
Glott HMM

GlottHMM

Analysis and synthesis software for statistical parametric speech synthesis

Version 1.1
March 19, 2015

Tuomo Raitio
tuomo.raitio@aalto.fi

Contents

1 Introduction

This manual describes speech analysis and synthesis tool GlottHMM. GlottHMM is primarily intended to be used as a vocoder in statistical parametric speech synthesis, but it can be used also for speech analysis and modification. GlottHMM is based on the source-filter theory of speech production, but the main differences compared to other common vocoder techniques is that it utilizes glottal inverse filtering in order to separate the contributions of the voice source and the vocal tract filter and detailed modeling of the voice source signal. This physiologically oriented speech parametrization and synthesis technique aims towards high-quality expressive speech synthesis.

1.1 Background

In 2007, a collaborative project between Aalto University and University of Helsinki was launched in order to develop a physiologically motivated vocoder for hidden Markov model (HMM) based speech synthesis. In 2008, as a result of a Master's thesis, the first version of GlottHMM was born. Since then, GlottHMM has been developed further to a full scale vocoder.

Until now, GlottHMM has only been published as an internal version for research purposes, but finally in March 19, 2015, a first official version of GlottHMM was released. However, it is important to note that GlottHMM is not a product or complete solution as a vocoder, but rather a research tool that can be used for testing and developing vocoder techniques in statistical parametric speech synthesis. Users are encouraged to experiment with the system and provide feedback. The author wishes that the GlottHMM package will foster both research and commercial development of speech synthesis technology, although he acknowledges that the usability of the GlottHMM package might not be the best due to the research-aimed design. The known limitations and issues with GlottHMM are described in Section 6.

1.2 License and distribution

The GlottHMM vocoder is intended to be used both for non-commercial and commercial purposes under the MIT license. The MIT License is a permissive free software license, which permits reuse within both open source and proprietary software. The software is licensed as is, and no warranty is given as to fitness for purpose or absence of infringement of third parties' rights, such as patents. Generally use for research activities is allowed regardless of any third party patents, but commercial use may be subject to a separate license.

If you use GlottHMM within an article, paper, report or any other work that you wish to publish, please cite it as (Raitio et al., 2011b):

> *T. Raitio, A. Suni, J. Yamagishi, H. Pulakka, J. Nurminen, M. Vainio, and P. Alku, "HMM-based speech synthesis utilizing glottal inverse filtering," IEEE Trans. on Audio, Speech and Lang. Proc., vol. 19, no. 1, pp. 153–165, Jan. 2011.*

You may also want to cite this manual or other publications in the references in case you find it informative. If you use the pulse library methods or DNN-based voice source modeling methods, please cite them as (Raitio et al., 2011a, 2014b) (respectively):

> *T. Raitio, A. Suni, H. Pulakka, M. Vainio, and P. Alku, "Utilizing glottal source pulse library for generating improved excitation signal for HMM-based speech synthesis," in Proc. ICASSP, 2011, pp. 4564–4567.*

T. Raitio, A. Suni, L. Juvela, M. Vainio, and P. Alku, "Deep neural network based trainable voice source model for synthesis of speech with varying vocal effort," in Proc. of Interspeech, 2014, pp. 1969–1973.

The MIT license of the program is the following:

For possible commercial use, the author wishes to note that the GlottHMM package may contain methods that are patented by the authors and Nokia Corporation. For more information, see patent "Method, apparatus and computer program product for providing improved speech synthesis" (Raitio et al.).

1.3 Authors and contact

The GlottHMM program is mainly written by Tuomo Raitio, with also contribution from Antti Suni, who has been developing the program especially from the statistical point of view.

Tuomo Raitio
Department of Signal Processing and Acoustics
Aalto University, Espoo, Finland
tuomo.raitio@aalto.fi

Antti Suni
Institute of Behavioural Sciences
University of Helsinki, Helsinki, Finland
antti.suni@helsinki.fi

1.4 Acknowledgements

The author would like to thank Paavo Alku and Martti Vainio who have been successfully leading this research. The author would also like to thank all the collaborators during this project. The research has been funded by Nokia Corporation, the Graduate School at Aalto University School of Electrical Engineering, Aalto University Multidisciplinary Institute of Digitalisation and Energy

(UI-ART), Tekes – the Finnish Funding Agency for Technology and Innovation, the Academy of Finland (LASTU Research Programme 135003 and 256961), and the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement no. 287678 (Simple4All). The author has also been supported in the form of personal grants by the Nokia Foundation, Emil Aaltonen Foundation, Finnish Science Foundation for Economics and Technology (KAUTE), HPY Research Foundation, and Research and Training Foundation of TeliaSonera Finland Oyj.

2 Getting started

This section gives an overview of the GlottHMM tool and helps to get started with it. Sections 3, 4, and 5 give a more detailed description of the current implementation and the use of GlottHMM. Over the years, the GlottHMM vocoder and consequent findings have been reported in various conference and journal papers (see, e.g., (Babacan et al., 2014; Bollepalli et al., 2014; Raitio et al., 2008, 2011a,b; Raitio, 2008; Raitio et al., 2012, 2013, 2014a,b,c; Suni et al., 2010, 2011, 2012, 2013, 2014)). The purpose of this document is to describe the latest version of GlottHMM in technical detail.

GlottHMM package consists of the following main components: Analysis, Synthesis, Matlab scripts, and other files. Analysis is a program that reads in speech files and converts them into suitable speech parameters and extracts glottal flow pulses. Synthesis reads in the speech parameters given by the Analysis or generated from statistical models, and reconstructs synthetic speech using one of the available voice source modeling methods in GlottHMM, for example, using either single glottal flow pulse, pulse library, or a pulse generated from deep neural networks (DNNs). The Matlab scripts are used to further refine data extracted by the Analysis. For example, they can be used to construct pulse libraries or to train deep neural network based voice source models. The other files are used, for example, to configure settings for Analysis and Synthesis. The files included in the GlottHMM package are listed in Table 1.

2.1 Installing

The `Makefile` (run by typing `make`) provided with in folder `src` should automatically compile the source code and build the executables `Analysis` and `Synthesis`. After that, by running `make install`, the binaries are also copied into a folder `bin` in the GlottHMM root. In case the `Makefile` does not work, you may want to read further and check the following points.

GlottHMM is mostly written in standard C, but the following libraries outside standard C are required for compiling and using Analysis and Synthesis:

- GNU Scientific library (GSL) v. 1.15 (`www.gnu.org/software/gsl`)
- Libconfig v. 1.4.9 (`www.hyperrealm.com/libconfig`)
- Libsndfile v. 1.0.25 (`www.mega-nerd.com/libsndfile`)

This version of GlottHMM works at least with the above mentioned versions of the libraries, other versions have not been thoroughly tested. For the C linker, the following libraries must be set (`-l`): `m, gsl, config, sndfile, gslcblas`. Since GlottHMM in not optimized for speed, you may want to increase the running speed of GlottHMM by letting the compiler optimize the binaries. This can be done by uncommenting the `optspeed` option in the Makefile to compile with the `-O3` option.

Mac users probably need to install the libraries. This can be done, for example, by first installing macports, and then running the following command in the terminal: `sudo port install gsl libsndfile libconfig-hr`.

After compiling and installing, Analysis and Synthesis should be ready to run. Settings are already configured for demonstrating the GlottHMM vocoder using the Matlab scripts. However, for further use, the configuration file (`config_default`) may need to be modified according to the speech material and methods to be used. If Analysis or Synthesis is run from a different destination, the paths to high-pass filter file (e.g., `hp_16khz`) and glottal pulse file (e.g., `gpulse`) need to be

Folder	Filename	Description
cfg	config_default	Default config file for Analysis and Synthesis
	config_user_2pp	User config file for 2-pitch-period pulse
	config_user_2pp_new	User config file for 2-pitch-period pulse (new)
	config_user_dnn	User config file for DNN-based synthesis
	config_user_noiaif	User config file for synthesis without IAIF
	config_user_pca	User config file for PCA-based synthesis
	config_user_plib	User config file for pulse library based synthesis
	hp_16khz	High-pass filter coefficients for 16 kHz speech
	hp_44khz	High-pass filter coefficients for 44.1 kHz speech
DNNTtrain	backprop.m	File related to DNN training
	CG_MNIST.m	File related to DNN training
	grbm.m	File related to DNN training
	makebatches.m	File related to DNN training
	prepare_data_sel_scale.m	File related to DNN training
	rbm.m	File related to DNN training
	rbmhidlinear.m	File related to DNN training
	rbm_pretraining.m	File related to DNN training
	save_weights.m	File related to DNN training
	TestDNN.m	Script for testing DNN
	TrainDNN.m	Script for training DNN
doc	glotthmm_manual.pdf	GlottHMM manual (this file)
pulses	gpulse	Single-pitch-period glottal flow pulse
	gpulse_2pp	Two-pitch-period glottal flow derivative pulse
scripts	analysis_synthesis_example.m	Matlab script for demonstrating vocoding
	create_dnn_training_data.m	Matlab script for creating DNN data
	create_pulse_library.m	Matlab script for creating pulse libraries
	hp_filter_design.m	Matlab script for designing filters for Analysis
	pulselib2separate_pulses.m	Matlab script for saving individual pulses
	pulselib_pca.m	Matlab script for performing PCA for a library
	synthesis_example.m	Matlab script for demonstrating synthesis
src	Analysis.c	Source code for speech analysis (main)
	AnalysisFunctions.c	Source code for speech analysis (functions)
	AnalysisFunctions.h	Source code for speech analysis (header file)
	Makefile	Makefile for compiling the source code
	Synthesis.c	Source code for speech synthesis (main)
	SynthesisFunctions.c	Source code for speech synthesis (functions)
	SynthesisFunctions.h	Source code for speech synthesis (header file)
wav	mv_0001.wav	Example speech file for demonstration
root	LICENSE	GlottHMM license
	README	Short tutorial on how to use GlottHMM

Table 1: Files in the GlottHMM package.

additionally defined in the configuration file. The Matlab scripts also need to be modified according to your environment (e.g., paths).

2.2 Analysis

Both Analysis and Synthesis are evoked by executing them from the command line. As input, Analysis and Synthesis require a file name to be analyzed/synthesized and a configuration file in which the Analysis/Synthesis process is defined. Analysis requires a high-pass filter file (e.g., `hp_16khz`), of which path and name is defined in the configuration file (`config_default`). The high-pass filter file is located in the folder `cfg` and defined accordingly in the configuration file. If another path or pulse is to be used, redefine configuration file setting `HPFILTER_FILENAME`. The configuration settings are described in detail in Appendix A.

Speech parameters of `speech_file.wav` are extracted by executing the following command:

```
>> Analysis speech_file.wav config_default
```

As a results, speech features are written to parameter files `speech_file.speechFeatureX`. The extracted speech features can be defined in the configuration file. The data format of the parameter files is either ASCII or binary, defined in the configuration file. Descriptions and help for Analysis can be evoked by executing the program without parameters.

Both Analysis and Synthesis can be given two different configuration files, one for default parameters, and one for user defined parameters. Thus, the most important and frequent settings can be defined in an additional user configuration file, which overrides the default configurations. For example, the following command line would run Analysis with the user configuration settings:

```
>> Analysis speech_file.wav config_default config_user
```

The default configuration file must include all the settings, but the user configuration file may contain only part of the settings, or need not to be given at all. The use of two separate configuration files is intended for easy tuning of the most common parameters in user configuration file, while the others can be left unchanged in default configurations. The most important settings in the configuration for Analysis are for example

- Define sampling frequency
- Define order of the all-pole model
- Define voicing thresholds (zero-crossings, gain, max/min $f0$)
- Define spectral modeling technique
- Define extracted speech features

The analysis methodology and the speech features are described in detail in Section 3. The further processing of the extracted speech parameters and glottal flow pulses, such as constructing pulse libraries and training deep neural network based voice source models, are described in Sections 3.3 and 3.4, respectively.

