User Manual

Lipi Core Toolkit

lipi-core-toolkit 3.0.0

Contents

1 Lipi core toolkit Overview

Lipi core toolkit is a generic toolkit whose aim is to facilitate development of online handwriting recognition engines for new scripts, and simplify integration of the resulting engines into real-world application contexts. The toolkit provides robust implementations of tools, algorithms, scripts and sample code necessary to support the activities of handwriting data collection, training and evaluation of recognizers, packaging of engines 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 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 engines 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 engines for new scripts. Finally, for the application developer, it should allow easy integration of engines 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 HWR engine for a new script, while addressing the challenges described. The following figure (Figure 1) describes the various categories of potential users of the toolkit.

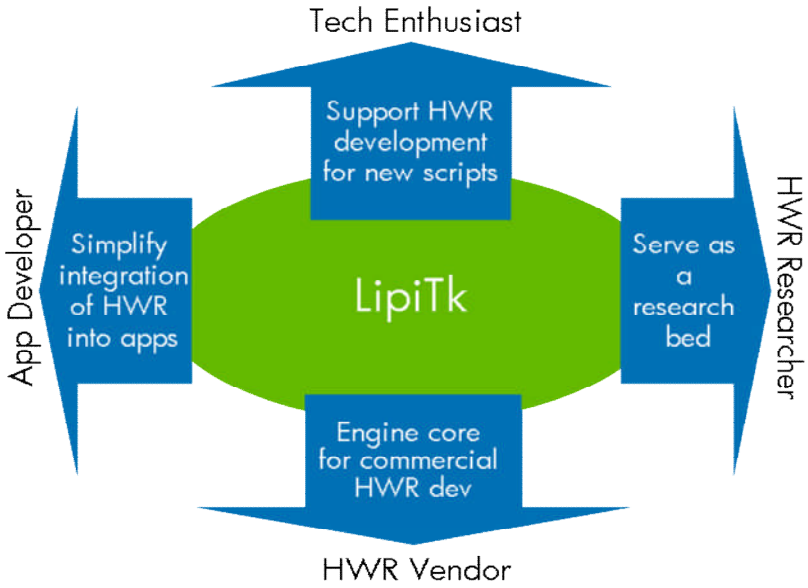


Figure 1: Lipi core toolkit users

There are three distinct stages in the lifecycle of the toolkit.

First Stage:

Standard tools and algorithms are packaged into a "downloadable toolkit".

Second Stage:

An intermediate user (such as a researcher or vendor) downloads the toolkit, experiments with it and potentially modifies the algorithms or adds new ones in order to build a custom engine for a new script and/or application context.

Third Stage:

An end-user (such as the application developer) integrates the built engine into a pen-based application.

The three stages can clearly be separated temporally and spatially and involve very different sets of users. While designing the toolkit, emphasis has been put primarily on the first stage, while ensuring that the toolkit supports the second and third stages in the hands of the respective user groups.

The current versions of the core toolkit support the recognition of isolated handwritten shapes and characters, as well as boxed words, captured as digital ink. The components of the toolkit make it

possible to carry out all the steps involved in building a character recognition engine. These components are described in section 1-3 Lipi core toolkit architecture. But first we take a look at some of the salient features of the toolkit in the light of previous efforts.

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 recognition engines. This implies tools for handwriting data collection standard script-independent algorithms for preprocessing and feature extraction, algorithms for training and pattern classification, and tools for subsequent evaluation and error analysis.

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 a minimum of 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 engines created using the toolkit into real-world applications.

lipi-core-toolkit also specifies standard shape and word recognition interfaces for recognition engines. All engines built using the toolkit hence expose a standard interface, simplifying their integration into pen-based applications. The shape recognition API is less ambitious in some aspects than previous efforts [11], but is distinct from them in that it includes training and online adaptation of shape recognizers.

1-3 Lipi core toolkit architecture

The Lipi core toolkit is designed to support Windows, Linux and Windows mobile 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.

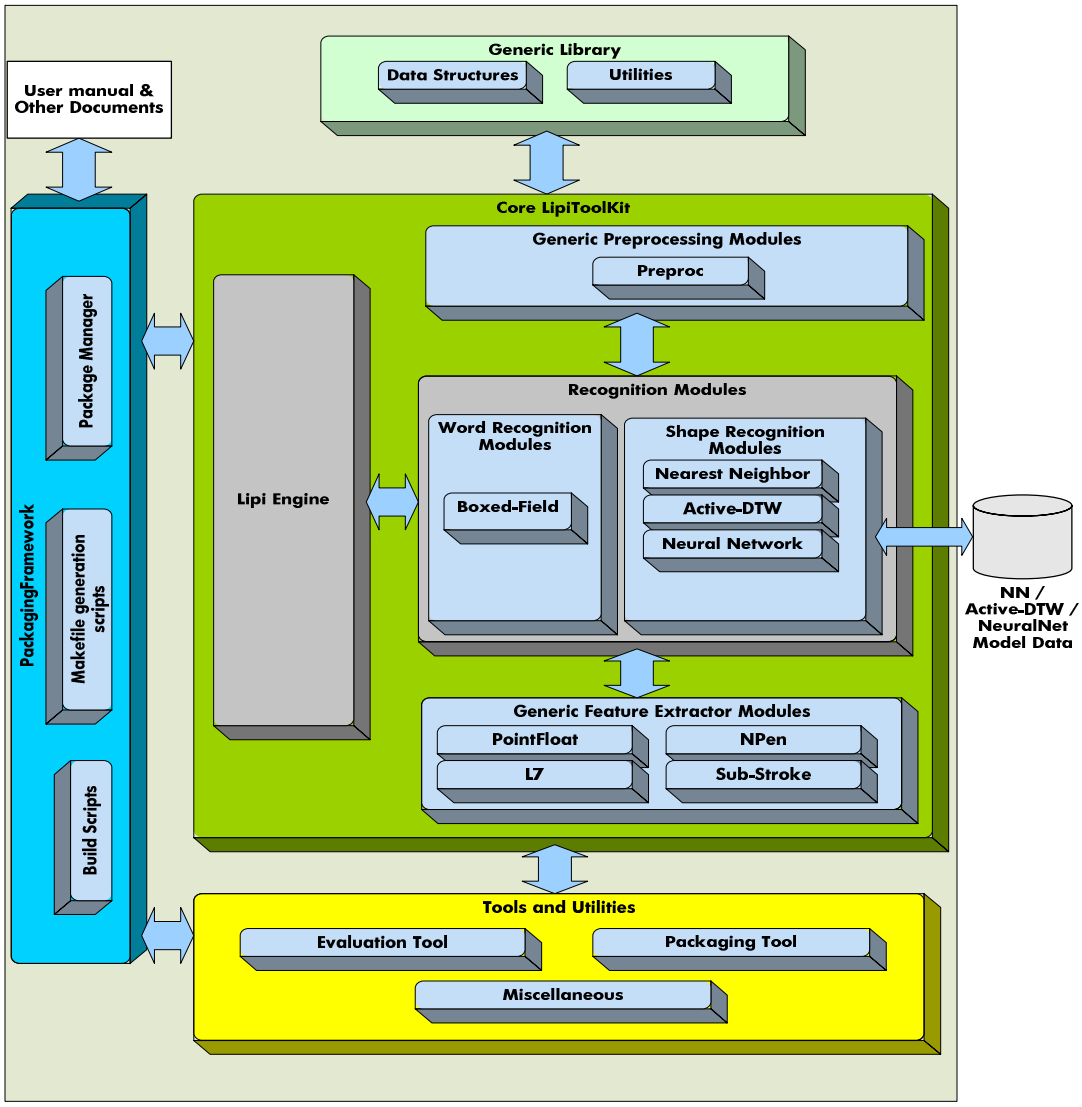


Figure 2 : Lipi core toolkit Architecture and Components

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 3.0.0* 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 3.0.0* are

- PointFloatShapeFeatureExtractor
- NPenShapeFeatureExtractor
- SubStrokeShapeFeatureExtractor
- L7ShapeFeatureExtractor

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].

1-3-4 Recognition Modules

1-3-4-1 Shape recognition

The shape recognition algorithms bundled with *lipi-core-toolkit 3.0.0* are

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

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.

1-3-4-2 Word recognition

The word recognition algorithm bundled with *lipi-core-toolkit 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 3.0.0 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
- package.pl - packaging the trained recognizers

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 Working with the toolkit

In this section, we take a brief look at the use of the toolkit to develop a recognizer for a set of shapes, for example, the 10 English numerals 0 to 9.

The first step in the process is the creation of a new shape recognition project, which we will call **numrec**. The associated project configuration file identifies the project as a shape recognition project (as opposed to a word recognition project), and the number of shape classes to be recognized as 10.

Within the **numrec** project, one or more profiles may be defined. Each profile contains configuration settings for the recognizer corresponding to a particular user-defined mode of operation or "flavor" of the recognizer. Profiles may be used to distinguish settings for writer-independent recognition vs.

specific writers, specific training data (e.g. US versus UK writers), specific recognition module (NN, ActiveDTW, Neural Network or Custom), specific preprocessing settings, and so on – at the discretion of the user. The profile also stores the results of training the shape recognizer (recognizer-specific model data) using the specific set of configuration settings.

The Perl script genmake.pl (Genmake.pl – Makefile generator) generates the makefile for a given *lipi-core-toolkit* project, which has the build instructions for the required source files. After generating the makefile, the user can build the project and all the dependent modules by executing the appropriate make command. For instance, if the default profile for the **numrec** project specifies "nn" as the recognizer, and NN's configuration file in turn specifies a sequence of preprocessing operations, building the project and profile would cause the generic preprocessing as well as the NN code to be built into dynamic link libraries. Building the *lipi-core-toolkit* project also builds a command line utility runshaperec (runshaperec tool) that links with these libraries and supports training and testing of the shape recognizer on labeled numeral data. (Note: The above process of using Genmake.pl and building need not be carried out if the binary package of the *lipi-core-toolkit* was downloaded.) The result file produced by testing may be provided as input to the evaluation tool (Evaluation tool) to compute the recognition accuracy and confusion matrix.

Once the cycle of training and testing is completed, the packaging script (Package.pl – packaging script) may be used to package the **numrec** project and the specified profiles into a self-extracting archive file or tar gzip file for deployment on the target machine where the pen-based application is going to be deployed.

1-5 Summary

In summary, the Lipi core toolkit aims to facilitate development of online handwriting recognition engines for new scripts, and simplify integration of the resulting engines 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 3.0.0 has been tested on the following platforms:

- Windows XP Professional Edition
- Redhat Enterprise Linux Edition 4.0 and
- Ubuntu Gutsy Gibbon 7.10
- Windows Mobile 6.0

2-3 Disk space requirements

lipi-core-toolkit 3.0.0 provides source and binary packages for Windows, Linux and Windows Mobile. Separate source package is provided for Windows and Linux, of size 1.28 MB. In the case of Windows, single source package is provided for VC6.0, VC2005, VC2008 and Windows Mobile. Whereas binary package, separate packages for Windows, Linux and Windows mobile, of size 1.28 MB. In the case of Windows, separate binary packages are provided for VC6.0, VC2005 and VC2008.

The space required to extract the source package is 4MB and binary package is 4MB. To build the source package after extracting you need 80MB of free space.

2-4 Software requirements

Item and Description	Windows XP Profession Edition	Windows Mobile	Linux
Extracting the contents of the package	Cabarc sdk (Please refer to **Appendix** for download instructions)	Not applicable	tar
Building lipi-core-toolkit code	Microsoft Visual Studio 6.0 with SP6 or Microsoft Visual C++ 2005 / MsBuild for VC2005 or Microsoft Visual C++ 2008 / MsBuild for VC2008	Microsoft Visual C++ 2005 or MsBuild for VC2005	GCC 3.4.6-9 or above
Executing scripts	Perl 5.0 or above	Not applicable	Perl 5.0 or above

Table 1: Software requirements

3 Installing Packages

3-1 lipi-core-toolkit 3.0.0 packages

lipi-core-toolkit 3.0.0 is available in the form of binary packages for Windows, Linux and Windows Mobile, with separate packages for VC 6.0, VC2005 and VC2008 on Windows and also available in the form of source package for Windows and Linux, with single package for VC6.0, VC2005, VC2008 and Windows Mobile. Select the package for your development platform.

Platform	package
Windows XP Professional Edition [VC6.0]	**Binary:** lipi-core-toolkit3.0.0-winvc6.0.cab **Source:** lipi-core-toolkit3.0.0-src-win.cab
Windows XP Professional Edition [VC2005]	**Binary:** lipi-core-toolkit3.0.0-winvc2005.cab **Source:** lipi-core-toolkit3.0.0-src-win.cab
Windows XP Professional Edition [VC2008]	**Binary:** lipi-core-toolkit3.0.0-winvc2008.cab **Source:** lipi-core-toolkit3.0.0-src-win.cab
Windows Mobile	**Binary:** lipi-core-toolkit3.0.0-wm.cab **Source:** lipi-core-toolkit3.0.0-src-win.cab
Linux	**Binary:** lipi-core-toolkit3.0.0-linux.tar.gz **Source:** lipi-core-toolkit3.0.0-src-linux.tar.gz

Table 2: lipi-core-toolkit packages

3-2 Unpacking/Extracting the source package

Use the following command to extract the package contents into a directory of your choice:

Platform	Command /Steps
Windows XP Professional Edition [VC6.0]	cabarc.exe –p x lipi-core-tool3.0.0-src-win.cab
Windows XP Professional Edition [VC2005]	cabarc.exe –p x lipi-core-tool3.0.0-src-win.cab
Windows XP Professional Edition [VC2008]	cabarc.exe –p x lipi-core-tool3.0.0-src-win.cab
Windows Mobile	Extract Windows Mobile package in Windows XP professional Edition. cabarc.exe –p x lipi-core-tool3.0.0-src-win.cab
Linux	tar –xzvf lipi-core-toolkit3.0.0-src-linux.tar.gz

Table 3: Package extraction command

3-3 Unpacking/Extracting the binary package

Use the following command to extract the package contents into a directory of your choice:

Platform	Command
Windows XP Professional Edition [VC6.0]	cabarc.exe –p x lipi-core-tool3.0.0-winvc6.0.cab
Windows XP Professional Edition [VC2005]	cabarc.exe –p x lipi-core-tool3.0.0-winvc2005.cab
Windows XP Professional Edition [VC2008]	cabarc.exe –p x lipi-core-tool3.0.0-winvc2008.cab
Windows Mobile	Extract Windows Mobile package in Windows XP professional Edition. eg., cabarc.exe –p x lipi-core-toolkit3.0.0-wm.cab
Linux	tar –xzvf lipi-core-toolkit3.0.0-linux.tar.gz

