

HOKKAIDO UNIVERSITY

Title	Recent Development of Open-Source Speech Recognition Engine Julius
Author(s)	Lee, Akinobu; Kawahara, Tatsuya
Description	APSIPA ASC 2009: Asia-Pacific Signal and Information Processing Association, 2009 Annual Summit and Conference. 4-7 October 2009. Sapporo, Japan. Oral session: Infrastructure Software for Speech Processing (5 October 2009).
Citation	Proceedings : APSIPA ASC 2009 : Asia-Pacific Signal and Information Processing Association, 2009 Annual Summit and Conference, 131-137
Issue Date	2009-10-04
Doc URL	https://hdl.handle.net/2115/39653
Type	conference paper
File Information	MP-SS1-3.pdf



Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

Recent Development of Open-Source Speech Recognition Engine Julius

Akinobu Lee* and Tatsuya Kawahara†
* Nagoya Institute of Technology, Nagoya, Aichi 466-8555, Japan
E-mail: ri@nitech.ac.jp
† Kyoto University, Kyoto 606-8501, Japan
E-mail: kawahara@i.kyoto-u.ac.jp

***Abstract*—Julius is an open-source large-vocabulary speech recognition software used for both academic research and industrial applications. It executes real-time speech recognition of a 60k-word dictation task on low-spec PCs with small footprint, and even on embedded devices. Julius supports standard language models such as statistical N-gram model and rule-based grammars, as well as Hidden Markov Model (HMM) as an acoustic model. One can build a speech recognition system of his own purpose, or can integrate the speech recognition capability to a variety of applications using Julius. This article describes an overview of Julius, major features and specifications, and summarizes the developments conducted in the recent years.**

I. Introduction

"Julius"[1] is an open-source, high-performance speech recognition decoder used for both academic research and industrial applications. It incorporates major state-of-the-art speech recognition techniques, and can perform a large vocabulary continuous speech recognition (LVCSR) task effectively in real-time processing with a relatively small footprint.

It also has much versatility and scalability. One can easily build a speech recognition system by combining a language model (LM) and an acoustic model (AM) for the task, from a simple word recognition to a LVCSR task with tens of thousands of words. Standard file formats are adopted to cope with other standard toolkits such as HTK (HMM Toolkit)[1], CMU-Cam SLM toolkit[2] and SRILM[3].

Julius is written in pure C language. It runs on Linux, Windows, Mac OS X, Solaris and other unix variants. It has been ported to SH-4A microprocessor[4], and also runs on Apple's iPhone[5]. Most of the research institutes in Japan uses Julius for their research[6][7], and it has been applied for various languages such as English, French[8], Mandarin Chinese[9], Thai[10], Estonian[11], Slovenian[12], and Korean[13].

Julius is available as open-source software. The license term is similar to the BSD license, no restriction is imposed for research, development or even commercial purposes[2].

The web page[3] contains the latest source codes and precompiled binaries for Windows and Linux. Several acoustic and language models for Japanese can be obtained from the site. The current development snapshot is also available via CVS. There is also a web forum for developers and users.

This article first introduces general information and the history of Julius, followed by its internal system architecture and decoding algorithm. Then, the model specification is fully described to show what type of the speech recognition task it can execute. The way of integrating Julius with other applications is also briefly explained. Finally, recent developments are described as a list of updates.

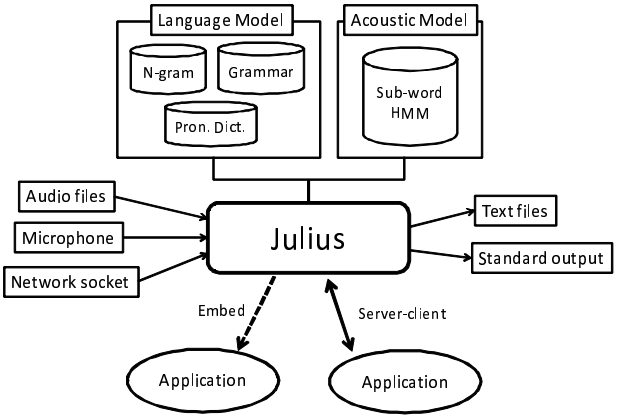


Fig. 1. Overview of Julius.

