Improved Minutiae Search in Latent Fingerprint

Dennis Kovarik
Department of Computer Science and Engineering
South Dakota School of Mines and Technology
501 E. St. Joseph Street, Rapid City, USA, 57701
dennis.kovarik@mines.sdsmt.edu

Mengyu Qiao
Department of Computer Science and Engineering
South Dakota School of Mines and Technology
501 E. St. Joseph Street, Rapid City, USA, 57701
mengyu.qiao@sdsmt.edu

Abstract

Latent fingerprint presents an invisible pattern to naked eyes, and thus needs special post-capture processing for identification. A crucial step of fingerprint identification is feature extraction. Most existing systems use level 2 features, which are regarded as minutiae. In this paper, we propose a novel method for image enhancement and minutiae extraction for the latent fingerprint images collected by using NIR-NIR upconversion nanoparticles. In the proposed method, a latent fingerprint image is segmented, filtered, adjusted to achieve a standard format. Then, minutiae features are extracted with their types, locations, and orientations. The proposed method uses a moving window with pre-defined masks to identify ridge endings and bifurcations. A verification step is then introduced after feature extraction to remove unreliable features by using quality measures. According to the experimental results, the proposed method was able to extract high-quality minutiae features and achieve accurate matching.

I. Introduction

Fingerprints, as a unique and irreproducible pattern, are one of the most popular methods for person identification and authentication for applications of law enforcement, business, and personal use. Fingerprint identification is the process of identifying individuals using their fingerprints [1]. Because of their uniqueness and permanence, fingerprints have been used to identify people since the 19th century [2]. A latent fingerprint is defined as "a fingerprint left on a surface by deposits of oils and/or perspiration from the finger" [3]. By collecting latent fingerprints found at crime scenes, law enforcement can use fingerprint identification in order to help link suspects to crimes [4]. There have been significant efforts in the past to automate this process, but despite the many advances made, it is still an area of continuing research.

A critical step in automatic fingerprint recognition systems is feature extraction. In this step, computational methods are used to identify and extract characteristic, quantifiable, and comparable features from the fingerprint. These features can be placed into 3 categories, which are referred to as level 1, level 2, and level 3 features. Level 1 features are the general patterns exhibited by the ridge orientations. Examples of these patterns include whorls, loops, and arches. Level 2 features refer to certain points found on the fingerprint, which are called minutiae. These minutiae usually identify the ridge endings and ridge bifurcations (where the ridge splits into two) that are found on the fingerprint. Finally, level 3 features include features such as sweat pore configurations and the ridge contours [5].

The features extracted during the feature extraction step are used in the matching step. During this step, the extracted features are used in a matching algorithm to determine how similar the latent fingerprint is to another fingerprint stored in some databases [6]. Most automatic fingerprint recognition systems extract and use level 2 features in order to match the fingerprints [7]. Although the poor quality of latent fingerprints can negatively affect the number and quality of the extracted minutiae, level 2 features have been shown to be a reliable set of features for use in the matching step for automatic fingerprint recognition systems.

The purpose of this study is to develop a method to automatically extract minutiae from images of latent fingerprints collected by using upconverting nanoparticles exhibiting near infrared NIR-to-NIR upconversion luminescence. We propose a minutiae extraction algorithm that is much like Crossing Number minutiae extraction method that works on thinned binarized images. After the fingerprint image is binarized and thinned, our algorithm uses a 3-by 3 mask containing predefined minutiae patterns to extract the minutiae. This is followed by a postprocessing step where the minutiae located along the edge of the fingerprint are removed. This study is a part of a larger project which is developing a portable field system to collect sensitive, interference-free latent fingerprints using these upconverting nanoparticles. The latent fingerprints collected using these nanoparticles results in images that look much like that presented in Figure 1. This study aims to develop part of the software that will perform the automatic fingerprint recognition for this device.

