User Manual

Lipi Core Toolkit

lipi-core-toolkit 4.0

Contents

1 Lipi core toolkit: Overview

Lipi core toolkit is a generic toolkit that provides handwriting recognizers for several scripts and facilitates development of online handwriting recognizers for new scripts. It provides sample applications to simplify integration of the resulting recognizers into real-world application contexts. The toolkit provides robust implementations of tools, algorithms, scripts and sample code necessary to support the activities of training and evaluation of recognizers, creating recognizers for new scripts and their integration into pen-based applications. The toolkit is designed to be extended with new tools and algorithms to meet the requirements of specific scripts and applications. The toolkit attempts to satisfy the requirements of a diverse set of users, such as researchers, commercial technology providers, do-it-yourself enthusiasts and application developers. This work has been published in [1].

Introduction

There are large parts of the world characterized by the extensive use of paper and handwriting in all facets of society, and poor penetration of traditional PCs and keyboards. In India the penetration of PC is very less, as compared to the US, and most Western European countries. In this setting, products and solutions with pen, touch and/or paper-based interfaces may play an important role in making the benefits of Information Technology more pervasive. Handwriting recognition [*HWR*] is an important technology to research on appropriate user interfaces, and to create innovative products, solutions and services for these markets.

Unfortunately, for many of the languages in these parts of the world, such as the Indic languages, no commercial handwriting recognition technology exists. The central problem being addressed in this toolkit is: how can we simplify the creation of HWR technology for a new script, and how can we simplify its integration into real-world applications? This problem has been addressed by sister language technology communities working on speech recognition, speech synthesis, and machine translation through the creation of toolkits comprised of tools and algorithms that can be used to create language technology for a new language [2,3]. The issue of integration has been addressed by the creation of standard interfaces/protocols such as MRCP for speech recognition engines [4]. However, to the best of our knowledge, no generic toolkits or standards exist for online handwriting recognition.

1-1 The Lipi core toolkit

There are many different challenges involved in developing a generic toolkit for online handwriting recognition. The first is that the toolkit should provide generic components to perform reasonably well on simple as well as complex scripts, while providing the flexibility to tune, extend or even replace them with more suitable components, to meet the challenge of a particular script or application. A second challenge is to balance the needs of different classes of potential users.

For researchers in handwriting recognition, the toolkit should serve as a test bed to experiment with new algorithms. For a certain class of do-it-yourself enthusiasts, it should allow the creation of recognizers for new shapes and scripts out of the box, without requiring much knowledge of the algorithms. For a potential vendor interested in building commercial HWR engines, it should support the building of robust recognizers for new scripts. Finally, for the application developer, it should allow easy integration of the recognizers built using the toolkit into any pen-based application.

The Lipi core toolkit ("lipi" being Sanskrit for "script") is an effort to create a generic toolkit whose components can be used to build an online Hand Writing Recognizer for a new script, while addressing the challenges described. The following figure (Figure 1) describes the various categories of potential users of the toolkit.

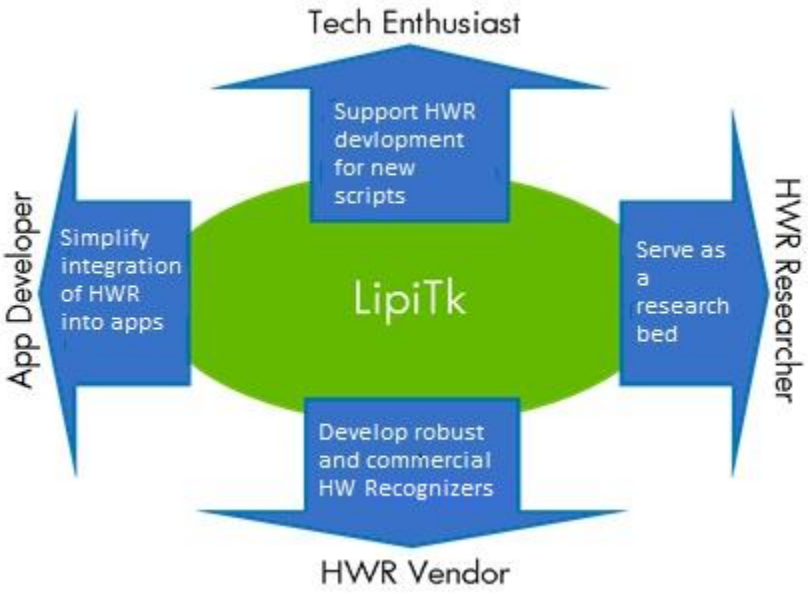


Figure 1: Lipi core toolkit users

1-2 Lipi core toolkit: Salient Features

While toolkits such as Sphinx from Carnegie Mellon University [2] and Festival from University of Edinburgh [3] exist for problems such as Automatic Speech Recognition and Speech Synthesis respectively, we believe Lipi core toolkit to be one of the first to address the problem of online HWR. Open source implementations of online gesture and character recognition such as Rosetta [5], XStroke [6] and WayV [7] are not primarily intended for experimentation with HWR algorithms, which is one of the (many) core goals of the toolkit.

Lipi core toolkit is designed to support a data-driven methodology for the creation of recognizers for new scripts. While many handwriting recognition algorithms are script specific, the Lipi core toolkit is intended to provide robust implementations of generic features and classifiers that are expected to perform reasonably well on any given set of symbols by learning the statistical shape properties of that set. This allows the researcher or enthusiast to build a reasonably performing recognition engine with minimum effort.

One of the key differences between handwriting recognition and (say) speech recognition is that no single approach or set of features/classification algorithms is known to work optimally for all scripts. Also, the nature and quality of the input (digital ink) tends to vary widely with the capture device. The core toolkit has been designed to accommodate different tools and algorithms specific to the device, script and/or application.

lipi-core-toolkit uses open standards such as UNIPEN [8] for the representation of digital ink and facilitating the creation of shareable linguistic resources within the community. Future versions may use W3C InkML [9] and UPX [10] for digital ink and annotation respectively.

There is a focus on robust and efficient implementation of algorithms in *lipi-core-toolkit*, in order to facilitate the integration of recognizers created using the toolkit into real-world applications.

1-3 Lipi core toolkit: Architecture

The Lipi core toolkit is designed to support Windows and Linux platforms, hence its design and implementation considers portability related issues. The components of the core toolkit are implemented using C++ & STL, using ANSI functions to address portability issues. Some of the utilities are written in Perl.

lipi-core-toolkit provides implementation for

- Preprocessing algorithms
- Feature extraction algorithms
- Shape recognition algorithms
- Word recognition algorithm

All the above listed modules are implemented as separate shared libraries that can be loaded at runtime

The figure below shows the Lipi core toolkit architecture.

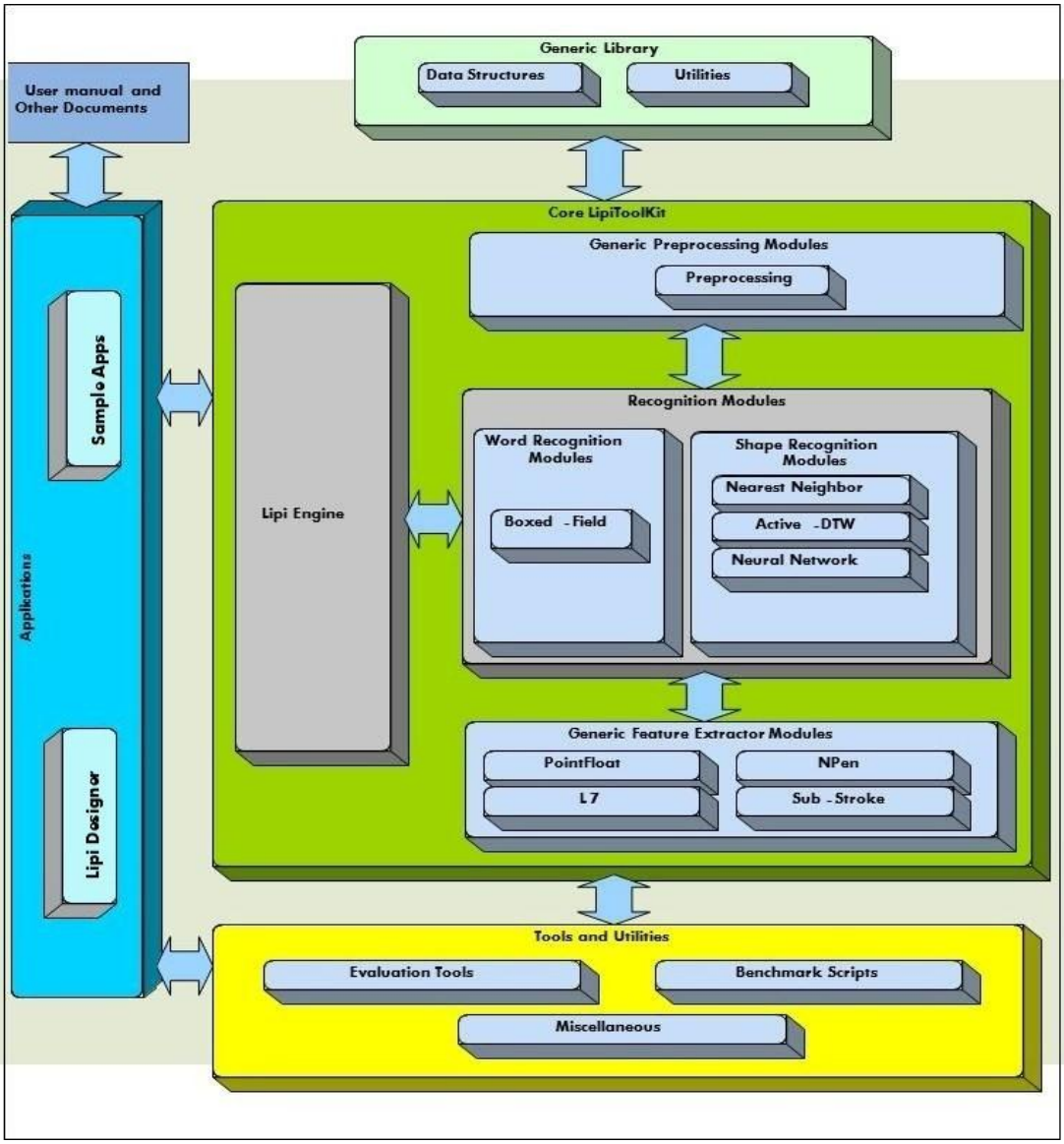


Figure 2 : Lipi core toolkit: Architecture and Components

The following sections describe each module in detail.

1-3-1 Generic classes and utilities library

The generic class library includes classes to store and manipulate ink traces, such as Trace and TraceGroup, and classes to store device and screen context. These classes are used by different modules and tool implementations. The design of these classes reflects a tradeoff between a conceptually intuitive and object-oriented data model, and efficient access to frequently accessed attributes, such as X and Y channels in the case of ink traces.

The utilities library provides utility functions to read *lipi-core-toolkit* configuration files, read and write UNIPEN data files, and so on.

1-3-2 Generic preprocessing

Lipi core toolkit exposes a standard set of interfaces for preprocessing the input ink. The default preprocessor bundled with *lipi-core-toolkit 4.0.0* is

- LTKPreprocessor

LTKPreprocessor module provides implementation for commonly used shape/character preprocessing operations such as

- moving-average smoothing
- size normalization
- equidistant resampling

The preprocessing sequence and attributes corresponding to each preprocessing operation can be specified in the shape recognizer's configuration file.

1-3-3 Generic feature extractor

Lipi core toolkit exposes a standard set of interfaces for all the feature extractor modules. This allows the user to dynamically configure and use any feature extractor at run-time. The feature extractors bundled with *lipi-core-toolkit 4.0.0* are

- PointFloatShapeFeatureExtractor
- NPenShapeFeatureExtractor
- SubStrokeShapeFeatureExtractor
- L7ShapeFeatureExtractor

1-3-4 Recognition modules

1-3-4-1 Shape recognition

The shape recognition algorithms bundled with *lipi-core-toolkit 4.0.0* are

- Nearest-Neighbor classifier
- ActiveDTW classifier
- Neural Network classifier

1-3-4-2 Word recognition

The word recognition algorithm bundled with *lipi-core-toolkit 4.0.0 is*

- Boxed-Field recognizer

Boxed-Field recognizer

The Boxed-Field recognizer recognizes a boxed field of characters by invoking a trained shape recognizer on each of the boxes. It employs Dynamic Programming for decoding the N-best strings based on the cumulative shape recognition confidences.

Significantly, the Boxed-Field recognizer exposes a generic word recognition API, allowing the possibility of plugging in a connected word recognizer in the future in a backward-compatible manner.

1-3-5 Tools and utilities

lipi-core-toolkit 4.0 provides the following tools and utilities

- listfiles.pl - creating train/test list files
- runshaperec - training and testing the shape recognizer
- runwordrec - testing the word recognizer
- eval.pl - evaluation and error analysis tool for the shape recognizer
- createAddOn.pl - packaging the configuration files and model files corresponding to the trained recognizers
- installAddOn.pl – installing the created recognizer

1-3-6 Lipi engine

The Lipi engine is the controller responsible for loading one or more shape/word recognition modules as specified in the configuration file.

1-4 Summary

In summary, the Lipi core toolkit aims to facilitate development of online handwriting recognizers for new scripts, and simplify integration of the resulting recognizers into real-world application contexts. The current version of the core toolkit provides robust implementations of tools, algorithms, scripts and sample code necessary to support the entire process starting from the creation of a recognition engine, to its deployment and integration.

The design of the core toolkit makes it possible to integrate new tools and algorithms (such as a different type of preprocessing, feature extraction or classification algorithm) into the toolkit.

Given the need to support the Lipi core toolkit user community (whether researchers or application developers) across multiple operating systems and computing platforms, the core toolkit is designed to simplify creation of versions for different platforms using a common code base.

As already indicated, there are several important research directions for the toolkit, including inclusion of other discriminative classification algorithms, native support for emerging standards such as W3C Ink Markup Language, data, annotation and error analysis, and even potential extensions to Offline HWR. However, our major focus at present is to validate the design and utility of the toolkit with different sets of users. We are also interested in collaborative projects with university research groups using the toolkit. We hope that some of these users can contribute by trying to use the toolkit and providing feedback, while others may contribute to the toolkit by way of new tools and algorithms.

