significant amount which is necessary to accurately determine "true" edges once fed into the edge detection algorithm. "True" edges as defined by us would be edges easily discernible to the human eye (circles of the iris and pupil) in the attempt to give these boundaries more weight in the resulting edge map. Since edge map algorithms base their output edges off certain thresholds, by smoothing the image enough, only the strongest edges of the pupil and iris boundaries should be dominant in order for the circle Hough transform to accurately estimate centers and radii. The result of applying these preprocessing functions is shown in Figure 1.

Fig. 1: Preprocessing steps.

In order to produce an edge map, we used the Canny Edge Detection method which can be broken down into several steps, some of which were handled in the preprocessing portion earlier. Following the smoothing filter, we found the intensity gradients of the smoothed image, increased the contrast, applied non-maximum suppression, and finally implemented a double-threshold hysteresis to obtain a final edge map. The gradient step is a way to view the image in a sort of "texture" format which is further augmented by increasing the contrast of the gradient map. Non-maximum suppression is used as a way to thin the edges found by the contrasted map since the left image of Figure 3 is quite blurry. The process by which we implemented this naive version of non-maximum suppression can be viewed as iterating through each pixel, determining the local maxima in a select region surrounding the pixel, and suppressing the non-maximum pixels. From here the resulting image still contains "noise" in the form of possible edges that needs to be subjected to a double-threshold hysteresis in order to determine true edges. In Figures 2 and 3, we used a test

image implemented in Python in order to see how well our naive implementation has done.

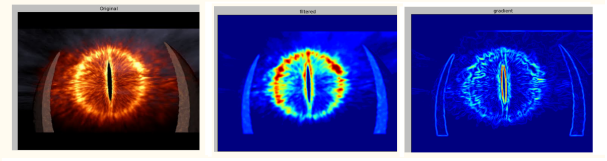

Fig. 2: Testing Canny Edge Detection

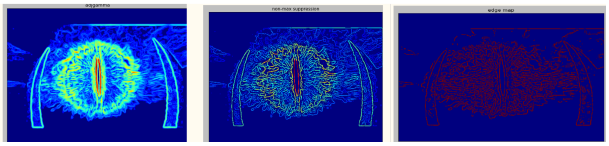

Fig. 3: Testing Canny Edge Detection (cont)

When the edge map is obtained, the circle Hough transform is used to find circles of the correct size. The circle Hough transform is described as a voting algorithm where the edges found in the edge map are used as centers of varying radii in order to accumulate a total for each region. To find the circle, the region with the most accumulation would be considered the center of the true circle with radii estimations based off the circles created through each candidate pixel of the edge map. This is better illustrated in Figure 4, taken from MathWorks' documentation of imfindcircles()[5]. The left image shows how each candidate pixel of the edge creates a circle of a given radius in order to add values to the accumulator bins indicated by the points at the edge of the dashed circle. The right image shows multiple candidate pixels doing the same, effectively voting for the region indicated by the gray circle in the middle. This gray circle would be the most likely candidate for the center of the true circle of that radius.

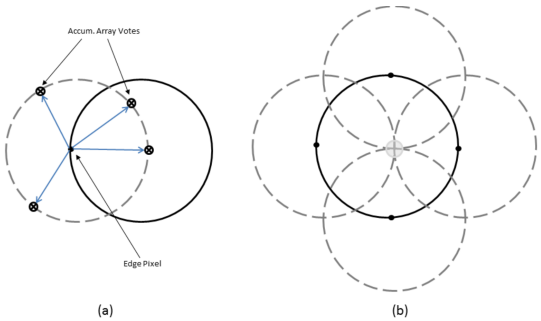

Fig. 4: Hough Circle [5]

After implementing our naive Canny edge detector and circle Hough transform, we realized that the runtime was excessively long (upward to several minutes) so we turned to MATLAB's imfindcircles() function which is part of the Image Processing Toolbox to optimize the code instead. This function uses graythresh() to determine a global threshold in its own algorithm to detect edges before applying the circle Hough transform. imfindcircle() takes parameters of a range of acceptable radii and the smoothed image as inputs while outputting its estimated centers, radii, and associated metric of magnitudes of peak accumulated arrays. Essentially, it outputs the most likely circles with centers and radii in descending order. Once we apply this function for both the pupil and the iris boundaries, we obtained centers and radii (for which we kept the most significant circles) and displayed them in Figure 5 using MATLAB's viscircles().

