An Open Source Speech Synthesis Frontend for HTS

Markus Toman and Michael Pucher

FTW Telecommunications Research Center Vienna
Donau-City-Straße 1, A-1220 Vienna, Austria
http://www.ftw.at
{toman,pucher}@ftw.at

Abstract. This paper describes a software framework for HMM-based speech synthesis that we have developed and released to the public. The framework is compatible to the well-known HTS toolkit by incorporating hts_engine and Flite. It enables HTS voices to be used as Microsoft Windows system voices and to be integrated into Android and iOS apps. Non-English languages are supported through the capability to load Festival format pronunciation dictionaries and letter to sound rules. The release also includes an Austrian German voice model of a male, professional speaker recorded in studio quality as well as pronunciation dictionary, letter to sound rules and basic text preprocessing procedures for Austrian German. The framework is available under an MIT-style license.

Keywords: speech synthesis, HTS, hidden markov model, frontend, software

1 Introduction

Hidden-Markov Model (HMM) based speech synthesis provides a methodology for flexible speech synthesis while keeping a low memory footprint [1]. It also enables speaker adaptation from average voice models, allowing the creation of new voice models from sparse voice data [2], as well as techniques like interpolation [3][4] and transformation [5][6] of voices. A well-known toolkit for creating HMM-based voice models is HTS [7]. Separate software toolkits are available to actually synthesize speech waveforms from HTS models. A popular, freely available framework is hts_engine [8]. Speech synthesis frontends on the other hand provide means for analyzing and processing text, producing the necessary input for the synthesizer. In HTS this input is a set of labels where usually each label represents a single phone and contextual information, including surrounding phones, position in utterance, prosodic information etc. While not exclusively being frontends and not specifically targeted for HTS, popular choices are Festival [9] or Flite [10]. Festival is a complex software framework for building speech synthesis systems focusing Unix-based operating systems. Flite was built as a lightweight alternative to Festival with low memory footprint and fast runtime in mind.

Our main goal when creating the presented frontend framework was to easily allow HTS voices to be used with the Speech Application Programming Interface 5 (SAPI5). This allows the framework to be installed on different versions of the Microsoft Windows operation system as speech synthesis engine, making HTS voice models available as system voices to applications like screen readers, e-book creators etc. The second goal was simple integration of new languages and phone sets. The third goal was

portability to mobile devices. The framework is available under an MIT-style license at `http://m-toman.github.io/SALB/`.

Flite has been adapted for HTS in the Flite+hts_engine software [8] and due to its small and portable nature it seemed like a good fit to our requirements. The structure of Flite makes integrating new languages rather cumbersome. [1] Therefore our framework integrates Flite for text analysis of English while additionally providing a second text analysis module that can utilize Festival style pronunciation dictionaries and letter to sound trees. Text preprocessing tasks (e.g. number and date conversion) can be added to the module in C++. Adding a completely new text processing module is also possible. The framework includes hts_engine for speech waveform synthesis and can be extended by other synthesizers. The framework also includes a free voice model of a male, professional speaker for Standard Austrian German.

2 Voice model "Leo"

Category	Phones (IPA)
Vowels (monoph.)	ɑ ɑː ɒ ɒː ɐ e ɛ eː i ɪ iː ɔ o oː øː æː ɐ œ œː ə u ʊ uː ʏ y yː
Vowels (monoph.) nasalized	ɒ̃ː ɔ̃ː ɑ̃ː œ̃ː
Diphthongs	aɪ̯ ɒːɐ̯ ɑːɐ̯ ɒɪ̯ ɑʊ̯ ɛɐ̯ ɛːɐ̯ ɪɐ̯ iːɐ̯ ɔɐ̯ ɔːɐ̯ oːɐ̯ ɔʏ̯ øːɐ̯ œɐ̯ ʊɐ̯ uːɐ̯ ʊːɐ̯ ʏɐ̯ yːɐ̯
Plosives (stops)	b̥ d̥ g̊ k ʔ p t
Nasal stops	m n ŋ
Fricatives	ç x f h s ʃ v z ʒ
Approximants	j
Trill	r
Lateral approx.	l

Table 1. Phone set used for Austrian German voice "Leo".