2.3 Synthesis

Speech defined by the parameters `speech_file.speechFeatureX` is reconstructed by executing the following command:

```
>> Synthesis speech_file config_default
```

As a result, Synthesis reads the speech features, and synthesizes a speech file `speech_file.syn.wav`. It is important to note that the speech file name is given without any extensions (e.g., `wav` in analysis/synthesis). The input parameters for Synthesis can be defined in the configuration file(s), similarly to Analysis. While using the default single pulse synthesis, the pulse file name can be defined using the setting `GLOTTAL_PULSE_NAME`. The pre-computed glottal flow pulse (`gpulse`) provided with the package is already set as default. The most important other settings in the configuration for Synthesis are:

- Select between analysis-synthesis and HMM synthesis using parameter `USE_HMM`. Speech parameters are slightly smoothed for analysis-synthesis in order to remove the effect of framing.
- Select voice source modeling technique (e.g., single pulse, pulse library, DNN)
- Select post-filtering method and strength

The synthesis methodology is described detailed in Section 3. Various voice source source modeling methods that can be used in Synthesis are also described, such as synthesis using single glottal flow pulse (Section 5.1), synthesis using pulse libraries (Section 5.2), and synthesis using a DNN-based voice source modeling (Section 5.3).

2.4 Configuration files

As mentioned before, there are two types configuration files that can be used with GlottHMM, although their format is identical. The default configuration file, provided with the GlottHMM package and given as the second argument for Analysis and Synthesis, includes all the possible configuration parameters. However, user defined configuration file may be given to Analysis and Synthesis as an additional argument, in which the most common or frequently changed parameters are given. The second configuration file will override the parameters in the default file. Thus, only the parameters that require change to the default values need to be defined separately, which makes the use of the program easier. The user configuration file can be created by copying the settings that the user wants to change easily or more often to a new file called, e.g., `config_user`.

The configuration file format is defined in the libconfig library (`www.hyperrealm.com/libconfig`). Note that integer parameters must be defined without decimals (e.g. sampling frequency is 16000), and decimal numbers must include the decimal point (e.g. frame length in milliseconds is 25.0)!

Parameters in the configuration file are roughly divided into four categories: Common, Noise reduction, Analysis, and Synthesis. Both Analysis and Synthesis share some common parameters, but the parameters that are specific to Analysis or Synthesis are defined separately. For detailed description of the configuration file parameters, see Appendix A.

2.5 Tutorial examples

GlottHMM package includes a few tutorial examples on how to use the vocoder and related methods. Simple tutorial instructions are also found in the `README` file. After compiling/installing the source code, the Matlab script `analysis_synthesis_example.m` can be run, which performs Analysis and Synthesis for a single wav file `mv_0001.wav` included in the folder `wav`. Analysis converts the speech file into parameters, including the ones required for building a pulse library. Synthesis creates speech file from the parameters and using a glottal flow pulse `gpulse`, and writes the file to the same directory named as `mv_0001.syn.wav`. The demo script automatically moves this file to folder `syn` in order to not distract the following demonstrations.

After the simple analysis/synthesis demo, a pulse library can be built by running the Matlab script `create_pulse_library.m`, which automatically extracts parameters from the file(s) and builds

a glottal flow derivative pulse library. It also plots the extracted pulses and some of the speech parameters. Matlab script `pulselib2separate_pulses.m` saves the extracted pulses of a pulse library into separate files to the folder `separate_pulses` in the pulse library directory. It also evaluates the mean of all the pulses and finds a "prototype pulse" among all the pulses (closest to the mean). The mean pulse and the prototype pulse are saved into files `pulse_mean` and `pulse_proto`.

A principal component analysis (PCA) can be performed for the already built pulse library using the script `pulselib_pca.m`. This script decomposes the pulse library into a mean pulse, principal components (PC) after mean subtraction, and their weights (saved in the pulse library as `pulselibname.pca_mean`, `pulselibname.pca_pc`, and `pulselibname.pca_w`). It also converts the pulses extracted from speech file(s) into PC weights (saved as `speech_file.pca_w`).

The pulse library data can be saved into a matrix form for further use by running the script `create_dnn_training_data.m`. A deep neural network (DNN) based voice source model can be then built from the pulse library by using the scripts in the folder `DNNTrain`. A voice source DNN can be trained using the script `TrainDNN.m`. During the training, the files `DNN_Errors.mat` and `DNN_Weights.mat` are updated for each epoch. After (or during) the DNN training, the DNN can be tested using the script `TestDNN.m` that plots the training error and compares the input data to the data generated by the DNN. The trained DNN can be saved into a file using the script `save_weights.m`, and the weights in the created folder `dnnw` can be then used for Synthesis.

After these examples, there are basically four possible voice source modeling methods available. These can be tested by running the Matlab script `synthesis_example.m`, which analyzes the same speech file and synthesizes it using the following voice source modeling methods: 1) Single pulse technique, 2) Pulse library technique, 3) PCA-based technique, and 4) DNN-based voice source modeling technique. Additionally, a single pulse technique using a precomputed and new (separated from the pulse library) 2-pitch period glottal flow derivative pulse, and a single pulse technique without glottal inverse filtering (IAIF) are also demonstrated.

It is important to note that these are only examples on how to use the GlottHMM vocoder and related methods. Naturally, a single file does not contain enough data for building practical pulse libraries or DNN-based voice source models.

3 Analysis

The goal of GlottHMM analysis is to separate the speech signal into two components using glottal inverse filtering: the vocal tract spectrum and the glottal flow signal. The glottal flow signal is further parameterized into various voice source features in order to enable modeling the source in the synthesis stage. The idea behind GlottHMM vocoding is explained in various publications in more detail (see, e.g., (Babacan et al., 2014; Bollepalli et al., 2014; Raitio et al., 2008, 2011a,b; Raitio, 2008; Raitio et al., 2012, 2013, 2014a,b,c; Suni et al., 2010, 2011, 2012, 2013, 2014)) while this manual concentrates on the technical description of the vocoder.

Analysis is executed from the command line by typing `Analysis arg1 arg2 (arg3)`. The first argument is the name of the speech file and the second argument is the default configuration (`config_default`) file name. Optionally, a third argument can be given, defining the name of the user config file that overrides the parameters in the default config file. Before analysis, user must define, e.g., the sampling frequency and the polarity of the speech signal in the config file.

In addition, a high-pass filter coefficient file is required. The purpose of the high-pass filter is to reduce low-frequency components that may cause fluctuations in the estimated glottal flow signal. The high-pass filter should be a finite impulse response (FIR) filter that has a cut-off frequency from 30 to 80 Hz, depending on the speech data, and a roll-off large enough. In the GlottHMM package, a 301-tap FIR filter coefficient file is given for 16 kHz speech signals (`hp_16khz`) and a 901-tap FIR filter coefficient file is given for 44.1 kHz speech (`hp_44khz`). Different types of filters can be easily created for example with Matlab. A simple script for designing FIR filters with Matlab is also provided with the package (`hp_filter_design.m`).

3.1 Technical description

The flow chart of Analysis is shown in Figure 1. Analysis is based on frame-by-frame analysis of speech signals. In the beginning of Analysis, the speech signal is high-pass filtered. The high-pass filter coefficient file is defined in the config file (`HPFILTER_FILENAME`). The high-pass filtering can also be set on/off in the config file (`HP_FILTERING`).

After the high-pass filtering, the speech signal is windowed with two types of windows. A short frame (usually around 25 ms) is used to evaluate (logarithmic) energy of the speech signal and vocal tract and voice source spectra. A longer frame (25–50 ms, depending on lowest f_0) is used to evaluate many voice source features that need slightly longer analysis frame in order to incorporate several pitch period. The lengths of the frames can defined in the config file (`FRAME_LENGTH, F0_FRAME_LENGTH`) in milliseconds. The shift between two adjacent frames is defined in `FRAME_SHIFT` (usually 5 ms).

Glottal inverse filtering (GIF), i.e., the estimation of the glottal flow signal from a speech signal, is applied for both frames separately. Iterative adaptive inverse filtering (IAIF) (Alku, 1992) or a modified version of it (see Figure 2) is used for GIF. GIF outputs the estimated vocal tract filter and the estimated voice source signal.

Inside GIF, various spectral modeling method can be utilized. Normally, basic linear prediction (LP/LPC) is used, but also weighted linear prediction (WLP) or extended linear prediction (XLP) may be used. Weighted linear prediction (Ma et al., 1993), or stabilized version of it, SWLP (Magi et al., 2009) use a function to weight the autocorrelation function that is used for evaluating the LP coefficients. Usually, the autocorrelation is weighted by the short time energy (STE) window of the signal, thus emphasizing high energy parts, but also other weights can be used, such as weight based on glottal closure instants (GCIs). (S)WLP is beneficial especially with high-pitched (female) voices since the weighting results in a spectrum that is less biased by the harmonics of the excitation signal. XLP (Keronen et al., 2011; Pohjalainen et al., 2010) has similar properties to WLP, but in the current implementation it uses only STE weight. Both methods, WLP and XLP can

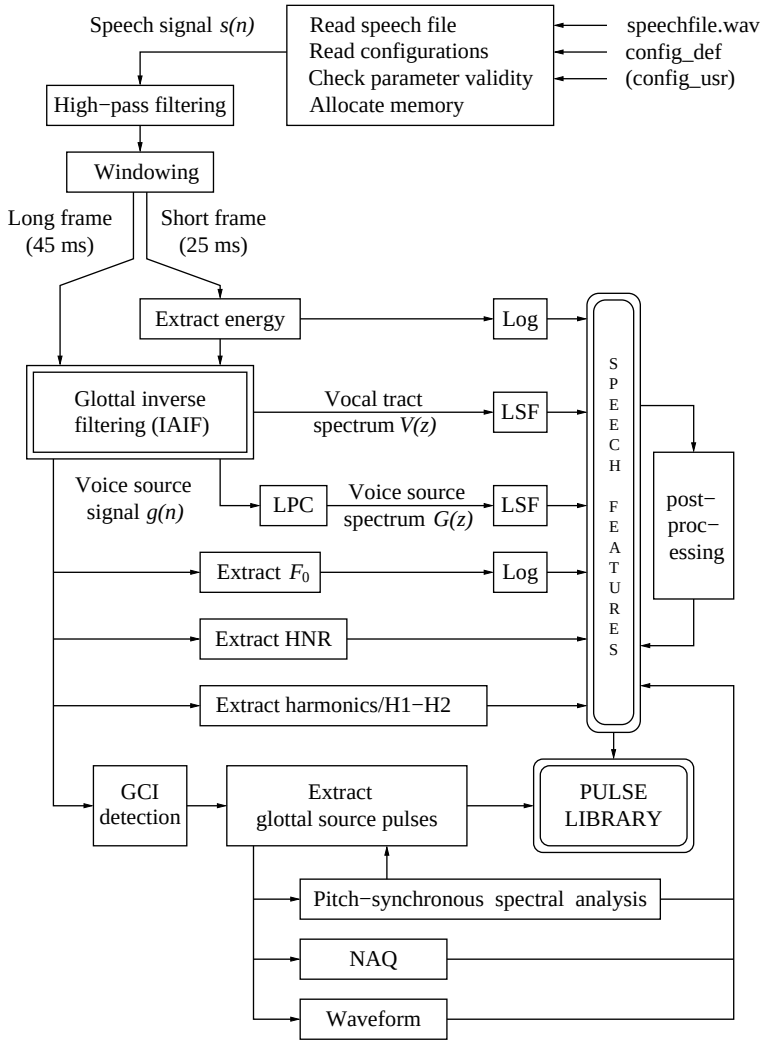

Figure 1: Flow chart of speech parametrization.

possibly produce unstable filters, and thus a stabilized version of them (SWLP, SXLP) can by used (`LP_STABILIZED`). However, the effect of weighting is reduced in the stabilization process. Thus, it is up to experimentation whether possible unstable or badly estimated filters will be averaged out in statistical modeling in order to produce stable and natural synthesis filters. The stabilization process will often result in slightly over-smooth spectrum.

The order of all-pole modeling (LP) depends on the sampling frequency, but also on the speech material. Usually 20th–30th order model is appropriate for 16 kHz speech, but higher sampling rates may require higher orders from 30 to 50. However, it is usually better to use warping of the spectral model with higher sampling rates than overly increase the LP order (e.g., for 44.1 kHz speech, `WARPING_VT` can be set to 0.4 and 30th order model can be used). The accuracy of LP analysis and statistical modeling decrease if the order is more than 50.