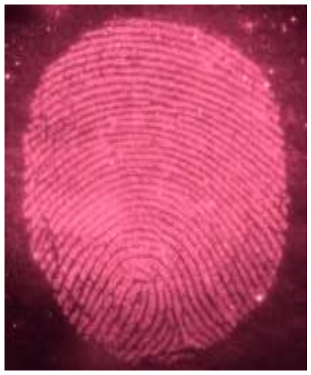
Figure 1: Original Latent Fingerprint

The rest of this paper is organized as follows. Section II briefly reviews related work on different minutiae extraction methods. Section III describes our approach for minutia extraction, which includes image preprocessing, feature extraction and verification. Section IV presents experiments with discussions followed by conclusions in Section V.

II. Related Work

Minutiae extraction is widely investigated topic where several methods have been proposed. These methods can be classified by whether they work on binarized images, or directly on gray-Level images. Furthermore, the minutiae extraction algorithms that work on binarized images can be further classified based on if they work on thinned or unthinned binarized fingerprint images.

2.1. Algorithms Based on Gray-Level images

This set of algorithms extract minutiae directly from the grayscale fingerprint images. There are several advantages for doing this, one of which is to avoid loss of information during the binarization step.

2.1.1. Minutiae Extraction by Following Ridge Flow Lines

This approach attempts to extraction the minutiae directly from the grayscale fingerprint image without the use of binarization. Using the fingerprint directional image, this technique follows the fingerprint ridgelines until the ridge either terminates or it intersects another ridge, at which point, a minutiae point is detected [8].

2.1.2. Fuzzy Techniques for Minutiae Extraction

In grayscale fingerprint images, there are two levels of pixel values, which can be classified as either DARK or LIGHT (corresponding to ridges or valleys in the fingerprint. These levels can be modeled using fuzzy logic, and then fuzzy rules can be applied in order to extract the minutiae [5].

2.2. Algorithms Based on Binarized images

In these algorithms, the fingerprint image is first enhanced and binarized before the minutiae are extracted. From the original image, a new image is created where each pixel can only take on one of two values. For example, a pixel value of 1 can indicated that a ridge is present at that pixel while a value of 0 can indicated the presence of a valley (or vice versa). These algorithms can further be divided into two categories, where one group works on unthinned binarized images while the other works on thinned binarized images.

2.2.1. Algorithms Based on Unthinned Binarized images

This group of algorithms work on binarized image where the ridges are unthinned, which means that the ridges in the fingerprint are not thinned down to be a single pixel wide.

2.2.1.1. Chaincode Processing

The binarized image is scanned from top to bottom and left to right looking for the ridge boundaries (where the pixel values transition from light to dark). After this is found, then the algorithm traces the boundary of the ridge in a counterclockwise direction. Ridge endings are detected when a significant left turn is made, while bifurcations (forks) are detected by a significant right turn [5].

2.2.1.2. Run Representation

This algorithm is based on horizontal and vertical run-length encoding from binary fingerprint images. The images are represented as a cascade of runs after run-length encoding. Then the adjacencies of the runs are checked, and characteristic runs are detected. This method is efficient in that it can reduce the memory requirement while speeding up image processing time [5].

2.2.1.3. Ridge Flow and Local Pixel Analysis

In this method, the average pixel value as calculated within a 3 by 3 window for each pixel. A pixel is considered to be a ridge ending if the calculated average is less than 0.25, while a pixel having an average greater than 0.75 is considered to be a bifurcation [5].

2.2.2. Algorithms that work on Thinned Binarized images

The second group works on binarized fingerprint images where the ridges within the fingerprint were first thinned until they were just a single pixel wide.

2.2.2.1. Crossing Number

This method is widely used on thinned binarized images because of its efficiency and simplicity. For each pixel in the image, the crossing number for the neighborhood is computed in a 3 by 3 window. The crossing number is computed by counting the number of adjacent pixels whose pixel value represents that it is part of a ridge. The crossing number is then used to determine that point is a minutia. For example, a crossing number of 1 indicates the presences of a ridge ending while a crossing number of 3 indicates a bifurcation [5].

