THE NBIS-EC SOFTWARE IS SUBJECT TO U.S. EXPORT CONTROL LAWS.

User's Guide to Export Controlled
Distribution of NIST Biometric Image Software (NBIS-EC)

Craig I. Watson (cwatson@nist.gov)

Michael D. Garris (mgarris@nist.gov)

Elham Tabassi (elham.tabassi@nist.gov)

Charles L. Wilson (cwilson@nist.gov)

R. Michael McCabe (mccabe@nist.gov)

Stanley Janet (sjanet@nist.gov)

Kenneth Ko (kenko@nist.gov)

National Institute of Standards and Technology
Bldg. 225, Rm. A216
100 Bureau Drive, Mail Stop 8940
Gaithersburg, MD 20899-8940

ACKNOWLEDGEMENTS

We would like to acknowledge the Federal Bureau of Investigation and the Department of Homeland Security who provided funding and resources in conjunction with NIST to support the development of this fingerprint image software.

NOTE TO READER

The NBIS-EC software is subject to U.S. export control laws.

This document provides guidance on how the NIST Biometric Image Software (NBIS) export controlled package are installed and executed. Its content and format is one of user's guide and reference manual. Some algorithmic overview is provided, but more details can be found in the cited references.

The Table of Contents provides the reader a map into the document, and the hyperlinks in the electronic version enable the reader to effectively navigate the document and locate desired information. These hyperlinks are unavailable when using a paper copy of the document. Any update to this export control software is only available on CD-ROM upon on request. For detail information on how to request a copy of the latest version of this export controlled NBIS source code, please follow the procedure in APPENDIX A.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

User's Guide to Export Controlled Distribution of NIST Biometric Image Software

(NBIS-EC)

C. I. Watson, M. D. Garris, E. Tabassi, C. L. Wilson, R. M. McCabe, S. Janet and K. Ko

ABSTRACT

The NBIS-EC software is subject to U.S. export control laws. This report documents the export controlled distribution of biometric image software developed by the National Institute of Standards and Technology (NIST) for the Federal Bureau of Investigation (FBI) and Department of Homeland Security (DHS). The export controlled NBIS software is organized into two major packages: 1. `NFSEG` is a fingerprint segmentation system useful for segmenting four-finger plain impressions, 2. `BOZORTH3` is a minutiae based fingerprint matching system. It is our understanding that this software falls within ECCN 3D980, which covers software associated with the development, production or use of certain equipment controlled in accordance with U.S concerns about crime control practices in specific countries. This documentation provided the overview of the export controlled distribution of NBIS software, software installation guide, some algorithmic overview of the NBIS export controlled software and Reference Manual describing each program in this distribution.

1. INTRODUCTION

The NIST Biometric Image Software (NIBS) is organized in two categories: non-export controlled and export controlled. The non-export controlled NBIS source code (PCASYS, MINDTCT, NFIQ, AN2K, and IMGTOOLS) is managed using the Perforce[1] source code management system on the NIST Image Group Open Source Sever (NIGOS). Also, the non-export controlled NBIS source code is available as a zip file which is updated daily at http://www.itl.nist.gov/lad/894.03/nigos/nigos.html. The export controlled NBIS source code (NFSEG and BOZORTH3) is only available on CD-ROM upon on request. It is our understanding that NFSEG and BOZORTH3 falls within ECCN 3D980, which covers software associated with the development, production or use of certain equipment controlled in accordance with U.S concerns about crime control practices in specific countries.

The NBIS non-export controlled software can be compiled and installed independently of the export controlled portions. However, the NIBS export controlled software is dependent on the successful compilation of NBIS non-export controlled libraries.

The export controlled CD-ROM distribution contains NFSEG and BOZORTH3 packages, related documentation and test examples/data all stored in a hierarchical collection of directories. A Bourne Shell (sh) script has been provided to copy the contents of the export controlled CD-ROM to a user specific directory where the non-export controlled packages are installed. Section 2.1 describes the installation in detail. Any new release distribution of the export controlled CD-ROM notification will be posted at http://fingerprint.nist.gov/NBIS.

NFSEG is a fingerprint segmentation system. It takes a four-finger plain impression fingerprint image (called a slap) and segments it into four separate fingerprint images. These single finger plain impression images can then be used for single finger matching versus either rolled images or other plain impression fingerprint images. NFSEG will also take a single finger rolled or plain impression image and isolate the fingerprint area of the image by removing the white space. Section 3.1 describes NFSEG in more detail.

BOZORTH3 is a fingerprint matching system. It uses the minutiae detected by MINDTCT to determine if two fingerprints are from the same person, same finger. It can analyze fingers two at a time or run in a batch mode comparing a single finger (probe) against a large database of fingerprints (gallery). Section 3.2 describes BOZORTH3 in more detail.

This software was produced by NIST, an agency of the U.S. government, and by statute is not subject to copyright in the United States. Recipients of this software assume all responsibilities associated with its operation, modification, and maintenance.

[1] Specific software products and equipment identified in this paper were used in order to adequately support the development of the technology described in this document. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the software or equipment identified is necessarily the best available for the purpose.

2. INSTALLATION GUIDE

This section describes the NBIS CD-ROM distribution organization and installation. This CD-ROM distribution contains the Export Control software, documentation, and test examples/data all stored in a hierarchical collection of directories. The top-level directory is `NBIS`. The `NBIS` directory contains `Main` and `Test` directories. `Main` contains NBIS packages and the software build utilities to compile applications. Documentation is also provided under `Main` in the `doc` and `man` directories. The `Test` directory contains testing scripts, input data files, and example results. The distributed source code has been written in ANSI "C," and compilation scripts for the Bourne shell are provided. The source code and compilation scripts have been designed and tested to work with the free, publicly available Linux operating system and GNU `gcc` compiler and `gmake` utility.[1][3] The software is also compiled and tested to work with Mac OS-X operating system. The software may also be compiled to run on computers running the family of Win32 operating systems by first installing the free, publicly available Cygwin library and associated tools.[2] The porting of the software to other operating systems, compilers, or compilation environments is the responsibility of the recipient.

2.1. Full Installation

The NBIS CD contains the entire NBIS stable release, both export controlled and non-export controlled software components. The version information for the NBIS software is provided in the `README_CD` file under directory `NBIS/Main` on the CD-ROM.

The NBIS software can be installed by first copying the contents of the CD-ROM to a write-able disk partition on your computer. The directory to which you copy is referred to as the *CD_install_directory*.

Once the CD-ROM contents are successfully copied to the CD_*install_directory*, the NBIS software can be built. For instructions on how to build the NBIS software, please consult the User's Guide to NIST Biometric Image Software.[10]

Successful compilation will produce two executable utilities in addition to those that are part of the non-export controlled packages. Table 1 lists these utilities broken out by package. To learn more about the utilities in this package, refer to the Reference Manual in APPENDIX B.

Table 1. NBIS export controlled utilities listed by package

NBIS Export Controlled Package Utilities					
BOZORTH3	**NFSEG**				
Bozorth3	nfseg				

2.2. Integrating the Export Controlled Software

The NBIS CD contains the entire NBIS stable release, both export controlled and non-export controlled software components. If you already have an existing non-export controlled development branch (i.e. synchronized against the Perforce repository), the procedures contained in this section can be used to integrate the export controlled software into your branch.

The version information for the NBIS software is provided in the `README_CD` file under directory `NBIS/Main` on the CD-ROM. Integrating the export controlled software with a version of the non-export controlled software other than the version that came with the CD-ROM distribution may cause the NBIS software to not compile, or improper behavior of the NBIS executables.

The NBIS software can be installed by first copying the contents of the CD-ROM to a write-able disk partition on your computer. The directory to which you copy is referred to as the *CD_install_directory*.

A Bourne Shell (sh) script has been provided (*CD_install_directory* `/NBIS/Main/integrate_export_control.sh`) to integrate the NBIS export controlled software to a directory where you have installed the non-export controlled software. The script may be invoked from a shell. Within the directory containing the script, type the following:

```
% sh integrate_export_control.sh <INSTALLATION DIR>
```

where the text `<INSTALLATION DIR>` is replaced by the specific directory path where you installed the NBIS non-export controlled software (your development branch). The directory to which you copy is referred to as the *installation_directory*.

Once the export controlled software is successfully copied to the *installation_directory*, you are required to rebuild the NBIS software. For instructions on how to build the NBIS software, please consult the User's Guide to NIST Biometric Image Software.[10]

Successful compilation will produce two executable utilities in addition to those that are part of the non-export controlled packages. Table 1 lists these utilities broken out by package. To learn more about the utilities in this package, refer to the Reference Manual in APPENDIX B.

A manual page is provided for each utility in the `man` directory under *final installation directory*. Please consult the User's Guide to NIST Biometric Image Software for the definition of the *final installation directory*.[10] To view a man, type:

```
% man -M <install_dir>/man <executable>
```

where the text `<install_dir>` is replaced by your specific *final_installation_directory* path and `<executable>` is replaced by the name of the utility of interest. These manual pages are also included at the end of this document, in the Reference Manual in APPENDIX B.