After the voice source signal is estimated (short frame), LP is used to estimate the spectrum of the voice source, i.e., the spectral tilt and the more detailed spectral structure. This spectral model

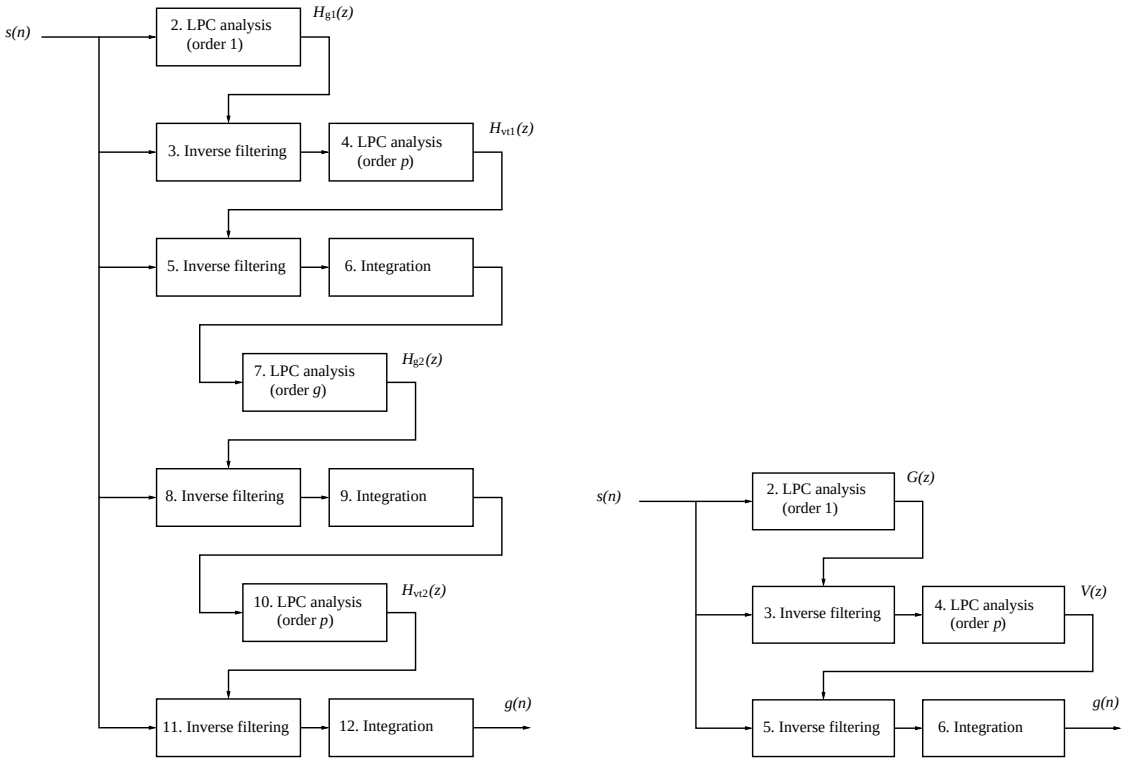

Figure 2: Iterative adaptive inverse filtering (IAIF) method (left) and its modified simpler version (right).

is used in synthesis, for example, to vary the voice source spectrum (single pulse technique) or to select appropriate glottal flow pulses from library. Both vocal tract and voice source LP coefficients are converted to line spectral frequencies (LSF) which provide stability and low spectral distortion in statistical modeling.

The voice source signal estimated from the longer frame is used to estimate the rest of the speech features. First, fundamental frequency (f_0) is estimated with the autocorrelation method. Although this is very simple technique, it yields rather robust estimates of f_0 if used for glottal inverse filtered signal and if the f_0 range is properly set. The resolution of f_0 estimation (limited by the sampling rate) is increased by parabolic interpolation of the autocorrelation peak position. User must define the minimum and maximum f_0 in the config file, and thus values outside this range are classified as unvoiced. Voicing threshold is used to assist in voiced/unvoiced decision; frames whose low-frequency energy is less than the given threshold are classified as unvoiced. Also zero-crossing rate (ZCR) is used to help in the classification; frames with ZCR values exceeding the defined threshold are classified as unvoiced. The f_0 vector is finally median filtered, and small isolated voiced/unvoiced gaps are fixed. Finally, a post-processing algorithm can be applied in order search for possible discontinuities (`RELATIVE_F0_THRESHOLD`) in f_0 trajectories, and to estimate a new trajectory by weighted linear fitting over a defined number of f_0 values (`F0_CHECK_RANGE`).

Alternatively, an external f_0 file can be used as an input for the system. The input vector must be in ASCII format, and each f_0 value must be on a separate line. The f_0 file need not to be exactly of correct length since an interpolation for the $f0$ vector is applied in case of different length. The interpolation process may produce occasional very small f_0 values and thus the lowest f_0 of the

interpolated f_0 vector is limited to F0_MIN.

A harmonic analysis is performed in order to extract the harmonic-to-noise ratio (HNR) of the speech frame (actually the HNR parameter is presented as noise-to-harmonic ratio, but for convenience, it is called HNR). This feature describes the degree of voicing, i.e. the amount of aperiodic noise in comparison the periodic glottal excitation. However, unfortunately this features does not distinguish between true aperiodicity and, e.g., irregularity of the voice and rapid changes in $f0$. Thus, it is possible that HNR is slightly biased by the aforementioned effects. The evaluation of HNR begins by first performed the fast Fourier transform (FFT), after which a peak picking algorithm is applied to find the harmonic peaks in the spectrum. Then, the magnitudes of the peaks and the harmonic valleys are estimated from the spectrum, from which the upper and lower envelopes, represented by the harmonic peak and valley magnitudes, respectively, are formed. The difference between these vectors defines the HNR at each frequency. The HNR values are averaged according to the equivalent rectangular bandwidth (ERB) scale to a specific number of frequency bands, defined in the config file (`HNR_CHANNELS`).

In addition to HNR which is crucial for successful synthesis, also the magnitude difference between the first and the second harmonics, called the H1-H2, is saved as a separate speech feature (H1H2). Also the first N harmonic magnitudes are saved to describe the low-frequency properties of the source. The number of harmonics is defined in config file (`NUMBER_OF_HARMONICS`). The harmonic values are saved as a difference to the first harmonic magnitude in dB. Thus, the first value of the harmonics vector is the difference between the first and the second harmonic, called H1-H2. It is important to note that these two features are additional speech that are NOT used in conventional GlottHMM synthesis, but they can be used, for instance, as target costs in pulse library based synthesis or as input parameters in DNN-based voice source modeling).

GlottHMM analysis also involves method for glottal closure instant (GCI) detection. GCI estimation is initiated by first searching for the minimum of the differentiated glottal volume velocity signal, which is expected to be the most prominent GCI in a voiced frame. From this point, other minima, representing the other GCIs, are searched backward and forward at fundamental period (T_0) intervals. If an appropriate minima is found in the vicinity of T_0, it is classified as a GCI. Note that the polarity of the speech signal must be correct in order estimate GCIs correctly. The polarity can be changed by setting the config parameter `INVERT_SIGNAL` to `true`.

After GCI detection, each complete two-period glottal flow derivative is extracted. The pulse delimited by two GCIs (start and end points) and there is a GCI in the center of the pulse. Although this segment is composed of two glottal flow pulses, the segment is simply called as *pulse* in this context. If the T_0 of the pulse is not close enough to the current f_0, the pulse segment is discarded. The threshold for discarding pulses can be tuned with the config parameter `MAX_PULSE_LEN_DIFF`. The maximum length of the pulse is defined in `PULSEMAXLEN`. Accepted pulses are windowed with the Hann window so that the GCI of the two-period segment is in the middle of the frame. This enables the use of overlap-add method in the synthesis stage for concatenating the pulses. The energy of each pulse is normalized, after which the pulses are stored to a matrix. In addition, a resampled version of the pulse is stored in order to enable easy evaluation of the concatenation cost in the synthesis stage. The length of the resampled pulse is defined in `RESAMPLED_PULSELEN`.

Note that the "pulses" extracted by Analysis are different from the pulse used in single pulse based synthesis using the provided `gpulse` file, which is a single-period glottal *flow* pulse. Here the *pulses* are actually *differentiated* glottal flow pulses and consist of two segments. These pulses are used in data-based voice source modeling methods, such as in pulse library method and DNN-based voice source modeling method. These two-pitch-period pulses can be also used in single pulse synthesis by setting the configuration parameter `TWO_PITCH_PERIOD_DIFF_PULSE` to `true`.

Two additional parameters are extracted from each pulse. First, the normalized amplitude quotient (NAQ) (Alku et al., 2002) of each pulse is estimated and saved as a separate feature. Also, the basic shape of the pulse is stored by downsampling the pulse only to a few samples of length

(usually around 20). The samples in the center (others are close to zero) are stored to a matrix to describe the pulse shape. Again, these features are not used or required in normal GlottHMM synthesis.

If config file parameter `PITCH_SYNCHRONOUS_ANALYSIS` is set to `true`, the glottal inverse filtering is performed pitch-synchronously for each pulse. This enables individual spectral parameters for each pulse instead of frame-wise parameters, and also more accurate estimates of the pulses. Additionally, pitch-synchronous analysis reduces the effect of the voice source harmonics to the vocal tract estimate. Thus, pitch-synchronous analysis is recommended especially for high-pitched voices. (S)WLP or (S)XLP is not used in pitch-synchronous analysis regardless of the spectral modeling settings. On the other hand, pitch-synchronous analysis may sometimes result in less accurate estimates if the GCIs are not properly estimated.

Finally, the speech feature trajectories are post-processed using, e.g., median filtering in order to remove possible small errors and outliers, and saved to files.

3.2 Speech features

As a results of Analysis, speech features are written to parameter files `speechFile.speechFeatureX`. The extracted speech features can be specified in the configuration file. The data format of the parameter files is either ASCII or binary, defined in the configuration file. All possible speech features are listed in Table 2, although only few of them are required for normal synthesis.

Additionally, if `EXTRACT_PULSELIB` is set to `true` (for constructing a pulse library), the glottal flow pulses and the corresponding parameters listed in Table 3 are also extracted. Analysis also writes an infofile that contains 15 values which define some of the analysis settings. The structure of the infofile described in Table 4. Finally, the result of the glottal inverse filtering, the complete overlap-added voice source signal, can be extracted by setting `EXTRACT_SOURCE = true`. The voice source is saved both to parameter `SourceSignal` and as a sound file `SourceSignal.wav`.

Speech feature	Description	Typical order
$f0$	Fundamental frequency	1
Gain	Energy of frame (dB)	1
LSF	Vocal tract spectrum	20–30
LSFsource	Voice source spectrum	6–14
HNR	Harmonic-to-noise ratio	5–10
Harmonics*	Magnitude differences of harmonics	5–10
H1H2*	Difference between H1 and H2	1
NAQ*	Normalized amplitude quotient	1
Waveform*	Downsampled waveform	10–20
infofile*	Information about analysis	15 values
FFT_VT*	Vocal tract FFT spectrum	256
FFT_SRC*	Voice source FFT spectrum	256
SourceSignal*	Estimated voice source signal (ASCII)	Signal length
SourceSignal.wav*	Estimated voice source signal (wav)	Signal length

Table 2: Speech features and the typical order of the parameters. Features marked with a star (*) are not mandatory for normal synthesis, but may be used, e.g., for voice analysis or in some voice source modeling methods. The extraction of these speech features can be set off in the configuration file (see Appendix A).

Speech feature	Description
PULSELIB.pulses	Library pulses
PULSELIB.rspulses	Resampled library pulses
PULSELIB.pulselengths	Actual lengths of the library pulses
PULSELIB.pulseinds	Sample indices indicating the beginning time of the pulse in the original speech file
PULSELIB.pulsepos	Frame indices indicating from which frame the pulses were extracted
PULSELIB.gain	Frame energy for the current pulse
PULSELIB.lsf	Vocal tract spectrum of the pulse frame
PULSELIB.lsfsource	Voice source spectrum of the pulse/pulse frame
PULSELIB.hnr	Harmonic-to-noise ratio of the frame
PULSELIB.harmonics*	Magnitude differences of harmonics
PULSELIB.h1h2*	Difference between H1 and H2
PULSELIB.naq*	Normalized amplitude quotient of the pulse
PULSELIB.waveform*	Downsampled waveform of the pulse

Table 3: Features for constructing pulse libraries. Features marked with a star (*) are not mandatory for normal synthesis, but may be used as additional features.