II. Overview

An overview of Julius system is illustrated in Fig. 1. Given a language model and an acoustic model, Julius functions as a speech recognition system of the given task.

Julius supports processing of both audio files and a live audio stream. For the file input, Julius assumes one sentence utterance per input file. It also supports auto-splitting of the input by long pauses, where pause detection will be performed based on level and zero cross thresholds. Audio input via a network stream is also supported.

A language model and an acoustic model is needed to run Julius. The language model consists of a word pronunciation dictionary and a syntactic constraint. Various types of language model are supported: word N-gram model, rule-based grammars and a simple word list for isolated word recognition. Acoustic models should be HMM defined for sub-word units. It fully supports HTK HMM definition file: any number of

[1]Julius was named after "*Gaius Julius Caesar*", who was a "dictator" of the Roman Republic in 100 B.C.

[2]See the license term included in the distribution package for details.

[3]`http://julius.sourceforge.jp`

states, any state transition and any parameter tying scheme can be treated as the same as HTK.

Applications can interact with Julius in two ways, socket-based server-client messaging and function-based library embedding. In either case, the recognition result will be fed into the application as soon as the recognition process ends for an input. The application can get the live status and statistics of the Julius engine, and control it. The latest version also supports a plug-in facility so that users can extend the capability of Julius easily.

A. Summary of Features

Here is a list of major features based on the current version.

Performance:

- Real time recognition of 60k-word dictation on PCs, PDAs and handheld devices
- Small footprint (about 60MB for 20k-word Japanese triphone dictation, including N-gram of 38MB on memory)
- No machine-specific or hard-coded optimization

Functions:

- Live audio input recognition via microphone / socket
- Multi-level voice activity detection based on power / Gaussian mixture model (GMM) / decoder statistics
- Multi-model parallel recognition within single thread
- Output N-best list / word graph / confusion network
- Forced alignment in word, phone or HMM-state level
- Confidence scoring
- Successive decoding for long input by segmenting with short pauses

Supported Models and Features:

- N-gram language model with arbitrary N
- Rule-based grammar
- Isolated word recognition
- Triphone HMM / tied-mixture HMM / phonetic tied-mixture HMM with any number of states, mixtures and models supported in HTK.
- Most mel-frequency cepstral coefficients (MFCC) and its variants supported in HTK.
- Multi-stream HMM and MSD-HMM[14] trained by HTS[15]

Integration / API:

- Embeddable into other applications as C library
- Socket-based server-client interaction
- Recognition process control by clients / applications
- Plug-in extension

B. History

Julius was first released in 1998, as a result of a study on efficient algorithms for LVCSR[16]. Our motivation to develop and maintain such an open-source speech recognition engine comes from the public requirement of sharing a baseline platform for speech research. With a common platform, researchers on acoustic models or language models can easily demonstrate and compare their works by speech recognition performances. Julius is also intended for easy development of speech applications. Now, the software is used as a reference of the speech technologies.

It had been developed as a part of the free software platform for Japanese LVCSR funded by IPA, Japan[17] from 1997 to 2000. The decoding algorithm was refined to improve the recognition performance in this period (ver. 3.1p2)[18]. After that, the Continuous Speech Recognition Consortium (CSRC)[19] was founded to maintain the software repository for Japanese LVCSR. A grammar-based version of Julius named "Julian" was developed in the project, and the algorithms were further refined, and several new features for a spontaneous speech recognition were implemented (ver. 3.4.2)[20].

In 2003, the effort was continued with the Interactive Speech Technology Consortium (ISTC)[21]. A number of features were added for real-world speech recognition: robust voice activity detection (VAD) based on GMM, lattice output, and confidence scoring.

