Hamburg University
Faculty of Mathematics,
Informatics and Natural Sciences
Department of Informatics

Master Thesis

Incremental Speech Recognition from Several Combined Sources

submitted on 04.09.2016

Natalia Orlova

2orlova@informatik.uni-hamburg.de
Degree Program: Master Informatics
Matr.-No. 6459416

First supervisor Hamburg University:	Dr. Timo Baumann
Second supervisor Hamburg University:	Prof. Dr. Wolfgang Menzel

Abstract

This master thesis investigates the possibilities of Google+Sphinx combination within the frames of Incremental Spoken Dialogue Processing Toolkit (InproTK). By contrast to Google the combination of Google+Sphinx returns timing information, which is of great importance to dialogue systems, involving non-verbal recognition. The central part of the the proposed architecture is a Incremental Unit (IU) module, which guarantees that Google incremental results are passed to Sphinx recognizer, working in a forced-alignment mode. Google+Sphinx alignment results as well as the results of Google and Sphinx alone are evaluated against the gold standard, using timing metrics. Furthermore, we propose approaches, allowing combination of alignment and recognition within Google+Sphinx architecture. The second aspect of the analyses is the Sphinx-4 search graph path alternations strategies. Finally, we discuss the problems of timing and timeliness results evaluation.

Zusammenfassung

Diese Masterarbeit untersucht die Möglichkeiten Google+Sphinx im Rahmen von Incremental Spoken Dialogue Processing Toolkit (InproTK) zu kombinieren. Im Vergleich zu Google die Kombination Google+Sphinx liefert zeitliche Information, die sehr wichtig für Dialogsystemen mit non-verbal Erkennung ist. Das zentralen Teil von vorgestelltem Architektur ist Incremental Unit (IU) Modul, das garantiert, dass die incrementelle Google Ergebnisse werden an Sphinx Erkenner, gestartet im forced alignenment Modus, weiter geleitet. Ergebnisse sowie von Google+Sphinx, als auch von Google und Sphinx allein werden gegen gold Standard evaluiert, dabei werden zeitliche Metriken benutzt. Weiterhin, we schlagen vor die Vorgehensweise, um die Kombination von Alignment und Erkennung in Rahmen von Google+Sphinx Architektur zu gewährleisten. Der zweiten Aspekt von Analyse sind die Strategien, die erlauben der Pfad im Sphinx-4 Suchgraph zu ändern. Zum Schluss werden die Problemen von zeitlichen Ergebnissen besprochen.

Contents

1 Introduction **v**
- 1.1 Motivation . . . v
- 1.2 Problem Statement . . . 1
- 1.3 Objectives . . . 1
- 1.4 Structure . . . 1

2 Related Work **3**

3 Theoretical Background **5**
- 3.1 Forced Alignment and Timing . . . 5
- 3.2 Alignment Quality Metrics . . . 6
  - 3.2.1 Mean Error . . . 8
  - 3.2.2 Standard Deviation . . . 9
  - 3.2.3 Root Mean Squared Error . . . 10
- 3.3 Timeliness in Speech Recognition . . . 10

4 Architecture of CMU Sphinx-4 as an Example of a typical ASR **13**
- 4.1 Overview . . . 13
- 4.2 Acoustic Analysis . . . 14
- 4.3 Acoustic Model . . . 15
- 4.4 Language Model . . . 16
- 4.5 Dictionary . . . 17
- 4.6 Decoder . . . 18
- 4.7 Search Graph and Linguist . . . 20

5 Implementation **21**
- 5.1 Development Environment Overview . . . 21
- 5.2 IU Modules and Data Processing . . . 21
- 5.3 INproTK IU Modules in Google+Sphinx Recognizer Architecture . 22
  - 5.3.1 Google IU Module . . . 22
  - 5.3.2 Sphinx ASR Module and Label Writer in INproTK . . . 24

5.4 Architecture Overview of the Google+Sphinx Recognizer 24
5.5 Forced Alignment of Google Incremental Results with Google+Sphinx Recognizer 25
5.5.1 Forced Alignment Mode of the Google+Sphinx Recognizer . 26
5.5.2 Alignment + Recognition Mode of the Google+Sphinx Recognizer 28
5.5.3 Alignment + Recognition with JSGF Grammar 30
5.5.4 Alignment + Recognition with N-Gram Grammar 31

6 Results and Evaluation 33
6.1 Testing Approach 33
6.2 Test Results 34
6.2.1 Alignment quality 34
6.2.2 Timeliness quality 35
6.3 Tests Evaluation 35

7 Discussion 37
7.1 Conclusion 37
7.2 Future Prospects 38

A Phonetic Alphabet Notation (IPA) 39

B Examples of Google+Sphinx Recognizer Output Files 41
B.1 Sphinx Recognition Results 41
B.2 Google Recognition Results 47
B.3 Sphinx+Google (Alignment) 48
B.4 Sphinx+Google (Alignment + Recognition and JSGF Grammar) . . 50
B.5 Sphinx+Google (Alignment + Recognition and N-Gram LM) 51

Bibliography 52

1 Introduction

1.1 Motivation

In recent years, Automatic Speech Recognition (ASR) systems have improved increasingly, being used in everyday applications: Siri, Google ASR (McGraw and Gruenstein 2012). However, most of such systems work asynchronously in respect to output, computing the result after the utterance is already finished. In the area of Human-Machine Interaction, where intermediate system reactions of ASRs are desirable, incremental output of the intermediate results becomes increasingly important. Benefits of incremental speech recognition include post-processing time savings, faster system feedback and more natural dialogue between humans and intelligent systems.

Commercial Google recognition engine, working in incremental mode, produces accurate results in non-specific domains, but demonstrates higher latency (McGraw and Gruenstein 2012). Non-commercial open source systems, like Sphinx-4, on the other hand, demonstrate very short delays and can be timed to specific applications, but are less reliable in accuracy.

Moreover, Google results lacks timing information that can a great benefit, for example, in robotics environment. In such kind of applications it is often important to know at what time a particular word was said to relate and refer the objects correctly to other non-verbal information, like gestures. The challenge is to combine the advantages of both systems and to enrich the Google results with timing information and try to overcome the latency problem of Google ASR.

1.2 Problem Statement

The aim of this master thesis is to investigate the possibility of timeliness and timing improvement of Google ASR incremental results by developing an incremental speech recognizer, using a combination of Google ASR (McGraw and Gruenstein 2012) and a Sphinx-4 speech recogniser (Baumann, Atterer, and Schlangen 2009).

1.3 Objectives

The following objectives states, how the aim of the master thesis is going to be addressed:

- to align incremental results,coming from Google ASR, using a combination of Google and Sphinx ASRs,
- to evaluate timing results of forced alignment, comparing Google alone, Sphinx alone and Google + Sphinx combination, using evaluation metrics,
- to investigate the possibility for Google timeliness improvement in multiple source recogniser,
- to evaluate results of timeliness improvement, comparing Google alone, Sphinx alone and a Google + Sphinx combination.

1.4 Structure

This master thesis is structured as follows. Chapter 1give a brief introduction of the research topic. Chapter 2 provides review of state of the art scientific papers, related to the topic of incrementality, general speech recognition and multiple source recognizers. Chapter 3 explains the terminology to be used in the context of this master thesis. In the first place there is a definition try of such terms as timing and timeliness an an explanation of what kind of metrics are relevant to evaluate their quality. Chapter 4 describes the architecture of CMU (Carnegie

Mellon University) Sphinx-4 recognizer as a typical ASR. Implementation of the Google+Sphinx recognizer is discussed in the chapter 5 of this paper. Chapter 6 describes the evaluation approach and results,received for Google+Sphinx recognizer, as well as for Google and Sphinx alone. Chapter 7 summarizes the most important ideas of the master thesis and provides some points for discussion and future work.

2 Related Work

Although architecture of all contemporary speech recognition systems includes such common ASR components as acoustic input, acoustic model, language model, search module and decoder, significant differences in performance and application approaches are still to be observed due to the system complexity.

During the process of speech recognition decoder gets feature vectors of audio signal and results of acoustic and language modelling as input and produces a decoded word-phone alignment as an output (Jurafsky and Martin 2009). Traditionally ASRs were designed sequentially, producing the first output result after the utterance is already finished, thus resulting in a delay up to 750-1500 ms. By contrast, incremental speech recognition, which can be seen as a part of dialogue processing systems, allows reducing of feedback time, bringing the dialogues in interactive environments closer to natural ones (Skantze and Schlangen 2009). Such dialogues systems can be understood very widely, including verbal and/or non-verbal communication. In all cases reactivity, meaning a prompt incremental result of the speech recognizer, is seen as important criteria for the system performance.

A good example of traditional ASR architecture is the implementation of the open-source CMU Sphinx-4 recogniser. Sphinx-4 is an open-source local recognition system, that allows tuning for specific applications and embedded environments (Lamere et al. 2003). Word Error Rate (WER) of Sphinx-4 varies from 0.168 for isolated digit recognition to 18.7 for large vocabulary (Carnegie Melon University). Though, Sphinx-4 was not designed as an incremental system, its modular design allows extension and adding new components, required for an incremental speech recogniser. As a result of such extension an incremental speech recognizer, based on sphinx-4 was implemented. The authors present very detailed analyses of timing for incremental results of Sphinx ASR (Baumann, Atterer, and Schlangen 2009).

Apart from Sphinx-4 the leading speech recognition systems is commercial Google Voice Search (Schalkwyk et al. 2010). Google runs in non-incremental as well as incremental mode. In comparison to Sphinx-4 cloud-based Google ASR uses language models, trained with large amount of audio data, presents more accurate results in the terms of WER. At the present moment WER of Google ASR for general tasks with large vocabulary equals about 8%. However, there is always trade-off between stability of the input and latency (McGraw and Gruenstein 2012).

Further restriction of Google is its task-orientated recognition, primarily aimed to interactive Google command search. This generally means that simple transfer of Google technology to domain-specific natural dialogue systems and context-specific HRI environments leads to lower accuracy. Besides, as Google ASR recognizer features, such as, for example, timing information, that present research interest in domain-dependent and/or embedded application, remains encapsulated in a "black box".

Recently, there has been proposed a solution for a domain-dependent applications, using a combination of Google phonetic post processing and Sphinx-4. Original frontend of Sphinx-4 is replaced by phoneme frontend, which converts the Google result to its phonetic representation. Test results of the combined approach have shown a significant improvement of recognition results for non-incremental tasks. Combination technique is considered to be transferable to the incremental problem solution (Twiefel et al. 2014).

3 Theoretical Background

3.1 Forced Alignment and Timing

One of the central algorithm of the modern ASRs is a token passing algorithm. Recognition is the process of producing hypotheses, that fit best with the audio input. During decoding these hypotheses are saved as a list of tokens, which contains information about alignment, states and transitions. (S.J. Young and Thornton 1989).

Generally, *forced alignment* is the process of finding the correspondence between a piece of text and audio speech segments. In the case, when the transcription is known *forced alignment* consists only in determining where in time these particular words occur in the speech segment. For example, if we take transcription *ich danke Ihnen und wir sehen uns dann* and run Sphinx ASR in the forced alignment mode with the corresponding audio input than we get the following incremental output (see 3.1). By giving the transcription to the recogniser we narrow its search space for hypotheses to one particular path. Even if the recognizer receives some text, that differs from the *gold standard* text, pronounced in our audio, it will still try to align it to the audio as good as possible. By contrast, during normal recognition we are dealing with numerous paths and possible states in the search graph. During decoding alignment is computed synchronously together with search states. Final path can differ from *gold standard* transcription, depending on the recognition quality. The more further recognizer in their hypotheses from the original transcription the worse is the recognition quality. Comparison between two sphinx outputs (see pictures 3.1 and 3.2) makes it obvious that alignment quality depends on the recognition quality.

