DEVELOPMENT AND EVALUATION OF JAPANESE TEXT-TO-SPEECH MIDDLEWARE FOR 32-BIT MICROCONTROLLERS

Nobuyuki Nishizawa, Tomohiro Obara and Gen Hattori

KDDI Research, Inc., Japan

{no-nishizawa, to-obara, ge-hattori}@kddi-research.jp

ABSTRACT

Japanese text-to-speech (TTS) middleware for 32-bit microcontrollers (MCUs) such as Arm Cortex-M4 has been developed. Our TTS middleware is based on HMM-based speech synthesis techniques and includes an analyzer to generate pronunciation from texts that consist of Kanji (ideographic) and Kana (syllabary) characters. The middleware has been highly optimized for MCUs with the succinct data structure for data compression, fixed-point arithmetic for fast processing and pipelined processing to reduce both the required RAM size and response time. In this study, it is demonstrated that a real-time TTS system implemented on a 14-pin DIP-size MCU board that consist mainly of an MCU and external serial NOR flash can synthesize 32 kHz-sampled speech sounds with quality comparable to that of the conventional implementation of the HMM-based speech synthesis. The peak current of the MCU board at that condition is approximately 15 mA.

***Index Terms*—** text-to-speech, HMM-based speech synthesis, microcontrollers, embedded systems

1. INTRODUCTION

Especially for compact devices, use of speech output is effective because it makes limitations of the size, shape or location of the devices looser. Using text-to-speech (TTS) systems, devices can generate various information with arbitrary text. However, many compact devices have no network connectivity or only narrow-band network connectivity like Bluetooth Low Energy (BLE) [1] or low-power wide-area (LPWA) networks [2] mainly for low power consumption, where transmission of speech data is impractical. For such devices, a stand-alone TTS system must be implemented on the devices.

In recent years, many wearable sensors or the Internet of Things (IoT) sensor devices are implemented using microcontroller units (MCUs), which are one-package devices including many kinds of components for applications such as CPU cores, clock generators, static random-access memory (SRAM), flash memory, timers, digital I/O interfaces, analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). As more complex data processing on the devices is required, higher-performance MCUs, for example, MCUs with 32-bit CPU cores, large RAM and flash memory, are used. If standalone TTS software for such MCUs is provided, the devices can easily have speech output functions at small additional cost for flash memory for storage of the TTS program and data, and a loud speaker and amplifier for the loud speaker for speech sound outputs. However, how to realize standalone TTS software for MCU is not straightforward because usable computational resources are still limited compared to those required for conventional TTS software.

In this paper, our TTS middleware developed for MCUs is introduced. It is based on HMM-based speech synthesis [3] since it can generation practical quality of sounds with a small footprint. Although some HMM-based TTS systems for embedded devices has already been reported [4, 5], the target processors were not for low power consumption. By contrast, our target processor is Arm Cortex-M4, which is one of the popular 32-bit CPU cores for MCUs. Since the clock frequency of the CPU and the memory size of the MCUs are often limited, our TTS middleware has been highly optimized for MCUs. Nevertheless, our TTS middleware maintains speech quality of our former TTS software for smartphones and PCs based on HMM-based speech synthesis since the similar algorithms and data are basically used. Furthermore, to reduce the required RAM size and shorten the response time, the processing of HMM speech synthesis is pipelined where parameter generation and waveform generation are alternately executed. Also, for evaluation in this study and practical applications in future, a miniature MCU board as the target platform of our TTS middleware was developed in this study. The size of the board is comparable to 14-pin dual inline packages (DIP).

The remainder of the paper is organized as follows. First, the specifications of the target MCU board are explained in Section 2. Then, in Section 3, the structure of our TTS system is introduced for later discussion. The techniques used in our TTS middleware are explained in Section 4. Section 5 shows results of a subjective evaluation and performance evaluation with the miniature MCU board. Finally, Section 6 concludes this study.

