Embedded, Speaker Independent Speech Recognition of Connected Digits on Java ME Enabled Cell Phones

Mattias Lingdell
IT University of Göteborg
lingdell@ituniv.se

Abstract

Cell phones are excellent candidates for speech recognition due to their sometimes cumbersome interface. However, few applications of speech recognition exist, especially for Java ME enabled cell phones. This dearth of speech recognition applications is unfortunate, since there exist several hundred millions Java ME enabled cell phones in the world, making the market and need for such applications huge. This case study investigates the feasibility of embedded, speaker independent speech recognition of connected digits on modern Java ME enabled cell phones. The developed system is, through a series of optimizations, capable of recognizing connected digits in about 11 times real time. Unfortunately, this performance is not nearly adequate to make the system practically usable in real world mobile applications that deal with user interaction.

1 Introduction

Speech recognition and cell phones at first sight seem to be the perfect match: the sometimes limited interface of cell phones could benefit greatly from being complemented with a speech based alternative. Such an interface could provide a more intuitive way of accessing some of the cell phone's functionalities, such as dialing and SMS-texting, as well as being a safer alternative in situations where the use of cell phones may be dangerous, for instance when driving a car.

The benefits of a speech interface are particularly prominent for people with difficulties interfacing with cell phones in the normal way, such as disabled and elderly people. For these people, speech based mobile applications can considerably ease the day-to-day life.

However, despite the many possible applications for speech recognition on cell phones, relatively few solutions actually exist. This is especially true for speech recognition applications developed for the Java ME platform. The main reason for this is the limited resources that Java ME enabled devices offer when it comes to memory and CPU performance.

This is an unfortunate situation since figures show that there are several hundred million Java ME enabled cell phones in the world [1, 2, 3], making the market and need for speech recognition software for Java ME very large.

In this case study I try to remedy the lack of mobile speech recognition solutions for the Java ME platform by investigating whether embedded, speaker independent speech recognition of connected digits is practically possible on today's high end cell phones.

2 Problem Definition

When developing applications for Java ME enabled cell phones, memory size and computation speed are two severely limiting factors. Since speech recognition is fairly resource heavy, both regarding memory and CPU, developing speech recognition applications for the Java ME platform can be a challenging task.

Possible solutions to this problem are to use simpler, speaker dependent techniques or to use a distributed approach where a remote back-end performs the main part of the speech recognition. Neither of these solutions are optimal, however.

Speaker dependent systems force the user to train the speech recognizer before using it. Although this process may be acceptable in some cases, it is still an additional effort that ideally should be avoided. Additionally, speaker dependent systems that use simpler techniques suitable for the low resources available on cell phones, such as Dynamic Time Warping, often have difficulties in handling variance in speech, for instance if the user is ill or has a temporarily changed voice for other reasons.

Distributed speech recognition systems must send data over a network connection to the back-end. This can be costly for the user who in most cases has to pay for transferring data over the network. If the network connection for any reason goes down, the system is not usable, leading to a less robust system.

A better solution is a speaker independent, embedded solution where all work is carried out on the cell phone. An embedded solution does not require data to be sent over the network and is therefore more robust than a distributed system. Furthermore, since no data needs to be sent, there is no additional, unwanted cost for the user. Speaker independent systems, for example those based on hidden Markov models, need no training; they can be used by different users out of the box.

In this light, the main research question in this case study is: *is embedded, speaker independent speech recognition of connected digits feasible on today's high end Java ME enabled cell phones?* This question in turn leads to two sub questions:

1. Is the cell phone's available memory sufficient?
2. Is the cell phone's processor performance sufficient?

Other research questions, for example regarding the quality of the audio captured by the cell phone's microphone, falls outside the scope of this study.

3 Background and Related Work

3.1 Speech Recognition

Speech recognition is the process of converting human speech into a textual representation of that speech. Speech recognition is performed in two main steps: feature extraction and decoding.[15]

In the feature extraction step, incoming audio data is analyzed and parameterized into acoustic features, normally mel frequency cepstral coefficients. These features are extracted by applying several techniques, such as fast Fourier transform and discrete cosine transform, to the audio data.[15, 11]

During the decoding phase the extracted acoustic features are scored against trained data and the most probable result is chosen [15].