2.3. Data and Testing Directories

For information on the data and testing directories, please consult the User's Guide to NIST Biometric Image Software.[10]

3. EXPORT CONTROL PACKAGES

3.1. NFSEG – Four-Finger Plain Segmentation

Several datasets used in NIST testing are from tenprint fingerprints cards.

Figure 1 shows a blank tenprint card where boxes 1-10 contain rolled impressions of the fingerprints and the plain impression are in the four boxes along the bottom of the card. NIST developed NFSEG in order to evaluate the performance of matching plain versus rolled images.[5]-[8], [9]. Providing a reliable and automated way of segmenting the four-finger plain image impressions has many other benefits. For example, fingerprint card processing centers can conduct automated finger sequence checks between rolled finger on the card and the plain impressions. While it is easy to have a clerical error when placing individual rolled fingerprints onto a card, it is quite difficult to change the order of the fingers in the plain impression.

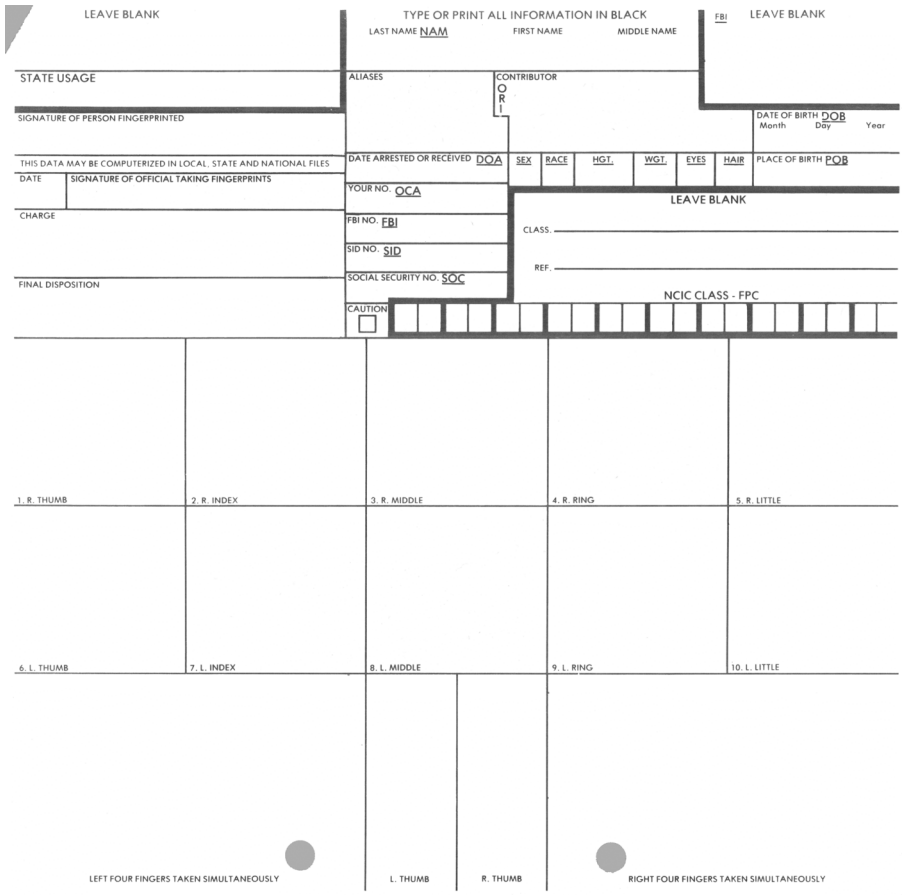

Figure 1. A blank fingerprint card.

`NFSEG` uses a down-sampled binary version of the four-finger plain image. A search is made, which includes rotation, for the best fit of four black ridges (fingers) and three white valleys (space between fingers). At each point of rotation in the search a normalized score is computed

and used to determine the best overall fit of four ridges and three valleys. After determining the location and orientation of the four fingers, the fingertips are isolated by a window sized just large enough to enclose the fingertip. Error flags are set if the spacing of the four fingers found are not evenly spaced within a certain tolerance and if the size of the segmented images do not meet minimum predefined requirements. The error flags only indicate a higher potential of problems with the segmented images. The segmented fingerprints are written to separate files and details about each image (including size, location in the original image, rotation, and error flags) are printed to the standard output. Details of the algorithm are given in Section 4.1.

3.2. BOZORTH3 – Fingerprint Matcher

The BOZORTH3 matching algorithm computes a match score between the minutiae from any two fingerprints to help determine if they are from the same finger. It's a modified version of a fingerprint matcher written by Allan S. Bozorth while at the FBI. The early version of the matching algorithm that NIST has used internally was named bozorth98.[5][7][8] The BOZORTH3 matcher is functionally the same as the bozorth98 matcher, improvements have been made to remove bugs in the code (specifically memory leaks in statically defined variables) and improve the speed of the matcher.

The BOZORTH3 matcher using only the location (x,y) and orientation (theta) of the minutia points to match the fingerprints. The matcher is rotation and translation invariant. The matcher builds separate tables for the fingerprints being matched that define distance and orientation between minutia in each fingerprint. These two tables are then compared for compatibility and a new table is constructed that stores information showing the inter-fingerprint compatibility. The inter-finger compatibility table is used to create a match score by looking at the size and number of compatible minutia clusters.

A detailed description of the BOZORTH3 matching algorithm is included in Section 4.2.

4. ALGORITHMS

It is our understanding that this software falls within ECCN 3D980, which covers software associated with the development, production or use of certain equipment controlled in accordance with U.S concerns about crime control practices in specific countries.

4.1. NFSEG

In order to perform matching of plain to rolled fingerprints the four-finger plain impressions at the bottom of a fingerprint card must be segmented into four single finger images. The NIST segmentation algorithm has proven to be a valuable tool for building testing data from 14 record fingerprint cards by allowing the segmentation of the four-finger plain images. NIST has been able to build large data sets of fingerprints for comparing plain images versus rolled images.[5]-[8] This version of `NFSEG` takes input images in ANSI/NIST [4], WSQ compressed [9], or lossless JPEG compressed format and outputs WSQ or lossless JPEG compressed images, or RAW images. It also allows the output images to be rotated to a vertical position or left unchanged from the original image.

4.1.1. Image Sampling and Binarization

`[/NBIS/Main/imgtools/src/lib/image/imgavg.c; average_blk(), and /NBIS/Main/src/lib/nfsegfing/nfseg.c; dynamic_threshold()]`

To segment the four-finger plain image `NFSEG` attempts to find a set of four evenly spaced fingerprints in the plain image. `NFSEG` first processes the image to prepare for detecting the fingerprints. Figure 2 shows an original image of a four-finger plain image from a right hand. The image is scaled to 1/8 its original size by averaging 8x8 blocks of pixels. The averaged image is then binarized using the mean pixel value as the threshold. If the image has low contrast the threshold is raised in an attempt to enhance the fingerprint areas. Figure 3 shows the scaled and binarized image. Finally, long vertical black lines, caused by the fingerprint card for inked plain impressions, are removed from the edge of the image. The processed image is then used to detect the four fingers in the image.