3.3 Creating pulse libraries

Pulse libraries can be constructed by using the Matlab function `create_pulse_library.m`. The function runs Analysis that extracts speech parameters and pulses from selected files, copies all parameters into a folder called `parameters`, and finally constructs a pulse library from the extracted pulses and corresponding parameters. The pulse libraries are saved in folder `pulse_libraries`. In order to construct pulse libraries, the configuration setting `EXTRACT_PULSELIB` must be set to `true`, and user also needs to define path to Analysis executable, wav file directory, and the configuration file (but these are set already correctly in the package for the tutorial examples).

The number of files (N) from which the pulse library is constructed can be set, and the first N files in the wav-directory will be used for building the pulse library. Usually only a few dozens of wav files is enough for constructing an appropriate pulse library for one person with single voice quality. Larger pulse libraries can be also built depending on the application, but the synthesis time will increase with the increased size of the pules library. However, in order to get a representative set of glottal flow pulses, speech files with different speaking styles may be required. Naturally, female voices contain much higher number of pulses than male voices, and thus the size of the pulse library may become very large with only a few female speech files. In order to reduce the size of the pulse library, it can be pruned using a simple k-means clustering method (implemented in the Matlab function) which selects only N centroids of the pulse library, thus reducing the number of pulses but aiming to maintain the variety of different types of pulses. For large pulse libraries, binary data formant is recommended for faster reading in Synthesis. The pulse library is finally saved to folder `pulse_libraries/pulse_library_name`.

The pulse library can be saved into 2-pitch period glottal flow derivative pulses in separate files using the Matlab script `pulselib2separate_pulses.m`. The pulses are saved into folder `separate_pulses` in the pulse library directory. The function also evaluates the mean of all the pulses and finds a "prototype pulse" among all the pulses (closest to the mean). The mean

Line	Description	Example value
1	Frame length (ms)	25.0
2	Frame shift (ms)	5.0
3	Number of frames	535
4	LP order for the vocal tract	30
5	LP order for the voice source	20
6	Warping coefficient for vocal tract	0.0
7	Warping coefficient for voice source	0.0
8	Number of HNR channels	5
9	Number of harmonics	10
10	Number of extracted pulses	1256
11	Maximum length of pulses (ms)	45.0
12	Length of resampled pulses (ms)	10.0
13	Number of samples in waveform parameter	10
14	Sampling frequency (Hz)	16000
15	Data format (1:ASCII, 2:Binary)	1

Table 4: Structure of the infofile.

pulse and the prototype pulse are saved into files `pulse_mean` and `pulse_proto`. The separate pulses can then be used in synthesis using the single pulse technique by setting the path and the name of an individual pulse to the configuration parameter `GLOTTAL_PULSE_NAME` and setting `TWO_PITCH_PERIOD_DIFF_PULSE` to `true`.

Principal component analysis (PCA) can be also performed for the pulse library in order to decompose the pulses into their principal components and the corresponding weights. This can be done using the Matlab function `pulselib_pca.m`. The function can also convert extracted glottal flow pulses from a speech file to the corresponding PC weights that can be, e.g., trained using HMM and used in synthesis for reconstructing the pulses from the generated weights.

Pulse libraries are used in Synthesis by setting the configuration file parameter `USE_PULSE_LIBRARY` to `true` and setting the pulse library name to the configuration file parameter `PULSE_LIBRARY_NAME` (e.g., `pulse_libraries/pulselib1/pulselib1`). Various other options for the pulse library can also be set. For further information on these, see Appendix A. PC weights can be used in synthesis for reconstructing pulses by setting `USE_PULSELIB_PCA` to `true`.

Figure 3 shows a few pulses from a pulse library of a male speaker. The pulse library methodology is described in more detail for example in (Raitio et al., 2011a). Utilizing the PCA-based synthesis is described in (Raitio et al., 2013).

3.4 Training deep neural network based voice source models

The GlottHMM package also includes methods for training deep neural network (DNN) based voice source models. The method is based on training a mapping between the extracted speech parameters and sample-wise GCI-centered pulses that are resampled to constant length. In synthesis, new pulses can be generated by generating speech parameters from HMMs and feeding these parameters to the DNN, which then outputs a pulse for synthesis.

The training methodology is originally based on the works of Ruslan Salakhutdinov and Geoff Hinton (Salakhutdinov and Hinton, 2015), but the some of the files have been modified for the purpose of training DNN-based voice source models. In the following, a practical guide how to use the function for building DNN-based voice source models is presented. The methodology is described

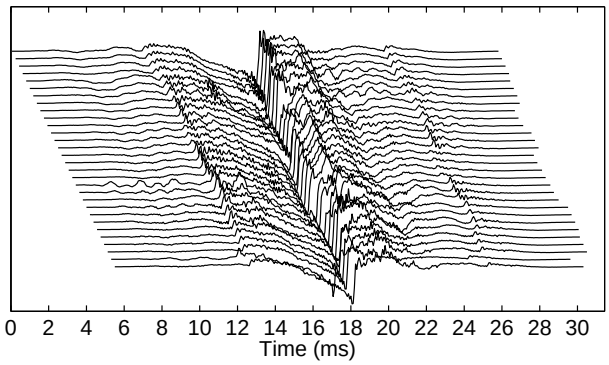

Figure 3: Windowed two-period glottal volume velocity pulse derivatives (pulses) from the pulse library of a male speaker extracted with the automatic analysis scheme.

in more detail for example in (Raitio et al., 2014a,b; Suni et al., 2014).

Methods for the DNN training are included in the folder `DNNTrain`. First, a pulse library needs to be constructed using the Matlab function in the main GlottHMM package (`create_pulse_library.m`). After that, the pulse library data is converted into two matrices, `data_in` and `data_out`, using the function `create_dnn_training_data.m`. In this Matlab function, the pulse library name and the number of samples in the output pulse are defined. The matrices are in Matlab data format, and their size are $N \times p_{in}$ and $N \times p_{out}$, where N is the number of data points, p_{in} is the order of input speech parameter vector, and p_{out} is the number of samples in the output pulse.

After this, a data file is saved to disk that can be then used for training a DNN-based voice source model using the function `TrainDNN.m`. In this function, various options and settings can be set, such as the number of hidden layers, the number of units in each layer, the maximum number of training epochs, batch size, learning rate, RBM pre-training, etc. Also, the input data can be processed in various ways. First, the input mat-file is defined after which the amount of data to be used is defined. Then the proportion of training data to evaluation data is defined. It is usually good to scale the input data to a restricted scale for more efficient training—this can be set on here as well (the scale is [0.1, 0.9]). Also, outlier data can be also discarded here, but it is not mandatory (and might even make the model too simple).

Before the training, the data is split into training and validation data (the latter is in this case named as test data), and separately saved into input and target data. During the training, the DNN weights and errors are saved to files `DNN_Weights.mat` and `DNN_Errors.mat`, respectively. After the training is converged, the data can be saved to a readable format for synthesis using the function `save_weights.m`. The data is saved into a folder, where each layer has its weight files. File `input.minmax` defines the scaling factors for each data stream in order to scale the input parameters accordingly also in the synthesis. In addition, a readme file is also saved for preserving the information about the trained DNN. Synthesis using DNN-based voice source modeling is described in Section 5.3.

The training and synthesis using the DNN-based voice source modeling is illustrated in Figure 4.

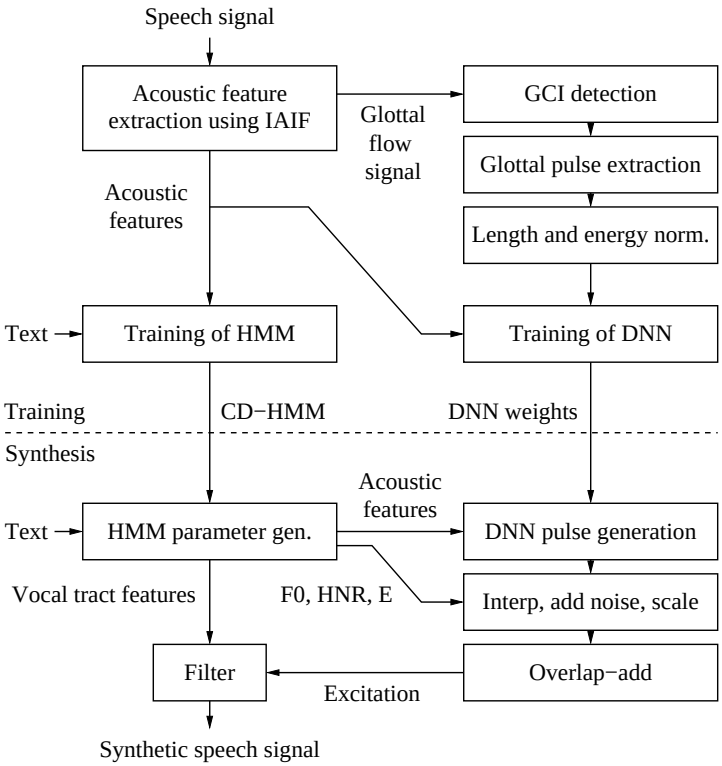

Figure 4: Illustration of speech synthesis using DNN-based voice source modeling.

4 HMM training and parameter generation

GlottHMM package as such does not provide direct means to build HTK (Hidden Markov Model Toolkit) compatible features. In order to integrate the GlottHMM vocoder into any speech synthesis system, some additional scripts are required. For example, required scripts can be found in the Simple4All automatic front-end builder Ossian (Simple4All, 2015).

4.1 Stream structure

The currently used configurations for HMM training are described below, but experimentation is encouraged.

- Four different streams:
  1. Vocal tract LSF (30 coefficients) + Gain (1 coefficient)
  2. Source LSF (6–14 coefficients)
  3. Harmonic-to-noise ratio, HNR (5–10 bands)
  4. Fundamental frequency, $f0$ (1 dimension)
- Additional parameters (H1-H2, NAQ, Waveform) can be trained for pulse selection.
- Only $f0$ has been modeled with multi-space distribution (MSD), but it is also possible for HNR and source LSFs (and H1-H2, NAQ, Waveform) too since they are redundant in unvoiced sections.

4.2 Feature extraction

For feature extraction, the following procedures are required:

- Combine vocal tract LSF and gain
- Add dynamic (delta and acceleration) features
- Handle MSD streams
- Build HTK-compatible features

Modification from, e.g., STRAIGHT-based scripts should not be very difficult

4.3 HTS training and clustering

The configurations need to be modified to suit selected model structure, otherwise no special modifications are required. The stream weights should be preferably set to vocal tract LSF, F0, and duration.

4.4 Parameter generation

In order to alleviate muffled speech caused by over-smoothing, post-filtering (formant enhancement) should be used, possibly in combination with global variance (GV) parameter generation. Typically small amount of GV (< 0.5) combined with LSF or LPC-based post-filtering (around 0.5) leads to good results.

5 Synthesis

This section describes the operation of Synthesis. Synthesis is executed from the command line by typing `Synthesis arg1 arg2 (arg3)`, where the arguments are the name of the speech file without `.wav` extension, default config file, and optional user config file. Before synthesis, either analysis of the same file must be performed, or parameters must be generated from HMMs. For analysis-synthesis, parameter `USE_HMM` should be set to `false`, and for speech synthesis using statistical modeling, the parameter should be set to `true`. For analysis-synthesis the speech parameters are slightly smoothed in time, but for HMM-based synthesis this is not performed. Additionally, instead of synthesizing a single file at a time, a synthesis list, defined in the config file (`SYNTHESIZE_MULTIPLE_FILES`, `SYNTHESIS_LIST`), can be used to synthesize multiple files at once. This is especially useful when synthesizing speech using a large pulse library so that the pulse library is loaded only once.