Forced alignment is of great importance in fast-moving environments, where additional non-verbal information pieces, for example gestures, are used. Without timing information in such kind of applications it not always possible to anal-

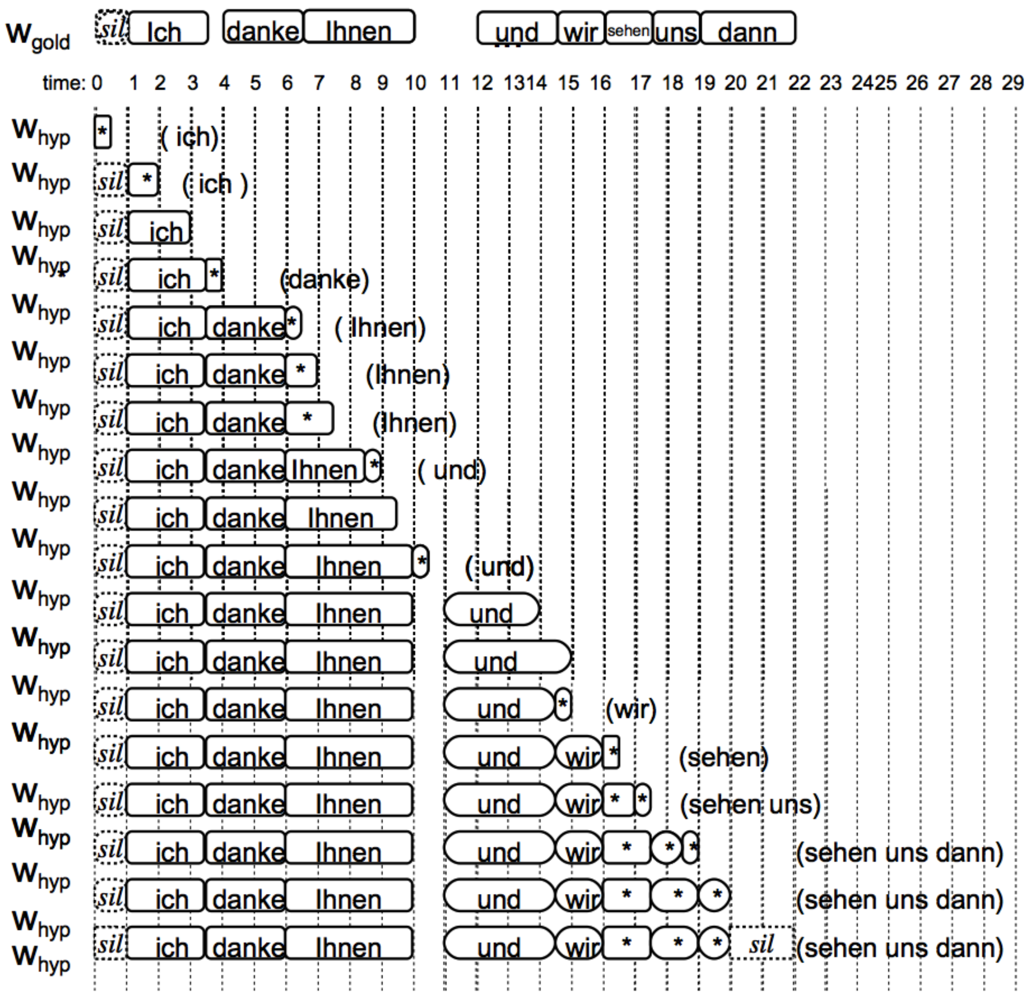

W_gold
sil
Ich
danke
Ihnen
und
wir
sehen
uns
dann
time: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
W_hyp
*
(ich)
(sehen uns dann)

Figure 3.1: Incremental output of Sphinx ASR, running in the forced alignment mode.

yse correctly which object is referenced (Baumann, Kennington, et al. 2016). In contrast to Sphinx, online Google, being a black box, does not output alignment timing, which makes it impossible to use this recognizer for applications, using non-verbal information and context-dependent object referencing.

3.2 Alignment Quality Metrics

Alignment quality metrics describe how good is the timing of the output, produced by the recognizer, compared with the *gold standard*. The main metrics that can be used for this purpose are: *mean error*, *standard deviation* and and *root*

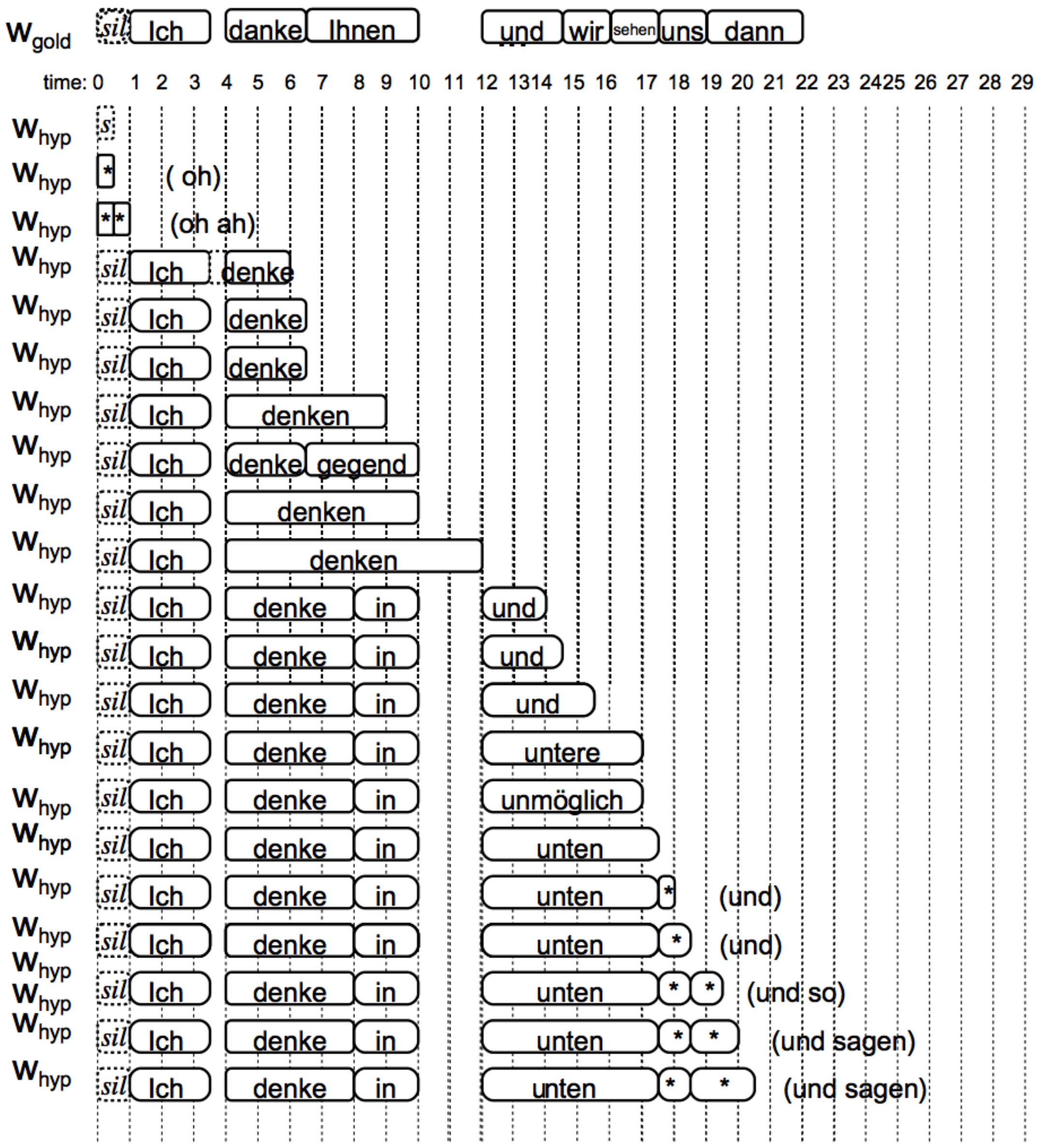


Figure 3.2: Incremental output of Sphinx ASR, running in the normal recognition mode.

mean squared error. In this paper we differentiate between incremental and non-incremental approach to the calculation of the errors. For non-incremental approach only the final timing result of the recognizer are compared with the *gold standard*. By contrast, incremental approach requires to compare incremental timing results with *gold standards*. How exactly the errors are computed is explained in details in the subsections below.

3.2.1 Mean Error

Mean error μ describes the average of absolute errors. Reformulated in accordance with the alignment problem of a recognizer *mean error* can be expressed mathematically as follows:

$$\mu = \frac{\sum_{i=0}^{N} |t_{i_{start}(rec)} - t_{i_{start}(gold)}| + |t_{i_{end}(rec)} - t_{i_{end}(gold)}|}{N} \tag{3.1}$$

In the formula 3.1, shown above, $t_{i_{start}(rec)}$ and $t_{i_{end}(rec)}$ equals the start and the end of a matched or substituted word in the recognizer output, $t_{i_{start}(gold)}$ and $t_{i_{end}(gold)}$ equals the start and the end of a corresponding word in the gold standard and N is the total number of matches and substitutions or only matches.

In the case of the forced alignment of a phrase, that is equal with the transcription, the total number of matches and substitutions is equals the number of words (compare 3.2). When ASR runs in the normal recognition mode the number of matches and substitutions can differ from the number of words as we are possibly dealing with some deletions or insertions. Matches and substitutions are relatively easily obtained from the Levenshtein distance matrix, containing substitutions, matches, deletions and insertions of characters, occurring in two strings (Levenshtein 1966).

For the above example (figure 3.2) the mean error, related to the final result *Ich danke ihnen und wir sehen uns dann*, is computed as follows:

$$\mu = \frac{(0 + 100 + 50 + 150 + 0 + 0 + 0 + 200)}{8} \approx 50ms \tag{3.2}$$

For incremental approach it is necessary first to calculate the *mean error* of the incremental results for every word, belonging to the final match or substitution. In addition, we have to consider only incremental results with changing timings. As soon as the result is stable it is not any more significant for the analysis of timing incrementality. The reason is that the words in the beginning of the utterance appear ofter in the hypotheses and will minimize their error with every new hypotheses, if the mean error is computed over all of them.

In the example above (figure 3.2) the word *ich* appears 18 times in the output, but only first 4 outputs are relevant for incremental timing analyses. Starting from the 5-th incremental result the timing for the *ich* does not change any more. The mean error of the word *ich* for the first 4 incremental results equals $\frac{400+200+50+0}{4} \approx 163ms$, whereas for 18 incremental results it equals only $\frac{400+200+50+0*15}{18} =\approx 36ms$.

The *mean error* for all words in incremental case in the above example (figure 3.2) is the following:

$$\mu = \frac{(163 + 200 + 240 + 233 + 225 + 67 + 83 + 250)}{8} \approx 183ms \tag{3.3}$$

3.2.2 Standard Deviation

Standard deviation is the square root of its variance :

$$\sigma = \sqrt{Var} \tag{3.4}$$

Variance $Var(X)$ in its turn is the average of the squared differences from the *mean*:

$$Var(X) = \frac{\sum_{i=0}^{N}(t_i(rec) - \mu)2}{N} \tag{3.5}$$

In the above formula $t_i(rec)$ is the actual recognizer timing and N is again is the number of matches and substitutions. Standard deviation in the Sphinx example (figure 3.2) with non-incremental approach is calculated as shown below:

$$\sigma = \sqrt{\frac{2500 + 2500 + 0 + 10000 + 2500 + 2500 + 2500 + 22500}{8}} \approx 75ms \tag{3.6}$$

For incremental case σ is first computed for every word in the utterance as the average of the squared differences of all relevant incremental results from the mean (equation 3.12). For the word *ich* in the above example (figure 3.2) it equals:

$$\sigma = \sqrt{\frac{(400 - 183)^2 + (200 - 183)^2 + (50 - 183)^2 + (0 - 183)^2}{4}} \approx 157ms \tag{3.7}$$

Finally, the σ for all words is computed as an average of the sigmas of all the words in the utterance:

$$\sigma_{incr} = \frac{(157 + 101 + 201 + 138 + 342 + 84 + 79 + 168)}{8} \approx 159ms \tag{3.8}$$

3.2.3 Root Mean Squared Error

Root mean squared error (RMSE) is the square root of *mean error* and *standard deviation*

$$RMSE = \sqrt{\mu^2 + \sigma^2}. \tag{3.9}$$

For the Sphinx example above (figure 3.2) in non-incremental case it equals:

$$RMSE = \sqrt{50^2 + 75^2} \approx 90ms. \tag{3.10}$$

In the incremental case the result equals:

$$RMSE = \sqrt{183^2 + 159^2} \approx 242ms. \tag{3.11}$$

3.3 Timeliness in Speech Recognition

Timeliness is the second term that is used to characterize word occurrences during recognition in relation to the time axis. To describe the term *timeliness* the following evaluation metrics are used:

- First Occurrence (FO) - is the time between the (true) beginning of a word and the first time it occurs in the output (regardless if it is afterwards changed).

- Final Decision (FD) - is the time between the (true) end of a word and the time when the recognizer decides on the word, without later revising it anymore.

Timeliness is only measured for the words that are correctly recognized or at least appear in the final output of the recognizer (Baumann, Kennington, et al. 2016).

Timeliness characterizes, how fast is the recognizer in its hypotheses. In the above depicted output of Sphinx ASR (see figures 3.1 and 3.2) FO as well as FD of the word *ich* in the first case (forced alignment mode) takes plays a half of the slot earlier than in the second case (normal recognition) and before the word *ich* ends in the audio.

Metrics that are used in this thesis for the timeliness quality evaluation are: mean, standard deviation and median. For the above example (see figure 3.1) mean is computed as shown below:

$$\mu(FO) = \frac{((-50) + 0 + (-50) + (-300) + 50 + 50 + 0 + 0)}{8} \approx -37,5ms \tag{3.12}$$

For standard deviation we have:

$$\begin{aligned} \sigma = \sqrt{\frac{156,25 + 1406,25 + 156,25 + 68906,25+}{8}} \dots \\ \dots \sqrt{\frac{+156,25 + 156,25 + 1406,25 + 1406,25}{8}} \approx 34ms \end{aligned} \tag{3.13}$$

Median, which is the "middle" value of the data set, equals 0. For FD the quality metrics are calculated analogue.

4 Architecture of CMU Sphinx-4 as an Example of a typical ASR

4.1 Overview

CMU Sphinx represents a classical speech recognizer, including the following main components: frontend, decoder and knowledge base with language and acoustic models and dictionary. In the next sections the components of the speech recogniser are analysed in more details.