2.2.2.2. Morphology based

This method is based on mathematical morphological operations. The image is first preprocessed by morphological operators in order to remove false minutiae such as spurs and bridges. Following this, a hit or miss transform is used to extract the true minutiae [5].

III. Proposed Approach

3.1. Image Enhancement

The images of latent fingerprints were filtered and enhance using the algorithm in Figure 2. Each image was converted to an 8-bit grayscale image and resized to a standard size of 533 by 400. Each image was converted to its complement. This way, each image is converted to a form where the ridges are black, and the background is white.

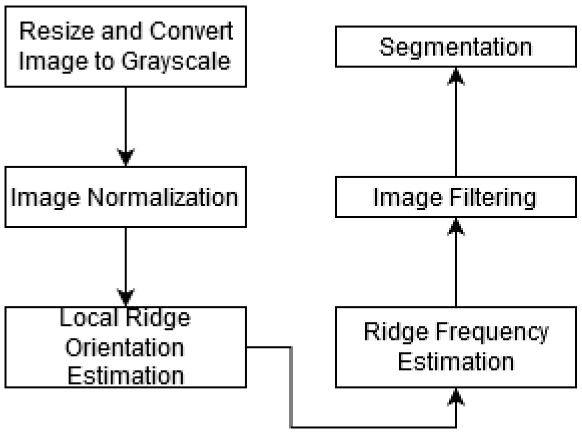
Resize and Convert Image to Grayscale
Image Normalization
Local Ridge Orientation Estimation
Ridge Frequency Estimation
Image Filtering
Segmentation

Figure 2: Image Enhancement Steps

Then a segment mask representing the location of the fingerprint in the image was found. This was accomplished by iteratively computing the standard deviations for the pixel values in the images with increasing block sizes (blksze). The standard deviations for each block would be stored in a separate image. More specifically, the image would be broken up into blksze x blksze blocks, and the standard deviation for each block would be computed with respect to the pixel values, which would be stored in the second image. This would be done iteratively with increasing values for blksze starting at 2 and increasing by 1 up till blksze equals 20. During each iteration after the image was block processed for the standard deviations and stored in a separate image called stddevim, an averaging filter of size 300 was applied to this temporary image, followed by global thresholding using Otsu's method. The resulting images for each iteration were added together and stored in another temporary image called ‘avg’. Then to remove holes in the segmentation mask, the image ‘avg’ had an averaging filter iteratively applied to it with increasing block sizes ranging from 30 to 100 pixels (increasing by 10 each iteration). All iterations were averaged together in order to produce the final mask representing the location of the fingerprint on the image. It is important to note that the fingerprint is not segmented until the end of the image filtering and enhancing step. This is because segmenting the image before the ridge orientations are found sometimes results in false ridges appearing in the final image. Instead, the mask produced from this step is saved for performing the image segmentation later.

In preparation for estimating the ridge orientation and frequencies, the image would be normalized to have zero mean and unit standard deviation. This is accomplished by first finding the mean and standard deviations of all the pixel values in the image. Each pixel would then be subtracted by the image mean pixel value before each pixel is divided by the standard deviation.

Then estimates of the local orientation of the ridges in the fingerprint were computed and stored in a temporary image called ‘orientim’. Note that the following operations were applied to each pixel within the normalized image. The image gradients were found by computing the 1st and 2nd derivatives of the image using the 7-tap coefficients. Then the local ridge orientation was found by finding the principal axis of variation in the image gradients. This produced the covariance data for the image gradients, which was then smoothed using gaussian filters.