2. MINIATURE MCU BOARD FOR TTS SYSTEMS

For practical applications in future, a miniature MCU circuit board was newly developed for our TTS middleware. The MCU was the smallest chip (100-ball, 4.618 mm $\times$ 4.142 mm, 0.4mm-pitch wafer-level chip-size package (WLCSP)) in MCUs that were easily available from the market and provided performance enough for real-time processing of the middleware. Since the internal flash memory size of the MCU was insufficient for our TTS middleware, which requires more than 4 megabyte random-accessible read-only data storages (while the requested size is significantly smaller than those of other Japanese TTS systems), an external NOR flash memory with quad-serial peripheral interface (SPI), which consists of six signal wires, was also adopted. The package size of the flash is 6 mm $\times$ 5 mm. Table 1 shows the specifications of the MCU board.

Figure 1 schematically illustrates the signal wiring of the MCU board. Seven pins of the board can be used for general-purpose I/Os (GPIOs), asynchronous serial communication or Inter-IC (I^2C) buses [6], which is a two-wire dual-direction communication protocol. In the current version, I^2C is mainly used to control the TTS system from other devices. The TTS outputs are analog outputs generated by the internal DAC of the MCU. Also, an external 32-kHz crystal is connected to the MCU to calibrate the frequency of the master clock generated by a phase-lock loop (PLL) circuit integrated in the MCU.

Figure 2 shows a top view of the MCU board. By reduction

Table 1. Specifications of the target MCU board

MCU	STM32L496VGY6 (STMicroelectronics)
Core	ARM Cortex-M4F (max freq. 80 MHz)
SRAM size	320 KiB
Flash size	1 MiB
Quad SPI flash	MX25R6435F (8MiB) (Macronix)
Crystal	32.768 kHz
Operating voltage	1.8 to 3.6 V
Dimensions	17.78 mm × 7.62 mm

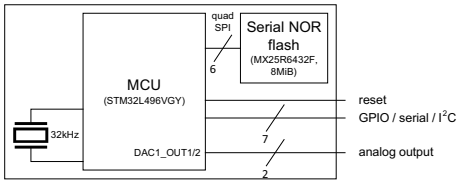

Fig. 1. The signal wiring of the target MCU board

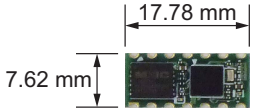

Fig. 2. Top view of the target MCU board (1:1 scale)

of the number of the devices on the board and signal traces, the board could be implemented by a 14-pin DIP-size 4-layer print circuit board. Because all devices are mounted on one side of the board and all terminals of the board are located at the sides of the board, the board can be mounted on other print circuit boards as a surface-mount device.

3. STRUCTURE OF OUR JAPANESE TTS SYSTEM

In general, input of Japanese TTS systems is text that appears Kanji (ideographic) and Kana (syllabary) characters simultaneously. Since phonemes corresponding to Kanji character depend on the neighboring characters in most cases and the context of the text in some cases, simple mapping rules from a Kanji letter to phonemes are not applicable to Japanese TTS systems, *i.e.* processing based on the word (or longer) unit is needed to predict pronunciation of Japanese text. Because there is no space between words in sentences in Japanese, morphological analysis is required to split a sentence into a sequence of words. In addition, prosodic (*e.g.* accent) expression plays very important role in naturalness of Japanese speech sounds while accent rules of Japanese are complex dependent on words [7, 8]. Consequently, text analysis in our Japanese TTS system is fundamentally based on morphological analysis with a large morpheme dictionary in which phonemic and prosodic information is bound to corresponding morpheme entries. The morphological analysis is based on the minimum-cost method that can be performed by dynamic programming.