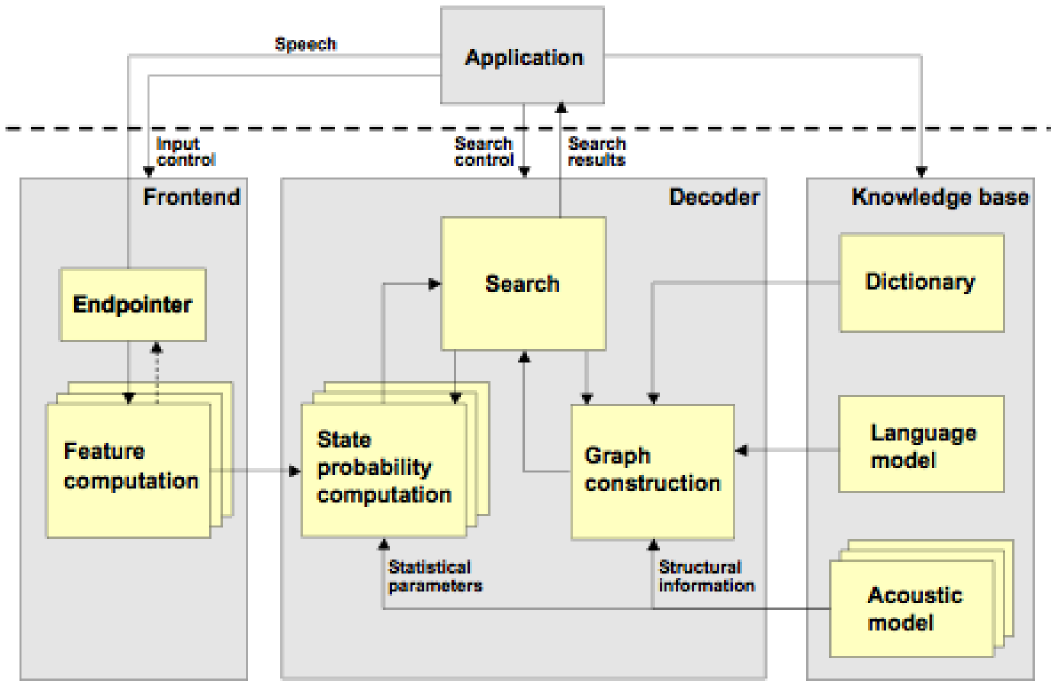


Figure 4.1: Sphinx-4 Architecture Overview (Lamere et al. 2003).

4.2 Acoustic Analysis

Frontend computes feature vector Y - representation of a speech signal, using acoustic analysis. It provides methods for manipulating the data processors, performing specific transformation functions on input data (Lamere et al. 2003).

In a physical sense human speech is a sound wave, characterized by a particular amplitude, frequency, period and length. Decoding is preceded by features extraction and digitalizing of a signal in such a way, that mathematical operations and calculations are made possible.

Preprocessing of speech as an analogous signal demands the following steps. First, the waves are sampled, which allows getting a discrete signal from the continuous one. Second, the samples are quantized, mapping the analog values to digitalised ones. Third, a digital signal is converted to its spectral representation, applying a short-time Fourier transform. The resulting sound spectrum contains the amount of vibration at each individual frequency. Figure 4.2 shows a wave form, pitch contour, spectral representation and transcription with alignment of the phrase *ich danke Ihnen und wir sehen uns dann*.

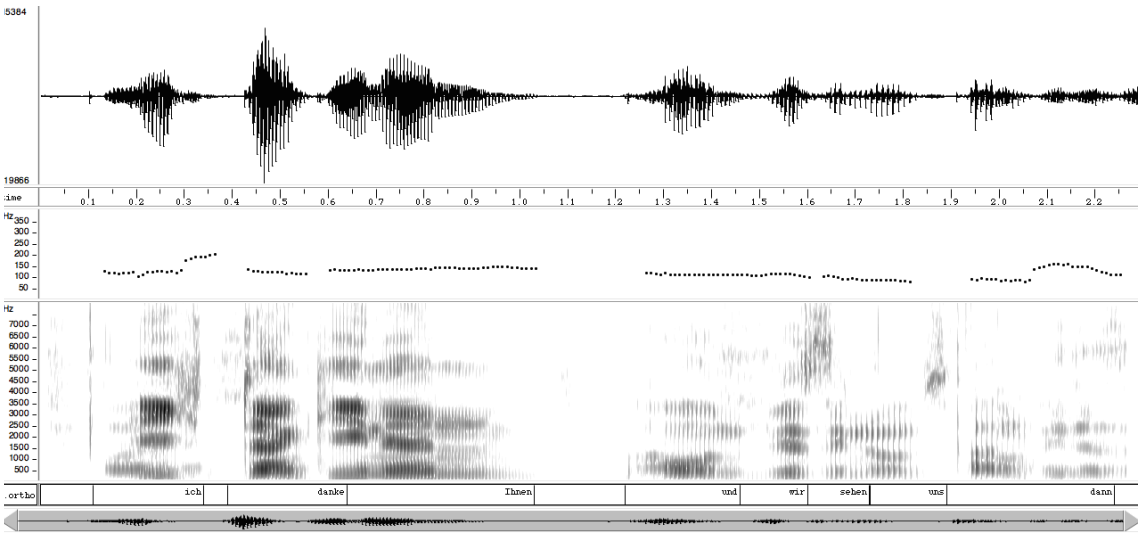


Figure 4.2: From the top of the figure: an acoustic waveform, pitch contour, spectrogram and transcription with gold alignment of the phrase *ich danke Ihnen und wir sehen uns dann*.

Finally, with the help the sound spectral representation acoustic vectors of features Y are created. One of the most commonly used for this purpose method is a mel frequency cepstrum (MFC). This method calculates coefficients representing the amplitudes of the resulting spectrum, so-called mel frequency cepstrum coefficients (MFCC). Resulting acoustic vectors of features are used as a representation of a sound waveform, forwarded to the decoder for recognition (Jurafsky and Martin 2009).

4.3 Acoustic Model

Acoustic model finds out the probability for an observing an acoustic vector Y given a sequence of words $P\ (Y \mid W)$. Theoretically, for a small vocabulary it can be found by sampling several variants of the words and computing statistical similarity of the inputs with the existing samples. However, when the vocabulary is very large we have to move down to the level of word subunits or phonemes.

Acoustic model contains statistical representation of each phoneme in the form a Hidden Markov Models (HMM), which abstracts the physical process of pronouncing of a particular phoneme. An example HMM is depicted in the figure 4.3. This HMM for phoneme recognition has 3 states S0, S1, S2. Letters 'A' shows the transition probabilities within the states, 'B' - output probabilities, 'Y' -observations. Sphinx-4 is a typical HMM-based speech recognizer with 3 and 5 states HMMs (Jurafsky and Martin 2009).

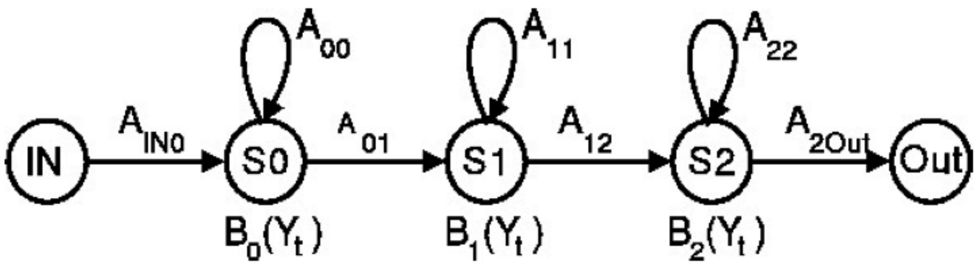


Figure 4.3: A typical HMM abstraction (Jurafsky and Martin 2009).

Word representations in acoustic model form a network of HMMs. Acoustic models are created from an hours-long audio recordings and their corresponding transcriptions. In order to get a statistical representation of phonemes, i. e, build a HMM for each phoneme, the speech corpus data is subjected to the training process. Speech corpus data and resulting models differs in speech parameters: microphone speech, broadcast speech, conversational speech, telephone speech. Too general models are not applicable for more specific language conditions, whereas too specific models are limited in its usage. The right choice of the acoustic model increases performance of a recogniser. To match specific conditions of an application an acoustic model model should be trained respectively, providing enough training data is available.

4.4 Language Model

Language model describes the probability of a word in a speech sequence Y P (W) and attempts to predict the next word in a sentence, depending on the prior context. Language models are based on n-gram sequences of words, where the number of words of the prior context is equal n-1. Language models are built on the basis of an application corpus (Jurafsky and Martin 2009). Depending on the application, language models can be either generalized or domain specific.

Sphinx-4 uses two types of language models: statistical and grammar-based ones. Statistical language model describes the probability of a word in a speech sequence (P (W)) and attempts to predict the next word in a sentence, depending on the prior context. Language models are based on n-gram sequences of words, where the number of words of the prior context is equal n-1. The most widely used sequence is tri-gram. An extract from a statistical language model is shown in the figure 4.4 .

Statistical language models are built on the basis of an application corpus. Corpus contains a list of sentences that are used to train the language model. The best results are achieved, when the model is tuned the application speech situation. Statistical language models for telephone conversations are less effective for robot instructions.

Grammars describe very simple types of languages in Java Speech Grammar Format (JSGF). The do not contain probabilities, but it is possible to weight some of the grammar elements. An example of a simple JSGF grammar for German digits is shown in the picture 4.5 .

N-gram grammar in Sphinx-4 is build from all the words, containing in the language model. By contrast aligner grammar allows to create a search graph, containing only one search path in accordance with the text be set for forced align-

```
...
\data\
ngram 1=4604
ngram 2=38548
ngram 3=80671
...
\3-grams:
...
-0.0512 wir sehen uns
-0.7566 wir senden Ihnen
-1.0576 wir sicher fertig
-1.0576 wir sicher im
-0.7566 wir sicherlich was
-1.3587 wir sie den
-1.3587 wir sie doch
-1.3587 wir sie hier
-1.3587 wir sie ja
-0.0941 wir sieben Uhr
...
```

Figure 4.4: An extract from Verbmobil language model for Sphinx-4.

```
grammar digits;

public <numbers> = ( eins | zwei | drei | vier | fünf |
 sechs |sieben | acht | neun | null) * ;
```

Figure 4.5: JSGF grammar for German digits from one to ten.

ment. For example, if we have a text *Ich danke Ihnen und wir sehen uns dann* we get a search graph, containing a starting node and and the nodes *ich, danke,Ihnen, und, wir sehen, uns, dann,* connected with each other in one simple path.

4.5 Dictionary

Pronunciation dictionary belongs to the knowledge base of Sphinx-4. It is made of all occurred in the application words and their transcriptions. Pronunciation dictionary is used by the decoder during word identification as a reference. A typical transcription file is shown in the figure 4.6.

The dictionary can be maid manually according to transcription conversions, or

```
spezifizieren   S p e t s i f i t s i: 6 n
grenz   g r E n t s
sechsund        z E k s U n t
Malediven       m a l e d i: v @ n
Gemütlicheres   g @ m y: t l I C @ r @ s
lau     l aU
zugereicht      t s u: g @ r aI j t
vornehmer       f o: 6 n e: m 6
gereist g @ r aI s t
Südfrankreich   z y: t f r a N k r aI C
zerteilen       t s E 6 t aI l n
```

Figure 4.6: An extract from the dictionary file. For correspondence between Sphinx-4 representation and Internation Phonetic Alphabet (IPA) see appendix A.

alternatively using generators. Some words in the dictionary can have several possible transcriptions. On the other hand, differently written words may sound identically (compare homophones TO and TWO). Difference between the homophones is only evident within context. Distinction of, for example, such homophones as ice-cream vs. I scream, real eyes vs. realize vs. real lies, addressed mail vs. a dressed male, etc. is a really challenging task for an ASR, especially when the context is not clear.

Words that are not in the dictionary are not recognized in Sphinx. So it is necessary either to use an extended dictionary, containing the missing words or to use sphinx grapheme to phomene converter g2p, which automatically generates pronunciations for missing words, based on pronunciation rules.

4.6 Decoder

Speech recognition decoder receives data from acoustic inputs, acoustic and language models and returns the most probable word sequence. The task that the decoder is solving can be formulated in plain text as follows:

What is the most likely sentence out of all sentences W in the language L given some acoustic input Y?

In the terms of mathematics we are dealing with an optimisation problem, formulated with a Bayes decision rule (Jurafsky and Martin 2009):

$$w_{opt} = argmaxP(W|Y) = \frac{P(Y|W)P(W)}{P(Y)} = argmaxP(Y|W)P(W)$$

Decoder solves the above optimisation problem, involving complicated search algorithms. One of the most commonly used for speech recognition algorithm is Viterbi algorithm. Viterbi is a dynamic programming search algorithm. It traverses the network of HMM states and finds the path with the best score.

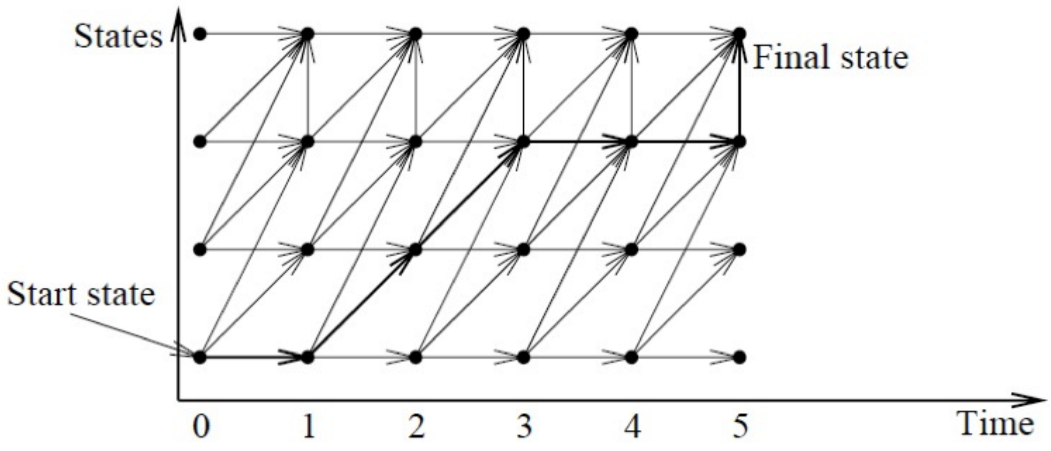


Figure 4.7: Viterbi search as dynamic programming (Ravishankar 1996).

