# ASR Suite for Dummies

## Overview

This document is a software manual, describing the tools used to yoke the simulated hearing aid to a model of the patients impaired hearing (the dummy), and how the dummy can be subsequently connected to an automatic speech recogniser. The idea is that if an automatic speech recogniser is trained using feature vectors derived from a model of normal hearing, then the speech recogniser will perform optimally when tested with feature vectors derived from the same normal-hearing dummy. If the recogniser is tested using feature vectors derived from an impaired dummy, then one would expect the recognition score to drop significantly. The representation of the acoustic stimulus can be modified by a hearing aid algorithm before being presented to the impaired dummy. The potential of the hearing aid signal processing to alleviate some of the reduction in speech intelligibility caused by the impairment may then be assessed by comparing the aided and unaided recognition scores. This software framework should eventually enable the tailoring of hearing aid parameters to individuals in their absence.

The software suite exists in a “userPrograms” folder within the main Matlab model of the Auditory Periphery (MAP) folder. This tutorial document assumes that the reader is already familiar and comfortable with using MAP. The full documentation for MAP can be found at:

<http://www.essex.ac.uk/psychology/department/research/hearing_models.html>

## Files included in the suite

The evaluation of the hearing aid is done with the assistance of 3 main classes.

### cEssexAid.m

This is the file that contains the class definition of the Essex Aid wrapper. The Essex Aid is a novel hearing aid algorithm developed at the University of Essex. Once the user is familiar with the software framework, it should be relatively straightforward to swap out the hearing aid algorithm so that any hearing aid algorithm can be tested.

### cJob.m

This is the class definition that is the real workhorse of the operation. It provides many utility functions to assist the user in creating feature sets for use in training and testing the speech recogniser. This class also contains code for scheduling, so that many sounds can be processed in parallel on one or many machines should you have the resources available to do so.

### cHMM.m

This is a wrapper class for the Hidden Markov Tookkit (HTK) <http://htk.eng.cam.ac.uk/>. This class contains helper functions that provide the recogniser with the necessary information for training a HMM and scoring the results produced.

\*\*\*

Other functions within the parent directory include

### worker.m

This is an autonomous little function that takes a path to a folder containing a list of jobs as an input argument. A job is defined here as a list of wav files that need to each be converted into a feature vector. It searches the directory for jobs to do and works until all of the jobs are complete.

### Exp\_Tutorial\_X.m

Files prefixed with “Exp\_” are the files that users will edit most frequently. They are at the top of the function call stack and initiate all of the tasks required to run a recognition experiment.

### MAPwrap.m

This is a very simple wrapper for MAP, making it easy to call from the classes within the suite. MAP also uses a lot of global variables, so the wrapper also provides a barrier to prevent potential conflicts between variable names.

\*\*\*

There are also two folders in the parent directory

### /def

This contains the definition files required by HTK, such as grammar rules, dictionaries and hmm prototypes. Full descriptions of these files are beyond the scope of this document, but detailed information can be found here <http://htk.eng.cam.ac.uk/>

### /ASRfiles

This is a folder containing some additional utility functions used by the main classes.

\*\*\*

## Tutorial

One of the best ways to learn is by doing. The tutorial is split into two sections that represent a typical workflow. The first part of the tutorial shows the reader how to make a new HMM and then test it. The second part shows the reader how to use an existing HMM to test new features. The new data might have come from a different dummy and/or hearing aid processing.

Before any work with the recogniser can be accomplished, the speech material and recogniser software must be in place.

### Install HTK

HTK needs to be compiled for your platform and added to the path. The method for doing this will be different under different operating systems. See the following link for information on how to add programs to the path under Windows.

<http://lmgtfy.com/?q=windows+add+to+path>

Once the HTK binaries have been successfully added to the path, the individual tools should be available from Matlab. This can be tested by issuing the following command (the >> should not be typed):

>> !HVite -V

This command should then output some version information about HTK into the command window. If the command fails to return version information, then logging off and then back on again should solve the problem.

### Get appropriate speech material

The software provided is designed to work with the AURORA 2.0 TI digits corpus available here:

<http://www.elda.org/article52.html>

The clean training data should be in wav format and placed into one directory. The clean, digit-triplet test sound files should placed in a separate directory.

