

REVERB Challenge

Baseline Training, Recognition and Scoring Scripts

Volker Leutnant, Marc Puels, Reinhold Haeb-Umbach

1. Purpose

This collection provides scripts for the baseline recognition system of the REVERB challenge. It contains scripts to train an HMM-based speech recognizer on the data of the WSJCAM0 database [1], retrain the acoustic model using the challenge multi-condition training data (REVERB_WSJCAM0_tr), and obtain recognition results on the challenge test sets which include SimData (i.e. REVERB_WSJCAM0) and RealData (i.e. MC_WSJ_AV [2]). The provided scripts and their usage will be introduced in detail in Sec. 3 and Sec. 4.

Note that to run the scripts you need to obtain the REVERB challenge data sets. See the site below for details how to obtain the data sets, <http://reverb2014.dereverberation.com/download.html>

The list of files included into this collection is given in Appendix B.

2. ASR tool Installation

2.1. Required external software/data

The provided Linux-scripts are based on the hidden MARKOV model toolkit (HTK) [3] and the NIST SPeech HEader REsources Package (SPHERE). While the former is required throughout almost all scripts, the latter is only required to handle the compressed audio files of the WSJCAM0 database, e.g., during feature extraction.

Further, the BEEP dictionary (version 1.0; `beep`) and the CMU dictionary (version 0.7a; `CMU`) are required.

It is recommended to use the provided scripts `installTools` and `installDicts` to get and install the current versions of the software and the pronunciation dictionaries (see Sec. 4 for details on these and all other provided tools).

The scripts have been tested with HTK v3.4.1, NIST SPHERE v2.6a, BEEP v1.0 and CMU dictionary v0.7a.

The installation procedure is as follows.

1. Run `installTools` script to download and install HTK, the HTK samples and the NIST SPHERE software. Note that the download of the HTK software and samples requires prior registration at <http://htk.eng.cam.ac.uk/register.shtml>, upon which username and password are provided.

If an already existing HTK and NIST SPHERE installation is intended to be used, `installTools` does not have to be called. However, make sure that only the desired installations bin directory is in your `${PATH}` variable! Edit the `LOCAL_CONFIG` accordingly. Further note that a patch is provided with this collection (*tools/sphere.patch.mkstemp*) that allows for parallel execution of the NIST SPHERE tools.

2. Run `installDicts` script to download and install the BEEP dictionary and the CMU pronunciation dictionary.

If the two dictionaries are already available on your system, edit the path to the dictionaries in the `LOCAL_CONFIG` accordingly.

Note that in case of connection problems while using `installTools` and `installDicts`, please check that the HTTP and FTP settings are well configured on your system.

2.2. Databases

The ASR scripts assume that you have downloaded the following from LDC (see <http://reverb2014.dereverberation.com/data.html> for details about the data):

- `wsjcam0` (clean speech data)
- `WSJ0_LangMod_REVERB` (language model for WSJ)
- `REVERB_WSJCAM0_dt` (development set of SimData)
- `MC_WSJ_AV_Dev` (development set of RealData)

To perform multi-condition retraining, you need the following data base

- `REVERB_WSJCAM0_tr` (REVERB challenge multi-condition training data)
This database can be generated using the tool available at
<http://reverb2014.dereverberation.com/download.html>

In addition the two following evaluation sets will be available on November 5th 2013.

- `REVERB_WSJCAM0_et` (evaluation set of SimData)
- `MC_WSJ_AV_Eval` (evaluation set of RealData)

Although special focus has been laid on making the provided scripts and tools independent of particular directory structures, the current version has been tested only on the directory structure summarized in Sec. C.1 and Sec. C.2.

3. Usage

To get things started, the following actions have to be taken:

1. If not already done, install HTK v3.4.1, NIST SPHERE v2.6a BEEP v1.0 and CMU dictionary v0.7a, and obtain the needed databases, as described in section 2.
2. Copy the file LOCAL_CONFIG.template to LOCAL_CONFIG and open it in your favorite text editor.
3. Now, go through the variables in the first part of LOCAL_CONFIG and modify them according to your system.

