Quantitative Research on Friction Ridge Patterns

Analysis of Level 3 Features at High Resolutions

User's Guide to L3TK
Level 3 Fingerprint Image Toolkit

March 2008

International Biometric Group

Revision History

Version	Date	Section	Changes
1.0	31 March 2008	N/A	N/A

Abstract

This document contains operation instructions for the applications provided as part of *Quantitative Research on Friction Ridge Patterns: Phase II Analysis of Level 3 Characteristics at High Resolutions* for the National Institute of Justice. The applications and the associated SDK are collectively known as L3TK. L3TK is designed to perform feature extraction, matching, and quality analysis on high resolution fingerprint images. For information on the SDK portion of L3TK, see the API documentation.

Table of Contents

1 Overview

L3TK consists of the collection of applications and the associated software development kit developed for Phase II Analysis of Level 3 Feature at High Resolutions for the National Institute of Justice. The software for extraction of Level 3 fingerprint features can function as a standalone application, but for extraction of Level 2 features and for matching of any feature classes, the NIST Biometric Image Software (NBIS) suite is required.

This guide contains instructions for execution of the L3TK command line applications. The L3TK command line applications include tools for feature extraction, bulk matching performance evaluation, and image quality analysis. For information on the SDK, see the API documentation for the SDK. For information on the algorithms and techniques used for extraction and matching, see the research report.

2 Installation

L3TK is packaged as a ZIP archive named `nij-l3tk-X.Y` where *X.Y* is the version. Unzip this archive into an empty directory, hereafter referred to as `install_dir`.

The files and directories listed in Table 1 should be present in the installation directory.

Directory	Files	Description

`install_dir`	`nij-extractor.jar`
	`nij-evaluator.jar`
	`iqm-analyzer.jar`
		L3TK application binaries

`install_dir/src`	`nij-l3tk-X.Y-src.jar`	Source code files for L3TK SDK

`install_dir/lib`	`ibg-mig.jar`
	`commons-logging.jar`
	`bailey.jar`
	`Jama-1.0.2.jar`
	`mindtct.exe`
	`cygwin1.dll`
	`blas.jar`
	`blas_simple.jar`
	`f2jutil.jar`
	`lapack.jar`
	`lapack_simple.jar`
	`xerbla.jar`
		Required software libraries and NBIS MINDTCT Windows binary executable.
		
		Libraries in blue are required only by certain parts of the SDK and are not required by the application executables.

`install_dir/doc`	`NIJ-QRFRP L3TK User's Guide.pdf`	This user's guide
`install_dir/doc/api`	`nij-l3tk-X.Y-doc.zip`	API documentation for L3TK SDK

`install_dir/doc/lic`	`gnu-gpl2.txt`
	`apache-LICENSE-2.0.txt`
	`f2j-license`
		Licenses for open source software distributed with L3TK

Table 1: L3TK files and directories

If all the above files and directories are present, the software is ready for use. For operation in non-Windows environments, the NBIS source must be compiled and the `mindtct` executable placed in the installation directory `install_dir`.

Platform Requirements

The L3TK applications and SDK are written in Java™ and require JRE 1.6+. Although the applications and SDK have been developed in a platform independent interpreted language, some extraction and all matching requires the NIST Biometric Image Software (NBIS) suite. The NBIS source is written in ANSI C and has been designed and tested to work with GNU/Linux and the Mac OS X operating system. NBIS may also be compiled to run on Windows machines by first installing the Cygwin library and associated tools. See the *User's Guide to NBIS* for more information.

External Software Requirements

The NBIS `mindtct` application is required for some extraction processes. The `mindtct` application is available in the non-export controlled portion of NBIS, and a precompiled Windows binary executable has been included with the L3TK distribution, along with the required `cygwin1.dll` library.

The NBIS `bozorth3` application is required for matching as evaluated in the *Quantitative Research on Friction Ridge Patterns* report. The `bozorth3` application is part of the export-controlled portion of NBIS, which must be requested from NIST and delivered on CD via post. Visit the NIST web site, nist.gov, for more information and to obtain a copy of the export-controlled software.

3 Feature Extractor

The feature extractor may be used to extract pores, minutia-constrained ridge contour points, edgeoscopic features, and/or minutiae from a fingerprint image. Each feature type is then saved to an output directory in a format compatible with the Bozorth3 matcher. The process flow is as follows:

1. The image is opened and preprocessed.
2. Desired features are extracted based upon the specified flags.
3. Extracted features are written to template files.
4. Extracted quality data is written to quality files.
5. Diagnostic images are written as specified by the configuration file.

If a directory of images is specified in the command-line, steps 1-5 are repeated for each image.