Today, the most common method of decoding is using hidden Markov models, first introduced by Baker [8]. Phonemes are represented as hidden Markov models and these models are combined to form a search graph. When decoding, a search, often a beam search, is performed through the search graph and the most probable path is presented as the solution.[22, 23, 15, 29]

Another approach to decoding is Dynamic Time Warping, introduced by Itakura [14]. Reference acoustic feature profiles are extracted for each word in the desired vocabulary. In the decoding phase, the input speech's acoustic features are compared to each of the stored references using a dynamic programming algorithm and the one most similar is chosen as the result.[9, 15, 23, 18, 24]

Dynamic Time Warping normally uses less resources than hidden Markov model based solutions [17, 5], but is outperformed by hidden Markov models when it comes to more complex tasks, such as continuous and connected speech recognition. Hidden Markov models also performs better than Dynamic Time Warping at speaker independent speech recognition.[17, 20]

A third, less common, method of decoding is using neural networks, where a neural network is used to classify acoustic features. The neural network can then be searched to find the best solution based on the inputted acoustic features.[27]

Speech recognition can be either speaker dependent or speaker independent. Speaker dependent systems need to be trained for each user whereas speaker independent systems can handle users that they are not specifically trained for.[15, 6]

Additionally, speech recognition systems can be divided into continuous, connected and isolated-word systems. Continuous systems allow the spoken words to run together naturally whereas connected systems needs a minimal pause between words. In isolated-word systems each word must be preceded and followed by a significant pause.[15, 6]

The performance of speech recognition systems is often determined by two measurements: word error rate and real time factor [15, 29].

Word error rate is calculated in this way:

$$\frac{insertions + substitutions + deletions}{total\ numbers\ of\ words\ in\ correct\ sentence} * 100$$

The number of insertions, substitutions and deletions necessary to transform the solution given by the speech recognizer into the correct reference string are added together, and the sum is divided by the number of words in the reference string. The lower the word error rate, the better the accuracy of the speech recognition system.[15]

The real time factor is derived in this manner:

$$\frac{process\ time}{total\ length\ of\ input}$$

The time required to perform the speech recognition is divided with the length of the input audio. A lower value indicates better run time performance.[29]

3.2 Speech Recognition on Java ME Enabled Cell Phones

There are three main approaches to speech recognition on mobile devices [30]:

1. **Embedded (client-based)**
   Both feature extraction and decoding are performed on the device

2. **Network (server-based)**
   The audio data is sent over the network to a back-end that performs both feature extraction and decoding

3. **Distributed (client-server)**
   The work is divided between the mobile device and a back-end. Usually the acoustic features are extracted on the device and sent over the network to a more powerful back-end which carries out the decoding

Due to the memory and CPU limitations developers face when developing for the Java ME platform, and due to the fact that speech recognition solutions are fairly resource hungry, there exist few speech recognition software solutions for Java ME. However, some do exist.

Zaykovskiy and Schmitt have developed a Java ME based front end for feature extraction suitable for use in distributed speech recognition solutions [25].

Vlingo Mobile, a Cambridge based company, launched their network based solution in 2007 [19, 4].

Tellme., a Microsoft owned company, develops another network based solution, called simply Tellme [28].

There is ongoing work towards providing a speech recognition API for Java ME enabled cell phones, the JSR-113 Java Speech API [21, 10]. If and when this API is adopted by cell phone manufacturers, embedded speech recognition should become easier to implement and a proliferation of solutions should occur. At this time, however, few, if any, consumer cell phones implement this standard [10].

4 Methodology

In this research project a case study is carried out in which a system is developed for handling the task of embedded, speaker independent speech recognition of connected digits.

4.1 Baseline System

Sphinx-4 [29], an open source, hidden Markov model based speech recognition system written in Java, was chosen as a base for the development. The Sphinx-4 system was trained with the TIDIGITS corpus [13], a spoken corpus of 326 persons, each pronouncing 77 digits sequences. When testing Sphinx-4, it was run on a computer equipped with a Pentium M 1.6 GHz processor and 2 GB of RAM.

