Reference Manual

Lipi Toolkit 1.1

Table of Contents

1 Introduction

Lipi Toolkit (LipiTk) is a generic toolkit for Online Handwriting Recognition (HWR). It provides a set of script-independent shape recognizers, tools and building blocks which can be used by different kinds of users for different scripts and shapes.

This document describes the source level details of all the core modules of LipiTk such as Preprocessor, PCA, DTW and Boxed Field. The interfaces and the data structures are also discussed in detail.

2 Supported platforms and environment

- Windows XP Professional edition, Windows 2000 Professional edition.
- Redhat Enterprise Linux Edition 3.0,
- GNU Linux 2.6.9.22
- GNU Linux (Ubuntu) 2.6.14.6

3 Overview and architecture

LipiTk is a collection of tools and shape recognizers for Online Handwriting Recognition. It defines the standard interfaces for HWR and also implements them using PCA and DTW shape recognizers. It contains various scripts and tools for the user to train and test the shape recognizer with data samples. It allows the user to build, package and deploy the customized recognizer component created out of LipiTk to other applications which require online HWR.

LipiTk was intended to support both Windows and Linux platforms, hence its design and implementation considers portability related issues for both. Most of the algorithms and tools are implemented using C++ & STL. Only ANSI functions are used for portability. Some of the utilities and scripts are written in Perl.

The major components of the toolkit are described below:

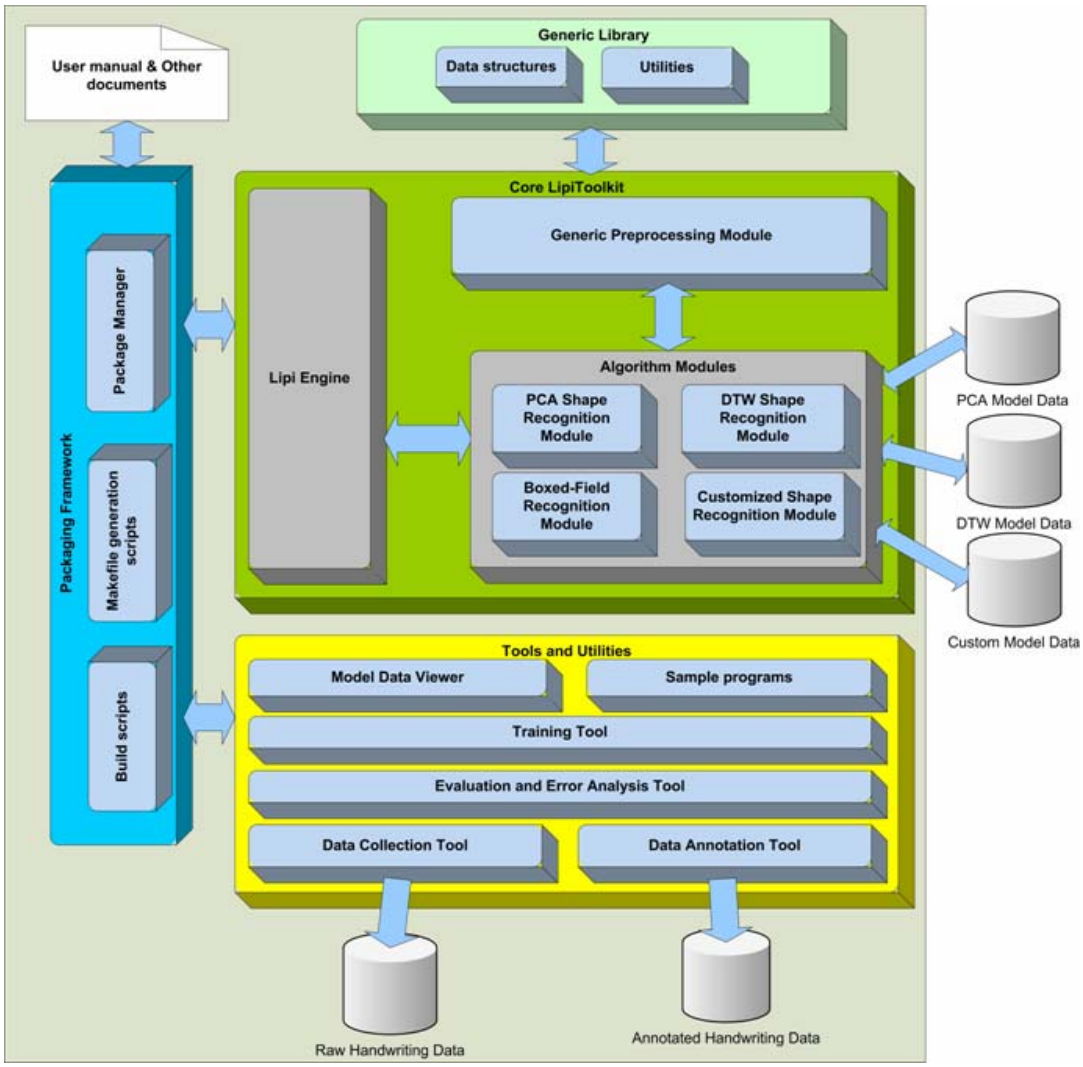


Figure 2. Lipi Toolkit Architecture and Components

Note: There is no annotation tool shipped along with LipiTk in version 1.0. You will need to use Uptools from the Intl. Unipen Foundation for annotation. Refer to Section 5 of the Tools manual for more details.

- **Generic class 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 shared by different algorithms 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 and write LipiTk configuration files, read and write Unipen data files, and so on.

- **HWR algorithms**

LipiTk 1.0 provides implementations of common preprocessing operations, common shape recognition algorithms, as well as a boxed field recognizer which iteratively calls one of the shape recognizers to interpret a boxed field of ink input. The preprocessing module, as well as the shape recognition and boxed field recognition modules are implemented as separate dynamic link libraries that can be loaded at runtime.

- **Generic preprocessing**

The generic preprocessing module provides implementations of commonly used shape/character preprocessing operations such as moving-average smoothing, size normalization, dehooking, and equidistant resampling. All of the operations have configuration options that can be varied using corresponding properties captured in a configuration file.

- **Shape recognition**

The two shape recognition algorithms bundled with LipiTk 1.0 are Subspace-based classification (PCA), and Nearest-Neighbor classification based on Dynamic Time Warping (DTW).

In subspace classification, each shape is represented by a set of principal components computed from a fixed length representation of the online shape, obtained after size normalization and equidistant resampling. The training method provided computes the principal components from the training data, and stores them in a standard binary format.

The DTW implementation uses the same fixed length representation as the subspace classifier, together with a Nearest Neighbor classifier. The training method exposed by the shape recognizer provides a choice of different prototype selection algorithms for prototype reduction.

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

In either case, important parameters (such as the number of principal components) as well as the sequence of preprocessing operations are externally configurable using configuration files.

- **Boxed field recognition**

As mentioned earlier, the boxed field recognizer is useful for recognizing a boxed field of shapes, and in turn invokes a trained shape recognizer on each of the boxes, and uses a simple trellis for decoding the 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.

- **Tools and utilities**

LipiTk 1.0 provides a number of tools and utilities to support the tasks of handwriting data collection, data annotation, and the training and evaluation of shape recognizers.

- **Data collection and annotation tools**

Collection and annotation of handwriting data is an important activity in the data-driven methodology for creating recognition engines. LipiTk 1.0 includes a generic TabletPC-based data collection tool capable of collecting isolated symbols, characters or words from writers. We hope to include in subsequent releases, tools based on Digital Pen and Paper or PDAs, as well as tools running on Linux.

The annotation tool supports the tagging of digital ink with labels corresponding to ground truth, writing style etc. at different levels of an appropriate hierarchy of annotation.

Note: There is no tool shipped along with LipiTk in version 1.0. You will need to use Uptools for annotation. Refer to Tools manual section 5 for more details.

- **Evaluation tool**

The evaluation tool computes statistics related to classification accuracy and performance of the built engine on the test data, and allows visualization of the results in ways that facilitate analysis of the errors. LipiTk 1.0 includes a basic evaluation tool written in Perl which renders top N accuracies and confusion matrices in the form of HTML pages.

- **Utilities**

In addition to the above tools, LipiTk 1.0 also includes a number of scripts to facilitate tasks such as extraction of isolated character data from the annotated data (which may be words), and performing DOS to UNIX conversion of dataset.

- **Packaging framework**

LipiTk provides build scripts to support the creation of specific engines from the source code. These scripts interpret project configuration files and build the necessary source code into libraries and binaries, using a hierarchy of static module-specific Makefiles.

LipiTk also provides scripts for packaging the built engine(s) for deployment, and integration into a pen-based application. The components of the package are fully user-configurable, and the packaging script creates a self-extract package file (or gzipped tar file in the case of Linux) that contains all the components selected for packaging by the user.

Finally, LipiTk provides sample code to assist the application developer in integrating an engine created using LipiTk into his or her application.

- **Lipi engine**

The Lipi engine is the run-time component of engines created using the LipiTk. It is responsible for loading one or more shape/word recognition modules as specified in its configuration file, routing requests for recognition from the user application to the appropriate modules, and returning recognition results to the application.

4 Module documentation

Note: You can use Doxygen to generate the detailed function level information. For details refer Appendix 9.1

4.1 Lipi engine module

The Lipi engine is the run-time component responsible for loading one or more shape/word recognition modules as specified in its configuration file, routing requests for recognition from the user application to the appropriate modules, and returning recognition results to the application.

Sl.no	Function name	Description
1	*createLTKLipiEngine*	Returns the instance handle of LTKLipiEngineModule to the client.
2	*getCurrentVersion*	Client can use this to get the version of this module
3	*startLogging*	Client can use this to start logging
4	*stopLogging*	Client can use this to stop the logging

Note: LTKLipiEngineModule is a singleton class and a client process can have only one instance of LTKLipiEngineModule at anytime.

