AUTORAPPER - AUTOMATIC ALIGNMENT OF SPEECH WITH A RHYTHM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — SEBASTIÁN POLIAK
AUTHOR

VEDOUCÍ PRÁCE — doc. Dr. Ing. JAN ČERNOCKÝ
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce popisuje návrh a implementaci aplikace, která automaticky převádí vstupní řeč na rap. Tento proces je založen na zarovnání řeči s rytmem, které je dosaženo pomocí rozpoznávání fonémů, slabikování a časové modifikáce řeči. Další funkce, jako je hudební podklad a vokální efekt jsou přidány za účelem přiblížení se ke skutečnému rapu. Výsledná aplikace je dostupná jako webová služba pro uživatele.

Abstract

This thesis describes a design and implementation of an application that automatically converts the input speech recording into a rap. This process is based on alignment of speech with a rhythm which is achieved by phoneme recognition, syllabification and time-scale modification. The external features such as beat and vocal effect are added in order to make the resulting signal as close as possible to the real rap. The resulting application runs as a web service available to the users.

Klíčová slova

rozpoznávání fonémů, slabikování, časová modifikáce řeči, WSOLA, syntéza řeči, rap, hudba, rytmus, beat, smyčka, chorus, webová služba

Keywords

phoneme recognition, syllabification, time-scale modification, WSOLA, speech processing, rap, music, rhythm, beat, loop, chorus, web service

Citace

Sebastián Poliak: AutoRapper - Automatic Alignment of Speech with a Rhythm, bakalářská práce, Brno, FIT VUT v Brně, 2015

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Jana Černockého.

. . . .
Sebastián Poliak
May 18, 2015

Poděkování

I would like to thank to my supervisor Jan Černocký for the guidance and a lot of valuable advices and ideas that he gave me. I would also like to thank to all the people that supported me during writing of this thesis and to people that participated in Autorapper survey.

Contents

Chapter 1

Introduction

The aim of this thesis is to create an application that would automatically convert a normal speech recording into a rap. The result should be as close to the normal rap as possible, including the alignment of the speech with a rhythm as well as adding an external beat into it.

The first chapter describes the general scheme that I followed to achieve the desirable output. The parts of the scheme are then described separately in the following 4 chapters. The 7th chapter describes my implementation of Autorapper as a web application which is now available to the users. The last chapter deals with the testing of application and the feedback from the users.

1.1 Rap in general

„Rapping can be traced back to its African roots. Centuries before hip hop music existed, the griots of West Africa were delivering stories rhythmically, over drums and sparse instrumentation.“[12] Today, rapping is strongly associated with hip hop music.

According to the definition the components of rapping include 'content', 'flow' (rhythm and rhyme) and 'delivery' which indicates that a rap is usually accompanied and performed in time with a beat. The most important component to distinguish a rap from normal speech or poetry is rhythm. And that is the part that I am dealing with while keeping the original content.

1.2 Existing applications

My original plan was to create Autorapper as a mobile application. However, after I have done some research I found out that something similar already existed for iOS platform. It is called Autorap by Smule. This was the only similar application that I was able to find.

Autorap By Smule [15] is an application that allows you to rap over the beats of famous artist's songs. The user has several default beats to choose from and it is possible to buy some more for real money. A user can also choose whether he wants to rap himself or just speak and it will rap the speech for him automatically. There is also a feature to have a rap battle with other user.

This application has a lot of positive feedback from the users. However, there were some negative comments as well on the issue that the users cannot download their rap into their phones or computers in any way. The users also cannot upload any recording into the application and get it rapped for them. The only way is to record it live.

I tried this application myself. It seems to use a different approach to achieve the rapping, compared to the general scheme of Autorapper described in the first chapter. I also noticed that it often repeats the same syllable in a word several times which is different from Autorapper where every syllable in a word occurs just once.

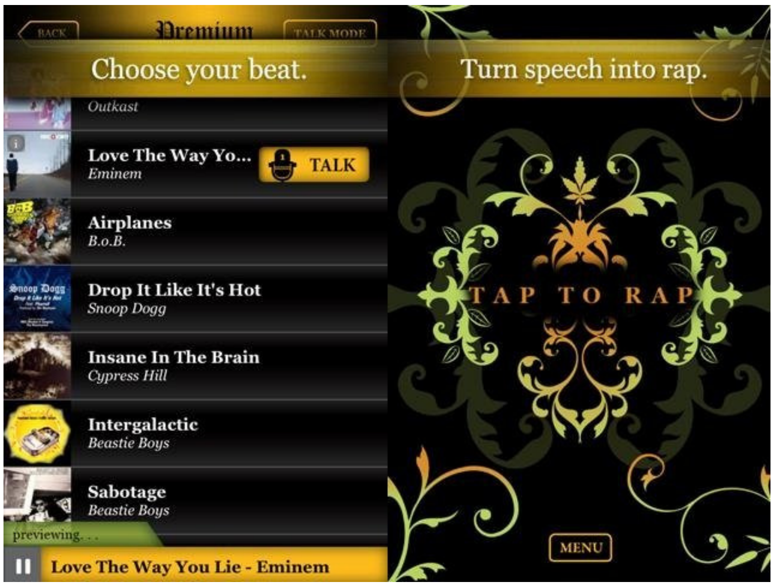


Figure 1.1: Interface of Autorap By Smule

1.3 Possibilities of Autorapper

The approach of Autorapper was from the beginning a bit different from Autorap by Smule. The users of Autorap by Smule are meant to speak to the phone on the spot and their speech recording is rapped and possibly compared with other users in a battle through the interface of the application. On the other hand, the users of Autorapper are mostly meant to choose a recording they want to get rapped, for example a lecture or an audio book. They get their result as a .wav recording and are free to use it in any way they want.

Since Autorap by Smule is a mobile application, the other possibilities for Autorapper were to be a desktop application or a web application. I decided to go with a web application, which is platform-independent and makes it easier for the users to use the service (no download and installation).

Chapter 2

General scheme

2.1 Objective

The main objective is to change the original rhythm of the speech and align it with a rhythm that is in time with a beat. The rhythm of every language is different. „Isochrony is the postulated rhythmic division of time into equal portions by a language.“[13] There are three ways how the languages are divided according to their timing:

1. **Syllable-timed** where the duration of every syllable is equal. Example languages can be Icelandic, Cantonese Chinese, Georgian, French or Welsh.
2. **Mora-timed** where the duration of every mora is equal. Mora is another phonological unit different from a syllable as we know it. An example of such a language is Japanese.
3. **Stress-timed** where the syllables may last different amount of time but there is a constant amount of time between the stressed syllables. These languages are for example English, German, Russian and also the languages such as Czech or Slovak.

We want Autorapper to work with the third category and use a syllable as a basic building block. Figure 2.1 shows an example of the different timings of the syllables. The aim is to change these timings according to the rhythm, therefore, some syllables are needed to be stretched and others shortened.

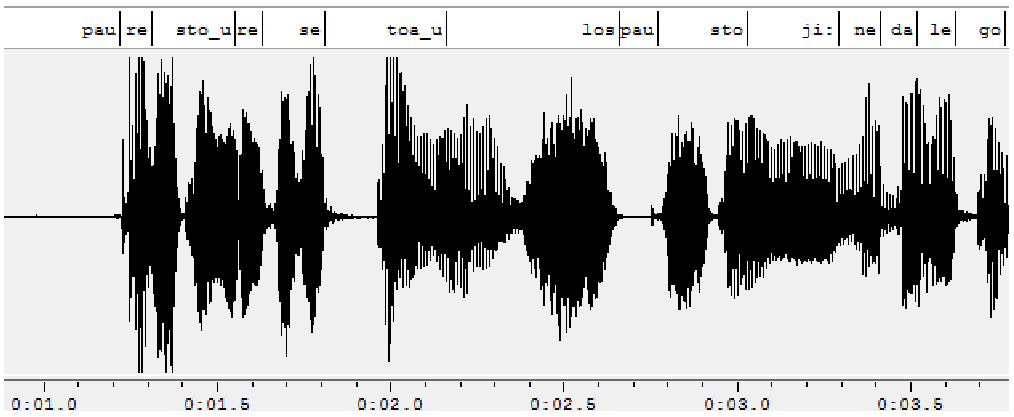


Figure 2.1: Timing of syllables in Czech recording (syllables created by Autorapper syllabification described in Chapter 4)

2.2 Scheme of Autorapper

Several different approaches can be used to change the rhythm of the speech and achieve rapping. Figure 2.1 shows the one that I used for Autorapper. The input is a normal speech recording and the output is a speech aligned to the different rhythm and in time with a beat. Its parts are described in the following sections.