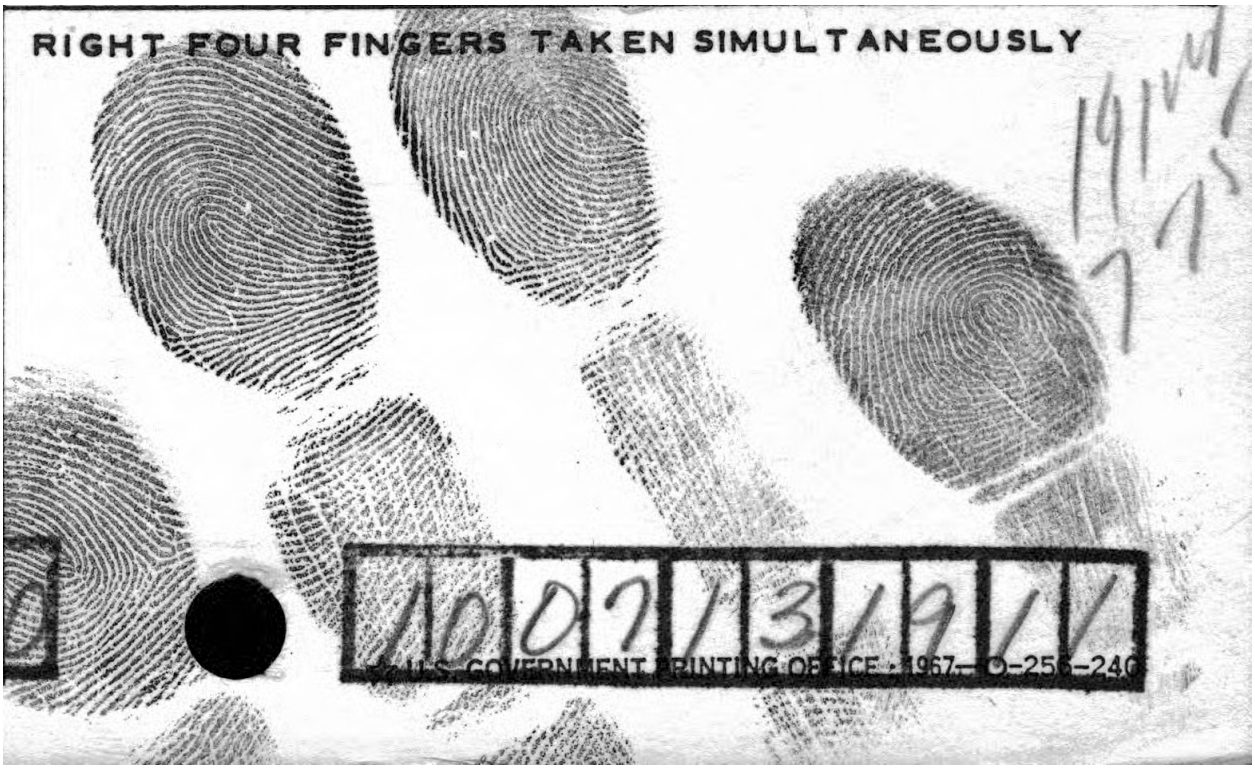

Figure 2. Right four-finger plain image from NIST Special Database 29 (a024.an2).

Figure 3. Image from figure 1 reduced by 1/8 and binarized.

4.1.2. Four-Finger detection `[/NBIS/Main/nfseg/src/lib/nfsegfing/nfseg.c; segment_fingers()]`

The rest of this description assumes the input image is a four-finger plain image from a right-hand which tend to be vertical or angled to the left. The process is the same for plains from the left-hand just flip all the parameters, as the left-hand plains tend to be vertical or angled to the right.

The location of the four fingers in the image is found by searching for large quantities of black pixels along lines that run from the top (x_1, y_1) to the bottom (x_2, y_2) of the image at various

angles. The line points always start at the top of the image ($y_1 = 0$) and end at the bottom ($y_2 = h-1$) where h is the height of the image in pixels. The starting x-values are varied so that the image is searched over a variety of angles to find a set of four fingers in the image. Table 2 shows the range of starting points used for the line end points, this assumes the top left corner of the image is (0, 0) and the bottom right corner is (w-1, h-1). Looking at the values in Table 2 and assuming angles to the right of vertical are positive and to the left negative, a right hand image is searched starting at an angle of about 20 degrees back to –45 degrees at approximately –2 degrees per step. If the hand type is unknown the algorithm will search the full range of angles (45 degrees to –45 degrees) to find the fingers and make a guess at the hand type based on the angle of rotation.

Table 2. Starting line end points used to search four-finger plain impression image.

(x_1,y_1)	(x_2,y_2)
(0,0)	(-45,h)
(0,0)	(+5 each step)
(0,0)	(-5,h)
(0,0)	(0,h)
(-5,0)	(0,h)
(-5 each step)	(0,h)
(-125,0)	(0,h)

At each line, starting at initial points (x_1, y_1), (x_2, y_2) moving left to right in the image by adding 1 to x_1 and x_2, an accumulation is made of the number of black pixels along that line and its +/- 10 neighboring lines (of that same angle). Any points on the line outside the image boundaries are not used. After the black pixel accumulations are made for each line in the image, the process is repeated for white pixel accumulators and the +/- 4 neighboring lines. A set of four potential finger locations are randomly detected based on the values in the black pixel accumulators by looking for the accumulator with the largest value and then setting to zero all that accumulator's +/- 24 neighbors to prevent detecting the same finger twice. This process is repeated four times to detect all four fingers. A normalized score is calculated for each of the four fingers which is the total number of black pixels in the accumulator for that line divided by the total number of pixels searched when accumulating black pixels for that line. The normalized finger scores are added to the total score for this set of four fingers. The set of four finger locations are then organized from left to right. The white accumulator values are used to select the best location to split neighboring fingers and to detect the outside edges of the two end fingers. At each split point and outside edge a normalized score is calculated in the same way as the black accumulator scores were calculated and added to the overall score for the four fingers. The highest scoring four finger sets at each angle are all compared to obtain the highest overall scoring four finger set which are then segmented into individual fingerprint images. Figure 4 shows the four fingers detected in the image from Figure 3.

Figure 4. Lines showing the location of the center and edges of the four fingers.

The next step in segmenting the individual fingers from the highest scoring set of four fingers is to cut out the rectangular areas where the fingers are located and rotate them into a vertical position for further fingerprint edge detection. These rectangular areas are bounded by the edge of the image. If the fingers are rotated, the edges are extended to obtain all pixels from the original image in the cut image. All areas that extend beyond the edges of the image are filled with white pixels. Figure 5 shows the rectangular areas that are segmented for each of the four fingers with the overflow areas in gray to make them more visible.

Figure 5. The four regions used to isolate the fingerprints.

4.1.3. Isolate the Fingerprint

`[/NBIS/Main/nfseg/src/lib/nfsegfing/nfseg.c; get_segfing_bounds()]`

Next, the four rectangular areas, presumed to contain the fingerprints, are used to isolate the plain impressions of all four fingers. This is done by first searching for the top of the fingerprint using a T shaped mask that accumulates white pixel counts along the top of the T and black pixel counts along the column of the T. The width of the T's top is equal to half the width of the image and it is five pixels high. The column of the T is ten pixels wide and half the width of the image in height. The T mask is searched along three vertical lines in the image located at the ¼ width, ½ width and ¾ width points in the image. This improves performance if the fingerprint is not centered in the image. The point along the three vertical lines that gets the largest score determines the top of the fingerprint image. The bottom of the fingerprint is found by starting 15

pixels below the top and then find the first horizontal white line that is over 70% white pixels, a maximum height of 600 pixels or the bottom of the image.

The sides are then located by searching from the center of the fingerprint to the edge looking for the first vertical line that is more than 50% of the fingerprint height in white pixels. If no line has more than 50% of the height in white pixels the edge is set to the edge of the segmented rectangle. Since all the edges were determined from the segmented rectangular region that was rotated to a vertical position for ease of computation, these parameters have to be translated, using geometry, back into the coordinates of the original image. The final step is to compute the center (sx, sy), width (sw), and height (sh) of the finger to be segmented from the original image. This step also looks to see if the corner of the image is being excluded based on the previous edge detection used. This can cause meaningful fingerprint information to be lost in the segmented image. If the corner is excluded on the segmented image the parameters are adjusted to keep this area of the original image in the segmented image.

Next, the size and spacing of the fingerprints are check to determine if they are within a certain tolerance and error flags are set if they are not. There are currently five error values: "0" means no errors were detected, "1" means the image width is less than 25 pixels, "2" means the image height was less than 32 pixels, "3" means the fingers spacing (center to center) was less than 25 pixels, and "4" means the finger spacing was greater than 60 pixels. These current parameters were selected so that anything above these values are assumed segmented correctly. This version of `NFSEG` will always write out four segmented slaps for each input image along with this error number indicating potential problems in the segmentation process.

4.1.4. Segment and save the individual fingerprint images.

`[/NBIS/Main/nfseg/src/lib/nfsegfing/nfseg.c; parse_segfing() and write_parsefing()]`

Finally, all parameters have to be scaled back to the original image size and the fingers can be segmented from the original image and either rotated to vertical or left as is. If the segmented fingerprints are not rotated to the vertical position, any areas outside the bounding box are filled with white pixels to keep from including another fingerprint in the segmented image. Figure 6 shows the segmented fingers before putting white pixels outside the bounding box and Figure 7 shows after filling with white pixels. Figure 8 shows the segmented fingers rotated to the vertical position. Figure 9 and Figure 10 show an example of another four-finger segmentation that has one finger at the top of the card and another in the bottom corner.