In particular, you should modify the variables that specify the paths to the databases (i.e. replace `${HOME}` with the path where you saved the databases). These variables are referred as (MANDATORY) in LOCAL_CONFIG. Do not modify anything below the line that says ("NO CHANGE BELOW THIS LINE REQUIRED").

NOTE: do not modify the LOCAL_CONFIG while the training/testing is running, since all scripts execute the LOCAL_CONFIG prior to any other processing!

4. The lib_basic/config directory keeps all HTK config files used for feature extraction and training. Modify these files according to your needs (usually only the HCopy configuration config.hcopy_common needs to be modified, since it keeps the parameters for the feature extraction). (This step is optional if you use the default configuration).

5. Now run WSJBuildAndTest. It will successively call scripts that

- prepare data, including,
 - analyze the databases and prepare required files to create files lists and reference transcriptions,
 - create file lists and reference transcriptions,
- train models, including,
 - extract the features for training sets of the WSJCAM0 database and for multi-condition training data,
 - train mono- and triphone HMMs on the WSJCAM0 training data,
 - retrain the HMMs using multi-condition training data,

Note that by default, the script uses already trained acoustic models and therefore the training stage is commented out. Modify the script if you would like to train the acoustic models.

- recognize and summarize results
 - extract the features for development and evaluation sets,
 - perform recognition with or without batch adaptation on the development and evaluation test sets of the WSJCAM0 database,
 - score and summarize the obtained results using the reference master label files

6. Upon success, the recognition results (transcriptions) and the evaluation results (insertions, deletions, substitutions,...) can be found in the results folders specified in `LOCAL_CONFIG`, i.e. by default *REVERBWSJCAM0/results* (for SimData) and *MCWSJAV/results* (for RealData).

Note that if you would like to retrain your system and recognize using speech data that you have processed, please modify the databases paths in `LOCAL_CONFIG` to the directory where your processed speech is located. If you stuck to the original directory structure the scripts should run without any further modification.

4. Tool description

The following paragraphs shortly describe the main scripts used to extract features, train mono- and triphone HMMs on the WSJCAM0 database and score the different test sets of the WSJCAM0 and MCWSJAV database. All scripts except `LOCAL_CONFIG.template`, `installTools`, `installDicts` and `WSJBuildAndTest` can be found in `scripts/` directory.

LOCAL_CONFIG.template The file `LOCAL_CONFIG.template` keeps all relevant information about the paths to be used during training, recognition and scoring. It is called once at the beginning of each script to make sure that all share the same variables.

The first thing to do is to copy `LOCAL_CONFIG.template` to `LOCAL_CONFIG` and modify the latter to suit your needs!

installTools The `installTools` script downloads the HTK software and the NIST SPHERE software from the provided URLs and installs them to

`"tools/HTK/htk/bin"`

and

`"tools/SPHERE/nist/bin"`.

Note that the download of the HTK software and samples requires prior registration at <http://htk.eng.cam.ac.uk/register.shtml>, upon which username and password are provided.

If an already existing HTK and NIST SPHERE installation is intended to be used, `installTools` does not have to be called. However, make sure that only the desired installations bin directory is in your `${PATH}` variable! Edit the `LOCAL_CONFIG` accordingly.

installDicts The `installDicts` script downloads the current version of the BEEP dictionary (version 1.0) and the CMU dictionary (version 0.7a) to `"lib_basic/dicts"`. Both are required, since the BEEP dictionary does not contain transcriptions for all words in the official training set of the WSJCAM0 database and the 5k word list used for recognition, respectively. Hence, the missing transcriptions are taken from the CMU dictionary. **This script should be called prior to any of the following scripts, unless the two dictionaries are already available on your system. Edit the path to the dictionaries in the `LOCAL_CONFIG` accordingly.**

WSJBuildAndTest The WSJBuildAndTest script is the main script to extract the features of the WSJCAM0 database and train speaker independent mono- and triphone HMMs. It calls the provided scripts in the required order and also runs recognition and scoring of the WSJCAM0 and MCWSJAV data.