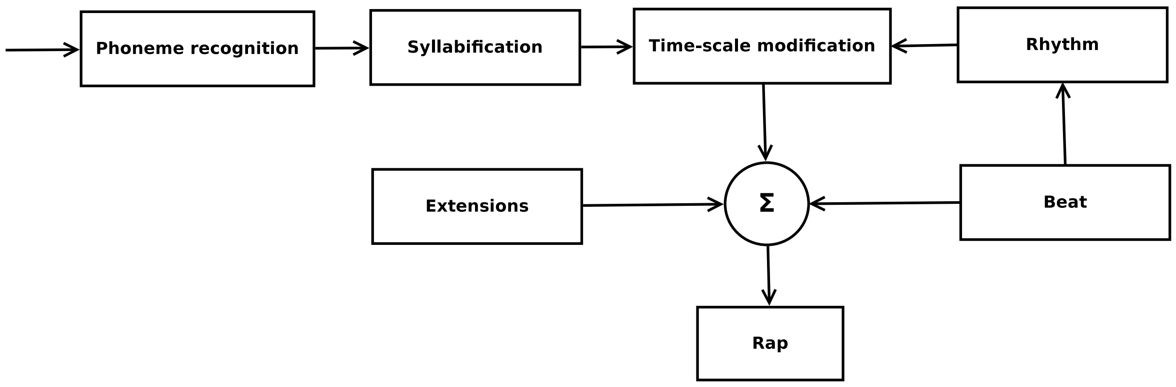


Figure 2.2: Basic draft of Autorapper

Phoneme recognition

At first the single phonemes and their time marks are recognized from the given speech recording. This process is described in Chapter 3.

Syllabification

Based on the phoneme recognition results, the syllables are created by adding the single phonemes together according to the rules of the language. Syllabification is described in detail in Chapter 4.

Time-scale modification

This is the most important part of the project. The length of the syllables needs to be changed according to the given rhythm without affecting their pitch. This modifies the speech to be aligned with a different rhythm. The rhythm is predefined and in time with the beat that is added later. The algorithm that I used to modify the time of syllables is described in Chapter 5.

Adding external rap features

In order to support the output rap and make it sound more natural, external sounds are added into it. These could be any type of a beat or bass line aligned with the rhythm of the speech to create a pleasurable output. There are also some other extensions like a vocal effect or putting the rap into a loop. All of them are described in Chapter 6.

Chapter 3

Phoneme recognition

This part describes the phoneme recognizer and its usage to get the single phonemes and their timing from the speech recording in order to create the syllables.

3.1 Phoneme recognizer in general

This section was based on [1].

A phoneme recognizer can be generally represented as a structure of three blocks shown in Figure 3.1.

In feature extraction, speech is divided into the overlapping frames usually 25 ms long and with 10 ms frame shift. The two most common feature extraction techniques are Mel Frequency Cepstral Coefficients (MFCC) and Perceptual Linear Prediction (PLP). In MFCC a filter to amplify the higher frequencies is used on each frame. Then a Hamming window is used and a Fourier spectrum is computed for the windowed signal. Mel filter bank is then applied to smooth the spectrum. After that, a logarithm is used followed by Discrete Cosine function. The resulting coefficients form a vector usually in 13 dimensional space.

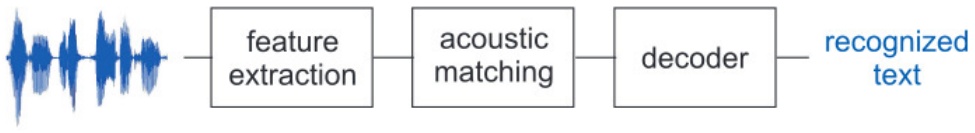


Figure 3.1: Common structure of speech recognizer (*taken from [1]*)

Acoustic matching matches the parts of the signal with other stored examples of speech and assigns the scores to the acoustic units. It usually uses Hidden Markov Models (HMM) where the likelihood is modeled by a probability density function. This can be a Gaussian Mixture Model (GMM) or an Artificial Neural Network (ANN).

A decoder is used to find the best path through the acoustic units, optionally using other knowledge about the language.

3.2 Phoneme recognizer by BUT Speech@FIT

This section was based on [2].

I was able to use the phoneme recognizer that was developed at BUT Speech@FIT group. It offers Czech, English, Russian and Hungarian phoneme recognition. I worked with Czech phoneme recognition and the syllabification that follows is also based on Czech language. The input to the phoneme recognizer is a mono recording (1 channel), 8 kHz sampling frequency and signed 16 bit per sample.

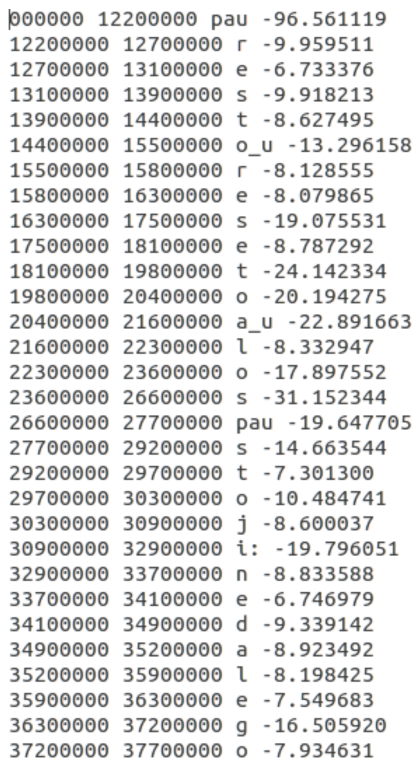
The phoneme recognizer uses a TRAP based system. It is a HMM - Neural Network hybrid. Speech signal is divided into 25 ms long frames with 10 ms shift. The Mel filter-bank is emulated by triangular weighting of FFT-derived short-term spectrum in order to obtain short-term critical band logarithmic spectral densities. TRAP feature vector describes a segment of temporal evolution of critical band densities with a single critical band. The actual frame is a central point and there is an equal number of frames in past and future. This vector forms an input to the classifier and the outputs are the posterior probabilities of sub-word classes. These outputs are combined into one using a merger, which is another classifier. Both merger and band classifiers are neural nets. Output contains the phoneme probabilities for the central frame that are put into a Viterbi decoder, producing phoneme strings. The Czech variant contains 1500 neurons in all nets.

3.3 Description of the result

The first two elements are the start and the end of the occurrence of a phoneme. The timing is in Hidden Markov Model Toolkit (HTK) format, where the basic unit is 100 ms. In my case, to get an actual time in a recording, the timing needs to be divided by 10^7 and multiplied by the frame rate of the recording.

The third element is the actual phoneme. Czech variant of phoneme recognizer contains 45 different labels. Occurrence of 'pau' means that there is a pause (silence) in a recording on the given place. This is useful later in syllabification process. The other signs could be also present in the value of a phoneme, for example ':' means that the phoneme is stretched or two phonemes are joined by '_' usually forming a diphthong.

The last element is the log likelihood of the phoneme. An example of a result from the phoneme recognizer is shown in Figure 3.1 where the input was an audio book starting with the sentence *„Restaurace Toulos stojí nedaleko ..“*.



```
000000 12200000 pau -96.561119
12200000 12700000 r -9.959511
12700000 13100000 e -6.733376
13100000 13900000 s -9.918213
13900000 14400000 t -8.627495
14400000 15500000 o_u -13.296158
15500000 15800000 r -8.128555
15800000 16300000 e -8.079865
16300000 17500000 s -19.075531
17500000 18100000 e -8.787292
18100000 19800000 t -24.142334
19800000 20400000 o -20.194275
20400000 21600000 a_u -22.891663
21600000 22300000 l -8.332947
22300000 23600000 o -17.897552
23600000 26600000 s -31.152344
26600000 27700000 pau -19.647705
27700000 29200000 s -14.663544
29200000 29700000 t -7.301300
29700000 30300000 o -10.484741
30300000 30900000 j -8.600037
30900000 32900000 i: -19.796051
32900000 33700000 n -8.833588
33700000 34100000 e -6.746979
34100000 34900000 d -9.339142
34900000 35200000 a -8.923492
35200000 35900000 l -8.198425
35900000 36300000 e -7.549683
36300000 37200000 g -16.505920
37200000 37700000 o -7.934631
```

Figure 3.2: Result from Phoneme Recognizer

3.4 Possible problems with phoneme recognition

An obvious problem that can occur is with the accuracy of phoneme recognition. According to the site [8], the error rate of the phoneme recognizer for Czech language is 24.24%. This can cause the problems later in syllabification and therefore the resulting rap can sound less natural. However, many times when the phoneme recognizer misclassified the phoneme it is still classified as a consonant or a vowel corresponding to the correct phoneme. This can be seen at Figure 3.1 where for example 'a' and 'k' are misclassified as 'e' and 'g' in a word 'restaurace'. This is essential for the syllabification which is based on the fact whether the given phoneme is a vowel or a consonant and so the correct phoneme is not that important.