The target platform for the system was a Sony Ericsson K810i, running version JP-7 of Java ME. This phone is a fairly new model and can be considered high end.

Since Sphinx-4 has been developed for the Java SE platform, it had to be converted to fit the target platform, Java ME. The two main issues necessary to address during the conversion were syntax and memory consumption.

Since Java ME only offers a subset of Java SE's functionality, many constructs in Java SE code had to be converted or rewritten. Examples include annotations, generics, collections and certain input streams and readers. In addition, a number of mathematical functions, most notably the natural logarithm, is not available in Java ME. Since Sphinx-4 uses logarithms extensively, the implementation of this function can have a profound impact on the performance of the system when running on the target platform.

A challenging part of the conversion was to reduce the memory consumption to a level where the system was runnable on the target platform. When running Java SE applications on standard computers, programs have access to several hundred megabytes of heap space. In contrast, the K810i offers a maximum heap space of approximately 9 megabytes. By using the float data type instead of double for the largest data arrays and by optimizing the code in other ways, the system became runnable on the target platform. Another helpful factor was the aggressiveness of the garbage collection on the target platform.

4.2 Optimizations

After the initial conversion, which resulted in a system able to run on the target platform, several optimization techniques were applied. Since memory optimizations had already been used and proved successful, making the application able to run with the limited memory available, further optimization efforts were concentrated on decreasing the run time of the system.

The accuracy of the beam search was adjusted by tuning the absolute and relative beam widths. These parameters control how aggressively paths are pruned during the search. A more aggressive pruning leads to faster run times but reduces the accuracy of the system [15]. Care was taken to find the best trade-off between run time performance and word error rate.

During calculation of probabilities, Sphinx-4 converts all values into the log domain. Unfortunately, the log operation is fairly computation heavy, especially on cell phones that lack a dedicated floating point unit. In order to reduce the time spent in log calculation, Köhler et al. suggest an approximation of the logarithm value by using a less complex and simplified logarithm function [16]. They also show one method of doing this approximation which, however, was not used in this system. Instead, log values were approximated with the following simplification (presented in Java code) [7]:

```
public static double log(double value) {
  double x = (Double.doubleToLongBits(value) >> 32);
  return (x - 1072632447) / 1512775;
}
```

Since this approximation could introduce some inaccuracies, it might have a detrimental effect on the word error rate. This is of course also dependent on the implementation of the logarithm in the baseline system on the target platform. Recall that the Java ME platform does not provide a natural logarithm function, this must consequently be implemented from scratch.

When allocating, the system reads float values for the probability density functions, the means and variances used to model acoustic probabilities, into data structures. The Java method used to read float values, `readFloat()`, is very inefficient compared to the method used for reading bytes, `read()`. Each call to `readFloat()` was therefore replaced with a call to `read()` and a conversion from the resulting bytes to a float:

```
Float.intBitsToFloat((0xff & byte4) |
                    ((0xff & byte3) << 8 ) |
                    ((0xff & byte2) << 16) |
                    ((0xff & byte1) << 24))
```

The data used by the probability density functions, the means and variances, must be transformed [15] before being used in the probability calculations. These calculations were performed beforehand to reduce the allocating time of the system.

The K810i does not have a dedicated floating point unit, i.e. there is no hardware support for floating point arithmetic. This could lead to a severe performance handicap for the system on the target platform. In order to fully understand this performance penalty, tests were performed comparing arithmetic operations with floating point types and integer types.

Profiling showed that most of the time spent in feature extraction was spent in the fast Fourier transform phase. The fast Fourier transform uses floating point operations extensively and this imposes substantial performance problems on the target platform. In light of this and in light of the result of the arithmetic tests of floating point and integer types, the fast Fourier transform was converted into fixed point representation. The format used was Q18.13, i.e. 18 bits for the integer part, also called the *range*, and 13 bits for the decimal part, the *precision*.

Similar to the fast Fourier transform, the evaluation of probability functions in the decoding phase, essentially the scoring mechanism of the speech recognition process, also relies heavily on floating point operations when comparing the incoming features to the trained data. To speed up this process, the scoring mechanism was translated into fixed point, something that has been tried with good results by Köhler et al. [16]. The format used was the same as in the fixed point version of the fast Fourier transform, i.e. Q18.13. In order to avoid the computation cost of scaling the training data from floating point to fixed point representation at run time, the data was scaled and precomputed beforehand.