If **LOCAL_CONFIG** has been modified to suit your needs, this script does all the work for you! Database preparation, feature extraction, training and recognition + scoring by calling the following scripts! Note that by default, all called scripts use already trained acoustic models and therefore the training stage is commented out. Modify the script if you would like to train the acoustic models.

wsjcam0_create_et_dt_dot_files The file `wsjcam0_create_et_dt_dot_files` creates the transcription files (.dot file) for the two evaluation sets `si_et_1` and `si_et_2` and the two development sets `si_dt5a` and `si_dt5b` of the WSJCAM0 database. These .dot files will be used to create the file list(s).

wsjcam0_create_audio_file_lists The `wsjcam0_create_audio_file_lists` script creates the list of audio files for the WSJCAM0 database based on the .dot files provided with the database or created by `wsjcam0_create_et_dt_dot_files`. The following sets are supported: `si_tr` (training), `si_dt5a`, `si_dt5b` (development sets a and b) and `si_et_1`, `si_et_2` (evaluation sets 1 and 2).

The created file lists will carry the prefix "audio" and can be found in "`{WSJLIB}/flists`".

wsjcam0_prepare_transcriptions The `wsjcam0_prepare_transcriptions` script prepares the transcription of the training, development and evaluation data in an MLF as required by HTK. It thereby uses the .dot files provided or created by `wsjcam0_create_et_dt_dot_files` and the list of audio files created by "`wsjcam0_create_audio_file_lists`".

The created MLFs can be found in "`{WSJLIB}/wlibs`".

wsjcam0_code_data The `wsjcam0_code_data` script extracts the features according to the configurations `{CONFIG_HCOPY_common}` (common feature extraction parameters) and `{CONFIG_HCOPY_WSJACM0}` (parameters related to the audio file format used in the WSJACM0 database) specified in the **LOCAL_CONFIG** file and creates the corresponding file list of feature vector files. It takes the audio file lists created by `wsjcam0_create_audio_file_lists` as input.

As the audio file lists, the feature file lists can be found in "`{WSJLIB}/flists`", however, are not prefixed by "audio".

wsj_prepare_monophone_dictionary The `wsj_prepare_monophone_dictionary` script prepares the monophone dictionary. Since not all words in the considered WSJCAM0 and MCWSJAV data are part of the BEEP dictionary, the final dictionary is composed of the BEEP dictionary (British English pronunciation) and the CMU dictionary (American English pronunciation). Note: though in a later stage, only the 5k vocabulary will be used for recognition, all words in the training data have to be specified in the dictionary. For that reason, the MLF for the `si_tr` set created by `wsjcam0_prepare_transcriptions` will be used to create a list of words used during training.

wsj_prepare_triphone_dictionary The `wsj_prepare_triphone_dictionary` script prepares the tri-phone dictionary including SIL (silence) and SP (short-pause). Further the monophone dictionary with SIL and SP is created.

wsj_train_monophones The `wsj_train_monophones` script uses the file list of training feature vector files and the corresponding MLF to train monophone HMMs. The structure of the HMM is thereby created based on the settings in the `LOCAL_CONFIG` file.

Upon completion, the final monophone HMM can be found in
`"${WSJCAM0HMMS}/W1/hmm4/MMF"`.

wsj_train_triphones The `wsj_train_triphones` script uses the file list of training feature vector files, the corresponding MLF and the monophone HMMs trained by the `wsj_train_monophones` script to train tied-state triphone HMMs.