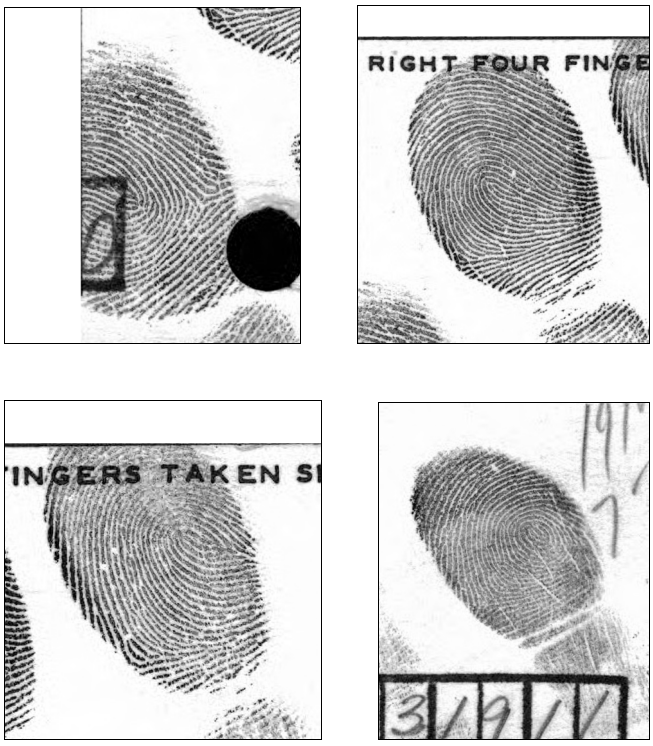

Figure 6. Segmented images before putting white pixels outside the bounding Figure 7 box.

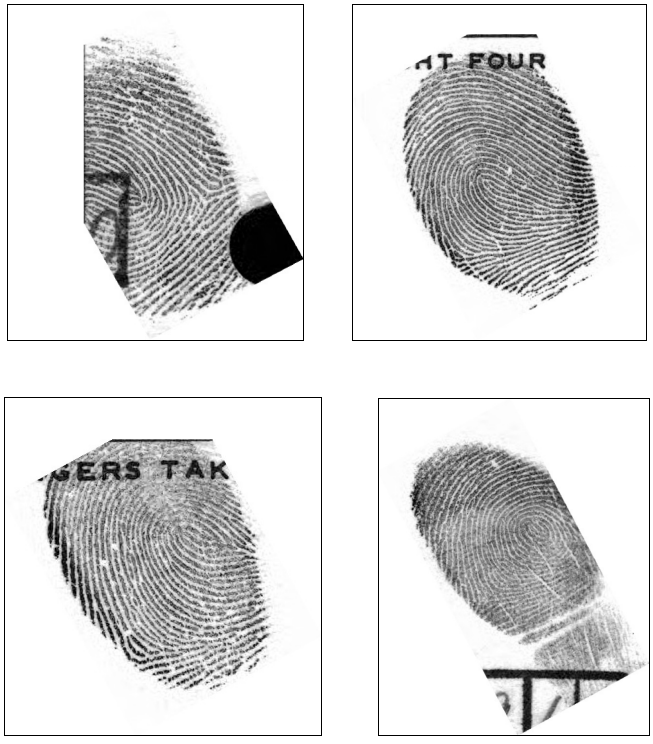

Figure 7. Segmented images after putting white pixels outside the bounding box.

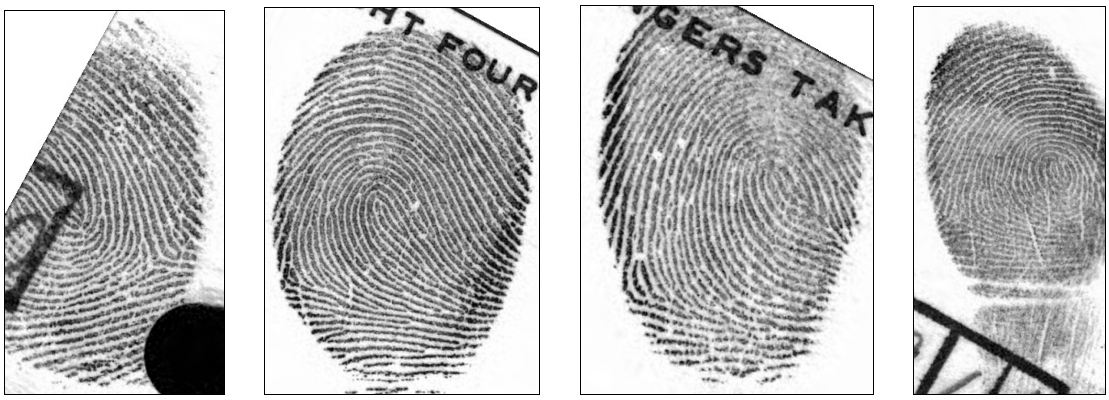

Figure 8. Segmented images rotated to the vertical position.

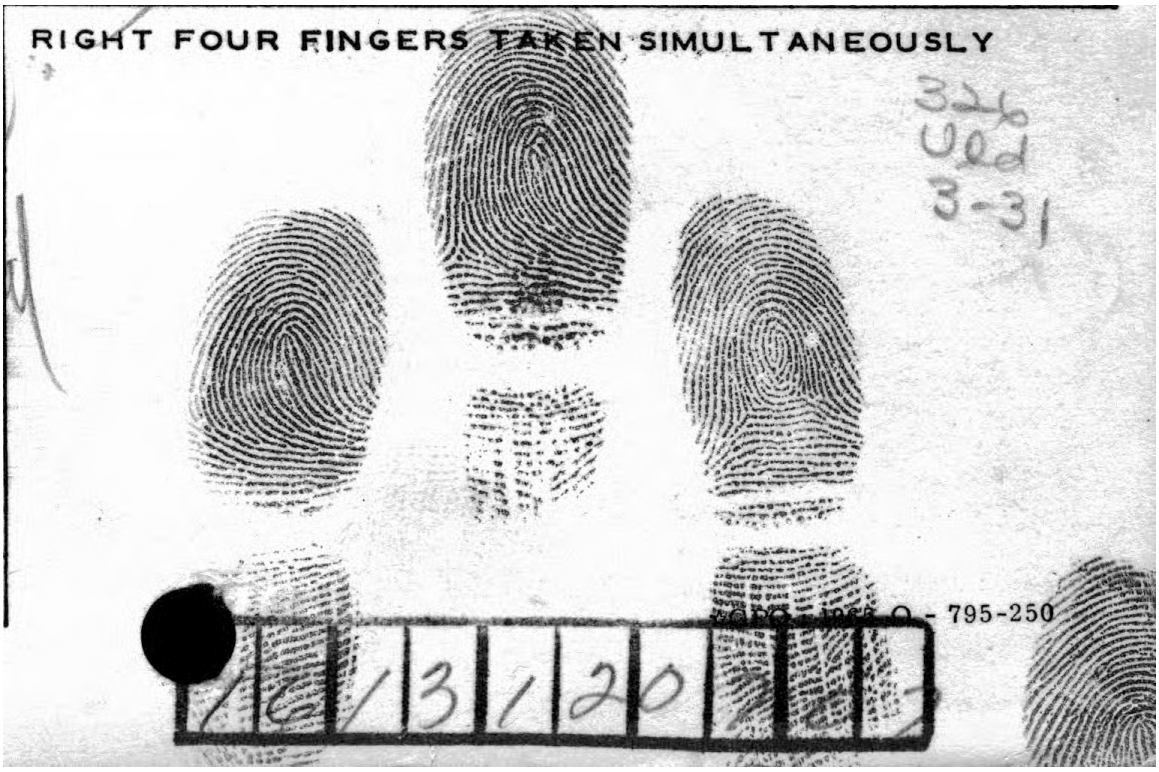

Figure 9. Four-finger plain image with fingers near the edges of the card (a040.an2) from Special Database 29. (a024.an2).

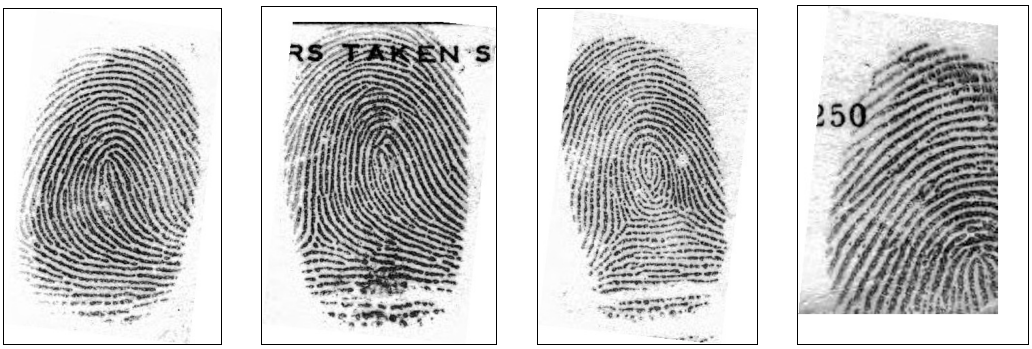

Figure 10. Segmented fingerprints from Figure 9.