The result of the phoneme recognizer is also dependent on the quality of the recording. When the recording is too noisy or there are some other background sounds, the result will not be very precise. The optimal recording should contain only the speaker. I used to test it with the audio books and it gave me pretty good results.

Chapter 4

Syllabification

After the phonemes are recognized, they are added together in order to create the syllables. Syllables are the phonological „building blocks“ of words. They can influence the rhythm of a language, its prosody, its poetic meter and its stress patterns. Therefore the syllables are the essential part to work with in Autorapper.

4.1 Syllable structure

„A syllable is typically made up of a syllable nucleus (most often a vowel) with optional initial and final margins (typically, consonants).“ [14] The length of a syllable is usually two or three phonemes. The most usual form of syllable contains two phonemes where the first one is a consonant and the second one is a vowel. The other form of a syllable that contains three phonemes starts with a consonant and is followed by two vowels. These vowels usually form a diphthong. The possible diphthongs are ’ia’, ’ie’, ’iu’ and ’uo’. The last form of a syllable also contains three phonemes where the first one and the last one are the consonants and the nucleus is a vowel. These type of syllables usually occurs at the end of the words.

4.2 Exceptions

There are some consonants that can sometimes behave like a vowel and create a syllable where they are the nucleus. Those are ’r’ and ’l’ and their stretched variants. They act as a vowel nucleus of a syllable for example in the words like ’krk’ or ’slza’. However they can also act as a normal consonant for example in a word ’ryba’ and therefore the program has to distinguish between these cases.

4.3 Process of syllabification

The process of syllabification has to take in consideration all the possible structures of a syllable. In the result of phoneme recognizer, the pauses between the words are represented as ’pau’. This is useful for detecting words that finish with a consonant. For example, if we take a word ’losos’ and try to divide it into the syllables, the first one will be obviously ’lo’ but the second one cannot be ’so’ which would otherwise be okay with the rules. Instead, a look ahead is used to determine whether the following consonant is followed by a pause. If yes, then the syllable is ’sos’. If the consonant is followed by other characters then the syllable remains 'so' and the syllabification continues (for example, in a word 'lososový').

Figure 4.1 shows a flowchart that is representing a process of creating a syllable covering the structures mentioned in this chapter. Appending means to use a start or end time of a phoneme as a time of syllable. The implementation of syllabification is then based on this flowchart and described in Chapter 7. Its result mapped to the recording can be seen in Figure 4.2.

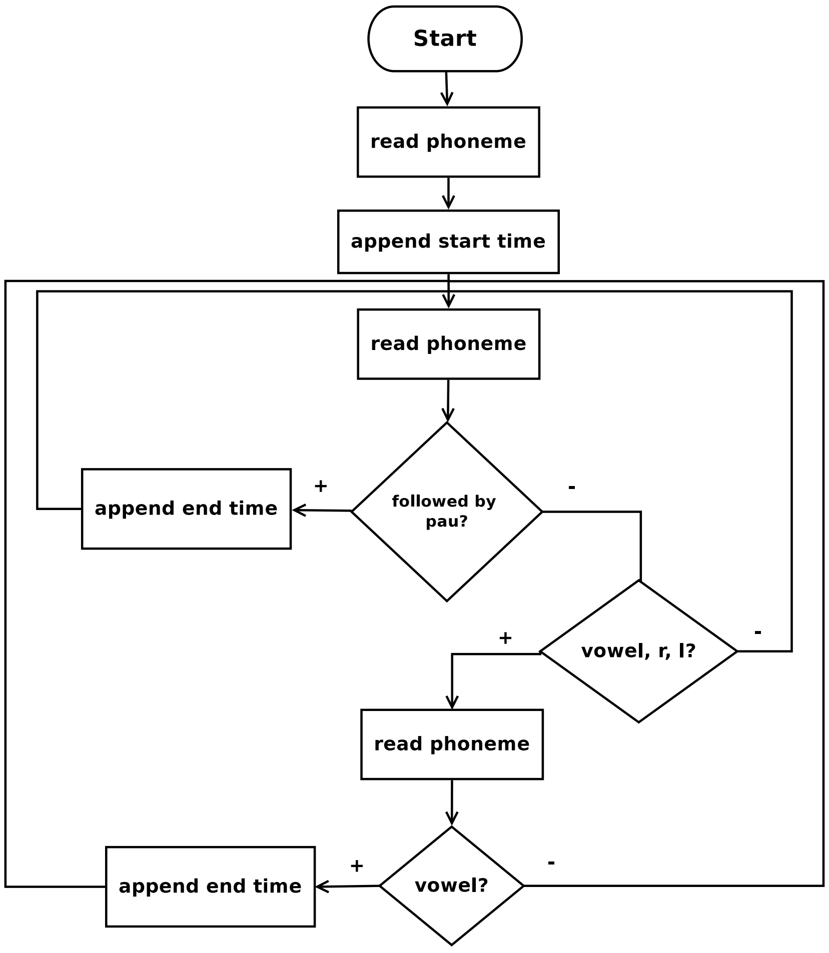


Figure 4.1: Flowchart of syllabification in Autorapper

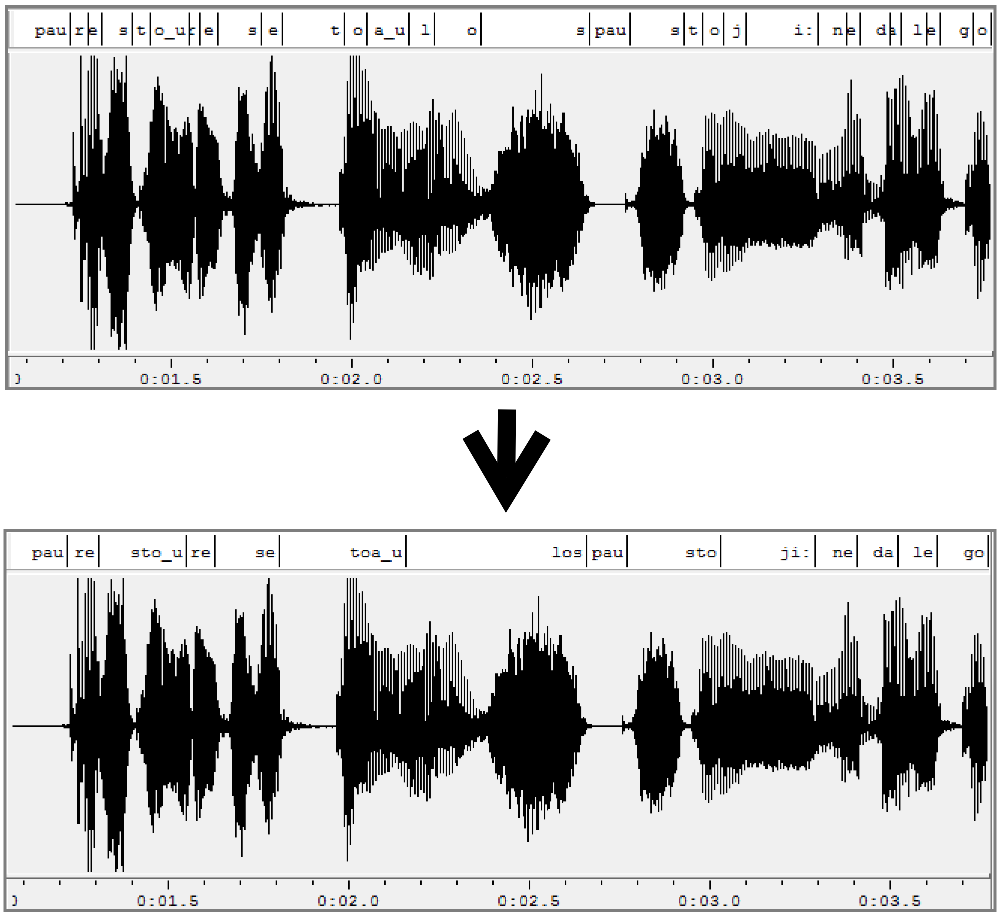

pau re s t o_u r e s e t o a_u l o s pau s t o j i: ne da le g o
0:01.5 0:02.0 0:02.5 0:03.0 0:03.5
pau re sto_u re se toa_u los pau sto ji: ne da le go
0:01.5 0:02.0 0:02.5 0:03.0 0:03.5

Figure 4.2: Result of syllabification in comparison with the single phonemes

Chapter 5

Time-scale modification

Time-scale modification (TSM) is a technique used to modify the duration of the audio signal while minimizing the distortion of other important characteristics, such as pitch and timbre. TSM has been widely used in a field of speech and audio processing. Over last three decades various overlap-and-add algorithms have been created. Among them, synchronized overlap-and-add (SOLA) based TSM, pitch synchronous overlap-and-add (PSOLA) based TSM, and waveform similarity overlap-and-add (WSOLA) are the ones that show relatively good performance regarding output quality.