Performance Evaluator application	
Executable	`nij-extractor.jar`
Dependencies	`ibg-mig.jar`
External Dependencies	`mindtct` exectuable

Table 2: Feature Extractor - key application information

Application Processing

For a detailed explanation of the feature extraction process, see the **Feature Extraction** section in the research report on.

Execution Syntax

The command syntax is as follows:

```
java –jar nij-extractor.jar pathname1 dpi pathname2 [pathname3] [OPTIONS]
```

where:

- *pathname1* is the path to a fingerprint image or directory of fingerprint images
- *dpi* is the resolution in DPI of the fingerprint image(s)
- *pathname2* is the path to write all of the templates, diagnostic images and quality files
- *pathname3* is the path to the config file (optional)

and `[OPTIONS]` are any options listed in Table 3. By default, if no options are specified, the input image is simply preprocessed. More than one extraction option can be used to extract multiple feature types from an image (e.g. `–p` and `–m` can be used together to create a pore template and a minutia template per image). A space must exist between any flags used.

The command

```
java –jar nij-extractor.jar ./samples/ 2000 /output/ ./extractor.conf -a
```

entered on a single line would perform the following:

- create the directory `/output` if it did not already exist;
- open the config file located at `./extractor.conf` and set the appropriate parameters;
- extract all features from each `.bmp` and `.jpg` image located in `/samples`;
- write templates for pores, ridges, edges and minutia for each image to `/output`; and

- write quality files for pore and ridge contours to `/output`.

Short flag	Long flag	Token syntax	Action
`-p`	N/A	`-p`	Extracts pores and writes them to a template for each image
`-r`	N/A	`-r`	Extracts minutia-constrained ridge contour points and writes them to a template for each image
`-m`	N/A	`-m`	Extracts minutia and writes them to a template for each image
`-e`	N/A	`-e`	Extracts edgeoscopic features and writes them to a template for each image
`-a`	N/A	`-a`	Extracts all features and writes them to templates for each image

Table 3: Extractor command line options

Configuration Parameters

Each feature extraction process has a set of associated configuration parameters. Where possible, these parameters are defined on a per-dpi basis as to allow resolution-independent extraction. These dpi-dependent parameters can be overridden with specific values by adding the appropriate keys to the configuration file. **Appendix A: Feature Extraction Configuration** lists the configuration keys and contains a sample configuration file.

Supported Image Formats

The feature extractor supports grayscale and RGB images in either bitmap or jpeg formats. A directory of images may contain files of each format. If an image cannot be opened for processing, the image is skipped and the extractor attempts to load the next image. If you receive an error regarding heap space,[*] the input image is too large to be processed. To resolve this, increase the heap size by including the `-Xmx<max_heap_size>` argument. For example, the argument line

```
java -Xmx256m -jar nij-extractor.jar [args]
```

would increase the max heap size to 256MB.

Supported Image Resolutions

As all configuration parameters are set with respect to resolution in dpi, any image between 500dpi and 4000dpi can be processed using the feature extractor. However, all default configuration parameter values were set for maximum effectiveness with 2000 dpi images and some parameters did not scale up as precisely as expected in small-scale A3791 testing. To alleviate potential problems with resolutions greater than 2000dpi, the block sizes need to be increased to prevent inaccuracies in sliding window contrast adjustment and estimation of low ridge flow orientation. The Mexican hat variance parameter also needs to be altered as the current convolution operation results in an undesired increase in noise. With these changes the feature extraction process should be capable of producing templates fit for matching.

Processing Dependencies

The extraction process for several of the feature types require data produced during other extraction processes (e.g. to extract edgeoscopic features, the ridges must first be extracted). In these cases, if the verbose option is selected in the configuration file you may see these processing steps occur, but a

[*] Such an error generally appears as the exception:
`java.lang.OutOfMemoryError: Java heap space`.

template for the extraneous feature type(s) will not be saved. Likewise, diagnostic images (if permitted by the configuration file) may be saved for the additional processing steps as they were necessary to extract the desired features.

4 Performance Evaluator

The performance evaluator may be used to evaluate the matching accuracy. By default, the NBIS Bozorth application is used to perform the template comparisons, but any external executable that accepts the same inputs and command line syntax as Bozorth and provides output in the same format may be swapped in. The process flow is as follows:

1. From a directory of templates, an experiment is designed and a list of genuine and impostor comparisons is created.
2. An external matcher application executable produces a raw score for each comparison in the list.
3. The raw scores are analyzed and match rate histograms are created.

The main requirement for correct operation is that the template files are named according to a specific convention and contain data appropriate for the external matcher executable that is employed.