For later discussion, processing in our Japanese TTS system is divided into three parts. Figure 3 schematically shows the parts. The first part is text processing where input text is converted to phonemic and prosodic symbols for speech synthesis. As previously mentioned, the process is based on morphological analysis. On the other hand, for the output, some notations of the Japanese phonemic and

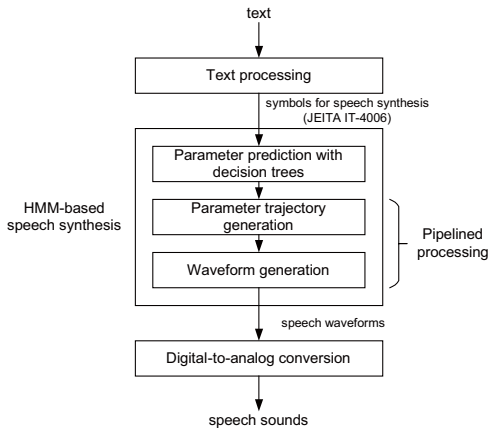

Fig. 3. The flow of the processing in our TTS system

prosodic information for speech synthesis have been standardized. In our TTS system, the standard of Japan Electronics and Information Technology Industries Association (JEITA) IT-4006 [9], which is based on Kana letters with some prosodic symbols, is used. The second part is speech waveform generation from speech synthesis symbols. In this study, HMM-based speech synthesis techniques are basically used for this part. Moreover, the HMM-based speech synthesis is divided into three subparts. They are parameter prediction with decision trees, parameter trajectory generation and waveform generation in this discussion. The parameter trajectory generation and waveform generation are pipelined. It is explained in Section 4.2. Finally, the third part is digital-to-analog (D/A) conversion of the synthesized speech waveforms. The first and the second parts are processed in the TTS middleware while the third one is out of the middleware due to hardware specificity.

4. KEY TECHNIQUES OF OUR TTS MIDDLEWARE

4.1. Compression of dictionary data for text analysis

Different from general-purpose text analyzers to process many sentences, longer processing time for text analysis may be acceptable for TTS. For example, a wait less than a second for the text processing would be less problematic for most embedded TTS applications. Reduction of the system footprints rather than improvement of the processing speed should be focused on for most embedded TTS systems.

One of adopted techniques for that sense is use of a succinct data structure [10], in which bit-wise operations are used. In our system, indices for a common-prefix search in morphological analysis is implemented by a trie based on the level-order unary degree sequence (LOUDS) [11], which is based on succinct data structures. Thus, our 244k-morpheme dictionary is compressed into an approximately 5.5-megabyte data structure that is directly used for morphological analysis. If the double-array trie [12] is used, the size of the indices alone becomes about 10 megabytes. For example, the total size of a 292k-morpheme dictionary of XIMERA [13] was about 39.4 megabytes. Although the access speed for the LOUDS is significantly slower than access for fast methods such as the double-array trie, the performance is sufficient for text analysis in embedded TTS systems even where the data are located at the external serial NOR flash. The performance is demonstrated in Section 5.

Moreover, conjugated forms of morphemes are dynamically generated in our TTS system. In Japanese, there are conjugated

forms of words in verbs, adjectives, adverbs and auxiliary verbs. Most modern morphological analyzers use morpheme dictionaries in which conjugated forms are also registered as other morphemes in advance [14]. Although the dynamic generation makes the system slower, the number of entries of the morpheme dictionary can be reduced. For example, our 244k-morpheme dictionary is comparable to a 337k-morpheme dictionary including all conjugated forms.

4.2. Computational cost reduction for HMM-based speech synthesis

Since computation of floating-point numbers is still costly[1], fixed-point arithmetic should be taken into account for systems using MCUs. For maximum-likelihood parameter generation (MLPG) [15], we use an algorithm to reduce rounding errors caused by the fixed-point arithmetic where differential trajectories from the mean sequences of the HMM outputs rather than the target trajectories are calculated [16].

Moreover, computational costs for waveform generation are often problematic in HMM-based speech synthesis. Some speech synthesizers use linear spectral pair (LSP) coefficients [4, 17] rather than cepstrum. Although use of autoregressive (AR) filters whose coefficients are calculated from the LSP coefficients can reduce the computational costs, spectral enhancement methods to improve the quality of the oversmoothed sounds caused by the HMM-based modeling are not straightforward for LSP-based systems [17]. By contrast, spectral enhancement in cepstrum-based systems is easily achieved by amplification of low-order coefficients of the cepstrum. It is similar for melcepstrum, which is cepstrum on the Mel scale. Thus, melcepstrum is adopted to simplify the system.