Abstract representation of the Viterbi algorithm is shown in the in the figure 4.7. One axis represent states in the HMM network, another time. Arrows means transitions through the network. Each point in this 2-D space represents the best path probability for the corresponding state at that time. Every state has a best predecessor and starting from the final state and using backtracking it is possible to find the best path sequence for the whole search. Complexity of Viterbi algorithm is equal $\mathcal{O}(N2T)$, where N is the total number of states and T is the total duration (Ravishankar 1996). In Sphinx-4 Viterbi search is implemented as *token passing algorithm* (S.J. Young and Thornton 1989).

4.7 Search Graph and Linguist

Search graph and linguist are the central components of the decoder search module. The linguist is responsible for representing and managing the search space for the decoder. The role of the linguist is to provide, upon request, the search graph that is to be used by the decoder. The linguist is a generic interface that provides language model services.

Search graph is build in Sphinx-4 by Linguist on the basis of acoustic model, language model and dictionary. When the grammar or language model is changed the search graph is updated. An example of the search graph for the words "one" and "two" is shown in the figure 4.8.

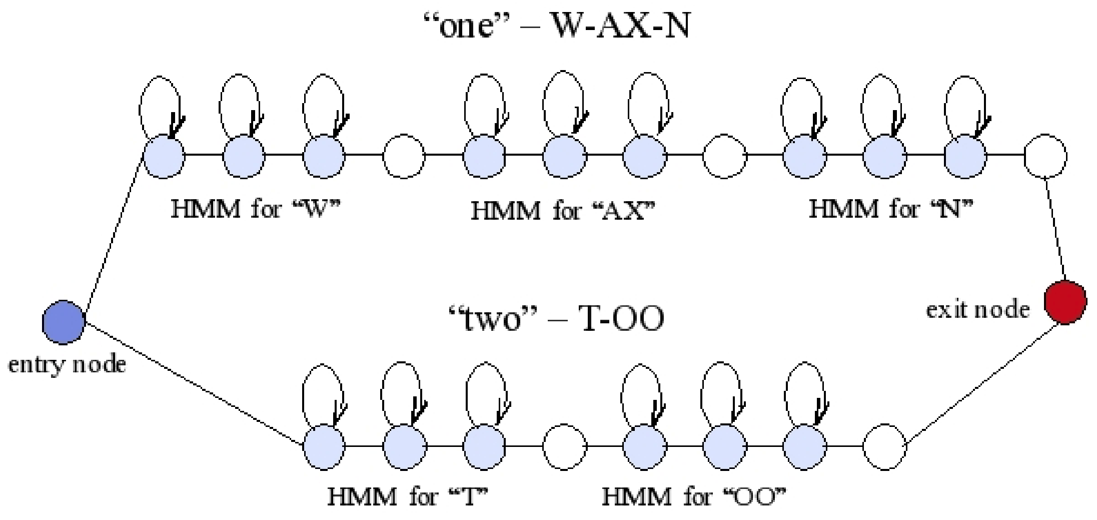


Figure 4.8: An example search graph for the words "one" and "two".

The main types of Linguist in Sphinx-4 are: flat linguist, dynamic flat linguist and lex tree linguist. Flat linguist is a simple version of Linguist, commonly used in combination with JSGF or aligner grammar. Dynamic flat linguist is a dynamic version of flat linguist, which is more efficient in the terms of memory usage and start up time. Lex tree linguist is also a dynamic version of linguist, appropriate for large vocabularies and using lexical trees, built from n-gram grammars.

5 Implementation

5.1 Development Environment Overview

Incremental recogniser is implemented in Java programming language, using an open-source project *Incremental Spoken Dialogue processing Toolkit* (InproTK), developed by the Universities of Potsdam, Bielefeld and Hamburg. InproTK includes modules for speech recognition, speech understanding, speech processing and dialogue management. Speech recognition module is based on implemented Java Sphinx-4 recogniser, a package of CMU Sphinx speech recognition toolkit.

5.2 IU Modules and Data Processing

Incremental Units (IUs) are minimal information units in InptoTK. IUs varies in the level abstraction (from low to high) and capable of describing diverse types of information (phonemes, words, phrases, ...). IUs of different abstraction levels form a IU network. An example of IU hierarchy is shown in the figure. Processing

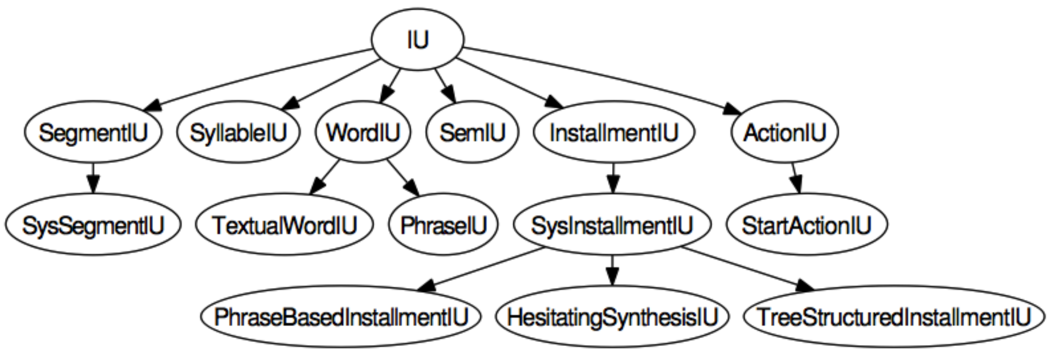


Figure 5.1: An excerpt of INproTk built-in type hierarchy (Baumann 2013).

of IUs is guarantied by IU modules. IU modules consist of left buffer, right buffer and processors. Different IUs modules are interconnected with each other, forming a pipeline of IU modules (Baumann 2013). A typical communication between two IU modules is shown in the figure 5.2.

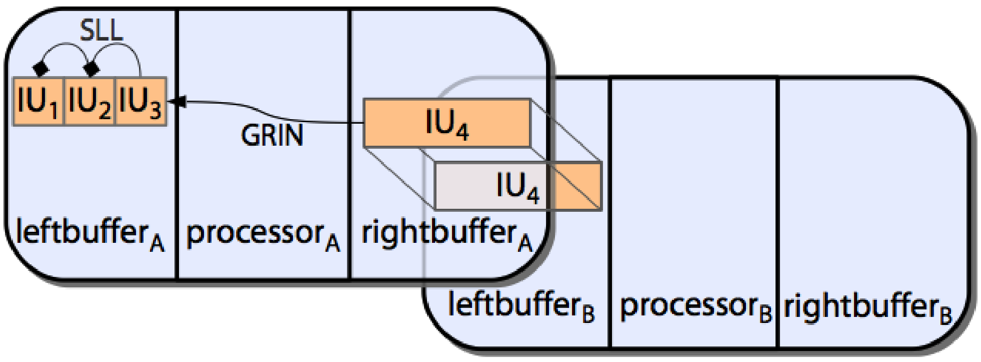


Figure 5.2: Incremental Modules and their intercommunication via left-right buffers (Baumann 2013).

5.3 INproTK IU Modules in Google+Sphinx Recognizer Architecture

5.3.1 Google IU Module

GoogleASR is a source IU module in InproTK, which listens to the results coming from Google's downstream connection in JavaScript Object Notation (JSON) format and passes results to its listeners by updating the current hypotheses and providing information about the hypotheses time (hypotheses start and end times).

Timing, implemented in Google ASR, has nothing to do with true alignment. With the help of this information it is only possible to calculate the timeliness: FO as well as FD. Manual reconstruction of alignment from timeliness is problematic as timeliness does not preserve any information about word duration.

For example, in the figure 5.3 the first hypothesis (word *ich*) equals 0 and end times equals about 7.5 time slots, whereas the borders of the word *ich* are between

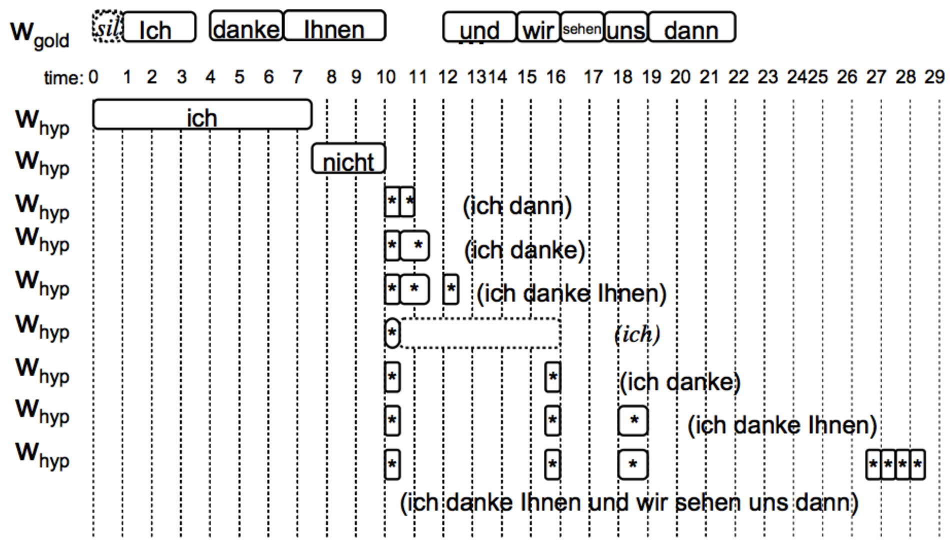


Figure 5.3: Incremental output of Google ASR with existing timing.

1 and 3.5 time slot. In addition, in case of Google we are dealing with additional output delay, resulted from network latency, which can be slightly variable approximately between 10 and 200 ms depending on the network configuration. Google has processed the file till the end of the word *ich* by the time 3.5, after some internal Google offset, the output was made. It is assumed, that Google offset reflects the problem of stability of the the output and latency (McGraw and Gruenstein 2012). Finally, Google ASR returned this result after a network delay.

```
2751     {"result":[{"alternative":[{"transcript":"ich danke Ihnen
und wir sehen uns dann","confidence":0.96583021},{"transcript":"ich
danke Ihnen und wir sehen uns danke"},{"transcript":"ich danke Ihnen
und wir sehen uns dann bitte"},{"transcript":"ich danke Ihnen und
wir sehen uns an"},{"transcript":"ich danke Ihnen und wir sehen uns
bald"},{"transcript":"ich danke Ihnen und wir sehn uns dann bitte"},
{"transcript":"ich danke Ihnen und wir sehn uns danke"},
{"transcript":"ich danke Ihnen und wir den uns danke"},
{"transcript":"ich danke Ihnen und wir in uns danke"},
{"transcript":"ich danke Ihnen unseren uns
danke"}],"final":true}],"result_index":0}
```

Figure 5.4: JSON file example

5.3.2 Sphinx ASR Module and Label Writer in INproTK

Sphinx ASR in INproTk is a module that receives results from Sphinx-4 recognizer, transforms them to IUs of recognized words and sends them to label writer IU. Label writer in its turn is responsible for output of the recognition results and their timing information in the console. Apart from that Label writer offers a configurable option to save the recognition and timing results to a file. An extract of such files can be found in the appendix B.

5.4 Architecture Overview of the Google+Sphinx Recognizer

Sphinx-Google architecture includes Sphinx-4 frontend, dictionary, language model and acoustic model, decoder with search graph, forced alignment module, Sphinx control thread, synchronous blocking queue and InproTK IUs: Google ASR, Sphinx ASR and label writer (see figure 5.5).

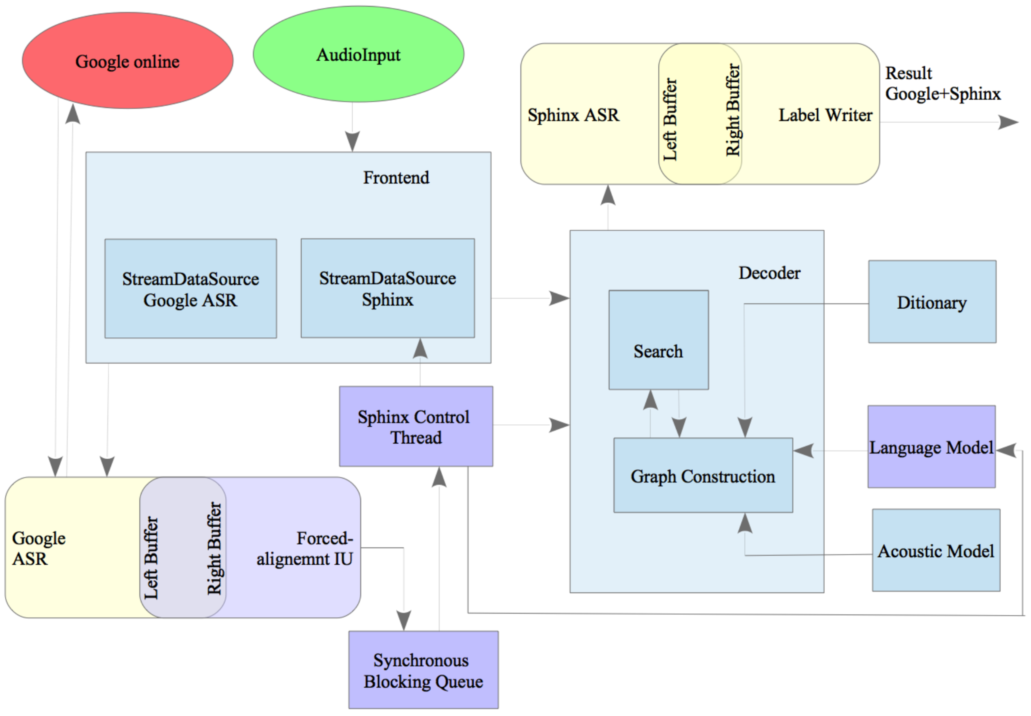


Figure 5.5: Implementation Overview.