It is also possible to make a custom corpus so long as the following rules and file naming conventions are adhered to.

1. Speech material should be recorded in (or converted to) wav file format. Any sampling rate can be used, as the Matlab scripts will resample the speech files appropriately. The speech should be recorded in single channel format.
2. Training data should be recorded as strings of digits between 1 and approximately 7 digits per file. Test data files should contain 3 digits. Digits should include a fairly even mixture of “oh”, “one”, “two”, “three”, “four”, “five”, “six”, “eight”, “nine”. The bisyllabic digits “seven” and “zero” should not be used.
3. Speech files should be named like the following example, “FAC\_8O4A.wav”. The first character in the string refers to the gender of the talker. The next two characters are a unique identifier for the specific person doing the talking. These two characters should be followed by an underscore. The next characters are the string of numbers that are uttered in the sound file. These numbers are terminated with a capital “A” and the wav extension. IMPORTANT NOTE: If the file contains the utterance “oh”, the alphabetic character “O” should be used rather than the numerical character “0”.
4. The files in the training and testing corpora should be unique.

### Get appropriate noise material

Any noise material of suitable duration (substantially longer than the longest sound file containing speech) can but used. The tutorials here use the “factory1” noise sample from the freely available NOISEX database.

<http://spib.rice.edu/spib/select_noise.html>

### Show the cJob class where to find the sound files

The main job class (cJob.m) needs to know the location of the speech material. The speech material can be stored at any location on the user’s computer, but the following piece of code needs to be updated accordingly. If different computers on different platforms are used then the paths can be set accordingly. For example, I use a windows machine in the office, a mac at home, and a linux machine to run large jobs. All of these systems store the corpora in different locations.

if isunix

if ismac

lWAVpath = '~/ASR/reducedAURORA/TrainingData-Clean/';

rWAVpath = '~/ASR/reducedAURORA/TripletTestData/';

obj.noiseFolder = '~/ASR/noises';

else

lWAVpath = '/scratch/nnn/corpora/AURORA digits (wav)/TrainingData-Clean/';

rWAVpath = '/scratch/nnn/corpora/AURORA digits (wav)/TripletTestData/';

obj.noiseFolder = '/scratch/nrclark/corpora/noises';

end

else

lWAVpath = 'C:\corpora\AURORA digits (wav)\TrainingData-Clean';

rWAVpath = 'C:\corpora\AURORA digits (wav)\TripletTestData';

obj.noiseFolder = 'C:\corpora\noises';

end

### Setting an output folder

The speech recognition experiments involve generating a large number of files that need to be stored somewhere. It is possible to change the output folder on an experiment by experiment basis, but the user may wish to have a top level directory in which all ASR data is stored. To do this, find and amend the following code segment.

else

if isunix

if ismac

obj.opFolder = '~/ASR/exps/\_foo';

else

obj.opFolder = '/scratch/nrclark/exps/\_foo';

end

else

obj.opFolder = 'D:\exps\\_foo';

end

end

### Training and testing a recogniser

If everything has been set correctly, it should now be possible to run Exp\_Tutorial\_1 without generating errors. This function can be used as a template to run all kinds of recognition experiments. The following text breaks down each part of the function, describing what happens in each block of code.

### Parallelism

The first line of the file declares the function and states that it takes one input argument, isMasterNode.

function Exp\_Tutorial\_1(isMasterNode)

Speech recognition experiments are very processor intensive, so the software suite has been carefully designed to run in parallel across many instances of Matlab. This allows results to be generated in a fraction of the time of a serial process if enough computing power is available. The simple scheduling software was written in house, and so it does not require any special Matlab licenses for clustering. The variable in the function definition above, isMasterNode, is a Boolean type that lets the current Matlab instance know if it is the master node.

The master node is the most important node, responsible for generating all of the job information and interfacing with HTK. Because the mater node has the responsibility of generating the job lists and storing them, the variable isMasterNode must be set to true when the experiment function is first called. While the experiment is running in one Matlab instance, it is then possible to share the workload by running the same command in another Matlab instance with isMasterNode set to false. In theory, there is no upper limit to the number of helper nodes.