For systems with melcepstrum, mel-log spectral approximation (MLSA) filters [18] can be used for speech waveform generation because the parameters of the filters correspond to the coefficients of melcepstrum. However, the computational cost of the MLSA filters is not low especially for embedded devices. Therefore, we also use a different waveform generation method from melcepstrum [19] to reduce the computational costs. It is based on sinusoidal synthesis [20] performed on a subband coding system. In this method, the power spectrum is first calculated from melcepstrum, and the amplitudes of the sinusoids that are harmonic components of the fundamental vibration are then calculated from the power spectrum. Because subband codes for sinusoids can be easily calculated with a reduced sampling rate, the computational cost for waveform generation can be greatly reduced even when the cost for the decoding of the subband codes is also taken into account. The used filter bank of the subband coding system is based on the pseudo quadrature mirror filter bank [21]. In our middleware, the same filter coefficients as MPEG audio [22] are used for the filter bank, where the length is 512 taps. In addition, aperiodic components of speech sounds can be also easily synthesized by the filter bank. This is because the filter for each subband can also be used as a simple bandpass filter by setting independent noise trains for each subband. Another superiority of this method is that possible ranges of values at each point of the filters can be easily estimated because all processing consists of weighted sums. In our implementation, all operations can be executed without FPU hardware or floating-point library software.

4.3. Pipelined processing including parameter generation

In our former version of TTS software for smartphones or PCs, all trajectories of parameters for one sentence were calculated before the step of waveform generation. In practice, its computational time was not problematic because the calculation by using our algorithms explained in 4.2 was sufficiently fast for smartphones and PCs. However, the storage size for the trajectories of one sentence is not small for RAM in MCUs. Therefore, as a technique for the middleware for MCUs, pipelined processing with parameter generation and waveform generation is newly introduced. It should be noted that the D/A conversion can be parallelly performed by MCU's hardware. By the pipelined processing, required RAM size can be reduced and become irrelevant to the length of the sentence. Moreover, the latency of the system from the input of texts to the output of speech sounds can be also reduced. By contrast, the text analysis is not pipelined because it seems inefficient. Since the length of the context that can be considered in the text analysis is limited in a pipelined version, the accuracy of the analysis can deteriorate. Also, decision tree processing in the HMM-based speech synthesis is not pipelined because the intermediate label data for the decision tree processing, which are called the full-context labels, include long-term information such as sentence-length context information.

For parameter trajectory generation, a recursive algorithm similar to the recursive-least-square (RLS) method has already been proposed [23]. However, it is costlier than the conventional MLPG algorithm where computation is proportional to the square of the considered frame length in parameter updating. Different from speech conversion, when parameter generation starts, all target phonemic and prosodic symbols have already been generated in our system. It means that short latency between the input time of each symbol and the output time of the corresponding speech waveform piece is unnecessary. Therefore, our TTS system instead uses the conventional algorithm where the calculation is repeatedly preformed for the limited period of the sentence with short overlap. In this study, the length of the overlap is two frames while the period of the calculation is 33 frames where the frame period is 5 ms. It was determined by preliminary listening of synthetic sounds.

5. EXPERIMENTS

5.1. Setup for the TTS system

For the HMM-based speech synthesizer in our TTS system, HMM was trained from 10.6-hour speech sounds spoken by a female narrator. The features for speech synthesis are 31st-order melcepstrum and fundamental frequency (F0) with voiced or unvoiced information. The features include their delta and delta-delta features for the MLPG. The sampling frequency of the training date and output was 32 kHz. Training of HMMs was conducted by using the HTS [24]. All phone labels for the training were automatically generated from the manual transcriptions written in the JEITA IT-4006 format from speech sounds. It means that not only phonemic information but also prosodic information was accurate in the labels. Each phone HMM has a 5-state left-to-right structure. To build a compact model for the MCU board, the weight parameter of the minimum description length (MDL) criterion in the context clustering for states tying of HMMs was changed to 2.0 from 1.0 as the default. Consequently, the total numbers of states of HMMs were reduced to 981, 5205 and 9694 for duration, melcepstrum and fundamental frequency, respectively, while those in the default setting were 1832, 7957 and 20895, respectively. On the MCU board, the total size of the HMM was approximately 2.2 megabytes.