With the framework we provide a free voice model of a male, professional speaker for Standard Austrian German called "Leo". The model is built from 3,700 utterances recorded in studio quality using a phonetically balanced corpus. The phone set used in the voice can be seen in Table 1. A pronunciation dictionary with 14,000 entries, letter to sound rules and procedures for number conversion are also included.

[1] We have published instructions on adding a new language to Flite: http://sourceforge.net/p/at-flite/wiki/AddingNewLanguage/

3 Framework architecture

The general architecture of the SALB framework is shown in Figure 1. Frontend modules provide means to communicate with the user or other applications through different channels. For example the user can directly trigger speech synthesis tasks by the Command-Line-Interface (CLI), other applications can use the SAPI5 interface to use the framework in a uniformly manner together with other synthesis engines that implement SAPI5. The frontend modules use the C++ Application Programming Interface (API) of the core module `manager`, which in turn coordinates the backend modules, performing text processing and the actual speech synthesis task. The C++ API can be used directly to embed the framework in other applications.

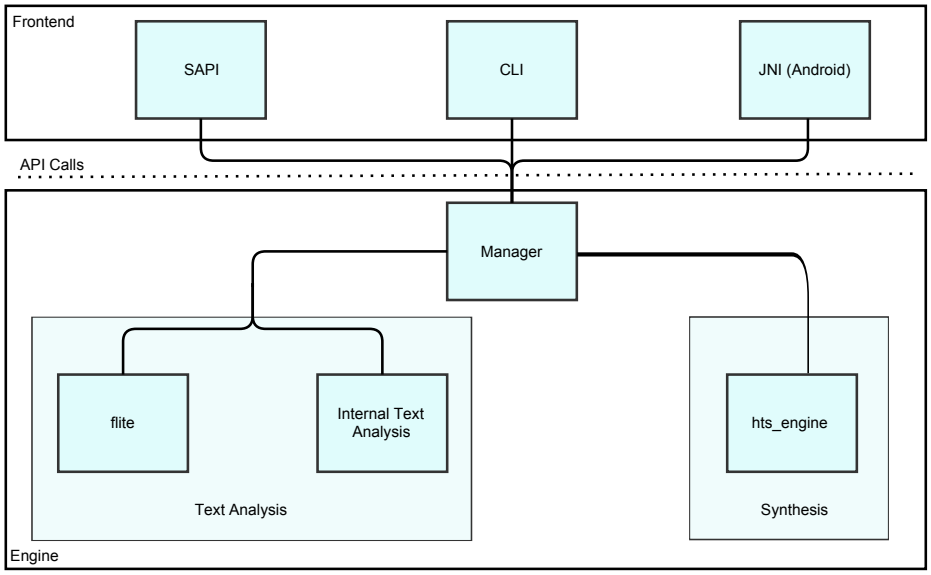

Fig. 1. General framework architecture

3.1 Manager module and API

The core of the framework is the manager module which provides a uniform API for frontend modules or other applications. It provides abstractions for different elements of the speech synthesis process. This API is provided by the `TTSManager` class. A `TextFragment` is a piece of text in a given language. Each `TextFragment` has `FragmentProperties` associated which control synthesis parameters (e.g. voice, speaking rate) for this text fragment. Multiple `TextFragment` objects can form a `Text` object. This can be used to synthesize a text consisting of fragments with different synthesis parameters (e.g. a text read by different voices). A `Text` or `TextFragment`