2 Components and usage

2-1 Before you get started

This chapter lists the prerequisites for installing and using Lipi core toolkit.

2-2 Supported platforms and environment

lipi-core-toolkit 4.0 has been tested on the following platforms:

- Windows 7 32 bit and 64 bit
- Ubuntu 10.10 32 bit and 64 bit

2-3 Disk space requirements

lipi-core-toolkit 4.0 provides source and binary packages for Windows and Linux. Separate source packages are provided for Windows and Linux.

The space required to extract the source package is 20 MB and binary package is 25 MB. To build the source package after extracting you need 130 MB of free space.

2-4 Software requirements

Item and Description	Windows 7	Linux
Building lipi-core-toolkit code	Microsoft Visual C++ 2008 / MsBuild for VC2008	G++ 4.4 or above
Executing scripts	Perl 5.1 or above and Archive::Zip	Perl 5.1 or above and Archive::Zip

Table 1: Software requirements

NOTE: Archive::Zip perl module can be obtained from

http://search.cpan.org/~adamk/Archive-Zip-1.30/lib/Archive/Zip.pm

3 Installing packages

3-1 lipi-core-toolkit 4.0.0 packages

lipi-core-toolkit 4.0 is available in the form of binary and source packages for 32 bit and 64 bit Windows and Linux platforms.

Platform	Package
Windows 7	**Binary:** lipi-core-toolkit4.0.0-bin-x86.exe lipi-core-toolkit4.0.0-bin-x64.exe **Source:** lipi-core-toolkit4.0.0-src-win-x86.exe lipi-core-toolkit4.0.0-src-win-x64.exe
Linux	**Binary:** lipi-core-toolkit4.0.0-linux-i686.tar.gz lipi-core-toolkit4.0.0-linux-x86_64.tar.gz **Source:** lipi-core-toolkit4.0.0-src-linux.tar.gz

Table 2: lipi-core-toolkit packages

3-2 Setting up the environment

Lipi-core-toolkit uses an environment variable called LIPI_ROOT as reference for locating the shared and dynamic library files, root location of data etc. On windows platform the LIPI_ROOT variable is automatically set on installation of the core tool kit. On Linux platform this variable has to be set manually.

For example, if the package was extracted to the directory '`\opt\hp\`', LIPI_ROOT should be set to "`\opt\hp\lipi-toolkit`".

Refer to the **Appendix** for instructions for setting environment variables.

Hereafter, we use $LIPI_ROOT to refer to the value of this environment variable.

NOTE: In order to support multiple installations of the *lipi-core-toolkit* on the same system, an option to specify LIPI_ROOT through command line has also been provided for all the components of the toolkit. This option can also be used to specify LIPI_ROOT in a single installation scenario, in that case the option specified through command line is given precedence over the value specified through an environment variable.

⚠ **CAUTION:** There should be no blank spaces in the path specified by LIPI_ROOT

---

4 Building source package

4-1 Building on Windows for VC2008

4-1-1 Setting up the Msbuild environment

To build the binaries on Windows for VC2008, devenv must be included in system PATH variable. This can be done by executing `<visual studio 2008 install dir>\ Common7\Tools\ vsvars32.bat` from the command prompt for each shell/command prompt.

4-1-2 Building the lipi-core-toolkit components

lipi-core-toolkit 4.0 provides the build files for all modules under directory windows/vc2008 at each directory level. The master build file is present at <lipi-core-toolkit directory>/windows/vc2008/lipitk.targets. A Module can be prevented from building by removing the module name from the DefaultTargets: list in the master project file. With these project files, user can build source code for VC2008 package by using the MsBuild.

To build lipi-core-toolkit 4.0, on Windows for VC2008, execute the following command from `<lipi-core-toolkit install directory>/windows/vc2008`

```
MsBuild lipitk.targets
```

4-2 Building on Linux

4-2-1 Building the lipi-core-toolkit components

On Linux, the makefiles for each module are available under the directory `linux/` with name `Makefile.linux`. The master makefile is present at `<lipi-core-toolkit directory>/linux/Makefile.linux`. To prevent a module from building; the **.PHONY** tag in the master makefile can be modified.

A global configuration file, `<lipi-core-toolkit install directory>/global.mk`, contains common platform-specific settings such as compiler, environment and linker etc. Refer to the **Appendix**, for the configurable makefile settings.

To build *lipi-core-toolkit 4.0*, on Linux, execute the following command from `<lipi-core-toolkit install directory>/linux`

```
make -f Makefile.linux
```

5 Shape recognizer project configuration

Project

Project is a logical name given to a group of recognizer configurations specific to a particular script or a set of shapes to be recognized.

project.cfg

All the project directories should be present under `<lipi-core-toolkit install directory>/projects`. For every project, the project specific settings are stored in a configuration file named `project.cfg`.

For example, consider a project called **demonumerals**. The project root (**PROJROOT**) would be `<lipi-core-toolkit install directory>/projects/demonumerals`.

The project configuration file, `project.cfg`, would be present under the directory `<lipi-core-toolkit install directory>/projects/numerals/config/`.

Typically, a `project.cfg` contains the following attributes

ProjectType = SHAPEREC

NumShapes = 10

S No.	Key	Possible values	Description
1	ProjectType	• SHAPEREC • WORDREC	Describes the type of the project. For a shape recognition project, user must specify the **ProjectType** as **SHAPEREC**, whereas it should be set to **WORDREC**, for a word recognition project.
2	NumShapes	• Dynamic • Any positive integer value specifying the number of shapes	Number of distinct shapes in the shape set to be recognized. The user can set the **NumShapes** to **Dynamic,** if the number of distinct shapes to be recognized is not known in advance. In this case, the number of shapes is treated as a variable and no checks are performed in the project to verify this value. However, if the exact number of distinct shapes to be recognized is known in advance, the **NumShapes** can be set to the exact value.

Table 3: Attributes in project.cfg

6 Profile configuration

Profile

Profile is a set of configuration files related to a particular project. There can be one or more profiles for the same project with a different set of tunable parameters of the algorithm used for shape recognition.

IMPORTANT: If profile name is not specified for a project, 'default' profile is chosen automatically.

profile.cfg

The different profiles are stored as subdirectories under `<lipi-core-toolkit install directory>/projects/<project_name>/config`. Each profile directory must contain the `profile.cfg` file.

The default profile is present under `<lipi-core-toolkit install directory>/projects/<project_name>/config/default`.

The `profile.cfg`, for a **shape recognition** project, has the following configurable attribute:

ShapeRecMethod = nn

Where, **nn** is the name of the shape recognition module to be used for the project-profile combination. In addition to the `profile.cfg`, each profile directory must also contain the configuration file corresponding to the shape recognition algorithm specified in `profile.cfg`. In this case it would be nn.cfg. Please refer to **Appendix** for default configuration file for nn. Other shape recognition modules that can be specified in `profile.cfg` are activedtw and neuralnet. Please refer to **Appendix** for default configuration file for activedtw and neuralnet.

For a **word recognition** project, the `profile.cfg` typically has the following attributes:

WordRecognizer = boxfld

RequiredProjects = numerals (default)

S No.	Key	Description
1	WordRecognizer	The word recognition module to be used for the project.
2	RequiredProjects	The shape recognition project required by the word recognizer. In above example, numerals(default) specifies the project (numerals) and profile (default)

Table 4: Attributes in profile.cfg for a word recognition project

NOTE: The recognition module name in the `profile.cfg`, for a shape recognition project, has to be the same as the directory name under `<lipi-core-toolkit install directory>/src/reco/shaperec`, e.g. nn.

7 lipiengine configuration

lipiengine

lipiengine is the controller that loads all the modules (logger, preprocessor, featureextractor, recognizer) required for a particular project configuration. The lipiengine also sets the logfile name and the loglevel for the current project.

lipiengine.cfg

The configuration parameters for lipiengine can be specified through this configuration file. lipiengine.cfg must be stored in `<lipi-core-toolkit install directory>/projects directory`. Typically, lipiengine.cfg contains the following attributes:

```
LogFile=project_lipi.log
LogLevel=DEBUG
KANNADA_CHAR = kannada_char(default)
```

S No.	Key	Possible values	Description
1	LogFile	Example: project_lipi.log	Define the name of the log file. LogFile parameter allows user to specify the name of the log file. By default, the log file is created in the current directory. If the log file already exists, the log messages are appended to the file. If the log file is not specified, the log messages are written to the default log file, `lipi.log`, in the current directory.
2	LogLevel	• *DEBUG* • *ERR* • *INFO* • *ALL* • *VERBOSE* • *OFF*	Define the type of the Log level. User can control the logging levels in Lipi by specifying it as a parameter. NOTE: Default log level is ERR if not specified at the command line.
3	Logical name	Example: *KANNADA_CHAR = kannada_char(default)* *NUMERALS = demonumerals(default)*	Logical name is used to encode a project-profile combination. These logical names are used by the Sample client applications to refer to the project-profile combinations. If the profile is not given, **default** profile is assumed.

Table 5: Attributes in lipiengine.cfg for a recognition project

8 Training and testing

8-1 runshaperec tool

The `runshaperec` tool is an executable found under `<lipi-core-toolkit install directory>/bin`, used for training and testing the shape recognizer. (Note: If the source pacakage was downloaded, then one has to build the source before finding runshaperec executable. Please refer to Building source Package section for further details.)

Training the shape recognizer results in the creation of a model data file under `<lipi-core-toolkit install directory>/projects/<project>/config/<profile>/`. In the case of testing, the tool stores the recognition results into the specified text file, which could be used with the Evaluation tool to analyze the recognition performance.

Usage: runshaperec

```
runshaperec
            -project          <project name>
            -train OR -test   <path of list file>
            OR -adapt
            [-h]              <model data header info file name>
            [-lipiroot]       <path of lipi-core-toolkit install
                              directory>
            [-profile]        <profile name>
            [-logfile]        <log file name>
            [-loglevel]       <log level>
            [-output]         <recognition results filename>
            [-confthreshold]  <threshold on confidence>
            [-numchoices]     <number of recognition choices>
            [-infiletype]     <feature|default ink>
            [-perf]
            [-ver]
            [-help]
```

Command line arguments

Command line argument	Argument type	Description
-project <project name>	Mandatory	For training or testing, the user needs to specify the project name.

		Example: -project numerals NOTE: lipi-core-toolkit searches for the directory having the name as that of the project under `<lipi-core-toolkit install directory>/projects`.
- train|test |adapt <path of list file>	Mandatory	Path of the input file should be passed. The Perl script listfiles.pl can be used to generate the list files, and validatelistfile.pl can be used to validate the list file. NOTE: Sample list file for train/test is mention in Appendix. Sample list file for adapt is mention in Appendix
-h <model data header info filename>	Optional	During training, the user can also pass an optional argument model data header information file name.
-lipiroot <path of lipi-core-toolkit install directory>	Optional	User can specify the path of lipi-core-toolkit install directory using this argument. All the dynamic libraries are retrieved from the `lib` directory under the path specified. NOTE: If lipiroot is not specified as a command line argument, its value is retrieved from the environment variable LIPI_ROOT.
-profile <profile name>	Optional	This argument allows user to specify the profile to be used for the project. NOTE: If the profile name is omitted, default profile is assumed.
-loglevel <loglevel>	Optional	User can control the logging levels in Lipi by specifying it as a command-line argument. Following log levels can be used • DEBUG • INFO • ERR • ALL • OFF NOTE: Default log level is ERR if not specified at the command line.
-logfile <logfile>	Optional	This argument allows user to specify the name of the log file. By default, the log file is created in the current directory. However, user can control the location of the log file by specifying the absolute path. If the log file already exists, the log messages are appended to the file.

		NOTE: If the log file is not specified, the log messages are written to the default log file, `lipi.log`, in the current directory.
-output <output filename>	Optional	During testing, this argument can be used to specify the output recognition results file. NOTE: If this argument is not specified, the tool generates the default output file, `runshaperec.out`, under the current directory.
-numchoices <numchoices>	Optional	In the case of testing, the user can specify the maximum number of recognition choices to be returned by the recognizer. NOTE: If this argument is not specified, all the choices and confidence values computed by the recognize method will be written to the output file.
-confthreshold <confthreshold>	Optional	In the case of testing, the user can specify the threshold on the confidence value. The choices with confidence values greater than or equal to the threshold are written to the output file. NOTE: If this argument is not specified, all the choices and confidence values computed by the recognize method will be written to the output file.
-infiletype	Optional	During training, this argument can be used to specify the type as feature or ink. If this argument is specified as feature, then specify –train as feature file path generated by using featurefilewriter tool. If this argument is specified as ink, then specify –train as train list file. NOTE: Default infile type is ink if not specified at the command line.
-perf	Optional	To find out the total time taken for training or testing, -perf can be used. Specifying this argument displays the time taken in seconds, at the end of program execution.
-ver	Optional	Displays the version number of the program.
-help	Optional	Displays usage information.

Table 6: runshaperec command line arguments

8-2 runwordrec tool

The `runwordrec` is an executable found under `<lipi-core-toolkit install directory>/bin`, which is used for testing the boxfield recognizer module. If source pacakage is downloaded, then build the source before finding runwordrec executable. Please refer to Building source. The recognition results are written as Unicode strings in the output file specified at the command line. Please see the **Appendix** for a sample ink file for word recognition.

NOTE: There is no training for runwordrec.