On the other hand, a 244k-morpheme dictionary was used for the text analysis. The size of the dictionary was approximately 5.5 megabytes. The HMM and dictionary data were stored on the external serial NOR flash memory on the MCU board.

[1]The MCU on the target board has a floating-point unit (FPU) but floating-point calculation is still costlier than integer calculation.

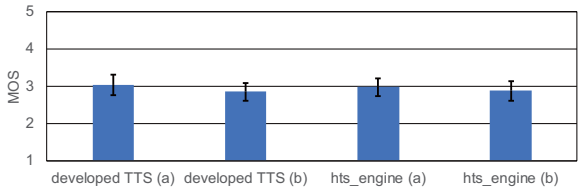

Fig. 4. Mean opinion scores of synthetic sounds. Error bars indicate 95 % confidence intervals.

5.2. Subjective experiment

To examine speech quality by our TTS system, a subjective experiment was conducted. 12 listeners evaluated quality of synthetic sounds in 5 grades (1 = bad, 2 = poor, 3 = fair, 4 = good, 5 = excellent). Ten synthetic sounds for the first ten sentences of the Group J of ATR 503-sentence speech corpus [25], which were not included in the training data for the HMMs, were examined in this experiment. For comparison, synthetic sounds by hts_engine_API Version 1.10 [26], which is a widely-used implementation of an HMM-based speech synthesizer but difficult to use on MCUs, were also evaluated. In a preliminary comparison on a PC, our middleware was approximately ten times faster than hts_engine_API. Moreover, the two sets of HMMs that were (a) an HMM set that were trained where the weight parameter of the MDL criterion was set to 1.0 as the default configuration and (b) an HMM set that were trained where the weight parameter was set to 2.0 to make a compact model for the target MCU board were used in this experiment. Consequently, forty sounds were presented for each listener. In this experiment, the target label of speech synthesis had been corrected by hand, *i.e.* the accuracy of the text analysis did not affect the results. All the sounds were 16-bit precision. The sounds were ordered randomly for each listener and presented to both ears through headphones in a silent room.

Figure 4 shows the MOS of the ten sentences for the four conditions. Although use of the compact model might degrade quality of sounds, no significant difference was found among all the conditions. At least, significant deterioration of the speech quality was not observed in the processing of the parameter generation and waveform generation in our TTS system.

5.3. Evaluation of processing speed

In this experiment, processing times for synthesizing of the 503 sentences in the ATR corpus were measured. To measure the processing times precisely, the cycle count register implemented on the MCU was monitored. Each clock cycle of the MCU increments the register. To measure the cycle counts for each part of the processing, some checkpoints were added to the program of our TTS middleware. In this experiment, the TTS processing was split into six categories: text analysis, decision tree processing for HMMs, initial parameter generation, initial waveform generation, parameter generation and waveform generation, where "initial" means processing before the D/A conversion starts. Overheads to read the cycle count register were ignored in this study because they were slight. The clock frequency of MCU was 80 MHz and the clock for the serial flash memory was 40 MHz.

Figure 5 shows average and the maximum (*i.e.* worst) relative processing times where the duration of the synthetic speech sound of each sentence equals 1. The figure indicates that our TTS system can generate speech faster than the real time. For example, the time for the (pipelined) parameter generation and waveform generation is 0.544 times the real time even in the worst case. Consequently,

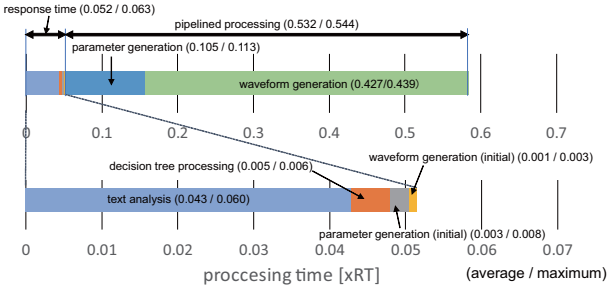

Fig. 5. The average and maximum relative processing times of the TTS system implemented on the MCU board. The length of horizontal bars corresponds to the average time.