All optimizations were tested independently from each other to enable easier comparison of the various techniques. As a last step, all optimizations were combined to see the total gain in performance.

4.3 Data Collection and Analysis

The following data was collected to investigate the research questions:

1. **Run time performance**
   Allocating time and real time factor were measured
2. **Memory consumption**
   Memory consumption was collected with a profiling tool
3. **Distribution of processing time**
   The percentages of the recognizing time spent in feature extraction and decoding were measured
4. **Accuracy**
   Accuracy was measured by calculating the word error rate

The original Sphinx-4 system and the new system were tested with 100 sequences of 13 digits spoken by two speakers, one male and one female. None of the speakers are native English speakers, a fact that potentially could affect the measured word error rate (the accuracy) of the systems.

The collected data was evaluated to see if the performance of the new system is acceptable, i.e. if the system is practically usable. During this evaluation, the performance of the new system was compared to the performance of the original Sphinx-4 system.

5 Results

The following sections show the results of the original Sphinx-4 system, the baseline system and the results of the various optimizations.

5.1 Original Sphinx-4 system

The performance of the original Sphinx-4 system is shown in table 1. The absolute beam width was set to -1, which means unlimited size of the active list, and the relative beam width was set to 1E-90.

Sphinx-4 performance	
allocating time	451 ms
real time factor	0.12
word error rate	1.000%
peak heap usage	39 MB
part in feature extraction	63%
part in decoding	37%

Table 1: Sphinx-4 performance

Sphinx-4 recognizes audio more than eight times faster than real time, making it suitable for use in applications that interact with the user in real time.

Memory consumption is fairly high, especially when taking the heap space available on the target platform into consideration.

The word error rate is slightly higher than expected when compared to the numbers achieved by the Sphinx-4 team [29]. This may be due to the non-nativeness of the test speakers, since the system has been trained with native American speakers. Another reason can be that different values are used for some of the configuration parameters.

5.2 Baseline System

The performance of the developed baseline system is shown in table 2.

Baseline system performance	
allocating time	57257 ms (57.3 s)
real time factor	20.50
word error rate	0.769%
peak heap usage	4.2 MB
part in feature extraction	49%
part in decoding	51%

Table 2: Baseline system performance

Surprisingly, the word error rate is slightly lower than in the original Sphinx-4 system. The cause for this is not clear, the differing implementations of various mathematical functions, such as the natural logarithm, may be the culprits. Another possible explanation is that some configuration parameters may have been inadvertently changed during the porting to the target platform. In any case, the accuracy has not suffered from the translation from Java SE to Java ME.

The real time factor and the allocating time show dramatic increases compared to the original Sphinx-4 system, nearly 171 and 127 times respectively. Both these increases are expected taking the difference in computing power and the absence of hardware support for floating point operations on the cell phone into account.

The memory consumption is well within the limits of the target platform and has decreased more than 9 times compared to the original system. This decrease can be attributed to the attempts to optimize the memory consumption as well as to the implementation of garbage collection on the target platform, which is far more aggressive than on the source platform.

5.3 Log Approximation

The performance of the system when approximating the natural logarithm is presented in table 3.

Log approximation performance	
allocating time	38544 ms (38.5 s)
real time factor	18.42
word error rate	0.615%
peak heap usage	4.2 MB
part in feature extraction	52%
part in decoding	48%

Table 3: Log approximation performance

Contrary to what was expected, the word error rate is actually lower than in the baseline system. This shows a potential accuracy problem with the logarithm implementation used in the baseline system. The approximated logarithm function seems to produce more accurate results, leading to a lower word error rate.

The real time factor is improved by about 10%, a fairly significant improvement. The decrease in allocating time is nearly 19 seconds, which shows that the logarithm function is used extensively during the allocating phase.

Memory consumption is identical to the baseline system, as expected.