The analytic solution of principal direction was found through using the following equations:

$$denom = \sqrt{G_{xy}^2 + \left(G_{xx} - G_{yy}\right)^2 + eps} \tag{1}$$

$$sin2theta = \frac{G_{xy}}{denom} \tag{2}$$

$$cos2theta = \frac{G_{xy}}{denom} \tag{3}$$

A gaussian filter was applied to the images produced for sin2theta and cos2theta with a block size of 30 and a sigma of 5. Then finally, the image representing the orientation of the ridges was computed by applying the following equation:

$$orientim = \frac{pi}{2} + \frac{\operatorname{atan2}(sin2theta, cos2theta)}{2} \tag{4}$$

Following this, the median ridge frequency was computed using the ridge orientation image produced in the previous step. The image is broken up into small blocks, and the ridge frequency within each small block is estimated. Each block is rotated and cropped so that the ridges are vertical. The columns are summed down to get a projection of the grey values down the ridges. Then the peaks are found in projected gray values by performing a greyscale dilation and then finding where the dilation equals the original values. Then the spatial frequency is determined by dividing the distance between the 1st and last peaks by the number of peaks. If no peaks are detected, or the wavelength is outside the allowed bounds, the frequency image is set to 0. Then the median ridge frequency is found among all the ridge frequencies computed.

The fingerprint image is then filtered via oriented filters. An array of filters is generated corresponding to the median ridge frequency and orientations present in the fingerprint image. The image is then filtered by using the generated filters corresponding to the median frequency and the local ridge orientations.

The filtered image is converted to a binary image where ridges are represented by a value of 1 and the valleys and background are represented by a value of 0 [9]. Finally, the image is segmented using the segment mask produced from the fingerprint segmentation step described earlier.

3.2. Minutiae Extraction

Using the binary image produced from the image filtering and enhancement step, the ridges were thinned down until they were only 1 pixel wide, producing an image which we will call thinnedRidges. The minutiae were then extracted from the thinnedRidges image using Algorithm 1. The thinnedRidges image was scanned from top to bottom and left to right with a 3-by-3-pixel mask. This mask had predefined patterns which represented the valid ridge endings and bifurcations that could be observed within the 3 by 3 window. When one of these predefined minutiae patterns were found, its (x, y) location and minutia number (ID identifying the structure of the minutiae) was stored in a separate image called ‘locations.

```
Procedure extractMinutiae(thinnedRidges)
    height = height of thinnedRidges image
    width = width of thinnedRidges image

    // thinnedRidges in the thinned binary enhance image
    // Initialize images to store the location and the angle
    // of orientation for the extracted minutiae/
    locations = zeros(height,width)
    Ɵ = zeros(height,width)

    // Scan image
    for i = 4 to height-4
        for j = 4 to width-4
            if thinIm(i,j) > 0
                window = cropImage(thinIm, j, i, 3, 3)
                locations (i,j) = isMinutiae(window)
                Ɵ (i,j) =  findƟ(i, j, locations(i,j), thinIm)

    return locations, Ɵ
```

```
Procedure isMinutiae(window)
    // window is a 3x3 binary image
    // minutiaeSet is an array of 3x3 mask of predefined
    // minutiae patterns
    minutiaeSet = createMinutiaeSet()

    // Match window to set of known minutia
    for i = 1 to number of masks in minutiaeSet
        if minutiaeSet(i) == window
            // return the ID of the minutiae
            return i

    // A minutiaeID of 0 indicates invalid minutiae
    return 0
```

Algorithm 1: Minutiae Extraction Pseudocode

In addition to the minutiae’s (x,y) coordinates in the fingerprint image, minutiae matching software such as the BOZORTH3 tool require the angle of orientation for each minutiae. The angle of orientation for the extracted minutiae followed the ANSI/NIST standard, and it was calculated using Algorithm 2. In order to find the orientation angle for a ridgeline, 2 reference points will be used. The coordinate of the minutiae will be the first reference point (x_1, y_1). Then the pixel located 10 pixels away along the ridgeline will be used as the second reference point (x_2, y_2). After these two points are found, the ridge orientation (in degrees) was found by using the following equation (5). Note the that values of x and y below are in reference to the origin (0, 0) located in the top left of the image.