Performance Evaluator application	
Executable	`nij-evaluator.jar`

Dependencies	`nij-meta.jar`
	`ibg-mig.jar`
	`commons-logging.jar`
	`bailey.jar`
	`Jama-1.0.2.jar`

External Dependencies	`bozorth3` exectuable*

Table 4: Performance Evaluator - key application information

Application Processing

The performance evaluator first iterates through the template directory and parses individual file metadata according to the naming convention under which the files in the sample datasets are named. Each file must be named

```
01234-56789-RI-1-2000-tag.ext
```

where

- `01234` is the subject ID (and must be a nonnegative integer with or without leading zeros);
- `56789` is the visit ID (and must be a nonnegative integer with or without leading zeros);
- `RI` is the finger position and must be one of `RI`, `RM`, `LI`, or `LM`;
- `1` is the visit capture number, which must be a nonnegative integer and should not have leading zeros;
- `2000` is the image resolution and must be a nonnegative integer and should not have leading zeros;
- `tag` is any string; and
- `ext` (the filename extension) is any string.

The resolution, tag, and extension are not used for experiment design.

* Any executable may be swapped in for bozorth3, provided it accepts the same command line syntax as bozorth and creates the raw score file in the same output format. For more information, see the *User's Guide to NBIS*, distributed with NBIS.

By default, an experiment is designed to perform all possible genuine comparisons and a roughly equal number of impostor comparisons. The list of comparisons is written to a text file in the user's default temporary directory as alternating probe and gallery template filepaths. An example is in Figure 1.

```
/temps/01121-00019-RI-1-2000-A.xyt
/temps/01121-00128-RI-1-2000-A.xyt
/temps/00345-00007-RI-1-2000-A.xyt
/temps/73911-00011-RI-2-2000-A.xyt
…
```

Figure 1: Mates list file excerpt

In the mates list file, the first two lines define the probe and gallery templates for the first comparison, the next two lines define the probe and gallery templates for the next comparison, and so on.

The mates list is provided as input to the matcher executable, which performs the comparisons and writes raw scores to a text file in the format displayed in Figure 2.

```
45 /temps/01121-00019-RI-1-2000-A.xyt /temps/01121-00128-RI-1-2000-A.xyt
6 /temps/00345-00007-RI-1-2000-A.xyt /temps/73911-00011-RI-2-2000-A.xyt
…
```

Figure 2: Raw scores file excerpt

Each line of the raw scores file describes one comparison as a score, probe template pathname, and gallery template pathname (in that order), delimited by single spaces.

The raw scores file is then processed and a comma-separated values text file is created that describes each comparison on one line, and each line contains delimited metadata for both templates and the score. The tokens on each line of a processed score file are described in Table 5.

Column Index	Column Header	Description
0	p_filename	Probe template filename
1	p_subject_id	Probe subject ID
2	p_visit_id	Probe visit ID
3	p_position	Probe position
4	p_capture	Probe visit capture number
5	p_dpi	Probe resolution
6	p_tag	Probe tag
7	p_ext	Probe filename extension
8	g_filename	Gallery template filename
9	g_subject_id	Gallery subject ID
10	g_visit_id	Gallery visit ID
11	g_position	Gallery position
12	g_capture	Gallery visit capture number
13	g_dpi	Gallery resolution
14	g_tag	Gallery tag
15	g_ext	Gallery filename extension
16	is_genuine	1 if the comparison is genuine, 0 otherwise
17	is_same_subject_impostor	1 if the probe and gallery templates are from the same subject but different positions, 0 otherwise
18	is_impostor	1 if the comparison is an impostor comparison, 0 otherwise
19	score	The comparison score (may be an integer or floating point value)

Table 5: Processed score file columns

The content of the processed score file is then parsed to analyze the match rates. Histograms of the impostor and genuine scores are created with score bin parameters defined as follows:

- If the number of bins is not specified, the default (100) is used. Otherwise, the specified number of bins is used.
- If the bin parameters (minimum of all bins and the bin size) are not specified, the minimum is the minimum score and the bin size is the score range (maximum score minus minimum score) divided by the number of bins. Otherwise the specified bin parameters are used.

Those score distributions are used to compute the false match rate and false non-match rate histograms. The histograms are written as comma-separated values to the match rate analysis output file in the format of Figure 3.

Bin	gen_count	gen_freq	imp_count	imp_freq	FM_count	FMR	FNM_count	FNMR
Lesser	0	0	0	0	0	0	0	0
0	91	0.0075	133	0.00872	133	0.009	12129	1
4.06	1260	0.10388	1917	0.12563	2050	0.134	12038	0.992
8.12	1967	0.16216	3134	0.20539	5184	0.34	10778	0.889
12.18	1692	0.13949	2433	0.15945	7617	0.499	8811	0.726
16.24	1147	0.09456	1477	0.0968	9094	0.596	7119	0.587
20.3	767	0.06323	890	0.05833	9984	0.654	5972	0.492
… … …								
803.88	0	0	0	0	15258	1	0	0
807.94	0	0	0	0	15258	1	0	0
Greater	1	8.2E-05	1	6.6E-05	1	7E-05	1	8E-05

Figure 3: Match rate analysis output file

The match rate analysis file also contains two columns (not shown) that are the same as the FMR and FNMR columns but where every zero value has been replaced by 1e-5, to facilitate plotting the DET curves on log charts where zero values are invalid.