The *LTKLipiEngineModule* class implements the interface *LTKLipiEngineInterface*. The list of all the methods and their description is given below. The client application calls all the methods with the *LTKLipiEngineModule* instance handle returned by *createLTKLipiEngine*.

LTKLipiEngineModule and its methods:

Sr.no	Function name	Description
1	*initializeLipiEngine*	Reads the lipiengine.cfg and initializes the Lipi engine.
2	*createShapeRecognizer*	Creates shape recognizer object, by passing the logical project name or by passing both project name and profile name (when this function call has one parameter then it is assumed as logical name).
3	*createWordRecognizer*	Creates word recognizer object, by passing the logical project name or by passing both project name and profile name (when this function call has one parameter then it is assumed as logical name).
4	*deleteShapeRecognizer*	Deletes the shape recognizer object created using createShapeRecognizer call.
5	*deleteWordRecognizer*	Deletes the word recognizer object created using createWordRecognizer call.
6	*createRecognitionContext*	Creates recognition context (used in boxed-field recognition).
7	*deleteRecognitionContext*	Deletes the recognition context created using createRecognitionContext call.

4.2 Common preprocessor module

This common preprocessor module implements preprocessing functions which are commonly used across various shape recognizers. i.e. PCA, DTW.

All the preprocessing functions in this module follow a standard function prototype as follows:

<Preproc_func_name>(constLTKTraceGroup&inTraceGroup,LTKTraceGroup& outTraceGroup)

This module can be found under the directory $LIPI_ROOT/src/reco/shaperec/preprocessing.

The table below describes the list of all common preprocessing functions with the descriptions.

The sequence in which these functions need to be executed can be configured in the shape recognizer's configuration file using the attribute PreprocSequence as follows:

PreprocSequence={CommonPreProc::<funcname1>, PCA::<funcname0>,CommonPreProc::<funcname2>…}

Note: No space is allowed in between the function names. Use commas to separate the function names.

Sr.no	Function name	Description
1	*normalizeSize*	Normalizes the size of the incoming trace group to a value specified in configuration file (Refer section 4.3.5)
6	*normalizeOrientation*	Normalizes the orientation of the incoming trace group
7	*smoothenTraceGroup*	Smoothens the given tracegroup using moving average
8	*centerTraces*	Centers the traces of a trace group to the center of its bounding box
9	*dehookTraces*	Dehooks the traces of the tracegroup
10	*removeDuplicatePoints*	Remove consecutively repeating x, y coordinates (thinning)

A sample preprocessing sequence in pca.cfg is given below:

PreprocSequence={CommonPreProc::normalizeSize,PCA::resampleTraceGroup}

In the above example, it calls normalizeSize from the common preprocessor module and resampleTraceGroup function from the PCA shape recognizer.

4.3 Shape recognition interface

LipiTk provides a standard set of interfaces for all the shape recognition modules. This allows the user to dynamically configure and use any new shape recognition module at run-time. This section describes the shape recognition interface in detail.

The class LTKShapeRecognizer defines the standard interface. The header file can be found under $LIPI_ROOT/src/include/LTKShapeRecognizer.h. Any new shape recognition module that you create should derive this interface and implement all the functions.

The methods that need to be implemented by any new shape recognition module are given below:

- **int initialize((string& strProjectName, string &strProfileName)**

Description

This method initializes the shape recognizer. I.e. reading the configuration file like pca.cfg

Input parameters

strProjectName contains the project name and strProfileName contains the profile name of the project

Return values

0 if successful and other value if unsuccessful.

- **int loadModelData()**

Description

This method loads the model data file into memory from the model data file. Model data file should be available under "$LIPI_ROOT/projects/<project folder>/config/<profile folder>" as <algomodulename>.mdt

Input parameters

None

Return values

0 if successful and other value if unsuccessful.

- **int recognize (const LTKTraceGroup& traceGroupObj, const LTKScreenContext& screenContext, const vector<bool>& shapeSubSet,float confThreshold, int numChoices, vector<LTKShapeRecoResult>& results)**

Description

This method is called by the client application to recognize the given traceGroupObj.

Input parameters

traceGroupObj is the trace group which is to be recognized

screenContext holds the co-ordinates of the writing area provided for the set of traces being sent for recognition.

shapeSubSet is a subset of the entire class space which is to be used for recognizing the input shape.

confThreshold is a threshold on the confidence value of the recognized class. This is used as rejection criterion.

numOfChoices is the number of top classes to be returned.

Output Parameters

Results contain shapeId and confidence for each of the top classes.

- **int train (const string& trainingList, string& strModelDataHeaderInfoFile, string &comment, string &dataset)**

Description

This method is invoked to train the recognizer with the set of model data. At the end of training, this method creates the model data file under $LIPI_ROOT/projects/<project folder>/config/<profile folder> as <algomodulename>.mdt

Input parameters

trainingList is the name of the file containing the list of files to be used for training each of the classes.

strModelDataHeaderInfoFile is the name of the file containing the list of attribute value pairs which are going to be part of the model data header.

comment is the string value which contains the general comment on the model data file

dataset is a string which contains the name of the dataset used for training

- **int unloadModelData()**

Description

This method frees the model data information used for recognition from memory

Input parameters

None

Return values

0 if successful and other value if unsuccessful.

- **int setDeviceContext(LTKCaptureDevice& deviceinfo)**

Description

This method allows the client application to set the device parameters to the shape recognizer. The application needs to set following parameters in the capture device structure:

- Sampling rate (points per second)
- Horizontal and vertical resolution (dots per inch)
- Latency time (ms)
- Flag denoting whether the device sampling is uniform

Input parameters

deviceinfo – structure which contains the device parameters

Return values

0 if successful and other value if unsuccessful.

- **int getLastError ()**

Description

This method is called by the client application to get the last error that occurred during any operation within the shape recognizer module.

Input parameters

None

Return values

Error code. [Refer section 5 for more details on error codes]

4.4 PCA module

The PCA module is an implementation of the PCA shape recognition algorithm. This algorithm uses distance to the subspace formed by the principal components of each class to find the nearest class to a given test sample. The following sections document the PCA shape recognition module in detail.

LipiEngine reads from the config file and initializes the PCA shape recognition module. The PCA module has three classes – PCAFeatureExtractor, PCAShapeModel and PCAShapeRecognizer.

These are described in detail below:

4.4.1 PCAFeatureExtractor

This class is used to extract the features from a given character. The following method is used to extract the features from the preprocessed trace group.

- **static int extractFeatures (LTKTraceGroup traceGroupObj, floatVector& featureVector);**

Description

This static function extracts the features from a trace group object. The feature vector is formed by concatenating the x and y channels of the trace group object.

Input parameters

traceGroupObj contains the trace group.

Output parameters

featureVector contains the extracted features.

4.4.2 PCAShapeModel

This class represents the model data for recognition. The model data comprises a set of eigenvalues and eigenvectors for each class. This class has methods to get and set the shape models.

4.4.3 PCAShapeRecognizer

This class contains the methods that implement a PCA based shape recognizer. The methods in this class can be categorized into two groups. The first group contains methods for training and the second for recognition.

Training is performed by taking the training list file as input, performing PCA on the samples of each class, and writing the computed eigenvectors and eigenvalues of each class in the model data file. The eigenvectors and eigenvalues of each class form the shape model.

Recognition function predicts the class label of the input sample by finding the distance of the sample to the subspaces formed by shape models of each class. The top N nearest classes along with confidence measures are returned in the result data structure.

- **int train (const string& trainingList, string& strModelDataHeaderInfoFile, string &comment, string &dataset)**

Description

This is the train method of the PCA shape recognizer.

Input parameters

trainingList is the name of the file containing the list of files to be used for training each of the classes.

strModelDataHeaderInfoFile is the name of the file containing the list of attribute value pairs which are going to be part of the model data header.

comment is the string value which contains the general comment on the model data file

dataset is a string which contains the name of the dataset used for training

- **int preprocess (const LTKTraceGroup& inTraceGroup, LTKTraceGroup& outTraceGroup)**

Description

This function calls the preprocessing methods in order, as specified in the preprocessing sequence of the config file (pca.cfg).

Input parameters

inTraceGroup is the input trace group.

Output parameters

outTraceGroup is the preprocessed trace group.

- **int computeEigenVectors (vector< vector<float> > &covarianceMatrix, int n, vector<float> &eigenValues, vector< vector<float> > &eigenVectorMatrix, int& nrot)**

Description

This function calculates the eigenvectors and eigenvalues of the input matrix.

Input parameters

covarianceMatrix is the covariance matrix of the samples of a class.

n is the size of the covariance matrix.

Output parameters:

eigenValues are the eigenvalues of the input covariance matrix.

eigenVectorMatrix is the matrix formed by the eigenvectors of the input covariance matrix.

nrot represents the number iterations taken by the eigenvector computation algorithm to converge.

- **int loadModelData ()**

Description

This method loads the shape models from the model data file.

- **Int recognize(constLTKTraceGroup&traceGroupObj,const TKScreenContext&screenContext, const vector<bool>& shapeSubSet, float confThreshold, int numChoices , vector<LTKShapeRecoResult>& results)**

Description

This is the recognize method of the PCA classifier.

Input parameters

traceGroupObj is the trace group which is to be recognized

screenContext holds the co-ordinates of the writing area provided for the set of traces being sent for recognition.

shapeSubSet is a subset of the entire class space which is to be used for recognizing the input shape.

confThreshold is a threshold on the confidence value of the recognized class. This is used as rejection criterion.

numOfChoices is the number of top classes to be returned.