5.1 Choosing the right TSM algorithm

This section was based on [3].

This section compares the three TSM algorithms mentioned above and determines which one would fit the best for the purpose of Autorapper.

SOLA-based and PSOLA-based TSM [4] have disadvantages compared to WSOLA-based TSM. In SOLA-based TSM the overlap-and-add is performed according to the output similarity and therefore the overlap position differs in each frame. This leads to the fact that the exact output length is not guaranteed, which could cause some problems in rhythm alignment of the speech. In PSOLA-based TSM, the output quality varies according to the performance of the pitch estimation algorithm. Therefore a high quality pitch estimation algorithm is required which would not be a problem because I had one available from the school. However, the usage of it also incurs more computational complexity. The advantage of PSOLA-based TSM is that it can be used to change the pitch of the signal as well but this feature is not really needed for Autorapper although it could be used for some possible extensions. That left us with WSOLA-based TSM algorithm which is discussed in more detail in the following sections.

5.2 WSOLA

Waveform Similarity and Overal Add (WSOLA) is an algorithm for High Quality Time-Scale Modification of Speech. It provides similar output quality as the other two algorithms mentioned above, while having a relatively smaller computational complexity. Another benefit is that the output length of WSOLA-based TSM is guaranteed (in the end there were some problems with that which are discussed in section 7.2.2).

5.2.1 Principle of WSOLA

The WSOLA algorithm is used to best align each signal block of the rate changed signal to the ideal signal block (no rate change) at each frame in order to minimize the distortion due to phase differences at the frame boundaries. The method also uses windowing in order to reduce the remaining effects of discontinuities at the boundaries between frames. The frame generation and overlap parameters, along with the duration of the alignment offset are all algorithm variables that are explicitly specified. The synthesis equation of WSOLA is:

$$y(n) = \frac{\sum_k v(n-kL)x(n+kL\alpha-kL+\delta_k)}{\sum_k v(n-kL)} \tag{5.1}$$

where *y(n)* is the corresponding time-scaled output signal, *x(n)* is the input signal and *v(n)* is a window signal. L represents the overlap-and-add length and α is a time-scale factor. Basically, if α is lower than 1.0 the length of the signal is expanded. If α is higher than 1.0 the signal is truncated.

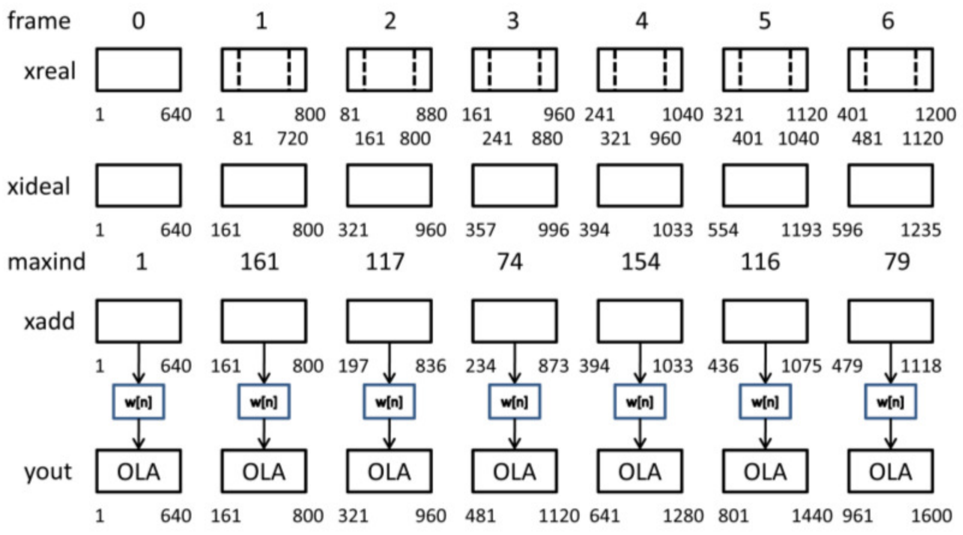


Figure 5.1: Frame-by-frame WSOLA processing (*taken from [6]*)

Figure 5.1 shows an example of WSOLA processing on each frame in several steps. The parameters in this case are α=0.5 and L=640, the sampling frequency of input signal is 16000 and Hamming window is used. The four arrays *xreal*, *xideal*, *xadd* and *yout* of length L are initialized in step 0. However, in the first step the length of *xreal* array is extended by δ from both sides which is in this case equal to 80. Another parameter that WSOLA uses is a frame shift R which is multiplied by α and represents the number of samples that we move along the signal in each iteration (shown as δ_k in formula 5.1). In case of Figure 5.1 the value of R=160. The *xideal* array contains the samples of signal from the place where the non-time-scale modified signal would ideally be. The *xreal* and *xideal* arrays are then cross-correlated (maximum index of correlation is represented by *maxind*) and the result is put into the *xadd* array. The signal in *xadd* array is then windowed and put into the output array *yout* at the given indices. This process iterates over the each frame of the input signal. We can see that after the first 6 steps the *yout* array is ending at frame 1600 while the xreal is only at 1200 and therefore the signal is being time-extended.

5.2.2 Alignment of speech with a rhythm

To align the speech with a rhythm, every syllable is put into WSOLA separately. Coefficient α determines how do we want to time-scale the given syllable. The α is computed from the original duration of the syllable divided by the time into which we want to time-scale the given syllable. This time is determined by the beat and a predefined rhythm that we want. Figure 5.2 shows a graphic representation of the signal before and after WSOLA processing with coeficient α=0.5.

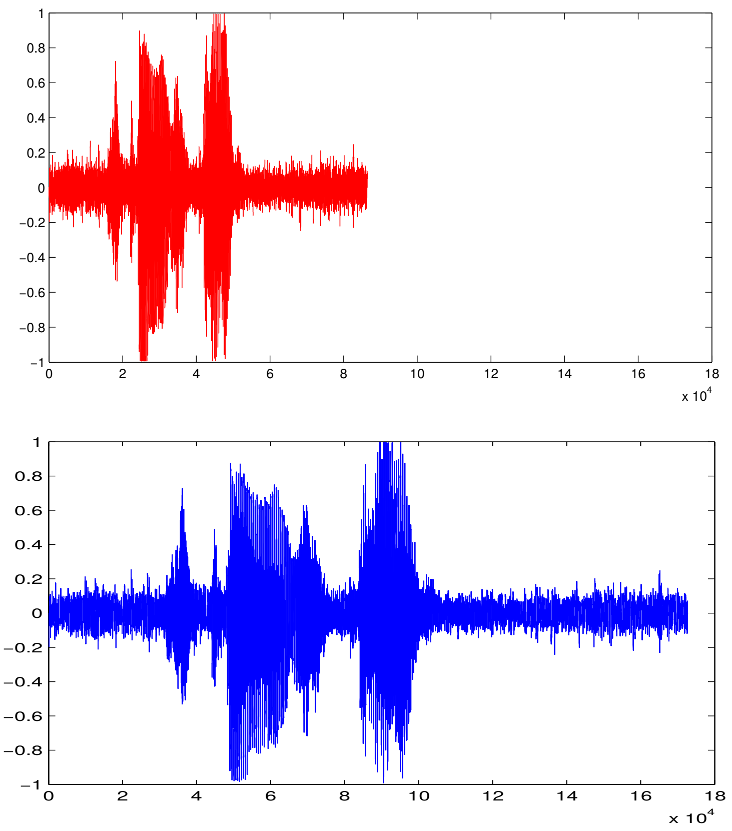


Figure 5.2: Signal before and after WSOLA processing with α=0.5

Chapter 6

Towards a real rap

If we want to achieve a rap which is similar to the real rap as we know it, it is necessary to add some external features. This really helps to support the rhythm of the speech and also hide some possible imperfections of the speech synthesis. This chapter describes several external features that I added to improve Autorapper as well as definition of the rhythm.

6.1 Rhythm

First we have to define a rhythm that the speech will be aligned with using the principle from section 5.2.2. The rhythms that I used are shown in Figure 6.1. The first one is a basic quarter notes rhythm and the second one is an eighth notes rhythm. Using these rhythms makes all the syllables last the same time. However, because the Czech language is stress-timed (explained in Chapter 2) some syllables are stretched and some are truncated in order to align them with this rhythm. This makes the speech sound different from what it was like before and when it is put together with the beat it really resembles a rap.