object with associated `FragmentProperties` can be passed to a `TTSManager` object which executes the speech synthesis process and returns a `TTSResult` object. This process is depicted in Figure 2. The `TTSManager` object first selects an adequate `TextAnalyzer` object based on the value in `FragmentProperties` specifying the text analyzer to use or a value specifying the language of the text. The `TextFragment` is then passed to the `TextAnalyzer` object which returns a series of `Label` objects in a container called `Labels`. A `Label` represents the basic unit for synthesis, usually being a single phone with contextual information. The `TTSManager` then selects an adequate `Synthesizer` implementation, again based on `FragmentProperties`, and passes the `Label` to it. The `Label` class is responsible for providing the desired format. Currently the only available format is the HTS label format, which can easily be stored as special character delimited string. The `Synthesizer` returns a `TTSResult` object containing the synthesized waveform as `vector` of discrete samples as well as meta information.

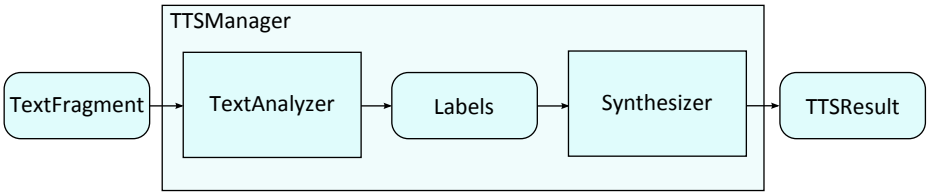

Fig. 2. Data flow in synthesis process

3.2 Frontend interfaces

The following sections describe the frontend interfaces currently included in the framework.

Speech Application Programming Interface (SAPI) The framework provides a frontend implementing the SAPI5. This allows the registration of the framework as speech synthesis engine on Microsoft Windows platforms, therefore enabling HTS voices to be registered as system voices. We provide a Microsoft Visual Studio project [11] to compile a SAPI-enabled dynamic link library for 32-bit and 64-bit systems that can be registered with the operating system. Subsequently, voices can be added using the `register_voice` tool that comes with the framework. Lastly a script, that allows to create installer packages that install or uninstall the engine and associated voice models, is bundled with the framework.

Command Line Interface The distribution also contains a simple command line tool which, given all necessary input for the text analysis and synthesis modules, produces RIFF wav output files from textual input.

Android Integration in Android apps is possible through the Java Native Interface (JNI) and the Android Native Development Kit (NDK) [12]. The framework comes with Android `make` files, a JNI wrapper and a Java class for demonstration purposes.

3.3 Text analysis modules

The following sections describe the text analysis modules currently included in the framework.

Flite For converting English text to a series of labels for synthesis, we integrated Flite. The class `FliteTextAnalyzer` (derived from `TextAnalyzer`) is a wrapper converting input and output data for and from Flite.

Internal text analyzer The distribution also comes with an internal text analysis module `InternalTextAnalyzer` (derived from `TextAnalyzer`). This module reads a specific rules file consisting of a pronunciation dictionary and letter to sound rules in Festival style format. Preprocessing of text (e.g. for numbers and dates) can be added by extending the `Normalizer` class which delegates to different implementations based on the chosen language. We provide a simple `Normalizer` for Austrian German (`AustrianGermanNormalizer`) as well as a comprehensive rules file which can be used as an example to integrate new languages. This module uses an utterance structure which consists of the classes `Phrase`, `Word`, `Syllable` and `Phone`. The `PhraseIterator` class is used to navigate in this structure to build the resulting `Label` object.

3.4 Synthesis modules

The following section describes the currently available synthesis module in the framework.

hts_engine This module provides a wrapper around hts_engine and is implemented by the class `HTSEngineSynthesizer` (derived from `Synthesizer`). The `Label` objects provide strings in HTS label format which are the input to hts_engine. The resulting waveform is then converted and encapsulated in a `TTSResult` object and returned to the `TTSManager` and subsequently to the caller. We have changed the algorithm for changing the speaking rate to linear scaling due to the results in [4].

4 Adding new languages

One main goal when developing the framework was the possibility to easily integrate voice models of other languages than English. As literature aiding this process is scarce, we present some basic guidelines in the following sections.