Execution Syntax

The command syntax is as follows:

```
java -jar nij-evaluator.jar [OPTIONS] pathname1 pathname2 pathname3
```

where:

- *pathname1* is the path to the template directory
- *pathname2* is the path to the matcher executable (e.g. bozorth3.exe)
- *pathname3* is the destination for the match rate analysis output file

and `[OPTIONS]` are any options listed in Table 6. Use of command line options is described in more detail in **Appendix C: Standardized Command Line Syntax**.

For example, the command

```
java -jar nij-evaluator.jar -vp 4 --mates-list /tmp/mateslist.txt
      /path/to/my_template_dir /usr/bin/bozorth3 /tmp/match_rates.csv
```

entered on a single line would perform the following:

- design an experiment using at most 4 finger positions from the templates in the directory `/path/to/my_template_dir`;
- send verbose messages to `stdout` during experiment design;
- write the list of comparisons to `/tmp/mateslist.txt`;
- compare templates using `/usr/bin/bozorth3`; and
- write the match rate analysis to `/tmp/match_rates.csv`.

Short flag	Long flag	Token syntax	Action
`-A`	`--a-errors`	`--a-errors` *`filepath`*	Writes errors encountered during match rate analysis stage to the specified file*
`-b`	`--bin-params`	`--bin-params` *`min size`*	Sets the score bin parameters used during match rate analysis. The minimum of all bins is set to *`min`* and the bin size is set to *`size`*. Tokens must be in format as required by the `Float.parseFloat` method in the Java™ standard library.
`-e`	`--boz-errors`	`--boz-errors` *`filepath`*	Writes matcher executable's error stream to the specified file*
`-c`	`--cross-visit`	N/A	Limits genuine comparison list to cross-visit comparisons only.
`-h`	`--help`	N/A	Displays help message and exits
`-i`	`--imp-comps`	`--imp-comps` *`num`*	Sets the target number of impostor comparisons per subject
`-M`	`--mates-list`	`--mates-list` *`filepath`*	Writes the list of comparisons to the specified file*
`-n`	`--num-bins`	`--num-bins` *`num`*	Sets the number of score bins used for match rate analysis (default is 100)
`-s`	`--num-subjects`	`--num-subjects` *`num`*	Limits the number of subjects used for the experiment to the specified value. If the number is less than the population size of the template directory, the subjects used for the experiment are chosen at random.
`-f`	`--positions`	`--positions` *`num`*	Limits the number of finger positions to be used in constructing the comparison list. The default is 1.
`-P`	`--proc-errors`	`--proc-errors` *`filepath`*	Writes errors encountered during the processing of raw scores to the specified file*
`-r`	`--raw-scores`	`--raw-scores` *`filepath`*	Writes the raw scores to the specified file*
`-v`	`--verbose`	N/A	Send verbose messages about experiment design to stdout
`-V`	`--vverbose`	N/A	Send debug messages about experiment design to stdout
`-p`	`--processed-scores`	`--processed-scores` *`filepath`*	Writes the processed scores to the specified file*

Table 6: Evaluator command line options

Known Bugs

The experiment designer has a few known bugs that result in the number of impostor comparisons not being exactly equal to the number of subjects multiplied by the target number of impostor comparisons per subject. When the comparisons are generated, any invalid comparisons (such as a same-subject, different position comparison) are caught and thrown out, and as a result, the final impostor comparison list is often a few comparisons short of the target total.

* If not specified, output is written to a file in the user's default temporary directory. This file's filename is displayed in the console output.

5 IQM Analyzer

The IQM application from the MITRE Corporation* was used in Level 3 matching performance evaluation to analyze image quality, and a Java™ interface was developed in order to facilitate the evaluation. The interface has only been tested with the Windows version of the IQM application, specifically with the executable iqm72cygwin.exe.

The IQM application requires a preferences text file and an auxiliary data text file ("auxdata file"), in addition to the image files themselves, for operation. The documentation distributed with the application contains more information on required input files.

IQM Analyzer application	
Executable	`iqm-analyzer.jar`
Dependencies	`ibg-mig.jar` `commons-logging.jar`
External Dependencies	`bozorth3` exectuable

Table 7: IQM Analyzer - key application information

Application Processing

The process flow is as follows:

1. From a directory of images, an auxdata file is created.
2. The preferences file is generated from parameters hard-coded into a class.
3. The native iqm72cygwin.exe process is executed, with the verbose output setting, from the JVM.
4. The output file is parsed, and a comma-separated values file is created where each line contains the image filepath and several quality measures. (This step is optional.)