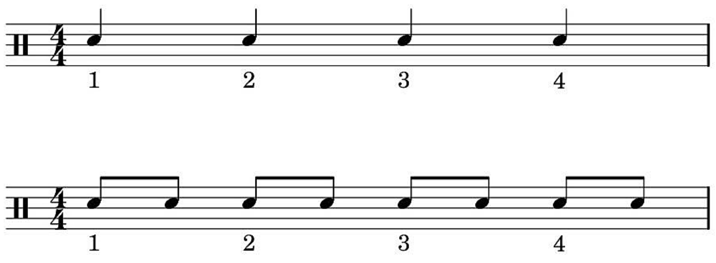


Figure 6.1: Basic quarter notes and eighth notes rhythms

Although this rhythm pattern works good, there is always a place for improvement. The other possible rhythms could be taken from the famous rappers and used here. There was also an idea that the rhythm could be randomly generated in each measure (4 beats in this case). So the result of Autorapper would be different each time you use it on the same recording. This is a feature that I would like to add in the future.

6.2 Beat

This is the most important feature that is needed to add. The speech is aligned with a rhythm that has to be in time with the beat. I used 3 different beats that the user can choose from. All of them were available free at `http://www.unbelievablebeats.com/free-beats-free-downloads` [7]. They differ in their speed (BPM - beats per minute) and also in their characteristics. The first one is called Bangin Beats and has 93 BPM . It is characterized by it's backing vocals. The second one is called 9th Wonder kit with 181 BPM and bongos. The last one is Battle kit with 209 BPM and a heavy bass.

I could use any number of beats for the users but for now 3 is enough. The duration of each strike in a beat in seconds is calculated as 60/BPM. This determines how much does one note in a predefined rhythm last. It is possible to put 2 or more notes into 1 strike depending on the beat and what sounds better.

6.3 Vocal effect

Another feature that is added to the result is a vocal effect. This helps to smooth out the speech and also add some robustness. Several vocal effects exist for example Echo, Chorus, Delay or Reverb. Sox [9] offers most of them, where it is also possible to enter several parameters describing the intensity of the effect. I decided to use Chorus, which gave nice results in case of Autorapper.

6.4 Loop

According to the user feedback, which is later discussed in a separate section, I noticed that people sometimes upload a recording with just few sentences. When it is put into Autorapper, it is usually even speeded up to fit the beat and so the whole rap of the recording takes only few seconds.

Therefore, I decided to put the rapped speech into a loop and repeat the section over and over again while the beat is going. The number of loops is counted according to the desired duration of the rap and the duration of the beat so it will not exceed it.

Chapter 7

Implementation

This chapter describes the implementation details of each of part of Autorapper and the way that these parts work together. The basic scheme of how the parts are joined together and what are their inputs and outputs can be seen in Figure 7.1. The user input is a speech recording and the choice of the beat. The scheme is described in the following section.

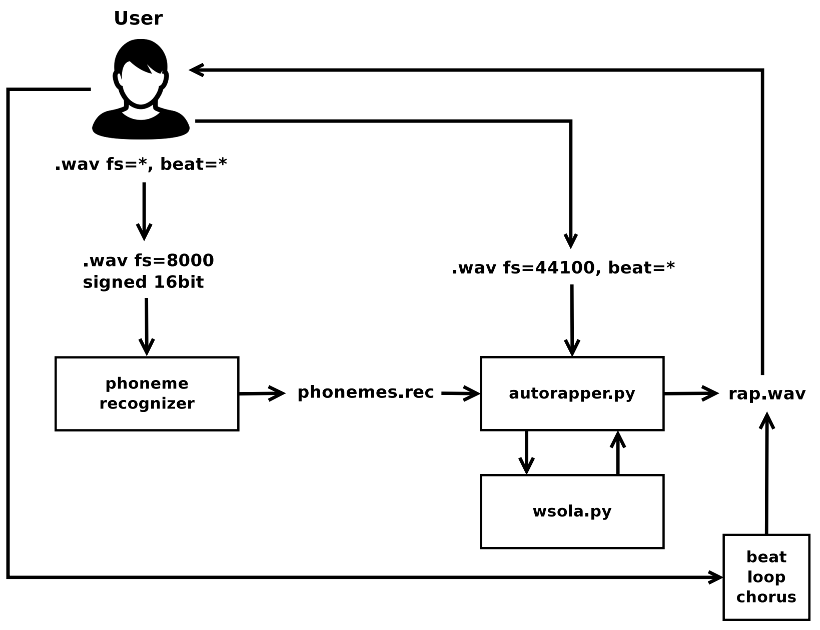


Figure 7.1: Implementation scheme with inputs and outputs of each part

7.1 Integration and pre-processing

This section describes how all the parts of Autorapper are joined together and work as a one part. The module that takes care about this is a bash script `rap.sh`. It makes sure that all the parts are executed in a right order and their inputs are in correct format according to the scheme in Figure 7.1.

The conversions of audio signal into different sampling frequency and encoding are done using Sox [9]. Sox is also used to add Chorus effect as well as mix the rap with the chosen beat. All the beats are saved in a directory and numbered. The module `autorapper.py` takes the input recording and the beat number as the parameters. It needs this number to know what is the speed (BPM) of the beat to align the speech with a rhythm in correct time.

The syllabification is done in `syl()` function of `autorapper.py` module and requires *.rec* file from phoneme recognizer to be present. When the syllables are ready the coefficients for WSOLA are counted for every syllable according to section 5.2.2 with respect to the beat timing. The WSOLA from `wsola.py` is called on every syllable and the results are joined together and written into the file. After that, the external features are added into it.

The bash script also takes care about the web service issues such as ftp access to the web server and invoking the php script on the web server to send the links of the rapped speeches to the users. These are explained in section 7.5 about the web based application.

7.2 Phoneme recognition

The phoneme recognizer by BUT Speech@FIT was available in binary form for both Windows and Linux systems. Therefore, I did not have to compile it myself. It only needs the recognition system folder to be present. In my case, `PHN_CZ_SPDAT_LCRC_N1500` folder for Czech variant. The language variant together with other parameters are selected in execution as `./phnrec -v -c PHN_CZ_SPDAT_LCRC_N1500 -w lin16 -i input.wav -o output.rec -p -3.0`.

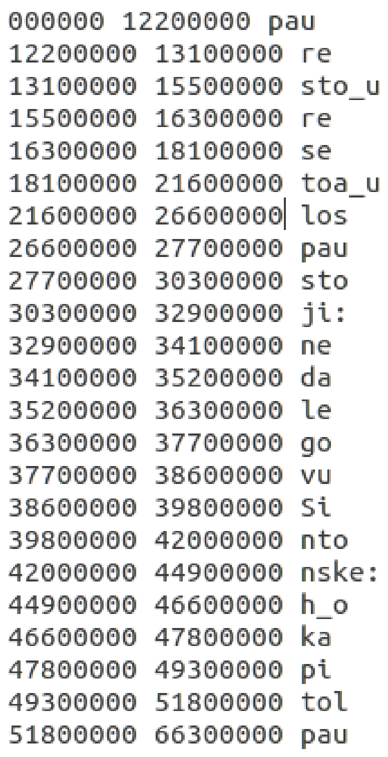
7.3 Syllabification

To create the syllables I used Python language that contains the dictionaries which I used for mapping the time of a phoneme to its value from the phoneme recognizer. The syllabification is implemented in `syl()` function in *autorapper.py* module. The principle of syllabification basically follows the flowchart shown at Figure 4.1 in Chapter 4.

The vowels are contained in an array `['a','e','i','o','u','y']` through which the program iterates and compares with the actual phoneme. The phonemes 'r' and 'l' are added to the test separately. When a phoneme contains ':' meaning that it's stretched it is considered to be a vowel as well. This principle seems to work because most of the stretched phonemes are vowels and the stretched consonants 'r:' and 'l:' are probably always used as a syllable nucleus in this case.

The diphthongs in the result from the phoneme recognizer can be represented in two ways. It can be just one character for example 'o_u' or it can be two separate vowels 'o' and 'u' in a row. In the first case the diphthong is treated as a single vowel. In the second case a look ahead for a second vowel is used.