5.4 Beam Width Tuning

In order to find absolute and relative beam widths that represent a good compromise between performance and accuracy, several absolute and relative beam widths were tested to see their impact on the word error rate and the real time factor.

Table 4 shows different absolute beam widths and the resulting word error rate (wer) and real time factor (rtf). The relative beam width was set to 1E-90 (same value as in the baseline system), in all tests.

Absolute beam width tuning		
beam width	**wer**	**rtf**
25	0.769%	18.10
20	2.000%	17.26
15	5.154%	16.38
10	17.769%	15.40

Table 4: Absolute beam width tuning

At absolute beam widths of 10 and lower, the word error rate degenerates rapidly. Higher values result in word error rates that could be acceptable depending on the needs of the system under development. For this system, an absolute beam width of 20 was chosen as it strikes a good balance between accuracy and performance.

Table 5 presents various values of the relative beam width and the corresponding word error rate and real time factor. The absolute beam was set to unlimited size (-1) throughout.

Relative beam width tuning		
beam width	**wer**	**rtf**
1E-85	1.000%	19.70
1E-70	2.769%	17.94
1E-50	7.923%	14.77
1E-48	10.308%	14.23
1E-45	100%	- (not calculated)

Table 5: Relative beam width tuning

Values up to 1E-48 are usable depending on the demands on the word error rate. Higher values result in total failure in speech recognition. For this system 1E-70 was chosen as a sensible trade-off.

Table 6 shows the result of combining the selected beam widths, 20 for absolute and 1E-70 for relative.

Absolute + relative beam width tuning performance	
allocating time	57090 ms (57.1 s)
real time factor	16.63
word error rate	4.385%
peak heap usage	4.2 MB
part in feature extraction	64%
part in decoding	36%

Table 6: Absolute + relative beam width tuning performance

The real time factor shows a decrease of 19% compared to the baseline system. This, however, comes at a degeneration in word error rate, from 0.769% to 4.385%, an increase of 470%.

5.5 Precomputing Means and Variances and Read Float Optimization

The optimizations concerning precomputing values for the probability functions, the means and variances, and optimization of the reading of float values were developed together and are therefore benchmarked together. The result is shown in table 7.

Precomputing means and variances + read float optimization performance	
allocating time	37457 ms (37.5 s)
real time factor	20.6
word error rate	0.769%
peak heap usage	4.2 MB
part in feature extraction	52%
part in decoding	48%

Table 7: Precomputing means and variances + read float optimization performance

As expected, the only value that differs from the baseline system is the allocating time. It shows an improvement of nearly 20 seconds, a decrease of almost 35%. All other changes are minor and can be attributed to normal variance.

5.6 Arithmetic Performance

Table 8 shows the result of comparing arithmetic operations using floating point types and integer types. Each operation was computed 10000000 times and the average of 10 tests was calculated. In the case of integer and long multiplications, the time necessary to shift the result back into the right format (Q18.13) is included. In these tests a shift of 13 was used, other shifts may produce other results.

Arithmetic performance of float, integer and long			
operation	**float**	**integer**	**long**
multiplication	5813.9 ms	882.4 ms	4206.0 ms
division	10619.9 ms	861.6 ms	7589.2 ms
addition	5944.3 ms	878.4 ms	2152.0 ms
subtraction	6455.9 ms	883.6 ms	2118.4 ms

Table 8: Arithmetic performance of float, integer and long

The figures show that integer arithmetic has a substantial performance advantage on the target platform when compared to floating point arithmetic. Integers are more than 6 times faster at multiplication, more than 12 times faster at division and roughly 7 times faster at subtraction and addition.

Arithmetic with the long data type is also faster than floating point arithmetic. The performance gain is, however, nowhere near the one achieved with integers. This is especially true when it comes to multiplication and division, where the performance gap to floating point is not very significant. This is problematic since, when doing multiplications in fixed point, the operands must be cast to longs to avoid overflows. The consequence of this is that a large part of the anticipated performance benefit from switching from floating point to fixed point representation may vanish if many multiplication operations are performed.

5.7 Fixed Point Fast Fourier Transform

The performance of the system when using a fixed point version of the fast Fourier transform is presented in table 9.