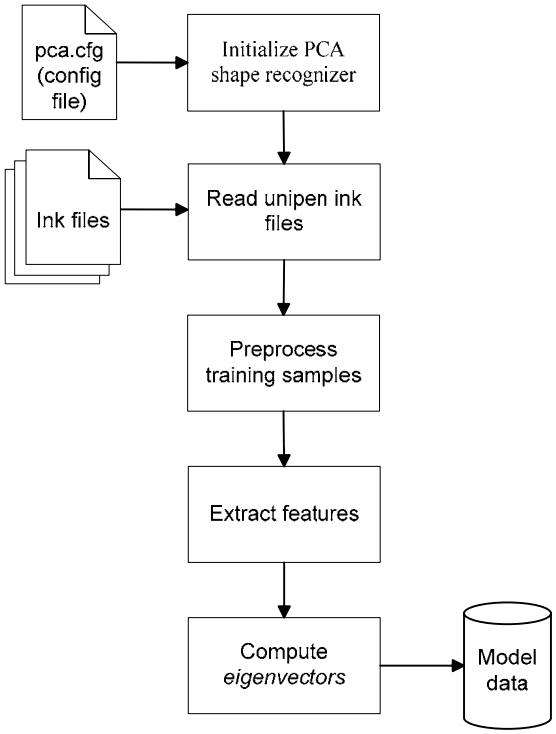
Output Parameters

results contains *shapeId* and confidence for each of the top classes.

4.4.4 Flow diagrams

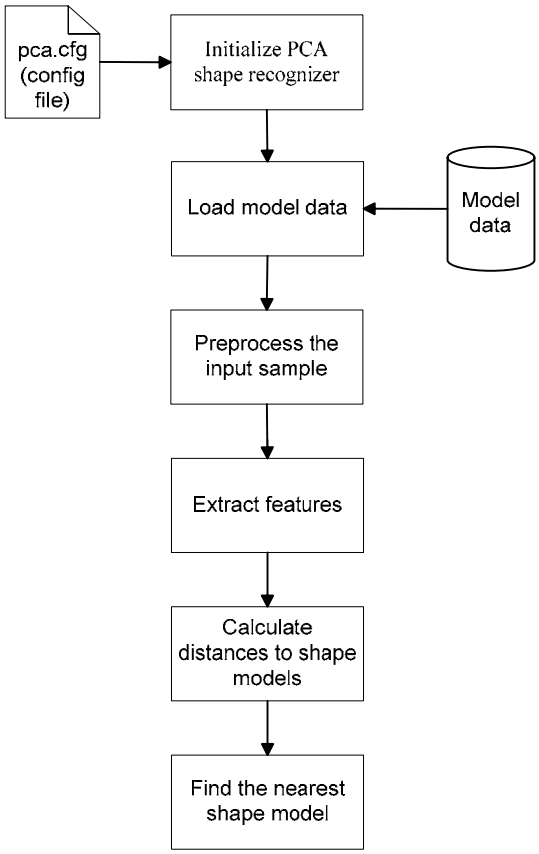
The flow of the training and testing modules are depicted below

Training



For details on training, refer User Manual section 12.

Recognize



For more details on recognition, refer User Manual section 12.

4.4.5 Configuration attributes

The PCA shape recognizer requires project, profile and configuration to be defined for each project. The project files should be under the project root, which is a subdirectory of the $LIPI_ROOT/projects directory. For example consider a project for the recognition of English. The project root could be $LIPI_ROOT/projects/eng_alpha (PROJROOT), where eng_alpha is the logical name of the project. The config directory under the PROJROOT contains project.cfg file, which has project details (shaperec, wordrec etc.) and number of shapes. Each project can have one or more profiles, one each for different algorithms used for shape recognition or different configuration parameters used in the same algorithm. The different profiles are stored as subdirectories under PROJROOT/config. The settings will be loaded from the profile that is requested at the time of creation of the shape recognizer. Each profile directory should contain the profile.cfg file and the configuration files corresponding to the algorithm used for shape recognition. In order to use the PCA algorithm for shape recognition the ShapeRecognizer attribute in profile.cfg should be set to pca and the profile directory should have the pca.cfg file.

Note: The values given against each attribute for pca.cfg are the factory defaults and to override the defaults, keep only the attributes with values which are being overridden. If the pca.cfg file is not present, then all the factory defaults are assumed automatically.

Filename	Path	Config file attributes (attribute = value)	Description
project.cfg	PROJROOT/config	ProjectType = SHAPEREC	This can be SHAPEREC or WORDREC depending on the input for recognition, which could be either an isolated character or a word
		NumShapes = 26	Number of distinct shapes in the shape set to be recognized
profile.cfg	PROJROOT/config/<profile>	ShapeRecMethod= pca	The algoritm to be used for shape recognition
pca.cfg	PROJROOT/config/<profile>	ReverseFeatureVectors = false	This flag denotes whether the stroke direction has to be reversed (in addition to the normal direction) to find out the distance to the shape models. [true OR false]
		TraceDimension = 60	The number of points to which the ink sample has to be resampled. Generally, it's value is set to the average number of points per character in the data set.
		NormalizedSize = 10	The coordinates of the input ink sample are normalized to this size. [1-10]
		SizeThreshold = 0.001	If the width or height is less than this threshold, ink is not renormalized in that dimension. This attribute is used only if the parameter PreserveAspectRatioThreshold is set to false
		AspectRatioThreshold = 5	This is used during size normalization <normalizeSize>. Aspect ratio is preserved if aspect ratio of the ink sample is above this threshold.
		DotThreshold = 0.1	This is used in <normalizeSize>. If the size of both dimensions are less than this threshold (in inch) this is normalized to NormalizedSize/2 value.
		NumEigenVectors = 11	This represents the number of eigenvectors to be used for recognition. The value of this attribute can vary from a minimum value of 1 to a maximum value, which is the size of feature vector. (2*TraceDimension)
		LoopThreshold = 0.25	This is used in the <normalizeOrientation> function. This parameter is used to determine whether a stroke is a loop or not. The range of this

			attribute is [0 - 1].
		HookLengthThreshold1 = 0.017	These thresholds are used in dehooking. A stroke segment is declared as hook, if length of the stroke is less than threshold 1 OR if the length of the stroke is less than threshold 2 and angle is less than AngleThreshold. The Valid range for HookLengthThreshold1 and HookLengthThreshold2 is [0 - 1]. The valid range for HookAngleThreshold is [0-90] degrees.
		HookLengthThreshold2 = 0.13	
		HookAngleThreshold = 10	
		SmoothFilterLength = 3	The value of this attribute is the length of the filter to be used in smoothing the traces. Valid range of this attribute is [1 - TraceDimension].
		DistanceMeasure = normal	This attribute is used to specify the distance function to be used for calculating the similarity. It can be either a weighted distance measure or a normal distance measure. The value this attribute takes is either normal or weighted.
		QuantizationStep = 5	The number of points allocated to each trace will be a multiple of this attribute. Value of this attribute << TraceDimension.
		PreprocSequence={CommonPreProc::normalizeSize, PCA::resampleTraceGroup,CommonPreProc::normalizeSize}	Sequence of preprocessing [Note: Refer section 6.1 for more details on the common preprocessing functions]
		PreserveAspectRatioThreshold=true	[true/false] If set to true the aspect ratio is preserved , depending on whether the aspect ratio of the ink sample is greater than the AspectRatioThreshold value. If set to false the aspect ratio is not preserved.
		PreserveRelativeYposition =false	[true/false] If set to true the relative y-position of the ink sample is preserved.

Source directory hierarchy

The PCA code is under $LIPI_ROOT/src/reco/shaperec/pca. The directory $LIPI_ROOT/src/reco/shaperec/common contains the code for common classes such as *LTKShapeRecoResult* etc. The directory $LIPI_ROOT/src/reco/shaperec/preprocessing contains the source for the preprocessing modules.

4.4.6 Build instructions

On Windows

- Make sure that the $LIPI_ROOT environment variable is set to lipitk directory
- Change directory to $LIPI_ROOT/src/reco
- Execute nmake /f Makefile.win pca (this builds all the dependent modules)

On Linux

- Make sure that the $LIPI_ROOT environment variable is set to lipitk directory
- Change directory to $LIPI_ROOT/src/reco directory
- Execute make –f Makefile.linux pca (this builds all the dependent modules)

4.4.7 Required libraries

Static libraries

Libraries required ($LIPI_ROOT/src/lib) Windows/Linux	Description
utils.lib/libutils.a	Utilities to read/write UNIPEN ink, etc.
common.lib/libcommon.a	Common data structures to represent and process the ink
shapereccommon.lib/ libshapereccommon.a	Shape recognition specific data structures

DLL/SO

Libraries required ($LIPI_ROOT/lib) Windows/Linux	Remarks
pca.dll/libpca.so	PCA implementation
preproc.dll/libpreproc.so	Preprocessing module
lipiengine.dll/liblipiengine.so	Interface to the create/delete shape recognizers

4.4.8 Exported functions

- createShapeRecognizer
- deleteShapeRecognizer
- getCurrentVersion
- startLogging
- stopLogging

4.4.9 Support header files

Header File	Location	Remarks
LTKshapeRecognizer.h	$LIPI_ROOT/src/include	Defines the shape recognizer interface
LTKMacros.h	$LIPI_ROOT/src/include	Defines global macros which are used across LipiTK
LTKInc.h	$LIPI_ROOT/src/include	Generic include file which includes all standard headers
LTKTypes.h	$LIPI_ROOT/src/include	Defines all the ink specific data types
LTKTrace.h, LTKTraceGroup.h	$LIPI_ROOT/src/include	Defines ink data types that are used to store ink info
LTKErrors.h	$LIPI_ROOT/src/include	Defines all the errors

4.4.10 Compile flags

Linux: None

Windows: Use always “Multithreaded” runtime library option on release builds and “Debug Multithreaded” runtime library option on debug builds (MT and MTd)

4.5 DTW module

The DTW module is an implementation of the DTW shape recognition algorithm. This algorithm implements the 1 nearest neighbor algorithm that uses DTW distance as a distance measure. The following sections document the DTW shape recognition module in detail.

LipiEngine reads from the config file and initializes the DTW shape recognition module. The DTW module has three classes – DTWFeatureExtractor, DTWShapeModel and DTWShapeRecognizer.