Table 4: Package extraction command

3-4 Contents of Source package

After extracting the *lipi-core-toolkit 3.0.0* source package, the directory structure looks as follows

```
lipi-core-toolkit3.0.0 ($LIPI_ROOT)
   |
   |--- global.winmk
   |--- global.mk
   |--- readme.txt
   |--- license.txt
   |
   +---windows
   |     |
   |     +--- vc6.0
   |     |    |--- Makefile.win
   |     |
   |     +--- vc2005
   |     |    |--- lipitk.targets
   |     |
   |     +--- vc2008
   |     |    |--- lipitk.targets
   |     |
   |     +--- wm5.0
   |     |    |--- lipitk.targets
   |
   |
   +--- bin {application after modules build}
   |
   +--- doc
   |      |--- {lipitk-core documents}
   |
   +--- lib {dynamic library after modules build}
   |
   +--- package
   |      |--- package.cfg
   |
   +--- projects
   |      |
   |      +--- numerals
   |      |      +--- config
   |      |      |      |--- project.cfg
   |      |      |      |
   |      |      |      +--- l7_activedtw
```

```
|       |       |       |       |--- profile.cfg
|       |       |       |       |--- activedtw.cfg
|       |       |       |       |--- activedtw.mdt
|       |       |       |
|       |       |       +--- npen_neuralnet
|       |       |       |       |--- profile.cfg
|       |       |       |       |--- neuralnet.cfg
|       |       |       |       |--- neuralnet.mdt
|       |       |       |
|       |       |       +--- pointfloat_nn
|       |       |       |       |--- profile.cfg
|       |       |       |       |--- nn.cfg
|       |       |       |       |--- nn.mdt
|       |       |       |
|       |       |       +--- substroke_nn
|       |       |       |       |--- profile.cfg
|       |       |       |       |--- nn.cfg
|       |       |       |       |--- nn.mdt
|       |       |
|       |       +--- data
|       |       |       +--- raw
|       |       |       |       |--- usr[0-9] directories and inkfiles
|       |       |
|       |       |
|       +--- boxfld_numerals
|       |       +--- config
|       |       |       |--- project.cfg
|       |       |       |
|       |       |       +--- pointfloat_nn
|       |       |       |       |--- profile.cfg
|       |       |       |       |--- boxfld.cfg
|       |       |
|       |       +--- data
|       |       |       |--- wordtestlist.txt (list file)
|       |       |       |--- testword.txt (ink file)
|       |       |       |--- testwordsecond.txt (ink file)
|       |       |
+--- scripts
|       |--- benchmark.pl
|       |--- benchmarkadapt.pl
|       |--- eval.pl
|       |--- evaladapt.pl
|       |--- extracthwdata.pl
|       |--- genmake.pl
|       |--- imagewriter.pl
|       |--- lipitk.css
|       |--- listfiles.pl
|       |--- package.pl
|       |--- resultviewer.html
|       |--- trimlines.pl
|       |--- validatelistfile.pl
|       |
+--- src
|       +--- apps
|       |       +--- samples
|       |       |       |--- shaperectst
|       |       |       |--- wordrectst
|       |       |       |--- UIApp {Windows Mobile application}
|       |--- common
|       |--- include
```

```
|      |--- lib {static library after modules build}
|      |--- lipiengine
|      +--- reco
|      |      |--- util
|      |      |--- shaperec
|      |      |      |--- common
|      |      |      +--- featureextractor
|      |      |      |      |--- common
|      |      |      |      |--- npen
|      |      |      |      |--- pointfloat
|      |      |      |      |--- substroke
|      |      |      |--- preprocessing
|      |      |      |--- activedtw
|      |      |      |--- neuralnet
|      |      |      |--- nn
|      |      |     +--- wordrec
|      |      |      |      |--- common
|      |      |      |      |--- boxfld
|      |      +--- util
|      |      |      |--- featurefilewriter
|      |      |      |--- imgwriter
|      |      |      |--- lib
|      |      |      |--- mdv
|      |      |     +--- run
|      |      |      |      |--- runshaperec
|      |      |      |      |--- runwordrec
```

3-5 Contents of Binary package

After extracting the *lipi-core-toolkit 3.0.0* Windows VC6.0 binary package, the directory structure looks as follows

```
lipi-core-toolkit3.0.0 ($LIPI_ROOT)
   |
   |--- global.winmk
   |--- global.mk
   |--- readme.txt
   |--- license.txt
   |
   +--- bin
   |      |--- imgwriter.exe
   |      |--- mdv.exe
   |      |--- runshaperec.exe
   |      |--- runwordrec.exe
   |      |--- featurefilewriter.exe
   |
   +--- doc
   |      |--- {lipitk-core documents}
   |
   +--- lib
   |      |--- neuralnet.dll {not available in Windows Mobile package}
   |      |--- nn.dll
   |      |--- npen.dll
   |      |--- pointfloat.dll
   |      |--- preproc.dll
   |      |--- substroke.dll
   |      |--- activedtw.dll {not available in Windows Mobile package}
```

```
|      |--- boxfld.dll
|      |--- l7.dll
|      |--- lipiengine.dll
|      |--- logger.dll
|
+--- package
|      |--- package.cfg
|
+--- projects
|      |
|      +--- numerals
|      |      +--- config
|      |      |      |--- project.cfg
|      |      |      +--- l7_activedtw
|      |      |      |      |--- profile.cfg
|      |      |      |      |--- activedtw.cfg
|      |      |      |      |--- activedtw.mdt
|      |      |      |
|      |      |      +--- npen_neuralnet
|      |      |      |      |--- profile.cfg
|      |      |      |      |--- neuralnet.cfg
|      |      |      |      |--- neuralnet.mdt
|      |      |      |
|      |      |      +--- pointfloat_nn
|      |      |      |      |--- profile.cfg
|      |      |      |      |--- nn.cfg
|      |      |      |      |--- nn.mdt
|      |      |      |
|      |      |      +--- substroke_nn
|      |      |      |      |--- profile.cfg
|      |      |      |      |--- nn.cfg
|      |      |      |      |--- nn.mdt
|      |      |      |
|      |      +--- data
|      |      |      +--- raw
|      |      |      |      |--- usr[0-9] directories and inkfiles
|      |      |      |
|      |      +--- boxfld_numerals
|      |      |      +--- config
|      |      |      |      |--- project.cfg
|      |      |      |
|      |      |      |      +--- pointfloat_nn
|      |      |      |      |      |--- profile.cfg
|      |      |      |      |      |--- boxfld.cfg
|      |      |      |      |
|      |      |      +--- data
|      |      |      |      |--- wordtestlist.txt (list file)
|      |      |      |      |--- testword.txt (ink file)
|      |      |      |      |--- testwordsecond.txt (ink file)
|      |      |      |
+--- scripts
|      |--- benchmark.pl
|      |--- benchmarkadapt.pl
|      |--- eval.pl
|      |--- extracthwdata.pl
|      |--- genmake.pl
|      |--- imagewriter.pl
|      |--- lipitk.css
|      |--- listfiles.pl
|      |--- package.pl
```

```
|      |--- resultviewer.html
|      |--- trimlines.pl
|      |--- validatelistfile.pl
|      |
+--- src
|      +--- apps
|      |      +--- samples
|      |      |      +--- shaperectst
|      |      |      +--- wordrectst
|      |      |      +--- UIApp {Windows Mobile application}
|      +--- include
|      +--- lib
|      |      |--- common.lib
|      |      |--- featureextractorcommon.lib
|      |      |--- shapereccommon.lib
|      |      |--- utils.lib
|      |      |--- wordreccommon.lib
|      +--- util
|      |      +--- lib
```

3-6 Setting up the environment

Lipi-core-toolkit uses an environment variable called LIPI_ROOT as refrence for locating the dlls, root location of data etc. After extracting/unpacking the package, this environment variable LIPI_ROOT has to be set to the directory to which the package was extracted.

For example, if the package was extracted to the directory `d:\user1`, LIPI_ROOT should be set to "`d:\user1\lipi`".

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: Inorder 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 VC6.0

4-1-1 Setting up the environment

To build the modules on Windows for VC6.0, msdev and make executables must be included in system PATH variable. This can be done by executing `<vc++6.0 install dir>\VC98\Bin\VCVARS32.BAT` from the command prompt

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

Makefiles, `Makefile.win`, are provided for every module under directory `windows/vc6.0`. The master makefile is present at `<lipi-core-toolkit install directory>/windows/vc6.0/Makefile.win`. A Module can be prevented from building by removing the module name from the **all:** tag list in the master makefile.

Common platform-specific settings such as compiler, environment and linker are configured and stored in a global configuration file, `<lipi-core-toolkit install directory>/global.winmk`. Refer to the **Appendix**, for the configurable makefile settings.

To build *lipi-core-toolkit 3.0.0*, on Windows for VC6.0, execute the following command from `<lipi-core-toolkit install directory>/windows/vc6.0`

```
nmake /f Makefile.win
```

4-2 Building on Windows for VC2005

4-2-1 Setting up the environment

To build the modules on Windows for VC2005, devenv must be included in system PATH variable. This can be done by executing `<visual studio 2005 install dir>\ Common7\Tools\ vsvars32.bat` from the command prompt.

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

lipi-core-toolkit 3.0.0 provides the project files for every module under directory windows/vc2005. The master project file is present at <lipi-core-toolkit install directory>/windows/vc2005/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 VC2005 package by using the MsBuild.

To build lipi-core-toolkit 3.0.0, on Windows for VC2005, execute the following command from `<lipi-core-toolkit install directory>/windows/vc2005`

```
MsBuild lipitk.targets
```

4-3 Building on Windows for VC2008

4-3-1 Setting up the environment

To build the modules 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.

4-3-2 Building the lipi-core-toolkit components

lipi-core-toolkit 3.0.0 provides the project files for every module under directory windows/vc2008. The master project file is present at <lipi-core-toolkit install 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 3.0.0, on Windows for VC2008, execute the following command from `<lipi-core-toolkit install directory>/windows/vc2008`

```
MsBuild lipitk.targets
```

4-4 Building on Windows for Windows Mobile 6.0

4-4-1 Setting up the environment

- Open the properties of **"My Computer"** from Windows Explorer.
- Go to **"Advanced"** sheet from property dialog.
- Click on **"Environment variables"** button and **Edit** variable **Path**
- Insert ";C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727" at the last followed by ";"
- Click on OK button to apply settings

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

lipi-core-toolkit 3.0.0 provides the project files for every module under directory windows/wm5.0. The master project file is present at <lipi-core-toolkit install directory>/windows/wm5.0/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 Windows Mobile 6.0 package by using the MsBuild.

To build *lipi-core-toolkit* 3.0.0, on Windows for Windows Mobile 6.0, execute the following command from `<lipi-core-toolkit install directory>/windows/wm6.0`

```
MsBuild lipitk.targets
```

4-5 Building on Linux

4-5-1 Building the lipi-core-toolkit components

On Linux, the makefiles for every module are available under the directory `linux/` with name `Makefile.linux`. The master makefile is present at `<lipi-core-toolkit install 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 3.0.0*, on Linux, execute the following command from `<lipi-core-toolkit install directory>/linux`

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

5 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 **numerals**. The project root (**PROJROOT**) would be `<lipi-core-toolkit install directory>/projects/numerals`.

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 5: 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 also contains the shape recognizer specific configuration files, i.e. `nn.cfg`. Please refer to **Appendix** for default configuration file for nn. Other shape recognition modules like activedtw and neuralnet can also be mention in `profile.cfg`. But use the specific configuration files with respective to shape recognition mention in `profile.cfg`. 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 6: 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/shaperec`, e.g. nn.

7 lipiengine configuration

lipiengine

lipiengine is the controller that loads all the modules (logger, preprocessor, featureextractor, recognizer) required for a specific 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 optional configuration file. The 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 = numerals(working)*	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 7: 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 module. (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.

NOTE: There is no runshaperec tool for Windows Mobile version.

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. Please refer to **Appendix** for more details.
-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 8: 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.

NOTE: There is no runwordrec tool for Windows Mobile version.

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. NOTE: 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 9: 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 10: 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.

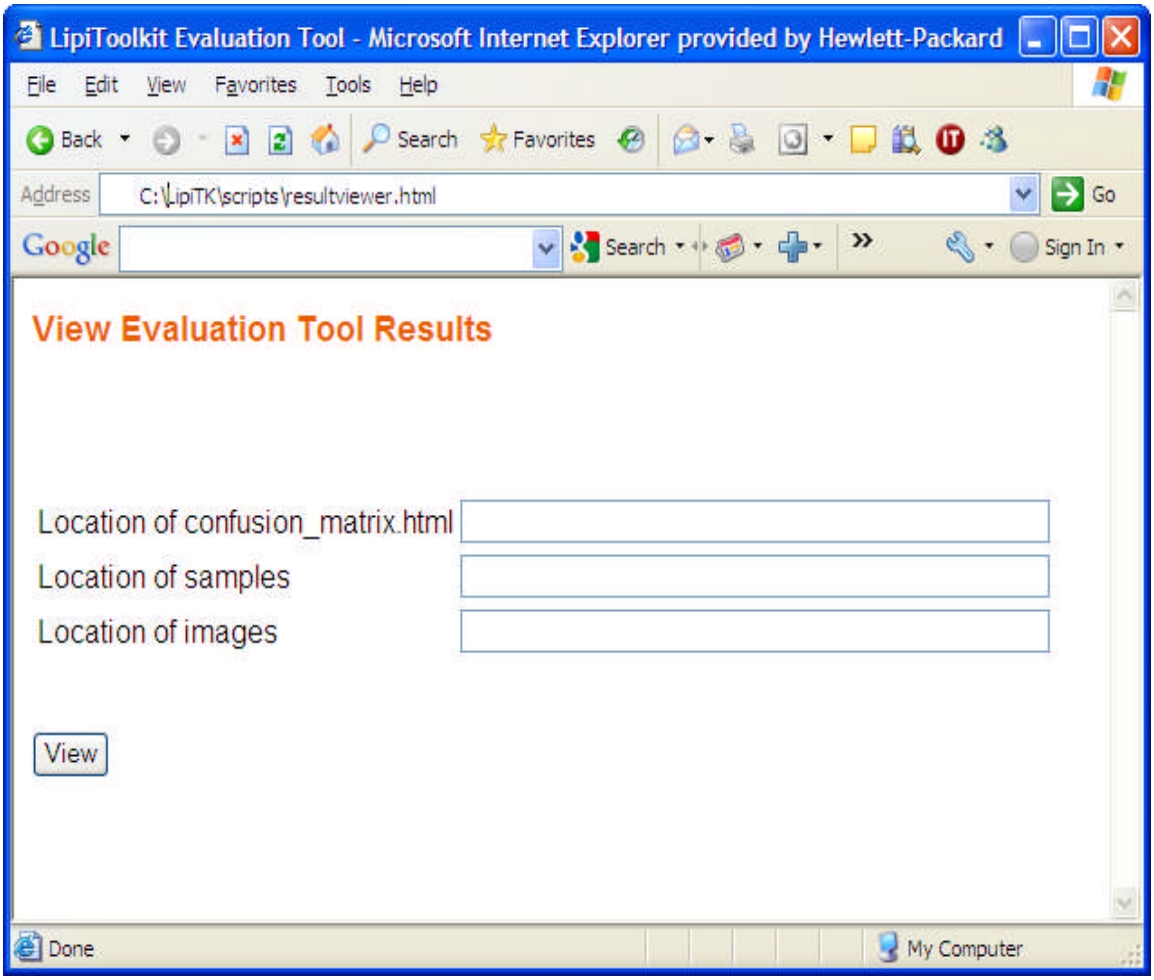


Figure 3: resultviewer.html

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.