The preferences and auxdata file generation process are designed only for 8-bit grayscale images in the formats accepted by IQM. See the IQM documentation for details on the IQM application process flow and valid input image formats.

Execution Syntax

The command syntax is as follows:

```
java -jar iqm-analyzer.jar [OPTIONS] pathname1 pathname2
```

where:

- *`pathname1`* is the path to the IQM executable (e.g. iqm72cygwin.exe)
- *`pathname2`* is the path to the image directory

and `[OPTIONS]` are any options listed in Table 8, Use of command line options is described in more detail in **Appendix C: Standardized Command Line Syntax**.

For example, the command

```
java -jar iqm-analyzer.jar -x bmp --parse --output C:\tmp\iqm_analysis
      C:\iqm\execs\iqm72cygwin.exe C:\path\to\my_image_dir
```

entered on a single line would perform the following:

* www.mitre.org

- create the directory `C:\tmp\iqm_analysis` if it does not already exist
- create an auxdata file listing all files in `C:\path\to\my_image_dir` that match `*.bmp`, along with the required auxdata parameters;
- create a default preferences file;
- execute `C:\iqm\execs\iqm72cygwin.exe`;
- parse the IQM_OUTPUT file created by IQM and write the quality data to the file `C:\tmp\iqm_analysis\iqm_output_records.csv`.

Short flag	Long flag	Token syntax	Action
`-c`	`--echo`	N/A	Echoes console output from the IQM application to `stdout`
`-x`	`--ext`	`--ext` *`extension`*	Restricts input image files to files with the filename extension matching extension
`-h`	`--help`	N/A	Display usage message and exit
`-o`	`--output`	`--output` *`pathname`*	Specifies the directory in which to write all output files. The directory will be created if it does not exist. If this option is not specified, then a directory is created in the user's default temporary directory.
`-p`	`--parse`	N/A	Parses the output file created by the IQM application and writes quality measures to a comma-separated values file named `iqm_output_records.csv` in the output directory
`-V`	`--vparse`	N/A	Causes verbose messages about the output parsing to be sent to `stdout`

Table 8: IQM Analyzer command line options

Notes about IQM Preferences

The preferences file that is generated by `iqm-analyzer.jar` is not identical to the `DefaultPrefs` file distributed by MITRE with the IQM application. The differences are noted in the source code for the class `ibg.image.iqm.IQMPreferences`.

Appendix A: Feature Extraction Configuration

Image Preprocessor configuration keys

Configuration Key	Description	Default
ip_verbose	Print image preprocessing progress data to the console	True
ip_diagnostics	Save image preprocessing diagnostics to the output directory	False
ip_minimumdiagnostics	Save only the most important image preprocessing diagnostics	False
ip_macroblocksize	Size of the blocks (in pixels) over which the ridge flow orientation is determined; overrides ip_macroblocksizeperdpi	30
ip_macroblocksizeperdpi	Size of the blocks (in pixels) over which the ridge flow orientation is determined (per DPI)	0.015
ip_gaussianvariance	Variance of the Gaussian kernel used to blur the image; overrides ip_gaussianvarianceperdpi	2.0
ip_gaussianvarianceperdpi	Variance of the Gaussian kernel used to blur the image (per DPI)	0.001
ip_uppercontrastthresh	The assumed ratio of undetermined valley pixels to total pixels in the image; this ratio determines the threshold at which pixels with a greater intensity are set to 255	0.85
ip_lowercontrastthresh	The assumed ratio of definitive ridge pixels to total pixels in the image; this ratio determines the threshold at which pixels with a lower intensity are set to 0	0.15

Pore Extractor configuration keys

Configuration Key	Description	Default
pe_verbose	Print pore extraction progress data to the console	True
pe_diagnostics	Save pore extraction diagnostics to the output directory	False
pe_minimumdiagnostics	Save only the most important pore extraction diagnostics	False
pe_minimumblobsize	Lower size bound on pore blobs to be detected; overrides pe_minimumblobsizeperdpi	0
pe_minimumblobsizeperdpi	Lower size bound on pore blobs to be detected (per DPI)	0.0
pe_maximumblobsize	Upper size bound on pore blobs to be detected; overrides pe_maximumblobsizeperdpi	90
pe_maximumblobsizeperdpi	Upper size bound on pore blobs to be detected (per DPI)	0.045
pe_mexhatvariance	Variance of the 2D Mexican hat wavelet kernel; overrides pe_mexhatvarianceperdpi	2.6
pe_mexhatvarianceperdpi	Variance of the 2D Mexican hat wavelet kernel (per DPI)	0.0013
pe_mexhatthreshold	Intensity threshold at which the Mexican hat processed image is thresholded	192
pe_border	Size of the border which is applied to the image to remove invalid data introduced by the various processing steps; overrides pe_borderperdpi	10
pe_borderperdpi	Size of the border which is applied to the image to remove invalid data introduced by the various processing steps (per DPI)	0.005