The result of the syllabifications are the syllables with the timing composed of the starting time of the first character and the ending time of the last character of the syllable. The spaces between the words are for now considered as a syllable however, in the final result Autorapper ignores the spaces at all. The result of the syllabification can be seen in Figure 7.2. The input was an audio book where the given text was *„Restaurace Toulos stojí nedaleko washingtonského Kapitolu“*.



```
000000 12200000 pau
12200000 13100000 re
13100000 15500000 sto_u
15500000 16300000 re
16300000 18100000 se
18100000 21600000 toa_u
21600000 26600000 los
26600000 27700000 pau
27700000 30300000 sto
30300000 32900000 ji:
32900000 34100000 ne
34100000 35200000 da
35200000 36300000 le
36300000 37700000 go
37700000 38600000 vu
38600000 39800000 Si
39800000 42000000 nto
42000000 44900000 nske:
44900000 46600000 h_o
46600000 47800000 ka
47800000 49300000 pi
49300000 51800000 tol
51800000 66300000 pau
```

Figure 7.2: Result of syllabification from Autorapper

7.4 WSOLA

This section describes my implementation of WSOLA-based TSM algorithm and it's usage to rap the speech. WSOLA is implemented in `wsola.py` module as a function taking 3 parameters - the input signal, alpha which is a coeficient by which the length of the signal should be modified and the framerate of the input signal. The algorithm follows the principle described in section 5.2.1.

7.4.1 Libraries

To implement WSOLA algorithm I used Python language. It might not be the most effective language to do this kind of computation however, it offers several libraries supporting audio and signal processing which are optimized and make the work more stable and easier. The libraries that I used are wave, numpy [11] and a write module from scipy.io.wavfile [10].

Wave library was useful for opening the input audio recording and getting its parameters such as number of frames, sample width, number of channels and a frame rate. It was also used for reading the given number of frames from the recording usually one syllable to separate it from the rest so it can be modified and added into the output.

Numpy library was useful for dealing with the signal arrays. The frames read by wave module can be reshaped to a numpy array. Numpy also offers some audio related methods such as Hamming window or correlation which were both useful in implementing WSOLA.

Scipy.io.wavfile write module was used only to write the final signal into the file directly from the numpy array so it does not need to be reshaped to the original wave format.

7.4.2 Problems with WSOLA

WSOLA always returns an array that is equal to the input array scaled by the factor as expected. However, in the beginning I had a problem that this array was not fully filled with the signal and had some zeros at the end as shown in Figure 7.2.

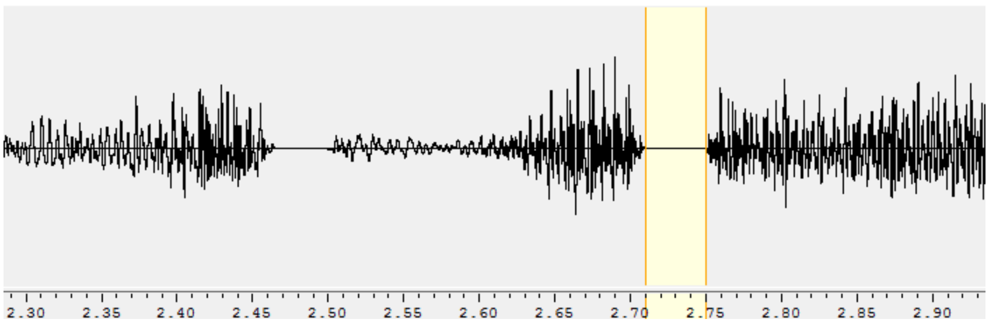


Figure 7.3: Problem with zeroes at the end of signal after WSOLA processing

This caused an unwanted effect in the result because the spaces between the syllables could be heard and so the syllables were not continuous. The problem could be reduced by increasing the sampling frequency. I originally worked with 8000 hz sampling frequency which was the input to the phoneme recognizer as well. This frequency was too low for WSOLA to work properly and the quality of the signal was bad as well compared to the input signal from the users which is usually in higher sampling frequency. Therefore all the inputs for `wsola.py` are converted to 44100 hz sampling frequency which is usually the original frequency anyways. This reduced the number of zeros at the end, however, some zeros still remained.

Another factor that had an effect on this issue were the constant parameters of WSOLA. Originally, the parameters were set to L=40, R=10 and δ=5. I tried to change the parameters in different ways and watch the effect that it has on the result. The ones that worked best were L=20, R=5 and δ=5. This mostly fixed the issue although some zeros still occur. The effect that it has on the result is minimal.

7.5 Web based application

Autorapper is currently running at www.autorapper.cz. From the user's point of view the user just uploads a recording and receives a link with the result by email. However, it was impossible to make Autorapper running directly on the hosting web server. I did not have the rights to install the needed libraries there and the phoneme recognizer did not work as well. I was able to run all the parts of Autorapper on the school's server Merlin. Although Merlin offers a web service as well, the `upload_max_filesize` was not enough for the average `.wav` recording that is needed to be uploaded and therefore, I could not run it this way either. So I had to come up with a compromise that all the user interactions as well as uploading a file are done on the hosting server while all the computations of Autorapper are done on Merlin. The usage of Merlin server is just temporary, in a long run, the service would need to be moved to a separate server. This principle is shown in Figure 7.4. Its parts are then described in client side and server side sections.

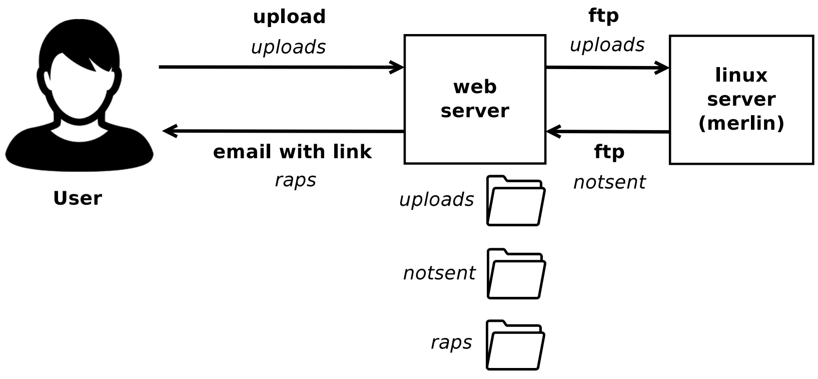


Figure 7.4: Scheme of the web service of Autorapper

7.5.1 Client side

This section describes the parts that the web server is responsible for, considering it to be the client side. It is a part that the user interacts with.

Firstly, it checks the format of recording which must be `.wav` and whether the user entered a valid email, which is necessary to get the result. After that, if the upload was successful the uploaded file is saved into `uploads` folder with the name composed of number of beat chosen by user followed by '_' and an email address. For example, '2_xpolia01@stud.fit.vutbr.cz'. This is an easy way to pass the information to the server side just by downloading the file. The file is now ready to be processed. The processing is done by the server side and after that, the rapped file is found in `notsent` folder. The server side executes `sendraplinks.php` script on the client side, which moves all the raps that are currently in `notsent` folder into `raps` folder and sends the links to the users. The files remain stored in this folder.

7.5.2 Server side

The server side is responsible for all the computations and rapping the speech. In this case it is the Merlin server.

The server side downloads the user's file through ftp from the `uploads` directory of the hosting server. This is all implemented in `rap.sh` script mentioned before. The script may check repeatedly whether there are some new uploads on the client side or it can somehow get noticed from the client side that the new files have been uploaded and it should be executed. The number of beat is obtained from the filename and so the script raps the speech accordingly. After that, the script uploads the file into the client side `notsent` folder through ftp and executes `sendraplinks.php` on the client side to send the links to the users.

7.5.3 Interface and content of the website

I tried to come up with as simple user interface as possible. The website contains only 3 pages - Home, Example and About. Home is the main page that is shown when a user enters the website. It is also the page to upload the recording, select the beat and enter the email for the result. A user can also like the facebook page of Autorapper. A screenshot of Home page can be seen in Figure 7.5. Example section contains an example of the speech that was rapped by Autorapper together with its original recording. This section is meant to give the users a basic idea about what Autorapper does. About section contains the credits for Autorapper project and the contact information.