$$\theta_{ridgeline} = \begin{cases} \tan^{-1}\frac{|y_1-y_2|}{|x_1-x_2|} * \frac{180}{\pi}, & y_2 \le y_1 \; and \; x_2 > \; x_1 \\ 180 - \left(\tan^{-1}\frac{|y_1-y_2|}{|x_1-x_2|} * \frac{180}{\pi}\right), & y_2 \le y_1 \; and \; x_2 < \; x_1 \\ 180 + \left(\tan^{-1}\frac{|y_1-y_2|}{|x_1-x_2|} * \frac{180}{\pi}\right), & y_2 > y_1 \; and \; x_2 \le \; x_1 \\ 360 - \left(\tan^{-1}\frac{|y_1-y_2|}{|x_1-x_2|} * \frac{180}{\pi}\right), & y_2 > y_1 \; and \; x_2 \ge \; x_1 \end{cases} \quad (5)$$

If the extracted minutia was a ridge ending, then there is only one ridgeline orientation present, and the ridge ending angle of orientation in the ANSI/NIST standard could be found by the following equation (6).

$$\theta_{ridge\ ending} = \left(\theta_{ridgeline} + 180\right) \% \; 360 \quad (6)$$

Finding the minutiae orientation for bifurcations was a little more complicated. A bifurcation is defined by 3 ridgelines converging at a single point, so first reference point will be the convergence point. Then, the orientation of all three ridges will be found using equation (5), and the same algorithm used for calculating the angle of orientation for a ridgeline (Algorithm 2). The angles of orientations for these three ridgelines will be represented in the following 3 equations as Θ_1, Θ_2, and Θ_3. For each of these ridgelines, the minimum difference in the angle of orientations between the neighboring ridgelines will need to be calculated. The difference between two neighboring ridgelines (Θ_1 and Θ_2) can be found using equation (7).

$$\theta \; Diff_{1,2} = \begin{cases} |\theta_1 - \theta_2| \;\; if \; |\theta_1 - \theta_2| \le 180 \\ 360 - \; |\theta_1 - \theta_2| \;\; if \; |\theta_1 - \theta_2| > 180 \end{cases} \quad (7)$$

Where Θ_1 is the angle of orientation for ridgeline 1, Θ_2 is the angle of orientation for ridgeline 2, and Θ $Diff_{1,2}$ is in reference to ridgeline 1 and represents the difference in the angle of orientations between Θ_1 and Θ_2. Following this, the minimum difference in the angles of orientations between a ridgeline's neighbors (in this case ridgeline 1's neighbors) can be found using equation (8).

$$min \; \theta \; Diff_1 = \min(\theta \; Diff_{1,2}, \; \theta \; Diff_{1,3}) \quad (8)$$

Where min Θ $Diff_1$ refers to the minimum difference in the angles of orientations between the neighboring ridgelines for Θ_1, Θ $Diff_{1,2}$ is the difference in the angles of orientation for Θ_1 and Θ_2, and Θ $Diff_{1,3}$ is the difference in the angles of orientation for Θ_1 and Θ_3. Finally, the angle of orientation for the bifurcation can be determined by finding the ridgeline with the highest absolute difference in the angles of orientation between the other 2 ridge neighboring ridgelines. This is represented in equation (9).