Upon completion, the final "clean-condition" triphone HMM can be found in
`"${WSJCAM0HMMS}/W2/hmm104/MMF"`.

wsj_prepare_language_models The `wsj_prepare_language_models` script prepares the compact backoff bigram language model for the 5k word task considered. It's essentially the one provided with the WSJ database [4].

reverbwsjcam0_code_data_training The `reverbwsjcam0_code_data_training` script extracts features from the artificially distorted training data (to be created from the WSJCAM0 training data using the scripts provided at <http://reverb2014.dereverberation.com/download.html>) specified in the "taskFiles/1ch/SimData_tr_for_1ch_A" task file list. Note that it only contains file names relative to the REVERB_WSJCAM0_tr database directory.

Both audio and feature file lists are stored in `"${WSJLIB}/flists/reverbWSJcam0"`. The first with and the latter without the prefix "audio".

reverbwsjcam0_retrain The `reverbwsjcam0_retrain` script takes the tied-state triphone HMMs provided by `wsj_train_triphones`, i.e., `"${WSJCAM0HMMS}/W2/hmm104/MMF"`, and uses the features of the artificially distorted training data extracted by `reverbwsjcam0_code_data_training` to train "multi-condition" HMMs.

Upon completion, the final "multi-condition" triphone HMM can be found in
`"${REVERBWSJHMMS}/W_reverb/hmm124/MMF"`.

reverbwsjcam0_code_data_testing The `reverbwsjcam0_code_data_testing` script extracts feature from the artificially distorted evaluation test (et) and development test (dt) data specified in the "taskFiles/1ch/SimData_[et|dt]_for_[cln|1ch_near|1ch_far]_[A]" file lists.

Note: the file lists again only contain file names relative to the REVERB_WSJCAM0_dt / REVERB_WSJCAM0_et database directories.

Both audio and feature file lists are stored in `"${WSJLIB}/flists/reverbWSJcam0"`. The first again with and the latter without the prefix "audio".

reverbwsjcam0_recognize[_multicond] The `reverbwsjcam0_recognize[_multicond]` script performs recognition and evaluation on all of the "RealData_[et|dt]_for_[cln|1ch_near|1ch_far]_[_A]" tasks using the feature vectors created by `reverbwsjcam0_code_data_testing`.

The results will be placed in either "\${REVERBWSJRESULTS}" for recognition using the "clean-condition" HMMs or in "\${REVERBWSJRESULTS_MC}" for recognition using the "multi-condition" HMMs.

reverbwsjcam0_recognize_cmllr The `reverbwsjcam0_recognize_cmllr` script performs unsupervised CMLLR model adaptation, recognition and evaluation on the task files created from "taskFiles/SimData_[et|dt]_for_1ch_[near|far]_room1_A". Unsupervised means that the *reference transcription* is obtained from a prior recognition stage performed with `reverbwsjcam0_recognize[_multicond]`.

The processing steps are: recognition → base transform estimation → estimation of transforms for regression classes based on base transform → recognition using the estimated model transformation.

Note: the unsupervised CMLLR model adaptation can either be applied using the "clean-condition" HMMs or the "multi-condition" HMMs. You need to modify the script to use "clean-condition" HMMs. The results will be placed in "\${REVERBWSJRESULTS}_cmllr" for recognition using the "clean-condition" HMMs and in "\${REVERBWSJRESULTS_MC}_cmllr" for recognition using the "multi-condition" HMMs.

mcwsjav_prepare_transcriptions The `mcwsjav_prepare_transcriptions` script prepares the transcription of the development and evaluation data in an MLF as required by HTK. **NOTE: Please make sure that you run this script again once the evaluation set ("Eval") and the corresponding task files are provided to you. Otherwise, the MLF used for the evaluation of the results will not contain transcriptions for the "Eval" utterances (filtering only the required utterance speeds up evaluation by HResults!)**

The created MLF can be found in "\${WSJLIB}/wlibs/mcwsjav/MC_WSJ_AV.mlf".

mcwsjav_code_data The `mcwsjav_code_data` script extracts the features according to the configurations \${CONFIG_HCOPY_common} (common feature extraction parameters) and \${CONFIG_HCOPY_MCWSJAV} (parameters related to the audio file format used in the MCWSJAV database) specified in the LOCAL_CONFIG file and creates the corresponding file list of feature vector files. It takes the audio files specified in the task files "taskFiles/RealData_[et|dt]_for_1ch_[near|far]_room1_A" as input. Note that the file lists again only contain file names relative to the MCWSJAV database directory.