4.1 Gathering data

Before creating a recording script, a defined phone set is needed. These already exist for many languages. If not, the inventory of all relevant phones of a language should be defined in cooperation with phoneticians. One possibility is to gather conversational speech data in the target language and then produce manual transcriptions. The granularity of the transcription is very important and has a direct impact on the quality of the final voice models. For example, diphthongs can be modeled as separate phone symbols or split into two symbols. When a phone set is defined, recording scripts can be generated either through manual transcription or by using orthographic text (e.g. from newspapers) and a letter to sound system. The recording script should contain all phones in as many triphone contexts, better quinphone contexts, as possible. In any case each phone should occur multiple times (preferably at least 10 times, considering that a set set will also be split off the training data). Given a data set of phonetically transcribed sentences, algorithms to solve the set cover problem can be employed to produce a final recording script. From this data set, a test set can be selected by the same procedure (e.g. select the smallest set that contains each phone at least 2 times). Speakers should be recorded in studio setting and with neutral prosody. The output of this step is a phone set, recording scripts and a corpus of recorded utterances in waveforms and transcriptions.

4.2 Integration

The internal text analyzer reads rules for text processing from an input stream. These rules consist of a lexicon and a letter to sound tree. A word is first looked up in the lexicon. If it can not be found there, the letter to sound rules are used to create the phonetic transcription. A voice model and the text processing rules are sufficient for bassic speech synthesis and building SAPI5 voice packages using the framework. Extending the source code is necessary if more sophisticated text processing is required, the C++ class `AustrianGermanNormalizer` can be used as an example for this. The method `InternalTextAnalyzer::TextFragmentToPhrase` can be adapted to implement alternatives to the Festival lexica and letter to sound rules.

Lexicon The lexicon (or pronunciation dictionary) part of the text rules is a set of mappings from orthography to phonetics. The framework uses Festival style lexica in Scheme syntax.[2] A lexicon can be derived from the data corpus used for the recording or from publicly available pronunciation dictionaries.

Letter-to-sound rules The internal text analyzer is able to read Festival style letter to sound rules.[3] A method for building letter to sound rules from an existing lexicon can be found in [13]. For languages with an orthography very close to the phonetics, the letter to sound rules can also be hand-crafted.

[2] Festival lexicon definition: http://www.cstr.ed.ac.uk/projects/festival/manual/festival_13.html
[3] Festival LTS rules: http://www.cstr.ed.ac.uk/projects/festival/manual/festival_13.html#SEC43

4.3 Training voice models

Voice models to be used with the SALB framework can be trained using the HTS toolkit. Available demonstration scripts for speaker dependent voices can be adapted for this purpose. When using the demonstration scripts, it is necessary to replace raw sound files, labels and question files. Label files can be produced using the SALB framework once at least a lexicon containing all words from the training set is available. The question files contain questions used in the decision tree based clustering [14]. An illustrative example of the usage of the phone questions in a decision tree can be seen in Figure 3. A minimal question file should at least contain all phone identity questions (e.g. for a phone "oh" and quinphone models, at least the following questions should be defined: "`LL-oh`", "`L-oh`", "`C-oh`", "`R-oh`", "`RR-oh`"). When all parameters (e.g. sampling rate) have been set, the training process can be started to produce an "`htsvoice`" model file that can be used with the SALB framework.