$$\theta\ Bifurcation = \max(min\ \theta\ Diff_1, min\ \theta\ Diff_2, min\ \theta\ Diff_3) \tag{9}$$

```
Procedure findΘ(i, j, minutiaeType, thinIm)
  // thinIm is the binary thinned image
  // thinIm(i,j) is the current row and column location
  if minutiaeType == ridge ending
    nextI = row, along the ridgeline, adjacent to thinIm(i,j)
    nextJ = column, along the ridgeline, adjacent to thinIm(i,j)
    ΘRidge = calcΘRidgeline(thinIm, i, j, nextI, nextJ, minutiaeType)
    return (ΘRidge + 180) mod 360
  elseif minutiaeType == Bifurcation
    // Find the orientations for the ridgelines adjacent to minutiae
    for p = 1 to 3
      nextI = row, along the ridgeline 1, adjacent to thinIm(i,j)
      nextJ = column, along the ridgeline 1, adjacent to thinIm(i,j)
      // dirs is an array of the ridgeline orientations
      dirs(p) = calcΘRidgeline(thinIm, i, j, nextI, nextJ, minutiaeType)

    // Determine the min diff between each ridge's neighbors
    // minDiffs is an array holding the minimum differences
    for p = 1 to 3
      angle1 (p) = abs(dirs(1) - dirs(2))
      if angle1 > 180
        angle1 = 360 - angle1
      angle2 = abs(dirs(1) - dirs(3))
      if angle2 > 180
        angle2 = 360 - angle2
      minDiffs(p) = min(angle1, angle2)

    // Find max dir diff
    max = -1;
    maxDiffIndex = 1
    for d = 1:3
      if minDiffs(d) > max
        max = minDiffs(d)
        maxDiffIndex = d
       // return the orientation of the ridgeline with the max difference
    return  dirs(maxDiffIndex)
  return -1            // indicate error occurred

Procedure  calcΘRidgeline(thinIm, i, j, p, q, minutiaeType)
  // thinIm(i,j) is the location of the minutiae
  // thinIm(p,q) is the direction of the reference pixel
  refI = row of reference pixel ten steps along the ridgeline
  refJ = column of reference pixel ten steps along the ridgeline

  // Find the current orientation angle in radians
  orient = atan(abs(refI - startI) / abs(refJ - startJ))

  if refI <= startI and refJ < startJ         // quadrant2
    orient = 3.141593 - orient
  elseif refI > startI and refJ < startJ      // quadrant3
    orient = 3.141593 + orient
  elseif refI > startI and refJ >= startJ     // quadrant4
    orient = 2 * 3.141593 - orient

  // convert orientation angle to degrees and return
  return orient * (180 / pi)
```

Algorithm 2: Minutiae Angle of Orientation Pseudocode

3.3. Post Processing to Eliminate False Minutiae

In addition to the algorithms described above, post processing steps were taken in an attempt the improve the quality of the extracted minutiae. This was done by eliminating the minutiae extracted along the edge of the segmented fingerprint (which were determined to unreliable).

The edge of the fingerprint could be determined by using the segmentation mask generated in the Image Enhancement and Binarization step. The edge of the segmentation mask represented the edge of the fingerprint. Any minutiae that was extracted within 5 pixels from the edge of the fingerprint was simply removed.

IV. Results and Discussion

4.1. Testing Dataset

The dataset used for testing consisted 23 images of whole fingerprints. Each of these images were collected using nanoparticles exhibiting near infrared NIR-to-NIR upconversion luminescence. The fingerprints in the dataset where manually matched, and it was determined that there was a total of 51 correct fingerprint matchings.

4.2. Third-Party Software Used to Analyze and Compare Results

To compare the results of the minutiae extraction algorithm presented in this paper, it was compared to the performance of the minutiae extraction software called MINDTCT, which was developed by NIST [10]. NIST released and set their software as the benchmark for fingerprint recognition software performance. MINDTCT is a part of NIST's fingerprint recognition software that performs the minutiae extraction, so MINDTCT would serve as a good comparison for the performance of the minutiae extraction algorithm presented in this paper [11].

The BOZOTH3 tool requires the extracted minutiae to be stored in a separate file and given the file extension '.xyt'. Within these files, each line in the file would represent an extracted minutia, and each line would contain 3 to 4 space separated integers. The first two integers would represent the minutiae's x and y position within the fingerprint image (in accordance with the ANSI/NIST standard). The third space separated integer is the minutiae's angle of orientation. Finally, there is a fourth optional integer that specifies the quality of the extracted minutiae [11]. Although MINDTCT includes the fourth integer by default, the minutiae extraction algorithm presented in this paper does not include this measure when compiling the '.xyt' files for the extracted minutiae.