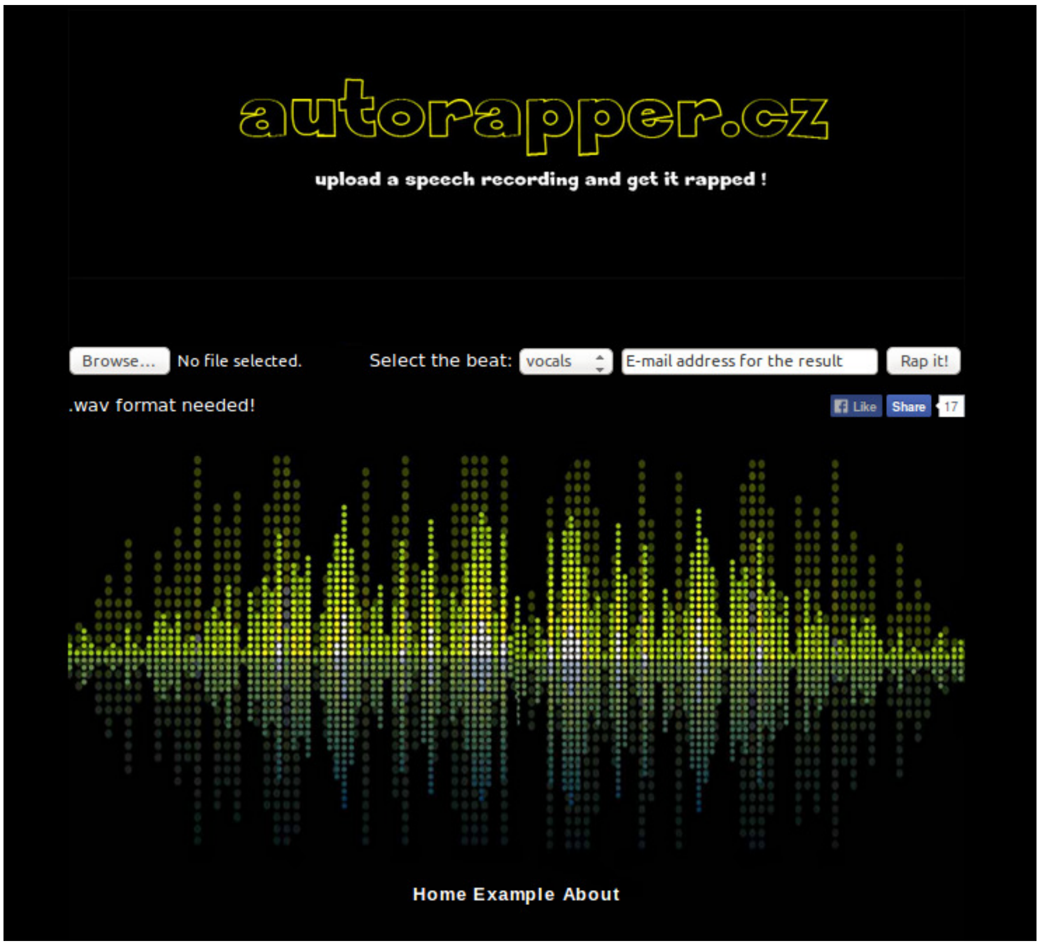


Figure 7.5: Interface of Autorapper as a web application

Chapter 8

Testing

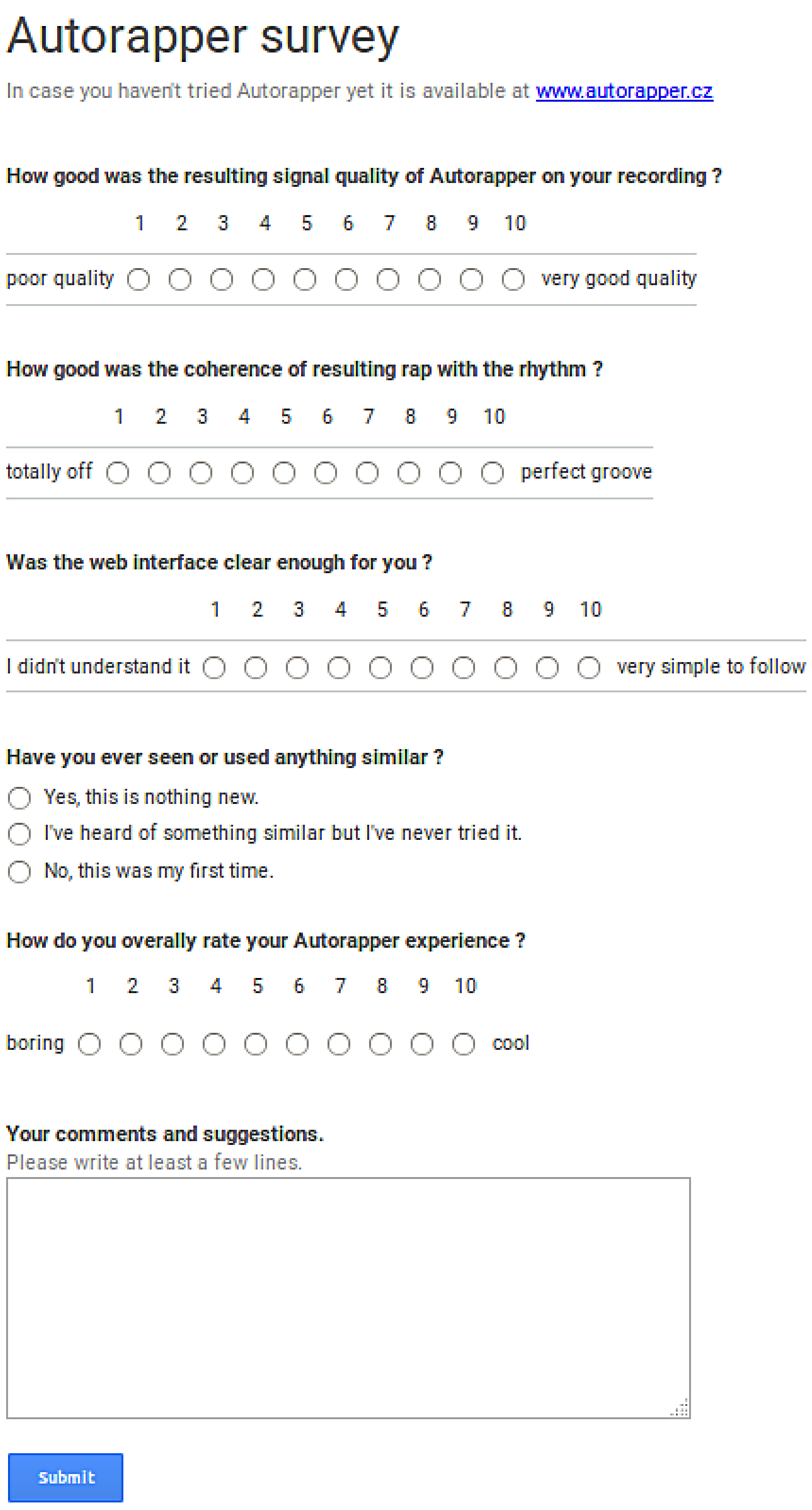
This chapter describes the ways that I tested Autorapper during the development, as well as the feedback from the users obtained by the survey. I used several different kinds of speech recordings in order to get the idea of what works and what is possible. The table shows the kinds of recording that I put into Autorapper and quality of the results.

Czech audio book	*satisfactory result*
Czech recording by non-native speaker	*satisfactory result*
Slovak recording	*few mistakes, still acceptable*
Recording with background noise	*not acceptable*
Singing	*not acceptable*
English recording	*not acceptable*

The main speech recording which I used to test Autorapper was an audio book in Czech language. Autorapper is mainly developed for Czech language, so the audio book was a perfect example of speech without any background noise. The results with the audio book were very nice. One of them is also available on the website as an example of what Autorapper does. However, the other kinds of speech recordings did not guarantee such a good results. Each part of Autorapper is totally dependent on the previous parts. If one part fails, the error just accumulates during the following parts and the result is unbearable. The one part that Autorapper is most dependent on is the phoneme recognizer. If this returns a failed result, the syllabification cannot create the correct syllables in any way.

I tried to use Autorapper on some Slovak recordings as well. The syllabification and phonemes of Slovak language are very similar to Czech and therefore I expected a satisfactory result. The result was not totally bad, but it was not as good as one from the Czech recording either. The phoneme recognizer uses its Czech variant for every recording. It was trained on the Czech samples and therefore I cannot expect the correct results for the other languages. A possibility to make Autorapper work for other languages as well would be to detect the language for each recording and switch the variant of phoneme recognizer accordingly. The phoneme recognizer offers three other languages that could be used, however, the syllabification for these languages would need to be remade as well.

I also tried to put a recording of singing into Autorapper to transform it into a rap. The result was not good because singing is very different from speech. The syllables usually last very long and even if the phoneme recognition would be correct, the quality of shortening such a long syllable by WSOLA processing is poor.



Autorapper survey

In case you haven't tried Autorapper yet it is available at www.autorapper.cz

How good was the resulting signal quality of Autorapper on your recording ?

	1	2	3	4	5	6	7	8	9	10	
poor quality	○	○	○	○	○	○	○	○	○	○	very good quality

How good was the coherence of resulting rap with the rhythm ?

	1	2	3	4	5	6	7	8	9	10	
totally off	○	○	○	○	○	○	○	○	○	○	perfect groove

Was the web interface clear enough for you ?