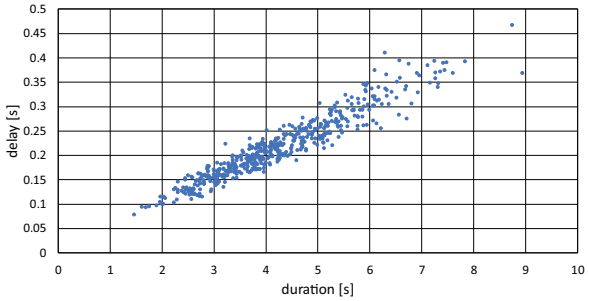

Fig. 6. The response time of the TTS system implemented on the MCU board.

the sizes of buffers for not only the pipelined processing but also generated waveforms can be reduced by temporally synchronizing the TTS processing to the D/A conversion timings.

On the other hand, figure 6 shows the relation between the duration of the sentence and the response time of the TTS system. Note that the response time in the figure does not include the delay time in the D/A conversion. The delay time is roughly proportional to the duration of the sentence because the text analysis and decision tree processing were not pipelined. Nonetheless, the response time of our TTS system implemented on the MCU board seems to be fast enough for most practical uses. For example, the delay time is only about 0.5 seconds even where the duration of the sentence is 10 seconds.

In the condition of this experiment, the peak current of the MCU board is less than 15 mA where VDD is 1.8 V. If applications accept a lower clock frequency, the current can be reduced further. In most applications with a loud speaker, the power consumption of our TTS system may be negligible because the power consumption for the loud speaker dominates the total power consumption.

6. CONCLUSION

Japanese TTS middleware and a real-time TTS system by the middleware implemented on a miniature MCU board whose size was comparable to 14-pin DIP have been introduced. Experiments showed that the quality of the sounds by the TTS system was comparable to that of the conventional implementation and the TTS system could achieve sufficient throughput for real-time processing and a short response time for 32-kHz sampled waveforms. The consumption current in this condition was only about 15 mA.

7. REFERENCES

[1] "Bletooth Specification Version 4.0," Bluetooth SIG, Jun. 2010.

[2] U. Raza, P. Kulkarni and M.Sooriyabandara, "Low Power Wide Area Networks: An Overview," IEEE Communications Surveys & Tutorials, vol. 19(2), pp. 855–873, Jan. 2017.

[3] K. Tokuda, Y. Nankaku, T. Toda, H. Zen, J. Yamagishi and K. Oura, "Speech synthesis based on hidden markov models," Proc. of IEEE, vol. 101(5), pp. 1234-1252, May 2013.

[4] S. J. Kim, J. J. Kim and M. Hahn, "HMM-based Korean speech synthesis system for hand-held devices," IEEE Transactions on Consumer Electronics, vol. 52(4), Nov. 2006.

[5] A. Rawoof, Kulesh and K. C. Ray, "ARM based implementation of Text-To-Speech (TTS) for real time Embedded System," 2014 Fifth International Conference on Signal and Image Processing, pp.192–196, Jan. 2014.

[6] "I2C-bus specification and user manual," NXP Semiconductors, UM10204, Apr. 2014.

[7] Y. Sagisaka and H. Sato, "Accentuation rules for Japanese word concatenation," Trans. IECE Jpn., vol. 66D, no. 7, pp. 849–856, 1983 (in Japanese).

[8] N. Minematsu, R. Kita and K. Hirose, "Automatic estimation of accentual attribute values of words for accent sandhi rules of Japanese text-to-speech conversion," Trans. IEICE, vol. E86-D, no. 3, pp. 550–557, Mar. 2003.