The BOZORTH3 tool takes as input two files with the file extensions '.xyt'. This tool will compute and output a score indicating how well the minutiae for the two files match, where a higher score indicates a better match between the two files [10]. NIST suggests that any score that is over 40 indicates a match between two fingerprints, so 40 will be the one of the thresholds used in this paper to indicate a match between two fingerprints [12]. In addition, since all of the tests ran using a threshold of 40 identified 100% of the true positives, another set of matching tests where run with a threshold of 80 in order to get a better understanding of performance for each of the minutiae extraction techniques being tested.

4.3. Matching Results

The minutiae from each of these fingerprints were extracted using three different minutiae extraction techniques and analyzed independently. The minutiae extraction techniques used includes the following: MINDTCT, the algorithm described in this paper (Algorithm 1) without removing the minutiae around the edge of the fingerprint, the same algorithm (Algorithm 1) with the edge minutiae removed. The average number of minutiae extracted by MINDTCT is 78. The average number of minutiae extracted using Algorithm 1 without removing the edge minutiae was 175, while the average number of minutiae extracted by Algorithm 1 with removing the edge minutiae was 108. An overview of the matching results using a matching threshold of 40 is presented in Table 1, while an overview of the matching results using a matching threshold of 80 is presented in Table 2.

Minutiae Extraction Technique	**BOZORTH3 Threshold of 40**		
	Total Correct Matches	***True Positives***	***False Positives***
MINDTCT	51	51	0
Algorithm 1 Without Removing Edge Minutiae	51	51	2
Algorithm 1 With Removing Edge Minutiae	51	51	0

Table 1: Overview of Matching Results using a threshold of 40 for the Bozorth3 Matcher

Minutiae Extraction Technique	**BOZORTH3 Threshold of 80**		
	Total Correct Matches	***True Positives***	***False Positives***
MINDTCT	51	35	0
Algorithm 1 Without Removing Edge Minutiae	51	43	0
Algorithm 1 With Removing Edge Minutiae	51	45	0

Table 2: Overview of Matching Results using a threshold of 80 for the Bozorth3 Matcher

For the first set of results obtained using a matching threshold of 40 for the BOZORTH3 tool (which is the recommended threshold by NIST), all minutiae extraction techniques identified 100% of the correct matches. The only difference in the results for these 3 techniques is with the false positives for Algorithm 1 without removing the edge minutiae. This technique falsely had 2 false positive matches, which results in 4% of all the matches identified by this technique resulting in a false match.

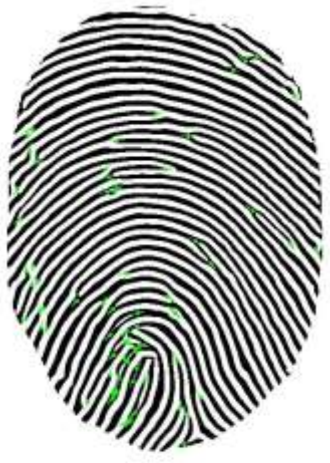

Figure 3: Extracted Minutiae from Algorithm 1 with Removing the Edge Minutiae

To obtain a better understanding of the performance for each of these three minutiae extraction techniques, a threshold of 80 for the BOZORTH3 tool was used to perform matching using the same extracted minutiae from earlier. These results are presented in Table 2. Unlike before, there are no false positive matchings identified by any of the 3 minutiae extraction techniques. The MINDTCT tool identified 69% of all correct matches, which amounts to it revealing 35 true positive matches of the 51 totally. Algorithm 1 without removing the minutiae around the edge of the fingerprint did significantly better by identifying 84% of the total correct matches. Using this technique, 43 true positives were identified out of the 51 correct matches. Finally, Algorithm 1 with removing the minutiae around the edges of the fingerprint caught 88% of the correct matchings, finding 45 true positives from the 51 total correct matchings.

V. Conclusion

5.1. Summary

The minutiae extraction algorithm presented in this paper has shown to be effective while also being simple to implement. By outperforming the MINDTCT tool, it was able to pass the baseline performance set by NIST on this dataset. Using the BOZORTH3 matcher with a threshold of 40, 100% of the correct matches in the test dataset where able to be identified, and when the minutiae around the edge of fingerprints are removed, it identified no false positive matches.