These have been described in detail below:

4.5.1 DTWFeatureExtractor

This class is used to extract the features from a given character. The class implements dominant point selection which is a down-sampling technique for feature selection.

- **static int getQuantisedSlope(const Character& character, vector<int>& qslopeVector)**

Description

This static function determines the quantized slopes at the points in the trace group

Input parameters

character is a vector of x-y coordinates of the ink sample.

Output parameters

qSlopeVector contains the quantized slope values at each point.

- **static int determineDominantPoints(const vector<int>& qSlopeVector,vector<int>& dominantPts,int flexibilityIndex)**

Description

This static function determines the dominant points of the trace group from the quantized slope values based on the value of the flexibility index. Dominant points are points where the change in slope exceeds the threshold controlled by flexibility index. Flexibility index takes values 0, 1 and 2 respectively.

Input Parameters

qSlopeVector contains the quantized slope values at each point.

flexibilityIndex takes values 0,1 and 2 respectively and determines the threshold during dominant point selection. Greater the flexibility index, higher the threshold, resulting in fewer dominant points.

Output Parameters

dominantPoints contains the indices of the dominant points in the tracegroup.

- **static int extractFeatures(const Character& incharacter, int flexibilityIndex, Character& outcharacter)**

Description

This static function takes in the preprocessed trace group and the value of the flexibility index and returns the extracted dominant points. This function in turn calls the getQunatisedSlope function and determineDominantPoints function.

Input Parameters

incharacter is the in-trace group.

flexibilityIndex takes care of the level of downsampling.

Output Parameters

out character contains the features extracted.

4.5.2 DTWShapeModel

This class represents the model data for recognition. The model data comprises a set of prototypes for each class. This class has methods to get and set the shape models.

4.5.3 DTWShapeRecognizer

This class contains the methods that implement a DTW based shape recognizer. The methods in this class can be categorized into two groups. The first group contains methods for training and the second for recognition.

Training is performed by taking the training list file as the input, performing Prototype Selection on the samples of each class, and storing the prototypes in a model file. Prototype Selection is performed using either of the two techniques – Accumulative or Hierarchical, and the choice of which of these to be used can be set in the configuration file. The prototypes of each class resulting from the clustering techniques form the shape models.

Recognition function predicts the class label of the input sample by finding the distance of the sample to the classes according to the 1-Nearest Neighbor rule. DTW distance is used as the distance metric in the 1-Nearest Neighbor classifier. The top N nearest classes along with confidence measures are returned in the result data structure.

There are three algorithms to compute the DTW distance between two samples – DTW, FASTDTW and Keogh Banding. By default the DTW distance is computed. In order to switch on the use of FASTDTW, the user needs to specify #define FASTDTW in the common module. Similarly in order to switch on the Keogh's Banding technique, the user needs to specify #define KEOGH.

- **int train (const string& trainingList, string& strModelDataHeaderInfoFile, string &comment, string &dataset)**

Description

This is the train method of the DTW shape recognizer.

Input parameters

trainingList is the name of the file containing the list of files to be used for training each of the classes.

strModelDataHeaderInfoFile is the name of the file containing the list of attribute value pairs which are going to be part of the model data header.

comment is the string value which contains the general comment on the model data file.

dataset is a string which contains the name of the dataset used for training.

- **int preprocess (const LTKTraceGroup& inTraceGroup, LTKTraceGroup& outTraceGroup)**

Description

This function calls the preprocessing methods in order, as specified in the preprocessing sequence of the config file (dtw.cfg).

Input parameters

inTraceGroup is the input trace group.

Output parameters

outTraceGroup is the preprocessed trace group.

- **int loadModelData ()**

Description

This method loads the shape models from the model data file.

- **int recognize (const LTKTraceGroup& traceGroupObj, const LTKScreenContext& screenContext, const vector<bool>& shapeSubSet, float confThreshold, int numChoices , vector<LTKShapeRecoResult>& results)**

Description

This is the recognize method of the DTW classifier.

Input parameters

traceGroupObj is the trace group which is to be recognized

screenContext holds the co-ordinates of the writing area provided for the set of traces being sent for recognition.

shapeSubSet is a subset of the entire class space which is to be used for recognizing the input shape.

confThreshold is a threshold on the confidence value of the recognized class. This is used as rejection criterion.

numOfChoices is the number of top classes to be returned.

Output Parameters

results contains *shapeId* and confidence for each of the top classes.

- **int computeDTWDistance(const Character& train, const Character& test,float bestSoFar, distPointer localDistance, float& distanceDTW)**

Description

This function computes the DTW distance between the training prototypes and the input character.

Input Parameters

train corresponds to the training prototype.character

test corresponds to the input character.

bestSoFar is the stopping criteria for the DTW distance matrix computation, which is used to terminate the matrix computation when the DTW distance exceeds a certain threshold.

localDistance function as input to calculate the Euclidean distance between two points

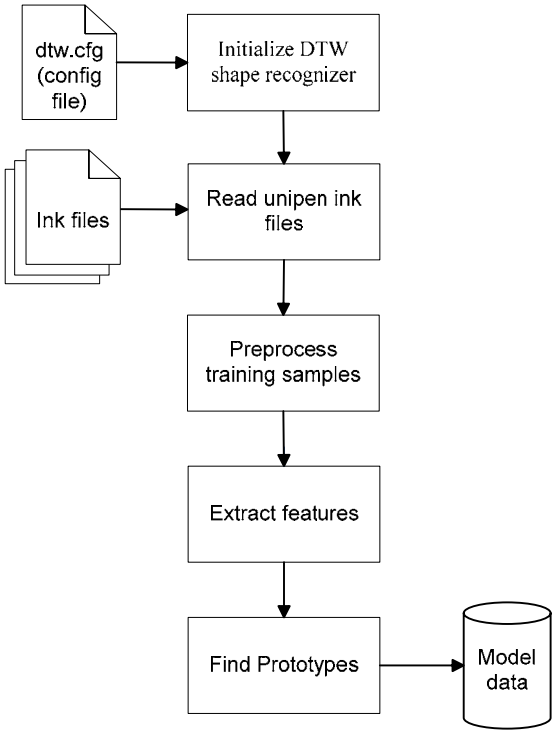
Output Parameters

DistanceDTW contains the DTW distance between the two characters.

4.5.4 Flow diagrams

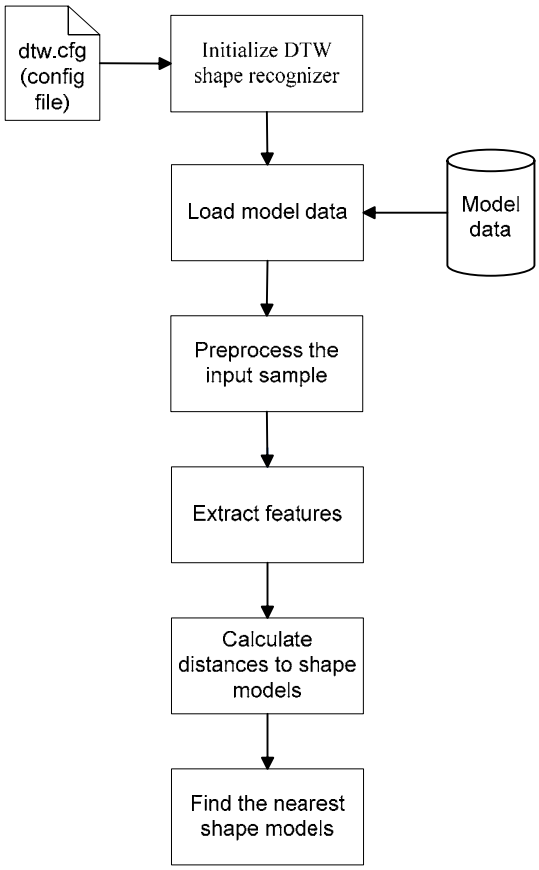
The flow of the training and testing modules are depicted below:

Training



For details on training, refer User Manual section 12.

Recognize



For more details on recognition, refer User Manual section 12.

4.5.5 Configuration attributes

The DTW shape recognizer requires project, profile and configuration to be defined for each project. The project files should be under the project root, which is a subdirectory of the $LIPI_ROOT/projects directory. For example consider a project for the recognition of English. The project root could be $LIPI_ROOT/projects/eng_alpha (PROJROOT), where eng_alpha is the logical name of the project. The config directory under the PROJROOT contains project.cfg file, which has project details (shaperec, wordrec etc.) and number of shapes. Each project can have one or more profiles, one each for different algorithms used for shape recognition or different configuration parameters used in the same algorithm. The different profiles are stored as subdirectories under PROJROOT/config. The settings will be loaded from the profile that is requested at the time of creation of the shape recognizer. Each profile directory should contain profile.cfg file and the configuration files corresponding to the algorithm used for shape recognition. In order to use the DTW algorithm for shape recognition the ShapeRecognizer attribute in profile.cfg should be set to dtw and the profile directory should have the dtw.cfg file.

Note: The values given against each attribute for dtw.cfg are the factory defaults and to override the defaults, keep only the attributes with values which are being overridden. If the dtw.cfg file is not present, then all the factory defaults are assumed.