Usage: runwordrec

```
runwordrec
            -project          <project name>
            -test             <path of list file>
            [-lipiroot]       <path of lipi-core-toolkit
                              install directory>
            [-profile]        <profile name>
            [-logfile]        <log file name>
            [-loglevel]       <log level>
            [-output]         <output Unicode results
                              filename>
            [-numchoices]     <number of recognition choices>
            [-perf]
            [-ver]
            [-help]
```

Command line arguments

Command line argument	Argument type	Description
-project <project name>	Mandatory	Specifies the project name. Example: -project numfld NOTE: lipi-core-toolkit searches for the directory having the name as that of the project under `<lipi-core-toolkit install directory>/projects`.
-test < path of list file>	Mandatory	Specify the path of the list-file to be used for testing the boxed field recognizer. The list-file contains the filenames of word samples that are annotated at the character level and in UNIPEN format. NOTE: The truth specified besides each file mentioned in the list file can be a dummy string as this is not considered in the current version of the

		toolkit.
-profile <profile name>	Optional	This argument allows user to specify the profile to be used for the project. If the profile name is omitted, default profile is assumed.
-lipiroot<path of lipi-core-toolkit install directory>	Optional	This argument enables the user to specify the path of lipi-core-toolkit root directory. All the dynamic libraries are retrieved from the `lib` directory under the path specified. NOTE: If lipiroot is not specified as a command line argument, it's value is retrieved from the environment variable LIPI_ROOT
-loglevel <loglevel>	Optional	User can control the lipi logging levels, by specifying the log level as a command line argument. Following log levels can be used • DEBUG • INFO • ERR • ALL • OFF NOTE: Default log level is ERR if it is not specified at the command line.
-logfile <logfilename>	Optional	This argument allows user to specify the name of the log file. The log file is created in the current directory. However, user can control the location of the log file by specifying the absolute path. If the log file already exists, lipi-core-toolkit log messages are appended to the same file. NOTE: If log file is not specified, the log messages are written to the default log file: `lipi.log` in the current directory.
-output <output filename>	Optional	During testing, this argument can be used to specify the location of the output UNICODE results file. NOTE: If this argument is not specified, the tool generates the default output file, `runwordrec.out`, under the current directory.
-numchoices <numchoices>	Optional	In case of testing, the user can specify the number of recognition choices to be displayed for each recognition result. NOTE: If this argument is not specified, all the choices and confidence values computed by the recognize method will be written to the output file.
-perf	Optional	To find out the time taken for training or testing, -perf can be used. Specifying this argument displays the time taken in seconds, at the end of program execution.
-ver	Optional	Displays the version number of the program.
-help	Optional	Displays the usage of this tool.

Table 7: runwordrec command line arguments

9 Evaluation tool

The evaluation tool facilitates the evaluation and benchmarking of different shape recognizers.

9-1 eval.pl

Responsibilities

The Perl script, `eval.pl`, accepts the output of the runshaperec utility (output file generated from testing the shape recognizer), and generates a report consisting of recognition accuracy and confusion matrix of the shape recognizer. It provides an easy HTML interface to interpret the recognition results.

Usage: eval.pl

```
perl  eval.pl
                -f                  <options file >
```

Or

```
perl  eval.pl
                -input              <input recognition results
                                    file>
                [-output]           <output file>
                [-lipiroot]         <path of lipi-core-toolkit
                                    install directory>
                [-generateHTML]     <1 to enable HTML output; 0
                                    to suppress HTML output
                                    (default 1) >
                [-topError]         <# top confusions required
                                    (default 5)>
                [-imgExt]           <image extension .bmp .png
                                    etc. include the "."
                                    (default .bmp) >
                [-accuracy]         <# top N accracies required
                                    (default 1)>
                [-images]           <# images displayed in a row
                                    (default 10)>
                [-ver]
                [-help]
```

Command line arguments

Command line argument	Argument type	Description

-input	Mandatory	The result file generated by testing the shape recognizer using the runshaperec tool.
-output	Optional	The output file to which the accuracy should be written. NOTE: If no output file is specified, the top accuracies are displayed on the screen.
-lipiroot	Optional	This argument allows the user to specify the path of lipi-core-toolkit install directory. NOTE: If lipiroot is not specified as a command line argument, it's value is retrieved from the environment variable LIPI_ROOT
-generateHTML	Optional	This argument allows the user to specify if HTML is required as output. To generate result in HTML, provide argument as 1 else provide 0 NOTE: The default value for the argument is 1, where HTML will be generated.
-topError	Optional	This argument allows the user to specify the number of top errors to be displayed in the performance analysis file. NOTE: The default value for number of top errors is five.
-imgExt	Optional	This argument allows the user to specify the extension of the image files generated by *imagewriter.pl* NOTE: Please include a "." [DOT] also while specifying the image extension. Example: .png, .bmp, etc. By default the image extension is ".bmp".
-accuracy	Optional	This argument can be used to specify the number of top accuracies to be displayed in performance analysis table. NOTE: The default value for this argument is one. The value of this parameter should be less than or equal to the number of choices in the result file.
-images	Optional	Number of images to be displayed, per row. NOTE: The default value is 10.
-ver	Optional	Displays version information.
-help	Optional	Displays usage information.

Table 8: eval.pl Command line arguments

All these command line arguments can be written into a file, which can be passed as an argument to the evaluation script `eval.pl` using the –f option. The format of the file containing the list of options is shown below

Option1 = value
Option2 = value

In the options file, input and output are similar to the command line arguments -input and –output. The command line arguments –generateHTML, -topError, -imgExt, -accuracy and -images can be specified

in the options file using lipiroot, topError, performance_choices, number_images_row, image_extension respectively. See the **Appendix** for more details on the options file for eval tool.

9-1-1 Output of the Evaluation tool

The evaluation tool creates as output the following files and directories

- resultviewer.html
- confusion_matrix.html
- performance.html
- HTML directory

The following section describes these in detail.

resultviewer.html

The `resultviewer.html` is an HTML interface provided to view the recognition results. The `resultviewer.html` is found under `<lipi-core-toolkit install directory>/scripts`. A screenshot of the `resultviewer.html` is shown below.

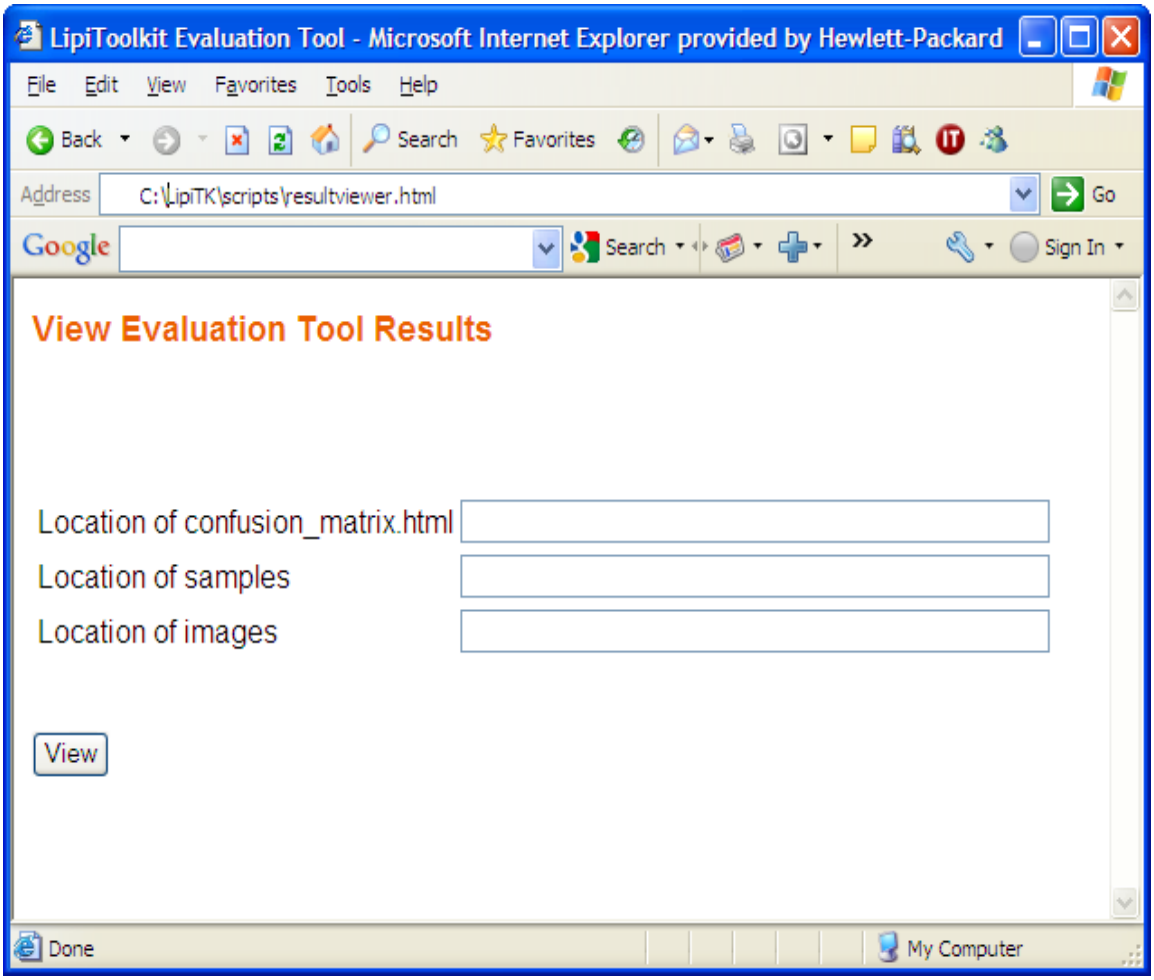


Figure 3: resultviewer.html

1. Enter the location of confusion_matrix.html , example - `<lipi-core-toolkit install directory>/bin`
2. Enter the root directory of the ink samples, example - `<lipi-core-toolkit install directory>/projects/numerals/data/raw`
3. Enter the root directory of the images generated by imagewriter.pl, example - `<lipi-core-toolkit install directory>/projects/numerals/data/out`
4. To display the confusion matrix, click on the **view** button.

confusion_matrix.html

The confusion matrix is a graphical representation of the recognition results. The first column of the confusion matrix represents the true classes and the first row the recognized classes. Each element of the confusion matrix in the *i*th row and *j*th column represents the number of samples of the class corresponding to row *i* confused to the class corresponding to the row *j*. Placing the mouse pointer on an element of the confusion matrix, specimen samples of the original and recognized class are displayed on either sides of the matrix. Click on a cell (i,j) of the confusion matrix to display all the samples of the *i*th class confused with the *j*th class as shown in Figure 6.

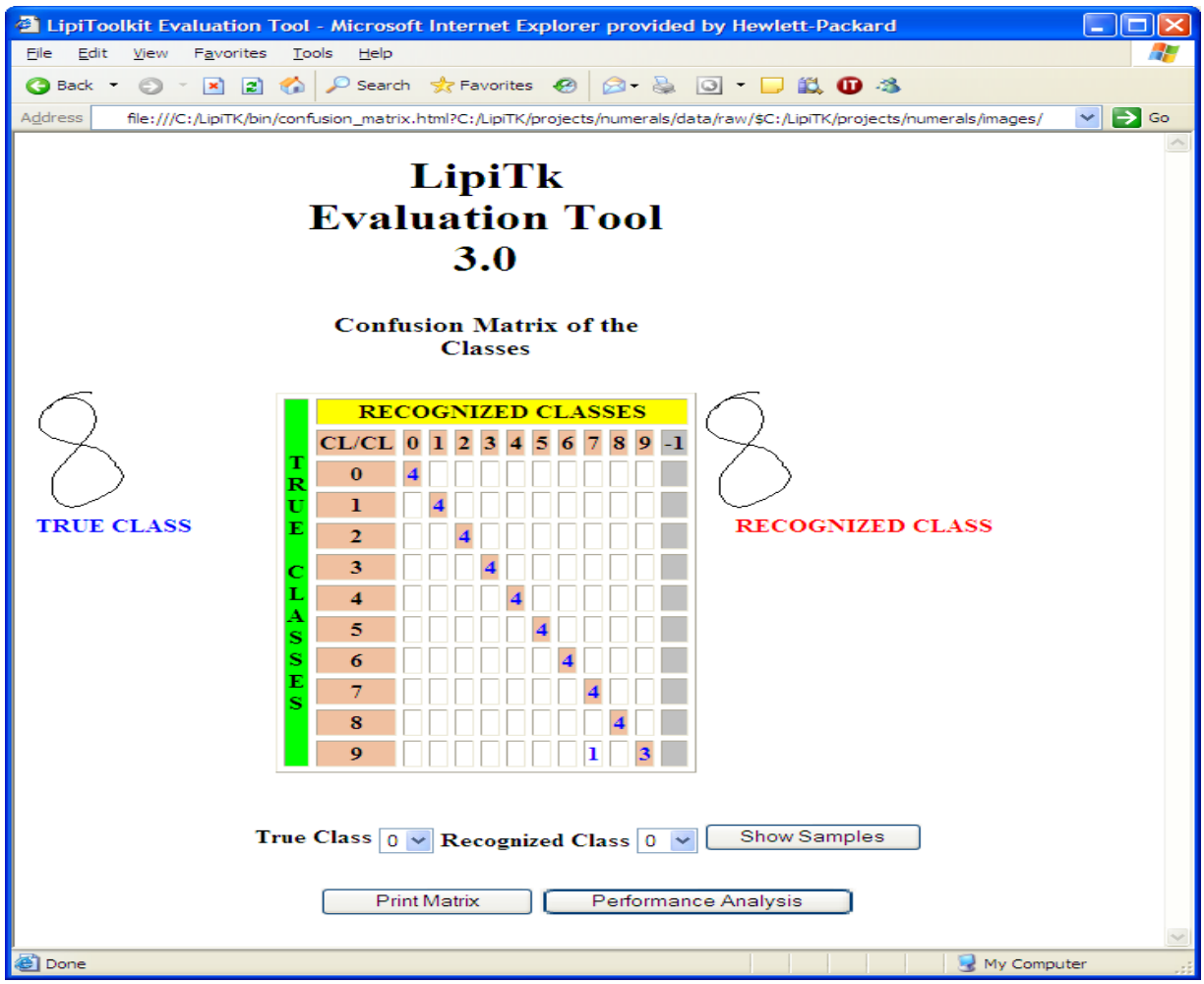


Figure 4 : confusion_matrix.html

For example the value 1 in the third row and ninth column of the confusion matrix represents the number of samples of class one confused to class seven.

performance.html

Clicking on the performance analysis button displays the performance analysis window, `performance.html`. The figure below shows the screenshot of the performance analysis file. The first table in the performance analysis window shows the top N choice accuracy, and the second table displays the top errors.