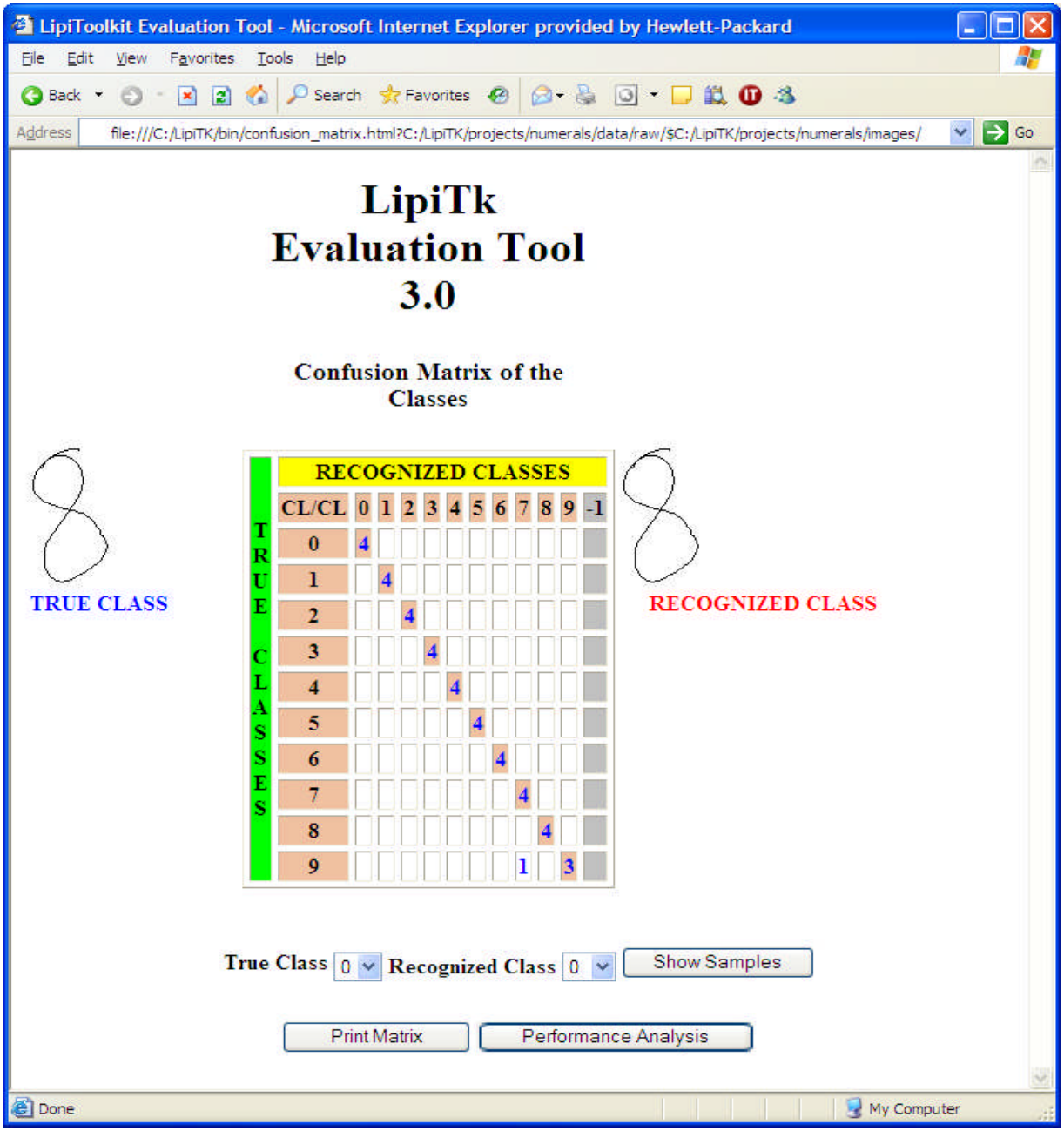


Figure 4 : confusion_matrix.html

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.

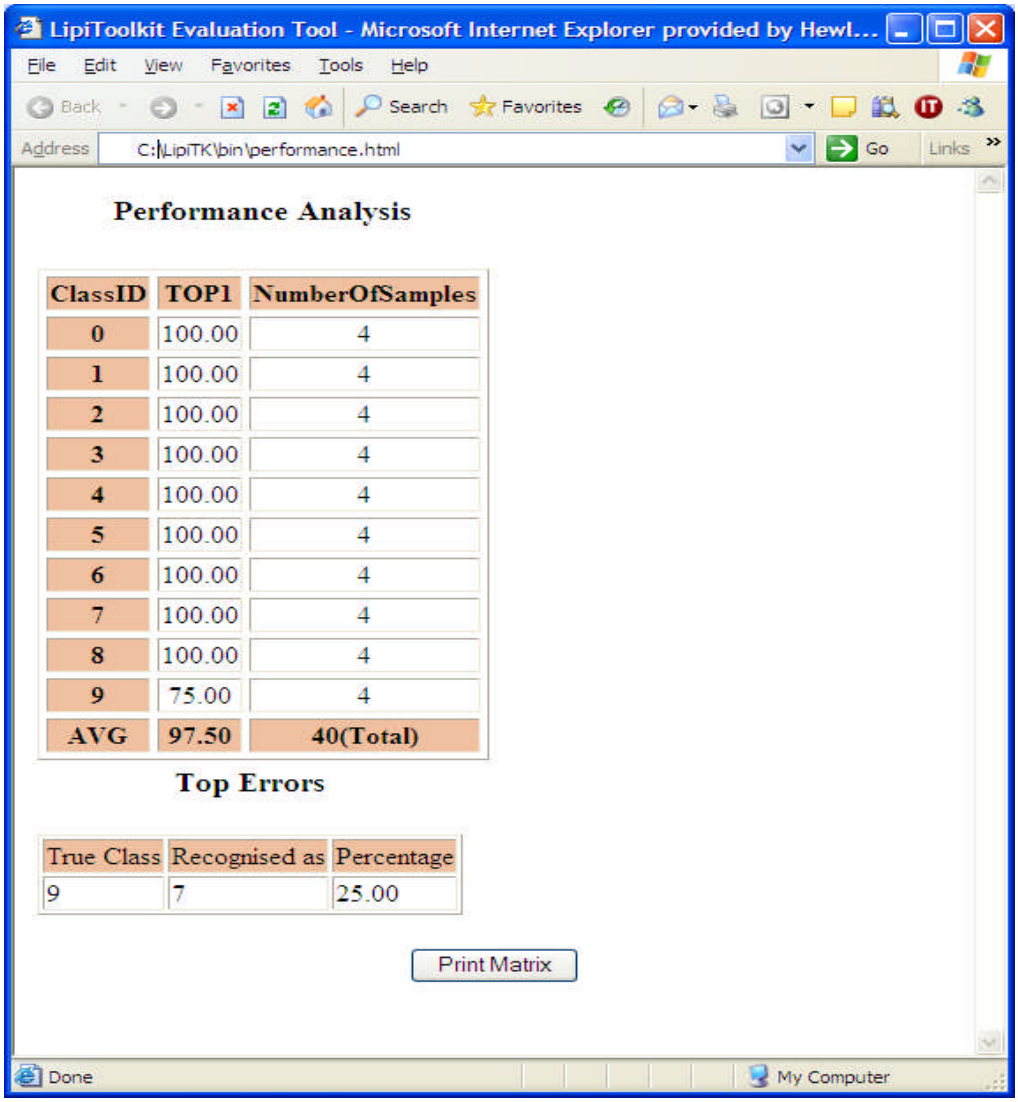


Figure 5 : performance.html

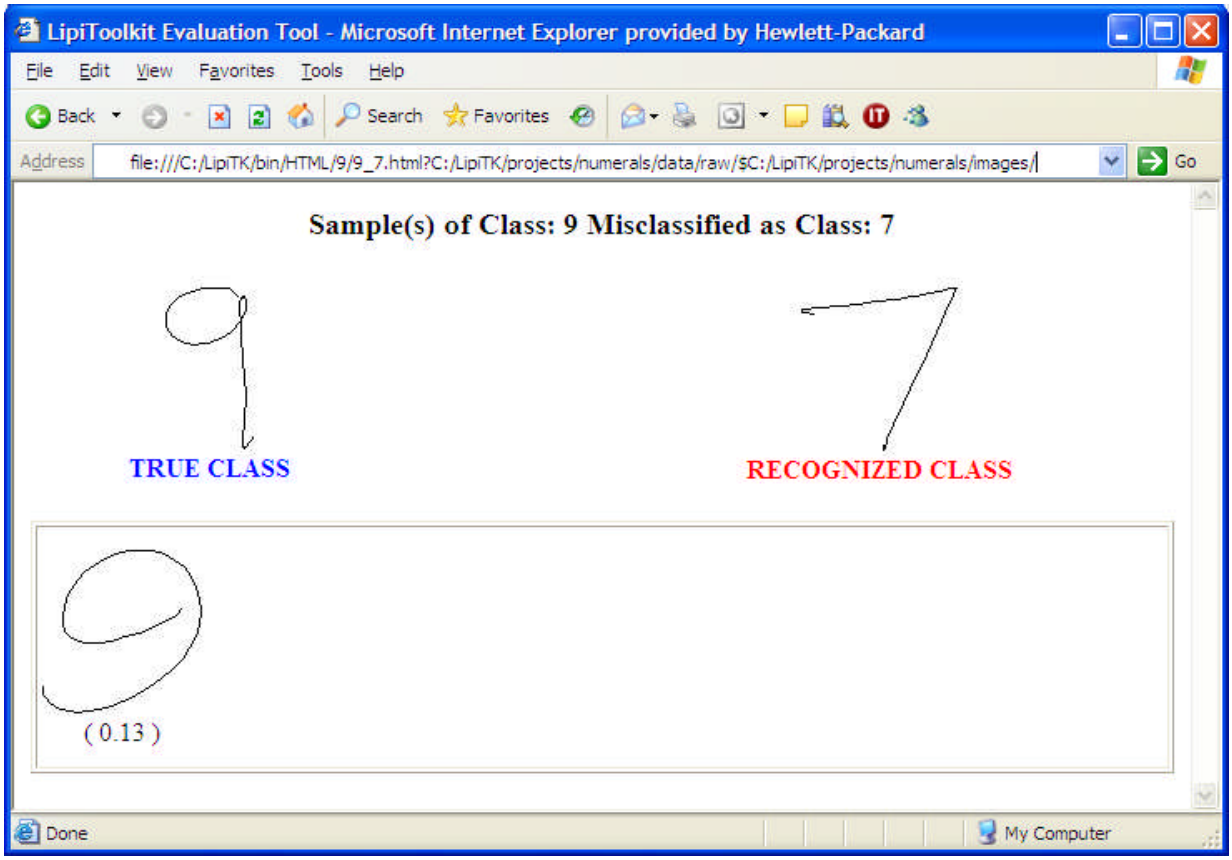


Figure 6 : Misclassified sample of class 1

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>
                    -output             <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.
-output	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 11: evalAdapt.pl Command line arguments

10 Genmake.pl – Makefile generator

Responsibilities

The Perl script, `genmake.pl`, generates the makefile for a given project, by determining all the dependent libraries and binaries required for that project and profile. The script is available under the directory `<lipi-core-toolkit install directory>/scripts`. The makefile is generated under the directory from which the script is executed.

Usage: genmake.pl

```
perl  genmake.pl
                  -project     <project 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	Name of the project for which the makefile is to be generated.
-profile	Optional	Profile to be included in makefile. NOTE: Default profile is assumed, if profile is not specified as a command line argument.
-lipiroot	Optional	Path of the 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 information.
-help	Optional	Displays usage information.

Table 12: genmake.pl command line arguments

Description

This script looks at the `profile.cfg` file for the modules to be built and generates the makefile, taking into consideration the dependent modules and the libraries required.

Example:

For a shape recognition project using Nearest Neighbor (i.e., nn) as a shape recognition module, the `profile.cfg` looks as follows:

```
ShapeRecMethod = nn
```

In this case, `genmake.pl` script generates a makefile to build NN and all the dependent modules.

For a word recognition project, the script generates the makefile for building the word recognition module as well as the shape recognition project specified under the **RequiredProjects** attribute of the `profile.cfg`. Consider a `profile.cfg` with the following contents

```
WordRecognizer = boxfld
RequiredProjects = numerals (default)
```

The script generates the makefile for the module **boxfld** and also all the modules required for the numerals project with the default profile.

After generating the makefile, the user can build the project and all the dependent modules by executing the following command

Platform	Command to clean the project	Command to build all projects and dependent libraries
Windows XP Professional Editions for VC6.0	nmake clean	nmake all
Linux	make clean	make all

Table 13: Command to build projects

After building the project

- Shared libraries are stored under `<lipi-core-toolkit install directory>/lib`
- Static libraries can be found under `<lipi-core-toolkit install directory>/src/lib` directory.
- The binaries are stored in `<lipi-core-toolkit install directory>/bin` directory.

The module dependencies for various shape recognizers are listed in **Appendix**.

NOTE: Genmake.pl does not generate any make file for building the VC2005, VC2008 and wm5.0 code.

11 Packaging

The packaging tool provides scripts for packaging the built engine(s) for deployment. The components of the package are fully user-configurable, and the packaging script creates a self-extracting package file (or tar.gz file for Linux), containing all the components selected by the user.

11-1 Package.pl – packaging script

Responsibilities

The `package.pl` script creates the package based on the configuration file, `<lipi-core-toolkit install directory>/package/package.cfg`, passed as a command line argument. It creates a package file (tar.gz for Linux and .cab for Windows) with all the components specified by the user. The script is available under the directory `<lipi-core-toolkit install directory>/scripts`.

Usage: package.pl

```
perl  package.pl
                  -pkg          <pkg config file >
                  -pkgname      <package name>
                  [-lipiroot]   <path of lipi-core-toolkit
                                install directory>
                  [-ver|v]
                  [-help]
```

Command line arguments

Command line argument	Argument Type	Description
-pkg	Mandatory	Path of the package configuration file. Refer to section 11-2 Customizing the package, for details.
-pkgname	Mandatory	Name of the final package.
-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 information.
-help	Optional	Displays usage information.

Table 14: package.pl command line arguments

Description

This script looks at the package configuration file for the modules to be packaged. On Linux, the script creates tar zip ball with an extension **.tar.gz**, whereas on Windows, the package extension is **.cab**. The root directory of all these packaged items is **lipi**.

The script names the final package as per the following conventions:

<package name>-<operating system-suffix>.<package extension>

Where, *package-name* is the name given by the user as a command line argument.

Example:

If the scripts is executed as