The segmented image data can then be written to individual output files as raw pixel data, lossless JPEG compressed data or WSQ compressed data (5:1 or 15:1 compression ratio). NFSEG also writes information about the segmented images to standard output. For each segmented image a line is written to stdout with the following information: output file name, error value, width and height of the segmented image, segmentation point (sx, sy) in the original image, rotation angle of the fingers (same for all four). For example for the images in Figure 7 the output is:

```
FILE a024_13_02.jpl -> e 0 sw 464 sh 528 sx  112 sy 664 th 28.8
FILE a024_13_03.jpl -> e 0 sw 544 sh 624 sx  288 sy 256 th 28.8
FILE a024_13_04.jpl -> e 0 sw 480 sh 512 sx  664 sy 192 th 28.8
FILE a024_13_05.jpl -> e 0 sw 560 sh 704 sx 1200 sy 464 th 28.8
```

4.1.5. Future work

While the current version of the NFSEG has been very valuable in preparing data sets used in evaluating both plain-to-rolled and plain-to-plain fingerprint matching, there are potential improvements to be made to the algorithm. First, better preprocessing of the input image to remove text and other noise could improve the rest of the algorithm's ability to detect the fingerprints. Second, improvements could be made to the edge detection process when isolating the fingerprint images. The current edge detection removes some of the ridges near the edge of the image. Finally, more detailed error checks at the end, which may include fingerprint image quality, could help determine the confidence of a successful segmentation.

4.2. BOZORTH3

Missing from the first distribution of N NFISBIS, was an algorithm and utility for taking features extracted from two fingerprints (such as the minutiae detected in `MINDTCT`) and matching them together for either the purpose of one-to-one verification or one-to-many identification. This type of algorithm is commonly referred to as a fingerprint "matcher." We consider it great news that a fingerprint matcher has been included for the first time in this new release of NIST fingerprint software referred to as NBIS.

4.2.1. Background

From 1995 through 2002, the FBI had a demonstration display to showcase the Integrated Automated Identification System (IAFIS) and its interoperability with the National Crime Information Center (NCIC2000). This included a full-size reconstructed police cruiser many floors up in the J. Edgar Herbert Hoover Building. One purpose for this demonstration was to simulate real-time fingerprint acquisition, search, and response remotely from the field.

Prior to this demonstration project, an FBI employee by the name of Allan S. Bozorth, had set off on an effort to investigate the notion of a translation and rotation invariant algorithm for matching two fingerprints to each other. Allan was successful in designing such an algorithm, and had implemented it in software. He tested the matcher extensively using NIST fingerprint data that was available, and in Allan's own words, "I was pleasantly surprised at the results," but "I still did not have a good feel for its performance."

It was around this time that the construction of the demonstration display began. In the demonstration, the Home Office algorithm for minutiae detection (the algorithm on which `MINDTCT` is based) was used; and when the need for a fingerprint matcher arose, Allan's algorithm was selected and integrated into both the IAFIS and NCIC portions of the display. The matcher performed at an adequate level to support the demonstration where a guest could be enrolled and searched against a very small background.

Much to Allan's surprise and credit, this algorithm has since been extensively used as a technology benchmark by NIST to support work under the U.S. Patriot Act.[5] The algorithm has been shown to perform respectably well for both verification and identification applications. In honor of Allan's hard work and accomplishments, NIST has chosen to name this matcher the "Bozorth Matcher," or in short "Bozorth." A description of the algorithm follows.

4.2.2. Bozorth Algorithm

`[/NBIS/Main/bozorh3/src/lib/bozorth3fing/bozorth3.c]`

Two key things are important to note regarding this fingerprint matcher:

1. Minutia features are exclusively used and limited to location (x,y) and orientation 't', represented as {x,y,t}.
2. The algorithm is designed to be rotation and translation invariant.

The algorithm is comprised of three major steps:

1. Construct Intra-Fingerprint Minutia Comparison Tables
   a. One table for the probe fingerprint and one table for each gallery fingerprint to be matched against
2. Construct an Inter-Fingerprint Compatibility Table
   a. Compare a probe print's minutia comparison table to a gallery print's minutia comparison table and construct a new compatibility table
3. Traverse the Inter-Fingerprint Compatibility Table
   a. Traverse and link table entries into clusters
   b. Combine compatible clusters and accumulate a match score

4.2.2.1. Construct Intra-Fingerprint Minutia Comparison Tables

[/NBIS/Main/bozorth3/src/lib/bozorth3fing/bozorth3.c; bz_comp() & bz_find()]

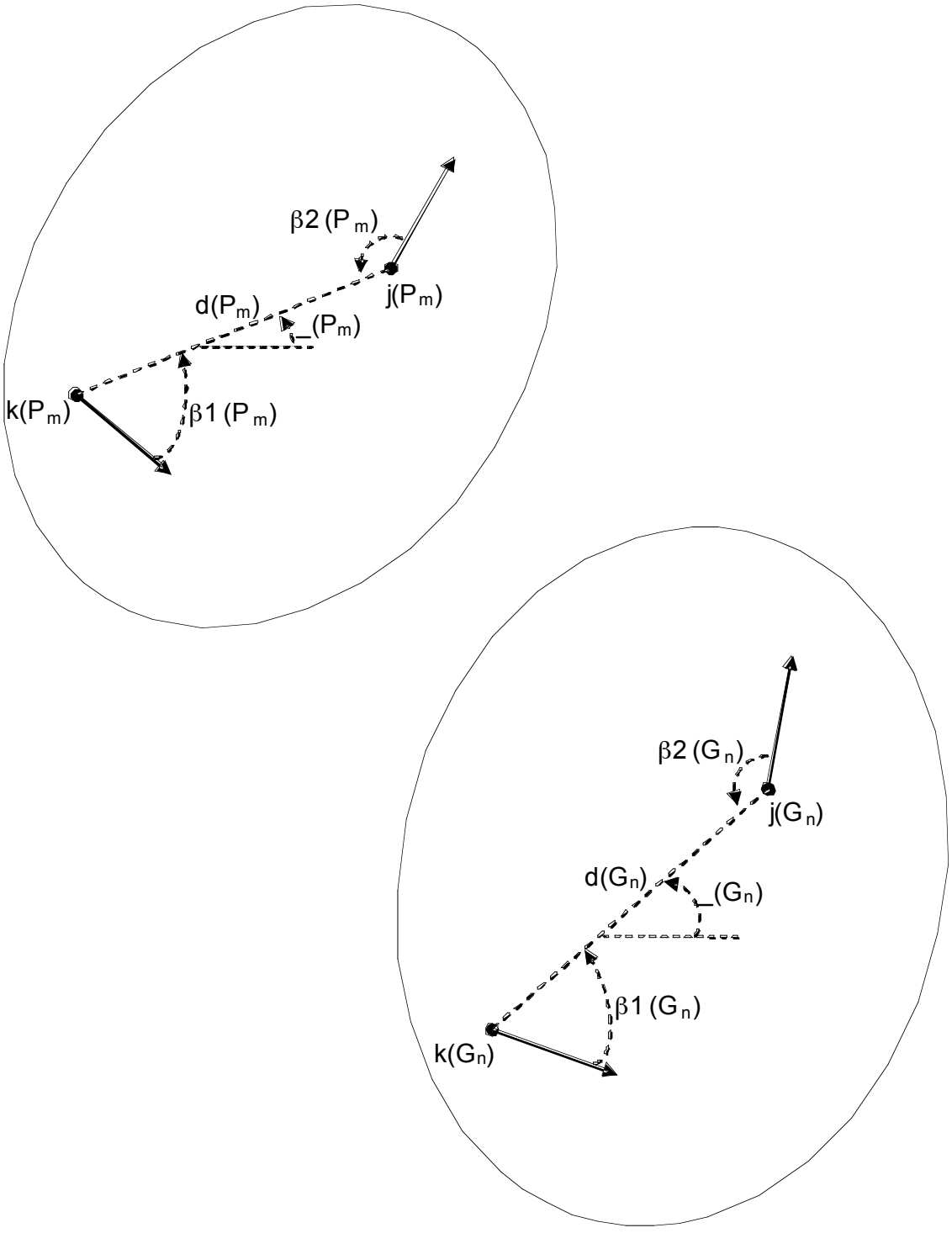

Figure 11. Intra-fingerprint minutiae comparison.

The first step in the Bozorth Matcher is to compute relative measurements from each minutia in a fingerprint to all other minutia in the same fingerprint. These relative measurements are stored in a *minutia comparison table* and are what provide the algorithm's rotation and translation invariance.

Figure 11 illustrates the inter-minutia measurements that are used. There are two minutiae shown in this example. Minutia k is in the lower left of the "fingerprint" and is depicted by the dot representing location (x_k,y_k) and the arrowed line pointing down and to the right representing orientation t_k. A second minutia j is in the upper right with orientation pointing up and to the right. To account for relative translational position, the distance d_{kj} is computed between the two minutia locations. This distance will remain relatively constant between corresponding points on two different finger impressions regardless of how much shifting and rotating may exist between the two prints.