When the matching threshold for the BOZORTH3 tool was doubled to 80, the proposed minutiae extraction algorithm presented without edge minutiae removal was able to outperform the MINDTCT tool by 15%. Since many of the minutiae present along the edge of the fingerprint are unreliable (due to the creation of false ridge endings from segmentation), the performance of the presented minutiae extraction technique was improved by simply removing the minutiae present along the edge of the fingerprint.

5.2. Future Work

Since the dataset used to evaluate the performance of the presented minutiae extraction algorithm was small, further evaluation of its performance would need to be completed by testing it on a much larger dataset containing images of less quality. Work will need to be done on extracting level 1 and 3 features from the latent fingerprints. By mitigating unfavorable factors in the use of latent fingerprint, a comprehensive scheme based on the fusion of diverse features of Level 1-3 is expected to achieve high accuracy and reliability. In addition, the extracted features will need to be represented in a way that would facilitate the ability to match fingerprints against a large database of fingerprints. One possible avenue to accomplish this is to investigate using machine learning for matching fingerprints.

Acknowledgment

This work was supported by the National Institute of Justice grant 2017-IJ-CX-0026. Opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Department of Justice. I would like to thank Dr. Mengyu Qiao for advising me on this project. Special thanks to Dr. William Cross, and Dr. Jon Kellar for their continuous support, feedback, and guidance on this project. In addition, I would like to acknowledge Dr. Rodney Rice for his help in developing the framework and sharing his expertise in technical writing.

References

[1] Mayhew, S. (2012). What is Fingerprint Identification? Retrieved August 2, 2019, from https://www.biometricupdate.com/201205/what-is-fingerprint-identification
[2] Watson, S. (2018). How Fingerprinting Works. Retrieved June 19, 2019, from https://science.howstuffworks.com/fingerprinting3.htm
[3] US Legal, Inc. (n.d.). Latent Fingerprint Law and Legal Definition. Retrieved July 29, 2019, from https://definitions.uslegal.com/l/latent-fingerprint/
[4] Fingerprints: An Overview. (2016). Retrieved June 19, 2019, from https://nij.gov/topics/forensics/evidence/impression/Pages/fingerprints.aspx
[5] Bansal, R., Sehgal, P., & Bedi, P. (2011). Minutiae Extraction from Fingerprint Images - a Review (Rep. No. 1694-0814). Retrieved June 20, 2019, from IJCSI International Journal of Computer Science Issues website: https://arxiv.org/ftp/arxiv/papers/1201/1201.1422.pdf

[6] Ratha, N. K., & Bolle, R. (2011). Automatic fingerprint recognition systems. New York: Springer.

[7] Nguyen, D. and Jain, A. (2019). End-to-End Pore Extraction and Matching in Latent Fingerprints: Going Beyond Minutiae. GroundAi. [online] Available at: https://www.groundai.com/project/end-to-end-pore-extraction-and-matching-in-latent-fingerprints-going-beyond-minutiae/1#bib.bib2, Accessed 19 Jul. 2019.

[8] D. Maio and D. Maltoni, "Direct gray-scale minutiae detection in fingerprints", IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(1):27–40.

[9] Kovesi, P. (2005). Retrieved from https://www.peterkovesi.com/matlabfns/#fingerprints

[10] Mariano, L. (2015, March 4). Retrieved from https://github.com/lessandro/nbis

[11] Watson, C. I., Garris, M. D., Tabassi, E., Wilson, C. L., McCabe, R. M., Janet, S., & Ko, K. (n.d.). User's Guide to Nist Biometric Image Software (Nbis).

[12] Watson, C. I., Garris, M. D., Tabassi, E., Wilson, C. L., McCabe, R. M., Janet, S., & Ko, K. (n.d.). User's Guide to Export Controlled Distribution of NIST Biometric Image Software (NBIS-EC).