The latest major revision 4 was released in 2007. The entire source code was re-organized from a stand-alone application to a set of libraries, and modularity was significantly improved. The details are fully described in section VI.

The current version is 4.1.2, released in February 2009.

III. Inside Julius

A. System Architecture

The internal module structure of Julius is illustrated in Fig. 2. The top-level structure is "engine instance", which contains all the modules required for a recognition system: audio input, voice detection, feature extraction, language model, acoustic model and search process.

An "AM process instance" holds an acoustic HMM and work area for acoustic likelihood computation. The "MFCC instance" is generated from the AM process instance to extract a feature vector sequence from speech waveform input. The "LM process instance" holds a language model and work area for the computation of the linguistic likelihoods. The "Recognition process instance" is the main recognition process, using the AM process instance and the LM process instance. These modules will be created in the engine instance according to the given configuration parameters. When doing multi-model recognition, a module can be shared among several upper instances for efficiency.

B. Decoding Algorithm

Julius performs a two-pass forward-backward search[16]. The overview of the decoding algorithm is illustrated in Fig. 3.

On the first pass, a tree-structured lexicon assigned with language model constraint is applied with a standard frame-synchronous beam search algorithm. For efficient decoding, the reduced LM constraint that concerns only the word-to-word connection, ignoring further context, is used on this pass. The actual constraint depends on the LM type: when using an N-gram model, 2-gram probabilities will be applied

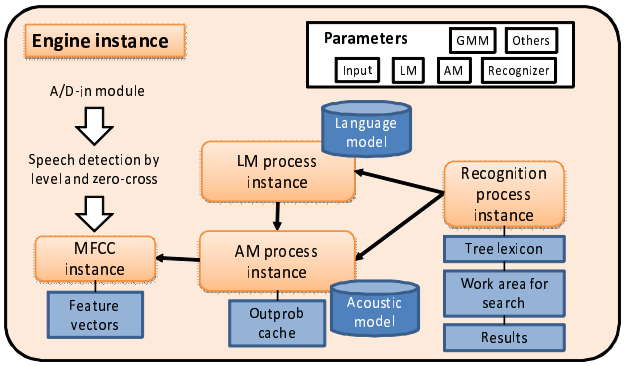


Fig. 2. Internal Structure of Julius.

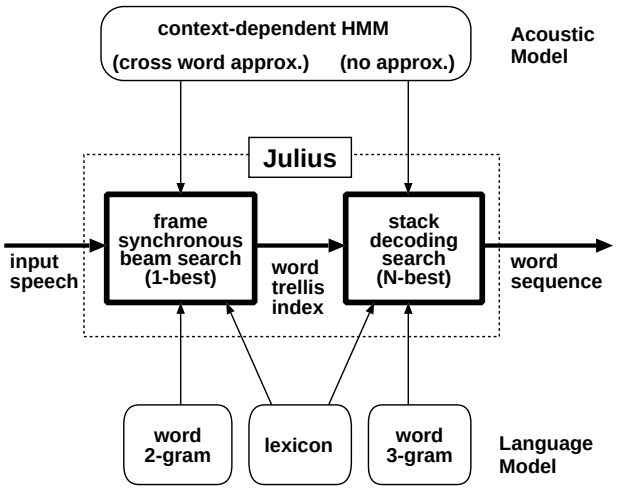


Fig. 3. Decoding Algorithm.

on this pass. When using a rule-based grammar, a word-pair constraint extracted from the given grammar will be applied. Many other approximations are introduced on the first pass for fast and efficient decoding. The cross-word context dependency is handled with an approximation which applies only the best model for the best history. Also we assume one-best approximation rather than word-pair approximation for the word context approximation[22].

The first pass generates a "word trellis index" at its end. It is a set of survived word-end nodes per frame, with their scores and their corresponding starting frames. It will be used to efficiently look up the word candidates and their scores on the later pass. Unlike the conventional word-graph rescoring, the second pass can conduct wider Viterbi scoring and wider word expansion, which will allow the later pass to overcome the accuracy loss by the approximations.