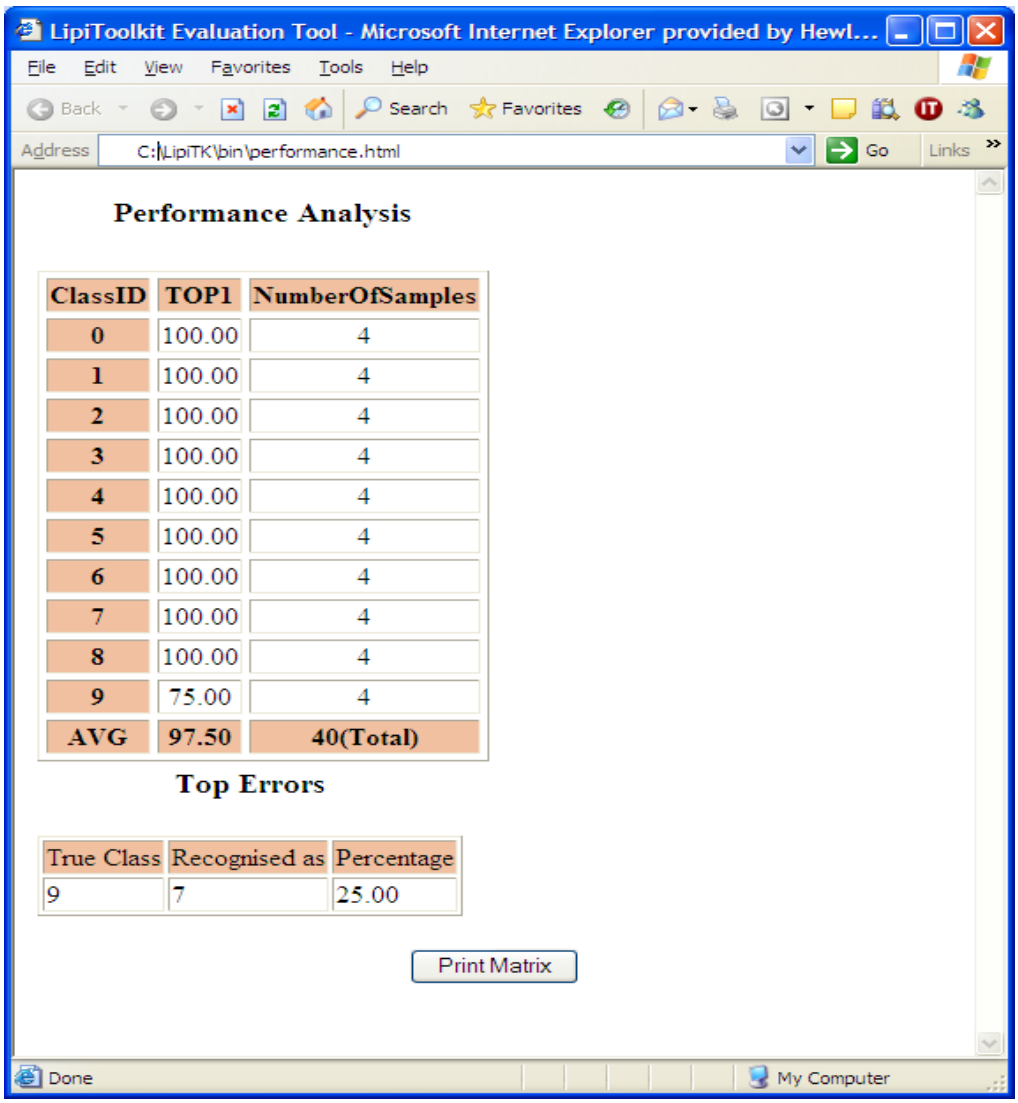


Figure 5 : performance.html

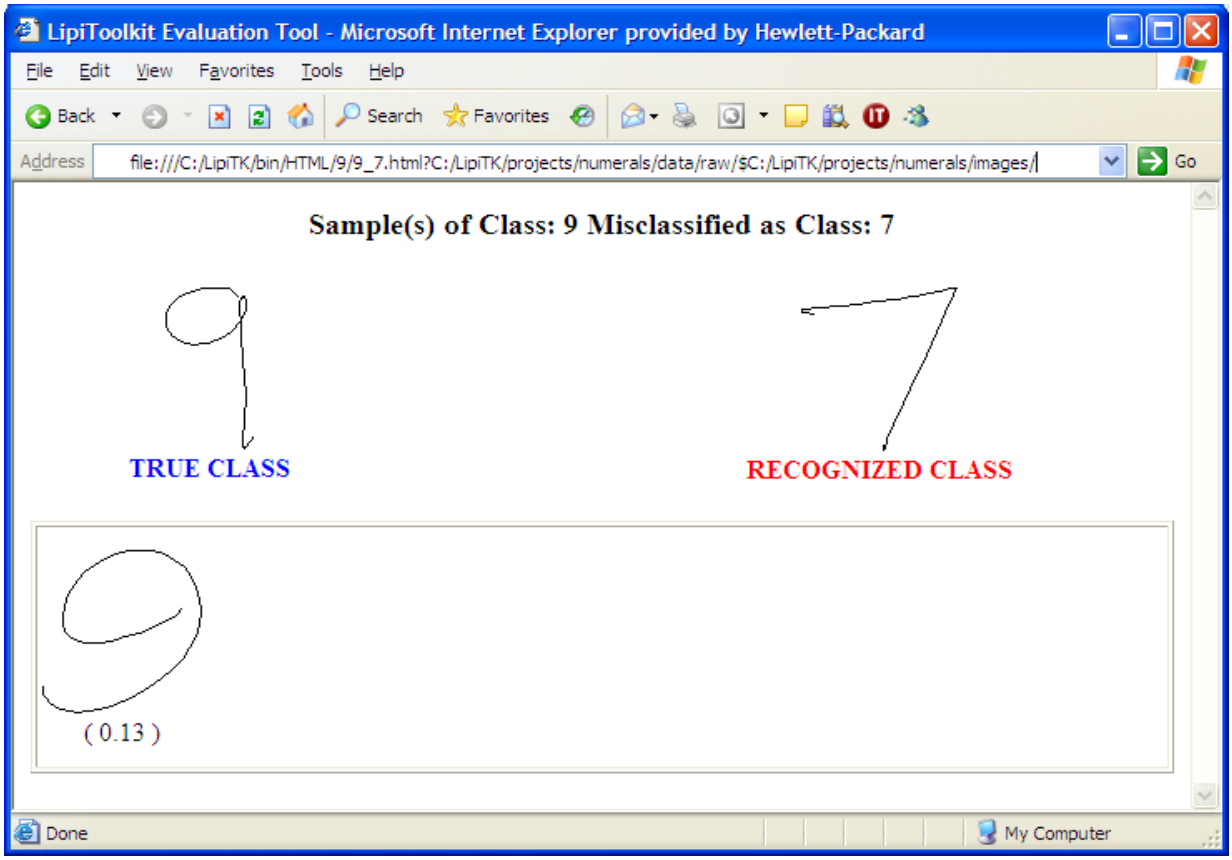


Figure 6 : Misclassified sample of class 1

Clicking on any sample image as shown above displays the ink file corresponding to that image.

HTML directory

The `HTML` directory, created in the same directory as the input file to the evaluation tool, contains a subdirectory for every class in the input file. The lipi-core-toolkit cascaded style sheet file, `lipitk.css`, available under `<lipi-core-toolkit install directory>/scripts` directory, is also copied to this directory.

NOTE: All the HTML files use the lipi-core-toolkit cascaded style sheet, `lipitk.css`. To change the look and feel of the HTML files, modify this css file. To create the images refer to the usage of imagewriter.pl utility.

9-2 evalAdapt.pl

Responsibilities

This script can be used to evaluate the performance of the adaptation algorithm. To compute the performance, the script computes the accuracy on bins of samples in the order in which they are presented to the recognizer. The Perl script, `evalAdapt.pl`, accepts the output of the runshaperec

utility (output file generated from running the shape recognizer in adaptation mode), and generates a text file consisting of recognition accuracies computed on the bins specified through binsize and overlap options. The final accuracy in the output text file is the accuracy computed on the final set of samples specified through final bin option.

Usage: evalAdapt.pl

```
perl  evalAdapt.pl
                    -input              <input recognition results
                                        file>
                    -outdir             <output directory>
                    -resultfile         <output text file to which
                                        the computed accuracies will
                                        be written>
                    -binsize            <the number of samples per
                                        bin on which accuracy has to
                                        be computed>
                    -overlap            <overlap value for each bin>
                    -finalbin           <the size of the final bin
                                        on which final accuracy has
                                        to be calculated>
                    [-lipiroot]         <path of lipi-core-toolkit
                                        install directory>
                    [-ver]
```

Command line arguments

Command line argument	Argument type	Description
-input	Mandatory	The result file generated by testing the shape recognizer using the runshaperec tool.
-outdir	Mandatory	The output directory to which the accuracy should be written.
-resultfile	Mandatory	Output text file to which the computed accuracies will be written.
-binsize	Mandatory	The input recognition results file is split into bins of this size for computing the accuracy and accuracy is calculated on each of these bins.
-overlap	Mandatory	While splitting the file into bins, the bins are created with this overlap value.
-finalbin	Mandatory	The size of the final bin on which final accuracy has to be calculated.
-lipiroot	Optional	This argument allows the user to specify the path of lipi-core-toolkit install directory. NOTE: If lipiroot is not specified as a command line argument, it's value is retrieved from the environment variable LIPI_ROOT
-ver	Optional	Displays version information.

Table 9: evalAdapt.pl Command line arguments

10 Packaging

10-1 createAddOn.pl

Responsibilities This script can be used to package/zip shape recognition projects created by lipi designer or by using the runshaperec tool.

Usage:

```
perl createAddOn.pl
                        -project            <project name, to be packaged>
                        -logicalname        <logical name of the project to be packaged>
                        [-lipiroot]         <toolkit install directory>
                        [-ver]              <version of the toolkit>
                        [-help]             <display all parameters supported>
```

Command line arguments

Command line argument	Argument type	Description
-project	Mandatory	Project name.
-logicalname	Mandatory	Logical name of the project
-lipiroot	Optional	Lipi toolkit install directory
-ver	Optional	Displays version information
-help	Optional	Displays usage information.

Table 10: createAddOn.pl command line arguments

10-2 installAddOn.pl

Responsibilities This script can be used to install shape recognizers packaged using createAddOn.pl.

Packaged shape recognizers will be installed to the projects directory and lipiengine.cfg will be updated with the details of the installed shape recognizer.

Usage:

```
perl installAddOn.pl
                        -[lipiroot]         <toolkit install directory>
                        -[ver]              <version of the toolkit>
                        -[help]             <display all parameters supported>
```

Command line arguments

Command line argument	Argument type	Description
-lipiroot	Optional	Lipi toolkit install directory
-ver	Optional	Displays version information
-help	Optional	Displays usage information.

Table 11: addproject.pl command line arguments

11 Scripts

lipi-core-toolkit 4.0 package comes with various scripts. All the scripts are written in Perl and are present under `<lipi-core-toolkit install directory>/scripts`. This section describes the various utility scripts provided in the lipi-core-toolkit package.

11-1 trimlines.pl

Responsibilities

The script removes the extraneous characters (^m character at the end of every line), incurred by transferring data files across platforms. The tool traverses recursively through the dataset directory, to look for all `*.txt` files and removes all the extraneous characters.

Usage: trimlines.pl

```
perl  Trimlines.pl
                    -indir     <root of input directory>
                    [-ver]
                    [-help]
```

Command line arguments

Command line argument	Argument type	Description
-indir	Mandatory	The root directory of the data set.
-ver	Optional	Displays version information
-help	Optional	Displays usage information.

Table 12: trimlines.pl command line arguments

11-2 extracthwdata.pl

Responsibilities

The script extracts handwriting data of specified hierarchy from a set of UNIPEN files. It splits a UNIPEN file into different files, each containing one element of the specified hierarchy.

Usage: extracthwdata.pl

```
perl  extracthwdata.pl
                          -indir    <Dataset root directory>
                          -outdir   <Output directory of extracted
                                    data>
                          -hlevel   <Hierarchy>
                          [-ver]
                          [-help]
```

Command line arguments

Command line argument	Argument type	Description
-indir	Mandatory	The root directory of the data set.
-outdir	Mandatory	The new path where the split files will be created.
-hlevel	Mandatory	The UNIPEN hierarchy level, e.g. CHARACTER.
-ver	Optional	Displays version information.
-help	Optional	Displays help information.

Table 13: extracthwdata.pl Command line arguments

Example:

When the script is invoked with hierarchy as CHARACTER, three files and_i1sym_a.txt, and_i1sym_n.txt, and_i1sym_d.txt are extracted from the following file and.txt. The extracted file name comprises the original file name, the symbol id and the instance number of the symbol in the original file.

```
.VERSION 1.0
.DATA SOURCE hpl
...
.COORD X Y T
.SEGMENT WORD 0,3,2,4-5 ok “and”
.SEGMENT CHARACTER 0,3,2 ok “a”
.SEGMENT CHARACTER 4 ok “n”
.SEGMENT CHARACTER 5 ok “d”
.PEN_DOWN
783 707 0
...
637 1178 0
.PEN_UP
.PEN DOWN
1023 1047 0
...
1132 1028 0
.PEN_UP
.PEN_DOWN
1408 751 0
...
1399 1150 0
.PEN UP
.PEN_DOWN
962 950 0
...
1417 950 0
.PEN_UP
.PEN_DOWN
850 654 0
...
2151 907 0
.PEN UP
.PEN_DOWN
2154 717 0
...
2606 1075 0
.PEN_UP
```

File-1: (and_i1sym_a.txt)	File-2: (and_i1sym_n.txt)	File-3: (and_i1sym_d.txt)
.VERSION 1.0	.VERSION 1.0	.VERSION 1.0
.DATA_SOURCE hpl	.DATA_SOURCE hpl	.DATA_SOURCE hpl
...	...	...
.COORD X Y T	.COORD X Y T	.COORD X Y T

.SEGMENT CHARACTER 0-2 ok "1" .PEN_DOWN 783 707 0 ... 637 1178 0 .PEN_DOWN 962 950 0 ... 1417 950 0 .PEN_UP .PEN_DOWN 1408 751 0 ... 1399 1150 0 .PEN_UP	.SEGMENT CHARACTER 1 ok "2" .PEN_DOWN 850 654 0 ... 2151 907 0 .PEN_UP	.SEGMENT CHARACTER 1 ok "3" .PEN_DOWN 2154 717 0 ... 2606 1075 0 .PEN_UP

Table 14: Output file generated after executing extracthwdata.pl

11-3 listfiles.pl

Responsibilities

The `listfiles.pl` script generates the list file for training and testing the shape and word recognizer modules. The input to the script is a map file containing regular expressions for each class. The format of the map file is as follows:

<Shape ID>SPACE<regular expression corresponding to the data file path relative to the data root directory>

where, the first column denotes the shape ID and the second column denotes a regular expression, which on expansion, gives all the data files corresponding to the shape ID.