Both audio and feature file lists are stored in "\${WSJLIB}/flists/mcwsjav". The first again with and the latter without the prefix "audio".

mcwsjav_recognize[_multicond] The `mcwsjav_recognize[_multicond]` script performs recognition on the features extracted by `mcwsjav_code_data`, i.e., considers the tasks "RealData_[et|dt]_for_1ch_[near|far]_room1_A".

The results will be placed in "\${MCWSJAVRESULTS}" for recognition using the "clean-condition" HMMs and in "\${MCWSJAVRESULTS_MC}" for recognition using the "multi-condition" HMMs.

mcwsjav_recognize_cmllr The mcwsjav_recognize_cmllr script performs unsupervised CMLLR model adaptation, recognition and evaluation on the task files created from "taskFiles/RealData_[et|dt]_for_1ch_[near|far]_room1_A". Unsupervised means that the *reference transcription* is obtained from a prior recognition stage performed with mcwsjav_recognize[_multicond].

The processing steps are: recognition → base transform estimation → estimation of transforms for regression classes based on base transform → recognition using the estimated model transformation.

Note: the unsupervised CMLLR model adaptation can either be applied using the "clean-condition" HMMs or the "multi-condition" HMMs. You need to modify the script to use "clean-condition" HMMs. The results will be placed in "\${MCWSJAVRESULTS}_cmllr" for recognition using the "clean-condition" HMMs and in "\${MCWSJAVRESULTS_MC}_cmllr" for recognition using the "multi-condition" HMMs.

summarize_results The summarize_results script creates the baseline result table containing WER for the different conditions and calculates average WER. It uses the print_results script provided in tools/bash/ to extract results.

Reference results created with the provided scripts can be found in the Sec. A.1 and Sec. A.2 of the appendix of this document for a recognition with "clean-condition" HMMs and "multi-condition" HMMs, respectively.

A. Reference results

The following results have been obtained with the provided configuration (MFCC_0_D_A_Z). Training has been carried out on the clean data of the WSJCAM0 database recorded by the primary microphone. For multi-condition training, we used the REVERB_WSJCAM0_tr data.

Note that if you carry out training by yourselves, results may slightly vary (e.g., due to a different number of processes (\$NBPROC) specified in your LOCAL_CONFIG).

A.1. Reference results for Clean condition

```
#####
Summarizing results for clean-condition HMM usage!
#####
```

```
# SimData_dt (w/o CMLLR)
```

Room 1	Room 2	Room 3	
cln	cln	cln	Avg.
10.50	11.51	10.81	10.93

```
# SimData_dt (w/ CMLLR)
```

Room 1	Room 2	Room 3	
cln	cln	cln	Avg.
10.40	10.97	10.31	10.55

```
# SimData_dt (w/o CMLLR)
```

Room 1_A	Room 2_A	Room 3_A	
1ch_near	1ch_far	1ch_near	1ch_far
15.29	25.29	43.90	85.80

```
# SimData_dt (w/ CMLLR)
```

Room 1_A	Room 2_A	Room 3_A	
1ch_near	1ch_far	1ch_near	1ch_far
12.93	17.72	24.11	72.57

```
# RealData_dt (w/o CMLLR)
```