Synthesis is based on first creating the voiced and unvoiced excitations and then filtering the combined excitation with the vocal tract filter. The voiced excitation can be constructed, e.g., the following voice source modeling methods: (1) Using a single glottal flow pulse extracted from natural speech, which is interpolated and scaled in magnitude to create an appropriate voice source. (2) Using a pulse library, consisting of various glottal flow pulses extracted from natural speech, from which the best candidates are selected for creating the voice source. (3) Using a trained DNN to automatically generate glottal flow pulses. The synthesis methodology varies depending on the voice source modeling method, and in the following the three main methods are described in more detail.

5.1 Synthesis with single pulse technique

Synthesis with a single pulse technique is illustrated in Figure 5. The process begins with creating the voiced and unvoiced excitations. The unvoiced excitation is composed of white noise scaled by the energy. The voiced excitation is constructed by interpolating the glottal flow pulse according to f_0 and scaling it according to magnitude, and concatenating the pulses for voiced sections. The glottal flow pulse can be either a single-pitch-period pulse extracted from natural speech or a two-pitch-period glottal flow pulse derivative waveform. The path and name of the glottal flow pulse is defined in the config file parameter `GLOTTAL_PULSE_NAME`, and the type of pulse (single-pitch-period/two-pitch-period derivative) is selected using the configuration setting `TWO_PITCH_PERIOD_DIFF_PULSE` (set to `true` for the latter). A single-pitch-period glottal flow pulse extracted from 16 kHz male voice (`gpulse`) and a two-pitch-period glottal flow derivative pulses (`gpulse_2pp`) are provided with the GlottHMM package. In practice, the pulses can be used for synthesizing various speakers (even both males and females) since the pulses are later modified in the synthesis stage to fit the intended speaker by, e.g., adding noise and modifying the spectral properties of the pulse (called spectral matching). However, it is probably beneficial to use pulses from the original speaker due to, for example, differences in phase and noise characteristics of the pulses (i.e., the temporal fine-structure). However, extracting an appropriate single-pitch-period pulses might not not straightforward due to possible discontinuity between the beginning and the end of the single pulse. However, extracting new two-pitch-period pulses is easier. This can be done by first constructing a pulse library, and the running the Matlab script `pulselib2separate_pulses.m` that saves the pulses of a pulse library into the two-pitch period glottal flow derivative pulses in separate files. The function also saves the mean pulse evaluated from the all the pulses in the library and a prototype pulse (the pulse closest to the mean) into files `pulse_mean` and `pulse_proto`. Then, by setting the name of the desired pulse (found in the folder `separate_pulses` in the pulse library directory) to the config parameter `GLOTTAL_PULSE_NAME` and setting `TWO_PITCH_PERIOD_DIFF_PULSE` to `true`, synthesis using these individual pulses is performed.

Before concatenating the pulses, noise is added according to the harmonic-to-noise ratio (HNR). This is performed separately for each pulse (after interpolation and gain adjustment) by calculating

the FFT of the pulse, adding a random component to each frequency bin of the spectrum according to the ERB-based HNR measure, and reconstructing the pulse by IFFT. Thus for each frequency band, a correct amount of noise is added. However, in order to prevent too much noise in the low frequencies (due to possible errors in HNR analysis), a low frequency limit (`NOISE_LOW_FREQ_LIMIT`) can be set so that below that frequency, no noise is added. For the single-pitch-period pulse, the noise has more adverse effects than for the two-pitch-period pulse due to the direct concatenation of the former. Thus, a lower low-frequency limit can be used for the latter. Also the gain of the unvoiced noise can be controlled using the parameter `NOISE_GAIN_VOICED`. Additionally, the HNR can be re-estimated from synthetic frames during synthesis and compared to the original HNR, and thus the difference can be compensated by setting the parameter `HNR_COMPENSATION` to `true`.

After adding the noise, the spectrum and HNR of each pulse is estimated again. This is done by reconstructing an analysis frame from the pulses, and evaluating the LP spectrum and HNR of the frame with the same methods as was performed in Analysis. Thus, the source spectrum and HNR of the excitation can be later modified to correspond to the desired one.

Once the voiced excitation is created, the spectrum of the excitation is modified in order to match it with the spectrum given in the voice source spectrum parameter. This is called spectral matching. This is performed by filtering the whole voiced excitation with an infinite impulse response (IIR) filter, consisting of the inverse of the estimated spectrum of the reconstructed excitation, and the the actual given voice source spectrum. This process can be described by first flattening (whitening) the reconstructed excitation, and then applying the desired voice source spectrum. This gives the created excitation the desired spectral tilt and the more detailed spectral variation that should exist in natural voice source. Finally, the lip radiation is applied to the reconstructed glottal flow, which corresponds to differentiating the signal.

Before filtering, the vocal tract filter coefficients are post-filtered, meaning that the spectral structure, i.e, the formants, are enhanced in order to alleviate the oversmoothing of the statistical modeling. There are two methods for post-filtering, which can be set on/off in the config file with parameter `POSTFILTER_METHOD`. Options for the parameter are `"NONE"`, `"LSF"`, and `"LPC"`. The LSF-based method (Ling et al., 2006) slightly shifts the vocal tract LSF pairs closer to each other in order to sharpen the formants. The LPC-based method (Raitio et al., 2010) modifies the spectrum generated by the LP coefficients so that formants are more prominent, and then re-evaluates LP coefficients through the autocorrelation method. The strength of the post-filtering is adjusted by the config parameter `POSTFILTER_COEFFICIENT`. The range of the formant enhancement coefficient is [0,1] so that smaller values result in stronger post-filtering.

The voiced and unvoiced excitations are finally combined and then filtered with the vocal tract filter. However, the spectral matching and filtering may cause variations in the signal energy, and thus the synthesis is performed again with normalized gain values.

5.2 Synthesis with pulse library technique

The pulse library technique is based on selecting the best matching pulses according to target and concatenation costs in order to reconstruct a realistic voice source. In this sense, the technique resembles the classical unit-selection framework, where speech units are selected from a database and concatenated to create new instances of speech. However, the fundamental difference between the conventional unit-selection framework and the pulse library technique is that the number of units required for natural sounding synthetic speech is very low in the pulse library technique. This is because the two components, the glottal source and the vocal tract filter, are separated by glottal inverse filtering, and thus only the varying context of the voice source need to be stored into a pulse library, and the variation due to vocal tract filter can be modeled by the HMM.

Synthesis with pulse library can be used by setting the configuration file parameter `USE_PULSE_LIBRARY` to `true` and typing the path and name of the pulse library to parameter

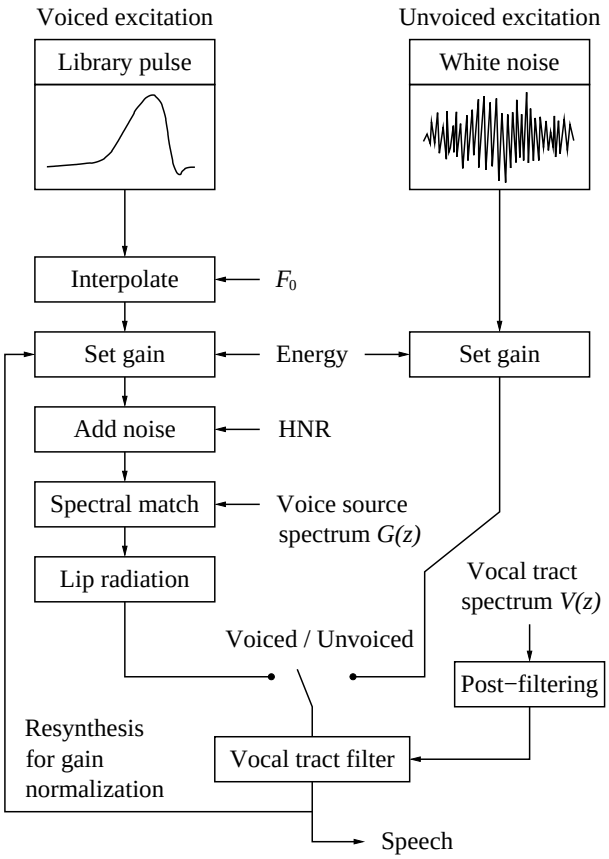

Figure 5: Illustration of synthesis with the single pulse technique.

`PULSE_LIBRARY_NAME`. Synthesis with the pulse library based method is illustrated in Figure 6. The pulse library consists of two-pitch-period segments of the glottal flow derivative waveform. These pulses are described by the speech features extracted in the analysis phase. These *pulses* are selected for the voiced sections of the excitation signal according to target cost based on the speech parameters and the concatenation cost between adjacent pulses. Minimizing the target cost of the voice source parameters ensures that a pulse with desired properties (fundamental period, spectral tilt, the amount of noise, etc.) is most likely to be chosen. The target cost consists of the error between the speech features of the library pulses and the given target speech features generated by HMM. For each target parameter (LSF, LSFsource, Harmonics, HNR, Gain, F0, Waveform, H1H2, and NAQ) an individual weight may be defined in the config file setting `PARAMETER_WEIGHTS`.

The concatenation cost consists of the root mean square (RMS) error between the consecutive downsampled pulses. Minimizing the concatenation cost ensures that adjacent pulse waveforms do not differ substantially from each other, possibly producing abrupt changes in the excitation signal leading to a harsh voice quality. In order to prevent selecting the same pulse in a row, possibly resulting in a bad speech quality, a bias can be set to the concatenation error with the config file parameter `PULSE_ERROR_BIAS`.

The weights of the target and concatenation costs can be defined with `TARGET_COST` and `CONCATENATION_COST` in the config file. The selection process is performed for each continuous voiced section and optimized with the Viterbi algorithm.

Since $f0$ is included in the target cost, a pulse with approximately correct fundamental period is very likely to be chosen, and thus interpolation of the pulse is not usually required. However, interpolation of the library pulses can be used by setting the config parameter `USE_PULSE_INTERPOLATION` to `true`. Also the HNR of the pulses should ideally be correct, but the noise addition can be enabled with the parameter `ADD_NOISE_PULSELIB`.

After the selection process, only the energy of the pulse is equalized to the energy measure given by the HMM. The excitation signal is finally generated by overlap-adding the selected pulses according to the current $f0$ value. Thus a pulse train comprising a series of individual glottal flow derivative pulses is generated.

The rest of the pulse library technique is similar to single pulse technique, consisting of combining the voiced and unvoiced sound sources, post-filtering of the vocal tract filter, filtering, and finally resynthesis for gain normalization.

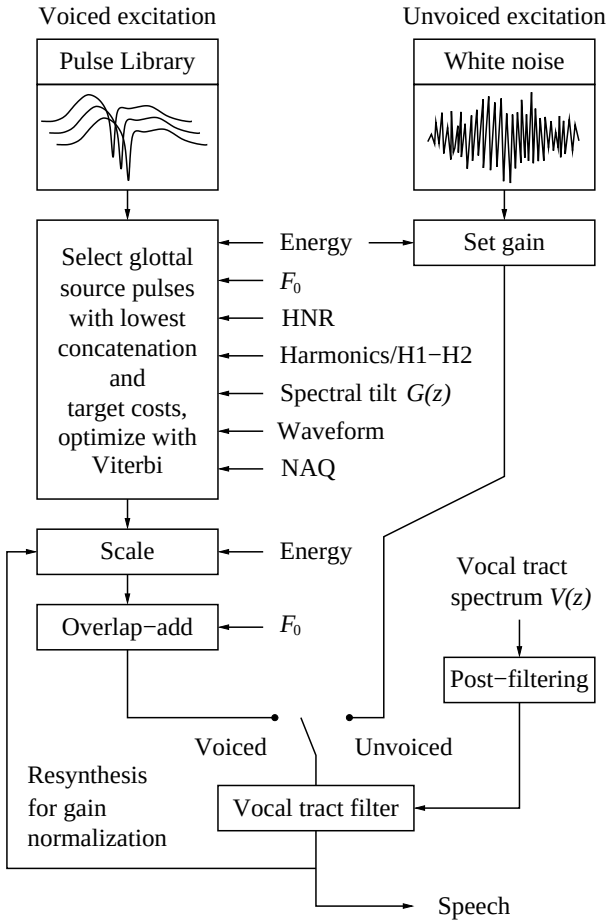

Figure 6: Illustration of synthesis with the pulse library technique.