Fig. 5: Determining iris band boundaries

Once we had found the circles visually through viscircles(), we verify our result by changing the pixels of the boundaries into a constant value, 255 in our case. To implement this we simply had to convert from polar coordinates of a circle into Cartesian points using the equations shown in equations 1, where x_c and y_c are the coordinates of the center of the circle. Since the following step of suppression technically marks the boundaries automatically, this step is simply used as verification. The result of the verification is shown in Figure 6.

$$\begin{aligned} x &= rcos\theta + x_c \\ y &= rsin\theta + y_c \\ r^2 &= x^2 + y^2 \end{aligned} \tag{1}$$

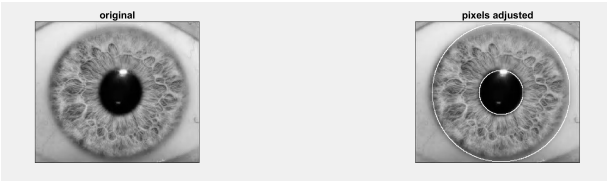

Fig. 6: Verifying found boundaries

By iterating through each pixel and comparing it with the appropriate inequalities given by the Cartesian equation in equations 1 for circles, we can quickly suppress all the pixels outside the bounds of the iris spectrum. After suppression, this image is ready to be normalized for feature extraction. The resulting image is shown in Figure 7.

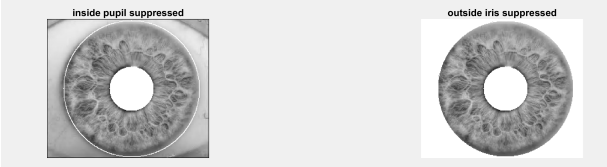

Fig. 7: Suppression of pixels

B. Feature Extraction and Encoding

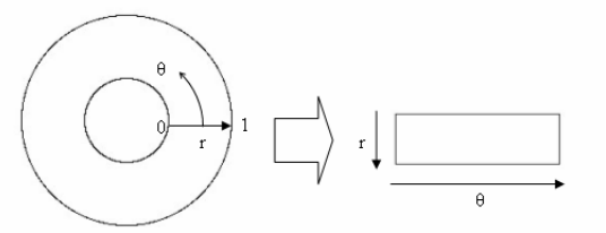

Fig. 8: The Daugman rubber sheet model.

Fig. 9: The normalized iris.

Once the iris was segmented, we needed to simplify our model to exclude the suppressed background. To do this, we implemented Daugman's rubber sheet model (Figure 8), which unravels the annulus into a rectangle in polar space [1] [2]. We cropped the image to center the iris/pupil, then implemented a function called ImToPolar [3] from the MATLAB file exchange. The resulting image can be seen in Figure 9.

In order to extract defining features in the iris, we used a 2-dimensional Gabor filter, a Gaussian filter modulated by a complex sinusoid. The Gabor filter is characterized by

$$G_{\lambda,\theta,\phi,\sigma,\gamma}(x,y) = e^{-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}} e^{j(2\pi\frac{x'}{\lambda}+\phi)}, \tag{2}$$

where

$$\begin{aligned} x' &= xcos\theta + ysin\theta \\ y' &= -xsin\theta + ycos\theta. \end{aligned} \tag{3}$$

The 2D Gabor filter takes five parameters, where λ is the sinusoid component's wavelength, θ is the angular orientation of the filter, ϕ is the phase offset, σ is the standard deviation of the Gaussian component, and γ is the spatial aspect ratio of the filter.