On the second pass, full language model and cross-word context dependency is applied for re-scoring. The search is performed in the reverse direction, and precise sentence-dependent Viterbi scores can be obtained by word-level stack decoding search. The speech input is again evaluated by connecting with the forward trellis as a result of the first pass. We enhanced the stack-decoding search by setting an upper limit of the hypotheses generation count at every sentence length, to avoid search failures for long inputs.

Julius basically assumes one sentence utterance per input. However, in natural spontaneous speech such as lectures and meetings, the input segment is sometimes uncertain and often gets long. To handle them, Julius has a function to perform successive decoding with short-pause segmentation, automatically splitting the long input on the way of recognition by short pauses. When a short pause is detected, it finalizes the current search at that point and then re-start recognition from the point.

IV. Speech Recognition System based on Julius

This section describes specifications of acoustic features, acoustic model, dictionary and language models supported in Julius.

A. Feature Extraction

Julius can extract a MFCC based feature vector sequence from speech input. It supports almost all variations of MFCC and energy parameters that are covered by HTK: base MFCC coefficients, energy, 0'th cepstrum and their delta and acceleration coefficients. It also supports utterance-based cepstral mean normalization (CMN), energy normalization, cepstral variance normalization (CVN) and MAP-CMN for live input. On live input, these normalization methods will be approximated by using the values of the previous input as initial normalization factors.

The parameter extraction configuration required for the given acoustic model will be partly set automatically from the AM header. However, most of the parameter values such as audio sampling frequency, window shift, window length, number of filter bank channels should be given manually to fit the acoustic condition of the acoustic model.

B. Acoustic Model (AM)

Julius supports monophone and triphone HMMs with any number of mixtures, states, and phone units. It can also handle tied-mixture models and phonetic tied-mixture models[23]. The current Julius also supports multi-stream HMM, and MSD-HMM[14] trained by HTS[15]. The output probability function should be a mixture of Gaussian densities with diagonal covariances. Speaker adaptation and full covariance models are not supported yet.

The file format should be in the HTK ASCII format. It can be converted to the Julius binary file by the tool `mkbinhmm` for faster loading at start-up. The tool can also be used to embed feature extraction parameters into a binary file.

C. Dictionary

The format of the pronunciation dictionary is common to all LM types. It is based on the HTK dictionary format. Each pronunciation should be a sequence of sub-word unit names as defined in the acoustic model. Multiple pronunciations of a word can be specified as separate entries.

Julius converts pronunciations to a context-aware form (ex. "a-k+i") when a triphone model is used. In order to specify

mapping from the logical triphone names in the dictionary to the defined (physical) model names in the acoustic model, a one-to-one mapping rule should be given as an "HMMList" file.

For memory efficiency, the maximum number of lexical (pronunciation) entries in the dictionary is limited to 65,535 by default. To use a larger dictionary, Julius should be configured so at the compilation time.

D. Language Models (LM)

Julius supports speech recognition based on N-gram model, rule-based grammars and a dictionary alone.

1) N-gram: Arbitrary length of N-gram is supported in the recent version. Class N-gram is also supported, in which case the in-class word probabilities should be given not in the language model but in the dictionary. Note that older versions (3.x) supports backward 3-gram only.

At the recognition, word 2-gram is used on the first pass and backward word N-gram is applied on the second pass. Julius can operate with single N-gram in either forward or backward direction, in which case it will compute the probabilities of the another direction using Bayes rule. However, for the best performance, both forward 2-gram for the first pass and backward N-gram for the second pass should be provided, respectively.

The file format should be in the ARPA standard format, which is the most popular format supported by various language model toolkits. Since the ARPA file is a text-based format and needs much time to parse, Julius provides a tool "`mkbingram`" to convert it into the pre-compiled binary format.

2) Rule-based Grammar: Julius can perform speech recognition based on a written grammar. The file format is an original one based on a BNF-like form, writing category-level syntax and per-category lexical entry in separate files. They should be compiled into a finite state automaton (FSA) and a recognition dictionary using a script "`mkdfa.pl`". Multiple grammars can be used at a time. In this case, Julius will output the best hypothesis among all grammars.