To make relative rotational measurements is a bit more involved. The objective for each of the minutiae in the pair-wise comparison is to compute the angle between each minutia's orientation and the intervening line between both minutiae. This way, these angles remain relatively constant to the intervening line regardless of how much the fingerprint is rotated. In the illustration above, the angle θ_{kj} of the intervening line between minutia k and j is computed by taking the arctangent of the slope of the intervening line. Angles β_k and β_j are computed relative to the intervening line as shown by incorporating θ_{kj} and each minutia's orientation t. It should be noted that the point-wise comparison is conducted on minutia positions sorted first on x-coordinate, then on y-coordinate, and that all orientations are limited to the period (-180°,180°] with 0° pointing horizontal to the right and increasing degrees proceeding counter clockwise.

For each pair-wise minutia comparison, an entry is made into a comparison table. Each entry consists of:

$$\{d_{kj}, \beta 1, \beta 2, k, j, \theta_{kj}\}$$

where

$$\beta 1 = \min(\beta_k,\beta_j) \text{ and } \beta 2 = \max(\beta_k,\beta_j)$$

so that in the illustration above,

$$\beta 1 = \beta_k \text{ and } \beta 2 = \beta_j$$

Entries are stored in the comparison table in order of increasing distance and the table is trimmed at the point in which a maximum distance threshold is reached. Making these measurements between pairs of minutiae, a comparison table must be constructed for each and every fingerprint you wish to match with or against.

4.2.2.2. Construct an Inter-Fingerprint Compatibility Table

`[/NBIS/Main/bozorth3/src/lib/bozorth3fing/bozorth3.c; bz_match()]`

The next step in the Bozorth matching algorithm is to take the minutia comparison tables from two separate fingerprints and look for "compatible" entries between the two tables. Figure 12 depicts two impressions of the same fingerprint with slight differences in both rotation and scale. Two corresponding minutia points are shown in each fingerprint.

The upper left print represents a probe print in which all its minutiae have been pair-wised compared with relative measurements stored in minutia comparison table P. The measurements computed from the particular pair of minutia in this example have been stored as the m^{th} entry in table P, denoted P_m. The notation of individual values stored in the table are represented as lookup functions on a given table entry. For example, the index of the lower left minutia is stored in table entry P_m and is referenced as $k(P_m)$, while the distance between the two minutiae is also stored in table entry P_m and is referenced as $d(P_m)$. The lower right fingerprint represents a gallery print, and uses similar notation, except that all its pair-wise minutia comparisons have been stored in table G, and the measurements made on the two corresponding minutia in the gallery print have been stored in table entry G_n.

The following three tests are conducted to determine if table entries P_m and G_n are "compatible." The first test checks to see if the corresponding distances are within a specified tolerance T_d. The last two tests check to see if the relative minutia angles are within a specified tolerance T_β. Δ_d () and Δ_β () are "delta" or difference functions.

$$\Delta_d(d(P_m),d(G_n)) < T_d$$

$$\Delta_\beta(\beta 1(P_m),\beta 1(G_n)) < T_\beta$$

$$\Delta_\beta(\beta 2(P_m),\beta 2(G_n)) < T_\beta$$

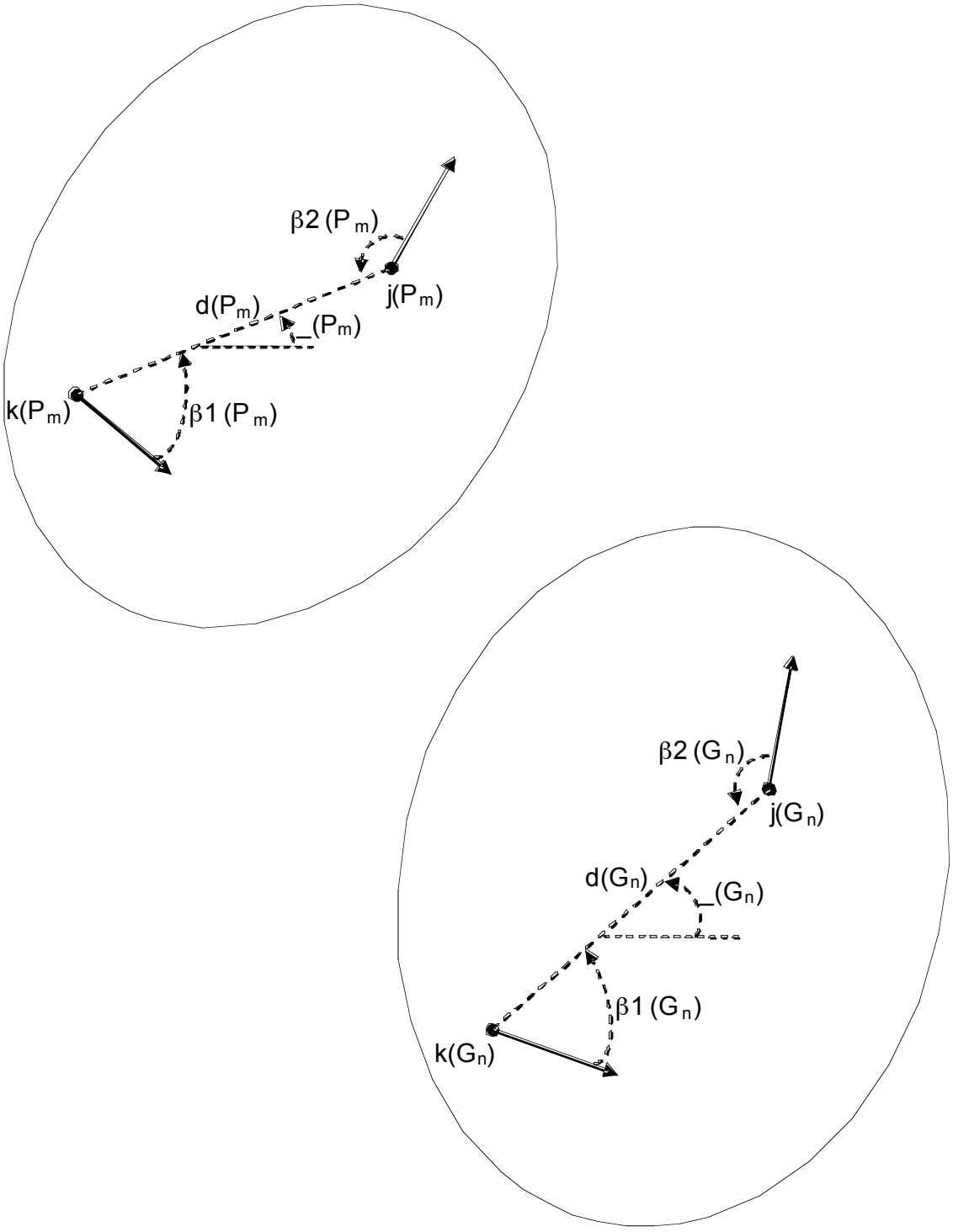

Figure 12. Compatible pair-wise minutia measurements between two fingerprints that generate an entry into an inter-fingerprint compatibility table

If the relative distance and minutia angles between the two comparison table entries are within acceptable tolerance, then the following entry is entered into a *compatibility table*:

$$\{\Delta_\beta(\theta(P_m),\theta(G_n)),\ k(P_m),\ j(P_m),\ k(G_n),\ j(G_n)\}$$

A compatibility table entry therefore incorporates two pairs of minutia, one pair from the probe fingerprint ($k(P_m)$, $j(P_m)$) and the other from the gallery fingerprint ($k(G_n)$, $j(G_n)$). The entry into the compatibility table then indicates that $k(P_m)$ corresponds with $k(G_n)$ and $j(P_m)$ corresponds with $j(G_n)$. The first term in the table entry, $\Delta_\beta(\theta(P_m),\theta(G_n))$, is used later to combine clusters that share a similar amount of global rotation between "compatible" probe and gallery minutiae.

4.2.2.3. Traverse the Inter-Fingerprint Compatibility Table

`[/NBIS/Main/bozorth3/src/lib/bozorth3fing/bozorth3.c; bz_match_score(), bz_sift() & bz_final_loop()]`

At this point in the process, we have constructed a compatibility table which consists of a list of compatibility association between two pairs of potentially corresponding minutiae. These associations represent single links in a *compatibility graph*. To determine how well the two fingerprints match each other, a simple goal would be to traverse the compatibility graph finding the longest path of linked compatibility associations. The match score would then be the length of the longest path.

There are some serious challenges to such a simple approach. These include:

1. The compatibility table is not a coherent graph, but rather a disjoint collection of single links within a graph.
2. Each node in the graph is potentially linked to many other nodes.
3. This leads to the potential for circuits.
4. There is no obvious root node in the graph that can be predicted to lead to the maximum path.
5. Occlusions and/or voids within either of the two fingerprints being matched will cause discontinuities in the graph.

To account for these issues, Allan Bozorth implemented an algorithm that processes the compatibility table so that traversals are initiated from various staring points. As traversals are conducted, portions or clusters of the compatibility graph are created by linking entries in the table. Once the traversals are complete, "compatible" clusters are combined and the number of linked table entries across the combined clusters is accumulated to form the match score. The larger the number of linked compatibility associations, the larger the match score, and the more likely the two fingerprints are from the same person, same finger.