Filename	Path	Config file attributes (attribute = value)	Description
project.cfg	PROJROOT/ config	Project Type = SHAPEREC	This can be SHAPEREC or WORDREC depending on the input for recognition, which could be either an isolated character or a word
		NumShapes = 26	Number of distinct shapes in the shape set to be recognized. If this parameter is set to "dynamic" the NumShapes can be variable. This Feature is useful if dynamic addition/deletion of shapes happen. However, the shape recognizer will not verify number of shapes across the training and recognition phases.
profile.cfg	PROJROOT/ config/<profile>	ShapeRecMethod = dtw	The algorithm to be used for shape recognition
dtw.cfg	PROJROOT/ config/<profile>	ReverseFeatureVectors = false	This flag denotes whether the stroke direction has to be reversed (in addition to the normal direction) to find out the distance to the shape models. [true or false]
		TraceDimension = 60	The number of points to which the ink sample has to be resampled. Generally, its value is set to the average number of points per character in the data set.
		NormalizedSize = 10	The coordinates of the input ink sample are normalized to this size. [1-10]
		SizeThreshold = 0.001	If the width or height is less than this threshold, ink is not renormalized in that dimension. This attribute is used only if the parameter PreserveAspectRatioThreshold is set to false
		AspectRatioThreshold = 5	This is used during size normalization <normalizeSize>. Aspect ratio is preserved if aspect ratio of the ink sample is above this threshold.
		DotThreshold = 0.1	This is used in <normalizeSize>. If the size of both dimensions are less than this threshold (in inch) this is normalized to NormalizedSize/2 value.
		FlexibilityIndex = 0	Flexibility index at which the points are to be matched [0, 1, 2]. Flexibility Index = 0 correspond to considering all the points.
		PrototypeSelection=accumulative	Protoype selection method. Values are [accumulative or clustering]
		Banding=40	Banding radius for DTW Computation

		PrototypeReductionFactor = automatic	This config parameter is used only when the prototype selection is clustering [automatic | none | 1-100] The default value of this parameter is automatic When the value of this config parameter is set to automatic number of clusters is 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
		Numfeatures=2	Number of features <currently x and y features>
		PreprocSequence={CommonPreProc::normalizeSize,DTW::resampleTraceGroup1,CommonPreProc::normalizeSize	Sequence of preprocessing [Note: Refer section 6.1 for more details on the common preprocessing functions]
		PreserveAspectRatioThreshold=true	[true/false] If set to true the aspect ratio is preserved , depending on whether the aspect ratio of the ink sample is greater than the AspectRatioThreshold value. If set to false the aspect ratio is not preserved. Default value of this parameter is true
		PreserveRelativeYposition=false	[true/false] If set to true the relative y-position of the ink sample is preserved. This parameter is set to false by default
		DTWEuclideanFilter = all	This config parameter is used to set the number of nearest meighbours to be considered for calculating dtw [all|1-N] all implies do not use the filter and pass all samples for the dtw computation. Any number k(1 < k < N, N is the size of prototype set) will use Euclidean filter and pass k samples to the DTW computation

4.5.6 Source directory hierarchy

The DTW code is under $LIPI_ROOT/src/reco/shaperec/dtw. The directory common under $LIPI_ROOT/src/reco/shaperec directory contains the code for common classes such as LTKShapeRecoResult etc. The directory preprocessing under $LIPI_ROOT/src/reco/shaperec directory contains the source for the preprocessing modules.

4.5.7 Build instructions

On Windows

- Make sure that the $LIPI_ROOT environment variable is set to lipitk directory
- Change directory to $LIPI_ROOT/src/reco
- Execute nmake /f Makefile.win dtw (this builds all the dependent modules)

On Linux

- Make sure that the $LIPI_ROOT environment variable is set to lipitk directory
- Change directory to $LIPI_ROOT/src/reco directory
- Execute make –f Makefile.linux dtw (this builds all the dependent modules)

4.5.8 Required Libraries

Static libraries

Libraries required ($LIPI_ROOT/src/lib) Windows/Linux	**Description**
utils.lib/libutils.a	Utilities to read/write UNIPEN ink, etc.
common.lib/libcommon.a	Common data structures to represent and process the ink
shapereccommon.lib/ libshapereccommon.a	Shape recognition specific data structures

DLL/SO

Libraries required ($LIPI_ROOT/lib) Windows/Linux	**Remarks**
dtw.dll/libdtw.so	DTW implementation
preproc.dll/libpreproc.so	Preprocessing module
lipiengine.dll/liblipiengine.so	Interface to the create/delete shape recognizers

4.5.9 Exported functions

- createShapeRecognizer
- deleteShapeRecognizer
- getCurrentVersion
- startLogging
- stopLogging

4.5.10 Support header files

Header File	Location	Remarks
LTKshapeRecognizer.h	$LIPI_ROOT/src/include	Defines the shape recognizer interface
LTKMacros.h	$LIPI_ROOT/src/include	Defines global macros which are used across LipiTK
LTKInc.h	$$LIPI_ROOT/src/include	Generic include file which includes all standard headers
LTKTypes.h	$$LIPI_ROOT/src/include	Defines all the ink specific data types
LTKTrace.h, LTKTraceGroup.h	$$LIPI_ROOT/src/include	Defines ink data types that are used to store ink info
LTKErrors.h	$LIPI_ROOT/src/include	Defines all the errors

4.5.11 Compile flags

Linux: None

Windows: Use always “Multithreaded” runtime library option on release builds and “Debug Multithreaded” runtime library option on debug builds (MT and MTd)

4.6 Word Recognition Interface

LipiTk provides a standard set of interfaces for all the word recognition modules. This allows the user to dynamically configure and use any new word recognition module at run-time. This section describes the word recognition interface in detail.

The class *LTKWordRecognizer* defines the standard interface. The header file can be found under $LIPI_ROOT/src/include/ LTKWordRecognizer.h. Any new word recognition module that you create should derive this interface and implement all the functions.

The method that needs to be implemented by any new word recognition module is given below:

- **int initialize((string& strProjectName, string &strProfileName)**

Description

This method initializes the word recognizer. It initializes the members and loads the model data.

Input parameters

strProjectName contains the project name and *strProfileName* contains the profile name of the project.

Return values

0 if successful and other value if unsuccessful.

- **int processInk (LTKRecognitionContext& rc)**

Description

This method is called from recognition context whenever new traces are added to it. The recognizer needs to process the new traces in this method and update the internal state.

Input parameters

rc is the reference of the recognition context that contains the new traces.

Return values

0 if successful and other value if unsuccessful.

- **int endRecoUnit()**

Description

This function is called from the recognition context to notify the end of logical segment in the input stream. This information could be used in constraining the recognizer choices.

Input parameters

None

Return values

0 if successful and other value if unsuccessful.

- **int recognize (LTKRecognitionContext& rc)**

Description

The recognition results are updated in the recognition context after this call.

Input parameters

rc contains the recognition context for the current recognition

Return values

0 if successful and other value if unsuccessful.

- **int reset (int resetParam = 0)**

Description

This method reset the recognizer.

Input parameters

By default it is 0 and it is used to reset.

Return values

0 if successful and other value if unsuccessful.

- **int unloadModelData()**

Description

This method unloads all the model data. Call initialize to re-initialize the word recognizer.

Input parameters

None

Return values

0 if successful and other value if unsuccessful.

- **int getLastError ()**

Description

This method is called by the client application to get the last error that occurred during any operation within the word recognizer module.

Input parameters

None

Return values

Error code [Refer section 5 for more details on error codes].

4.7 Word recognition module

The word recognition module contains algorithms for word level recognition. The module consists of a Word Recognition Interface (*LTKWordRecognizer*), a

Recognition Context (*LTKRecognitionContext*) and a Boxed-field recognizer (*BoxFieldRecognizer*).

The application sets the context of recognition (such as device parameters or UI information) and input ink to be recognized in a recognition context object. The recognition context object invokes a word recognizer (e.g. Boxed-field recognizer) for recognition.

4.7.1 LTKRecognitionContext

This class holds the ink for recognition, context information such as UI and device parameters, language models and the results of recognition. The object exposes methods for adding ink, setting the UI and device parameters, retrieval of results etc. The client application creates and maintains different recognition context objects for different fields (numerals, alpha etc). The main methods in LTKRecognitionContext are listed below:

- **int setWordRecoEngine(LTKWordRecognizer *wordReco)**

This method sets the pointer to the word recognizer to be used in the recognition context. The recognition context invokes word recognizer methods via this pointer

- **int addTrace (const LTKTrace& trace)**
- **int addTraceGroup (const LTKTraceGroupVector& fieldInk)**

These methods are used to add digital ink in the recognition context.
trace is a single trace to be added to the recognition context
fieldInk is a vector of *LTKTraceGroup* objects.
These methods call *processInk* method of the recognizer.

- **const LTKTraceVector& getAllInk () const**

This method returns a reference to the stored ink data in the recognition context.

- **int beginRecoUnit ()**
- **int endRecoUnit ()**

These methods mark begin and end of logical segments in the input stream. An empty stroke added between the beginRecoUnit and endRecoUnit will insert a space in the output string.

- **int recognize ()**

The client application calls this method for recognition. This in turn calls the word recognizer's recognize method.

- **int reset (int resetParam)**

This function resets different components of the recognition context. The reset parameter can have following values

Value	Description
LTK_RST_INK	Clear all the Ink in the recognition context
LTK_RST_RECOGNIZER	Calls reset method of the word recognizer
LTK_RST_ALL	Clear the Ink and calls the reset method of the word recognizer

- **int clearRecognitionResult ()**

This method clears all the ink, recognition results stored in the recognition context. Also calls the reset method of the recognizer.

- **int setDeviceContext (const LTKCaptureDevice& dc)**
- **const LTKCaptureDevice& getDeviceContext () const**

Methods to set/get the device parameters. The application needs to set the following parameters in the capture device structure:

  - sampling rate (points per second)
  - horizontal and vertical resolution (dots per inch)
  - latency time (ms)
  - flag denoting whether the device sampling is uniform

- **int setScreenContext (const LTKScreenContext& sc)**
- **const LTKScreenContext& getScreenContext ( ) const**

Methods to set/get user interface parameters. The screen context structure contains bounding box information and a list of horizontal and vertical lines.

- **int setFlag(string key, int value)**
- **int getFlag(string key)**

These methods set/get flags for recognition in the recognition context.