Gabor filters are popular in computer vision for analyzing textures [1] [2]. In our case, the Gabor filter is used to extract the defining features of the iris in the form of a real and imaginary component of each pixel. We stored the output of the Gabor filter into 2 bits, representing $Re\{G(x,y)\} >= 0$ and $Im\{G(x,y)\} >= 0$ respectively. The final output is a unique "barcode" (Figure 10), characterizing the iris sample. The entropy of each barcode ensures that false positives are unlikely to occur by random chance.

Fig. 10: The unique barcode characterizing our iris sample.

C. Matching Samples

The first matching algorithm we implemented was the Hamming distance. The Hamming distance is defined by the total number of substitutions that need to be made for the two samples to match. For example, the Hamming distance between the words "cat" and "bat" is 1. The only substitution that needs to be made is to change 'c' to 'b'. The Hamming distance between two iris samples is defined by

$$\sum_{i=0}^{N}\sum_{j=0}^{M} B_{1,ij} \oplus B_{2,ij} \tag{4}$$

where N is the number of rows, M the number of columns, and B is a barcode representing an iris sample.

Another way to compare iris samples is by correlation [1][2]. We simply used the corr2() function in the MATLAB library.

III. Experiments

Our goals for the experiment were to determine and optimize the accuracy of our detection algorithm. To do this we examined each main step of the iris detection algorithm with varying parameters. For segmentation, the main parameter to adjust was the sigma of the preprocessing Gaussian smoothing filter. Due to the limitations of imfindcircles(), there are many cases for which the function will not find any circles. Through the process of trial and error, we found that a σ value of 6.5 allowed all of our test images to both find circles and allow for proper segmentation. Our results with σ = 6.5 are shown in Figure 11.

Once the Hamming distance was implemented we could finally test our overall implementation of the biometric iris scan. The matching algorithm performed quite well, with mismatched images returning ∼70% match and matching images returning ∼75%, but there was still room for improvement. Several parameters throughout the process were chosen fairly arbitrarily, and required optimization. These factors included the Gabor filter σ and frequency, as well as the normalization's angular and radial resolutions.

Figure 12 shows our experimentation of these parameters. In order to quantify the success of our iris scan, we said that a matching iris should return 100%. The expected value for a different iris should be 50%, where roughly half the bits will match based on random chance. The error of our measurement is therefore

$$Err_{avg} = \frac{\sum_D |MP_i - 50| + \sum_S (100 - MP_i)}{N}, \tag{5}$$

where D represents the results of different irises, S represents the results from the same iris, MP is the match percentage output, and N is the total number of comparisons. From our results, we found most parameter values were fairly flexible within a certain range.

The final values we chose were $\sigma = 0.15$, $f = 0.001$, $rad_{res} = 20$, and $ang_{res} = 75$. After optimizing the parameters, we were able to successfully define a match threshold of

Fig. 11: Segmentation with Gaussian $\sigma = 6.5$

70%, where high quality images of matching irises consistently reported >70% match, and different irises reported <65% match.

Using the correlation values, we were able to mark a threshold at a correlation of 0.4. Iris samples from the same eye consistently had a correlation of >0.4, while those from separate eyes were always <0.4 and often <0.3. Our final comparison output can be seen in Figure 13 which compares an image of a teammate's eye with itself, a different image of the same eye, and someone else's eye.

IV. Discussion

Since segmentation is an important first step in the iris detection algorithm, our goal was to implement robust code in order to accommodate for a variety of images. However, it became very apparent that both the relative pupil and iris sizes along with the image quality play a large role in detecting the boundaries properly. Since the MATLAB function imfindcircles() and basic implementations of the circle Hough transform require a range of acceptable radii, the relative sizes of both pupil and iris boundaries must be fairly consistent through the dataset. Even slight adjustments to this range affect the voting algorithm and may not choose the correct circle as a result. We attempted to mitigate this effect through smoothing with