A tool to convert the popular HTK standard lattice format (SLF) file to the Julius FSA grammar file is also available at the website.

3) Isolated Word Recognition: When giving only a dictionary without any language model, Julius will perform isolated word recognition. Since the recognition process requires no inter-word approximation, recognition will ends at the first pass.

V. Application Development with Julius

Currently, Julius offers several ways to interact with other applications or to extend its capability.

Julius can be run as a recognition server that receives an audio stream and sends the events and the results via network socket. The audio format is a raw audio stream, and the output format is an XML text. Various events will be sent such as voice detection events, partial recognition results or the engine status. Moreover, clients can pause/resume the recognition process, send a rule-based grammar, or activate/deactivate the current grammars in Julius. The interaction will be done immediately without being blocked even if the recognition process is running.

The core engine is now implemented as a C library. It can be embedded to other applications. Application programs can assign callback functions to the main recognition loop to handle events.

The recent version also has a plug-in facility to extend its capability. Users can extend Julius with their own plug-in as a DLL or a shared object. The types of plug-ins currently supported are extensions for audio input, audio post-processing, feature extraction, feature extraction post-processing, Gaussian computation and recognition result processing.

Several sample codes are included in the source archive to help development. A simple program to conduct speech recognition is located at `julius-simple` folder. Sample codes and documents for plug-in are included under `plugin` folder. They will be a good starting point for making applications with Julius.

VI. Latest version: Julius-4

Julius version 3.x was a stand-alone program mainly for research purpose. As a result of continuous incremental improvements for years, it contains many legacy codes and was not modularized well.

The version 4 of Julius (Julius-4) was released in December 2007. The core engine part was re-written as a C library to be easily incorporated into other applications. The internal structure was re-organized and modularized to enhance readability and flexibility. As a benefit of the modularization, it now enables multi-model decoding with multiple acoustic models and/or language models at the same time. The major features are listed below:

- The core engine as a separate C library
- Grammar-based Julian are integrated
- Decoding with multiple LMs and AMs.
- Support for N-gram longer than 3
- Support for user-defined LM functions
- Confusion network output
- GMM-based VAD and decoder-based VAD[24]
- Performance optimization for memory efficiency
- New tools and functions
- Improved documentation

Julius-4 keeps the full backward compatibility with the older versions. and can be used as the same way. The performance of Julius-4.0 is almost same as the old versions, with a slight memory improvement.

This section summarizes the major features of Julius-4.

A. Re-organized internal structure

The internal structure of Julius has been greatly modified through an anatomical analysis of source codes. All the global variables has been gathered, re-organized and encapsulated into a hierarchical structure to provide much modularity and

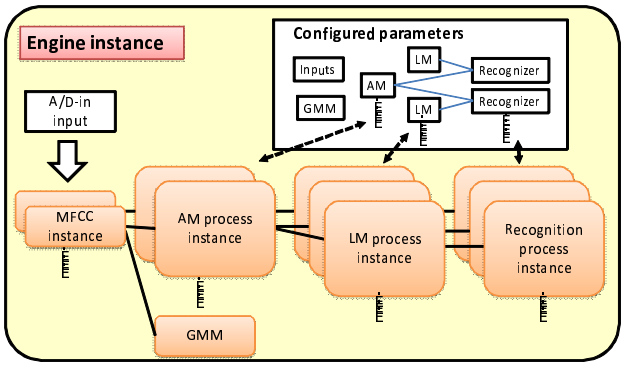


Fig. 4. Multi-model Decoding.

flexibility. This successful modularization also contributes to other new features in Julius-4, namely the unified LM implementation and the multi-model decoding.

B. Engine now becomes library: JuliusLib

The core recognition engine now moved into a C library. In the old versions, the main program consists of two parts: a low-level library called "libsent" that handles input, output and model in directory `libsent`, and the decoder itself in directory `julius`. In Julius-4, the decoder part has been divided into two parts: the core engine part (in directory `libjulius`) and the application part (in directory `julius`). Functions such as character set conversion, waveform recording, server-client modules are moved to `julius`.