Ridge extraction configuration keys

Configuration Key	Description	Default
pe_verbose	Print ridge extraction progress data to the console	True
pe_diagnostics	Save ridge extraction diagnostics to the output directory	False
pe_minimumdiagnostics	Save only the most important ridge extraction diagnostics	False
re_microblocksize	Size of the blocks (in pixels) over which the ridge contour orientation is determined; overrides ip_microblocksizeperdpi	4
re_microblocksizeperdpi	Size of the blocks (in pixels) over which the ridge contour orientation is determined (per DPI)	0.002
re_minimumblobfillsize	Lower size bound on blobs to be filled; overrides pe_minimumblobfillsizeperdpi	0
re_minimumblobfillsizeperdpi	Lower size bound on blobs to be filled (per DPI)	0.0
re_maximumblobfillsize	Upper size bound on blobs to be filled; overrides pe_maximumblobfillsizeperdpi	250
re_maximumblobfillsizeperdpi	Upper size bound on blobs to be filled (per DPI)	0.125
re_threshold	Intensity threshold at which the preprocessed image is thresholded prior to blob detection	127
re_border	Size of the border which is applied to the image to remove invalid data introduced by the various processing steps; overrides re_borderperdpi	12
re_borderperdpi	Size of the border which is applied to the image to remove invalid data introduced by the various processing steps (per DPI)	0.006
re_pixelspacing	Controls the interval at which pixels are kept along each ridge contour; overrides re_pixelspacingperdpi	8
re_pixelspacingperdpi	Controls the interval at which pixels are kept along each ridge contour (per DPI)	0.004
re_suppressionradius	Radius around minutia; ridge contour pixels outside of this radius are removed; overrides re_suppressionradiusperdpi	30
re_suppressionradiusperdpi	Radius around minutia; ridge contour pixels outside of this radius are removed (per DPI)	0.015

Edgeoscopic feature extraction configuration keys

Configuration Key	Description	Default
ee_verbose	Print ridge extraction progress data to the console	True
ee_diagnostics	Save ridge extraction diagnostics to the output directory	False
ee_minimumdiagnostics	Save only the most important ridge extraction diagnostics	False
ee_pixelspread	Distance over which the orientation difference is calculated; overrides ee_pixelspreadperdpi	6
ee_pixelspreadperdpi	Distance over which the orientation difference is calculated (per DPI)	0.003
ee_numberofpoints	Quantity of edgeoscopic points to extract	200
ee_proximitythreshold	Maximum distance two edgeoscopic points may fall within each other; overrides ee_proximitythresholdperdpi	9
ee_proximitythresholdperdpi	Maximum distance two edgeoscopic points may fall within each other (per DPI)	0.0045

Sample Config File (`extractor.conf`)

```
#----------------------------------------
#Image preprocessing parameters
#----------------------------------------
ip_verbose = true
ip_diagnostics = true
ip_minimumdiagnostics = false

#ip_macroblocksize = 30
ip_macroblocksizeperdpi = 0.015

#ip_gaussianvariance = 2.0
ip_gaussianvarianceperdpi = 0.001

ip_uppercontrastthresh = 0.85
ip_lowercontrastthresh = 0.15


#----------------------------------------
#Pore extraction parameters
#----------------------------------------
pe_verbose = true
pe_diagnostics = true
pe_minimumdiagnostics = false

#pe_minimumblobsize = 0
pe_minimumblobsizeperdpi = 0.0

#pe_maximumblobsize = 90
pe_maximumblobsizeperdpi = 0.045

#pe_mexhatvariance = 2.6
pe_mexhatvarianceperdpi = 0.0013

pe_mexhatthreshold = 192

#pe_border = 10
pe_borderperdpi = 0.005

#----------------------------------------
#Ridge extraction parameters
#----------------------------------------
re_verbose = true
re_diagnostics = true
re_minimumdiagnostics = false

#re_microblocksize = 4
re_microblocksizeperdpi = 0.002

#re_minimumblobfillsize = 0
re_minimumblobfillsizeperdpi = 0.0

#re_maximumblobfillsize = 250
re_maximumblobfillsizeperdpi = 0.125

re_threshold = 127

#re_border = 12
re_borderperdpi = 0.006

#re_pixelspacing = 8
re_pixelspacingperdpi = 0.004
```

```
#re_suppressionradius = 30
re_suppressionradiusperdpi = 0.015

#----------------------------------------
#Edgeoscopic extraction parameters
#----------------------------------------
ee_verbose = true
ee_diagnostics = true
ee_minimumdiagnostics = false

#ee_pixelspread = 6
ee_pixelspreadperdpi = 0.003

ee_numberofpoints = 200

#ee_proximitythreshold = 9
ee_proximitythresholdperdpi = 0.0045
```

Appendix B: Software Libraries

In the development of Level 3 extraction and matching tools, several more general software libraries were produced in order to perform tasks not directly related to Level 3 extraction and matching. Such libraries are included in the software deliverable, and descriptions are below. Any dependencies are also listed.