5.3 Synthesis using a deep neural network based voice source model

In DNN-based voice source modeling, a DNN is first trained for a particular voice (Raitio et al., 2014a), or it can be trained for various voice qualities (Raitio et al., 2014b) or even for various

speakers (Suni et al., 2014). In synthesis, the speech parameters generated from HMM are used as input features to the DNN that then automatically outputs an appropriate pulse corresponding to the features. Synthesis with the DNN-based voices source modeling method is illustrated in Figure 7. The process is otherwise similar to the pulse library method, but the various pulses are generated directly from the DNN.

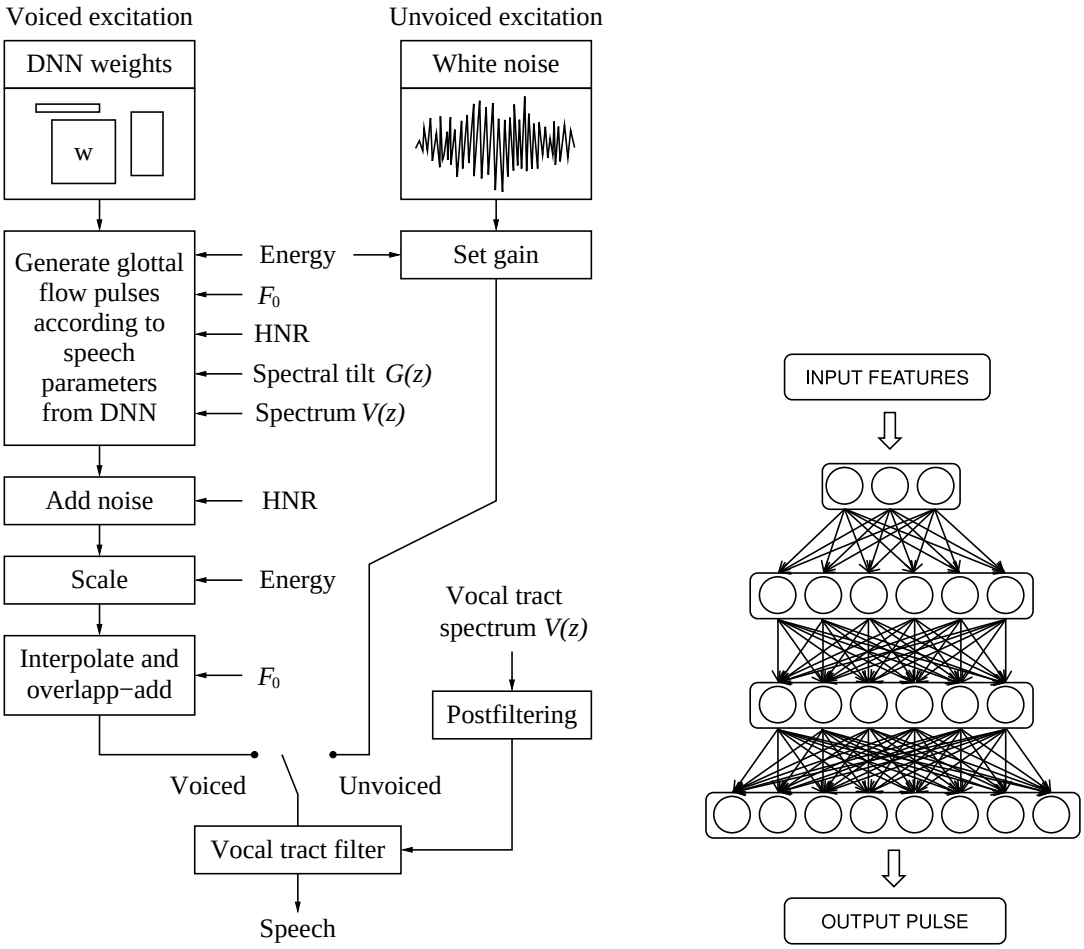

Figure 7: Illustration of synthesis with the DNN-based voice source modeling technique (left) and illustration/example of a deep neural network used for training/generating pulses from speech parameters (right).

In order to use the DNN-based voice source modeling, the config parameter `USE_DNN_PULSEGEN` must be set to true and also `DNN_WEIGHT_PATH` need to be defined. Also the dimensions of the DNN must be defined in `DNN_WEIGHT_DIMS`, where each number defines the 1st and 2nd dimensions of the weight matrices. Thus, if the input parameter vector has 47 dimensions (F0-1, Gain-1, HNR-5, LSFsource-10, LSF-30), the first number is 47 + 1 = 48, and the additional one is due to the input bias. The second number for a 200 unit-hidden layer is 200. The third number, corresponding to the input dimensions of the second layer is 201 (including the bias), and output again should correspond to the dimension of the next hidden layer. The last number in the vector corresponds to the dimension of the pulse, i.e., the number of samples used for describing the pulse waveform

(usually 400).

Since DNN automatically generates an appropriate pulse (regarding e.g. spectrum), no spectral matching is applied by default. However, since the DNN tends to average the pulses slightly, which creates a slight low-pass effect, spectral matching can be set on by using parameter `USE_DNN_SPECMATCH`. `DNN_INPUT_NORMALIZED` normalizes the DNN input based on the values in the file `input.minmax`—if the normalization is done before training, this must be set on here as well. After the pulse generation, the synthesis process is identical to the pulse library method, i.e., the unvoiced and voiced excitations are combined and filtered with the vocal tract filter (and finally resynthesis with gain normalization).

Alternatively, the DNN pulse can be used as a target cost for selecting a pulse from a pulse library. This option can be used by setting parameter `USE_DNN_PULSELIB_SEL` to `true`. In order to use this option, an appropriate pulse library must be also defined in the config file. The combined DNN-based pulse library method is illustrated in Figure 8.

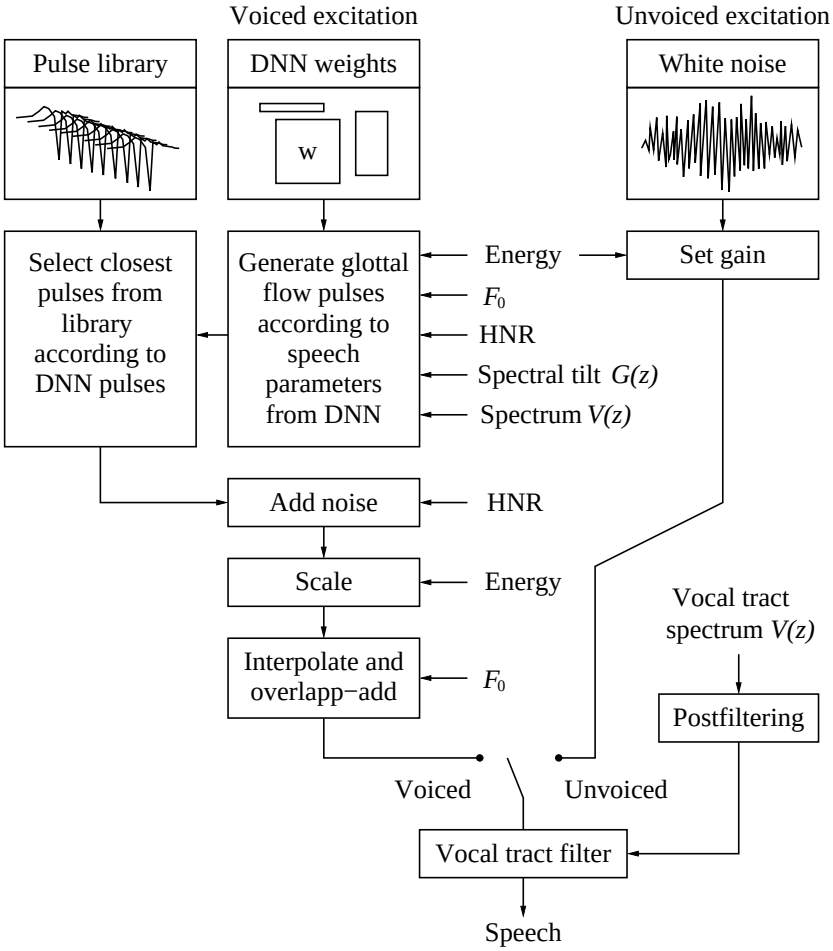

Figure 8: Illustration of synthesis with the combined DNN-based voice source modeling and pulse library selection technique.

6 Known limitations and issues

The maim focus of GlottHMM has been to test various methods and assumptions for statistical parametric speech synthesis. Thus it is not a complete product with all the flexibility and robustness. There are many limitations in the vocoder due to the special methodology and implementation, which will be shortly described here.

First of all, since the GlottHMM vocoder is based on glottal inverse filtering (GIF), it must be acknowledged that GIF in an extremely difficult inverse problem that is hard to solve accurately. Thus, the decomposition of speech into a glottal flow estimate and a vocal tract filter is an approximation, which in the best case works well for practical purposes. However, in many cases the decomposition is not that accurate, resulting in erroneous decomposition, thus producing a lot of noise to the estimated parameters. This can be avoided by using the more simple IAIF method or just ignoring GIF completely and using simple LPC inverse filtering instead.

The glottal inverse filtering also imposes various problems in vocoding. Since GIF is performed only for voiced frames, this makes the analysis of unvoiced and voiced parts of the speech different and thus there are two implications from that: (1) Accurate unvoiced/voiced decisions are especially important, (2) The change from voiced to unvoiced (and vice versa) imposes a discontinuity in the speech parameter trajectories as the analysis method is switched from one to another. This is not optimal behavior considering statistical modeling of the speech trajectories.

Also the GCI-estimation is not optimal since it is performed individually for each frame instead of performing a continuous analysis over the speech signal. Thus, there might be occasional errors in the GCI-estimates and thus spurious glottal flow pulses. The aim with GlottHMM has been to perform voice source analysis and to collect various glottal flow pulses from the speech material to construct data-based voice source models. Due to the frame-based analysis, the same glottal flow pulse may be estimated and saved several times in different frames. There are options in the configuration file to only save individual glottal flow pulses or to estimate only single glottal flow pulse per frame.

It is also important to note that there is no automatic polarity detection within GlottHMM analysis so users must ensure that the speech signal is of correct polarity. There are several methods publicly available for performing polarity detection, but a correct polarity can be also rather easily seen from the shape of the estimated glottal flow pulses (but this might require a bit of practice).

The f_0 estimation using the glottal flow based autocorrelation method usually performs rather well, but it requires that the f_0 limits are properly set. If the limits are too wide, there might easily be octave jumps up or down in the estimated f_0. Thus, it might be beneficial to use a separate method that estimates the rough f_0 limits for GlottHMM analysis. For normal read-aloud speech material, constant f_0 limits work usually well, but for expressive speech, more carefully selected limits for each utterance may be useful.

The HNR estimated during analysis stage is sensitive to perturbations in the actual f_0 and also to possible errors in f_0 estimation. Thus, the estimated HNR may often over-emphasize the amount of noise present in the speech frame. In order to alleviate this, a low-frequency limit `NOISE_LOW_FREQ_LIMIT` can be set to prevent too much noise especially in the lowest frequencies. The excess noise has especially adverse effect while using the single-pitch-period pulse due to the direct concatenation of the noise-added pulse.

In the pulse library method, the weights are rather arbitrarily defined by hand based on experience. It is also common that occasional erroneous/misclassified pulses are selected for constructing the excitation, thus leading in a bad or harsh quality of synthesis. This is the same problem as any unit selection based systems has, and there is no easy solution to prevent such phenomenon.

The DNN-based voice source modeling aims to map the speech features to the sample-wise glottal flow derivative waveform. However, the pulses contain also a lot of noise due to both speech production and possible errors in glottal inverse filtering and GCI detection. Thus, the DNN training

averages the sample-wise pulses so that they have a slight low-pass characteristics in comparison to natural pulses. This can be compensated using the spectral matching scheme `USE_DNN_SPECMATCH` or by using a separately constructed constant pre-emphasis (not implemented). Also, the DNN input parameter are at the moment hard-coded in Synthesis so that only specific features with specific dimensions can and must be used for input. At the moment, the parameters are (in order): [F0, Gain, HNR, LSFsource, LSF], and their dimensions are: [1 1 5 10 30], which results in the total length of the input vector of 47.