Fixed point FFT performance	
allocating time	57238 ms (57.2 s)
real time factor	17.05
word error rate	0.769%
peak heap usage	4.2 MB
part in feature extraction	40%
part in decoding	60%

Table 9: Fixed point FFT performance

Compared to the baseline system, the real time factor is improved by almost 17%. This figure could have been higher, but the necessity to cast the operands of every multiplication to longs hurts performance, as shown in 5.6. Scaling of the fixed point values back into floating point values also consumes some of the gain, as does the need to scale the incoming acoustic features into fixed point.

Although there is inevitably some rounding errors introduced when using a fixed point representation, the word error rate has not degraded compared to the word error rate of the baseline system.

5.8 Fixed Point Scoring

Table 10 shows the performance of the system using fixed point evaluation of the probability functions, i.e. fixed point scoring.

Fixed point scoring performance	
allocating time	23112 ms (23.1 s)
real time factor	19.60
word error rate	0.769%
peak heap usage	4.2 MB
part in feature extraction	51%
part in decoding	49%

Table 10: Fixed point scoring performance

The real time factor is only improved by 4%, significantly less than anticipated. The reasons are the same as for the fixed point fast Fourier transform: the scaling of values from and to floating point representation and the casting of integers to longs in order to avoid overflows when performing multiplications.

The allocating time is about 34 seconds shorter than in the baseline system. This is due to the time saved when precomputing the values used in the probability functions and due to that integers can be read more efficiently from files than floats.

Despite the rounding errors inherent in fixed point arithmetic, the use of fixed point values has not affected the word error rate, it remains unchanged from the baseline system's.

5.9 Optimizations Combined

The system's performance with all optimizations described in the preceding sections combined is shown in table 11.

Optimizations combined performance	
allocating time	19544 ms (19.5 s)
real time factor	11.17
word error rate	3.769%
peak heap usage	4.2 MB
part in feature extraction	61%
part in decoding	39%

Table 11: Optimizations combined performance

The final system improves the real time factor with almost 46%. This performance gain results in a word error rate of 3.769%, almost 5 times worse than in the baseline system. Compared to the result from the beam width tuning, presented in 5.4, the word error rate is slightly better, due to the effect of the approximated logarithm, as observed in 5.3

The allocating time is nearly 38 seconds shorter compared to the baseline system, an improvement of almost 66%.

As expected, the memory consumption remains unchanged. There was no motivation to lower the heap usage since the current consumption lies well within the capabilities of the target platform. All the optimizations regarding memory were applied during the conversion from Java SE to Java ME, which shows when comparing the memory consumptions of the baseline system to the memory consumption of the original Sphinx-4 system.

The part spent in feature extraction has increased from 49% to 61%, indicating that most of the performance gain is due to a more efficient decoding phase.

6 Discussion

This case study shows that it is possible to perform embedded, speaker independent speech recognition of connected digits on a Java ME enabled cell phone. Unfortunately, the performance is not close to being adequate for use in mobile applications where response time is critical, i.e. almost all applications dealing with user interaction.

The available heap space on the target platform was initially regarded as the main obstacle in developing the system. However, thanks to successful memory optimizations and thanks to the aggressive implementation of garbage collection on the target platform, it proved to be a smaller problem than anticipated. Granted, a significant amount of work and time was necessary to reduce the memory consumption to an acceptable level, but in the end the system's heap space usage was well below what is available on the target cell phone.

The main challenge instead became to reduce the real time factor to an acceptable level, thus optimization techniques were chosen that addresses the run time performance of the system.

Of these optimizations, the tuning of the relative and absolute beam widths, i.e. the tuning of the behaviour of the beam search, turned out to be most effective. Depending on the tolerance for errors, significant performance gains are possible through aggressive choices of beam widths. As the results in 5.4 show, by accepting a word error rate of about 10-15%, significantly larger time savings are possible than in the current system, since the selected beam widths were chosen to maintain a fairly high accuracy.

The performance gain achieved from approximating the logarithm values was expected, since the new logarithm function is less complex than the one used in the baseline system. The decrease in word error rate, however, was not expected. It suggests that the baseline version, which had to be implemented from scratch since Java ME does not provide a natural logarithm operation, is not optimal and could be replaced by a more accurate version. However, further developing resources are better spent on finding an even more efficient and accurate approximation of the logarithm function.