For boxed-field recognition the flags are:

<table>
<tr><th>Key</th><th>Values and description</th></tr>
<tr><td>REC_UNIT_INFO</td><td>This flag specifies the interpretation of each logical segment. Different possible values are
<table><tr><td>REC_UNIT_UNKNOWN</td><td>No segmentation information available</td></tr><tr><td>REC_UNIT_SYMBOL</td><td>Symbol level segmentation available</td></tr><tr><td>REC_UNIT_CHAR</td><td>Character level segmentation available</td></tr><tr><td>REC_UNIT_WORD</td><td>Word level segmentation available</td></tr></table></td></tr>
<tr><td>REC_MODE</td><td>This flag specifies the recognition mode
<table><tr><td>REC_MODE_BATCH</td><td>All ink is updated in the recognition context before recognition</td></tr><tr><td>REC_MODE_STREAMING</td><td>Streaming mode recognition. The ink is added as collected from the UI</td></tr></table></td></tr>
</table>

- **int addRecognitionResult (const LTKWordRecoResult& result)**

This method called by the word recognizer sets the results in the recognition context.

result contains vector of Unicode ids corresponding to the result word and associated word confidence.

- **int getTopResult (LTKWordRecoResult& result)**
- **int getNextBestResults (int numResults, LTKWordRecoResultVector& results)**

These methods are called by the application to retrieve results from the recognition context. The *LTKWordRecoResult* structure contains the result word with associated confidence value. The parameter numResults denotes the maximum number of results to be retrieved.

- **int setNumResults (int numResults)**
- **int getNumResults () const**

These methods set/get the number of results required from the word recognition shape recognizer.

- **int setConfidThreshold (float thresh)**
- **float getConfidThreshold () const;**

These methods set/get the confidence threshold for word recognition.

4.7.2 BoxedFieldRecognizer

Boxed-field recognizer (*BoxedFieldRecognizer* class) implements the word recognizer interface and performs word recognition when character level segmentation is available. The word recognizer invokes a trained shape recognition module such as PCA or DTW for recognizing individual characters. The shape recognizer returns multiple recognition choices for each character. A trellis is constructed with the recognition alternatives and a Dynamic Programming (DP) search is performed to identify the best possible paths. These choices are converted to Unicode ids and returned to the application. This module has the following methods.

- **int initialize (string& projectName, string& profileName)**

This function loads all configuration parameters and creates the shape recognizer object.

- **int processInk (LTKRecognitionContext& rc)**

The Ink in the recognition context is sent for recognition if a character is completed and the recognition flag is set to streaming.

- **int recognize (LTKRecognitionContext& rc)**

results are updated in the recognition context object (rc) after this call.

- **Int updateRecognitionResults (const vector<LTKShapeRecoResult>& results, LTKRecognitionContext& rc)**

This method updates the word recognition result with new shape recognition choices (results). rc is the recognition context.

- **int reset (int resetParam)**

This method reset the recognizer.*resetParam* is not used currently.

- **int unloadModelData()**

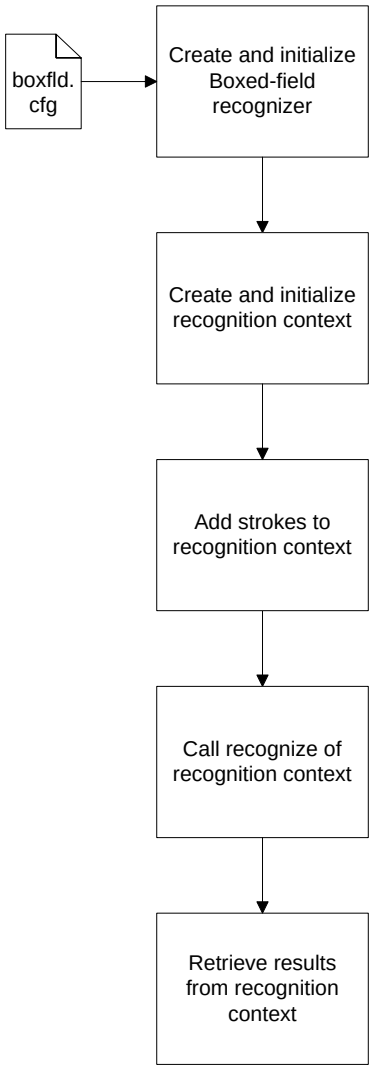
This method unloads all the model *data.of* of the shape recognizer Call initialize API to re-initialize the recognizer.

- **int LTKStrEncoding::shapeStrToUnicode(const string shapeRecProjectName, const vector<unsigned short>&shapeIDs, vector<unsigned short>& unicodeString)**

The mapping of shape ids to Unicode is performed in the *LTKStrEncoding* class in $LIPI_ROOT/src/utils. This static function shapeStrToUnicode maps the string of shape recognizer IDs to Unicode depending on the project name. An example mapping is present for Tamil characters. The user needs to modify LTKStrEncoding::shapeStrToUnicode(…) and add his own mapping for conversion to Unicode.

4.7.3 Flow diagrams

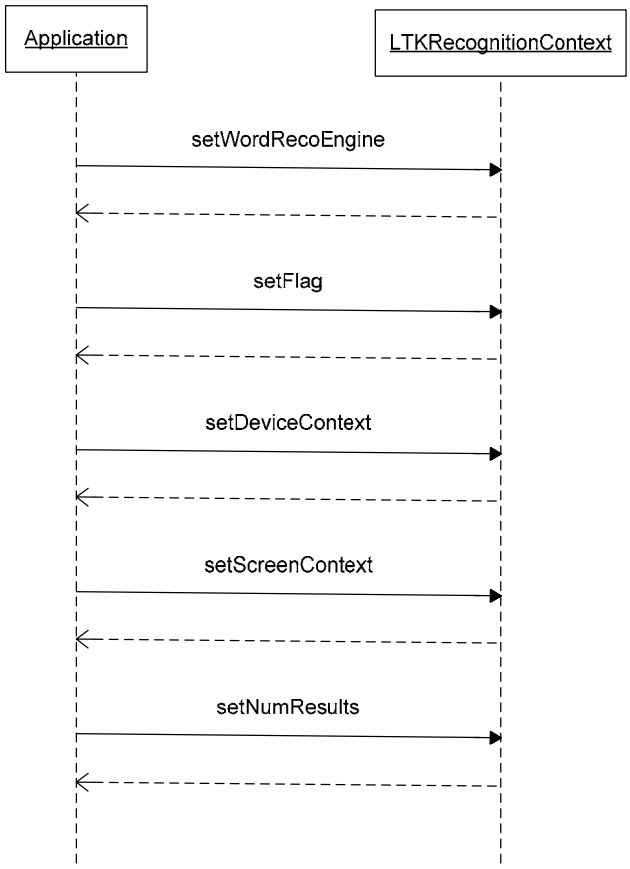
The general flow of a word recognition application will be as follows:



4.7.4 Sequence Diagrams

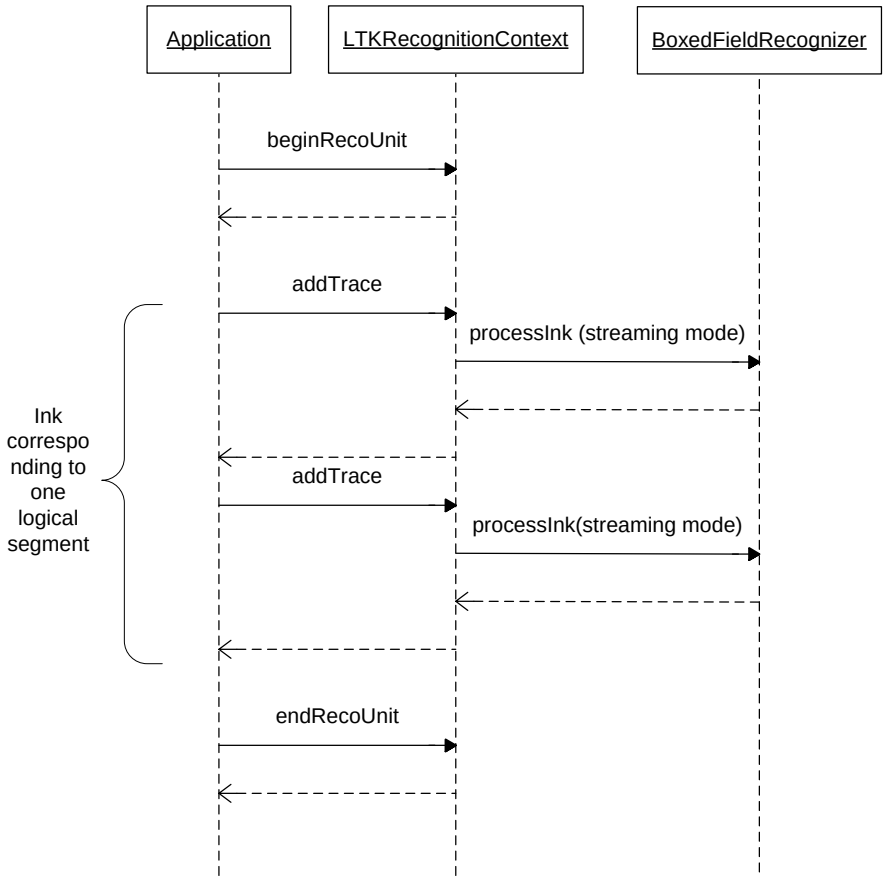
Set up the recognition context

This involves setting up the recognition context with proper device parameters, UI information etc.