The new engine library is called "JuliusLib". It contains all recognition procedures, configuration parsing, decoding and miscellaneous parts required for the speech recognition. It provides public functions to stop and resume the recognition process, and to add or remove rule-based grammars and dictionaries to the running process. It further supports addition/removal and activation/deactivation of models and recognition process instances, not only at start-up but also while running.

Julius-4 is re-implemented using the new libraries described above. It still keeps full backward compatibility with the older versions. For more information about API and the list of callbacks, please refer to the HTML documents and other documents on the website.

C. Multi-model decoding

Julius-4 newly supports multi-model recognition, with an arbitrary number of AMs, LMs and their combinations. Users can add, remove, activate and deactivate each recognition process in the course of the recognition process from the application side. Fig. 4 illustrates creating multiple instances corresponding to the multiple model definition given in configuration parameters, and their assignment in the engine. LMs of different types can be used at once. AMs of different feature parameters can also be used together. The acoustic feature (MFCC) instances will be created for each parameter type required by the AMs. Different types and combination of MFCC calculation, CMN, spectral subtraction, and other front-end processing can be used. However, the sampling frequency, window size and window shift should be shared by the AMs.

In multi-model decoding, the first pass will be performed frame-synchronously for all recognition process instances concurrently in a single thread. Then, the second pass will be performed sequentially for each process instance. After all recognition results are obtained, the engine will output all results.

D. Longer N-gram support

The old versions support only 3-gram, and always require two N-gram files, forward 2-gram and backward 3-gram. Julius-4 now supports arbitrary length of N-gram, and can operate with only one N-gram in any direction.

When forward N-gram only is given, Julius-4 uses its 2-gram part on the first pass, and use the full N-gram on the second pass by calculating backward probabilities from the forward N-gram using the Bayes rule,

$$P(w_1|w_2^N) = \frac{P(w_1, w_2, ..., w_N)}{P(w_2, ..., w_N)} = \frac{\Pi_{i=1}^{N} P(w_i|w_1^{i-1})}{\Pi_{i=2}^{N} P(w_i|w_2^{i-1})}.$$

When only backward N-gram is given, Julius-4 calculates forward 2-gram from the 2-gram part of the backward N-gram on the first pass, and applies the full N-gram on the second pass. When both forward and backward N-gram models are specified, Julius uses the 2-gram part of the forward N-gram on the first pass, and the full backward N-gram on the second pass to get the final result.

The backward N-gram should be trained from the corpus in which the word order is reversed. When using both forward N-gram and backward N-gram, they should be trained on the same corpus with the same cut-off value.

E. User-defined LM function support

Julius-4 allows word probabilities to be given from user-defined functions. When a set of functions is defined, which returns an output probability of a word on a given context, Julius uses the functions to compute the word probabilities during the decoding. This feature enables incorporating a user-side linguistic knowledge or constraints directly into the search stage of the recognition engine.

When users want to use this feature, they need to define these functions, and register to JuliusLib using an API function. Also an option `-userlm` must be specified at start-up to tell Julius to switch to the registered functions internally.

F. Isolated word recognition

Julius-4 has a dedicated mode for simple isolated word recognition. Given a dictionary only, it performs one-pass isolated word recognition.

G. Confusion network output

Julius can output recognition results as a confusion network using Mangu's method[25]. The output will present word candidates at a descending order of the confidence score. Note that search parameters should be set to search for many hypotheses to get a large network.

H. Enhanced voice activity detection (VAD)

To improve robustness for real-world speech recognition, Julius-4 features two new VAD methods. One is a GMM-based VAD, and the other is called "decoder-VAD", an experimental method using acoustic model and decoding status for speech detection[24].

Currently, both methods are experimental and not activated by default. They can be activated by specifying configuration option `--enable-gmm-vad` and `--enable-decoder-vad` at the compilation time.