4.2.3. BOZORTH3 Implementation

```
[/NBIS/Main/bozorth3/src/lib/bozorth3fing/bz_drvrs.c; bozorth_main(), also
see files /NBIS/Main/bozorth3/src/lib/bozorth3fing/bozorth3.c and
NBIS/Main/bozorth3/src/lib/bozorth3fing/bz_*.c]
```

Given the above matching algorithm, Allan Bozorth's original program computed a match score from a single pair of fingerprints. An early version of this one-to-one matcher, named "bozorth98," has been used extensively by NIST in benchmark studies.[5] The implementation included in this distribution, named `BOZORTH3`, has been modified to compute a match score between the minutiae from any number of fingerprint pairs. For example, it can compute match scores between a probe fingerprint and any number of gallery fingerprints. To process a probe set of size P against a gallery of size G, the original program needed to read and preprocess 2*(P*G) files. Preprocessing consists of parsing the minutia {x,y,t}'s from file, sorting the {x,y,t}'s and trimming the period on the orientations, and then computing a minutia compatibility table. The one-to-many capability means each probe file is read and preprocessed just once, reducing the number of files read and preprocessed to P+(P*G). The only additional memory requirement is the space needed to internally store a single preprocessed probe finger's minutia compatibility table. In addition to the modifications to support more efficient one-to-many matching, `BOZORTH3` has been reorganized, optimized, and made more portable. Several minor bugs were also fixed.

By default, `BOZORTH3` produces one line for each match it computes, containing only the match score. Ideally, the match score is high if the two sets of input minutiae are from the same finger of one subject, and low if they're from different fingers. The implementation of the match table traversal described above is non-exhausted and therefore does not guarantee an optimal outcome. The resulting match score roughly (but not precisely) represents the number of minutiae that can be matched between the two fingerprints. As a rule of thumb, a match score of greater than 40 usually indicates a true match. For performance evaluation, see [5][7][8]. The match scores from `BOZORTH3` are usually, but not always, identical to those published from "bozorth98." Changes in floating point computations, one logic correction effecting the match table traversal, and modifications to avoid illegal array indices are primarily responsible for the scores differing.

By default, only the 150 best-quality minutiae (minutiae quality are determined by the minutiae extraction algorithm, currently `MINDTCT`) from each input fingerprint are used. That should be more than enough -- a finger typically has fewer than 80 minutiae -- so a minutiae extractor can be used even if it's overly sensitive producing many false minutiae.

4.2.3.1. Troubleshooting

If the program dies immediately on start-up with no useful error message, it's likely that too much memory is being allocated for the implementation's many static arrays. Depending on what platform the program is being used to compile and execute the software, there may be ways to increase the amount of space a process can allocate for static arrays. For example, the shells "tcsh" and "bash" have the built-in commands "limit" and "ulimit," respectively, to control various run-time limits, such as a process' maximum stack and data segment sizes. The "gcc" compiler has many options that can also affect a program's memory allocation, although many are platform-dependent.

4.2.3.2. Contact

Direct any technical questions, bug reports, or suggestions, etc. to Stan Janet <sjanet@nist.gov>. For bug reports, please send the input files that triggered the problem, and document all relevant information including:

* The `BOZORTH3` command line used.

* Any error codes and messages the command produced.

* The OS version and kernel version.

* The compiler version and options.

Run `BOZORTH3` with the "-v" flag to produce verbose output that may help pinpoint the problem. If possible, compile the program with the debugging option "-g" and use a debugger to pinpoint the error and dump as many variable values right before that point.

5. REFERENCES

(Some of the references listed can be downloaded at http://www.itl.nist.gov/iad/894.03.)

[1] GNU project - free UNIX-like utilities. Learn more at http://www.gnu.org.

[2] Cygwin tools - free GNU utility port for Win32 machines. Learn more at http://www.cygwin.com/.

[3] Linux - a freely available clone of the UNIX operating system. Learn more at http://www.linux.org.

[4] R.M. McCabe, "Data Format for the Interchange of Fingerprint, Facial, Scar Mark & Tattoo (SMT) Information," American National Standard ANSI/NIST-ITL 1-2000, July 2000. Available from R.M. McCabe at NIST, 100 Bureau Drive, Stop 8940, Gaithersburg, MD 20899-8940.

[5] C. Wilson, M. Garris, C. Watson, A, Hicklin, "Studies of Fingerprint Matching Using the NIST Verification Test Bed (VTB)," Technical Report NISTIR 7020, July 2003. http://www.itl.nist.gov/iad/894.03/pact/pact.html.

[6] E. Tabassi, C. Wilson, C. Watson, "Fingerprint Image Quality," Technical Report 7151, August 2004. Appendices for NISTIR 7151 can be found at http://fingerprint.nist.gov/NBIS.

[7] C. Wilson, A. Hicklin, B. Ulery, H. Korves, M. Zoepfl, P. Grother, R. Michaels, C. Watson, S. Otto, M. Bone, "Fingerprint Vendor Technology Evaluation (FpVTE) 2003," Technical Report NISTIR 7123, June 2004. http://fpvte.nist.gov/

[8] C. Watson, C. Wilson, M. Indovina, R. Snelick, K. Marshall, "Studies of One-to-One Matching with Vendor SDK Matchers," Technical Report NISTIR 7119, July 2004.

[9] "WSQ Gray-scale Fingerprint Image Compression Specification," Criminal Justice Information Services, FBI, December 1997.

[10] "User's Guide to NIST Biometric Image Software" is located in `NBIS/Main/doc/refs/nbis_non_export_control.pdf` on the CD-ROM.

APPENDIX A. REQUEST NBIS EXPORT CONTROL SOURCE CODE CD-ROM

NFSEG and BOZORTH3 are software packages that are subject to U.S. export control laws. It is our understanding that this software falls within ECCN 3D980, which covers software associated with the development, production or use of certain equipment controlled in accordance with U.S. concerns about crime control practices in specific countries. Therefore, the source code, documentation, and testing utilities for NFSEG and BOZORTH3 are only distributed via CD-ROM to qualifying requesters.

To request a CD-ROM copy of these export controlled packages, please send email to nbis_ec@nist.gov with the following information:

Email Header -

Subject : NBIS EXPORT CONTROL SOURCE CODE CD-ROM

Email Text Body -

Name : (Your Name)
Organization : (Organization Name or Affiliation)
Mailing Address : (Complete Postal Mailing Address)
Phone Number : (Working Phone Number)

For new releases information, please visit http://fingerprint.nist.gov/NBIS/index.html.

APPENDIX B. REFERENCE MANUAL

This appendix contains manual pages describing the invocation and use of the utilities provided in this software distribution. The utilities are listed in alphabetical order. Those belonging to the NFSEG package are designated as belonging to command set (1A), PCASYS as (1B), MINDTCT as (1C), NFIQ as (1D), BOZORTH3 as (1E), AN2K as (1F), IMGTOOLS as (1G), and IJG utilities are designated as (1H).

The manual pages listed in this section are in directory /NBIS/Main/man. To view a man, type:

```
% man -M <install_dir>/man <executable>
```

where the text <install_dir> is replaced by your specific installation directory path and <executable> is replaced by the name of the utility of interest.

NAME

bozorth3 – Computes match scores between fingerprints

SYNOPSIS

bozorth3 [*options*] *probe-file.xyt gallery-file.xyt ...*
bozorth3 [*options*] **–M** *mates.lis*
bozorth3 [*options*] **–P** *probe.lis gallery*.xyt*
bozorth3 [*options*] **–G** *gallery.lis probe*.xyt*

bozorth3 [*options*] **–p** *probe-file.xyt gallery*.xyt*
bozorth3 [*options*] **–p** *probe-file.xyt* **–G** *gallery.lis*
bozorth3 [*options*] **–g** *gallery-file.xyt probe*.xyt*
bozorth3 [*options*] **–g** *gallery-file.xyt* **–P** *probe.lis*

DESCRIPTION

The program *bozorth3* computes match scores from fingerprint minutiae files. The files are expected to be in xyt-format, a simple text file format that is produced by the minutiae detector program *mindtct*, which is also part of the NFIS distribution.

By default, each pair of arguments on the command line is considered to be a probe file and a gallery file, in that order, that are to be matched to yield a score of similarilty. The higher the score, the more closely the minutiae in them match. The match score for known mates is often close to the number of minutiae in the probe or gallery file, but it can be lower or higher than that number, sometimes much higher.

There are two main mechanisms that allow running the *bozorth3* matcher other than by simply specifying pairs of xyt-files on the command line. The mechanisms are useful or necessary under several circumstances. For example, with large data sets, the number of pairs of files to be matched could easily exceed the maximum size the user's shell permits on a command line, or that's permitted by *exec*()* system calls. And there are cases where it's just more logical to have input filenames stored in a file. So one mechanism uses *list files* — they contain xyt-filenames, one per line, with newline characters as line-endings. The other mechanism fixes the probe (or gallery) file for an entire run, so that the filename doesn't have to be specified over and over again.