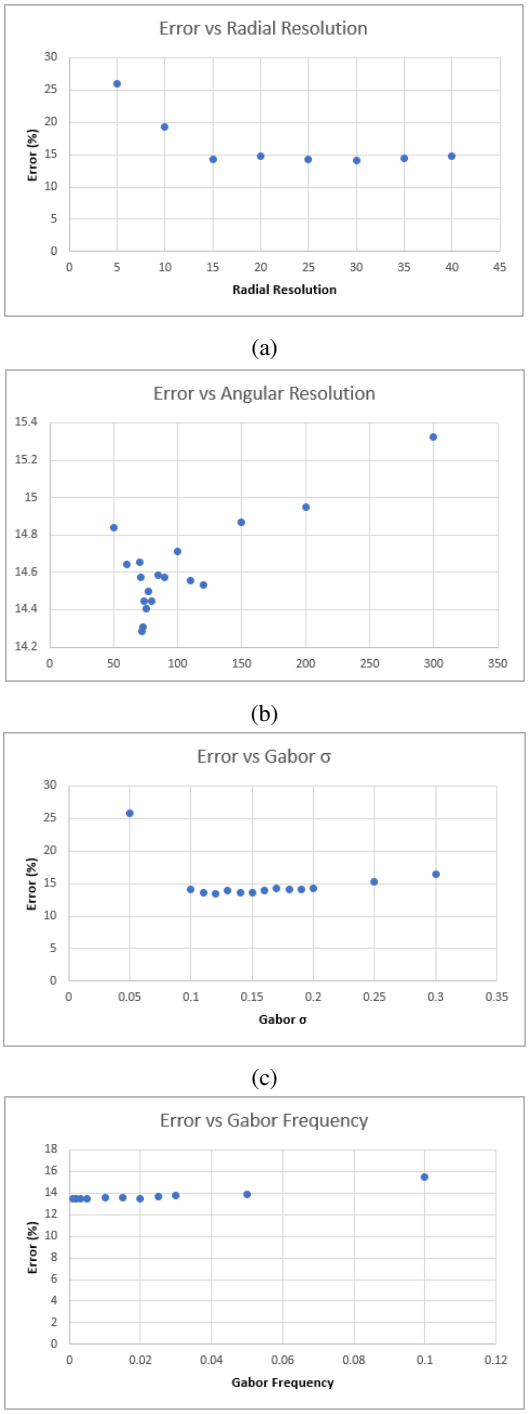

Fig. 12

a Gaussian filter of a relatively high σ value, but these would also affect images of varying qualities in different ways. Since the creation of an edge map (inherent to the imfindcircles()) also relies on some thresholds, the quality of the images must be relatively the same so the desired edges are marked dominantly, making it easier for the circle Hough transform

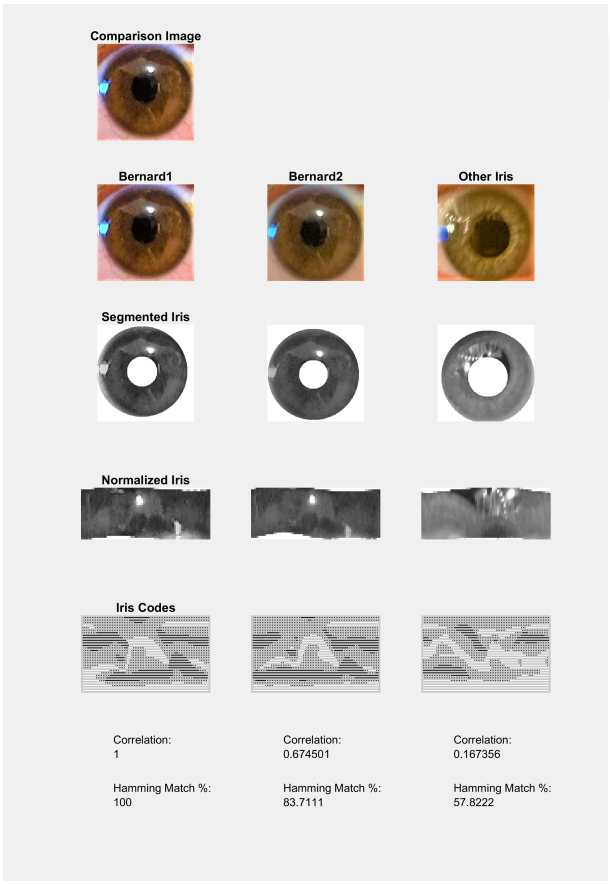

Fig. 13: Comparison of different iris templates

to select the correct boundaries.