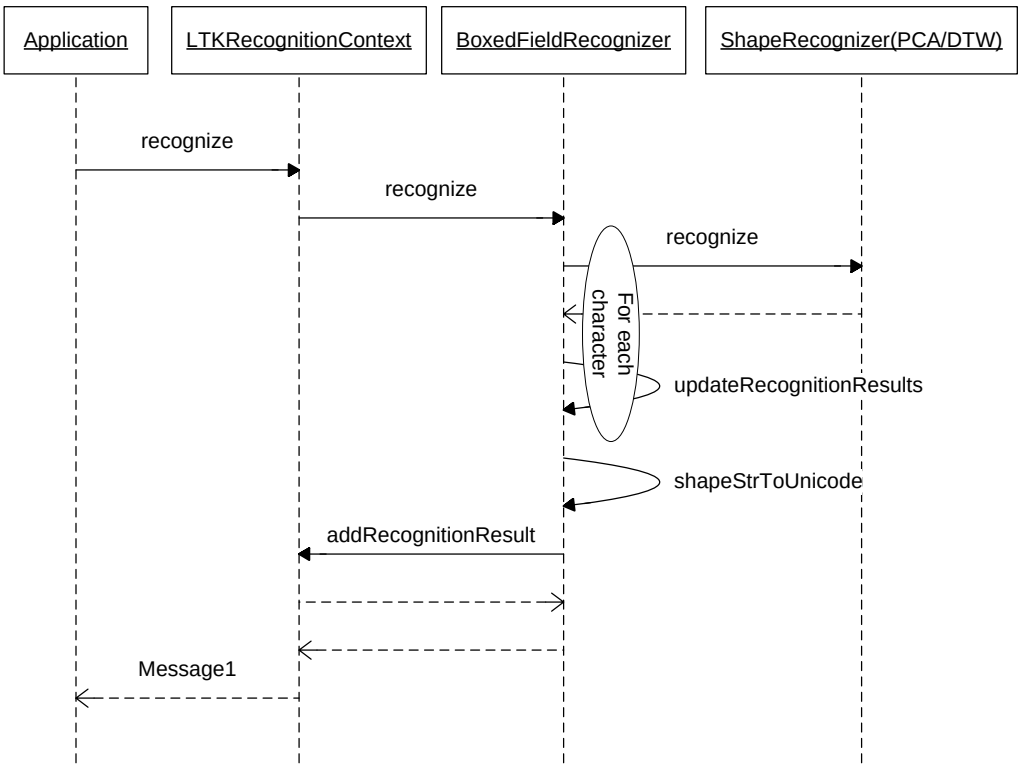


Adding ink to recognition context

Application adds Ink to be recognized to the recognition context. The application needs to call *beginRecoUnit* (*endRecoUnit*) before (after) adding the group of ink corresponding to one character.


Application
LTKRecognitionContext
BoxedFieldRecognizer
beginRecoUnit
addTrace
processInk (streaming mode)
Ink corresponding to one logical segment
addTrace
processInk(streaming mode)
endRecoUnit

Recognize


Application
LTKRecognitionContext
BoxedFieldRecognizer
ShapeRecognizer(PCA/DTW)
recognize
recognize
recognize
For each character
updateRecognitionResults
shapeStrToUnicode
addRecognitionResult
Message1

4.7.5 Configuration Attributes

The word recognizer requires project, profile and configuration to be defined for each project. The project files should be under the project root, which is a subdirectory of the $LIPI_ROOT/projects directory. For example consider a project for the recognition of English. The project root could be $LIPI_ROOT/projects/eng_alpha (PROJROOT), where *eng_alpha* is the logical name of the project. The config directory under the PROJROOT contains *project.cfg* file, which has project type (WORDREC) and a logical name. Each project can have one or more profiles, one each for different algorithms used for shape recognition or different configuration parameters used in the same algorithm. The different profiles are stored as subdirectories under PROJROOT/config. The settings will be loaded from the profile that is requested at the time of creation of the word recognizer. Each profile directory should contain *profile.cfg* file and the configuration files corresponding to the algorithm used for word recognition. In order to use the Boxed-field recognition algorithm for word recognition the *WordRecognizer* attribute in *profile.cfg* should be set to *boxfld* and the profile directory should have the *boxfld.cfg* file.

Filename	Path	Config file attributes (attribute = value)	Description
project.cfg	PROJROOT/config	ProjectType = WORDREC	This project is used to recognize words
		ProjectName= "English boxed-field recognizer"	This is the logical name of the project
profile.cfg	PROJROOT/config/<profile>	WordRecognizer = boxfld	The Boxed-field recognizer is used
		RequiredProjects = eng_char(default)	This project uses data from the project eng_char with default profile
boxfld.cfg	PROJROOT/config/<profile>	BoxedShapeProject = numerals BoxedShapeProfile = default	The Boxed-field recognizer use this project and profile configuration for shape recognition
		NumSymbols = 26	Size of character set for the shape recognition project
		MinShapeConfid = 0	Threshold for shape recognition result confidence
		NumShapeChoices = 3	Number of shape recognizer results requested

4.7.6 Source Directory Hierarchy

The BoxedFieldRecognizer code is under $LIPI_ROOT/src/reco/wordrec/boxfld. The directory $LIPI_ROOT/src/reco/wordrec/common contains the code for common classes such as LTKRecognitionContext, LTKWordRecoResult etc.

4.7.7 Build instructions

On Windows

- Make sure that the $LIPI_ROOT environment variable is set to lipitk directory
- Change directory to $LIPI_ROOT/src/reco
- Execute nmake /f Makefile.win boxfld (this builds all the dependent modules)

On Linux

- Make sure that the $LIPI_ROOT environment variable is set to lipitk directory
- Change directory to $LIPI_ROOT/src/reco directory
- Execute make –f Makefile.linux boxfld (this builds all the dependent modules)

4.7.8 Exported Functions

- createWordRecognizer
- deleteWordRecognizer
- getCurrentVersion
- startLogging
- stopLogging

4.7.9 Required Libraries

Static libraries

Libraries required ($LIPI_ROOT/src/lib) Windows/Linux	**Description**
utils.lib/libutils.a	Utilities to read/write UNIPEN ink, etc.
common.lib/libcommon.a	Common data structures to represent and process the ink
shapereccommon.lib/ libshapereccommon.a	Shape recognition specific data structures
wordrecocommon.lib/ libwordrecocommon.a	Word recognition specific data structures

DLL/SO

Libraries required ($LIPI_ROOT/lib) Windows/Linux	**Remarks**
boxfld.dll/libboxfld.so	Boxed-field recognizer implementation
lipiengine.dll/liblipiengine.so	Interface to the create/delete word recognizers
Shaperecognizer dll	For shape recognition

4.7.10 Support header files

Header file	Location	Remarks
LTKWordRecognizer.h	$LIPI_ROOT/src/include	Defines the word recognition interface
LTKshapeRecognizer.h	$LIPI_ROOT/src/include	Defines the shape recognizer interface
LTKMacros.h	$LIPI_ROOT/src/include	Defines global macros which are used across LipiTk
LTKInc.h	$LIPI_ROOT/src/include	Generic include file which includes all the standard include headers
LTKTypes.h	$LIPI_ROOT/src/include	Defines all the LipiTk specific data types
LTKTrace.h, LTKTraceGroup.h	$LIPI_ROOT/src/include	Defines ink data types that are used to store ink info
LTKErrorList.h		Define all the error macros used in LipiTk

4.7.11 Compile Flags

Linux: None

Windows: Use always “Multithreaded” runtime library option on release builds and “Debug Multithreaded” runtime library option on debug builds (MT and MTd)

5 LipiTk 1.0 errors and descriptions

To handle errors from various modules in LipiTk, all the errors are defined in a single place and all the modules return with that error code. At anytime the user can get the last error that occurred in a module by calling getLastError method, which is implemented by all the modules in LipiTk.

All the error codes are defined in $LIPI_ROOT/src/include/LTKErrorsList.h. All the strings corresponding to these error codes are defined in $LIPI_ROOT/src/util/lib/LTKErrors.cpp. To add or update any error code, refer to the above files and add or update as required.

Use the macro LTKReturnError to return any error from any function in any module, which is defined under $LIPI_ROOT/src/include/LTKInc.h

Error Code	Name	Definitions
100	EINK_FILE_OPEN	Unable to open ink file. Look at the Log file for more info
101	ECONFIG_FILE_OPEN	Unable to open configuration file. Look at the Log file for more info

102	EHEADER_INFO_FILE_OPEN	Unable to open model header information file
103	EMODEL_DATA_FILE_OPEN	Unable to open model data file. Look at the Log file for more info
104	ETRAINLIST_FILE_OPEN	Unable to open train list file. Look at the Log file for more info
105	EMODEL_DATA_FILE_FORMAT	Incompatible model data file. The header is not in the desired format. Look at the Log file for more info
106	EMODEL_DATA_FILE_CORRUPT	Model data file is corrupted. Look at the Log file for more info
107	ELOAD_SHAPEREC_DLL	Error while loading shape recognition module
108	ELOAD_WORDREC_DLL	Error while loading word recognition module
109	ELOAD_PREPROC_DLL	Error while loading preprocessing module
110	EDLL_FUNC_ADDRESS	Exported function not found in module. Look at the Log file for more info
111	ECREATE_SHAPEREC	Error while creating shape recognizer instance
112	ECREATE_WORDREC	Error while creating word recognizer instance
113	ECREATE_PREPROC	Error while creating preprocessor instance
114	ELIPI_ROOT_PATH_NOT_SET	Environment variable $LIPI_ROOT is not set
115	EINVALID_PROJECT_NAME	Invalid or no entry for project name
116	EINVALID_CONFIG_ENTRY	Invalid configuration entry in project.cfg file
117	ENO_SHAPE_RECOGNIZER	No shape recognizer specified in profile.cfg file
118	ENO_WORD_RECOGNIZER	No word recognizer specified in profile.cfg file
119	EINVALID_NUM_OF_TRACES	Invalid number of traces processed. Look at the Log file for more info
120	EINVALID_NUM_OF_SHAPES	Invalid value for number of shapes. Look at the Log file for more info
121	EINVALID_TRACE_DIMENTION	Invalid value for trace dimension. Look at the Log file for more info
122	EINVALID_NUMEIGENVECTOR	Invalid value for eigen vector. Look at the Log file for more info
123	EINVALID_FLOAT_SIZE	Invalid float size entry in model data File. Look at the Log file for more info
124	EINCOMPATIBLE_VERSION	Incompatible algorithm version. Look at the Log file for more info
125	EINVALID_PREPROC_SEQUENCE	Wrong preprocessor sequence entry in cfg file. Look at the Log file for more info
126	EINVALID_PROJECT_NAME	Invalid or no value specified for project name for recognizer