The Matlab scripts for creating pulse libraries and constructing DNN training data are not optimized for memory. With large amount of speech material, the computer may easily run out of memory. With 16 GB of memory, most tasks should be possible, but speech databased with more than one hour of speech material and large amount of glottal flow pulses (e.g., female voices) may cause problems.

References

Alku, P. Glottal wave analysis with pitch synchronous iterative adaptive inverse filtering. *Speech Commun.*, 11(2–3):109–118, 1992.

Alku, P., Bäckström, T., and Vilkman, E. Normalized amplitude quotient for parametrization of the glottal flow. *The Journal of the Acoustical Society of America*, 112(2):701–710, 2002.

Babacan, O., Drugman, T., Raitio, T., Erro, D., and Dutoit, T. Parametric representation for singing voice synthesis: A comparative evaluation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2583–2587, Florence, Italy, May 2014.

Bollepalli, B., Urbain, J., Raitio, T., Gustafson, J., and Cakmak, H. A comparative evaluation of vocoding techniques for HMM-based laughter synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 255–259, Florence, Italy, May 2014.

Keronen, S., Pohjalainen, J., Alku, P., and Kurimo, M. Noise robust feature extraction based on extended weighted linear prediction in LVCSR. In *Interspeech*, pages 1265–1268, 2011.

Ling, Z.-H., Wu, Y., Wang, Y.-P., Qin, L., and Wang, R.-H. USTC system for Blizzard Challenge 2006: an improved HMM-based speech synthesis method. In *Blizzard Challenge Workshop*, 2006.

Ma, C., Kamp, Y., and Willems, L. Robust signal selection for linear prediction analysis of voiced speech. *Speech Comm.*, 12(1):69–81, 1993.

Magi, C., Pohjalainen, J., Bäckström, T., and Alku, P. Stabilised weighted linear prediction. *Speech Comm.*, 51(5):401–411, May 2009.

Pohjalainen, J., Saeidi, R., Kinnunen, T., and Alku, P. Extended weighted linear prediction (XLP) analysis of speech and its application to speaker verification in adverse conditions. In *Interspeech*, pages 1477–1480, 2010.

Raitio, T., Suni, A., Vainio, M., Alku, P., and Nurminen, J. Method, apparatus and computer program product for providing real glottal pulses in HMM-based text-to-speech synthesis. Patent, US8386256 B2, CA2724753A1, CN102047321A, EP2279507A1, EP2279507A4, US20090299747, WO2009144368A1), filing date: May 29, 2009, publication date: February 26, 2013.

Raitio, T., Suni, A., Pulakka, H., Vainio, M., and Alku, P. HMM-based Finnish text-to-speech system utilizing glottal inverse filtering. In *Proc. Interspeech*, pages 1881–1884, 2008.

Raitio, T., Suni, A., Pulakka, H., Vainio, M., and Alku, P. Comparison of formant enhancement methods for HMM-based speech synthesis. In *SSW7*, pages 334–339, Sep. 2010.

Raitio, T., Suni, A., Pulakka, H., Vainio, M., and Alku, P. Utilizing glottal source pulse library for generating improved excitation signal for HMM-based speech synthesis. In *Proc. ICASSP*, pages 4564–4567, 2011a.

Raitio, T., Suni, A., Yamagishi, J., Pulakka, H., Nurminen, J., Vainio, M., and Alku, P. HMM-based speech synthesis utilizing glottal inverse filtering. *IEEE Trans. on Audio, Speech and Lang. Proc.*, 19(1):153–165, Jan. 2011b.

Raitio, T. Hidden Markov model based Finnish text-to-speech system utilizing glottal inverse filtering. Master's thesis, Department of Signal Processing and Acoustics, Helsinki University of Technology, Finland, 2008.

Raitio, T., Suni, A., Vainio, M., and Alku, P. Wideband parametric speech synthesis using warped linear prediction. In *Interspeech*, 2012.

Raitio, T., Suni, A., Vainio, M., and Alku, P. Comparing glottal-flow-excited statistical parametric speech synthesis methods. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7830–7834, 2013.

Raitio, T., Lu, H., Kane, J., Suni, A., Vainio, M., King, S., and Alku, P. Voice source modelling using deep neural networks for statistical parametric speech synthesis. In *22nd European Signal Processing Conference (EUSIPCO)*, Lisbon, Portugal, September 2014a.

Raitio, T., Suni, A., Juvela, L., Vainio, M., and Alku, P. Deep neural network based trainable voice source model for synthesis of speech with varying vocal effort. In *Proc. of Interspeech*, pages 1969–1973, 2014b.

Raitio, T., Suni, A., Vainio, M., and Alku, P. Synthesis and perception of breathy, normal, and Lombard speech in the presence of noise. *Computer Speech & Language*, 28(2):648–664, March 2014c.

Salakhutdinov, R. and Hinton, G. Training a deep autoencoder or a classifier on MNIST digits. http://www.cs.toronto.edu/ hinton/MatlabForSciencePaper.html, 2015.

Simple4All. Ossian. http://simple4all.org/product/ossian/, 2015.

Suni, A., Raitio, T., Vainio, M., and Alku, P. The GlottHMM speech synthesis entry for Blizzard Challenge 2010. In *The Blizzard Challenge 2010 workshop*, 2010. http://festvox.org/blizzard.

Suni, A., Raitio, T., Vainio, M., and Alku, P. The GlottHMM entry for Blizzard Challenge 2011: Utilizing source unit selection in HMM-based speech synthesis for improved excitation generation. In *The Blizzard Challenge 2011 Workshop*, 2011. http://festvox.org/blizzard.

Suni, A., Raitio, T., Vainio, M., and Alku, P. The GlottHMM entry for Blizzard Challenge 2012 – Hybrid approach. In *Blizzard Challenge 2012 Workshop*, Portland, Oregon, September 2012.

Suni, A., Karhila, R., Raitio, T., Kurimo, M., Vainio, M., and Alku, P. Lombard modified text-to-speech synthesis for improved intelligibility: Submission for the Hurricane Challenge 2013. In *Interspeech*, pages 3562–3566, Lyon, France, August 2013.

Suni, A., Raitio, T., Gowda, D., Karhila, R., Gibson, M., and Watts, O. The Simple4All entry to the Blizzard Challenge 2014. In *Blizzard Challenge 2014 Workshop*, 2014.

A Description of configuration file parameters

In the following list, all the configuration file parameters and their purpose are described. The default values and/or the recommended range/possible options are shown for each parameter. Note again, that integer parameters must be defined without decimals (e.g. sampling frequency is 16000), and decimal numbers must include the decimal point (e.g. frame length in milliseconds is 25.0).

Analysis and Synthesis: Common parameters:

`SAMPLING_FREQUENCY:` Sampling frequency in Hz (16000)

`FRAME_LENGTH:` Analysis frame length in milliseconds (25.0)

`UNVOICED_FRAME_LENGTH:` Analysis frame length for unvoiced segments in milliseconds (20.0)

`F0_FRAME_LENGTH:` Analysis frame length for $f0$ evaluation in milliseconds (45.0)

`FRAME_SHIFT:` Analysis frame shift in milliseconds (5.0)

`LPC_ORDER:` Order of the spectral model of the vocal tract (30)

`LPC_ORDER_SOURCE:` Order of the spectral model of the voice source (20)

`WARPING_VT:` Warping coefficient for vocal tract model (0.0). Default value produces linear spectral resolution; positive values produce finer resolution in the low frequencies; negative values produce finer resolution in the high frequencies. The value must be in the range [-1,1].

`WARPING_GL:` Warping coefficient for voice source spectral model (0.0). Default value produces linear spectral resolution; positive values produce finer resolution in the low frequencies; negative values produce finer resolution in the high frequencies. The value must be in the range [-1,1].

`HNR_CHANNELS:` Number of harmonic-to-noise ratio (HNR) frequency channels (5). Channels are distributed according to the equivalent rectangular bandwidth (ERB) scale in the frequency domain.

`NUMBER_OF_HARMONICS:` Number of harmonics to be modeled in the harmonics parameter (10).

`SEPARATE_VU_SPECTRUM:` Extract and use separate spectra for voiced and unvoiced segments (false).

`DIFFERENTIAL_LSF:` Use differential line spectral frequencies (LSFs) (false). Differential LSFs improves the sharpness of the formants, but the formant positions are less accurate.

`UNVOICED_PRE_EMPHASIS:` Unvoiced frames are pre-emphasized during spectral analysis in order to alleviate the discontinuity between IAIF and LPC analysis for voiced and unvoiced frames, respectively. This, however, may introduce artefacts in synthesis as the unvoiced excitation need to be de-emphasized (false).

`LOG_F0:` Convert F0 parameter to natural logarithmic scale.

`DATA_FORMAT:` Data format for writing speech features ("ASCII"). Reading and writing binary data ("BINARY") is faster and saves some space.

Noise reduction:

`NOISE_REDUCTION_ANALYSIS:` Use noise reduction in Analysis.

`NOISE_REDUCTION_SYNTHESIS:` Use noise reduction in Synthesis.

`NOISE_REDUCTION_LIMIT_DB:` Limit for noise reduction in dB; values of gain lower than the limit will be reduced (0.0, depends on speech material).

`NOISE_REDUCTION_DB:` Amount of noise reduction in dB (30.0).

Analysis: General parameters:

`PITCH_SYNCHRONOUS_ANALYSIS:` Option to use pitch-synchronous analysis (false).

`INVERT_SIGNAL:` Set this true if the polarity of the speech signal is inverted (false). This is especially important when using pulse library.

`HP_FILTERING:` Use high-pass filtering in order to remove possible low-frequency ripple in the glottal inverse filtering (true).

`HPFILTER_FILENAME:` Define high-pass filter path and file name (e.g., "/home/syn/hp_16khz"). The filter coefficients must in ASCII format, one coefficient in each line. The filter must be a FIR filter with a preferred cut-off frequency around 30–70 Hz.

Analysis: Parameters for F0 estimation:

`F0_MIN:` Minimum fundamental frequency (40.0).

`F0_MAX:` Maximum fundamental frequency (400.0).

`VOICING_THRESHOLD:` Voicing threshold with respect to gain in the low-frequency band (0–1000 Hz) (100.0). Speech segments that have very low gain in the low frequencies are classified as unvoiced according to this relative measure. Increasing the value will results in more frames to be classified as unvoiced.

`ZCR_THRESHOLD:` Zero-crossings threshold (120.0). Speech segments that have more zero crossings than the threshold value are classified as unvoiced according. Decreasing the ZCR threshold will result in more frames to be classified as unvoiced.

`USE_F0_POSTPROCESSING:` Post-processing of fundamental frequency vector (false). Processing refines the $f0$ vector by median filtering, filling small gaps, and estimating weighted linear $f0$ estimates in the case of discontinuities in the vector.

`RELATIVE_F0_THRESHOLD:` Discontinuity measure in $f0$ post-processing (0.5). Relative $f0$ jumps greater that this value will be considered unnatural and will be fixed.

`F0_CHECK_RANGE:` Range (in frames) within which the $f0$ post-processing will operate when estimating and correcting possible incorrect $f0$ values (10).

`USE_EXTERNAL_F0:` Use external $f0$ file in Analysis (false). All evaluations that require fundamental frequency are thus based on the given values. The name of the external $f0$ file is indicated in the next parameter.

`EXTERNAL_F0_FILENAME:` Name of the external $f0$ file.

Analysis: Parameters for extracting pulse libraries:

`MAX_NUMBER_OF_PULSES:` Maximum number of pulses to be stored per speech file (10000). This is just to save memory, value may be increased with long speech files.

`PULSEMAXLEN:` Maximum length of the two-period library pulse in milliseconds (45.0).

`RESAMPLED_PULSELEN:` Length of the resampled library pulse in milliseconds (10.0).

`WAVEFORM_SAMPLES:` Number of the samples which describe the shape of the library pulse (10).

`MAX_PULSE_LEN_DIFF:` Maximum relative difference between pulse length and the estimated T_0 (0.05). If the relative difference is greater that the defined value, the pulse is discarded. The goal is to discard possibly misaligned pulse segments due to failure in the detection of glottal closure instant (GCI).

`EXTRACT_ONLY_UNIQUE_PULSES:` Due to a long frame and small frame shift, the same pulse occurs many times in several different frames. By setting this true, duplicates are not produced (false).