I. Miscellaneous updates

- Memory improvement in the lexicon tree.
- A new tool `generate-ngram` to output random sentence from N-gram
- Fully HTK-compliant dictionary format (output string field can be omitted)
- Updated all source-code documentation for Doxygen[26]

VII. Summary of Changes

The recent version (as of July 2009) is 4.1.2. The major changes between milestone versions from 3.4.2 (released on May 2004) to 4.1.2 are summarized as below.

A. Changes from 4.1 to 4.1.2

- SRILM[3] support
- N-gram size limit expanded to 4 GByte
- Improved OOV mapping on N-gram
- Multiple-level forced alignments at a time
- Faster start-up

B. Changes from 4.0 to 4.1

- Support plug-in extension
- Support multi-stream HMM
- Support MSD-HMM
- Support CVN and VTLN
- Improved microphone API handling on Linux
- Support "USEPOWER=T" as in HTK

C. Changes from 3.5.3 to 4.0

New features:

- Multi-model recognition
- Output each recognition result to a separate file
- Log to a file instead of stdout, or no entire output
- Allow environment variables in jconf file ("`$VARNAME`")
- Allow audio input delay time via an environment variable
- Input rejection based on average power (`-powerthres`, `--enable-power-reject`)
- GMM-based VAD
- Decoder-based VAD
- Support N-gram longer than 3-gram
- Support recognition with forward-only or backward-only N-gram
- Initial support of user-defined LM
- Support isolated word recognition using a dictionary only
- Confusion network output

Compatibility issues:

- Grammar-based Julian is merged to Julius.
- Multi-path mode is integrated, Julius will automatically switch to the multi-path mode when the AM requires it.
- Module mode enhanced
- Dictionary format becomes the same as HTK
- Dictionary allows quotation

D. Changes from 3.5 to 3.5.3

- Speedup by approx. 30% by code optimization
- Greatly reduced memory access
- New grammar tools to minimize finite state automaton
- New tool `slf2dfa` to convert HTK SLF file to Julius
- Full support of all variations of MFCC extraction
- MAP-CMN for real-time input
- Parsing feature parameters of HTK Config file
- Embedding feature parameters in binary HMM files

E. Changes from 3.4.2 to 3.5

- Input rejection based on GMM
- Word lattice output
- Recognition with multiple rule-based grammars
- Character set conversion in output
- Use integrated zlib instead of executing external gzip command
- Integrate all variants (Linux / Windows / Multi-path ...) into one source tree
- MinGW support
- Source code documentation using Doxygen

VIII. Conclusion

This article briefly introduces the open-source speech recognition software Julius and describes its recent developments. Julius is a product of over eleven year's work, and the development still continues on the academic volunteer basis.

Future work should include speaker adaptation, integration of robust front-end processing, and supporting standard grammar format such as Java speech grammar format (JSGF).

References

[1] http://htk.eng.cam.ac.uk/

[2] P.R. Clarkson and R. Rosenfeld. Statistical Language Modeling Using the CMU-Cambridge Toolkit. In *Proc. of ESCA Eurospeech'97*, vol.5, pages 2707–2710, 1997.

[3] A. Stolcke. SRILM - An Extensible Language Modeling Toolkit. In *Proc. ICSLP*, pp. 901–904, 2002.

[4] H. Kokubo, N. Hataoka, A. Lee, T. Kawahara and K. Shikano. Real-Time Continuous Speech Recognition System on SH-4A Microprocessor. In *Proc. International Workshop on Multimedia Signal Processing (MMSP)*, pp. 35–38, 2007.

[5] http://www.creaceed.com/vocalia/

[6] T. Cincarek et al. Development, Long-Term Operation and Portability of a Real-Environment Speech-oriented Guidance System. In *IEICE Trans. Information and Systems*, Vol. E91-D, No. 3, pp. 576–587, 2008.