Example:

0 usr[0-5]/000000t.txt*
1 usr[0-5]/000001t.txt*
…
…
9 usr[0-5]/000009t.txt*

IMPORTANT: The shape IDs should be listed sequentially in ascending order.

Usage: listfiles.pl

```
perl  listfiles.pl
                    -indir          <Dataset root directory>
                    -output         <Output list file>
                    -config         < The map file containing
                                    the regular expressions>
                    [-adapt]        <randomize the input file
                                    for the evaluation of
                                    adaptation>
                    [-prototype]    <numer of prototypes for
                                    adaptation. To be used with
                                    -adapt>
                    [-ver | v]
                    [-help]
```

Command line arguments

Command line argument	Argument type	Description
-indir	Mandatory	The root directory of the data set.
-output	Mandatory	Name of the output list file. NOTE: By default, the output list file is created in the current directory. However, user can provide absolute or relative path to control the location of the output file.
-config	Mandatory	The path of the map file.
-adapt	Optional	To randomize the input file for the evaluation of adaptation
-prototypes	Optional	Number of prototypes that will be added to the model during adaptation. To be used with –adapt
-ver | -v	Optional	Displays version information.
-help	Optional	Displays help information.

Table 15: listfiles.pl Command line arguments

11-4 validatelistfile.pl

Responsibilities

Training the shape recognizer using a list file requires the shape ID to be listed in an ascending order of shape ids. This script enables the user to validate his/her list file. Given the path of input list file, this script generates a new list file with the shape IDs listed in ascending order.

The script creates a new list file in the same directory as the input list file, with "_new" added as suffix to the name.
Example: Executing the following command

```
perl validatelistfile.pl -input listfile.txt
```

results in the creation of file `listfile_new.txt`, with shape IDs listed in ascending order.

Usage: validatelistfile.pl

```
perl  validatelistfile.pl
                          -input        <input list file path>
                          [-ver|v]
                          [-help]
```

Command line arguments

Command line argument	Argument type	Description
-input	Mandatory	Takes the path of the list file to be validated
-ver|v	Optional	Displays version of the script.
-help	Optional	Displays usage information.

Table 16: validatelistfile.pl Command line arguments

11-5 benchmark.pl

Responsibilities

This script performs training, testing and evaluation of a shape recognizer for the project and profile specified as command line arguments. Testing the shape recognizer, results in the creation of an output file, `runshaperec.out`. This is used as an input for evaluating the recognition accuracy.

Usage: benchmark.pl

```
perl  benchmark.pl
                    -project       <project name>
                    -train         <path of training list
                                   file>
                    -test          <path of testing list
                                   file>
                    [-logfile]     <log file name>
                    [-profile]     <profile name>
                    [-lipiroot]    <path of lipi-core-toolkit
                                   install directory>
```

```
                    [-ver|v]
                    [-help]
```

Command line arguments

Command line argument	Argument type	Description
-project	Mandatory	For benchmarking, the user needs to pass the project name. Example: -project numerals NOTE: lipi-core-toolkit searches for the directory having the name as that of the project under `<lipi-core-toolkit install directory>/projects..`
-train	Mandatory	Takes the path of the training list file. Refer to listfiles.pl, for the format of the list file.
-test	Mandatory	Takes the path of the list file for testing. Refer to listfiles.pl, for the format of the list file.
-logfile	Optional	This argument allows the user to specify the log file name for the lipi-core-toolkit log messages. NOTE: By default, the log messages are written to lipi.log in the current directory.
-profile	Optional	This argument allows the user to specify the profile to be used for the project. NOTE: If the profile name is omitted, 'default' profile is assumed.
-lipiroot	Optional	Path of lipi-core-toolkit install directory. NOTE: If lipiroot is not specified as a command line argument, it's value is retrieved from the environment variable LIPI_ROOT
-ver|v	Optional	Displays version of the script.
-help	Optional	Displays usage information.

Table 17: benchmark.pl Command line arguments

11-6 imagewriter.pl

Responsibilities

This script generates the images of the UNIPEN ink files. Make sure that the executable, `imgwriter.exe` exists in the `<lipi-core-toolkit install directory>/bin` directory, before invoking the script.

Usage: imagewriter.pl

```
perl  imagewriter.pl
                    -indir        <root of the dataset>
```

```
-outdir         <output directory>
[-infileext]    <extension of the ink
                files>
[-imagesize]    <Size of the output image
                in Pixels>
[-lipiroot]     <path of lipi-core-toolkit
                root directory>
[-ver | v]
[-help]
```

Command line arguments

Command line argument	Argument type	Description
-indir	Mandatory	The root directory of the data set.
-outdir	Mandatory	The output directory to store the image files.
-infileext	Optional	The extension of the input data file. The script creates images only for the files with the same extension as specified. NOTE: If no file extension is specified, the default file extension ".txt" is assumed.
-imagesize	Optional	The size of image in pixels. NOTE: If image size is not specified, the default size of 100 pixels is assumed.
-lipiroot	Optional	Path of lipi-core-toolkit root directory. NOTE: If lipiroot is not specified as a command line argument, it's value is retrieved from the environment variable LIPI_ROOT
-ver	Optional	Displays version information
-help	Optional	Displays usage information.

Table 18: imagewriter.pl Command line arguments

12 Utilities

12-1 featurefilewriter – feature writer

The feature writer application, `featurefilewriter`, available under `<lipi-core-toolkit install directory>/bin`, is use to generate file that features extracted for UNIPEN ink file. To build the executable, please follow the instructions given below.

Usage: featurefilewriter

```
featurefilewriter
                    -cfg            <cfg file path>
                    -list           <list filename>
                    -output         <output filename>
                    [-lipiroot]     <path of lipi-core-toolkit root
                                    directory>
                    [-loglevel]     <log level: Debug|Error|info|all>
                    [-ver]
```

Command line arguments

Command line argument	Argument type	Description
-cfg	Mandatory	Takes the profile.cfg file path.
-list	Mandatory	Takes the path of the list file. Refer to listfiles.pl, for the format of the list file.
-output	Mandatory	Path to generate file after the features are extracted.
-lipiroot	Optional	Path of lipi-core-toolkit install directory. NOTE: If lipiroot is not specified as a command line argument, it's value is retrieved from the environment variable LIPI_ROOT
-loglevel	Optional	This argument allows the user to specify the log level {*Debug|Error|info|all*}.
-ver	Optional	Displays the version of the tool

Table 19: featurefile Command line arguments

Description

The featurefilewriter tool reads the profile.cfg and list file as input, extract the features from the each UNIPEN ink file mention in the list file and generate a feature file specified as an ouput command-line argument.

12-2 Imagewriter – image writer

The image writer application, `imgwriter`, available under `<lipi-core-toolkit install directory>/bin`, is invoked by the Perl script `imagewrite.pl` to generate the images for UNIPEN ink file. To build the executable, please follow the instructions given below.

12-3 mdv – model data viewer

The mdv tool, an executable residing under `<lipi-core-toolkit install directory>/bin` directory, accepts a model data file as an input and displays the model data header information.

Usage: mdv

```
mdv
        -input          <path of model data file>
        [-projname]
        [-numshapes]
        [-recname]
        [-recver]
        [-checksum]
        [-createtime]
        [-modtime]
        [-headerlen]
        [-dataoffset]
        [-headerver]
        [-byteorder]
        [-platform]
        [-preproc]
        [-ver]
        [-all]
        [-help]
```

Command line arguments

Command line argument	Argument type	Description
-input	Mandatory	Takes the path of the input model data file.
-projname	Optional	Displays the name of the project.
-numshapes	Optional	Displays the number of shapes.

-recname	Optional	Displays the name of the shape recognizer.
-recver	Optional	Displays the version the shape recognizer.
-checksum	Optional	Displays the checksum of the file.
-createtime	Optional	Displays the date of creation of the input file.
-modtime	Optional	Displays the date of last modification of the input file.
-headerlen	Optional	Displays the length of the model data header.
-dataoffset	Optional	Displays the byte offset value of the start of data in file.
-headerver	Optional	Displays the version of model data header.
-byteorder	Optional	Displays the byte order • Little endian or • Big endian
-platform	Optional	Displays the platform, on which the input file is created.
-preproc	Optional	Displays the preproc fields
-ver	Optional	Displays the version of the tool
-all	Optional	Displays all the fields.
-help	Optional	Displays the tool usage.

Table 20: mdv Command line arguments

Description

The tool reads the input model data file, validates the checksum for the file and displays the header information. The output can be customized using the command-line arguments.

12-3-1 mdv – errors

Error Scenario	Error code	Error message
Input file not found	EMODEL_DATA_FILE_OPEN	Unable to open model data file.
Failed to open input file for reading	EMODEL_DATA_FILE_OPEN	Unable to open model data file.
The input file does not contain header.	EMODEL_DATA_FILE_FORMAT	Incompatible model data file. The header is not in the desired format.
The header in the input file is corrupt	EMODEL_DATA_FILE_CORRUPT	Model data file is corrupted.

Table 21: mdv errors

13 Sample client applications

13-1 Introduction

The sample programs demonstrate how to use APIs exposed by recognition components and modules of lipi-core-toolkit with client applications which require shape or word recognition.

lipi-core-toolkit 4.0 provides two sample applications for windows and linux

- shaperectst: sample client application for shape recognizers.
- wordrectst: sample client application for word recognizers.

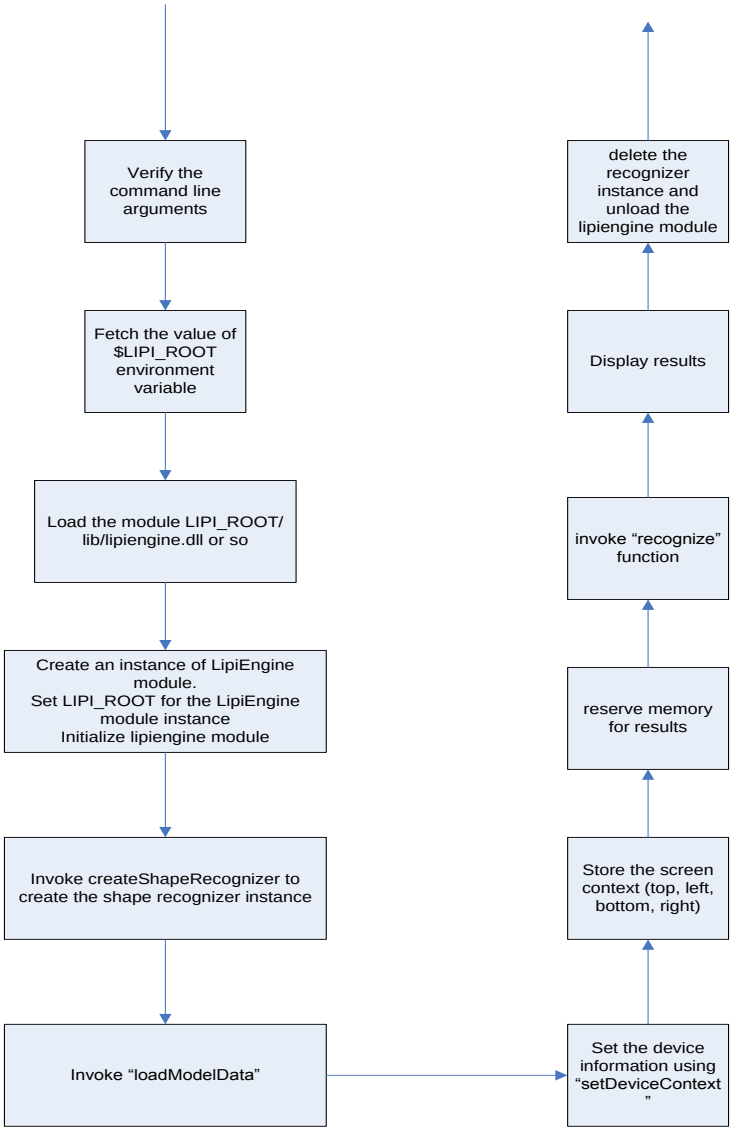
Apart from these, it provides a sample UI application for windows, where user can load existing bundled shape recognizers and test.

- shaperectstui: sample ui client application for shape recognizers (Windows only).

The following sections cover all the technical details for the sample client applications provided by Lipitk.

13-2 Sample program `shaperectst`

The sample program `shaperectst` is provided as an example of how the character recognizers can be invoked from an application program. It illustrates the steps for instantiating the recognizers using their logical names, passing digital ink to them and obtaining recognition results. The major steps in the program are illustrated below:



13-2-1 Included source code, headers and binaries

13-2-1-1 Source directory

File description	Location
Sample programs for shape recognizer	*$LIPI_ROOT/src/apps/samples/shaperectst*
Common header files	*$LIPI_ROOT/src/include*
Common libraries for linking	*$LIPI_ROOT/src/lib*

Table 22: shaperectst file locations

13-2-1-2 Required libraries

Library ($LIPI_ROOT/src/lib)	Remarks
common.lib/common.a	Common data structures and manipulating functions
shapereccommon.lib/shapereccommon.a	Shape recognition specific data structures
featureextractorcommon.lib/featureextractorcommon.a	Feature extractor specific data structures
utils.lib/utils.a	Utilities to read/write *UNIPEN* ink, Logger, etc.