```
perl package.pl –pkg package.cfg -pkgname numerals
```

On Linux, the name of the final package would be **numerals-linux.tar.gz** whereas on Windows, the final package would be named as **numerals-winvc6.0.cab or numerals-winvc2005.cab or numerals-winvc2008.cab** based on the compiler used for building.

11-2 Customizing the package

The user can include any source directory, tools, scripts etc. into the package by configuring the `<lipi-core-toolkit install directory>/package/package.cfg`. In the following section, we describe the various attributes of the package configuration file.

Tools

This attribute holds a comma-separated list of tools to be included in the final package.

Example:

Tools = mdv, runshaperec, runwordrec

Src

This attribute allows the user to specify all the source directories as a comma separated list. The script also packages the dependent modules even if they are not explicitly specified.

Example:

Src = apps, common, include, lipiengine, reco, util

Projects

This attribute specifies the projects to be included in the final package. Multiple projects can be specified as a comma separate list.

To package a specific profile under a project, the project name must be followed by the profile name, enclosed within brackets. However, if the profile name is not specified, the script packages the default profile only. To package all the profiles available under the project, write **(all)** after the project name.

Example:

Projects = kannada_char, english_alpha(all), numerals(working)

In the above example, default profile is packaged for the project *kannada_char.* All the profiles available under project *english_alpha* are included in the final package. But for the numerals project, only the profile named *working* is packaged.

Logical names under [export] section

The *export* section in the `<lipi-core-toolkit install directory>/package/package.cfg` can be used to specify the logical names for the projects listed in the **Projects** attribute. The script writes these logical names to the `lipiengine.cfg`, available under the `<lipi-core-toolkit install directory>/projects` directory.

Scripts

This attribute specifies the name of the Perl scripts to be included in the final package. To specify more than one script, separate them with comma.

Example:

Scripts = eval.pl, extracthwdata.pl, listfiles.pl, trimlines.pl

Sample data

This attribute specifies location of the data directory, which should to be packaged with the recognizer. The specified path must be relative to `<lipi-core-toolkit install directory>`.

Example:

data = projects/numerals/data

TargetPlatform

This attribute allows the user to specify the target platform for the package. If the target platform is specified as Linux, the final package is created as a tar.gz file. For Windows, user can specify the TargetPlatform as vc6.0 to create a VC 6.0 package. Specifying the TargetPlatform as vc2005 creates a VC2005 package. Specifying the TargetPlatform as vc2008 creates a VC2008 package.

Example:

TargetPlatform =Linux

11-3 Creating a package using package.pl – an example

11-3-1 Assumptions

Consider the following `package.cfg`,

```
[package]
Projects = kannada_char, tamil_char(special)
Tools = mdv,runshaperec
Src = apps/samples/shaperectst

#for Windows '/' should be changed to '\'
Scripts = eval.pl,extracthwdata.pl,listfiles.pl
TargetPlatform =linux

[export]
KANNADA_CHAR = kannada_char(default)
TAMIL_CHAR = tamil_char(special)
```

11-3-2 Directory structure

After extraction, the directory structure of the package may look as follows depending on the configuration settings associated with project/profile:

```
lipi [$LIPI_ROOT]
  +--- bin
  |      |--- runshaperec.exe
  |      |--- mdv.exe
  +--- projects
  |      |--- lipiengine.cfg
  |      +--- kannada_char
  |      |      +--- config
  |      |      |      |--- project.cfg
  |      |      |      +--- default
  |      |      |      |      |--- profile.cfg
  |      +--- tamil_char
  |      |      +--- config
  |      |      |      |--- project.cfg
  |      |      |      +--- special
  |      |      |      |      |--- profile.cfg
  +--- src
  |      +--- include
  |      +--- apps
  |      |      +--- samples
  |      |      |      +--- shaperectst
  |      +--- lib
  |      |      |--- common.lib
  |      |      |--- util.lib
  |      |      |--- shapereccommon.lib
  |      |      |--- wordreccommon.lib
  |      |      |--- featureextractorcommon.lib
  +--- lib
  |      |--- nn.dll
  |      |--- lipiengine.dll
  |      |--- preproc.dll
  |      |--- pointfloat.dll
  +--- scripts
  |      |--- eval.pl
  |      |--- extracthwdata.pl
  |      |--- listfiles.pl
```

12 Scripts

lipi-core-toolkit 3.0.0 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.

12-1 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 15: trimlines.pl command line arguments

12-2 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 16: 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 17: Output file generated after executing extracthwdata.pl

12-3 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 18: listfiles.pl Command line arguments

12-4 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 19: validatelistfile.pl Command line arguments

12-5 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 20: benchmark.pl Command line arguments

12-6 benchmarkadapt.pl

Responsibilities

This script performs training, testing in adaptation mode 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, `result.txt`. This is used as an input for evaluating the adaptation accuracy.

Usage: benchmarkadapt.pl

```
perl  benchmarkadapt.pl
                        -project        <project name>
                        -indir          < root path of the dataset>
                        -prototypes     <Number of prototypes per
                                        class for adaptation>
                        -runs           <number of iterations
                                        required>
                        -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>
                        [-profile]      <profile name>
                        [-loglevel]     <log level:
                                        Debug|Error|info|all>
                        [-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`..
-indir	Mandatory	The result file generated by testing the shape recognizer using the runshaperec tool.
-prototypes	Mandatory	Number of prototypes that will be added to the model during adaptation.
-runs	Mandatory	Number of iterations required
-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.
-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.
-loglevel	Optional	This argument allows the user to specify the log level {*Debug*|*Error*|*info*|*all*}.
-ver|v	Optional	Displays version of the script.
-help	Optional	Displays usage information.

Table 21: benchmark.pl Command line arguments

12-7 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 22: imagewriter.pl Command line arguments

13 Utilities

13-1 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.

13-1-1 Build instructions

- Run command prompt
- Make sure that the $LIPI_ROOT environment variable is set to *lipi-core-toolkit* directory
- Go to `<lipi-core-toolkit install directory>/src/util/featurefilewriter` directory
- Go to linux or windows\vc6.0 or windows\vc2005 or windows\vc2008 based on your platform.
- Execute the commands in Table 23 based on your platform
- binary is available under the directory *<lipi-core-toolkit install directory>/bin*.

Platform	Package
Windows XP Professional Editions for VC6.0	1. <vc++6.0 install dir>\VC98\Bin\VCVARS32.BAT 2. nmake /f Makefile.win
Windows XP Professional Editions for VC2005	1. <visual studio 2005 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release imgwriter.vcproj
Windows XP Professional Editions for VC2008	1. <visual studio 2008 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release imgwriter.vcproj
Linux	make –f Makefile.linux

Table 23: Command for building featurefilewriter module

Usage: featurefilewriter

```
mdv
        -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 24: 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.

13-2 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.

13-2-1 Build instructions

- Run command prompt
- Make sure that the $LIPI_ROOT environment variable is set to *lipi-core-toolkit* directory
- Go to `<lipi-core-toolkit install directory>/src/util/imgwriter` directory
- Go to linux or windows\vc6.0 or windows\vc2005 or windows\vc2008 based on your platform.
- Execute the commands in Table 25 based on your platform
- binary is available under the directory *<lipi-core-toolkit install directory>/bin*.

Platform	Package
Windows XP Professional Editions for VC6.0	1. <vc++6.0 install dir>\VC98\Bin\VCVARS32.BAT 2. nmake /f Makefile.win
Windows XP Professional Editions for VC2005	1. <visual studio 2005 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release imgwriter.vcproj

Windows XP Professional Editions for VC2008	1. <visual studio 2008 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release imgwriter.vcproj
Linux	make –f Makefile.linux

Table 25: Command for building Imagewriter module

13-3 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 26: 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.

13-3-1 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 27: mdv errors

14 Sample client applications

14-1 Introduction

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

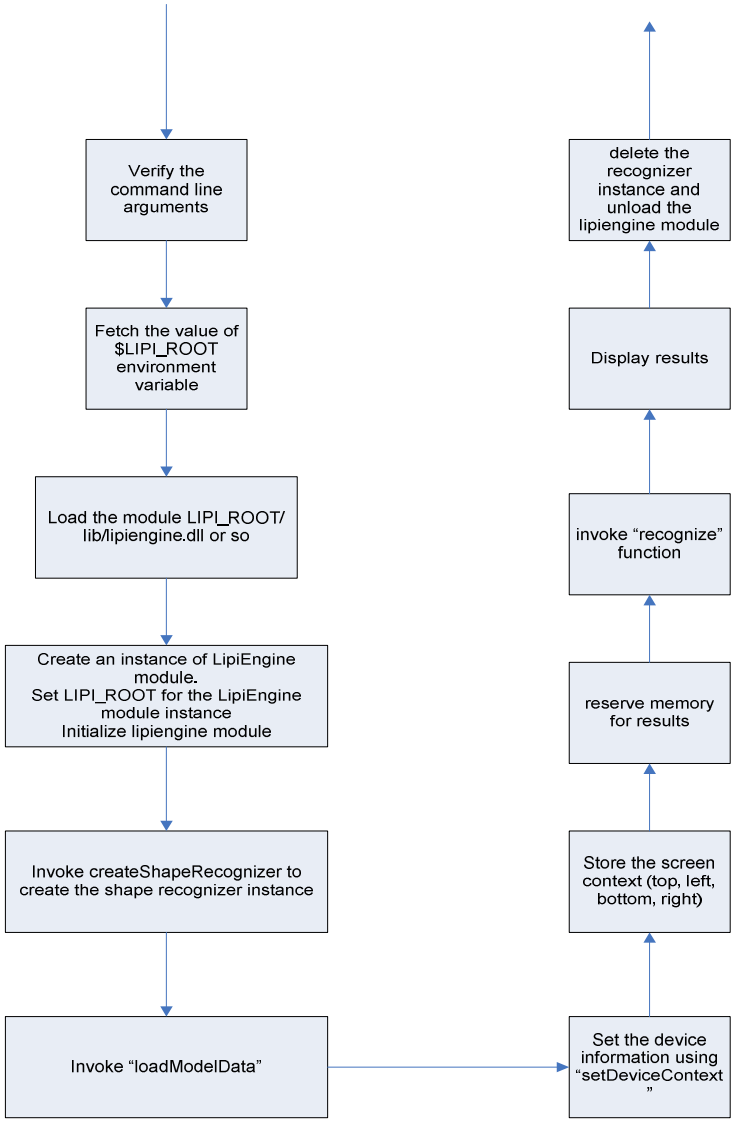
lipi-core-toolkit 3.0.0 provides two sample applications

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

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

14-2 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:



14-2-1 Included source code, headers and binaries

14-2-1-1 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 28: shaperectst file locations

14-2-1-2 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 29: 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 30: shaperectst Shared libraries required

14-2-1-3 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 31: shaperectst Header files required

14-2-2 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 32: 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 33: LTKDeviceContext member functions used: shaperectst

Ink data structure (LTKTraceGroup): Stores the digital ink

Member functions used

None

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 34: LTKShaperecResult member functions used: shaperectst

14-2-3 Building shaperectst

The sample program may be built using the provided `Makefiles` in the directory `<lipi-core-toolkit install directory>/src/apps/samples/shaperectst/windows/vc6.0` for windows and `$LIPI_ROOT/src/apps/samples/shaperectst/linux` for Linux. To build the VC2005 code, the vcproj file has been provided in the directory `<lipi-core-toolkit install directory>/src/apps/samples/shaperectst/windows/vc2005`. To build the VC2008 code, the vcproj file has been provided in the directory `<lipi-core-toolkit install`

directory>/src/apps/samples/shaperectst/windows/vc2008. Execute the following command based on your platform

Platform	Package
Windows XP Professional Editions for VC6.0	1. <vc++6.0 install dir>\VC98\Bin\VCVARS32.BAT 2. nmake /f Makefile.win
Windows XP Professional Editions for VC2005	1. <visual studio 2005 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release shaperectst.vcproj
Windows XP Professional Editions for VC2008	1. <visual studio 2008 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release shaperectst.vcproj
Linux	make –f Makefile.linux

Table 35: Command for building shaperectst

The makefiles may be modified to build your own application (in lieu of shaperectst).

The application builds into an executable, available under

<lipi-core-toolkit install directory>/src/apps/samples/shaperectst/windows/vc6.0/Release for VC6.0 (Windows),

<lipi-core-toolkit install directory>/src/apps/samples/shaperectst/windows/vc2005/release for VC2005 (Windows),

<lipi-core-toolkit install directory>/src/apps/samples/shaperectst/windows/vc2008/release for VC2008 (Windows) and

<lipi-core-toolkit install directory>/src/apps/samples/shaperectst/linux on Linux.

Usage: shaperectst

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

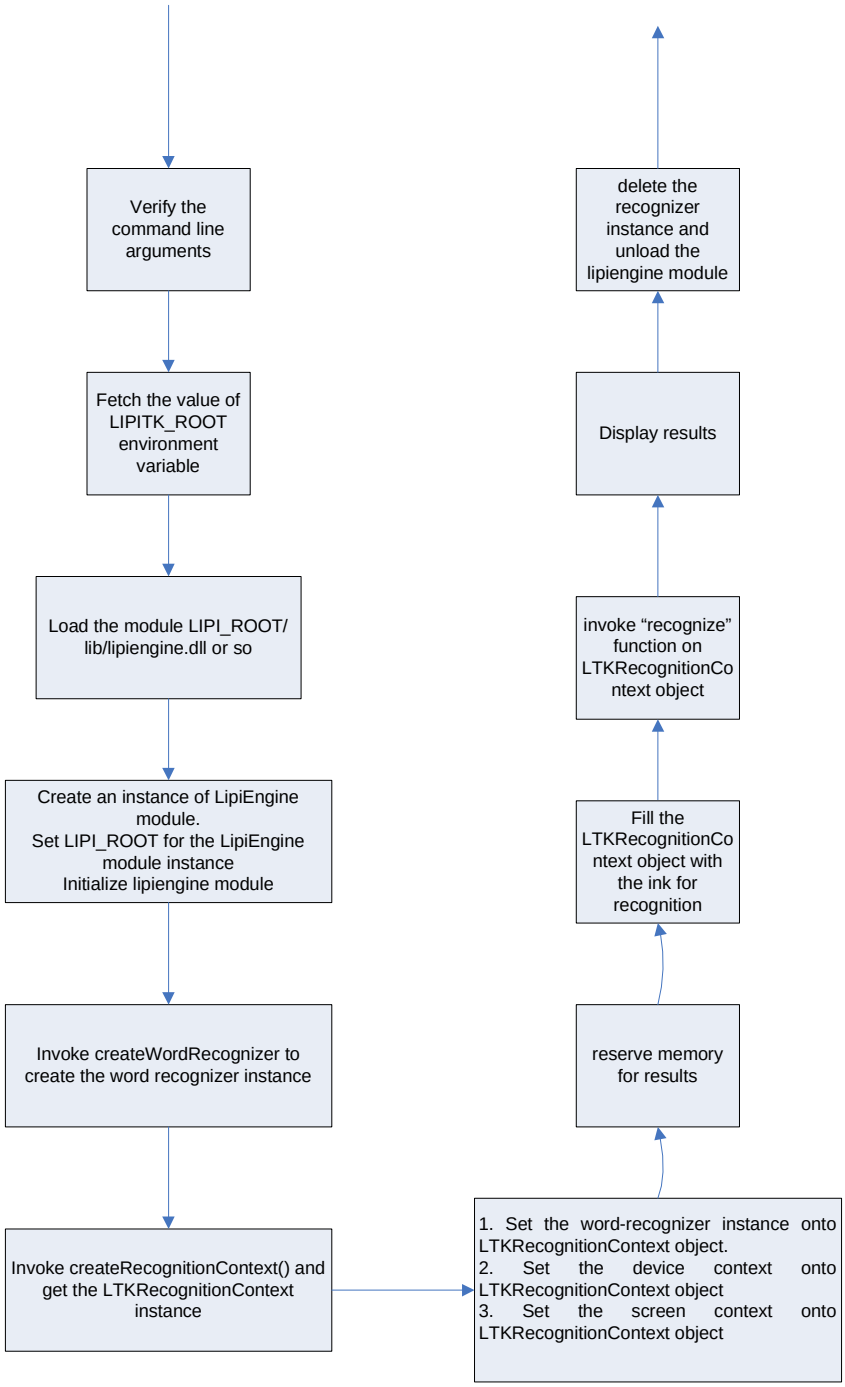
Command line arguments

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: NUMERALS_NUM, for numerals project
< ink file to recognize>	Mandatory	Path of the UNIPEN ink file to be recognized should be passed.

Table 36: Command line arguments: shaperectst

NOTE: Make sure environment variable LIPI_ROOT is set to the Lipi installation directory.

14-3 Sample program `wordrectst`



14-3-1 Included source code, headers and binaries

14-3-1-1 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 37: File locations: wordrectst

14-3-1-2 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 38: 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 39: Shared libraries required: wordrectst

14-3-1-3 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 40: Header files required: wordrectst

14-3-2 Important Data structures – BoxedField 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 41: Some of the member functions of LTKRecognitionContext

14-3-3 Building `wordrectst`

The sample program may be built using the provided `Makefiles` in the directory `$LIPI_ROOT/src/apps/samples/wordrectst/windows/vc6.0` for windows and `$LIPI_ROOT/src/apps/samples/wordrectst/linux` for Linux. To build the VC2005 code, the vcproj file has been provided in the directory `$LIPI_ROOT/src/apps/samples/wordrectst/windows/vc2005`. To build the VC2008 code, the vcproj file has been provided in the directory `$LIPI_ROOT/src/apps/samples/wordrectst/windows/vc2008`. Execute the command based on your platform

Platform	Package
Windows XP Professional Editions for VC6.0	1. <vc++6.0 install dir>\VC98\Bin\VCVARS32.BAT 2. nmake /f Makefile.win
Windows XP Professional Editions for VC2005	1. <visual studio 2005 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release wordrectst.vcproj
Windows XP Professional Editions for VC2008	1. <visual studio 2008 install dir>\ Common7\Tools\ vsvars32.bat 2. devenv /build Release wordrectst.vcproj
Linux	make –f Makefile.linux

Table 42: Command for building wordrectst

The makefiles may be modified to build your own application (in lieu of `wordrectst`).

The application builds into an executable, available under

<lipi-core-toolkit install directory>`/src/apps/samples/wordrectst/windows/vc6.0/Release` for VC6.0 (Windows)

<lipi-core-toolkit install directory>`/src/apps/samples/wordrectst/windows/vc2005/release` for VC2005 (Windows),

<lipi-core-toolkit install directory>/src/apps/samples/wordrectst/windows/vc2008/release for VC2008 (Windows) and

<lipi-core-toolkit install directory>/src/apps/samples/wordrectst/linux on Linux.

NOTE: Make sure environment variable LIPI_ROOT is set to the Lipi installation directory.

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 43: Command line arguments: wordrectst

14-4 Sample Windows mobile application

The sample application uses Lipi Core Toolkit 3.0 for recognizing pre-build recognizers on Windows Mobile Pocket PC. The application source code can be used as a reference to build handwriting recognition applications.

14-4-1 Building application

The sample application may be built using the provided Makefiles in the directory $LIPI_ROOT/src/apps/samples/UIApp/IMEApp/windows/wm5.0 for windows mobile. Execute the command based on your platform

Extract Windows Mobile package in Windows XP professional Edition.

1. run command prompt.
2. <visual studio 2005 install dir>\ Common7\Tools\ vsvars32.bat

3. if source package downloaded, first build lipi-cor-toolkit in windows mobile version (Building on Windows for Windows Mobile 6.0).
4. go to `$LIPI_ROOT/src/apps/samples/UIApp/IMEApp/windows/wm5.0`
5. devenv /build Release IMEApp.vcproj (or) Msbuild IMEApp.targets

After building the sample application, create cab file by using IMESetup.vddproj available under `$LIPI_ROOT/src/apps/samples/UIApp/IMESetup`

1. devenv /build Release `IMESetup.vddproj`
2. Follow the Installation instructions to Install setup in windows mobile

---

NOTE: There is no runwordrec tool for Windows Mobile version.

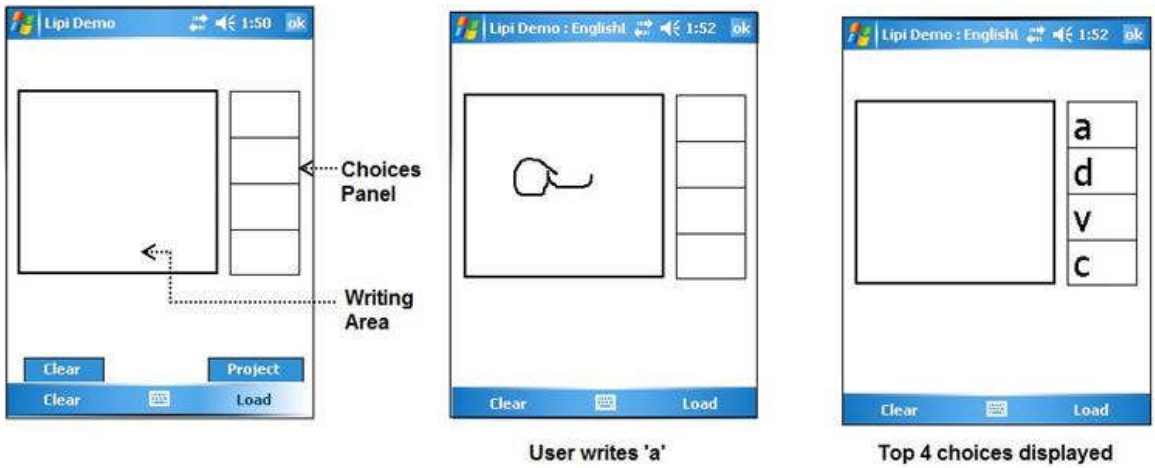
14-4-2 Installation instructions

1. Use ActiveSync to transfer the **<setup package>** created in the above steps from the Windows PC to the Pocket PC device.
2. Locate the **<setup package>** file in the Pocket PC (default location is 'My Documents') and then tap on the file to install the application.
3. The sample application packaged with Lipi Core Toolkit 3.0.0 library and pre-built recognizer for English lowercase alphabets will be deployed.

NOTE: You may refer the PocetPC Central Tutorial for information on installing a CAB file on Windows Mobile.

14-4-3 User interface

A snapshot of the User Interface is given below.



Launch the application from the installed location (Open 'File explorer' and locate 'LIPITK' folder) in Windows Mobile. On start up, the application loads the Lipi Core Toolkit library and instantiates the Lipi recognition engine.

Load Project: Click on the Load -> Project menu item button to load the English lowercase pre-built recognizer.
Note: The LIPITK/projects directory must have the pre-built recognizer data files.

Writing Area: Write a lowercase English alphabet in the writing area. As soon as the user completes writing, the top 4 choices of recognized character are displayed in the choices panel.

Clear: The Clear Menu Item can be used to clear the results displayed in the choices panel.

14-4-4 Changing the default pre-built recognizer

To use an alternative pre-built recognizer instead of the default English lowercase data, replace the files under LIPITK/projects directory with the pre-built recognizer of choice.

14-4-5 Preparing Unicode Mapping file

The application uses the file *LIPITK/projects'/*<pre-built recognizer*>/config/mapping.txt// to map the class IDs returned by the recognizer to the corresponding character Unicode. This is then displayed in the choices panel. Users may refer* ***Unicode Charts*** *for finding the Unicode character code by scripts. If this mapping file relevant for the user defined pre-built recognizer is not provided, then the application displays the Class IDs received from the recognizer in the choices panel.*

15 Using Lipitk

15-1 Creating and using a Shape Recognizer

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.

15-2 Creating a Handwritten Numeral Recognizer

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

Installing the toolkit

Please refer to the 15Installing Packages.

Setting up a Shape Recognition Project

lipi-core-toolkit 3.0.0 provides the **numerals** 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/numerals`.

The project configuration file, `project.cfg`, for the **numerals** 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 3.0.0 provides the **default** profile for this project. The profile configuration file, `profile.cfg`, available under the directory `$PROJ_ROOT/numerals/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 handwritten samples from ten users. 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>/numerals/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 `numerals_trainlist.txt` and `numerals_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 numerals 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.

Building the Project

The steps to build the **numerals** project are as follows:

- Go to the project root