Room 1_A	
1ch_near	1ch_far
88.71	88.31

```
# RealData_dt (w/ CMLLR)
```

Room 1_A	
1ch_near	1ch_far
83.16	84.48

A.2. Reference results for Multi condition

```
#####
Summarizing results for multi-condition HMM usage!
#####
```

```
# SimData_dt (w/o CMLLR)
```

Room 1	Room 2	Room 3	
cln	cln	cln	Avg.
25.79	28.44	27.32	27.18

```
# SimData_dt (w/ CMLLR)
```

Room 1	Room 2	Room 3	
cln	cln	cln	Avg.
13.25	14.79	14.09	14.04

```
# SimData_dt (w/o CMLLR)
```

Room 1_A	Room 2_A	Room 3_A	
1ch_near	1ch_far	1ch_near	1ch_far
15.49	18.90	23.51	42.40

```
# SimData_dt (w/ CMLLR)
```

Room 1_A	Room 2_A	Room 3_A	
1ch_near	1ch_far	1ch_near	1ch_far
13.27	17.08	20.80	36.83

```
# RealData_dt (w/o CMLLR)
```

Room 1_A	
1ch_near	1ch_far
52.96	51.61

```
# RealData_dt (w/ CMLLR)
```

Room 1_A	
1ch_near	1ch_far
47.91	46.55

B. Content

The following files are provided by this collection,

```
.
|-- LOCAL_CONFIG.template
|-- README.txt
|-- WSJBuildAndTest
|-- doc
|   '-- ReverbDoc.pdf
|-- installDicts
|-- installTools
|-- lib_basic
|   |-- configs
|   |   |-- config.align
|   |   |-- config.build
|   |   |-- config.hcopy_MCWSJAV
|   |   |-- config.hcopy_WSJCAMO
|   |   |-- config.hcopy_common
|   |   '-- config.herest
|   '-- wsjquests.hed
|-- models
|   |-- clean_cond
|   |   '-- HMM
|   |       '-- MFCC_0_D_A_Z_CEPLIFTER_1
|   |           '-- W2
|   |               |-- hmm104
|   |               |   '-- MMF
|   |               '-- winttree.list
|   '-- multi_cond
|       '-- HMM
|           '-- MFCC_0_D_A_Z_CEPLIFTER_1
|               '-- W_reverb
|                   |-- hmm124
|                   |   '-- MMF
|                   '-- winttree.list
|-- printlib
|-- scripts
|   |-- mcwsjav_code_data
|   |-- mcwsjav_prepare_transcriptions
|   |-- mcwsjav_recognize
|   |-- mcwsjav_recognize_cmllr
|   |-- mcwsjav_recognize_multicond
|   |-- prepare_data
|   |-- recognize_and_evaluate
|   |-- reverbwsjcam0_code_data_testing
|   |-- reverbwsjcam0_code_data_training
|   |-- reverbwsjcam0_recognize
|   |-- reverbwsjcam0_recognize_cmllr
|   |-- reverbwsjcam0_recognize_multicond
|   |-- reverbwsjcam0_retrain
|   |-- summarize_results
|   |-- train_models
|   |-- wsj_prepare_language_models
|   |-- wsj_prepare_monophone_dictionary
|   '-- wsj_prepare_triphone_dictionary
```

```

|   |-- wsj_train_monophones
|   |-- wsj_train_triphones
|   |-- wsjcam0_code_data
|   |-- wsjcam0_create_audio_file_lists
|   |-- wsjcam0_create_et_dt_dot_files
|   '-- wsjcam0_prepare_transcriptions
|-- taskFiles
|   |-- 1ch
|   |   |-- RealData_dt_for_1ch_far_room1_A
|   |   |-- ...
|   |   '-- SimData_tr_for_1ch_A
|   |-- 2ch
|   |   |-- RealData_dt_for_2ch_far_room1_A
|   |   |-- ...
|   |   '-- SimData_tr_for_2ch_B
|   |-- 8ch
|   |   |-- RealData_dt_for_8ch_far_room1_A
|   |   |-- ...
|   |   '-- SimData_tr_for_8ch_H
|   '-- README.txt
'-- tools
    |-- bash
    |   |-- parallelHTK
    |   '-- print_results
    |-- perl
    |   |-- dot2mlf
    |   |-- get_sentnb
    |   |-- getwer
    |   '-- mapsym
    '-- sphere.patch.mkstemp