Table 23: shaperectst - static libraries required

Library ($LIPI_ROOT/lib)	Remarks
preproc.dll/libpreproc.so	Preprocessing functions
pointfloat.dll/libpointfloat.so, l7.dll/libl7.so, npen.dl/libnpen.so, substroke.dll/libsubstroke.so	Functions for feature extraction
lipiengine.dll/liblipiengine.so	Controller class for loading and creating shape recognizers
nn.dll/libnn.so, adaptivedtw.dll/libadaptivedtw.so, neuralnet.dll/libneuralnet.so	Nearest neighbor/adaptivedtw/neuralnet shape recognizer

Table 24: shaperectst - shared libraries required

13-2-1-3 Required header files

Header file ($LIPI_ROOT/src/include)	Remarks
LTKInkFileReader.h	To read ink files in *UNIPEN* format
LTKLipiEngineInterface.h	Defines the interface for LipiEngine module
LTKMacros.h	Defines global macros which are used across Lipitk
LTKInc.h	Generic include file which includes all standard include headers
LTKTypes.h	Defines all the lipi-core-toolkit specific common datatypes
LTKTrace.h	Defines LTKTrace datatype which is used to store ink info

Table 25: shaperectst - header files required

13-2-2 Important data structures - shape recognition

This section describes some of the main data structures from `common.lib` and their usage in the `shaperectst` code

Screen context (LTKScreenContext): Stores the coordinates of the writing area.

Member Functions Used

Member function name	Description
setBboxLeft()	Sets bottom left x co-ordinate of the writing area
setBboxBottom()	Sets bottom left y co-ordinate of the writing area
setBboxRight()	Sets top right x co-ordinate of the writing area
setBboxTop()	Sets top right y co-ordinate of the writing area

Table 26: LTKScreenContext member functions used: shaperectst

Device context (LTKDeviceContext): Stores information about the device used for input.

Member functions used

Member function name	Description
setSamplingRate ()	Stores the sampling rate of the device.
setXDPI ()	Stores the horizontal direction resolution of the device
setYDPI ()	Stores the vertical direction resolution of the device
setLatency ()	Stores the interval between the time of actual input to that of its registration
setUniformSampling()	Sets the flag to indicate if the sampling is uniform

Table 27: LTKDeviceContext member functions used: shaperectst

Ink data structure (LTKTraceGroup): Stores the digital ink

Results data structure (LTKShapeRecoResult): Stores recognition result and returns it back to the application program

Member functions used

Member function name	Description
getShapeId ()	Returns the shape id
getConfidence ()	Returns the confidence corresponding to the shape id

Table 28: LTKShaperecResult member functions used: shaperectst

Usage: shaperectst

```
shaperectst
                    <logical project name>
                    <ink file to recognize>
```

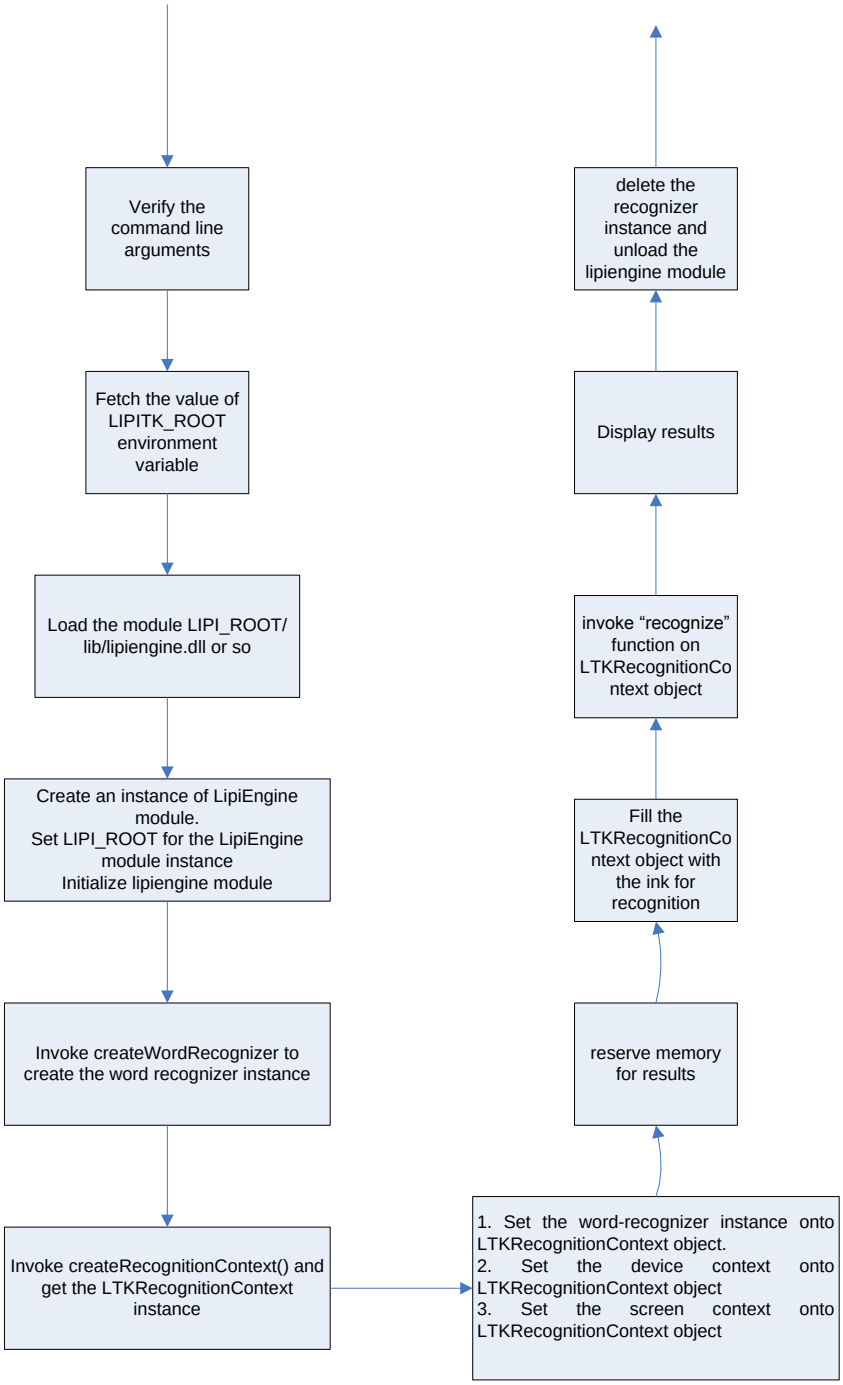
Command line arguments

Command line argument	Argument type	Description

<logical project name>	Mandatory	The user needs to pass the logical project name as specified in the lipiengine.cfg. Example: SHAPEREC_NUMERALS, for numerals project
< ink file to recognize>	Mandatory	Path of the UNIPEN ink file to be recognized should be passed.

Table 29: Command line arguments: shaperectst

13-3 Sample program `wordrectst`



13-3-1 Included source code, headers and binaries

13-3-1-1 Source directory

File description	Location
Sample program for boxedfield recognizer	*$LIPI_ROOT/src/apps/samples/wordrectst*
Common header files	*$LIPI_ROOT/src/include*
Common libraries for linking	*$LIPI_ROOT/src/lib*

Table 30: File locations: wordrectst

13-3-1-2 Required libraries

Library ($LIPI_ROOT/src/lib)	Remarks
common.lib	Common data structures and manipulating functions
shapereccommon.lib	Shape recognition specific data structures
featureextractorcommon.lib	Feature extractor specific data structures
wordreccommon.lib	Data structures specific to any word recognizer
utils.lib	Utilities to read/write *UNIPEN* ink, Logger, etc.

Table 31: Static libraries required: wordrectst

Library ($LIPI_ROOT/lib)	Remarks
boxfld.dll/libboxfld.so	Boxedfield word recognizer
preproc.dll/libpreproc.so	Preprocessing functions
pointfloat.dll/libpointfloat.so, l7.dll/libl7.so, npen.dl/libnpen.so, substroke.dll/libsubstroke.so	Functions for feature extraction
lipiengine.dll/liblipiengine.so	Controller class for loading and creating shape and word recognizers
nn.dll/libnn.so, adaptivedtw.dll/libadaptivedtw.so, neuralnet.dll/libneuralnet.so	Nearest neighbor/adaptivedtw/neuralnet shape recognizer

Table 32: Shared libraries required: wordrectst

13-3-1-3 Required header files

Header file ($LIPI_ROOT/src/include)	Remarks
LTKInkFileReader.h	To read ink files in *UNIPEN* format
LTKLipiEngineInterface.h	Defines the interface for LipiEngine module
LTKMacros.h	Defines global macros which are used across Lipitk
LTKInc.h	Generic include file which includes all standard include headers
LTKTypes.h	Defines all the lipi-core-toolkit specific common datatypes
LTKTrace.h	Defines LTKTrace datatype which is used to store ink info

Table 33: Header files required: wordrectst

13-3-2 Important data structures – Boxed - Field recognition

Recognition context (LTKRecognitionContext): Specifies UI parameters, application specific parameters and recognition related configurations.

Member functions used

Member function name	Description
beginRecoUnit ()	Marks the beginning of a recognition unit of Ink
addTraceTroup()	Adds a vector of trace group for recognition in the recognition context.
endRecoUnit()	Marks the end of a recognition unit of Ink
setWordRecoEngine()	Sets the word recognizer to be used.

Table 34: Some of the member functions of LTKRecognitionContext

Usage: wordrectst

```
wordrectst
                    <logical project name>
                    < list file to recognize>
                    < outputfile>
```

Command line arguments

Command line argument	Argument type	Description
<logical project name>	Mandatory	The user needs to pass the logical project name as specified in the lipiengine.cfg. Example:NUMERALS_FLD, for numerals project
< list file to recognize>	Mandatory	Path of the list file to be recognized should be passed.
< outputfile>	Mandatory	This argument is used to specify the output file for the wordrectst.

Table 35: Command line arguments: wordrectst

13-4 Sample program `shaperectstui`

The sample program `shaperectstui` is provided as an example of how the character recognizers can be invoked from an application program. It illustrates the steps for instantiating the recognizers using their logical names, passing digital traceGroup to them and obtaining recognition results.

13-4-1 Included source code, headers and binaries

13-4-1-1 Source directory

File description	Location
Sample programs for shape recognizer	*$LIPI_ROOT/src/apps/samples/shaperectstui*
Common header files	*$LIPI_ROOT/src/include*
Common libraries for linking	*$LIPI_ROOT/src/lib*

Table 44: shaperectstui file locations

13-4-1-2 Required libraries

Library ($LIPI_ROOT/src/lib)	Remarks
common.lib	Common data structures and manipulating functions
shapereccommon.lib	Shape recognition specific data structures
featureextractorcommon.lib	Feature extractor specific data structures
utils.lib	Utilities to read/write *UNIPEN* ink, Logger, etc.

Table 44: shaperectstui - static libraries required

Library ($LIPI_ROOT/lib)	Remarks
preproc.dll	Preprocessing functions
pointfloat.dll	Functions for feature extraction
lipiengine.dll	Controller class for loading and creating shape recognizers
nn.dll	Nearest neighbor shape recognizer

Table 36: shaperectstui - shared libraries required

13-4-1-3 Required header files

Header file ($LIPI_ROOT/src/include)	Remarks
LTKInkFileReader.h	To read ink files in *UNIPEN* format
LTKLipiEngineInterface.h	Defines the interface for LipiEngine module

LTKMacros.h	Defines global macros which are used across lipitk
LTKInc.h	Generic include file which includes all standard include headers
LTKTypes.h	Defines all the lipi-core-toolkit specific common datatypes
LTKTrace.h	Defines LTKTrace datatype which is used to store ink info

Table 46: shaperectstui - header files required

13-4-2 Important data structures - shape recognition

This section describes some of the main data structures from `common.lib` and their usage in the `shaperectstui` code

Screen context (LTKScreenContext): Stores the coordinates of the writing area.

Member Functions Used

Member function name	Description
setBboxLeft()	Sets bottom left x co-ordinate of the writing area
setBboxBottom()	Sets bottom left y co-ordinate of the writing area
setBboxRight()	Sets top right x co-ordinate of the writing area
setBboxTop()	Sets top right y co-ordinate of the writing area

Table 47: LTKScreenContext member functions used: shaperectstui

Device context (LTKDeviceContext): Stores information about the device used for input.

Member functions used

Member function name	Description
setSamplingRate ()	Stores the sampling rate of the device.
setXDPI ()	Stores the horizontal direction resolution of the device
setYDPI ()	Stores the vertical direction resolution of the device
setLatency ()	Stores the interval between the time of actual input to that of its registration
setUniformSampling()	Sets the flag to indicate if the sampling is uniform

Table 48: LTKDeviceContext member functions used: shaperectstui

Ink data structure (LTKTraceGroup): Stores the digital ink

Results data structure (LTKShapeRecoResult): Stores recognition result and returns it back to the application program

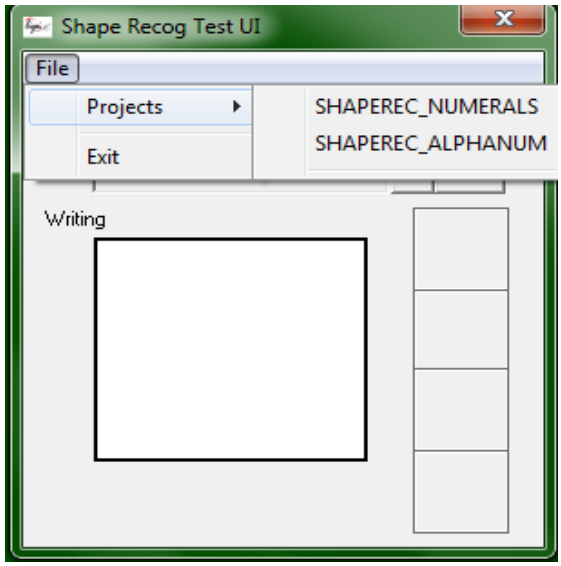
Member functions used

Member function name	Description
getShapeId ()	Returns the shape id

getConfidence ()	Returns the confidence corresponding to the shape id

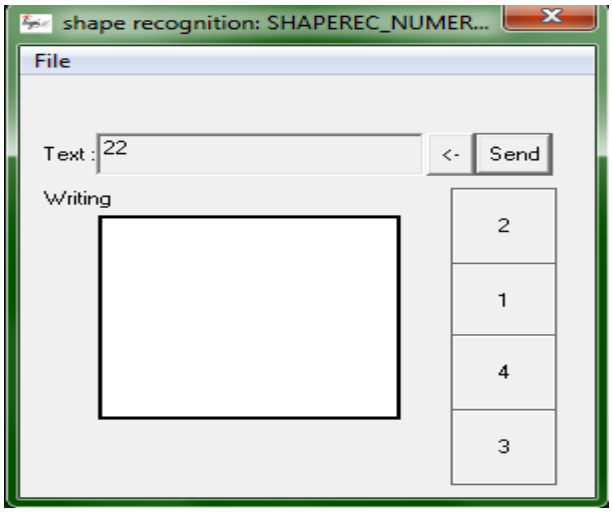
Table 49: LTKShaperecResult member functions used: shaperectstui

Usage: All logical manes for the projects mentioned in lipiengine.cfg are displayed in **File** menu **Projects** submenu



User can select the project to be tested like SHAPEREC_ALPHANUM, user can draw the shape to be recognized in Writing area, when user is finished drawing, in a timeout period it will process the shape and display 4 best matching shapes in recognized results. Results are displayed in descending order of recognition accuracy.