```
cd <lipi-core-toolkit install directory>/projects/numerals
```

- Generate the Makefile for the Project

```
<lipi-core-toolkit install directory>/scripts/genmake.pl -project
numerals
```

- Run nmake for Windows and make on Linux

```
nmake /n make
```

This results in compiling all the required sources (such as nn). In addition, a utility called *runshaperec* is also built to facilitate training and testing of the recognizer.

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/numerals_trainlist.txt
            -project numerals
```

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

Once the recognizer has been trained, tested, evaluated and found to be satisfactory, it may be packaged for deployment on the machine where we expect it to be invoked from an application. The steps in creating the package are as follows:

- Go to the package directory

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

The config file `package.cfg` specifies the project (numerals) and profile (default) that need to be packaged, and a logical name (**NUMERALS_NUM**) that may be used to refer to the shape recognizer from the application (details in Section11 11Packaging).

- Execute the packing script to create the final package

```
<lipi-core-toolkit install
directory>/scripts/package.pl -pkg package.cfg
                              -pkgname num_reco
```

The package `num_reco-linux.tar.gz` (Linux) or `num_reco-winvc6.0.cab` (Windows for VC6.0) or `num_reco-winvc2005.cab` (Windows for VC2005) or `num_reco-winvc2008.cab` (Windows for VC2008) or `num_reco-wm6.0.cab` (Windows for wm 6.0) is created in the directory `$LIPI_ROOT/package`. For more details for the packaging scripts, please refer to section 11-1 .

15-3 Integrating the shape recognizer with a client application

In this section, we will look at how to integrate a 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, and that the sample test application `shaperectst` is part of the package.

The steps involved are as follows:

- Installing the packaged recognizer(s) on the target machine
- Setting up the environment
- Integrating the recognizer into an application

The following section describes each step in detail.

Installing the package on the target machine

The **target machine** is the computer on which we intend to use the recognizer by integrating it into a (pen-based) application and running the application, and is typically different from the machine on which the recognizer was developed. Note that Lipi Toolkit is in general NOT installed on the target machine. The package contains everything needed for the recognizer to function on the target machine.

Copy the recognizer package (`num_reco-winvc6.0.cab` or `num_reco-winvc2005.cab` or `num_reco-winvc2008.cab` on Windows or `num_reco-linux.tar.gz` on Linux) from where it was created (as shown in the previous section) onto the target machine.

Follow the instructions in section 153-3 to unpack the package to a local directory.

Setting up the environment

Set the environment variable LIPI_ROOT to the directory where the package was unpacked.

Integrating the recognizer into an application

Essentially, once installed, the numerals recognizer becomes available to applications via the logical name, **NUMERALS_NUM**, as specified in the package configuration file.

The instructions for integration are provided as part of the "Sample client applications" documentation. Since these steps are elaborate, they are not repeated here.

Following the integration, you should be able to use the application to get digital ink corresponding to a numeral character from a digitizer or mouse or from a file, pass the ink to the recognizer, and get back the most plausible shape IDs and corresponding confidence values.

16 Creating and Using a Word Recognizer

lipi-core-toolkit 3.0.0 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 numerals, as detailed in section 15-2 .

16-1 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 field recognizer are limited to:

- Setting up a Word Recognition Project
- Building the Project
- Packaging the Word Recognizer for deployment

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/numerals`. 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 = 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 **numerals** 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-core-toolkit install directory>/src/reco/wordrec/boxfld` to `$PROJ_ROOT/config/default`. The default contents of the file are as follows and may be modified as needed.

```
MaxBoxCount=  30
BoxedShapeProject = numerals
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.

Building the Project

The steps in building the word recognition project are as follows:

- Go to the project directory

```
cd $PROJ_ROOT
```

- Generate the makefile for the project

```
<lipi-core-toolkit install
directory>/scripts/genmake.pl -project num_fld
                              -profile default
```

- Build the project (for Linux)

```
make /n Makefile
```

This results in compiling all the required sources (such as boxfld). In addition, it requires shape recognizer project numerals to be built using the corresponding Makefile.

Packaging the Word Recognition Project for deployment

As mentioned earlier, training and evaluation are not supported at the word level for the BoxedField recognizer, in this version of Lipitk. Once the associated shape recognizer has been trained, tested, evaluated and found to be satisfactory, the word recognizer may be packaged for deployment on the machine where we expect it to be invoked from an application. The steps in creating the package are as follows:

- Go to the package directory

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

- Create `package.cfg` with the following contents:

```
[package]
Projects = num_fld (default)
Src = apps/sample/wordrectest
[export]
NUM_FLD = num_fld(default)
```

The `package.cfg` shown above specifies the project (num_fld) and profile (default) that need to be packaged, and a logical name (**NUM_FLD**) that may be used to refer to the word recognizer from the application. The sample test application (wordrectst) may be left out of the package by commenting out the corresponding line from `package.cfg`.

- Execute the packaging script to create the final package

```
<lipi-core-toolkit install
directory>/scripts/package.pl -pkg package.cfg
                              -pkgname num_fld_reco
```

The package `num_fld_reco.tar.gz` (Linux) or `num_fld_reco.cab` (Windows) is created in the directory `<lipi-core-toolkit install directory>/package`. The package not only includes the num_fld project and profile specified in the package configuration file, but it also includes the required shape recognition project numerals and its default profile, as specified in `boxfld.cfg`.

16-2 Integrating the numeric boxed field recognizer with a client application

In this section, we will look at how to integrate a word recognizer created using Lipi Toolkit into a pen-based application. For the purpose of illustration, we will assume that the numeric field recognizer has been packaged as per the previous section, and that the sample test application `wordrectst` is part of the package.

The steps involved are as follows:

- Installing the packaged recognizer(s) on the target machine
- Setting up the environment
- Integrating the recognizer into an application

The following sections describe each step in detail.

Installing the package on the target machine

Copy the recognizer package (`num_fld_reco.cab` on Windows or `num_fld_reco.tar.gz` on Linux) from where it was created (as shown in the previous section 16-1) onto the target machine.

Follow the instructions in section 3-3 for unpacking the package to a local directory. Among other things, this causes the num_fld and numerals projects and their default profiles to appear under the `<lipi-core-toolkit install directory>/projects` directory.

Setting up the environment

Set the environment variable $LIPI_ROOT to the directory where the package was unpacked.

Integrating the recognizer into an application

Essentially, once installed, the numeric field recognizer becomes available to applications via the logical name **NUM_FLD** as specified in the package configuration file.

The instructions for integration are provided as part of the "Sample programs" documentation in section 14. Since these steps are elaborate, they are not repeated here.

Following the integration, you should be able to use the application to get digital ink corresponding to a boxed field of numerals (e.g. a phone number) from a digitizer or mouse or from a file, pass the ink to the recognizer, and get back the mostly plausible numeric strings and corresponding confidence values.

17 Appendix

17-1 Setting environment variables in Windows

To set the environment variable LIPI_ROOT:

1. Open properties of "**My Computer**" from Windows Explorer
2. Go to "**Advanced**" tab from the property dialog.
3. Click on "**Environment Variables**" button and add this new variable under "**User variables**"

To set an environment variable from command prompt, use the following command