Most implementations of this basic iris detection algorithm assume high levels of consistency among the images in order for the segmentation to work correctly [1] [2]. Since we did not have access to databases nor did we have specialized hardware to create our own hardware, our code is limited to a range of select images for which these assumptions are met.

When adjusting the normalization and encoding parameters, we noticed a couple things. While the angular resolution didn't appear to have a strong correlation for lower values, there was a clear pattern of increased error with increased resolution. This is because the iris is scaled down prior to normalization, and there simply aren't 360 degrees of resolution for the smaller image.

It is apparent from Figure 12(a) that radial resolution has a significant effect on the minimizing of error. As the radial resolution increases, error decreases, but starts to taper off where the radial resolution exceeds the actual image resolution.

Though Figure 12(d) doesn't show it well, there was a clear relationship between increased frequency, and increased error, especially in the lower values. The standard deviation of our Gabor filter, on the other hand, didn't seem to have a linear relationship. Rather, there was a certain range of frequencies that minimized error, which increased outside of that range.

We were able to successfully differentiate between different irises with no problems (no false positives). We did, however, have a few false negatives between our sample images. Iris scanners are often implemented with IR cameras, and we found that the images taken with our phone cameras were not sufficient. There was often glare or reflections from external light sources, which covered or blurred distinguishing features in the iris. Lighting conditions played a big role in image quality, and therefore successful matching.

Overall, our implementation of a biometric iris scan functions fairly well. Areas of improvement include higher quality imagery, more robust code that doesn't require specially cropped photos, and better calibration of our parameters. Higher quality imagery can be achieved using a specialized infrared camera with a user interface that guides the user to position his/her eye. The infrared camera captures all of the defining features of the iris, but doesn't take in the glare or reflections from external light sources. We would have also liked to have access to a large database with iris samples, but in limited time, we were unable to gain access to any such resource. A larger database of high quality images would have allowed for us to better calibrate our scanner and define better thresholds for a match. Additionally, access to more data would have been helpful in defining the rate of success of our scanner.

References

[1] L. Masek, "Recognition of Human Iris Patterns for Biometric Identification", B.S. Thesis, University of Western Australia, Crawley, WA, Australia, 2003. Available: *http://www.peterkovesi.com/studentprojects/libor/LiborMasekThesis.pdf*. [Accessed: Dec 9, 2017].

[2] R. Youmaran, Algorithms to Process and Measure Biometric Information Content in Low Quality Face and Iris Images, Ph. D. Thesis, University of Ottawa, Ottawa, Canada, 2011. Available: *http://www.sce.carleton.ca/faculty/adler/publications/2011/ryoumaran-2011-PhD-thesis.pdf*. [Accessed: Dec 9, 2017]

[3] P. Manandhar, "Polar To/From Rectangular Transform of Images - File Exchange - MATLAB Central", *www.mathworks.com*, Dec. 17, 2007. [Online]. Available: *https://www.mathworks.com/matlabcentral/fileexchange/17933-polar-to-from-rectangular-transform-of-images?focused=5095463*. [Accessed: Dec. 13, 2017].

[4] S.S. Dias, "Improved 2D Gabor Filter - File Exchange - MATLAB Central", *www.mathworks.com*, Jun. 2, 2008. [Online]. Available: *https://www.mathworks.com/matlabcentral/fileexchange/13776-improved-2d-gabor-filter*. [Accessed: Dec. 13, 2017].

[5] Anonymous, "Find circles using circular Hough transform - MATLAB imfindcircles", *www.mathworks.com*, Sept. 17, 2012. [Online]. Available: *https://www.mathworks.com/help/images/ref/imfindcircles.html*. [Accessed: Dec. 13, 2017].