`EXTRACT_ONE_PULSE_PER_FRAME:` This setting extracts only a single pulse per frame (false). If several pulses are found in a frame, the one closest to the mean in terms of MSE of the pulses is selected.

Analysis: Parameters for spectral modeling:

`USE_IAIF:` Use iterative adaptive inverse filtering (IAIF) (true).

`LPC_ORDER_GL_IAIF:` Order of the LPC analysis for the voice source inside IAIF (8).

`USE_MOD_IAIF:` Use modified version of the IAIF method (true). This simpler glottal inverse filtering method gives less varying decomposition, but may not be as accurate as full IAIF.

`LP_METHOD:` Select between different spectral modeling methods, linear prediction (LPC), weighted linear prediction (WLP), or extended linear prediction (XLP) ("LPC"/"WLP"/"XLP").

`LP_STABILIZED:` Stabilize spectral models if WLP or XLP is used (false).

`LP_WEIGHTING:` IF WLP is used, select between short time energy (STE) weight and glottal closure instant (GCI) weight ("STE"/"GCI").

`FORMANT_PRE_ENH_METHOD:` Select formant enhancement method for pre-enhancement. There are two methods, LSF-based and LPC-based. Usually pre-enhancement is not used, and enhancement is used in synthesis stage ("NONE"/"LSF"/"LPC").

`FORMANT_PRE_ENH_COEFF:` Formant enhancement coefficient (0.4 for LPC, 0.6 for LSF). The effect gets stronger as the value is decreased. Range: [0,1].

`FORMANT_ENH_LPC_DELTA:` Tuning parameter for the LPC-based formant enhancement method (20.0). Defines the width of the area around formants that is left unmodified in the power spectrum.

Analysis: Select parameters to be extracted:

`EXTRACT_F0`: Extract fundamental frequency (true).

`EXTRACT_GAIN`: Extract gain of the speech signal (true).

`EXTRACT_LSF`: Extract vocal tract spectrum LSFs (true).

`EXTRACT_LSFSOURCE`: Extract voice source spectrum LSFs (true).

`EXTRACT_HNR`: Extract harmonic-to-noise ratio of the voice source (true).

`EXTRACT_HARMONICS`: Extract the magnitude differences between N first harmonics of the voice source (true).

`EXTRACT_H1H2`: Extract the magnitude difference between the first and the second harmonics of the voice source (true).

`EXTRACT_NAQ`: Extract the normalized amplitude quotient (NAQ) of the glottal pulses (true).

`EXTRACT_WAVEFORM`: Extract the approximate shape of the glottal waveform (true).

`EXTRACT_INFOFILE`: Write infofile that describes the used analysis parameters (true).

`EXTRACT_PULSELIB`: Extract all parameters needed to construct a pulse library (false). This is only required when constructing a pulse library.

`EXTRACT_FFT_SPECTRA`: Extract FFT of the vocal tract spectrum and the glottal flow spectrum and write to files (false). This is only additional option for speech analysis.

`EXTRACT_SOURCE`: Extract the estimated glottal flow signal to file and to wav file (false). This is only additional option for speech analysis.

Synthesis: General parameters:

`SYNTHESIZE_MULTIPLE_FILES`: Switch for synthesizing multiple files at once (false).

`SYNTHESIS_LIST`: Full path and name of a synthesis list, e.g., "/home/syn/synlist".

`USE_HMM`: Select between direct analysis-synthesis (false) and HMM modeling (true). For direct analysis-synthesis some smoothing of the parameters is applied in order to produce more natural sounding speech.

Synthesis: Single pulse:

`GLOTTAL_PULSE_NAME`: Full path and name of a single pulse, e.g., "/home/syn/gpulse".

`TWO_PITCH_PERIOD_DIFF_PULSE`: Set false if the glottal flow pulse defined above is single period glottal flow (not derivative). If the pulse defined above is a two-pitch period glottal flow derivative pulse, extracted from a pulse library, set this true.

Synthesis: DNN pulse generation:

`USE_DNN_PULSEGEN:` Use DNN-based voice source modeling (false)

`USE_DNN_PULSELIB_SEL:` Use DNN-based voice source modeling for generating a target cost for pulse library based synthesis (false).

`USE_DNN_SPECMATCH:` Use spectral matching of the DNN-pulse (false).

`DNN_WEIGHT_PATH:` = Path to the DNN weights, e.g., "/home/syn/DNNTrain/dnnw/"

`DNN_WEIGHT_DIMS:` = Dimensions of the DNN weight matrices. For example, if input speech feature vector has 47 dimensions and there are 2 hidden layers with 200 units in each, and the final pulse length is 400 samples, the dimensions are [48, 200, 201, 200, 201, 400]. Additional 1 in the first (input) dimension of each matrix is due to the bias weight.

`DNN_INPUT_NORMALIZED:` Set true if normalization is used before DNN training (true). The min and max values are read from the trained DNN folder.

`DNN_NUMBER_OF_STACKED_FRAMES:` If the input parameters for the DNN are stacked from several frames, define the number of stacked frames here. The default is 1 (no stacking) and only small scale experiments have been done to find out if stacking is useful at all.

Synthesis: Pulse library method:

`USE_PULSE_LIBRARY:` Switch for using a pulse library (false).

`PULSE_LIBRARY_NAME:` Full path and name of a pulse library, e.g., "home/syn/pulse_libraries/pulselib1/pulselib1".

`NORMALIZE_PULSELIB:` The means of the pulse library parameters are normalized according to synthesis parameters (true).

`USE_PULSE_INTERPOLATION:` Interpolate pulse library pulses according to $f0$ (false/true). However, this is not mandatory since pulses with approximately correct $f0$ are usually selected and overlap-added according to $f0$.

`AVERAGE_N_ADJACENT_PULSES:` Average adjacent pulse library pulses in order to get smoother excitation (also creates a low-pass effect). The number of averaged adjacent pulses is defined by this parameter, 0 sets off this feature (0).

`ADD_NOISE_PULSELIB:` Add noise to pulse library pulses according to the HNR measure (true/false). This is not mandatory since pulses with approximately correct HNR value are usually selected.

`USE_PULSE_CLUSTERING:` Switch for using pulse clustering in synthesis. This requires clustering of the pulse library according to decision trees (false).

`MAX_PULSES_IN_CLUSTER:` Define the maximum number of pulses per cluster, if pulse clustering is used (2000).

`NUMBER_OF_PULSE_CANDIDATES:` Define the number of best matching pulse library candidates for each time step in the Viterbi search (200).

`PULSE_ERROR_BIAS:` Set bias in order to prevent the same pulse to be selected to the excitation more than once in a row (0.3). The selection of the same library pulse multiple times in a row may cause audible artifacts as the excitation is too periodic.

`CONCATENATION_COST:` Set concatenation cost for selecting the library pulses (1.0).

`TARGET_COST`: Set target cost for selecting the library pulses (1.0).

`PARAMETER_WEIGHTS`: Set parameter weights for for selecting the library pulses. A vector consisting of the parameter weights of each parameter is defined as [LSF SRC HARM HNR GAIN F0 WAV H1H2 NAQ], e.g., [0.0, 2.0, 0.0, 2.0, 3.0, 5.0, 1.0, 1.0, 1.0].

Synthesis: Select used parameters:

Select parameters used in Synthesis. Some parameters are required only in a specific configurations, such as in pulse library method. F0, Gain, and LSFs are always required and thus they are not in this list.

`USE_LSFSOURCE`: Use voice source spectrum (true).

`USE_HNR`: Use harmonic-to-noise ratio (true).

`USE_HARMONICS`: Use harmonics (false).

`USE_H1H2`: Use the magnitude difference of the first and the second harmonic (false).

`USE_NAQ`: Use normalized amplitude quotient (NAQ) (false).

`USE_WAVEFORM`: Use downsampled waveform (false).

Synthesis: Set level and band of voiced noise:

`NOISE_GAIN_VOICED`: The the level of added noise in voiced sections (0.5).

`NOISE_LOW_FREQ_LIMIT`: Set low-frequency limit for the added noise (2000.0). The value is decimal number in Hz between [0,FS/2]. This prevents artifacts due to large low-frequency fluctuations in the single pulse method. For other voice source modeling methods, 0.0 Hz can be used.

`HNR_COMPENSATION` Re-estimate HNR and compensate HNR values accordingly in synthesis (false).

Synthesis: Smoothing of parameters for analysis-synthesis:

Smoothing lengths (in frames) of the parameter vectors/matrices for analysis-synthesis. All smoothing is set off if parameters generated from HMMs are used.

`LSF_SMOOTH_LEN`: LSF smoothing length (5).

`LSFSOURCE_SMOOTH_LEN`: Voice source spectrum smoothing length (3).

`GAIN_SMOOTH_LEN`: Gain smoothing length (5).

`HNR_SMOOTH_LEN`: Harmonic-to-noise ratio (HNR) smoothing length (15).

`HARMONICS_SMOOTH_LEN`: Harmonics smoothing length (5).

Synthesis: Gain related parameters:

`GAIN_UNVOICED`: Set gain for unvoiced part (1.0).

`NORM_GAIN_SMOOTH_V_LEN`: Smoothing length of the voiced part of the gain normalization algorithm (0).

`NORM_GAIN_SMOOTH_UV_LEN`: Smoothing length of the unvoiced part of the gain normalization algorithm (0).

`GAIN_VOICED_FRAME_LENGTH`: Voiced frame length of the gain normalization algorithm in milliseconds (25.0).

`GAIN_UNVOICED_FRAME_LENGTH`: Unvoiced frame length of the gain normalization algorithm in milliseconds (25.0).

Synthesis: Postfiltering:

`POSTFILTER_METHOD`: Select formant enhancement method for post-filtering. There are two methods, LSF-based and LPC-based. Select between "NONE"/"LSF"/"LPC".

`POSTFILTER_COEFFICIENT`: Formant enhancement strength (common parameter for both methods) (0.5). The effect gets stronger as the value of alpha is decreased. Range: [0,1].

Synthesis: Utils:

`USE_HARMONIC_MODIFICATION`: If single pulse technique is used, a harmonic modification procedure can be used to correctly match the low-frequency spectrum of the voice source (false).

`HP_FILTER_F0`: Adaptive low-pass filtering of the synthesized speech signal below current $f0$ value (false).

`FILTER_UPDATE_INTERVAL_VT`: Vocal tract filter update interval in milliseconds (0.3).

`FILTER_UPDATE_INTERVAL_GL`: Spectral matching filter (voice source) update interval in milliseconds (0.05).

`WRITE_EXCITATION_TO_WAV`: Write generated excitation signal to a wav file (false).

Synthesis: Voice adaptation:

`PITCH`: Modify pitch relative to the original pitch (1.0).

`SPEED`: Modify speed relative to the original speed (1.0).

`JITTER`: Add jitter to consecutive pulses (relative to pulse length) (0.0).

`ADAPT_TO_PULSELIB`: Adapt synthesis parameters according to pulse library parameters by normalizing the mean of the parameters. This will have an effect that the synthetic voice will sound more like the voice in the pulse library (false).

`ADAPT_COEFF`: Coefficient for changing the magnitude (and direction) of the adaptation of the synthesis parameters by the pulse library parameters (1.0).

`USE_PULSELIB_LSF:` Replace vocal tract LSFs with the closest LSFs from the pulse library, apply smoothing (false). This will have an effect that the synthetic voice will sound more like the voice in the pulse library (false). This may also create severe artefacts if appropriate LSF-vectors are not found.

`NOISE_ROBUST_SPEECH:` Create Lombard style speech. The quality of the resulting speech is degraded, but the intelligibility in the presence of (low-frequency) noise is increased.

Synthesis: Pulse library PCA:

`USE_PULSELIB_PCA:` Use PCA-based reconstruction of the pulse (false).

`PCA_ORDER:` Define PCA order (12).

`PCA_ORDER_SYNTHESIS:` Define the number of PC weights and components to be used for reconstructing pulses (0). This can be lower than the original number of PC vectors. If set to 0, only the mean pulse will be used in PCA.

`PCA_SPECTRAL_MATCHING:` Use spectral matching in addition to PCA-based method (false).

`PCA_PULSE_LENGTH:` Define the length of the PCA pulse (800).