The conversion to fixed point representation of the fast Fourier transform and the scoring did not result in the speed-up that was anticipated. As noted in 5.7 and 5.8, this is due to the necessity to cast the operands in multiplications to longs and due to the scaling from and to fixed point representation. Both result in operations with bad performance on the target platform. This performance penalty can be alleviated by using fixed point representation throughout the entire system, thereby removing the need to scale most of the values.

To make the system practically usable, more optimizations need to be applied. Some of the possible future optimizations are presented in section 8.

7 Conclusion

This case study resulted in a system that is capable of embedded, speaker independent speech recognition of connected digits on a Java ME enabled cell phone, the Sony Ericsson K810i.

Sphinx-4, an open source speech recognition system, was modified to run on the target platform, Java ME. This was achieved by converting the syntax and optimizing the memory consumption of the system in order to make it fit into the more restricting Java ME mould.

The memory consumption of the system was not as problematic as first anticipated. The final system's peak memory usage, approximately 4 megabytes, lies well within the boundaries of the target platform.

The resulting baseline system recognizes spoken digits in approximately 20 times real time. Through a series of optimizations, including approximation of the logarithm function and tuning of the beam search, the real time factor was lowered by almost 46%, resulting in a real time factor of around 11. This performance gain came at the cost of reduced accuracy: the word error rate increased from 0.769% to 3.769%, a deterioration of almost 400%.

Despite the considerable performance gain stemming from the various optimizations, the performance of the final, optimized system is not adequate for use in real world mobile applications, where users demand a fast response time.

8 Further Work

There are several known optimizations that could potentially result in additional speed-ups of the system. These were however outside the scope of this study, mostly due to time constraints. A summary of some of these optimizations is given below.

Since speech audio consists of real valued data, the imaginary part of each value is set to 0 when used in the fast Fourier transform. Therefore, the symmetry properties of the fast Fourier transform algorithm can be exploited [26], effectively reducing the N-point fast Fourier transform into an N/2-point transform. This can lead to performance gains of up to 50% for the fast Fourier transform phase.

In this study, only the fast Fourier transform and the scoring part of the system were ported to fixed point representation. By implementing the rest of the system's components in fixed point, further gains in run time performance should be possible, probably at the cost of reduced accuracy.

The number of acoustic features can be minimized by using early feature vector reduction [16]. The incoming acoustic feature vectors are merged by taking their arithmetic mean, thereby reducing the number of vectors to decode.

The decoding phase can be sped up by using the bucket box intersection technique [12, 16]. However, the performance gain comes at the price of lower accuracy and increased memory consumption.

One obvious augmentation of the current system that does not fall into the realm of optimization would be to incorporate the cell phone's microphone by feeding the speech recognizer speech data directly from it. However, to do this a cell phone is needed capable of recording sound in a uncompressed, raw format, such as the WAV-format. The cell phone used in this study, the Sony Ericsson K810i, only supports sound recordings in the AMR-format, which is a compressed format, rendering it useless for speech recognition purposes.

References

[1] Sun in telecom. Press kit, Sun Microsystems, March 2005. [Online] Available: http://www.sun.com/aboutsun/media/presskits/ctia2005/Sun_Telecom_External_V2.pdf.

[2] Sun microsystems celebrates first decade of java technology innovation at the world's premier java developer conference, javaone 2005. Press release, Sun Microsystems, June 2005. [Online] Available: http://www.sun.com/smi/Press/sunflash/2005-06/sunflash.20050627.9.xml.

[3] Java technology at-a-glance. Press kit, Sun Microsystems, November 2006. [Online] Available: http://www.sun.com/aboutsun/media/presskits/2006-1113/Java_aag_100306.pdf.

[4] Revolutionizing voice ui for mobile: Vlingo unconstrained speech recognition. White paper, Vlingo Mobile, August 2007.

[5] W.H. Abdulla, D. Chow, and G. Sin. Cross-words reference template for dtw-based speech recognition systems. In *Proceedings of TENCON 2003: Conference on Convergent Technologies for Asia-Pacific Region*, volume 4, pages 1576–1579, Bangalore, India, October 2003.