```
set LIPI_ROOT=c:\lipi
```

17-2 Setting environment variables in Linux

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

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

17-3 Downloading `cabarc.exe`

You can download the Microsoft Cabinet SDK from the following link:

http://support.microsoft.com/default.aspx?scid=kb;en-us;310618

After downloading, execute `cabsdk.exe` and it will prompt for the directory where the files can be extracted. Give any directory and ensure that the bin directory (after extracting) is added to System PATH variable for you to execute `cabarc.exe` from anywhere.

17-4 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

17-5 Configurable make settings for Windows for VC 6.0

The different configurations that the user can modify for Windows are given below:

Configurable options	Remarks
RELOPT=Release	This flag is used for the final target release option • Release • Debug
BUILD=BUILD	This flag is used to build the project. Another option is REBUILD – this flag is used to clean the project every time before building the project.
CLEAN=CLEAN	This flag is used for cleaning the project.
MSDEV=msdev	This flag is used to compile the VC++6.0 project space.

Table 44: Configurable make settings for Windows for VC 6.0

17-6 Default config file nn.cfg

```
#-------------------------------------------------------------------------
# nn.cfg
#
# Configuration file for Nearest Neighbor Classification Method for
# Lipi Toolkit 3.0.0
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
# The standard format for the configuration entries is the name of the
# configuration parameter seperated by an equal to sign and then the value of
# the configuration parameter. For example:
# ConfigurationEntryName = value
#
# Lines starting with a # are commnet lines
#
# A cfg entry is strictly a key value pair and leaving the key without the
# value or specification of a value out of the range is not permitted
#
# If a cfg entry is not specified at all, then default values are used by the
# recognizer
#-------------------------------------------------------------------------

#------------------------------
#    PREPROCESSING
#------------------------------

#-------------------------------------------------------------------------
# ResampTraceDimension
#
# Description: The number of target points for resampling. In other words,
# each character will be resampled to this number of points. In case of
# multistroke characters, this number of points will be distributed between
# the strokes in proportion to their lengths in proportion to their initial
# number of points.
#
# Valid values: Any integer > 0
# Units: Points
```

```
# Default value: 60
# Typical value: Average number of points per character in the training data set.
#-------------------------------------------------------------------------------
ResampTraceDimension = 60

#-------------------------------------------------------------------------------
# ResampPointAllocation
#
# Description: Method to be used for point allocation among different strokes
# during resampling. Two schemes have been implemented lengthbased and point
# based. In lengthbased allocation scheme, the number of points allocated to
# each stroke is proportional to the length of the stroke. Length of a stroke
# is calculated as the sum of the distances between each point in the stroke.
# In the pointbased allocation scheme, the target stroke point allocation is
# proportional to the number of points in the initial stroke.
#
# Valid value: [lengthbased | pointbased]
# Default value: lengthbased
#-------------------------------------------------------------------------------
ResampPointAllocation = pointbased

#-------------------------------------------------------------------------------
# NormDotSizeThreshold
#
# Description: This threshold is used to determine whether a character is a dot.
# It is expressed in real length terms (inches) and converted internally to
# points using knowledge of the device's spatial resolution. If the width
# and height are both less than this threshold, then all the points are replaced
# with the center of the of the normalized character, basically to represent it
# as a dot
#
# Valid values: Any real number > 0
# Units: inches
# Default value: 0.01
# Typical value: < 0.1
#-------------------------------------------------------------------------------
NormDotSizeThreshold = 0.001

#-------------------------------------------------------------------------------
```

```
# NormLineWidthThreshold
#
# Description: This threshold is used to detect whether the character is a
# vertical or horizontal line. If only the height is less than this threshold
# then the character is detected as a horizontal line and if only the width is
# less than this threshold then the character is detected as a vertical line.
# Assuming the height is along the y-dimension and width is along the x-
# dimension, during normalization of a horizontal line only the x-coordinates
# are scaled and the y-coordinates are translated to the center of the character,
# with out scaling. Similarly for the vertical line only the y-coordinates are
# normalized and the x-coordinates are translated to the center with out scaling
#
# Valid values: Any real number > 0
# Units: inches
# Default value: 0.01
# Typical value: < 0.1
#-------------------------------------------------------------------------------
NormLineWidthThreshold = 0.001

#-------------------------------------------------------------------------------
# NormPreserveAspectRatio
#
# Description: This parameter is used to indicate whether the aspect ratio
# has to be preserved during normalization. The aspect ratio is the calculated
# as maximum of (height/width , width/height). The aspect ratio is preserved only
# if the calculated aspect ratio is greater than the threshold value specified
# through NormPreserveAspectRatioThreshold and this configuration variable is
# set to true. If this configuration variable is set to false the aspect ratio
# is not preserved during normalization.
#
# Valid value: [true | false]
# Default value: true
#-------------------------------------------------------------------------------
NormPreserveAspectRatio = false

#-------------------------------------------------------------------------------
# NormPreserveAspectRatioThreshold
#
# Description: Aspect ratio is preserved during normalization if the computed
```

```
# aspect ratio (max(height/width, width/height)) is greater than this threshold
# and the configuration value NormPreserveAspectRatio is set to true. During
# aspect ratio preserving normalization, the larger of the two dimensions is
# normalized to the standard size and the other dimension is normalized
# proportional to the initial height and width ratio, so that the initial
# aspect ratio is maintained.
#
# Valid values: Any real number >= 1
# Default value: 3
# Typical value: >= 1.5
#---------------------------------------------------------------------------
NormPreserveAspectRatioThreshold = 1

#---------------------------------------------------------------------------
# NormPreserveRelativeYPosition
#
# Description: The relative Y position is the mean of the y-coordinates in the
# input character. During normalization if this parameter is set to true, each
# y-coordinate of the character point is translated by the initial y-mean value,
# so that the mean of the y-coordinates remains the same before and after
# normalization. This is typically used in the word recognition context where
# each stroke of the character has to be normalized separately and the relative
# position of the strokes should be maintained even after normalization.
#
# Valid value: [true | false]
# Default value: false
#---------------------------------------------------------------------------
NormPreserveRelativeYPosition = false

#---------------------------------------------------------------------------
# SmoothWindowSize
#
# Description: The configuration value specifies the length of the moving
# average filter (size of the window) for smoothing the character image.
# If this value is set to N, then each point in the input character is replaced
# by the average of value of this point, (N-1)/2 points on the right and (N-1)/2
# on the left of this point.
#
# Valid value: Any integer > 0
```

```
# Units: Points
# Typical value: 5
# Default value: 3
#-------------------------------------------------------------------------------
SmoothWindowSize = 3


#-------------------------------------------------------------------------------
# PreprocSequence
#
# Description: This variable is used to specify the sequence of preprocessing
# operations to be carried out on the input character sample before extracting
# the features. A valid preprocessing sequence can consist of combination of one
# or more of the functions selected from the valid values set mentioned below.
# The CommonPreProc prefix is used specify the default preprocessing module of
# LipiTk. The user can add his own preprocessing functions in other modules and
# specify them in the preprocessing sequence.
#
# Valid values: Any sequence formed from the following set
# CommonPreProc::normalizeSize;
# CommonPreProc::removeDuplicatePoints;
# CommonPreProc::smoothenTraceGroup;
# CommonPreProc::dehookTraces;
# CommonPreProc::normalizeOrientation;
# CommonPreProc::resampleTraceGroup;
# Default value:
{CommonPreProc::normalizeSize,CommonPreProc::resampleTraceGroup,CommonPreProc::no
rmalizeSize}
#-------------------------------------------------------------------------------
PreprocSequence={CommonPreProc::smoothenTraceGroup,CommonPreProc::normalizeSize,C
ommonPreProc::resampleTraceGroup,CommonPreProc::normalizeSize}


#--------------------------------------
#    TRAINING
#--------------------------------------


#-------------------------------------------------------------------------------
# NNTrainPrototypeSelectionMethod
#
# Description: This is used to specify the prototype selection method to be used
# while training the shape recognizer. When set to hier-clustering, the
```

```
# prototypes are selected using hierarchical clustering method.
#
# Valid value: [hier-clustering]
# Default value: hier-clustering
#-------------------------------------------------------------------------------
NNTrainPrototypeSelectionMethod=hier-clustering

#-------------------------------------------------------------------------------
# NNTrainPrototypeReductionFactorPerClass
#
# Description: This config parameter is used only when the prototype selection
# is clustering. This config parameter is used to specify the amount of the
# initial prototypes to be excluded during prototype selection.
# Set it to automatic if the number of clusters is to be determined
# automatically. Set it to none if no prototype selection is required. If the
# value of this parameter is set to a number between 1-100, say 25, then 75%
# (i.e 100-25) of the initial training data are retained as prototypes.
# This parameter can be specified only if the NNTrainNumPrototypesPerClass
# is not specified.
#
# Valid value: [automatic | none | any real number from 0-100]
# Default value: automatic
#-------------------------------------------------------------------------------
NNTrainPrototypeReductionFactorPerClass = 50

#-------------------------------------------------------------------------------
# NNTrainNumPrototypesPerClass
#
# Description: This config parameter is used only when the prototype selection
# is clustering. This is used to specify the number of prototypes to be selected
# from the training data. This parameter can be specified only if
# PrototypeReductionFactor is not specified. This config entry is commented as
# only one of NNTrainPrototypeReductionFactorPerClass or
# NNTrainNumPrototypesPerClass can be active in a valid cfg file.
#
# Valid value: [automatic | none | any integer from 1-N]
#              (N is the number of samples # per class)
# Default value: automatic
#-------------------------------------------------------------------------------
```

```
#NNTrainNumPrototypesPerClass=automatic

# Note: Only one of either PrototypeReductionFactor or NumClusters can be
# enabled at any particular instance

#----------------------------------------
#    FEATURE EXTRACTION
#----------------------------------------

#-------------------------------------------------------------------------------
# FeatureExtractor
#
# Description: The configuration value is used to specify the feature extraction
# module to be used for feature extraction. The point float feature extraction
# module extracts the x,y,cosine and sine angle features at every point of the
# character.
#
# Valid value: [PointFloatShapeFeatureExtractor|L7ShapeFeatureExtractor|
#               NPenShapeFeatureExtractor|SubStrokeShapeFeatureExtractor]
# Default value: PointFloatShapeFeatureExtractor
#-------------------------------------------------------------------------------
FeatureExtractor=PointFloatShapeFeatureExtractor

#----------------------------------------
#    RECOGNITION
#----------------------------------------

#-------------------------------------------------------------------------------
# NNRecoDTWEuFilterOutputSize
#
# Description: This config parameter is used to set the number of nearest
# neighbours (filtered based on euclidean distance)to be considered for
# calculating dtw distance. Set to all if all samples are to be considered for
# calculating dtw distance. This is mainly used to increase the speed of
# recognition.
#
# Valid value: [all| any integer from 1-N](N is the size of prototype set)
# Default Value: all
#-------------------------------------------------------------------------------
```

```
NNRecoDTWEuFilterOutputSize = 15

#-------------------------------------------------------------------------------
# NNRecoRejectThreshold
#
# Description: Threshold to reject the test sample. If the confidence obtained
# for the recognition of test sample is less than this threshold then the test
# sample is rejected.
#
# Valid value: Any real number from 0-1
# Default value: 0.001
#-------------------------------------------------------------------------------
NNRecoRejectThreshold = 0.001

#-------------------------------------------------------------------------------
# NNRecoNumNearestNeighbors
#
# Description: Number of nearest neighbors to be considered during recognition
# and computation of confidence. If the value is set to 1, nearest neighbor
# classifier is used, otherwise k-nearest neighbor or Adaptive k-nearest
# neighbor classifiers are used. By default, nearest neighbor classifier is used.
#
# Valid value: Any integer >= 1
# Default value: 1
#-------------------------------------------------------------------------------
NNRecoNumNearestNeighbors = 1

#-------------------------------------------------------------------------------
# NNRecoUseAdaptiveKNN
#
# Description: This parameter is used to specify whether Adaptive k-nearest
# neighbor recognizer (A-kNN) is to be used. If set to true, A-kNN recognizer is
# used, otherwise kNN recognizer is used.  The A-kNN recognizer automatically
# determines the number of nearest neighbors to be considered for recognition in
# each class. If NNRecoNumNearestNeighbors is set to 1, this parameter is
# automatically set to false and the manually set value will not be considered.
#
# Valid value: [true | false]
# Default value: false
```

```
#-------------------------------------------------------------------------------
NNRecoUseAdaptiveKNN = false

#-----------------------------------------
#      COMMON FOR TRAINING AND RECOGNITION
#-----------------------------------------

#-------------------------------------------------------------------------------
# NNPrototypeDistanceMeasure
#
# Description: This configuration parameter is used to specify the distance
# measure to be used in clustering and recognition. DTW or Euclidean distance
# measures can be used.
#
# Valid value [dtw | eu]
# Default value: dtw
#-------------------------------------------------------------------------------
NNPrototypeDistanceMeasure = dtw

#-------------------------------------------------------------------------------
# NNDTWBandingRadius
#
# Description: This configuration parameter specifies the banding radius
# to be used for DTW computation. This is used to speed up the computation
# process. If this value is zero no banding is done. The value is specified as
# fraction of ResampTraceDimension to be used while computing the DTW
# distance.
#
# Valid values: Any real number > 0 and <= 1
# Default Value: 0.33
#-------------------------------------------------------------------------------
NNDTWBandingRadius=0.33

#-------------------------------------------------------------------------------
# NNMDTFileUpdateFreq
#
# Description: This configuration parameter specifies the number of iterations
after
# which MDT file is to be updated.
```

```
# Every call to addClass or deleteClass will add/delete the given class. These
# in-memory changes will be reflected in nn.mdt only after the specified
# number of such iterations and on application exit.
#
# Valid values: Any integer > 0
# Default value: 5
# Typical value: 5
#-------------------------------------------------------------------------------
NNMDTFileUpdateFreq = 5


#-------------------------------------------------------------------------------
# NNDTWBandingRadius
#
# Description: This configuration parameter specifies the mode for
# opening the mdt file.
#
# Valid values: ascii, binary
# Default Value: ascii
#-------------------------------------------------------------------------------
NNMDTFileOpenMode=binary
```

17-7 Default config file activedtw.cfg

```
#-------------------------------------------------------------------------
# activedtw.cfg
#
# Configuration file for Active-DTW Classification Method for
# Lipi Toolkit 3.0.0
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
# The standard format for the configuration entries is the name of the
# configuration parameter seperated by an equal to sign and then the value of
# the configuration parameter. For example:
# ConfigurationEntryName = value
#
# Lines starting with a # are commnet lines
#
# A cfg entry is strictly a key value pair and leaving the key without the
# value or specification of a value out of the range is not permitted
#
# If a cfg entry is not specified at all, then default values are used by the
# recognizer
#-------------------------------------------------------------------------

#------------------------------
#PREPROCESSING
#------------------------------

#-------------------------------------------------------------------------
# ResampTraceDimension
#
# Description: The number of target points for resampling. In other words,
# each character will be resampled to this number of points. In case of
# multistroke characters, this number of points will be distributed between
# the strokes in proportion to their lengths in proportion to their initial
# number of points.
#
# Valid values: Any integer > 0
# Units: Points
```

```
# Default value: 60
# Typical value: Average number of points per character in the training data set.
#-------------------------------------------------------------------------------
ResampTraceDimension = 60