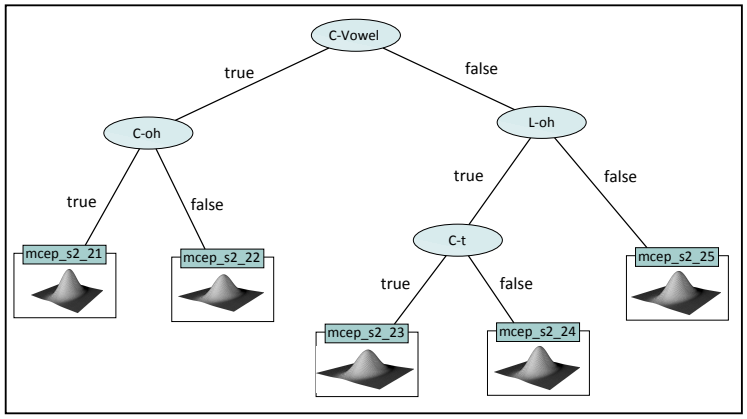

Fig. 3. Questions in decision tree based clustering

5 Conclusion

We have presented an open source speech synthesis framework, a software that bridges existing tools for HTS-based synthesis like hts_engine and Flite with SAPI5 to enable HTS voices to be used as Windows system voices. It allows to load and use Festival style lexica and letter to sound rules for easy integration of new languages. The C++ API allows embedding in other applications as well as Android and iOS apps. Also included is a voice model of a male, professional Standard Austrian German speaker with lexicon and letter to sound rules. We have also given a brief tutorial on how to train voice models for further languages and add use them with the SALB framework.

Acknowledgements

This work was supported by the Austrian Science Fund (FWF): P22890-N23. The Competence Center FTW Forschungszentrum Telekommunikation Wien GmbH is funded within the program COMET - Competence Centers for Excellent Technologies by BMVIT, BMWA, and the City of Vienna. The COMET program is managed by the FFG.

References

1. Zen, H., Tokuda, K., Black, A.W.: Statistical parametric speech synthesis. Speech Communication **51**(11) (2009) 1039–1064
2. Yamagishi, J., Kobayashi, T.: Average-voice-based speech synthesis using HSMM-based speaker adaptation and adaptive training. IEICE Transactions on Information and Systems **E90-D**(2) (February 2007) 533–543
3. Pucher, M., Schabus, D., Yamagishi, J., Neubarth, F., Strom, V.: Modeling and interpolation of Austrian German and Viennese dialect in HMM-based speech synthesis. Speech Communication **52**(2) (feb 2010) 164–179
4. Valentini-Botinhao, C., Toman, M., Pucher, M., Schabus, D., Yamagishi, J.: Intelligibility Analysis of Fast Synthesized Speech. In: Proc. Interspeech, Singapore (September 2014) 2922–2926
5. Wu, Y.J., Nankaku, Y., Tokuda, K.: State mapping based method for cross-lingual speaker adaptation in HMM-based speech synthesis. In: Proceedings of the 10th Annual Conference of the International Speech Communication Association (INTERSPEECH), Brighton, United Kingdom (2009) 528–531
6. Toman, M., Pucher, M., Schabus, D.: Cross-variety speaker transformation in HSMM-based speech synthesis. In: Proceedings of the 8th ISCA Workshop on Speech Synthesis (SSW), Barcelona, Spain (August 2013) 77–81
7. Zen, H., Nose, T., Yamagishi, J., Sako, S., Masuko, T., Black, A.W., Tokuda, K.: The HMM-based speech synthesis system (HTS) version 2.0. In: Proceedings of the 6th ISCA Workshop on Speech Synthesis (SSW), Bonn, Germany (August 2007) 294–299
8. HTS working group: hts-engine. `http://hts-engine.sourceforge.net/`
9. University of Edinburgh: Festival. `http://www.cstr.ed.ac.uk/projects/festival/`
10. Carnegie Mellon University: Flite. `http://www.festvox.org/flite/`
11. Microsoft Corporation: Visual Studio. `https://www.visualstudio.com/`
12. Google Inc: Android NDK. `https://developer.android.com/tools/sdk/ndk/`
13. Black, A.W., Lenzo, K., Pagel, V.: Issues in building general letter to sound rules. In: The Third ESCA Workshop in Speech Synthesis. (1998) 77–80
14. Yoshimura, T., Tokuda, K., Masuko, T., Kobayashi, T., Kitamura, T.: Simultaneous modeling of spectrum, pitch and duration in HMM-based speech synthesis. In: Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH), Budapest, Hungary (September 1999) 2374–2350