Under windows, the simplest way to do this is to open a few Matlab instances and then set the experiment running with one master node. The other Matlab windows can be used as helper nodes. The demo experiments are designed to run entirely in the command window. No other aspects of the Matlab integrated development environment are needed. Therefore, small performance gains may be attained by running Matlab from the command prompt with the flag “-nodesktop”. This will run Matlab with the command window only. From there, just change directory to the working directory and invoke the experiment function from the command line. Trial and error must be used to find the optimum number of Matlab instances for a specific machine.

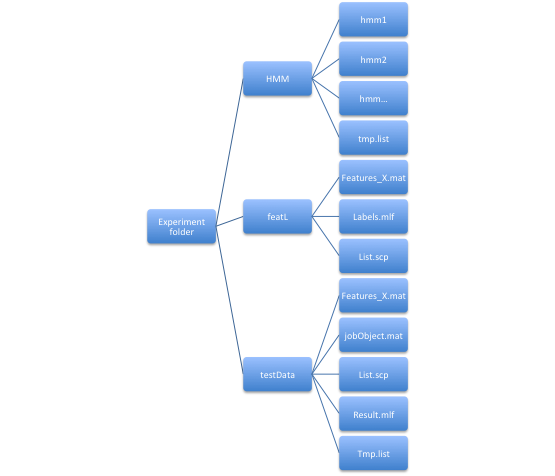
This setup is particularly powerful in a \*nix environment when used in conjunction with GNU screen <http://www.gnu.org/s/screen/>. From each virtual terminal, a separate, low-resource instance of Matlab can be launched with the command flag “-nodisplay”. Each instance of Matlab can then spread the job load as in the Windows example. If a number of machines are available with access to the same network attached storage, then the nodes can be spread across the different machines, with multiple nodes running on each machine. Of course, these nodes can be running a mixture of any operating system supported by Matlab.

### Experiment data directory structure

The speech recognition experiment process involves creating many small configuration and data files. The hierarchy chart below shows the directory structure generated for a typical experiment. All files related to a particular HMM are stored within a top-level experiment folder. The experiment folder contains at least two other folders. These are the HMM directory and the featL directory. There can also be any number of folders containing test features.

The HMM directory contains numbered hmm subdirectories that each contain a different version of the HMM after the sequential parameter re-estimations that occur during the training stage. The hmm36 folder contains the most refined HMM that is used to evaluate the experimental data. The HMM directory also contains a file called tmp.list. This is just a data file used by HTK to locate the training material.

The featL directory contains the training features. The “L” is used to signify learning as opposed to “R” for recognition. This is to avoid the obvious confusion that might arise if single character representations were used for training and testing features. The “L” and “R” abbreviations are also used throughout the software scripts. The saved training feature files correspond to individual wav files and thus have the same file names but with a .mat instead of a .wav extension. The featL directory also contains a file called labels.mlf. This text file explains the digits contained within each feature file in an HTK readable format. The list.scp file is a text document with the names of all of the feature files in the folder that HTK should use for training the recogniser.



The last file type in the training feature directory is the jobObject.mat file. This is a Matlab readable data file containing a copy of the instance of the cHMM class associated with the data set. This data object contains all of the information about the experiment, including MAP parameters, HMM paramaters, and other instructions on how to generate features. Each processing node loads this object when initialised, so that it knows how to process the wav files appropriately. Each note processes a randomly assigned set of wav files and then updates this data object so that other nodes know which files are not yet processed, which files are currently cued for processing, and which files have been processed.

There can be an unlimited number of test data folders within the main experiment folder (explain example of 5 different SNRs)

The next bit of code sorts out some directory structure housekeeping.

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Set up the basic folders

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

expName = 'Tutorial';

dataFolderPrefix = 'hello\_world';

if isunix

expFolderPrefix = '/scratch/nrclark/exps/';

else

expFolderPrefix = 'D:\Exps';

end

% expFolderPrefix = pwd;

expFolder = fullfile(expFolderPrefix,expName);

hmmFolder = fullfile(expFolder,'hmm');

The experiment name is defined by the variable expName. This is the name of the top-level folder that contains training features, testing features, and the hmm. The dataFolderPrefix variable is a character string that precedes each folder containing test features. Typically, the user will create a HMM and then test that hmm using many different test configurations on different occasions as inspiration is drawn from previus sets of results