	1	2	3	4	5	6	7	8	9	10	
I didn't understand it	○	○	○	○	○	○	○	○	○	○	very simple to follow

Have you ever seen or used anything similar ?

- ○ Yes, this is nothing new.
- ○ I've heard of something similar but I've never tried it.
- ○ No, this was my first time.

How do you overally rate your Autorapper experience ?

	1	2	3	4	5	6	7	8	9	10	
boring	○	○	○	○	○	○	○	○	○	○	cool

Your comments and suggestions.
Please write at least a few lines.

Figure 8.1: Questionnaire of Autorapper survey

8.1 Autorapper survey

To get the feedback from the users which is especially important in this kind of projects, I made a survey composed of the questions shown in Figure 8.1. The users could rate several aspects of Autorapper on scale from 1 to 10, as well as add their own comments or suggestions which are especially important. I put a link to the survey on the website of Autorapper, as well as on its facebook page. A few of my friends and other people participated. Most of them were students in young age.
The results are shown and discussed in section below.

8.1.1 User feedback

An average score for the first question about the signal quality was 7.25. The question about the rhythm coherence got 8.25 and the question about the web interface scored 8.75. None of the people that took this survey has every seen or used anything similar. The overall rating of Autorapper experience was 8.75. Some people also left the comments which are shown below:

„The output quality was not a disaster as I thought. It was good but could be better. Anyways, I completely like the idea.“

„I didn't have any wav recording to upload so I just listened to the example. I liked it!“

„I think it has a huge potential!“

„It is a good idea. A support for English language as well would be great.“

„I tried several different recordings. Some results were better than the others. Anyways good job!“

Generally the results of the survey were quite good. However the survey would need much more people to participate in order to make a conclusion. At this time, Autorapper does not have that kind of community. Only 11 people took the survey and some of them did not leave any comment. Many of the mentioned things could be improved. The possible improvements are discussed in section 9.2 about Future. It is also interesting that none of the people who took the survey has ever tried or seen anything similar. The only one similar application that I found is Autorap by Smule and it does not seem to be very known.

Chapter 9

Conclusion

9.1 Summary

In this thesis I became acquainted with the phoneme recognizer and the Time-Scale modification algorithms. I implemented the syllabification and WSOLA algorithm. Using them I aligned the speech with the rhythm and added external features such as beat and vocal effect into it. The result is an application which automatically raps the speech running as a web service. Not all the results are perfect. They depend on the kind of the recording that is given to the application. The possibilities of usage of the application are open to the users. The best results could be obtained on the audio books, lectures or public speeches.

9.2 Future

Autorapper has a lot of possibilities and ways how to be improved. The improvements can be done in each of part of Autorapper.

9.2.1 Easily achievable improvements

The easiest thing to improve would be to add more beats for the users to choose from. This could be done by adding more parameters of beats to Autorapper to calculate the speed of one tick in rhythm and align the speech accordingly.

Another thing that could be easily achievable is to align the speech with different rhythm patterns. Those could be taken from the songs of the famous rappers.

The quality of the Time-Scale modification could be improved by time-scaling only the voiced part of the syllable instead of the whole syllable. The coefficient of the modification would be calculated by subtracting the time of the unvoiced parts from the desirable time of the resulting syllable.

9.2.2 More difficult, but still possible improvements

The website could be improved as well. I noticed that many people that want to try Autorapper do not have any *.wav* recording nearby. It would be nice to put a javascript recorder directly on the website and take it as a source of the speech recording. This would make it easier for the users to try the application.

A support for other languages would be welcomed as well, especially for English. The phoneme recognizer offers English variant but the syllabification would need to be remade.

Another idea was to make a rhythm generator so the rhythm patterns would be generated dynamically. This would be possible and had an interesting effect on the recordings.

9.2.3 Possible exploitation

It would be possible to monetize Autorapper as well. Not in a sense of paying for the service but to put the adverts on the website. However, this would need a lot of traffic on the website in order to be efficient. So at first Autorapper would need to be publicized to let the people know that is exists. I set up the facebook page but I have not done any advertising yet so for now only my friends like it and know about it. The future of this project depends on my professional future and I still have to decide whether I want to run Autorapper at full-scale or leave it as a nice pet-project.

Bibliography

[1] P. Schwarz, *Phoneme Recognition based on Long Temporal Context, PhD Thesis*, Brno University of Technology, 2009

[2] P. Schwarz, P. Matejka, J. Černocký, *Towards Lower Error Rates in Phoneme Recognition* [online], in Proc. TSD2004, Brno, Czech Republic, 2004 [cit. 2015-05-16]. Available at: http://www.fit.vutbr.cz/~schwarzp/publi/2004/tsd2004phn.pdf

[3] FGCN 2010, Held as part of the Future Generation Information Technology Conference. *Communication and Networking International Conference, FGCN 2010, FGIT 2010, Jeju Island, Korea, December* [online]. Berlin: Springer-Verlag New York Inc, 2010, s. 155-161 [cit. 2015-05-03]. ISBN 9783642176036. Available at: https://books.google.cz/books?id=VCa7BQAAQBAJ&pg=PA155&lpg=PA155&dq=Communication+and+Networking+Complexity+Reduction+of+WSOLA-Based+Time-Scale+Modification+Using+Signal+Period+Estimation&source=bl&ots=LmjLDlTDSG&sig=H0jWgifBX-ifE6WP5Zv9V7J9gCk&hl=en&sa=X&ei=1w__VIa_KdftaP2UgLAD&ved=0CCoQ6AEwAg#v=onepage&q&f=false

[4] DUTOIT, Thierry. *Introduction to text-to-speech synthesis.* Boston: Kluwer Academic Publishers, 1997, s. 251-269. ISBN 0-7923-4498-7.

[5] *Mluvíme s počítačem česky.* Vyd. 1. Praha: Academia, 2006, s. 582-599. ISBN 80-200-1309-1.

[6] *Waveform Similarity and Overlap Add (WSOLA) for Speech and Audio: MATLAB Excercise* [online]. 07 Feb 2014, 18 Apr 2014 [cit. 2015-05-03]. Available at: http://www.mathworks.com/matlabcentral/fileexchange/45451-waveform-similarity-and-overlap-add--wsola--for-speech-and-audio

[7] UnbelievableBeats.com. FRIEDMAN, Shaun. *Unbelievable Beats (ASCAP)* [online]. 2014 [cit. 2015-05-03]. Available at: http://www.unbelievablebeats.com/free-beats-free-downloads

[8] *Phoneme recognizer based on long temporal context* [online]. Brno University of Technology, 2004-2006 [cit. 2015-05-03]. Available at: http://speech.fit.vutbr.cz/cs/software/phoneme-recognizer-based-long-temporal-context

[9] SoX. NORSKOG, Lance. *SoX - Sound eXchange* [online]. 1991, 2015 [cit. 2015-05-04]. Available at: http://sox.sourceforge.net/Main/HomePage

[10] Jones E, Oliphant E, Peterson P, *SciPy: Open Source Scientific Tools for Python* [online]. 2001- [cit. 2015-05-04], Available at: http://www.scipy.org/

[11] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. *The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science Engineering* [online]. 2011 [cit. 2015-05-04], Available at: http://www.numpy.org/

[12] Rapping. 2001-. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-09]. Available at: http://en.wikipedia.org/wiki/Rapping

[13] Isochrony. 2001-. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-09]. Available at: http://en.wikipedia.org/wiki/Isochrony

[14] Syllable. 2001-. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-09]. Available at: http://en.wikipedia.org/wiki/Syllable

[15] *Autorap by Smule* [online]. Smule Inc 2012-2014 [cit. 2015-05-10]. Available at: http://www.smule.com/apps#autorap

Appendix A

Content of DVD

Attached DVD contains the following directories:

- `pdf` - technical report in *pdf* format
- `latex` - source codes of technical report
- `web` - *php* source codes and other content of the website
- `autorapper` - main source codes of Autorapper
  - `beats` - beats used by Autorapper
  - `PHN_CZ_SPDAT_LCRC_N1500` - Czech recognition system for the phoneme recognizer
- `examples` - example results from Autorapper
- `video` - video

Appendix B

How to run

The best and easiest way to run the application is through the website, available at www.autorapper.cz.

To run the application without using the website, all the needed files are contained in `autorapper` folder. The main script `rap.sh` executes all the parts of Autorapper. However, the script contains the ftp connection to the hosting server, from where the input file is downloaded. To rap the file which is not downloaded from the website, put the file in `.wav` format into the directory and it will behave like the file was downloaded from the website. The filename must be in `beat_email.wav` format, where `beat` is the number of beat (1-3) and email is your email address to receive the result.