[9] "Symbols for Japanese Text-to-Speech Synthesizer," Std. of JEITA (Japan Electronics and Information Technology Industries Association), IT-4006, Mar. 2010 (in Japanese).

[10] G. Jacobson, "Succinct Static Data Structures," Carnegie Mellon University, Computer Science Department, 1989.

[11] O. Delpratt, N. Rahman and R. Raman, "Enginnering the LOUDS Succinct Tree Representaion," WEA 2006, pp. 134–145, Oct. 2006.

[12] J. Aoe, K. Morimoto and T. Sato, "An Efficient Implementation of Trie Structures," Software Practice and Experience, vol. 22(9), pp. 695–721, Sep. 1992.

[13] H. Kawai, T. Toda, J. Ni, M. Tsuzaki and K. Tokuda, "XIMERA: A new TTS from ATR based on corpus-based technologies," 5th ISCA Speech Synthesis Workshop (SSW5), pp. 179–184, Jun. 2004.

[14] T. Kudo, "Theory and implementation of morphological analysis," Kindai kagaku sha, 2018 (in Japanese)

[15] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi and T. Kitamura, "Speech parameter generation algorithms for HMM-based speech synthesis," in Proc. ICASSP, vol. 3, pp. 1315–1318, Jun. 2000.

[16] N. Nishizawa and T. Kato, "Accurate parameter generation using fixed-point arithmetic for embedded HMM-based speech synthesizers," in Proc. ICASSP, pp. 4696–4699, Prague, Czech Republic, May. 2011.

[17] Z. H. Ling, Y. J. Wu, Y. P. Wang, L. Qin and R. H. Wang, "USTC system for Blizzard Challenge 2006 – An improved HMM-based speech synthesis method," in Proc. of Blizzard Challenge 2006 workshop, Sep. 2006.

[18] S. Imai, "Cepstral analysis synthesis on the mel frequency scale," in Proc. ICASSP '83, vol. 8, pp. 93–96, Apr. 1983.

[19] N. Nishizawa and T. Kato, "Speech synthesis using a maximally decimated pseudo QMF bank for embedded devices," in Proc. 8th ISCA Speech Synthesis Workshop (SSW8), pp. 47–52, Aug. 2013.

[20] T. F. Quatieri and R. J. McAulay, "Speech transformations based on a sinusoidal representation," IEEE Trans. on ASSP, vol. 34(6), pp. 1449–1464, Dec. 1986.

[21] J. P. Princen, A. W. Johnson and A. B. Bradley, "Subband/transform coding using filter bank designs based on time domain aliasing cancellation," in Proc. ICASSP '87, Dallas, TX, U.S.A., vol. 4, pp. 2161–2164, Apr. 1987.

[22] ISO/IEC, JTC1/SC29/WG11 MPEG, "Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s Part 3: Audio," IS11172-3, 1992.

[23] K. Koishida, K Tokuda, T. Masko and T. Kobayashi, "Vector Quantization of Speech Spectral Parameters Using Statistics of Static and Dynamic Features," in Proc. IEICE Trans. Inf. & Syst., Vol. E84-D, No. 10, pp. 1427-1434, Oct. 2001.

[24] K, Tokuda, K. Oura, K. Hashimoto, S. Takaki, H. Zen, J. Yamagishi, T. Toda, T. Nose, S. Sako and A. W. Black, "HMM-based Speech Synthesis System (HTS)," http://hts.sp.nitech.ac.jp/.

[25] M. Abe, Y. Sagisaka, T. Umeda and H. Kuwabara, "Speech Database User's Manual," ATR Interpreting Telephony Research Laboratories Technical Report, TR-I-0166, Japan, Aug. 1990 (in Japanese).

[26] K, Tokuda, K. Oura, K. Hashimoto, S. Takaki, H. Zen, J. Yamagishi, T. Toda, T. Nose, S. Sako and A. W. Black, "The HMM-Based Speech Synthesis Engine "hts_engine API" version 1.10," http://hts-engine.sourceforge.net/.