The only external biometric software utilized for the project is the National Institute of Science and Technology Biometric Image Software (NBIS) collection, which is also described below. Other non-biometric external software libraries are employed. All software libraries and applications are either

(a) in the public domain;
(b) distributed under an open source license; or
(c) distributed for non-commercial use.

Relevant license information for the latter is included in the descriptions below, and where applicable, the software packages themselves contain copies of the licenses.

IBG MIG Libraries

The IBG Math, Image & General Libraries (MIG) contain classes dedicated to various mathematical, image processing, or other general operations. The full API documentation for these libraries is included in API Documentation appendix. MIG is contributed to the public domain along with the Level 3 software deliverable. Table 9 contains descriptions of the constituent packages.

Package	Description	Dependencies
`ibg.math`	Classes that support common algebraic, geometric, and general mathematical operations. The core of this package is its support for floating-point vectors implemented as matrices with a row or column dimension of one, and thus compatible with matrix operations as supported by the JAMA package (see note on dependencies). Although the general `ColumnVector` and `RowVector` classes are sufficient to perform linear algebra operations for any vector, most users of this package will probably find the `Point2D` or `Point3D` extensions more convenient. Other classes are provided to facilitate file I/O related to sets or arrays of vectors, and others are dedicated to facilitating printing matrices to standard output.	**`ibg.util`** JAMA
`ibg.math.stat`	Classes that facilitate common statistics operations, such as computation of the mean, median, quartile values, variance, etc. Also includes support for histogram creation.	(none)
`ibg.math.graph`	Classes that facilitate common vertex-edge graph operations.	Structures

Package	Description	Dependencies
ibg.math.plot	Classes that facilitate simple plotting in an arbitrarily sized 2D coordinate plane. All viewable components of plotted data are objects that can be made visible or invisible. This includes data sets as well as the administrative components of a plot (the axes, grid, labels, and background area). Aesthetic settings such as color and point appearance can be automatically generated or manually configured.	**ibg.math**
ibg.math.nonrigid **ibg.math.reg**	Classes that implement the iterative closest point algorithm with non-rigid transformations as described in "A new point matching algorithm for non-rigid registration" by Anand Rangarajan and Haili Chui. The source code for this package has been adapted from the Matlab® source code released by the authors under the GNU GPLv2.	**ibg.util** **ibg.math**
ibg.image **ibg.image.io** **ibg.image.conv** **ibg.image.filter**	Classes that facilitate common image processing operations and support the more narrowly focused subpackages.	(none)
ibg.util	Classes that support a variety of operations, including file I/O, external binary execution, and command line parsing. This package forms a base library on which the other **ibg.*packagename*** packages rely.	(none)

Table 9: IBG MIG Packages

JAMA

JAMA is a JAva MAtrices package developed by NIST and The MathWorks (www.mathworks.com). From the JAMA web page:

> JAMA is a basic linear algebra package for Java. It provides user-level classes for constructing and manipulating real, dense matrices. It is meant to provide sufficient functionality for routine problems, packaged in a way that is natural and understandable to non-experts.

The `Matrix` class in the JAMA package serves as the superclass for many classes in **ibg.math**.

The JAMA package has been released to the public domain.

JLAPACK

From the `f2j` project site: "The goal of the Fortran-to-Java project is to provide Java Application Programming Interfaces (APIs) to numerical libraries originally written in Fortran (particularly BLAS and LAPACK)."

JLAPACK is the collection Java packages providing support for various linear algebra operations. The classes contained in the packages have been automatically generated by the `f2j` application.

The license under which JLAPACK is distributed is unclear. The `f2j` source code is distributed with the GNU General Public License, version 2, with the addition of the BSD advertising clause. However, the project web site states that the project uses a BSD License, but does not specify which BSD License or if merely a BSD-style license is implied. In addition, the JLAPACK binary downloads are transmitted with no

license attached, and it is not clear that binaries compiled from source produced from the `f2j` Fortran translator would fall under the same license.

Apache Commons Logging

Apache Commons Logging is a logging interface for Java. From the Apache Commons Logging web site:

> The Logging package is an ultra-thin bridge between different logging implementations. A library that uses the commons-logging API can be used with any logging implementation at runtime. Commons-logging comes with support for a number of popular logging implementations, and writing adapters for others is a reasonably simple task.