As an audio input both Google ASR and Sphinx receive the same audio input. Input stream of Google ASR is set only once, whereas the length of the stream is equal to the whole audio. By contrast, the Sphinx input stream is updated every time, Google ASR returns a new hypothesis.

GoogleASR listens to JSON results coming from online Google, running in incremental mode, and passes result to forced alignment IU. Google ASR result is transferred to forced alignment IU via standard IU left-right buffer. Google ASR result contains hypothesis, and the start and end times, computed by Google ASR for every hypothesis.

Forced alignment IU puts every incremental Google result into synchronous blocking queue. Sphinx control thread takes the last Google hypothesis from the synchronous blocking queue and starts Sphinx decoding. Sphinx control thread updates language model, resetting text for alignment. As soon as the grammar of the language model is changed, search graph of the decoder module is rebuild anew.

5.5 Forced Alignment of Google Incremental Results with Google+Sphinx Recognizer

Every time a new hypothesis is taken from the queue Sphinx input stream is also reset. Analysis of Google JSON files, containing the hypotheses times of Google, show that this time can be used to calculate the length of the input stream to be updated. In the GoogleASR INproTK module output (see figure 5.3) Google hypotheses time equals the end times of the word IU, put to the left buffer.

In the example, shown in the figure 5.3, the first hypothesis *ich* came from Google at 750 ms. By this time in audio the words *ich danke I…* are already pronounced. If we subtract network latency from the hypotheses time, than we know that that Google was ready with its hypothesis several dozens milliseconds earlier. In the case when latency is equals 100 ms, that is 650 ms. However, 650 ms is still far from the true end of the word *ich* in audio (350 ms). This difference of 300 ms is actually the latency time Google needs to come to a confident result, before it produces the output. Under this consideration the following assumption is made: Google hypothesis time (t_{hyp}) consists of a time till the matched word end (t_{we})

im audio plus Google offset (t_{offset}) and finally network latency (t_{lat}):

$$t_{hyp} = t_{we} + t_{offset} + t_{lat} \tag{5.1}$$

With the first output Google waits 650 ms, the next outputs come after 0 to 850 ms relative to the previous output.

In order to get correct alignment from Sphinx for first Google result *ich* it is enough to restart Sphinx with the grammar, containing the word *ich* and piece of audio between 350 ms and 650 ms. Alignment will also function if the whole input stream is resetted with every new hypotheses. However, with such approach the processing time will grow quadratically (?) and the quality of alignment will decrease. So the more accurate file length matches the transcription the better are the alignment results.

5.5.1 Forced Alignment Mode of the Google+Sphinx Recognizer

Sphinx-4 in its standard implementation is able to do either alignment or recognition. Sphinx-4 Linguist example configuration, required for alignment, is shown in the figure 5.6. As it is seen from the example in the Google+Sphinx recognizer uses dynamic flat linguist and aligner grammar.

```
...
<component name="linguistFA" type=
"edu.cmu.sphinx.linguist.dflat.DynamicFlatLinguist">
        <property name="logMath" value="logMath"/>
       <property name="grammar" value="forcedAligner"/>
        <property name="acousticModel" value="acousticModel"/>
        <property name="wordInsertionProbability" value=
        "${wordInsertionProbability}"/>
        <property name="silenceInsertionProbability" value=
        "${silenceInsertionProbability}"/>
        <property name="languageWeight" value="${languageWeight}"/>
        <property name="unitManager" value="unitManager"/>
    </component>
...
```

Figure 5.6: An extract from the configuration file for Google+Sphinx recognizer in the alignment mode.

Timing results of Sphinx, started after Google, are shown in the figure below (5.7):

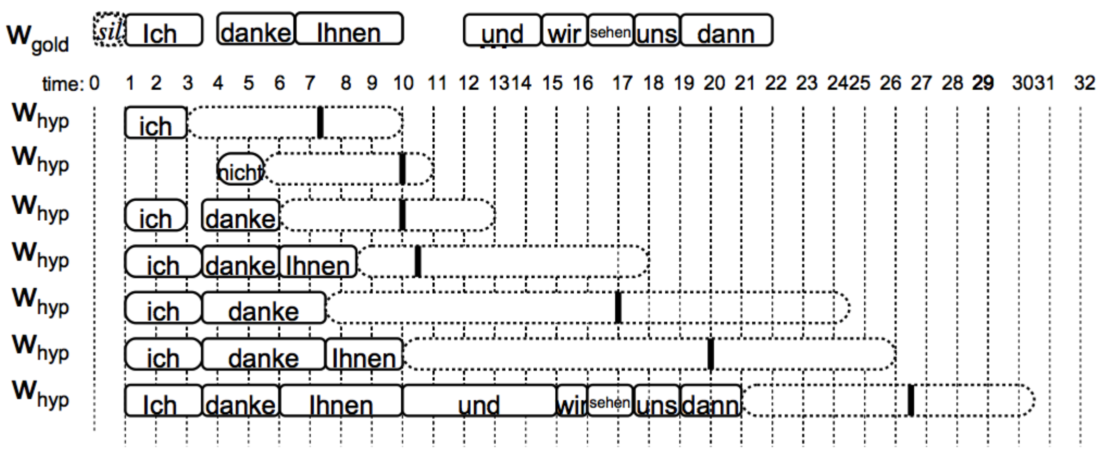

W_{gold}
sil
Ich
danke
Ihnen
und
wir
sehen
uns
dann
time: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
W_{hyp}
ich
nicht
danke
Ihnen
und
wir
sehen
uns
dann

Figure 5.7: Google+Sphinx output with improved timing. Vertical lines show the time, by which Google was ready with the corresponding hypothesis. Right borders of the dotted boxes show at what point of time Sphinx was ready with its alignment.

Comparison of visualization results with the corresponding Google output (5.3) demonstrates qualitative timing improvement in terms alignment quality, whereas the WER remains unchanged, as the output contains the same final result as that of Google alone. The number of hypotheses is smaller, because Sphinx gets incremental results for processing only when forced alignment of one result is finished. If during Sphinx processing time Google produce additional hypotheses, these hypotheses are skipped and only the last one is taken for new processing.

However, an obvious disadvantage of this alignment approach is additional latency produced by Sphinx during alignment. Alignment task, where the search path is known and defined is less time consuming as recognizing, demands nevertheless computation time. Computation time depends on the processing power. If the Sphinx processing time could have been neglected, than we get an model of Google+Sphinx recognizer, comparable to Google in WERs and having an improved timing.

5.5.2 Alignment + Recognition Mode of the Google+Sphinx Recognizer

In the second approach Sphinx returns not only alignment result, based on Google ASR, but continues with recognition, while waiting for the new Google incremental result. This approach solves several tasks. First, we get more precise alignment in case of excessive input stream. Moreover, excessive input stream is even desirable as only additional frames guarantees recognition start after alignment is finished. Second, Sphinx produces some results with better timeliness, if it has finished alignment and is waiting for new Google hypotheses. Third, it possible to use the Google result to manipulate the search path in search graph of Sphinx and theoretically change to a path proposed by Google, if this path exist.

Configuration of Google+Sphinx recognizer in alignment and recognition mode uses dynamic flat linguist, because of the advantages, described in the chapter 4. As grammars the following types were tested: JSGF grammar and n-gram grammar. However, by contrast with standard grammars in Sphinx these grammars are combined with alignment grammar in their implementation.

As language models two types of language models were available. The first one is verbmobil language model, generated from the selected number of texts from verbmobil domain with 4604 1-grams,38548 2-grams and 80671 3-grams. The second one is a simplified language model, generated only from the transcriptions, contained in the test data with 146 1-grams, 248 2-grams and 342 3-grams. The simplified language model is also runnable in combination with flat linguist. Whereas with the flat linguist and the bigger language model the application runs out of memory.

The grammars are combined in the following way. The first node is starting node. After the starting node there is a path, consisting of nodes, built from a set text (see figure 5.9). The last alignment node is connected to multiple paths in the second part of the graph, build from n-gram or JSGF grammar. The last node of the graph is the end node.

```
...
      <component name="linguistFA"type=
      "edu.cmu.sphinx.linguist.dflat.DynamicFlatLinguist">
      <property name="logMath" value="logMath"/>
        <property name="grammar" value="ngramGrammar"/>
        <!-- <property name="grammar" value=
        "myjsgfGrammar"/>-->
        <property name="acousticModel" value=
        "acousticModel"/>
        <property name="wordInsertionProbability" value=
        "${wordInsertionProbability}"/>
        <property name="silenceInsertionProbability" value=
        "${silenceInsertionProbability}"/>
        <property name="languageWeight" value=
        "${languageWeight}"/>
        <property name="unitManager" value="unitManager"/>
    </component>
...
```

Figure 5.8: An extract from the configuration file for Google+Sphinx Recognizer in alignment and recognition mode.

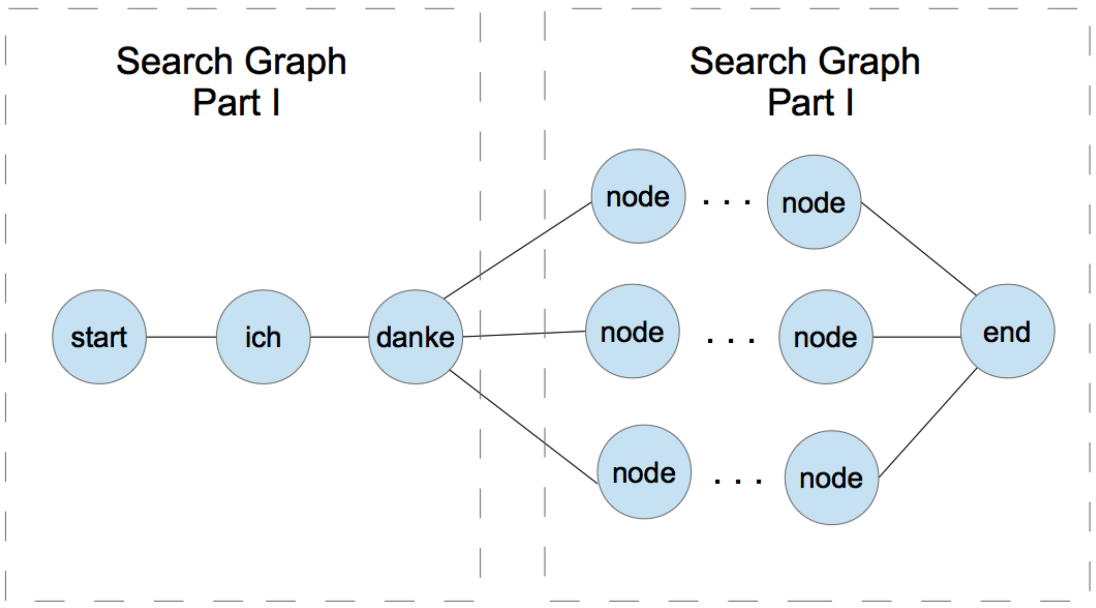


Figure 5.9: Schematic representation of the search graph of Google+Sphinx recognizer in combined mode. In the example the text set for alignment is *ich danke*.

5.5.3 Alignment + Recognition with JSGF Grammar

To demonstrate that alignment and recognition are able to be combined any type of JSGF grammar can be used. However, as JSGF grammar describe relative small command languages, we used a phone loop grammar in JSGF format, suitable for more universal tasks.

An example of such grammar is shown in the figure 5.10

```
grammar phones;

public <phones> = (2: | 6 | 9 | @ | C | E | E: | I | N | O |
OY | Q |S | SIL | U | Y | Z  | a | a: | aI | aU | b | d |
e |  e: | f | g |h | i | i: |j | k | l | m | n | o | o: |
p| r | s |  t | u | u: | v | x | y: | z)*;
```

Figure 5.10: An extract from the dictionary file.

Described phone-loop grammar is prototypical and uses no weights. Result of the recognition of audio with transcription *ich danke ignen und wir sehen uns dann*, using the phone-loop grammar 5.10 are shown in the figure 5.11. Combination of alignment grammar with phone-loop JSGF grammar shows, that it is possible to get some faster results from Sphinx, after the Google path is already finished. At the moment the new result from Google is not yet considered, because it is either not there due to latency or because the processing of the audio is still running.