#-------------------------------------------------------------------------------
# ResampPointAllocation
#
# Description: Method to be used for point allocation among different strokes
# during resampling. Two schemes have been implemented lengthbased and point
# based. In lengthbased allocation scheme, the number of points allocated to
# each stroke is proportional to the length of the stroke. Length of a stroke
# is calculated as the sum of the distances between each point in the stroke.
# In the pointbased allocation scheme, the target stroke point allocation is
# proportional to the number of points in the initial stroke.
#
# Valid value: [lengthbased | pointbased]
# Default value: lengthbased
#-------------------------------------------------------------------------------
ResampPointAllocation = lengthbased

#-------------------------------------------------------------------------------
# NormDotSizeThreshold
#
# Description: This threshold is used to determine whether a character is a dot.
# It is expressed in real length terms (inches) and converted internally to
# points using knowledge of the deviceï¿½s spatial resolution. If the width
# and height are both less than this threshold, then all the points are replaced
# with the center of the of the normalized character, basically to represent it
# as a dot
#
# Valid values: Any real number > 0
# Units: inches
# Default value: 0.01
# Typical value: < 0.1
#-------------------------------------------------------------------------------
NormDotSizeThreshold = 0.01

#-------------------------------------------------------------------------------
```

```
# NormLineWidthThreshold
#
# Description: This threshold is used to detect whether the character is a
# vertical or horizontal line. If only the height is less than this threshold
# then the character is detected as a horizontal line and if only the width is
# less than this threshold then the character is detected as a vertical line.
# Assuming the height is along the y-dimension and width is along the x-
# dimension, during normalization of a horizontal line only the x-coordinates
# are scaled and the y-coordinates are translated to the center of the character,
# with out scaling. Similarly for the vertical line only the y-coordinates are
# normalized and the x-coordinates are translated to the center with out scaling
#
# Valid values: Any real number > 0
# Units: inches
# Default value: 0.01
# Typical value: < 0.1
#-------------------------------------------------------------------------------
NormLineWidthThreshold = 0.01

#-------------------------------------------------------------------------------
# NormPreserveAspectRatio
#
# Description: This parameter is used to indicate whether the aspect ratio
# has to be preserved during normalization. The aspect ratio is the calculated
# as maximum of (height/width , width/height). The aspect ratio is preserved only
# if the calculated aspect ratio is greater than the threshold value specified
# through NormPreserveAspectRatioThreshold and this configuration variable is
# set to true. If this configuration variable is set to false the aspect ratio
# is not preserved during normalization.
#
# Valid value: [true | false]
# Default value: true
#-------------------------------------------------------------------------------
NormPreserveAspectRatio = true

#-------------------------------------------------------------------------------
# NormPreserveAspectRatioThreshold
#
# Description: Aspect ratio is preserved during normalization if the computed
```

```
# aspect ratio (max(height/width, width/height)) is greater than this threshold
# and the configuration value NormPreserveAspectRatio is set to true. During
# aspect ratio preserving normalization, the larger of the two dimensions is
# normalized to the standard size and the other dimension is normalized
# proportional to the initial height and width ratio, so that the initial
# aspect ratio is maintained.
#
# Valid values: Any real number >= 1
# Default value: 3
# Typical value: >= 1.5
#---------------------------------------------------------------------------
NormPreserveAspectRatioThreshold = 3

#---------------------------------------------------------------------------
# NormPreserveRelativeYPosition
#
# Description: The relative Y position is the mean of the y-coordinates in the
# input character. During normalization if this parameter is set to true, each
# y-coordinate of the character point is translated by the initial y-mean value,
# so that the mean of the y-coordinates remains the same before and after
# normalization. This is typically used in the word recognition context where
# each stroke of the character has to be normalized separately and the relative
# position of the strokes should be maintained even after normalization.
#
# Valid value: [true | false]
# Default value: false
#---------------------------------------------------------------------------
NormPreserveRelativeYPosition = false

#---------------------------------------------------------------------------
# SmoothWindowSize
#
# Description: The configuration value specifies the length of the moving
# average filter (size of the window) for smoothing the character image.
# If this value is set to N, then each point in the input character is replaced
# by the average of value of this point, (N-1)/2 points on the right and (N-1)/2
# on the left of this point.
#
# Valid value: Any integer > 0
```

```
# Units: Points
# Typical value: 5
# Default value: 3
#-------------------------------------------------------------------------
SmoothWindowSize = 3

#-------------------------------------------------------------------------
# NNPreprocSequence
#
# Description: This variable is used to specify the sequence of preprocessing
# operations to be carried out on the input character sample before extracting
# the features. A valid preprocessing sequence can consist of combination of one
# or more of the functions selected from the valid values set mentioned below.
# The CommonPreProc prefix is used specify the default preprocessing module of
# LipiTk. The user can add his own preprocessing functions in other modules and
# specify them in the preprocessing sequence.
#
# Valid values: Any sequence formed from the following set
# CommonPreProc::normalizeSize;
# CommonPreProc::removeDuplicatePoints;
# CommonPreProc::smoothenTraceGroup;
# CommonPreProc::dehookTraces;
# CommonPreProc::normalizeOrientation;
# CommonPreProc::resampleTraceGroup;
# Default value:
{CommonPreProc::normalizeSize,CommonPreProc::resampleTraceGroup,CommonPreProc::no
rmalizeSize}
#-------------------------------------------------------------------------
PreprocSequence={CommonPreProc::normalizeSize,CommonPreProc::resampleTraceGroup,C
ommonPreProc::normalizeSize}


#--------------------------------------
#TRAINING
#--------------------------------------


#-------------------------------------------------------------------------
# NNTrainPrototypeSelectionMethod
#
# Description: This is used to specify the prototype selection method to be used
# while training the shape recognizer. When set to hier-clustering, the
```

```
# prototypes are selected using hierarchical clustering method.
#
# Valid value: [hier-clustering]
# Default value: hier-clustering
#-------------------------------------------------------------------------------
NNTrainPrototypeSelectionMethod=hier-clustering

#-------------------------------------------------------------------------------
# NNTrainPrototypeReductionFactorPerClass
#
# Description: This config parameter is used only when the prototype selection
# is clustering. This config parameter is used to specify the amount of the
# initial prototypes to be excluded during prototype selection.
# Set it to automatic if the number of clusters is to be determined
# automatically. Set it to none if no prototype selection is required. If the
# value of this parameter is set to a number between 1-100, say 25, then 75%
# (i.e 100-25) of the initial training data are retained as prototypes.
# This parameter can be specified only if the NNTrainNumPrototypesPerClass
# is not specified.
#
# Valid value: [automatic | none | any real number from 0-100]
# Default value: automatic
#-------------------------------------------------------------------------------
NNTrainPrototypeReductionFactorPerClass = 25

#-------------------------------------------------------------------------------
# NNTrainNumPrototypesPerClass
#
# Description: This config parameter is used only when the prototype selection
# is clustering. This is used to specify the number of prototypes to be selected
# from the training data. This parameter can be specified only if
# PrototypeReductionFactor is not specified. This config entry is commented as
# only one of NNTrainPrototypeReductionFactorPerClass or
# NNTrainNumPrototypesPerClass can be active in a valid cfg file.
#
# Valid value: [automatic | none | any integer from 1-N]
#              (N is the number of samples # per class)
# Default value: automatic
#-------------------------------------------------------------------------------
```

```
#NNTrainNumPrototypesPerClass=100

# Note: Only one of either PrototypeReductionFactor or NumClusters can be
# enabled at any particular instance

#-------------------------------------------------------------------------
# ActiveDTWRetainPercentEigenEnergy
#
# Description: This config parameter is used to specify the amount of Eigen
#              energy to be included to select the number of eigen vectors
#
# Valid value: [any integer from 0-100]
#
# Default value: 90
#-------------------------------------------------------------------------
ActiveDTWRetainPercentEigenEnergy= 90

#-------------------------------------------------------------------------
# ActiveDTWMinClusterSize
#
# Description: This config parameter is used to specify the minimum number
#              of samples required to form a cluster
#
# Valid value: [any postive integer > 1]
#
# Default value: 2
#-------------------------------------------------------------------------
ActiveDTWMinClusterSize = 2

#---------------------------------------
#FEATURE EXTRACTION
#---------------------------------------

#-------------------------------------------------------------------------
# FeatureExtractor
#
# Description: The configuration value is used to specify the feature extraction
# module to be used for feature extraction. The point float feature extraction
# module extracts the x,y,cosine and sine angle features at every point of the
```

```
# character.
#
# Valid value: [PointFloatShapeFeatureExtractor|L7ShapeFeatureExtractor|
#               NPenShapeFeatureExtractor|SubStrokeShapeFeatureExtractor]
# Default value: PointFloatShapeFeatureExtractor
#-------------------------------------------------------------------------------
FeatureExtractor=PointFloatShapeFeatureExtractor

#---------------------------------------
#RECOGNITION
#---------------------------------------

#-------------------------------------------------------------------------------
# NNRecoDTWEuFilterOutputSize
#
# Description: This config parameter is used to set the proportion of nearest
# cluster or singleton vectors from a class (filtered based on euclidean
distance)
# to be considered for calculating deformations or dtw distance. Set to 100 if
# all clusters or singletons are to be considered for calculating dtw distance.
# This is mainly used to increase the speed of recognition.
#
# Valid value: [all | any number from 1-100]
# Default Value: all
#-------------------------------------------------------------------------------
NNRecoDTWEuFilterOutputSize = 30

#-------------------------------------------------------------------------------
# ActiveDTWEigenSpreadValue
#
# Description: This value is used to configure the range of values the
# bound constraint optimization algorithm will take to calculate the
# optimal deformation sample
# Valid value: [greater than 0| default = 16]
#-------------------------------------------------------------------------------
ActiveDTWEigenSpreadValue = 16

#-------------------------------------------------------------------------------
# ActiveDTWUseSingleton
```

```
#
# Description: This value is used to configure whether singleton vectors
# from classes will be taken into consideration during the recognition
# process
# Valid value: [true | false]
# Default Value: true
#-------------------------------------------------------------------------
ActiveDTWUseSingleton = true

#-------------------------------------------------------------------------
# NNRecoRejectThreshold
#
# Description: Threshold to reject the test sample. If the confidence obtained
# for the recognition of test sample is less than this threshold then the test
# sample is rejected.
#
# Valid value: Any real number from 0-1
# Default value: 0.001
#-------------------------------------------------------------------------
NNRecoRejectThreshold = 0.001

#-------------------------------------------------------------------------
# NNRecoNumNearestNeighbors
#
# Description: Number of nearest neighbors to be considered during recognition
# and computation of confidence. If the value is set to 1, nearest neighbor
# classifier is used, otherwise k-nearest neighbor or Adaptive k-nearest
# neighbor classifiers are used. By default, nearest neighbor classifier is used.
#
# Valid value: Any integer >= 1
# Default value: 1
#-------------------------------------------------------------------------
NNRecoNumNearestNeighbors = 1

#-------------------------------------------------------------------------
# NNRecoUseAdaptiveKNN
#
# Description: This parameter is used to specify whether Adaptive k-nearest
# neighbor recognizer (A-kNN) is to be used. If set to true, A-kNN recognizer is
```

```
# used, otherwise kNN recognizer is used.  The A-kNN recognizer automatically
# determines the number of nearest neighbors to be considered for recognition in
# each class. If NNRecoNumNearestNeighbors is set to 1, this parameter is
# automatically set to false and the manually set value will not be considered.
#
# Valid value: [true | false]
# Default value: false
#-------------------------------------------------------------------------------
NNRecoUseAdaptiveKNN = false

#-----------------------------------------
#      ADAPTATION
#-----------------------------------------

#-------------------------------------------------------------------------------
# ActiveDTWMaxClusterSize
#
# Description: This config parameter is used to specify the maximum number
#              of samples a cluster is permitted to have
#
# Valid value: [any postive integer > 1 And Greater than ActiveDTWMinClusterSize]
#
# Default value: 2
#-------------------------------------------------------------------------------
ActiveDTWMaxClusterSize = 30

#-----------------------------------------
#      COMMON FOR TRAINING AND RECOGNITION
#-----------------------------------------

#-------------------------------------------------------------------------------
# NNDTWBandingRadius
#
# Description: This configuration parameter specifies the banding radius
# to be used for DTW computation. This is used to speed up the computation
# process. If this value is zero no banding is done. The value is specified as
# fraction of ResampTraceDimension to be used while computing the DTW
# distance.
#
```

```
# Valid values: Any real number > 0 and <= 1
# Default Value: 0.33
#-------------------------------------------------------------------------------
NNDTWBandingRadius=0.33


#-------------------------------------------------------------------------------
#ActiveDTWMDTFileUpdateFreq
#
# Description: This configuration parameter specifies the number of iterations
after
# which MDT file is to be updated.
# Every call to addClass or deleteClass will add/delete the given class. These
# in-memory changes will be reflected in nn.mdt only after the specified
# number of such iterations and on application exit.
#
# Valid values: Any integer > 0
# Default value: 5
# Typical value: 5
#-------------------------------------------------------------------------------
ActiveDTWMDTFileUpdateFreq = 100


#-------------------------------------------------------------------------------
# NNMDTFileOpenMode
#
# Description: This configuration parameter specifies the mode for
# opening the mdt file.
#
# Valid values: ascii, binary
# Default Value: ascii
#-------------------------------------------------------------------------------


NNMDTFileOpenMode=ascii
```

17-8 Default config file neuralnet.cfg

```
#-------------------------------------------------------------------------
# neuralnet.cfg
#
# Configuration file for Neural Net Classification Method for
# Lipi Toolkit 3.0
#-------------------------------------------------------------------------

#-------------------------------------------------------------------------
# The standard format for the configuration entries is the name of the
# configuration parameter seperated by an equal to sign and then the value of
# the configuration parameter. For example:
# ConfigurationEntryName = value
#
# Lines starting with a # are commnet lines
#
# A cfg entry is strictly a key value pair and leaving the key without the
# value or specification of a value out of the range is not permitted
#
# If a cfg entry is not specified at all, then default values are used by the
# recognizer
#-------------------------------------------------------------------------

#------------------------------
#PREPROCESSING
#------------------------------

#-------------------------------------------------------------------------
# ResampTraceDimension
#
# Description: The number of target points for resampling. In other words,
# each character will be resampled to this number of points. In case of
# multistroke characters, this number of points will be distributed between
# the strokes in proportion to their lengths in proportion to their initial
# number of points.
#
# Valid values: Any integer > 0
# Units: Points
```

```
# Default value: 60
# Typical value: Average number of points per character in the training data set.
#-------------------------------------------------------------------------------
ResampTraceDimension = 60



#-------------------------------------------------------------------------------
# ResampPointAllocation
#
# Description: Method to be used for point allocation among different strokes
# during resampling. Two schemes have been implemented lengthbased and point
# based. In lengthbased allocation scheme, the number of points allocated to
# each stroke is proportional to the length of the stroke. Length of a stroke
# is calculated as the sum of the distances between each point in the stroke.
# In the pointbased allocation scheme, the target stroke point allocation is
# proportional to the number of points in the initial stroke. In the
# interpointdistbased scheme, the distance between consecutive points is fixed
# resulting in variable number based on the length of the trajectory.
#
# Valid value: [lengthbased | pointbased | interpointdistbased]
# Default value: lengthbased
#-------------------------------------------------------------------------------
ResampPointAllocation = lengthbased