```

C. Directory tree for databases

C.1. Directory tree for WSJACM0 database

NOTE: Although the provided version has the data for si_et_1 under primary_microphone and the ones for si_et_2 under secondary_microphone, both si_et_1 and si_et_2 contain data of both microphones (files *.wv1 for the primary microphone and files *.wv2 for the secondary microphone), i.e., the "1" and "2" just indicate the set, not the microphone used!

```
.
|-- data
|   |-- primary_microphone
|   |   |-- doc
|   |   |-- etc
|   |   |-- si_dt
|   |   |   |-- c31
|   |   |   |-- ...
|   |   |   '-- c49
|   |   |-- si_et_1
|   |   |   |-- c30
|   |   |   |-- ...
|   |   |   '-- c4a
|   |   '-- si_tr
|   |       |-- c02
|   |       |-- ...
|   |       '-- c21
|   '-- secondary_microphone
|       |-- si_dt
|       |-- si_et_2
|       '-- si_tr
'-- docs

264 directories
```

C.2. Directory tree for MCWSJAV database (development set)

```

.
|-- audio
|   '-- stat
|       |-- T10
|           |-- array1
|           |-- array2
|           |-- headset1
|           '-- lapel1
|       |-- T6
|           |-- array1
|           |-- array2
|           |-- headset1
|           '-- lapel1
|       |-- T7
|           |-- array1
|           |-- array2
|           |-- headset1
|           '-- lapel1
|       |-- T8
|           |-- array1
|           |-- array2
|           |-- headset1
|           '-- lapel1
|       '-- T9
|           |-- array1
|           |-- array2
|           |-- headset1
|           '-- lapel1
|-- etc
|   '-- sentencelocation
'-- mlf
    '-- WSJ.mlf

```

30 directories

C.3. Directory tree for REVERB_WSJCAM0 database (development set)

```

.
|-- data
|   |-- cln_test
|   |   '-- primary_microphone
|   |       '-- si_dt
|   |           |-- c31
|   |           |-- ...
|   |           '-- c49
|   |-- far_test
|   |   '-- primary_microphone
|   |       '-- si_dt
|   |           |-- c31
|   |           |-- ...
|   |           '-- c49
|   '-- near_test
|       '-- primary_microphone
|           '-- si_dt
|               |-- c31
|               |-- ...
|               '-- c49

```

71 directories

C.4. Directory tree for REVERB_WSJCAM0_tr database (training set)

```

.
|-- data
|   '-- mc_train
|       '-- primary_microphone
|           '-- si_tr
|               |-- c02
|               |-- ...
|               '-- c21

```

96 directories

References

- [1] T. Robinson, J. Fransen, D. Pye, J. Foote and S. Renals, “WSJCAM0: A British English speech corpus for large vocabulary continuous speech recognition”, in *”Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)”*, pp. 81–84, IEEE, 1995.
- [2] M. Lincoln, “The multi-channel Wall Street Journal audio-visual corpus (MC-WSJ-AV): Specification and initial experiments”, in *”Proc. of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)”*, pp. 357–362, 2005.
- [3] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev and P. C. Woodland, *The HTK Book, version 3.4*, Cambridge, UK: Cambridge University Engineering Department, 2006.
- [4] D. B. Paul and J. M. Baker, “The design for the Wall Street Journal-based CSR corpus”, in *HLT ’91: Proceedings of the workshop on Speech and Natural Language*, pp. 357–362, Morristown, NJ, USA: Association for Computational Linguistics, 1992.