One form of the list file mechanism allows the pairs of files to be read from a single file. The *-M mates.lis* option requires a single list file of filenames to be matched against each other. The probe filenames are on the odd lines, and the gallery filenames are on the even lines.

Similarly, the *-P probes.lis* option specifies that the probe filenames are in the file, and the gallery filenames come from the command line. The *-G gallery.lis* option specifies just the opposite. Both options may be present, in which case all filenames will be read from the two files, and there will be no xyt-files on the command line.

The other subset of mechanisms fix a single file to be matched against a gallery (or probe) set of any size. For example, *-p probe-file* fixes the probe file for the entire run; it will be matched against a gallery consisting of all other files on the command line (or, if *-G gallery.lis* is specified, against a gallery read from a file).

The *-g gallery-file* option specifies just the opposite. While it may seem illogical to reverse the notion of probe and gallery files by allowing a single gallery file to be compared against a probe set, it's allowed both for consistency and to make it easier to test how close scores are when the files are matched in reverse order.

Fixing both the probe and gallery file is legal, but it's equivalent to having just a single pair of filenames on the command line without the *-p* and *-g*.

The score for a probe file *a* matched to a gallery file *b* is often identical to the score for *b* matched to *a*. One one data set, the scores were the same more than 75% of the time, and only a very small number were different by more than 3.

Minutiae file format

Each line in a minutiae file contains three integers, representing the x- and y-coordinates and direction of the minutiae, and an optional fourth column of integers representing the quality of the minutiae at those coordinates. If the quality column isn't present in a file, all minutiae are assumed to be of the same quality.

A finger typically has 40-80 minutiae. Any automated minutiae extractor will, of course, flag some things as minutiae that aren't. To work with highly sensitive minutiae detectors such as *mindtct*, the *bozorth3* matcher allows each xyt-file to contain as many as 1000 minutiae lines. However, by default, only the 150 highest-quality minutiae are used to compute the match score. That number may be changed to any number from 0 to 200. If multiple minutiae have the same quality value at the cut-off point, the tie-breaking method is simple truncation of the list, sorted by quality but with an undefined sort order among its equal-quality elements.

The optimal number of minutiae that should be used depends on the fingerprint images and the minutiae detector that processes them. Using more than is necessary typically reduces the accuracy of the matcher and increases its run time.

To compute a match score between two fingerprints, both sets must have at least a minimum number of minutiae. That number is 10 by default, and can be changed to any non-zero integer. Otherwise the computation returns a match score of 0.

OPTIONS

The command line options can be logically grouped into four classes:

General options

-h Print a help screen detailing the command line options.

-V Print "bozorth3" version and exit.

-v Enable verbose mode.

-A verbose=<section>
: Enable verbose mode in a section of the code; the recognized sections are: main, load, bozorth, threshold.

Input options

-m1 all xyt files use representation according to ANSI INCITS 378-2004. This flag must be used if it was used by the **mindtct** algorithm when extracting the minutiae points.

-n max-minutiae
: Set maximum number of minutiae to use from any file [150]; the legal range is [0,200].

-A minminutiae=#
: Set minimum number of minutiae required for the match score to be more than 0 [10].

-A maxfiles=#
: Set maximum number of files in any gallery, probe, or mates list file [10000].

-A plines=#-#
: Process a subset of files in the probe file.

-A glines=#-#
: Process a subset of files in the gallery file.

-A dryrun
: Test mode only. Do not compute and print any match scores, just print the filenames between which match scores would be computed.

Thresholding options

-T threshold
: Set match score threshold. By default, all match scores are printed. However, when a threshold specified, only match scores meeting or exceeding that value are printed.

-q
: Quit processing the probe file when a gallery file is found for which the match score meets or exceeds the specified threshold.

Output options

-A nooutput
: Compute match scores, but don't print them.

-A outfmt=[spg]*
: Output lines will contain (s)core, (p)robe and/or (g)allery filename. By default, only scores are output.

-O score-dir
: Set the directory to write score files in.

-o score-file
: Set the filename to store scores in.

-e stderr-file
: Set the filename to store all other output in.

-b
: Use the default Standard I/O buffering to print the match scores. This is equivalent to line-buffering when the output is being printed to a terminal, and to block-buffering when the output is being printed to a file.

-l
: Use line-buffering to print the match scores. By default, output lines are stored and printed just prior to the *bozorth3* exiting.

SEE ALSO

mindtct (1C)

AUTHOR

Allan S. Bozorth of the FBI; modified by Michael Garris and Stan Janet, both of NIST/ITL/DIV894/Image Group.

NAME

nfseg – Segments a four finger plain impression into four individual fingers or removes white space from a single finger rolled/plain impression isolating just the fingerprint portion of the image.

SYNOPSIS

nfseg *<FGP> <BTHR_ADJ> <ROT_SEARCH> <COMP_SEG> <ROT_SEG> <file>*

DESCRIPTION

Nfseg takes an four finger plain impression fingerprint image (FGP finger positions 13 "right hand", 14 "left hand", or 00 "Unknown Hand") and segments it into four individual fingerprint images. For single finger images (FGP 01-12) **nfseg** can be used to isolate the fingerprint in the image by removing white space around the fingerprint. The input image can be in ANSI/NIST format, WSQ compressed format, baseline JPEG, or lossless JPEG. **Nfseg** will output the results to WSQ compressed files (.wsq at 5:1 or 15:1 compression ratio), lossless JPEG (.jpl) or RAW pixel data (.raw). **Nfseg** can also rotate the segmented image to a vertical position or leave it in its original position. If the fingerprint is left in its original position all pixels in the image outside the fingerprint bounding box are set to white.

OPTIONS

<FGP>

indicates the finger position of the input image.

1-5 - right hand rolled impressions
6-10 - left hand rolled impressions
11 - right thumb plain impression
12 - left thumb plain impression
13 - right hand four finger plain impression (tend to be rotated left of vertical)
14 - left hand four finger plain impression (tend to be rotated right of vertical)
00 - unknown hand (for four finger plain impression will search all angles and make guess at hand based on rotation angle)

For segmentation purposes **1-12** are treated the same. Plain impressions **13-14** are treated similar the exception being the angles of rotation that are search for the four fingers. Unknow hand type **00** for four finger plain impression are searched over the whole range of angles used in both left and right hand images. The algorithm will make a guess at left or right hand based on the rotation angle of the fingerprints.

<BTHR_ADJ>

Adjust binarization threshold to improve segmentation of low contrast images.

0 - No threshold adjustment.
1 - Adjust binarization threshold higher.

<ROT_SEARCH>

Allows user to search the image for rotated fingerprints (ie. 2" data) or assume the fingerprints are vertical (ie. 3" data).

0 - Assume fingerprints are not rotated.
1 - Assume fingerprints may be rotated in the image.

<COMP_SEG>

indicates the type of output image for the segmented images.

0 - Lossless JPEG
1 - WSQ 5:1
2 - WSQ 15:1

3 - RAW pixel data

<*ROT_SEG*>
indicates if output image is to be rotated to vertical position.

0 - Do not rotate
1 - Rotate to vertical

<*file*> the input file to be segmented.

NFSEG OUTPUT

For each output image, **nfseg** will print a line of text to standard output as follows:

<*filname*> -> <*error flag*> <*sw*> <*sh*> <*sx*> <*sy*> <*theta*>

<*filename*> the name of the segmented output file.

<*error flag*>
a flag set if the segmented image does not meet certain minimum requirements.

0 - no errors
1 - segmented image width is less than "FING_WIDTH_MIN" parameter in nfseg.h
2 - segmented image height is less than "FING_HEIGHT_MIN" parameter in nfseg.h
3 - spacing between adjacent fingers was less than "FING_SPACE_MIN" parameter in nfseg.h
4 - spacing between adjacent fingers was more than "FING_SPACE_MAX" parameter in nfseg.h

Error flags 3 and 4 usually indicate the image did not segment correctly.

<*sw*> <*sh*>
the width and height of the segmented image file

<*sx*> <*sy*>
where the center of the segmented image was located at in the original four finger plain imression image.

<*theta*>
rotation angle of the fingers in the four finger plain impression image, positive values are left of vertical and negative values are right.

EXAMPLES

From *test/nfseg/execs/nfseg/nfseg.src*:

% nfseg 13 1 1 1 0 a011_13.wsq
segments the image into four single finger plain images. Creates the output files: a011_13_0[2-5].wsq

SEE ALSO

dwsq(1G), **djpegl**(1G), **an2ktool**(1F), **txt2an2k**(1F), **nfiq**(1D)

AUTHOR

NIST/ITL/DIV894/Image Group