[7] T. Kawahara, H. Nanjo, T. Shinozaki and S. Furui. Benchmark Test for Speech Recognition Using the Corpus of Spontaneous Japanese. In *Proc. ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition (SSPR)*, 2003.

[8] D. Fohr, O. Mella, C. Cerisara and I. Illina. The Automatic News Transcription System: ANTS, some Real Time Experiments. In *Proc. INTERSPEECH*, pp. 377–380, 2004.

[9] D. Yang, K. Iwano and S. Furui. Accent Analysis for Mandarin Large Vocabulary Continuous Speech Recognition. In *IEICE Technical Report, Asian Workshop on Speech Science and Technology*, SP-2007-201, pp. 87–91, 2003.
[10] M. Jongtaveesataporn, C. Wutiwiwatchai, K. Iwano and S. Furui. Development of a Thai Broadcast News Corpus and an LVCSR System. In *ASJ Annual meeting*, 3-10-1, 2008.
[11] T. Alumäe. Large Vocabulary Continuous Speech Recognition for Estonian using Morphemes and Classes. In *Proc. ICSLP*, pp. 389–392, 2004.
[12] T. Rotovnik, M. S. Maucec, B. Horvat and Z. Kacic. A Comparison of HTK, ISIP and Julius in Slovenian Large Vocabulary Continuous Speech Recognition. In *Proc. ICSLP*, pp. 671–684, 2002.
[13] J.-G. Kim, H.-Y. Jung, H.-Y. Chung. A Keyword Spotting Approach Based on Pseudo N-Gram Language Model. In *Proc. SPECOM*, pp. 256–259, 2004.
[14] K. Tokuda, T. Masuko, N. Miyazaki and T. Kobayashi. Hidden Markov Models Based on Multi-Space Probability Distribution for Pitch Pattern Modeling. In *Proc. IEEE-ICASSP*, Vol.1, pp. 229–232, 1999.
[15] http://hts.sp.nitech.ac.jp/
[16] A. Lee, T. Kawahara and S. Doshita. An Efficient Two-pass Search Algorithm using Word Trellis Index. In *Proc. ICSLP*, pp. 1831–1834, 1998.
[17] T. Kawahara, et al. Free Software Toolkit for Japanese Large Vocabulary Continuous Speech Recognition. In *Proc. ICSLP*, vol. 4, pp. 476–479, 2000.
[18] A. Lee, T. Kawahara and K.Shikano. Julius – an Open Source Real-Time Large Vocabulary Recognition Engine. In *Proc. EUROSPEECH*, pp. 1691–1694, 2001.
[19] A. Lee et al. Continuous Speech Recognition Consortium — an Open Repository for CSR Tools and Models —. In *Proc. IEEE International Conference on Language Resources and Evaluation*, pp. 1438–1441, 2002.
[20] T.Kawahara, A.Lee, K.Takeda, K.Itou and K.Shikano. Recent Progress of Open-Source LVCSR Engine Julius and Japanese Model Repository. in *Proc. ICSLP*, pp. 3069–3072, 2004.
[21] http://www.astem.or.jp/istc/index_e.html
[22] R.Schwartz and S.Austin. A Comparison of Several Approximate Algorithms for Finding Multiple (N-best) Sentence Hypotheses. In *Proc. IEEE-ICASSP*, Vol.1, pp. 701–704, 1991.
[23] A.Lee, T.Kawahara, K.Takeda and K.Shikano. A New Phonetic Tied-Mixture Model for Efficient Decoding, In *Proc. IEEE-ICASSP*, Vol.3, pp. 1269–1272, 2000.
[24] H. Sakai et al. Voice Activity Detection Applied to Hands-Free Spoken Dialogue Robot based on Decoding using Acoustic and Language Model. In Proc. ROBOCOMM), pp. 1–8, 2007.
[25] L. Mangu, et al. Finding Consensus in Speech Recognition: Word Error Minimization and Other Applications of Confusion Network. In *Computer Speech and Language*, vol. 14, no. 4, pp. 373–400, 2000.
[26] http://www.doxygen.org/