. . . here also some comments to the picture above

However, an obvious disadvantage of the above described approach that it is not possible to get use of the already recognized result, produced by Google as two grammars (alignment and JSGF) are practically independent from each other. Google produce always finished words or phrases as incremental output, followed by silence. If the Google hypotheses were unfinished words it would possible to look in the JSGF Grammar for the most possible words endings. In our case however,the only way to improve the recognition quality in the combination is to improve the quality of phone-loop grammar output. For future there should be considered a possibility to develop a more sophisticated grammar, based on phones frequency and alternation rules generally in the language and in the selected vocabulary in particular.

```
0.000    0.110    <sil>
0.110    0.340    ich
0.340    0.610    danke
0.610    1.050    ihnen
1.050    1.190    <sil>
1.190    1.230    r
1.230    1.290    o
1.290    1.400    o:
1.400    1.450    n
1.450    1.520    m
1.520    1.560    i
1.560    1.610    l
1.610    1.650    s
1.650    1.690    @
1.690    1.730    n
1.730    1.800    o
1.800    1.840    n
1.840    1.890    s
1.890    1.930    h
1.930    1.980    a
1.980    2.050    n
2.050    2.090    b
2.090    2.140    i
2.140    2.180    m
2.180    2.220    o
2.220    2.280    v
2.280    2.290    e
```

Figure 5.11: Alignment + Recognition result, using combined grammar, based on fixed input "ich danke ihnen" and phone-loop JSGF grammar for the audio, having transcription "ich danke ihnen und wir sehen uns dann" (5.11).

5.5.4 Alignment + Recognition with N-Gram Grammar

The second approach to the combination of Google+Sphinx in alignment and recognition mode presupposes the use of the implemented mixed grammar, based on standard alignment and n-gram grammar of CMU Sphinx. The search graph built from this grammar looks as depicted in the figure 5.9, whereas the first part of the graph is build from the set text string and the second part of the graph is created on the basis of the n-gram grammar, loaded from the default language model.

In the first case, when in the second part of the graph there is no path, proposed by Google, the last node of first part of the graph is connected with the starting node in the second search graph. In the second case, when there is a path, proposed by Google, the last node in the first part of the graph is deleted and second last node is connected to the a branch node in the second part of the graph, that equals to the deleted last node in the first part of the graph.

In the present implementation when new text string is set the condition is checked if the whole path, proposed by the Google hypotheses exists in the second part of the graph, build from n-gram grammar. For example, if the set text equals *ich danke* the next node could be the first node of the whole second graph or the node ich is connected to the node *danke* in the second search graph. Additionally the existence of the whole path *ich danke* in the graph is checked.

Result of the recognition of audio with transcription *ich danke ihnen und wir sehen uns dann*, using n-gram grammar is shown in the figure 5.12.

```
0.000   0.110   <sil>
0.110   0.300   ich
0.300   0.400   sage
0.400   0.610   danke
0.610   1.020   ihnen
1.020   1.470   und
1.470   1.590   wir
1.590   1.750   sehen
1.750   1.880   uns
1.880   2.290   dann>
```

Figure 5.12: Alignment + Recognition result, using combined grammar, based on fixed input "ich sage danke" and n-gram grammar and simplified language model for the audio, having transcription "ich danke ihnen und wir sehen uns dann" (5.11).

6 Results and Evaluation

6.1 Testing Approach

For testing purposes a corpus of Verbmobil-Data (dialogues about meeting appointments) is available. Each record contains audio file, used by the recognizer as input in one of the configurations, and *gold standard* with the true words from the audio, including true alignments. Depending on the program configuration the following types of outputs were available for evaluation:

1. Google incremental output (1).

2. Sphinx incremental output: Sphinx in recognition mode, using verbmobil or simplified language language model (2).

3. Google+Sphinx: Sphinx in forced alignment mode after Google incrementally, alignment of the incremental results as described in the chapter 5 (3).

4. Google+Sphinx: Sphinx in forced alignment and recognition mode, combined alignment and phone loop grammar (4).

5. Google+Sphinx: Sphinx in forced alignment and recognition mode,combined alignment grammar (5).

Alignment quality and timeliness were evaluated with the help of InTELiDA (Incremental Timing Evaluation of Linguistic Data), providing *perl* modules and a graphical user interface for incremental data analysis (Baumann 2013). For alignment quality analyses a Perl program, that computes the *mean, standard deviation and RMSE* as described in the chapter 3 was used. Timeliness metrics FO and FD were evaluated, using graphical application of InTELiDA - *interactivetool.pl*.

Table 6.1: Alignments quality of the recognizer for matches and substitutions in different configurations.

Recognizer	mean	stddev	RMSE
Google (1)	1398 ms	775 ms	1672 ms
Sphinx (2)	154 ms	167 ms	238 ms
Google+Sphinx (3)	348 ms	308 ms	387 ms

Table 6.2: Alignments quality of the recognizer only for matches in different configurations.

Recognizer	mean	stddev	RMSE
Google (1)	1471 ms	680 ms	1735 ms
Sphinx (2)	167 ms	360 ms	320 ms
Google+Sphinx (3)	288 ms	212 ms	401 ms

6.2 Test Results

6.2.1 Alignment quality

Results of the alignment quality analyses are shown in the following tables. For analyses of the timing quality only Google+Sphinx in the forced alignment mode was used. The first table 6.2.1 shows results for matches and substitutions, whereas the table 6.2.1 shows results only for matches. Evaluation results for single files could be found in the appendix **??**.

The quality of alignment, using Google+Sphinx has at least 400 % increased, whereas the the errors for *mean, standard deviation and RMSE* are within the range of 400 ms. The results of Google+Sphinx are comparable with the results of Sphinx alone. The small difference between Sphinx+Google and Google is explained by the fact that the length of audio input, passed to Sphinx together with Google hypotheses is not always precisely computed. For future there should be considered a more sophisticated algorithm to compute the offset of Google.

Table 6.3: Incremental alignments quality of the recognizer in different configurations.

Recognizer	mean	stddev	RMSE
Google (1)	ms	ms	ms
Sphinx (2)	ms	ms	ms
Google+Sphinx (3)	ms	ms	ms

Table 6.4: Timeliness quality of the recognizer in different configurations. FO results.

Recognizer	mean	stddev	median
Google (1)	3202 ms	3744 ms	1466 ms
Sphinx (2)	175 ms	445 ms	130 ms
Google+Sphinx (3)	2742 ms +offset	3787 ms + offset	1000 ms + offset
Google+Sphinx (5)	ms		

6.2.2 Timeliness quality

Results of the timeliness quality analyses are shown in the tables below. The first table 6.2.2 shows results for FO, whereas the table 6.2.2 shows results for FD.

6.3 Tests Evaluation

Table 6.5: Timeliness quality of the recognizer in different configurations. FD results.

Recognizer	mean	stddev	median
Google (1)	3702 ms	4263 ms	1705 ms
Sphinx (2)	213 ms	445 ms	150 ms
Google+Sphinx (3)	2857 ms + offset	3788 ms + offset	1259 ms + offset
Google+Sphinx (5)	ms		

7 Discussion

. . . to be done I don't really know what I should write here.

7.1 Conclusion

This master thesis has presented a prototypical solution for incremental speech recognizer combination of Google and Sphinx.

Incremental speech recognizer is possible to run in 2 modes: alignment only and alignment + recognition. Incremental results, coming from Google, have been aligned using forced alignment IU module and Sphinx ASR. Evaluation of alignment results quality have shown that Google + Sphinx combination produces missing in Google alone timing information. The quality of Google+Sphinx alignment is comparable to the alignment of the Sphinx alone. Under the assumption that processing time of Sphinx, which depends on computational power, could have been neglected, Google+Sphinx in the alignment mode is competitive with Google in its performance.

To overcome the Google latency two 2 approaches have been tested. The first one uses a combination of alignment and JSGF grammar. The second one combines the alignment and n-gram grammars. In both cases the search graph consists of two connected parts. The first part represent results, coming from Google. The second contains the nodes, reconstructed from the corresponding grammar types. Moreover the tests have proved that, using a n-gram grammar it is possible to reduce the search space in the second part of the graph by modifying the search path, depending on Google hypotheses.

Finally, timeliness of Google, Sphinx and Google+Sphinx results have been analysed. Evaluation has shown that Google+Sphinx in alignment+recognition mode has shown an improvement in comparison to Google alone and Google+Sphinx in alignment mode. Timeliness quality growth in alignment+recognition mode is explained by the possibility of Google+Sphinx to deliver intermediate recognition results while waiting for Google hypotheses.

7.2 Future Prospects

. . . to be done Timo, probably you could give me some hints.

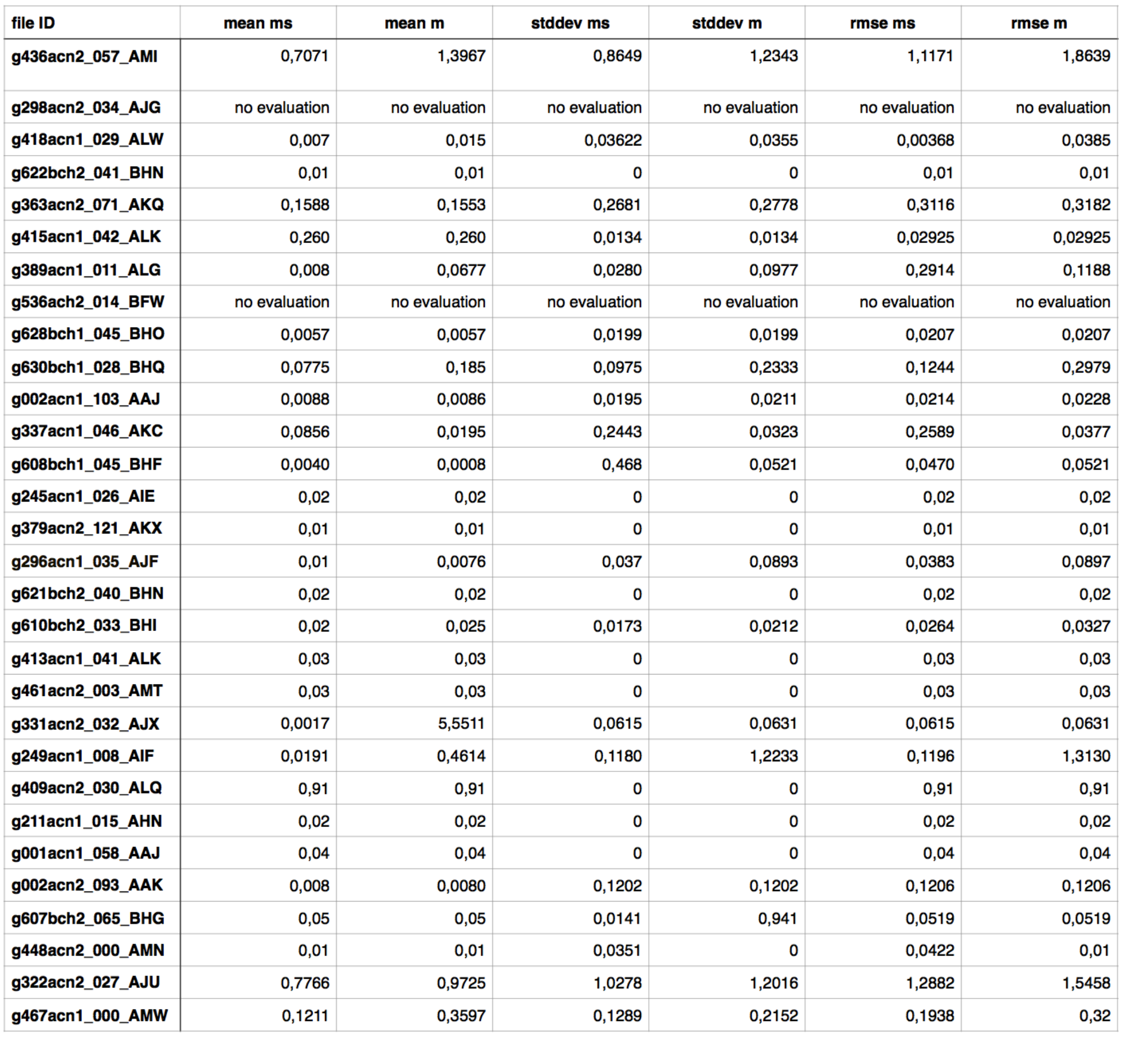
A Phonetic Alphabet Notation (IPA)

Sphinx dictionary notation	German phoneme in IPA notation
2:	ʌː
6	ɒ
9	ɘ
C	ç
E	ɛ
E:	ɛː
I	ɪ
N	ŋ
O	ɔ
OY	ɔʏ
Q	ʕ
S	ʃ
U	ʊ
Y	ʏ
Z	ʒ
a	a
a:	aː
aI	aɪ
aU	aʊ
b	b
d	d
e	e

e:	eː
f	f
g	g
h	h
i	i
i:	i
j	j
k	k
l	l
m	m
n	n
o	o
o:	oː
p	p
r	r
s	s
t	t
u	u
u:	uː
v	v
x	x
y:	yː
z	z