The binaries are distributed under the Apache License, version 2.0. A copy of the license has been included with the software.

NIST Biometric Image Software

NIST Biometric Image Software (NBIS) is a collection of fingerprint software tools developedy by NIST for the Federal Bureau of Investigation and Department of Homeland Security.

The current implementation of Level 3 feature matching involves the utilization and manipulation of Level 2 (minutia) extraction and matching results. NBIS is used for all Level 2-related image and data processing. Although more refined commercial fingerprint software for processing Level 2 features exists, NBIS, the entirety of which is in the public domain, was chosen to ensure that users of the software developed for Level 3 extraction and matching may obtain its dependencies freely and with minimal restrictions.

From the NBIS User's Guide (reformatted):

> The NBIS software is organized in two categories: nonexport controlled and export controlled. The non-export controlled NBIS software is organized into five major packages:
>
> 1. `PCASYS` is a neural network based fingerprint pattern classification system;
> 2. `MINDTCT` is a fingerprint minutiae detector;
> 3. `NFIQ` is a neural network based fingerprint image quality algorithm,
> 4. `AN2K` is a reference implementation of the ANSI/NISTITL 1-2000 "Data Format for the Interchange of Fingerprint, Facial, Scar Mark & Tattoo (SMT) Information" standard; and
> 5. `IMGTOOLS` is a collection of image utilities, including encoders and decoders for Baseline and Lossless JPEG and the FBI's WSQ specification.
>
> The export controlled NBIS software is organized into two major packages:
>
> 6. `NFSEG` is a fingerprint segmentation system useful for segmenting four-finger plain impressions,
> 7. `BOZORTH3` is a minutiae-based fingerprint matching system.

The Level 3 software developed for this project depends on both the non-export- and export-controlled packages. All NBIS packages have been released to the public domain.

Structures

The Structures package contains classes implementing various data structures. The software was originally developed for a textbook, *Data Structures in Java, for the Principled Programmer*, written by Duane A. Bailey and published by McGraw-Hill Professional, 2002. The text of the book and the associated software have been released to the public for educational use. Duane A. Bailey is a professor of computer science at Williams College in Williamstown, MA. The text of the book and the software are available on Professor Bailey's home page.[*] The license conditions under which the software is released are unclear, but the author's intent is presumed to be to restrict use to educational and non-commercial purposes.

[*] www.cs.williams.edu/~bailey/JavaStructures

Appendix C: Standardized Command Line Syntax

The `ibg.util.CommandLineParser` class is employed in some applications to perform parsing of the command line. The processing performed by the relevant class methods is described below.

The `process` method is called to parse the command line (which is passed as the `args[]` parameter of the `main` method). Parsing modifies the contents of each individual `CommandLineOption` object to reflect the command line arguments. If an option's keyword (or key character) was present in the command line argument, that `CommandLineOption` object is marked as *flagged*. If that option was defined to require arguments, then the associated arguments are stored and may be retrieved with the `getArguments` instance method in the `CommandLineOption` class.

After all options and their individually required arguments have been processed, there may be arguments left over that are not associated with any specific option. These are known as "leftovers", and they are stored in a `java.util.List`. For some applications, it may be redundant to require an option keyword and an argument for the option. For example, an application may require only a single argument, always require it, and not have any other command line options defined. In those cases, the `CommandLineParser` should be constructed with the `numRequiredLeftovers` parameter equal to one.

To review, the parser does the following:

1. consumes all long flags and associated arguments
2. consumes all short flags and associated arguments
3. consumes all leftover arguments

Short (single character) and long (string literal) flags are processed differently. The short flags should be entered first on the command line and be all in a group, with only a single hyphen at the beginning. For example, the command line

```
java MyApp -xvf --long-flag arg1 arg2 arg3 arg4
```

contains the short flags `x`, `v`, and `f`.

Long flags can go anywhere and are processed first. Arguments associated with a particular long flag must immediately postcede the flag. In the example above, if `--long-flag` is defined to require a single argument, then `arg1` is the argument that will be associated with it.

After the long flags are processed, the short flags are processed. If a short flag requires an argument, the first argument on the line (that is not another flag or an argument that was consumed by a long flag) is associated with it. For multiple short flags, the first flag will eat the first unassociated argument (or arguments, depending on how many it requires), the second flag will eat the next, and so on. In the example above, if `--long-flag` is defined to require a single argument and `-x` and `-f` are each defined to require a single argument (while `-v` takes no arguments), then `--long-flag` will consume `arg1`, `-x` will consume `arg2`, and `-f` will consume `arg3`.

Once the short and long flags and their associated arguments have been processed, the leftover arguments are stored in the leftovers list. In the example above, where `--long-flag`, `-x` and `-f` each consume a single argument, `arg4` is the argument leftover after all options have been processed.