127	EINVALID_LOGICAL_NAME	Invalid or no value specified for logical name for recognizer
128	EINVALID_SEGMENT	Invalid segment, boxed-field recognizer requires character level segment info
129	EINVALID_REC_MODE	Unsupported recognizer mode
130	EUNSUPPORTED_STATISTICS	Unsupported or invalid statistics to be computed
131	EMAP_NOT_FOUND	No function implemented to convert to a Unicode string
132	EINVALID_SHAPEID	Invalid value for shape id
133	ENOMAPFOUND_LIPIENGINECFG	Cannot map the logical name, no entries in lipiengine.cfg
134	EINVALID_NUM_OF_POINTS	Number of points in the tracegroup is not normalized
135	EEMPTY_TRACE	Empty trace
136	EEMPTY_TRACE_GROUP	Empty TraceGroup
137	ECONFIG_FILE_RANGE	The config file variable is not within the correct range
138	EINITSHAPE_NONZERO	Initial shape id is not zero
139	EINVALID_LINE_LISTFILE	Invalid line in the listfile (train or test)
140	EINVALID_ORDER_LISTFILE	Invalid order of shape-ids in the list file (train)
141	ENUM_NNS	Invalid number of nearest neighbors specified
142	EINKFILE_EMPTY	Ink file name is empty
143	EINKFILE_CORRUPTED	Incorrect or corrupted Unipen ink file.

Logging error messages

Whenever the error is encountered in the application the error will be logged into the log file. For logging the error messages the user has to supply the log file name to the application.

6 Using LipiTk – A walk through

(On Windows by combining the results of existing shape recognizers PCA and DTW)

6.1 Writing a new shape recognizer module ABC

1 Installation

Follow the download instructions and unpack the sources.

(Refer User Manual Section 8)

2 Setup

A. Set environment variable LIPI_ROOT to the directory where the sources are unpacked.

B. Make a new profile under PROJROOT/config/abc.

C. Create PROJROOT/config/abc/project.cfg and add the following: ProjectType = SHAPEREC NumShapes=26

D. Create PROJROOT/config/abc/profile.cfg and add the following: ShapeRecMethod = abc

E. Copy pca.cfg from $LIPI_ROOT/src/reco/shaperec/pca/pca.cfg to PROJROOT/config/abc and copy dtw.cfg from $LIPI_ROOT/src/reco/shaperec/dtw/dtw.cfg to PROJROOT/config/abc

3 Creating the new shape recognizer module

A. Create a new directory under $LIPI_ROOT/src/reco/shaprec/abc.

B. Copy the contents of the folder $LIPI_ROOT/src/reco/shaperec/tst to $LIPI_ROOT/src/reco/shaprec/abc.

C. Replace the prefix EXP by ABC in all the files and filenames. So the name of new shape recognizer class would be ABCShapeRecognizer.

D. Modify the makefile and change the final executable/binary name to “abc” (abc.dll on Windows and libabc.so on Linux)

E. Add the code to “initialize” function of ABC

a. Load pca.dll and dtw.dll

b. Create an object of pca and dtw using createShapeRecognizer method in the above mentioned dlls

c. Invoke the initialize function of both PCA and DTW modules by just passing the values of ABC’s initialize function parameters

F. Modify the train function of ABC recognizer

a. Call train method of pca and dtw shape recognizers

G. Modify the loadModelData function of ABC recognizer

b. Call loadModelData method of pca and dtw shape recognizers

H. Modify the recognize function of ABC recognizer

a. Call recognize method of pca and dtw shape recognizers

b. Add code for combining results of PCA and DTW shape recognizers. (This will combine the set of results from pca and dtw and determine the final results)

c. Return the combined results

I. Tunable parameters required for the combination scheme should go into abc.cfg config file under the PROJROOT/config/abc

Tip: The contents of the abc.cfg can be read into a map using the utility function "ConfigFileReader::getMap(<config file name>)" in $LIPI_ROOT/src/util/lib/ConfigFileReader.cpp

4 Build

Open the makefile under $LIPI_ROOT/src/reco folder. Add a dependency for abc to pca and dtw. Run make to compile all the binaries.

5 Package for deployment

A. cd $LIPI_ROOT/package

B. Create package.cfg as follows:

```
[package]

Projects = english_alpha(default), english_alpha(special),english_alpha(abc)

Src = apps/samples/shaperectst

[export]

ENGLISH_ALPHA_COMBINED = English_alpha(abc)
```

C. $LIPI_ROOT/scripts/package.pl -pkg package.cfg -pkgname Eng_alpha_reco

Find the package created under $LIPI_ROOT/package directory.

6.2 Adding new preprocessing methods and the configuration

Assumptions

The shape recognizer for English alphabets using PCA module is installed and available.

PROJROOT = $LIPI_ROOT/projects/English_alpha

DATAROOT = PROJROOT/data (or) $LIPI_ROOT/data/English

The new preprocessing function which will be added is "newPreprocFunc".

1 Adding a new preprocessing function

A. Define the new method newPreprocFunc in $LIPI_ROOT/src/include/ LTKPreprocessorInterface.h.

Note: the function signature should follow the following convention:

```
virtual int newPreprocFunc (const LTKTraceGroup& inTraceGroup, LTKTraceGroup& outTraceGroup) = 0;
```

B. Add the implementation declaration in $LIPI_ROOT/src/include/ LTKPreprocessor.h as follows:

```
int newPreprocFunc (const LTKTraceGroup& inTraceGroup, LTKTraceGroup& outTraceGroup);
```

C. Implement the function and add the functionality in the source file under

$LIPI_ROOT/src/reco/shaperec/preprocessing/ LTKPreprocessor.cpp

2 Configuring the preprocessing sequence

The user can configure or change the preprocessing sequence by specifying the sequence in configuration file.

For example incase of the “English alpha” shape recognizer, the configuration file will be under $LIPI_ROOT/projects/English_alpha/config/default/pca.cfg

The contents of PCA.cfg will have an entry:

```
PreprocSequence={CommonPreProc::normalizeSize,PCA::resampleTraceGroup,CommonPreProc::normalizeSize}
```

Since the newly added function “newPreprocFunc” is available in common preprocessing module, we can configure this as follows:

```
PreprocSequence={CommonPreProc::normalizeSize,CommonPreProc::
newPreprocFunc, PCA::resampleTraceGroup,CommonPreProc::normalizeSize}
```

7 Appendix

This appendix provides information on how to use Doxygen for detailed source documentation.

7.1 Using doxygen to generate detailed source documentation

Doxygen is an on-line documentation browser (in HTML) from a set of documented source files. You can generate the documentation from the LipiTk source using this tool by following the steps given below.

This tool can be downloaded from the following link (source and binary) http://www.stack.nl/~dimitri/doxygen/download.html

Refer to http://www.stack.nl/~dimitri/doxygen/install.html for installation and configuration instructions.

Usage:

1 Generate configuration file as follows:

doxygen -g <configName>

2 Modify the following attributes

PROJECT_NAME = "LipiTk"

PROJECT_NUMBER = Version 1.0.0

OUTPUT_DIRECTORY = htmldoc/

USE_WINDOWS_ENCODING = YES

EXTRACT_ALL= YES

EXTRACT_PRIVATE= YES

MULTILINE_CPP_IS_BRIEF = YES

INPUT= .

RECURSIVE= YES

EXAMPLE_RECURSIVE= YES

SOURCE_BROWSER= YES

INLINE_SOURCES= YES

ALPHABETICAL_INDEX= YES

GENERATE_HTML= YES

HTML_OUTPUT= .

GENERATE_TREEVIEW= YES

HAVE_DOT= YES

TEMPLATE_RELATIONS= YES

CALL_GRAPH= YES

DOT_CLEANUP= YES

GENERATE_LATEX= YES

3 Generate documentation using an existing configuration file

```
doxygen <configName>
```

Refer the following link for downloading additional required graphs and diagram tools:

http://www.stack.nl/~dimitri/doxygen/diagrams.html

7.2 References

Deepu V. and Sriganesh M. (2004). Deepu V. and Sriganesh Madhvanath. Principal Component Analysis for Online Handwritten Character Recognition. In *Proc. 17th International Conference on Pattern Recognition(ICPR ´ 04)*, pages 327.330, Cambridge, UK, August 2004.

Niranjan J. (2004). Niranjan Joshi, G. Sita, A. G. Ramakrishnan and Sriganesh Madhvanath. Comparison of Elastic Matching Shape recognizers for Online Tamil Handwritten Character Recognition. In *Proc. Ninth International Workshop on Frontiers in Handwriting Recognition(IWFHR ´ 04)*, pages 444-449, Tokyo, Japan, October 2004

Raghavendra B. S. (2005). Raghavendra B.S., Narayanan C.K., Sita G., Ramakrishnan A.G. and Sriganesh Madhvanath. Prototype Learning Methods for Online Handwriting Recognition. In *Proc. Eighth International Conference on Document Analysis and Recognition (ICDAR 2005)*, Seoul, Korea, October 2005.

7.3 Glossary

Project	LipiTk 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
DTW	**D**ynamic **T**ime **W**arping – An algorithm for matching shapes; also the identifier of a shape recognition algorithm provided with LipiTk 1.0
PCA	**P**rincipal **C**omponent **A**nalysis – a form of statistical data analysis involving the extraction of Principal Components from the data; also the identifier of a shape recognition algorithm provided with LipiTk 1.0
DAT	**D**ata **A**nnotation **T**ool – A tool for annotation of handwriting data shipped with LipiTk 1.0
DCT	**D**ata **C**ollection **T**ool – A tool for collection of handwriting data shipped with LipiTk 1.0
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 compression 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.