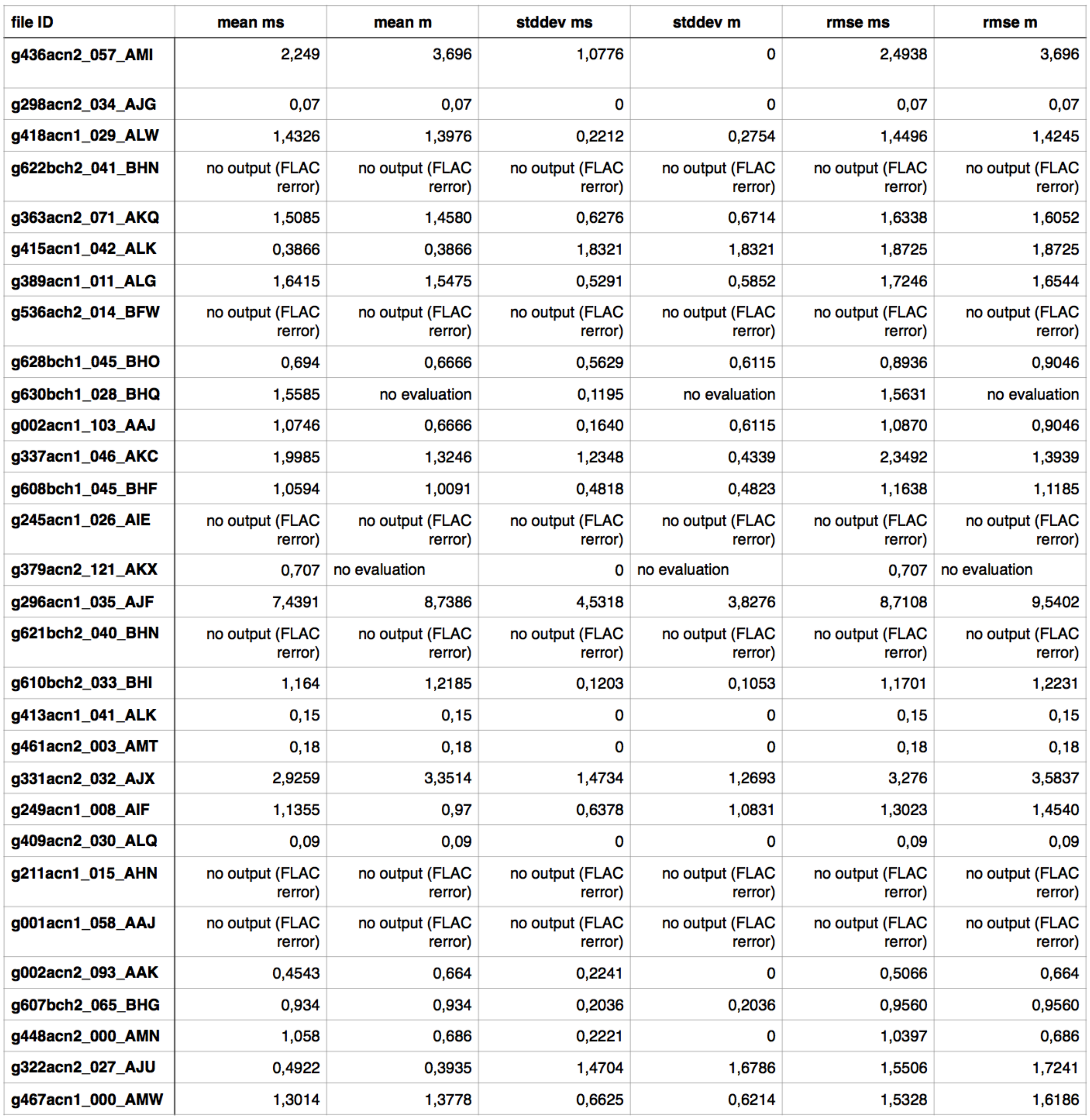
B Alignment Evaluation Results

This section contains results of alignment evaluation for matches as well as for matches and substitutions for Sphinx, Google and Google+Sphinx.



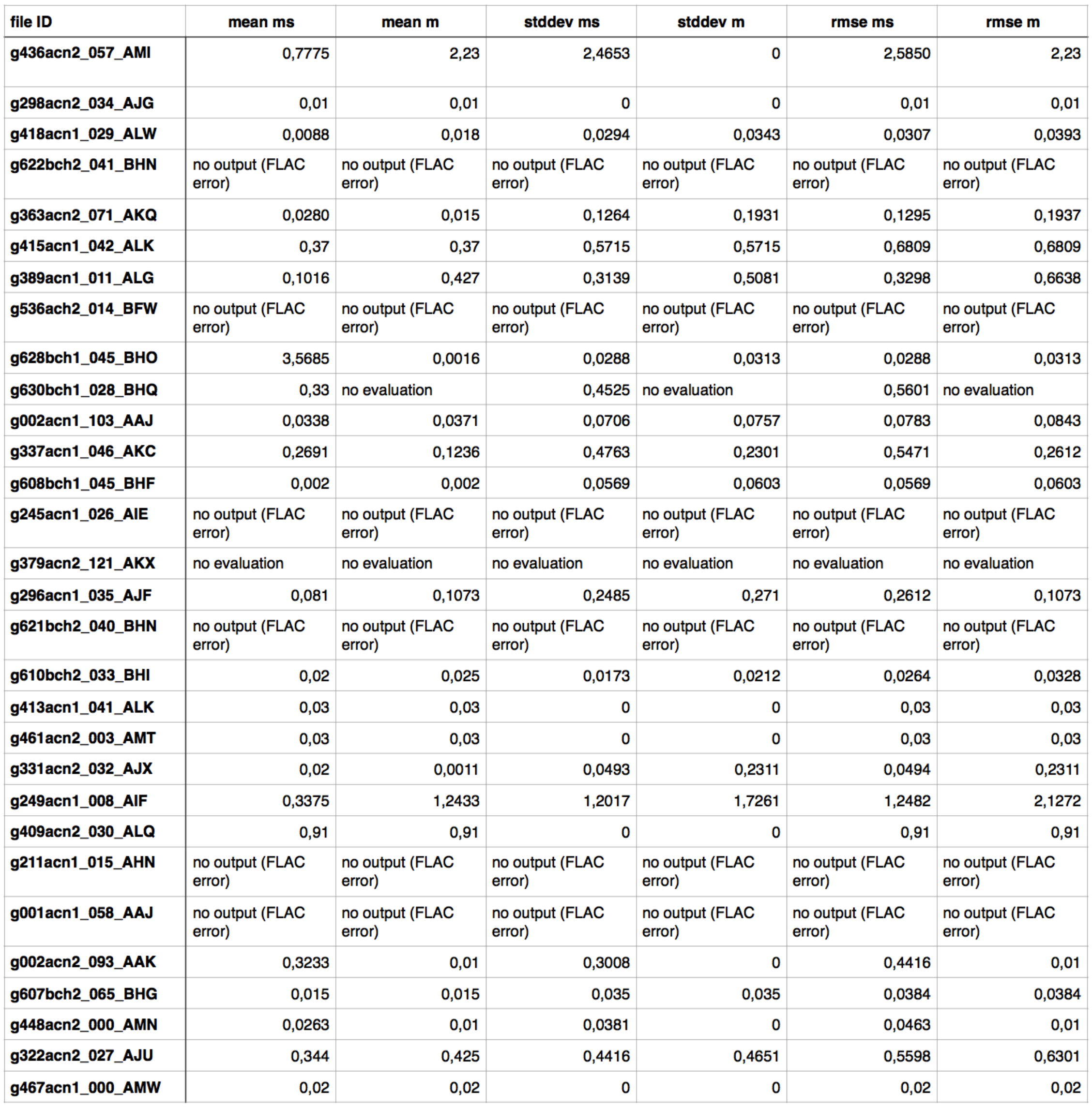
file ID	mean ms	mean m	stddev ms	stddev m	rmse ms	rmse m
g436acn2_057_AMI	0,7071	1,3967	0,8649	1,2343	1,1171	1,8639
g298acn2_034_AJG	no evaluation	no evaluation	no evaluation	no evaluation	no evaluation	no evaluation
g418acn1_029_ALW	0,007	0,015	0,03622	0,0355	0,00368	0,0385
g622bch2_041_BHN	0,01	0,01	0	0	0,01	0,01
g363acn2_071_AKQ	0,1588	0,1553	0,2681	0,2778	0,3116	0,3182
g415acn1_042_ALK	0,260	0,260	0,0134	0,0134	0,02925	0,02925
g389acn1_011_ALG	0,008	0,0677	0,0280	0,0977	0,2914	0,1188
g536ach2_014_BFW	no evaluation	no evaluation	no evaluation	no evaluation	no evaluation	no evaluation
g628bch1_045_BHO	0,0057	0,0057	0,0199	0,0199	0,0207	0,0207
g630bch1_028_BHQ	0,0775	0,185	0,0975	0,2333	0,1244	0,2979
g002acn1_103_AAJ	0,0088	0,0086	0,0195	0,0211	0,0214	0,0228
g337acn1_046_AKC	0,0856	0,0195	0,2443	0,0323	0,2589	0,0377
g608bch1_045_BHF	0,0040	0,0008	0,468	0,0521	0,0470	0,0521
g245acn1_026_AIE	0,02	0,02	0	0	0,02	0,02
g379acn2_121_AKX	0,01	0,01	0	0	0,01	0,01
g296acn1_035_AJF	0,01	0,0076	0,037	0,0893	0,0383	0,0897
g621bch2_040_BHN	0,02	0,02	0	0	0,02	0,02
g610bch2_033_BHI	0,02	0,025	0,0173	0,0212	0,0264	0,0327
g413acn1_041_ALK	0,03	0,03	0	0	0,03	0,03
g461acn2_003_AMT	0,03	0,03	0	0	0,03	0,03
g331acn2_032_AJX	0,0017	5,5511	0,0615	0,0631	0,0615	0,0631
g249acn1_008_AIF	0,0191	0,4614	0,1180	1,2233	0,1196	1,3130
g409acn2_030_ALQ	0,91	0,91	0	0	0,91	0,91
g211acn1_015_AHN	0,02	0,02	0	0	0,02	0,02
g001acn1_058_AAJ	0,04	0,04	0	0	0,04	0,04
g002acn2_093_AAK	0,008	0,0080	0,1202	0,1202	0,1206	0,1206
g607bch2_065_BHG	0,05	0,05	0,0141	0,941	0,0519	0,0519
g448acn2_000_AMN	0,01	0,01	0,0351	0	0,0422	0,01
g322acn2_027_AJU	0,7766	0,9725	1,0278	1,2016	1,2882	1,5458
g467acn1_000_AMW	0,1211	0,3597	0,1289	0,2152	0,1938	0,32

Figure B.1: Sphinx alignment quality for single files.



file ID	mean ms	mean m	stddev ms	stddev m	rmse ms	rmse m
g436acn2_057_AMI	2,249	3,696	1,0776	0	2,4938	3,696
g298acn2_034_AJG	0,07	0,07	0	0	0,07	0,07
g418acn1_029_ALW	1,4326	1,3976	0,2212	0,2754	1,4496	1,4245
g622bch2_041_BHN	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)
g363acn2_071_AKQ	1,5085	1,4580	0,6276	0,6714	1,6338	1,6052
g415acn1_042_ALK	0,3866	0,3866	1,8321	1,8321	1,8725	1,8725
g389acn1_011_ALG	1,6415	1,5475	0,5291	0,5852	1,7246	1,6544
g536ach2_014_BFW	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)
g628bch1_045_BHO	0,694	0,6666	0,5629	0,6115	0,8936	0,9046
g630bch1_028_BHQ	1,5585	no evaluation	0,1195	no evaluation	1,5631	no evaluation
g002acn1_103_AAJ	1,0746	0,6666	0,1640	0,6115	1,0870	0,9046
g337acn1_046_AKC	1,9985	1,3246	1,2348	0,4339	2,3492	1,3939
g608bch1_045_BHF	1,0594	1,0091	0,4818	0,4823	1,1638	1,1185
g245acn1_026_AIE	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)
g379acn2_121_AKX	0,707	no evaluation	0	no evaluation	0,707	no evaluation
g296acn1_035_AJF	7,4391	8,7386	4,5318	3,8276	8,7108	9,5402
g621bch2_040_BHN	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)
g610bch2_033_BHI	1,164	1,2185	0,1203	0,1053	1,1701	1,2231
g413acn1_041_ALK	0,15	0,15	0	0	0,15	0,15
g461acn2_003_AMT	0,18	0,18	0	0	0,18	0,18
g331acn2_032_AJX	2,9259	3,3514	1,4734	1,2693	3,276	3,5837
g249acn1_008_AIF	1,1355	0,97	0,6378	1,0831	1,3023	1,4540
g409acn2_030_ALQ	0,09	0,09	0	0	0,09	0,09
g211acn1_015_AHN	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)
g001acn1_058_AAJ	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)	no output (FLAC rerror)
g002acn2_093_AAK	0,4543	0,664	0,2241	0	0,5066	0,664
g607bch2_065_BHG	0,934	0,934	0,2036	0,2036	0,9560	0,9560
g448acn2_000_AMN	1,058	0,686	0,2221	0	1,0397	0,686
g322acn2_027_AJU	0,4922	0,3935	1,4704	1,6786	1,5506	1,7241
g467acn1_000_AMW	1,3014	1,3778	0,6625	0,6214	1,5328	1,6186

Figure B.2: Google alignment quality for single files.



file ID	mean ms	mean m	stddev ms	stddev m	rmse ms	rmse m
g436acn2_057_AMI	0,7775	2,23	2,4653	0	2,5850	2,23
g298acn2_034_AJG	0,01	0,01	0	0	0,01	0,01
g418acn1_029_ALW	0,0088	0,018	0,0294	0,0343	0,0307	0,0393
g622bch2_041_BHN	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)
g363acn2_071_AKQ	0,0280	0,015	0,1264	0,1931	0,1295	0,1937
g415acn1_042_ALK	0,37	0,37	0,5715	0,5715	0,6809	0,6809
g389acn1_011_ALG	0,1016	0,427	0,3139	0,5081	0,3298	0,6638
g536ach2_014_BFW	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)
g628bch1_045_BHO	3,5685	0,0016	0,0288	0,0313	0,0288	0,0313
g630bch1_028_BHQ	0,33	no evaluation	0,4525	no evaluation	0,5601	no evaluation
g002acn1_103_AAJ	0,0338	0,0371	0,0706	0,0757	0,0783	0,0843
g337acn1_046_AKC	0,2691	0,1236	0,4763	0,2301	0,5471	0,2612
g608bch1_045_BHF	0,002	0,002	0,0569	0,0603	0,0569	0,0603
g245acn1_026_AIE	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)
g379acn2_121_AKX	no evaluation	no evaluation	no evaluation	no evaluation	no evaluation	no evaluation
g296acn1_035_AJF	0,081	0,1073	0,2485	0,271	0,2612	0,1073
g621bch2_040_BHN	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)
g610bch2_033_BHI	0,02	0,025	0,0173	0,0212	0,0264	0,0328
g413acn1_041_ALK	0,03	0,03	0	0	0,03	0,03
g461acn2_003_AMT	0,03	0,03	0	0	0,03	0,03
g331acn2_032_AJX	0,02	0,0011	0,0493	0,2311	0,0494	0,2311
g249acn1_008_AIF	0,3375	1,2433	1,2017	1,7261	1,2482	2,1272
g409acn2_030_ALQ	0,91	0,91	0	0	0,91	0,91
g211acn1_015_AHN	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)
g001acn1_058_AAJ	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)	no output (FLAC error)
g002acn2_093_AAK	0,3233	0,01	0,3008	0	0,4416	0,01
g607bch2_065_BHG	0,015	0,015	0,035	0,035	0,0384	0,0384
g448acn2_000_AMN	0,0263	0,01	0,0381	0	0,0463	0,01
g322acn2_027_AJU	0,344	0,425	0,4416	0,4651	0,5598	0,6301
g467acn1_000_AMW	0,02	0,02	0	0	0,02	0,02

Figure B.3: Google+Sphinx alignment quality for single files.