If we draw '2' in writing area following results are displayed :



14 Using Lipitk

14-1 Creating and using a shape secognizer

In this chapter, we will walk through the steps for creating a shape recognizer from scratch, using the shape recognition modules provided by *lipi-core-toolkit*.

In broad terms, the steps involved in creating a shape recognizer are as follows:

- Installing the toolkit
- Setting up a *lipi-core-toolkit* Shape Recognition Project
- Collecting shape samples (handwriting data)
- Preparing the data for Training and Testing
- Building the Project
- Training the recognizer
- Testing the recognizer
- Evaluating recognition performance
- Packaging the Shape Recognizer for deployment

We will first create a shape recognizer for numerals using the numeral data provided with the toolkit. We will then look at the integration of these recognizers into applications.

NOTE: The scenarios described here do not require any changes to the code provided.

14-2 Creating a handwritten numeral recognizer

This section describes the steps for creating a numeral recognizer, using the **demonumerals** project and data already provided with the toolkit.

Setting up a shape recognition project

lipi-core-toolkit 4.0 provides the **demonumerals** project under the project directory `<lipi-core-toolkit install directory>/projects/`. Hereafter, we use $PROJ_ROOT to refer to the directory `<lipi-core-toolkit install directory>/projects/demonumerals`.

The project configuration file, `project.cfg`, for the **demonumerals** project is available under the directory `$PROJ_ROOT/numerals/config/`. This file contains the following settings:

```
ProjectType = SHAPEREC
NumShapes = 10
```

where, the key *ProjectType* defines the project as a shape (as opposed to word) recognition project. The key *NumShapes* specifies the number of shapes to be recognized.

lipi-core-toolkit 4.0.0 provides the **default** profile for this project. The profile configuration file, `profile.cfg`, available under the directory `$PROJ_ROOT/demonumerals/config/default/`, contains the following settings:

```
ShapeRecMethod = nn
```

where, the key *ShapeRecMethod* specifies nn as the shape recognition module to be used. In addition to the `profile.cfg`, the default profile also provides the configuration file for the nearest neighbor shape recognizer, `nn.cfg`.

Collecting shape samples (handwriting data)

The numerals project is packaged with ten handwritten samples of each numeral. Hence, this step can be skipped for this project. However, the data collection tools are available under lipi-core-toolkit downloads on sourceforge (**https://sourceforge.net/project/showfiles.php?group_id=165380&package_id=206908**). User is advised to use any of the tools for collecting hand written data. The tool comes with a user manual which describes the usage.

Henceforth, we will use $DATA_ROOT to refer to the `<lipi-core-toolkit install directory>/demonumerals/data/` directory.

Data preparation for Training and Testing

Both training and testing requires lists (text files) of file names of data samples with their full paths and the corresponding shape ids. The files `trainlist.txt` and `testlist.txt` have already been provided in the $DATA_ROOT. The first 6 samples for each numeral are specified for training, the remainder for testing.

IMPORTANT: The data for the demonumerals project is only provided for the purposes of illustrating the use of the toolkit. Building a good numeral recognizer and evaluating it thoroughly would require much larger amounts of data.

Training the recognizer

The recognizer now needs to be trained using the list of training samples prepared earlier:

- Go to the bin directory

```
cd <lipi-core-toolkit install directory>/bin
```

- Execute *runshaperec* to train the recognizer

```
runshaperec -train $DATA_ROOT/trainlist.txt
            -project numerals
```

The steps listed above cause the profile configuration file to be read from `$PROJ_ROOT/demonumerals/config/default` and used for training the chosen shape recognition module (in this case, nn) using the training samples specified in `trainlist.txt`. Training the shape recognizer results in the creation of model data, in this case `nn.mdt`. The model data file is generated in the profile directory `$PROJ_ROOT/demonumerals/config/default`.

Testing the recognizer

The trained shape recognizer may now be tested on the test data earmarked earlier:

- Go to the bin directory

```
Cd <lipi-core-toolkit install directory>/bin
```

- Execute `runshaperec`, this time in test mode

```
runshaperec  -test $DATA_ROOT/testlist.txt
             -project demonumerals
             -output results.txt
```

This causes the `nn.cfg` and `nn.mdt` to be read from the profile directory `$PROJ_ROOT/config/default`, and used for recognizing the samples specified in `testlist.txt`. The results of testing are written to the results file `results.txt`, created in the current directory.

Evaluating recognition performance

lipi-core-toolkit provides the evaluation tool for assessing and analyzing the recognition performance of a shape recognizer. The steps are as follows:

- Go to the scripts directory

```
cd <lipi-core-toolkit install directory>/scripts
```

- Execute the evaluation tool using the results obtained from testing

```
perl eval.pl -input <lipi-core-toolkit install
directory>/bin/results.txt
```

The evaluation tool computes the recognition accuracy and prints to the output `stdout` or file if specified, and generates HTML pages corresponding to confusion matrices etc in the current directory.

Notes:

In the above example, we used the supplied **default** profile. Additional profiles may be created that use other recognition modules, other parameters in `nn.cfg`, or other datasets for training and testing.

The best profile for a particular problem may be arrived at by a process of benchmarking recognition accuracy using the different profiles.

Since the Training-Test-Evaluation cycle may be repeated multiple times in the course of developing or tuning a recognizer (perhaps with multiple profiles), you may also use the utility script benchmark.pl that internally runs these steps in succession:

- Go to the scripts directory

```
cd <lipi-core-toolkit install directory>/scripts
```

- Execute the benchmark script

```
perl benchmark.pl -project numerals
                  -train $DATA_ROOT/training.lst
                  -test  $DATA_ROOT/test.lst
```

Packaging the shape secognizer

Once the recognizer has been trained, tested, evaluated and found to be satisfactory, it can be integrated into sample applications on the same machine or it may be packaged for deployment on a new machine that has lipi tool kit already installed. The recognizer can be packaged using the script createAddOn.pl. Execute the following command from command line to create recognizer package.

Go to 'scripts' directory in $LIPI_ROOT and execute following command

createAddOn.pl -project < name of trained project available in $LIPI_ROOT 'projects' directory >

-logicalname <logical name for the project for eg. SHAPEREC_NUMERALS>

IMPORTANT: In case LipiDesigner is used to create the shape recognizer, the recognizer can be packaged using the save option in the user interface of LipiDesigner.

NOTE: Packaging script createAddOn.pl supports only SHAPEPEC projects and not WORDREC projects.

14-3 Integrating the shape recognizer with a sample application on same machine

To integrate the new shape recognizer with sample applications on the same machine the shape recognizer project name and a suitable logical name must be added to lipiengine.cfg as shown in figure. After this the shape recognizer becomes available for integration into sample applications. Please refer to the sample apps section 13-2 to get more information about developing a shape recognition based application.

```
LogFile=project_lipi.log
LogLevel=DEBUG
KANNADA_CHAR = kannada_char(default)
```

14-4 Integrating the shape recognizer with a sample application on client machine

In this section, we will look at how to integrate the shape recognizer created using lipi-core-toolkit into a pen-based application. For the purposes of illustration, we will assume that the numerals recognizer has been packaged as per the previous section.

The steps involved are as follows:

- Installing the packaged recognizer(s) on the client machine
- Integrating the recognizer into a sample application

The following section describes each step in detail.

Installing the package on the client machine

NOTE: Lipi Toolkit should be installed on client machine

The **client machine** is the computer on which we intend to use the recognizer by integrating it into a (pen-based) application and is typically different from the machine on which the recognizer was developed.

NOTE: The package contains only configuration and data files needed for the recognizer to function on the target machine.

Copy the recognizer package (`num_reco-packagename.zip`) on onto the client machine. Go to 'scripts' directory in $LIPI_ROOT and execute following command

installAddOn.pl –file <name of zip file containing packaged project for eg.

`num_reco-packagename.zip>`

Zip file has to be in same directory as of installAddOn.pl script file.

Integrating the recognizer into an application

Essentially, once installed, the numerals recognizer becomes available for integration into applications. The installed recognizer can be used by specifying the logical name, **SHAPEREC_LOGICALNAME**, (This is the name that was specified with –logicalname option while creating the package using createAddOn.pl)

Please refer to the sample apps section 13-2 to get more information about developing a shape recognition based application.

This section 13-4 also gives information about capturing digital ink corresponding to a shape made on a digitizer or using the mouse or from a file, passing the ink to the recognizer, and getting back the most plausible shape IDs and corresponding confidence values.

15 Creating and using a word Recognizer

lipi-core-toolkit 4.0 supports a limited form of word recognition – that of boxed fields of characters (shapes), via the included Boxed-Field recognizer. The Boxed-Field recognizer is a wrapper around the shape recognizer for recognizing isolated shapes, but has a different "word recognition" API that takes an entire field of ink as input, and returns strings as the output.

In this chapter, we will walk through the steps in creating a boxed field recognizer for numeric fields (digit strings), which might be applied for instance in a form filling application. We will assume that a shape recognizer for isolated numerals has already been created in the project demonumerals, as detailed in section 14-2 .

15-1 Creating a Boxed-Field recognizer for numeric fields

The effort of creating a Boxed-Field recognizer is essentially the effort of creating and evaluating the corresponding isolated shape recognizer. Once the latter has been accomplished, the steps involved in creating a Boxed-Bield recognizer are limited to:

- Setting up a word recognition project
- Building the project

Let us look at these steps in turn. We will assume here that a shape recognizer for isolated numerals is already available at `<lipi-core-toolkit install directory>/projects/demonumerals`. We will use the shorthand `$SHAPEREC_PROJ_ROOT` to refer to this directory.

Setting up the word recognition project

The steps involved in creating a new word recognition project are similar to those for a shape recognition project:

- Create a new project directory

  Create a new project directory, `num_fld`, for the recognizer under the projects directory `<lipi-core-toolkit install directory>/projects`, henceforth referred to as $PROJ_ROOT.

- Create a project configuration file

  Create a project configuration file `$PROJ_ROOT/<project_name>/config/project.cfg` with the following contents:

```
ProjectName = "Numeric Field Recognizer"
ProjectType = WORDREC
```

Note that, the configuration file identifies the project as a word recognition project, as opposed to a shape recognition project. Also, information about the number of shapes is NOT included.

- Create a default profile directory

  Create a default profile directory at `$PROJ_ROOT/<project_name>/config`, and a profile configuration file `$PROJ_ROOT/<project_name>/config/default/profile.cfg` with the following contents:

  ```
  WordRecognizer = boxfld
  RequiredProjects =demo numerals
  ```

  Note that the profile configuration specifies **boxfld** as the word recognition module to be used for the word recognition problem. This allows the possibility of specifying alternative word recognition module, e.g., capable of handling numeral sequences written without boxes.

  The profile configuration also specifies the project **demonumerals** as a prerequisite for the **num_fld** project.

- Copy the `boxfld.cfg` to default profile directory

  Copy the configuration file for the boxfld word recognizer, `boxfld.cfg`, from `$LIPI_ROOT/docs` to `$PROJ_ROOT/config/default`. The default contents of the file are as follows and may be modified as needed.

  ```
  MaxBoxCount= 30
  BoxedShapeProject = demonumerals
  BoxedShapeProfile = default
  ```

  The boxfld recognizer requires a shape recognizer project and profile in order to function, and these are specified in its configuration file. Here, MaxBoxcount refers to the maximum number of characters in the boxed field.

Please refer to the sample apps section 13-3 to get more information about developing a word recognition based application.

16 Appendix

16-1 Setting environment variables in Linux

In case of Linux, set the environment variable using the appropriate shell command:

```
export LIPI_ROOT=/home/testusers/lipi
```

16-2 Perl for Windows

Download Perl from the following link:

http://www.activestate.com/Products/ActivePerl/

For install instructions, follow the link:

http://aspn.activestate.com/ASPN/docs/ActivePerl/install.html

16-3 Default config file nn.cfg

nn.cfg is bundled with toolkit installer and is available at:

$LIPI_ROOT/docs

16-4 Default config file activedtw.cfg

activedtw.cfg is bundled with toolkit installer and is available at:

$LIPI_ROOT/docs

16-5 Default config file neuralnet.cfg

neuralnet.cfg is bundled with toolkit installer and is available at:

$LIPI_ROOT/docs

16-6 Sample ink file for runwordrec

An ink for word recognition must have the following attributes

1. The .HIERARCHY tag should identify it as a WORD CHARACTER, as opposed to CHARACTER for a shape recognition ink file.
2. The stroke indices and the truth corresponding to them should also be written in the ink file.