#-------------------------------------------------------------------------------
# NormDotSizeThreshold
#
# Description: This threshold is used to determine whether a character is a dot.
# It is expressed in real length terms (inches) and converted internally to
# points using knowledge of the device's spatial resolution. If the width
# and height are both less than this threshold, then all the points are replaced
# with the center of the of the normalized character, basically to represent it
# as a dot
#
# Valid values: Any real number > 0
# Units: inches
# Default value: 0.01
# Typical value: < 0.1
#-------------------------------------------------------------------------------
```

```
NormDotSizeThreshold = 0.00001

#-------------------------------------------------------------------------
# NormLineWidthThreshold
#
# Description: This threshold is used to detect whether the character is a
# vertical or horizontal line. If only the height is less than this threshold
# then the character is detected as a horizontal line and if only the width is
# less than this threshold then the character is detected as a vertical line.
# Assuming the height is along the y-dimension and width is along the x-
# dimension, during normalization of a horizontal line only the x-coordinates
# are scaled and the y-coordinates are translated to the center of the character,
# with out scaling. Similarly for the vertical line only the y-coordinates are
# normalized and the x-coordinates are translated to the center with out scaling
#
# Valid values: Any real number > 0
# Units: inches
# Default value: 0.01
# Typical value: < 0.1
#-------------------------------------------------------------------------
NormLineWidthThreshold = 0.01

#-------------------------------------------------------------------------
# NormPreserveAspectRatio
#
# Description: This parameter is used to indicate whether the aspect ratio
# has to be preserved during normalization. The aspect ratio is the calculated
# as maximum of (height/width , width/height). The aspect ratio is preserved only
# if the calculated aspect ratio is greater than the threshold value specified
# through NormPreserveAspectRatioThreshold and this configuration variable is
# set to true. If this configuration variable is set to false the aspect ratio
# is not preserved during normalization.
#
# Valid value: [true | false]
# Default value: true
#-------------------------------------------------------------------------
NormPreserveAspectRatio = true
```

```
#---------------------------------------------------------------------------
# NormPreserveAspectRatioThreshold
#
# Description: Aspect ratio is preserved during normalization if the computed
# aspect ratio (max(height/width, width/height)) is greater than this threshold
# and the configuration value NormPreserveAspectRatio is set to true. During
# aspect ratio preserving normalization, the larger of the two dimensions is
# normalized to the standard size and the other dimension is normalized
# proportional to the initial height and width ratio, so that the initial
# aspect ratio is maintained.
#
# Valid values: Any real number >= 1
# Default value: 3
# Typical value: >= 1.5
#---------------------------------------------------------------------------
NormPreserveAspectRatioThreshold = 3

#---------------------------------------------------------------------------
# NormPreserveRelativeYPosition
#
# Description: The relative Y position is the mean of the y-coordinates in the
# input character. During normalization if this parameter is set to true, each
# y-coordinate of the character point is translated by the initial y-mean value,
# so that the mean of the y-coordinates remains the same before and after
# normalization. This is typically used in the word recognition context where
# each stroke of the character has to be normalized separately and the relative
# position of the strokes should be maintained even after normalization.
#
# Valid value: [true | false]
# Default value: false
#---------------------------------------------------------------------------
NormPreserveRelativeYPosition = false

#---------------------------------------------------------------------------
# SmoothWindowSize
#
# Description: The configuration value specifies the length of the moving
# average filter (size of the window) for smoothing the character image.
# If this value is set to N, then each point in the input character is replaced
```

```
# by the average of value of this point, (N-1)/2 points on the right and (N-1)/2
# on the left of this point.
#
# Valid value: Any integer > 0
# Units: Points
# Typical value: 5
# Default value: 3
#-------------------------------------------------------------------------------
SmoothWindowSize = 3


#-------------------------------------------------------------------------------
# PreprocSequence
#
# Description: This variable is used to specify the sequence of preprocessing
# operations to be carried out on the input character sample before extracting
# the features. A valid preprocessing sequence can consist of combination of one
# or more of the functions selected from the valid values set mentioned below.
# The CommonPreProc prefix is used specify the default preprocessing module of
# LipiTk. The user can add his own preprocessing functions in other modules and
# specify them in the preprocessing sequence.
#
# Valid values: Any sequence formed from the following set
# CommonPreProc::normalizeSize;
# CommonPreProc::removeDuplicatePoints;
# CommonPreProc::smoothenTraceGroup;
# CommonPreProc::dehookTraces;
# CommonPreProc::normalizeOrientation;
# CommonPreProc::resampleTraceGroup;
# Default value:
{CommonPreProc::normalizeSize,CommonPreProc::resampleTraceGroup,CommonPreProc::normal
izeSize}
#-------------------------------------------------------------------------------
PreprocSequence={CommonPreProc::normalizeSize,CommonPreProc::resampleTraceGroup,Commo
nPreProc::normalizeSize}


#---------------------------------------
#TRAINING
#---------------------------------------


#-------------------------------------------------------------------------------
```

```
# SeedValueForRandomNumberGenaretor
#
# Description: The generation of pseudo-random numbers is a common task in computer
# simulations. Computational algorithms (pseudorandom number generators) producing
# pseudo-random numbers (sequences of apparently random numbers), are in fact
# completely determined by a shorter initial value, called a seed or key. Any integer
# value can be used for this seed. Different values of seed should produce different
# sequences of pseudo-random numbers depending upon the periodicity of the generating
# algorithm. This quantity is used to initialise the random number generator. Random
# numbers are used to initialize connection weights of the network. Different seed
# values should produce different sequences of pseudo-random numbers depending upon
# the periodicity of the generating algorithm.
#
# Valid value: [Any integer > 0]
# Default value: 426
#-------------------------------------------------------------------------------
SeedValueForRandomNumberGenaretor = 456


#-------------------------------------------------------------------------------
# NormalizationFactor
#
# Description: Input feature components are divided by this quatity because present
# implementation of BackPropagation algorithm needs input feature vector components
# in the range 0 to 1. Usually, this quantity is greater than or equal to the
possible
# maximum value of input feature components.
#
# Valid value: [Any real number grater then 0.0]
# Default value: 10.0
#-------------------------------------------------------------------------------
NeuralNetNormalizationFactor = 10.0


#-------------------------------------------------------------------------------
# NeuralNetLearningRate
#
# Description: This is the parameter of the learning algorithm that controls step
size along
# the steepest descent in the error (system error) surface. Connection weights are
adjusted
# by adding the weight increment multiplied by this quantity to the previous weight.
```

```
The
# learning rate indicates how far in the direction of steepest descent the network weights are
# shifted at each iteration. For example, if this value is 1.0, the shift amount will be equal
# to the value of the resultant gradient. For parctical problem in which the error surface is
# generally nonlinear, a smaller learning rate must be used to slowly and smoothly guide the
# descent towards the optimum weights; therefore, this value is normally between 0 and 1, and
# indicates the proportion of the gradient length that will be traversed in the direction of the
# steepest descent.
#
# Valid value: Any real number from 0-1
# Default value: 0.005
#-------------------------------------------------------------------------------
NeuralNetLearningRate = 0.005


#-------------------------------------------------------------------------------
# NeuralNetMomemtumRate
#
# Description: A quantity involving momentum rate was introduced in the weight modification
# rule of BackPropagation algorithm for incorporating influence of the past iterations during
# current updation of the connection weights.  The momentum introduces a "damping" effect on
# the search procedure, thus avoiding possible oscillations on the error surface by averaging
# gradient components with opposite sign and accelerating the convergence in long flat areas.
# Also, in situations of falling into local minima, it possibly helps to certain limit to avoid
# the same. Momentum may be considered as an approximation to a second-order method, as it uses
# information from the previous iterations.
#
# Valid value: Any real number from 0-1
# Default value: 0.0025
#-------------------------------------------------------------------------------
NeuralNetMomemtumRate = 0.0025
```

```
#-------------------------------------------------------------------------
# NeuralNetTotalError
#
# Description: After presentation of the complete set of training samples to the
network
# during training, the network computes the total error corresponding to this whole
set. A
# threshold for this error is set to decide when the training should be terminated.
#
# Valid value: Any real number from 0-1
# Default value: 0.00001
#-------------------------------------------------------------------------
NeuralNetTotalError = 0.00001


#-------------------------------------------------------------------------
# NeuralNetIndividualError
#
# Description: After each presentation of a training sample to the network during its
# training the system calculates the error value for the particular sample. A
threshold
# for this error is set to decide when the training should be terminated.
#
# Valid value: Any real number from 0-1
# Default value: 0.00001
#-------------------------------------------------------------------------
NeuralNetIndividualError = 0.00001


#-------------------------------------------------------------------------
# NeuralNetHiddenLayersSize
#
# Description: In a full-connected, feed-forward, perceptron neural network the
values
#only move from input to hidden to output layers. All neural networks have an input
layer
#and an output layer, but the number of hidden layers may vary. When there is more
than
#one hidden layer, the output from one hidden layer is fed into the next hidden layer
#and separate weights are applied to the sum going into each layer. For nearly all
problems,
#one hidden layer is sufficient. Two hidden layers may be helpful in a few cases such
as
#modeling data with discontinuities, e.g., a saw tooth wave pattern. Using two hidden
```

```
layers
#rarely improves the model, and it may introduce a greater risk of converging to a local
#minima. There is no theoretical reason for using more than two hidden layers.  Three layer
#models with one hidden layer are recommended.
#
# Valid value: [any integer from 1-50]
# Default value: 1
#-------------------------------------------------------------------------------
NeuralNetHiddenLayersSize = 1


#-------------------------------------------------------------------------------
# NeuralNetHiddenLayersUnitSize
#
# Description: One of the important issues of a MLP network is the choice of number of neurons
# in the hidden layer(s). An inadequate number of neurons causes failure to model complex data,
# and the final result is bound to be poor. If too many neurons are used, the training time may
# become excessively long, and, worse, the network may over fit the data. When overfitting occurs,
# the network will begin to model random noise in the data. Use of a validation set helps to
# detect overfitting. There are several approaches in the literature for automatic selection of the
# optimal number of neurons in the hidden layer. A better approach is to build models using varying
# numbers of hidden neurons and measure the quality of training using cross validation or hold-out
# data not used for training. This is a highly effective method for finding the optimal number of
# neurons, but it is computationally expensive, because many models must be built, and each model
# has to be validated.
#
# Valid value: [any integer]
# Default value: 175:
#-------------------------------------------------------------------------------
NeuralNetHiddenLayersUnitSize = 175:


#-------------------------------------------------------------------------------
```

```
# ReestimateNeuralnetConnectionWeights
#
# Description: Training of MLP may be done in repeated sessions. For example, during
the initial
# session one may continue training for 100 iterations (presentation of the whole set
of training
# samples to the network). Check the network performance on the validation or test
set. In the next
# session the trained network may be further trained for some more iterations
#
# Valid value: [true | false]
# Default value: false
#--------------------------------------------------------------------------------
ReestimateNeuralnetConnectionWeights = false


#--------------------------------------------------------------------------------
# NeuralnetTraningIteration
#
# Description: The number of training iterations is another important factor and it
is
# required to be chosen suitably to get best possible performance from the network.
If
# too many iterations are performed, the training time may become excessively long,
and,
# worse, the network may overfit the data. When overfitting occurs, the network will
begin
# to model random noise in the data. Use of a validation set helps to detect
overfitting.
#
# Valid value: [any integer]
# Default value: 600
#--------------------------------------------------------------------------------
NeuralnetTrainingIteration = 600


#--------------------------------------------------------------------------------
# PrepareTraningSequence
#
# Description:
#
# Valid value: [true | false]
# Default value: true
#--------------------------------------------------------------------------------
```

```
PrepareTrainingSequence = true

#----------------------------------------
#FEATURE EXTRACTION
#----------------------------------------

#-------------------------------------------------------------------------------
# FeatureExtractor
#
# Description: The configuration value is used to specify the feature extraction
# module to be used for feature extraction. The point float feature extraction
# module extracts the x,y,cosine and sine angle features at every point of the
# character.
#
# Valid value: [PointFloatShapeFeatureExtractor|L7ShapeFeatureExtractor|
#               NPenShapeFeatureExtractor|SubStrokeShapeFeatureExtractor]
# Default value: PointFloatShapeFeatureExtractor
#-------------------------------------------------------------------------------
FeatureExtractor=PointFloatShapeFeatureExtractor

#----------------------------------------
#RECOGNITION
#----------------------------------------

#-------------------------------------------------------------------------------
# NNRecoRejectThreshold
#
# Description: Threshold to reject the test sample. If the confidence obtained
# for the recognition of test sample is less than this threshold then the test
# sample is rejected.
#
# Valid value: Any real number from 0-1
# Default value: 0.001
#-------------------------------------------------------------------------------
NNRecoRejectThreshold =0.001

#-----------------------------------------
#      COMMON FOR TRAINING AND RECOGNITION
#-----------------------------------------
```

```
#-------------------------------------------------------------------------
# NNDTWBandingRadius
#
# Description: This configuration parameter specifies the mode for
# opening the mdt file.
#
# Valid values: ascii, binary
# Default Value: ascii
#-------------------------------------------------------------------------

NNMDTFileOpenMode=ascii
```

17-9 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
```

17-10 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>/usr1/000t02.txt 0
<Data-files-path>/usr10/000t01.txt 0
<Data-files-path>/usr100/000t01.txt 0
<Data-files-path>/usr101/000t01.txt 0
<Data-files-path>/usr101/000t02.txt 0
...
...
...
<Data-files-path>/usr27/001t01.txt 1
<Data-files-path>/usr27/001t02.txt 1
<Data-files-path>/usr28/001t01.txt 1
<Data-files-path>/usr28/001t02.txt 1
<Data-files-path>/usr29/001t01.txt 1
<Data-files-path>/usr29/001t02.txt 1
<Data-files-path>/usr3/001t01.txt 1
...
...
...
```

17-11 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>/usr117/026t02.txt 26 #
<Data-files-path>/usr145/029t01.txt 29 #
<Data-files-path>/usr146/018t01.txt 18 #
<Data-files-path>/usr142/001t01.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
<Data-files-path>/usr135/006t02.txt 6
...
...
...
```

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

17-12 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 45: Configurable make settings for Linux

17-13 Module dependencies on Windows

Module	Dynamic Libraries (DLLs)	Static Library
Nn	1 nn.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
ActiveDTW	1 activedtw.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
Neural network	1 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 46: Module dependencies for Windows

17-14 Module dependencies on Linux

Module	Shared Libraries (.so)	Static Library
Nn	1 libnn.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
ActiveDTW	1 libactivedtw.so	1 libcommon.a

	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	**2** libutils.a **3** libshapereccommon.a **4** libfeaturextractor.a
Neural Network	**1** 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 **6** liblogger.so	**1** libcommon.a **2** libutils.a **3** libshapereccommon.a **4** libfeaturextractor.a **5** libwordreccommon.a

Table 47: Module dependencies for Linux

17-15 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
```

17-16 Model data header information file

This model data header information file contains the metadata information about the model data file.

Sample model data header information file

```
<SCRIPT=Hindi>
<DATASETNAME=SchoolData>
<COMMENT= This data is collected from St.Joseph school, Bangalore on July 14th. This contains hindi gestures collected from 1st to 4th std students>
```

Module	Possible values	Description
SCRIPT	TAMIL, DEVANAGARI, GENERAL	Name of the script/language on which this model data created
DATASETNAME	String field	A string holding the name of the dataset. For example: DATASETNAME=SchoolData
COMMENT	String field	Free field to store any text

Table 48: Module dependencies for Linux

17-17 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

18 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.