C Examples of Google+Sphinx Recognizer Output Files

C.1 Sphinx Recognition Results

```
Time: 0.03
0.000   0.030   <s>

Time: 0.16
0.000   0.150   <s>
0.000   0.160   oh

Time: 0.18
0.000   0.150   <s>
0.000   0.100   <sil>
0.100   0.180   ja

Time: 0.19
0.000   0.150   <s>
0.000   0.190   oh

Time: 0.21
0.000   0.150   <s>
0.000   0.100   <sil>
0.100   0.210   und

Time: 0.55
0.000   0.150   <s>
0.000   0.100   <sil>
0.100   0.350   und
```

```
0.350    0.380    <sil>
0.380    0.550    dann

Time: 0.56
0.000    0.150    <s>
0.000    0.100    <sil>
0.100    0.560    und

Time: 0.57
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.570    denke

Time: 0.61
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.610    denke

Time: 0.89
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    0.890    innen

Time: 1.10
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.100    <sil>
```

```
Time: 1.23
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.230    <sil>

Time: 1.31
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.310    ja

Time: 1.33
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.330    oh

Time: 1.42
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
```

```
1.050    1.180    <sil>
1.180    1.420    und

Time: 1.49
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.490    und

Time: 1.51
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.510    unten

Time: 1.72
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.590    unten
1.590    1.720    sind

Time: 1.82
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
```

```
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.590    unten
1.590    1.750    sind
1.750    1.820    und

Time:  1.86
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.590    unten
1.590    1.750    sind
1.750    1.860    und

Time:  1.89
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.590    unten
1.590    1.750    sind
1.750    1.890    uns

Time:  1.95
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
```

```
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.590    unten
1.590    1.750    sind
1.750    1.950    unser

Time: 2.03
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.590    unten
1.590    1.750    sind
1.750    2.030    unser'n

Time: 2.10
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
1.050    1.180    <sil>
1.180    1.590    unten
1.590    1.750    sind
1.750    2.060    unser'n
2.060    2.100    <sil>

Time: 2.11
0.000    0.150    <s>
0.000    0.110    <sil>
0.110    0.350    ich
0.350    0.390    <sil>
0.390    0.590    denke
0.590    1.050    innen
```

```
1.050     1.180     <sil>
1.180     1.590     unten
1.590     1.750     sind
1.750     2.110     unser'n

Time:  2.26
0.000     0.150     <s>
0.000     0.110     <sil>
0.110     0.350     ich
0.350     0.390     <sil>
0.390     0.590     denke
0.590     1.050     innen
1.050     1.180     <sil>
1.180     1.590     unten
1.590     1.750     sind
1.750     2.260     unser'm
```

C.2 Google Recognition Results

```
Time:  0.70
0.000     0.702     ich

Time:  0.96
0.702     0.965     nicht

Time:  0.98
0.965     0.974     ich
0.974     0.983     dann

Time:  0.99
0.965     0.974     ich
0.987     0.991     danke

Time:  1.24
0.965     0.974     ich
```

```
0.987     0.991     danke
1.159     1.243     ihnen

Time: 1.55
0.965     0.974     ich

Time: 1.57
0.965     0.974     ich
1.560     1.566     danke

Time: 1.87
0.965     0.974     ich
1.560     1.566     danke
1.766     1.866     ihnen

Time: 2.63
0.965     0.974     ich
1.560     1.566     danke
1.766     1.866     ihnen
2.490     2.515     und
2.515     2.540     wir
2.540     2.565     sehen
2.565     2.590     uns
2.590     2.615     dann
```

C.3 Sphinx+Google (Alignment)

This section shows incremental results for Google+Sphinx recognition in alignment mode. Sphinx computation starts after it receives the first string for alignment. In the example below the time is intentionally modified by adding the time, when the first hypotheses, was received from Google. In the brackets - Sphinx computational time + the time of the first Google hypotheses. Implementation output results put contain only Sphinx computational time without taking an account of Google delay.

```
Time: 1.00 (0.27+0.73)
```

```
0.000     0.110     <sil>
0.110     0.270     ich

Time: 1.28 (0.55+0.73)
0.000     0.400     <sil>
0.400     0.550     nicht

Time: 1.30 (0.57+0.73)
0.000     0.110     <sil>
0.110     0.340     ich
0.340     0.570     danke


Time: 1.56 (0.83+0.73)
0.000     0.110     <sil>
0.110     0.340     ich
0.340     0.610     danke
0.610     0.830     ihnen

Time: 2.47 (1.74+0.73)
0.000     0.060     <sil>
0.060     1.110     ich
1.110     1.740     danke

Time: 2.57 (1.84+0.73)
0.000     0.110     <sil>
0.000     0.110     <sil>
0.110     0.340     ich
0.340     0.610     danke
0.610     1.840     ihnen


Time: 3.00 (2.27+0.73)
0.000     0.110     <sil>
0.110     0.340     ich
0.340     0.610     danke
0.610     1.020     ihnen
1.020     1.470     und
```

```
1.470    1.590    wir
1.590    1.750    sehen
1.750    1.880    uns
1.880    2.270    dann
```

C.4 Sphinx+Google (Alignment + Recognition and JSGF Grammar)

```
Time: 0.33
0.000    0.230    ich
0.040    0.070    <sil>
0.070    0.100    <sil>
0.100    0.150    @
0.150    0.200    @
0.200    0.250    l
0.250    0.300    q
0.300    0.330    <sil>


Time: 0.60
0.000    0.330    ich
0.110    0.460    dann
0.290    0.320    <sil>
0.350    0.380    <sil>
0.380    0.430    @
0.430    0.480    l
0.480    0.530    n
0.530    0.580    h
0.580    0.600    <sil>


Time: 1.58
0.000    0.330    ich
0.110    0.550    danke
0.340    0.580    ihnen
0.580    0.610    <sil>
```

```
0.610    0.680    <sil>
0.720    0.840    <sil>
0.840    1.050    <sil>
1.050    1.200    @
1.200    1.250    @
1.250    1.300    @
1.300    1.360    l
1.360    1.410    h
1.410    1.520     2:
1.520    1.580    <sil>

Time:  2.29
0.000    0.330    ich
0.110    0.550    danke
0.340    0.580    ihnen
0.580    0.610    <sil>
0.610    0.720    und
0.840    1.020    <sil>
1.020    1.470    wir
1.470    1.590    sehen
1.590    1.750    uns
1.750    1.880    dann
1.880    1.930    <sil>
1.990    2.050    <sil>
2.050    2.100    e
2.100    2.150    l
2.150    2.200    o
2.200    2.270    n
2.270    2.290    <sil>
```

C.5 Sphinx+Google (Alignment + Recognition and N-Gram LM)

Bibliography

Baumann, Timo (2013). "Incremental Spoken Dialogue Processing: Architecture and Lower-level Components". PhD thesis. Universität Bielefeld, Germany. URL: `http://www.timobaumann.de/diss/tb-diss-201305231753-publishedversion-compressed.pdf`.

Baumann, Timo, Michaela Atterer, and David Schlangen (2009). "Assessing and Improving the Performance of Speech Recognition for Incremental Systems". In: *Proceedings of NAACL-HLT 2009*. Boulder, USA.

Baumann, Timo, Casey Kennington, et al. (2016). "Recognising Conversational Speech: What an Incremental ASR Should Do for a Dialogue System and How to Get There". In: *Proceedings of the International Workshop Series on Spoken Dialogue Systems Technology (IWSDS) 2016*.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd. Pearson International.

Lamere, Paul et al. (2003). "Design of the cmu sphinx-4 decoder". In: *IN 8TH EUROPEAN CONF. ON SPEECH COMMUNICATION AND TECHNOLOGY (EUROSPEECH)*.

Levenshtein, V. (1966). "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals". In: *Soviet Physics-Doklady* 10.8, pp. 707–710.

McGraw, Ian and Alexander Gruenstein (2012). "Estimating Word-Stability During Incremental Speech Recognition." In: *INTERSPEECH*. ISCA, pp. 1019–1022.

Ravishankar, Mosur K. (1996). *Efficient algorithms for speech recognition*. Tech. rep.

Schalkwyk, Johan et al. (2010). ""Your Word is my Command": Google Search by Voice: A Case Study". In: *Advances in Speech Recognition: Mobile Environments, Call Centers and Clinics*. Chap. 4, pp. 61–90.

S.J. Young, N.H. Russell and J.H.S. Thornton (1989). *Token passing: a simple conceptual model for connected speech recognition systems*. Tech. rep. Cambridge University Engineering Department.

Skantze, Gabriel and David Schlangen (2009). “Incremental Dialogue Processing in a Micro-domain”. In: *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. EACL ’09. Athens, Greece: Association for Computational Linguistics, pp. 745–753.

Twiefel, Johannes et al. (2014). “Improving Domain-independent Cloud-Based Speech Recognition with Domain-Dependent Phonetic Post-Processing”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.* Pp. 1529–1536.

List of Figures

3.1 Incremental output of Sphinx ASR, running in the forced alignment mode. 6
3.2 Incremental output of Sphinx ASR, running in the normal recognition mode. 7

4.1 Sphinx-4 Architecture Overview (Lamere et al. 2003). 13
4.2 From the top of the figure: an acoustic waveform, pitch contour, spectrogram and transcription with gold alignment of the phrase *ich danke Ihnen und wir sehen uns dann*. 14
4.3 A typical HMM abstraction (Jurafsky and Martin 2009). 15
4.4 An extract from Verbmobil language model for Sphinx-4. 17
4.5 JSGF grammar for German digits from one to ten. 17
4.6 An extract from the dictionary file. For correspondence between Sphinx-4 representation and Internation Phonetic Alphabet (IPA) see appendix A. 18
4.7 Viterbi search as dynamic programming (Ravishankar 1996). 19
4.8 An example search graph for the words "one" and "two". 20

5.1 An excerpt of INproTk built-in type hierarchy (Baumann 2013). . . 21
5.2 Incremental Modules and their intercommunication via left-right buffers (Baumann 2013). 22
5.3 Incremental output of Google ASR with existing timing. 23
5.4 JSON file example 23
5.5 Implementation Overview. 24
5.6 An extract from the configuration file for Google+Sphinx recognizer in the alignment mode. 26
5.7 Google+Sphinx output with improved timing. Vertical lines show the time, by which Google was ready with the corresponding hypothesis. Right borders of the dotted boxes show at what point of time Sphinx was ready with its alignment. 27

5.8 An extract from the configuration file for Google+Sphinx Recognizer in alignment and recognition mode. 29
5.9 Schematic representation of the search graph of Google+Sphinx recognizer in combined mode. In the example the text set for alignment is *ich danke*. 29
5.10 An extract from the dictionary file. 30
5.11 Alignment + Recognition result, using combined grammar, based on fixed input "ich danke ihnen" and phone-loop JSGF grammar for the audio, having transcription "ich danke ihnen und wir sehen uns dann" (5.11). 31
5.12 Alignment + Recognition result, using combined grammar, based on fixed input "ich sage danke" and n-gram grammar and simplified language model for the audio, having transcription "ich danke ihnen und wir sehen uns dann" (5.11). 32

List of Tables

6.1 Alignments quality of the recognizer for matches and substitutions in different configurations. 34
6.2 Alignments quality of the recognizer only for matches in different configurations. 34
6.3 Incremental alignments quality of the recognizer in different configurations. 35
6.4 Timeliness quality of the recognizer in different configurations. FO results. 35
6.5 Timeliness quality of the recognizer in different configurations. FD results. 35

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Hamburg, den __________ Unterschrift: _________________________