Example:

```
.SEGMENT CHARACTER 0 GOOD "0 0 __"
.SEGMENT CHARACTER 1,2 GOOD "1 0 __"
.SEGMENT CHARACTER 3 GOOD "2 0 __"
```

The above mentioned lines specify that

1. The truth associated with the character with stroke index 0 is class 0
2. Stroke index 1 and 2 correspond to class 1
3. Stroke index 3 corresponds to class 2

```
.VERSION 1.0 0
.HIERARCHY WORD CHARACTER
.COORD X Y T
.SEGMENT WORD
.X_POINTS_PER_INCH 2500
.Y_POINTS_PER_INCH 2500
.POINTS_PER_SECOND 1200
.SEGMENT CHARACTER 0 GOOD "0 0 __"
.SEGMENT CHARACTER 1,2 GOOD "1 0 __"
.SEGMENT CHARACTER 3 GOOD "2 0 __"
.PEN_DOWN
3814 182 17184
3816 181 0
3814 181 0
3594 298 0
...
...
...
3583 260 0
3583 260 0
.PEN_UP
.PEN_DOWN
```

```
3908 730 2513
3919 716 0
3936 694 0
3955 671 0
...
...
...
4423 1482 0
4431 1475 0
4433 1466 15632
.PEN_UP
.PEN_DOWN
3754 1667 2513
3766 1672 0
3787 1671 0
...
...
...
5010 1604 0
5105 1606 0
5184 1611 0
5229 1624 17605
.PEN_UP
.PEN_DOWN
3763 418 17184
3750 426 0
3735 432 0
...
...
...
4531 1473 0
4587 1473 25396
.PEN_UP
```

16-7 Sample list file for train/test

```
<Data-files-path>/usr0/000t01.txt 0
<Data-files-path>/usr0/000t02.txt 0
<Data-files-path>/usr1/000t01.txt 0
…
<Data-files-path>/usr27/001t01.txt 1
<Data-files-path>/usr27/001t02.txt 1
…
```

16-8 Sample list file for adapt

```
<Data-files-path>/usr118/017t02.txt 17 #
<Data-files-path>/usr125/007t01.txt 7 #
<Data-files-path>/usr130/001t02.txt 1 #
<Data-files-path>/usr142/037t02.txt 37 #
<Data-files-path>/usr143/032t01.txt 32 #
<Data-files-path>/usr138/044t02.txt 44 #
…
<Data-files-path>/usr139/035t01.txt 35
<Data-files-path>/usr143/003t01.txt 3
…
…
```

Above sample the lines with a # in the end are initial prototypes that are added to the model before adaptation.

16-9 Configurable make settings for Linux

In the case of Linux, the configurable options are specified in `global.mk`

Configurable options	Remarks
CC=g++	GNU compiler used for compiling cpp files

LINKLIB=-ldl –lc	Add any standard required libraries, e.g.–lm links math library
CFLAGS = -c	To compile in debugging mode add -g to CFLAGS
SHFLAGS=-shared –fpic	Flags required to build .so shared object

Table 37: Configurable make settings for Linux

16-10 Module dependencies on Windows

Module	Dynamic Libraries (DLLs)	Static Library
NN/ ActiveDTW/ Neural network	1 nn.dll/ activedtw.dll/ neuralnet.dll 2 preproc.dll 3 pointfloat.dll/npen.dll/l7.dll/substroke.dll (Feature extractor dll, based on configuration) 4 lipiengine.dll 5 logger.dll	1 common.lib 2 utils.lib 3 shapereccommon.lib 4 featureextractorcommon.lib
Boxfld	1 boxfld.dll 2 nn.dll /activedtw.dll/neuralnet.dll (Shape recognizer dll, based on configuration) 3 preproc.dll 4 pointfloat.dll/npen.dll/l7.dll/substroke.dll (Feature extractor dll, based on configuration) 5 lipiengine.dll 6 logger.dll	1 common.lib 2 utils.lib 3 shapereccommon.lib 4 featureextractorcommon.lib 5 wordreccommon.lib

Table 38: Module dependencies for Windows

16-11 Module dependencies on Linux

Module	Shared Libraries (.so)	Static Library
NN/ ActiveDTW/ Neural network	1 libnn.so/ libactivedtw.so/ libneuralnet.so 2 libpreproc.so 3 libpointfloat.so/libnpen.so/libl7.so/libsubstroke.so (Feature extractor shared library, based on configuration) 4 liblipiengine.so 5 liblogger.so	1 libcommon.a 2 libutils.a 3 libshapereccommon.a 4 libfeaturextractor.a
Boxfld	1 libboxfld.so 2 libnn.so/libactivedtw.so/libneuralnet.so (Shape recognizer shared library, based on configuration) 3 libpreproc.so 4 libpointfloat.so/libnpen.so/libl7.so/libsubstroke.so (Feature extractor shared library, based on configuration) 5 liblipiengine.so	1 libcommon.a 2 libutils.a 3 libshapereccommon.a 4 libfeaturextractor.a 5 libwordreccommon.a

	6 liblogger.so	

Table 39: Module dependencies for Linux

16-12 Options file for the eval tool

```
Optionsfile.txt
# File containing the results
input=/home/user/lipitk/resultfile.txt

# Output file to write the accuracy <optional>
# If not mentioned the output is displayed on the screen
output=outputfile.txt

# Number of top errors, to be printed in the performance analysis matrix
# <optional>
# default value is 5
topErrors=1

# Number of images per row in the HTML file of each class <optional>
# default value is 10
images=12

# Number of topchoices to consider for the performance analysis
# <optional>
# default value is 1
top_choices=3
```

16-13 Feature extraction

PointFloatShapeFeatureExtractor extracts the following features from each point along the stroke trajectory:

- X dimension - The X-Coordinate of the point
- Y dimension - The Y-Coordinate of the point
- Sine theta – Sine of the angle between the line segment joining two adjacent points and the X-axis (Note: Though the value of sine theta ranges from [-1 1] the extracted value for this feature has been normalized to the range [0 10])
- Cosine theta – Cosine of the angle between the line segment joining two adjacent points and the X-axis (Note: Though the value of sine theta ranges from [-1 1] the extracted value for this feature has been normalized to the range [0 10])

- Pen up – This is true if the point is the last point in a trace; otherwise set to false.

NPenShapeFeatureExtractor extracts the following features from each point along the stroke trajectory:

- X dimension - The X-Coordinate of the point
- Y dimension - The Y-Coordinate of the point
- Cosine alpha - Cosine of the angle subtended by the line segment joining the neighboring points on either side with the x-axis
- Sine alpha - Sine of the angle subtended by the line segment joining the neighboring points on either side with the horizontal line.
- Cosine beta - Cosine of the angle formed by the line segments joining the point of consideration and its second neighboring points on either side.
- Sine beta - Sine of the angle formed by the line segments joining the point of consideration and its second neighboring points on either side.
- Aspect - Captures the aspect ratio of the bounding box containing the points in the 'vicinity' of the point under consideration
- Curliness - Curliness at a point gives the deviation of the points in the vicinity from the line joining the first and last point
- Linearity - Average squared distance between every point in the vicinity and the line joining the first and last point
- Slope - Cosine of the angle formed by the line joining the first and last point of the vicinity with the x-axis
- Pen Up - This is true if the point is the last point in a trace; otherwise set to false.

NOTE: Detailed descriptions of the above features along with the formulae for computing them may be found in the NPen++ paper [12].

SubStrokeShapeFeatureExtractor splits the strokes into sub-strokes based on the complexity of the trajectory and extracts the following features from each sub-stroke:

- $\text{Theta}_{0\text{-}4}$ – Angles between two adjacent points of the sub-stroke and the X-axis after resampling the sub-stroke to six points.
- X center of gravity – The mean of X-coordinate values of the sub-stroke normalized by the width of the character.
- Y center of gravity - The mean of Y-coordinate values of the sub-stroke normalized by the height of the character.
- Length - The length of the sub-stroke normalized by the height of the character.

NOTE: Detailed descriptions of the above features along with the formulae for computing them may be found in this paper [13].

L7ShapeFeatureExtractor extracts the following features from each point along the stroke trajectory:

- X dimension - The X-Coordinate of the point
- Y dimension - The Y-Coordinate of the point
- X first derivative - Normalized first derivative with respective to X.

- Y first derivative - Normalized first derivative with respective to Y.
- X second derivative - Normalized second derivative with respective to X.
- Y second derivative - Normalized second derivative with respective to Y.
- Curvature - Curvature of the stroke at the point.
- Pen Up - This is true if the point is the last point in a trace; otherwise set to false.

NOTE: Detailed descriptions of the above features along with the formulae for computing them may be found in these papers [16] and [17].

16-14 Shape recognition

Nearest-Neighbor classifier

The Nearest-Neighbor (NN) classifier implements the standard Nearest Neighbor algorithm for shape recognition. Nearest Neighbor algorithm is a method of classifying the test sample based on the closest training samples in the feature space. The training method of the NN classifier exposed by the shape recognizer implements two prototype selection algorithms one based on Hierarchical Clustering (HC) and the other based on Learning Vector Quantization (LVQ). Dynamic Time Warping (DTW) is used as the measure of similarity between the character samples. The recognition method provides Euclidean filter for pruning candidate neighbors based on Euclidean distance in order to reduce the number of prototypes to be considered for relatively expensive DTW computation. The algorithm also supports adaptation where in the prototypes are morphed gradually to learn the style of the writer.

ActiveDTW classifier

Active DTW is a classifier for shape recognition [14]. The classifier's training method creates models for each class known as Active Shape Models. Each of these Active Shape Models consists of cluster models and singleton prototypes. The clusters are formed based on the Hierarchical clustering module bundled with Lipi toolkit. The most significant Eigen Values, Eigen Vectors of the cluster, the cluster mean and the number of samples in a cluster forms the cluster model. Testing comprises of computing the nearest neighbor of the new test sample with DTW distance as the measure of similarity. The similarity measure between a test sample and a singleton prototype of a class is the DTW distance. Computation of a similarity measure between the test sample and a cluster model involves constructing an optimal deformation which is closest to the test sample, according to the ActiveDTW algorithm. The distance of a test samples from a class is considered as the minimum of the two distances. A Euclidean filter, similar to the one provided in NN is available to prune the candidate neighbors based on the Euclidean distance in order to reduce the computation time.

ActiveDTW classifier also provides a framework to adapt the shape models created by Active-DTW classier during training [15], using new labeled samples encountered during testing. Incrementally adapting a model of class to a new sample requires computation of the modified model from the already learned model and the test sample. Based on whether the test sample is closer to model of a cluster or to a free sample of the class, either the model corresponding to the cluster or the set of free samples is modified.

Neural Network classifier

The Neural Network classifier implements the standard Multi Layer Perceptron for shape recognition. It uses the Back Propagation (BP) algorithm for training.

All the above shape recognizers expose a standard shape recognition API, which allows the recognizer to be trained, and invoked for recognition with a TraceGroup (group of traces) corresponding to a single or multi-stroke shape or character.

16-15 References

[1] Sriganesh Madhvanath, Deepu Vijayasenan and Thanigai Murugan Kadiresan, "Lipitk: A Generic Toolkit for Online Handwriting Recognition," Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition, La Baule, France, Oct 2006.

[2] The Carnegie Mellon Sphinx Project, http://cmusphinx.sourceforge.net

[3] The Festival Speech Synthesis System, http://www.cstr.ed.ac.uk/projects/festival

[4] Shanmugham, S., Monaco, P. and B. Eberman, "MRCP: Media Resource Control Protocol," Internet Draft draft-shanmugham- mrcp-05, January 2004

[5] Rosetta – Multistroke/Full Word Handwriting Recognition for X, http://www.handhelds.org/project/rosetta

[6] XStroke: Full-screen Gesture Recognition for X,

http://www.usenix.org/events/usenix03/tech/freenix03/full_papers/worth/worth_html/xstroke.html

[7] WayV Project, http://www.stressbunny.com/wayv

[8] UNIPEN 1.0 Format Definition, http://www.unipen.org/pages/5/index.htm

[9] InkML – The Digital Ink Markup Language, www.w3.org/2002/mmi/ink

[10] Mudit Agrawal, Kalika Bali, Sriganesh Madhvanath, Louis Vuurpijl, "UPX: A New XML Representation for Annotated Datasets of Online Handwriting Data," 8th International Conference on Document Analysis and Recognition, Seoul, Korea, Aug 29 - Sept 1, 2005.

[11] HRE API: A Portable Handwriting Recognition Engine Interface, http://playground.sun.com/pub/multimedia/handwriting/hre.html

[12] S. Jaeger, S. Manke, J. Reichert, and A. Waibel, "Online handwriting recognition: The NPen++ recognizer," International Journal on Document Analysis and Recognition, vol. 3, no. 3, pp. 169–180, Mar. 2001

[13] Parui, S.K., Guin, K. Bhattacharya, U., Chaudhuri, B.B., "Online handwritten Bangla character recognition using HMM," International Conference on Pattern Recognition, pp. 1-4, Dec.2008

[14] Muralikrishna Sridhar, Dinesh Mandalapu, Mehul Patel, "Active-DTW : A Generative Classifier that combines Elastic Matching with Active Shape Modeling for Online Handwritten Character Recognition," International Workshop on Frontiers in Handwriting Recognition, 2006

[15] Vandana Roy, Sriganesh Madhvanath, Anand S., Ragunath R. Sharma, "A Framework for Adaptation of the Active-DTW Classifier for Online Handwritten Character Recognition," International Conference on Document Analysis and Recognition, pp. 401-405, 2009

[16] M.Pastor, A. Toselli, and E.Vidal, "Writing Speed Normalization for On-Line Handwritten Text Recognition," International Conference on Document Analysis and Recognition, 2005

[17] Jagadeesh Babu V , Prasanth L , Raghunath Sharma R , Prabhakara Rao G. V , Bharath A , "HMM-based Online Handwriting Recognition System for Telugu Symbols," International Conference on Document Analysis and Recognition, 2007

17 Glossary

Project	lipi-core-toolkit parlance for a grouping of recognizer configurations, targeted at a particular shape or word recognition problem.
Profile	Specific set of configuration files associated with, and generally addressing a specific aspect of, a particular Project. Specific Profiles of the same Project may be created for specific writers, specific datasets, and so forth.
Lipitk	**Lipi T**ool**k**it
HWR	**H**and-**W**riting **R**ecognition
DLL	**D**ynamic **L**ink **L**ibrary - On Windows platforms, a library linked dynamically as needed
SO	**S**hared **O**bject (Linux) - On Linux platforms, a library linked dynamically as needed
Stroke	The sequence of pen points between two consecutive pen events, pen down and pen up
Tar	A file compression format and utility generally found on UNIX platforms; the act of compressing a file using this utility
Untar	A utility for uncompressing files compressed using tar, generally found on UNIX platforms; the act of uncompressing a tar'd file using this utility
tarball or tar file	A file in the tar format, generally a compressed collection of files
UNIPEN 1.0	A standard format from the International Unipen Foundation (www.unipen.org) to store on-line handwriting data (as digital ink) and its annotations.

18 Acknowledgement

We would like to thank IRESTE, University of Nantes (France) for providing the handwriting database termed IRONOFF. The pre-built recognizers are trained using the IRONOFF data. For more details see http://www.infres.enst.fr/~elc/GRCE/news/IRONOFF.doc