[6] M.M. Ahmed and A.M.B. Ahmed. Review and challenges in speech recognition. In *Proceedings of ICCAS 2005: International Conference on Control, Automation and Systems*, Kintex, Gyeonggi-Do, Korea, June 2005.

[7] M. Ankerl. Optimized pow() approximation for java and c / c++, October 2007. [Online] Available: http://martin.ankerl.com/2007/10/04/optimized-pow-approximation-for-java-and-c-c/.

[8] J.K. Baker. Stochastic modeling for automatic speech understanding. In A. Waibel and K.-F. Lee, editors, *Readings in speech recognition*, pages 297–307. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[9] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[10] Conversay. Conversay :: Speed, accuracy, safety, value, 2007. [Online] Available: http://www.conversay.com/Technology/JSAPI2/tabid/55/Default.aspx.

[11] S.B. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, August 1980.

[12] J. Fritsch and I. Rogina. The bucket box intersection (bbi) algorithm for fast approximative evaluation of diagonal mixture gaussians. In *Proceedings of ICASSP '96: IEEE International Conference On Acoustics, Speech And Signal Processing 1996*, pages 837–840, Atlanta, GA, USA, May 1996.

[13] Texas Instruments. Texas instruments-developed studio quality speaker-independent connected-digit corpus (tidigits). *NIST Speech Discs*, February 1991. 4-1.1–4-3.1 (3 discs).

[14] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, February 1975.

[15] D. Jurafsky and J.H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Prentice Hall, Upper Saddle River, NJ, USA, 2000.

[16] T.W. Köhler, C. Fügen, S. Stüker, and A. Waibel. Rapid porting of asr-systems to mobile devices. In *Proceedings of Interpeech '05: 9th European Conference on Speech Communication and Technology*, pages 233–236, Lisbon, Portugal, September 2005.

[17] C. Lévy, G. Linarès, P. Nocera, and J.F. Bonastre. Reducing computational and memory cost for cellular phone embedded speech recognition system. In *Proceedings of 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 309–312, Montreal, Quebec, Canada, May 2004.

[18] J. Martino. Dynamic time warping algorithms for isolated and connected word recognition. In R.D. Mori and Y. Suen, editors, *New systems and architectures for automatic speech recognition and synthesis*, pages 405–418. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[19] Vlingo Mobile. vlingo, 2008. [Online] Available: http://www.vlingomobile.com/.

[20] B.I. Pawate and P.D. Robinson. Implementation of an hmm-based, speaker-independent speech recognition system on the tms320c2x and tms320c5x. Technical report, Texas Instruments Inc., 1996.

[21] Java Community Process. Jsrs: Java specification requests - detail jsr-113, 2008. [Online] Available: http://jcp.org/en/jsr/detail?id=113.

[22] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[23] L.R. Rabiner and B.-H. Juang. *Fundamentals of speech recognition.* Prentice-Hall, Upper Saddle River, NJ, USA, 1993.

[24] S. Salvador and P. Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *Proceedings of ACM KDD '04: 3rd Workshop on Mining Temporal and Sequential Data*, pages 70–80, Seattle, WA, USA, August 2004.

[25] A. Schmitt and D. Zaykovskiy. Java (j2me) frontend for distributed speech recognition. In *Proceedings of AINAW '07: 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 353–357, Niagara Falls, Canada, 2007.

[26] H.V. Sorensen, D.L. Jones, M.T. Heideman, and C.S. Burrus. Real-valued fast fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(6):849–863, June 1987.

[27] J. Tebelskis. *Speech Recognition using Neural Networks.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1995.

[28] Tellme. Tellme: Say it. get it., 2008. [Online] Available: http://www.tellme.com/about.

[29] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel. Sphinx-4: A flexible open source framework for speech recognition. Technical Report TR-2004-139, Sun Mircosystems, November 2004.

[30] D. Zaykovskiy. Survey of the speech recognition techniques for mobile devices. In *Proceedings of SPECOM 2006: 11th International Conference "Speech and Computer"*, pages 88–93, St. Petersburg, Russia, June 2006.