Phonetic transcription

Automatic transcription of vowels to IPA in real time

Marie Hildebrand Grevil

Electronic Engineering & IT

Aalborg University

June 2022

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems

Electronic Engineering & IT

Fredrik Bajers Vej 7
DK-9220 Aalborg Øst
https://www.es.aau.dk/

Title

Phonetic transcription

Subtitle

Automatic transcription of vowels to IPA in real time

Project period

Spring 2022

Student

Marie Hildebrand Grevil

Supervisor

Flemming Christensen

Page count

Body: 22

Total: 37

Abstract

With the ultimate goal of automatically transcribing speech phonetically in real time, this report focuses on the ability to automatically transcribe vowels in real time. For the first part of the report, human speech production was examined, including the resulting acoustic signal, as well as how it can be analyzed using signal processing methods, such as zero-crossing rate and linear predictive coding. With this background, an algorithm was created for automatically detecting vowel segments of speech, as well as estimating which vowel was spoken. The algorithm was programmed and tested in MATLAB. The results did not fulfill the requirements defined in the report. Vowel detection may be difficult to improve on with the goal of running the algorithm in real time, however the vowel estimation had an obvious flaw which may be solvable with further development.

Resumé

Denne rapport er udarbejdet i forbindelse med et 4 måneder langt Elektronik og IT diplomingeniør afgangsprojekt ved Aalborg Universitet. Det langsigtede formål med projektet har været at udvikle en algoritme, der er i stand til automatisk at transskribere tale fonetisk i realtid. Kortsigtet er projektet afgrænset til at transskribere vokaler i realtid. Som grundlag for udviklingen undersøges, hvordan mennesker producerer tale, og hvordan denne tale kan analyseres ved hjælp af signalbehandling. I den forbindelse undersøges sammenhængen mellem zero-crossing rate (ZCR) og stemt tale, herunder vokaler, samt hvordan formanter i særligt vokaler kan estimeres ved hjælp af linear predictive coding (LPC). Der opstilles en række krav, således at det kan undersøges, hvorvidt den udviklede algoritme er i stand til at transskribere vokaler samt automatisk registrere, hvornår tale består af en vokal. Den udviklede algoritme overholdt ikke kravene. Dog var det muligt at se ud fra testresultaterne, at algoritmen er i stand til at nå noget af vejen. Det vurderes, at det ikke umiddelbart vil være muligt at overholde vokalregistreringskravene, så længe algoritmen skal kunne køre i realtid. Til gengæld lader det til, at vokalestimeringen kan optimeres ved også at inddrage den grundlæggende frekvens.

Preface

This report was written as a part of a 4-month final project in Electronic Engineering and IT at Aalborg University.

mhg

Marie Hildebrand Grevil
mgrevi17@student.aau.dk

Contents

Figures

1 Introduction

Automatic speech recognition (ASR) has made serious steps forward in the recent years, in large part due to the incorporation of neural networks and artificial intelligence rather than humans having to build a model. This is generally aimed towards machines being able to interpret what a person is saying and responding to it in a meaningful manner. Just think of home and phone AI assistants like Alexa and Siri, allowing you to check information or even schedule tasks or play music without having to touch the machine. Another great example of ASR being useful is e.g. YouTube's auto-generated closed captions, allowing people with hearing impairments to better enjoy videos without anyone having to sit down and do the work every time a new video is released.

ASR can also be used to interpret the acoustic characteristics of speech rather than the meaning of what is said. This is especially the case when it comes to speech therapy for people with speech impairments, but it could also be extended to people who wish to train their pronunciation in e.g. another language.

In the case of the latter, language learning applications such as Duolingo currently use ASR designed to interpret meaning rather than the exact sounds that the user is articulating, meaning that it is somewhat forgiving when it comes to the spoken exercises. This is no doubt useful for many people whose goal is only to make themselves understood and not necessarily to perfect their pronunciation. However there are also no doubt people who would appreciate the ability to train specific sounds.

A final case where this phonetic ASR would come in handy is with the phonetic transcription of speech. For some linguists, transcribing speech samples is a time-consuming effort which simply has to be done in order for them to analyze or present their work. It can also be useful for e.g. a language teacher or hobby language creator who wants to write the way they say something phonetically without much hassle.

In any case, there is a use and desire for such a tool. This project cannot encompass all aspects of automatic phonetic transcription in the given time frame, however, it can take on a part of the challenge, like being able to automatically transcribe vowels phonetically, before moving on to another.

2 Problem analysis

With the goal of developing a phonetic transcription algorithm focused on vowels, a number of questions are formulated:

1. How can human speech be represented phonetically in writing?
2. What are the below listed characteristics of human speech, and how are they formed in the human vocal tract?
    a. Unvoiced
    b. Voiced
        i. Fundamental frequency
        ii. Formants
3. How can characteristics of human speech, in particular vowels, be extracted using signal processing techniques?

Answering these questions throughout the rest of this chapter is expected to lay a solid foundation for the development of a vowel transcriber.

2.1 Orthography

A language's orthography is essentially its use of the characters in its writing system. The writing system can broadly speaking be logographic (e.g. Egyptian hieroglyphs and Chinese *hanzi*), syllabic (e.g. Japanese *hiragana* and *katakana*), alphabetic (e.g. the Latin and Arabic alphabets), or a mix.

Alphabets are of particular interest when it comes to pronunciation, as there is a great if not full overlap between its symbols and the **phonemes** used in the spoken language that it represents. As such, with optimal use of an alphabet, it is possible to pronounce a written word despite having no prior knowledge of it or its meaning. This contrasts with logographic systems, where each symbol conveys a meaning rather than a pronunciation, so while a logographic symbol may retain its meaning over millenniums, the speaker must know of it, and it could have represented a word with an entirely different pronunciation in the past.

As implied, with optimal use of an alphabet there is no ambiguity regarding its pronunciation. However, the pronunciation of individual languages is not static over time, and the official orthography of a single language may have its origins in a specific region not representative of all speakers. In practicality this means that a person may be using spelling that is more widely understood within their language, despite it not matching their own pronunciation exactly.

Adding to the inaccuracy, a single alphabet can be used by different languages, with each language having its own standard interpretation of how symbols or clusters of symbols are to be pronounced. For example, the English word *hi* and the Danish word *hej* carry the same meaning and are pronounced approximately the same by speakers of their respective languages. However, if asked to pronounce the word exactly as it is written, an English speaker would pronounce *hej* vastly different from a Danish speaker, and vice versa. Even within a single language, there is no guarantee that the same series of symbols represents the exact same pronunciation, e.g. -*ome* in the individual words of the English phrase *come home*.

2.1.1 The International Phonetic Alphabet (IPA)

If the goal of writing is to accurately represent a pronunciation, a common option is using the **International Phonetic Alphabet** (IPA) [1], which is intended to be universal for all listeners. Continuing the example of English *hi* and Danish *hej* from the previous section, with IPA they are both written as [haɪ]. The IPA is based on symbols available in the various versions of the Latin alphabet, mainly supplemented by the Greek alphabet, as well as rotated or capitalized versions of individual symbols. These symbols generally represent the purest version of each sound or **phone**, categorized by how the vocal tract moves to produce said phone. These movements of the vocal tract will be covered briefly later in the chapter. In addition to the base symbols, a variety of diacritics and punctuation is available to describe features such as duration, stress, tone, nasalization, and many more.

The goal of this report is not to explain the meaning of every IPA symbol, and it would be difficult to do so without first going into detail about the production of speech through the vocal tract. Likewise, it would be difficult to go into detail about the latter in a written report without a visual reference for the resulting sounds. For this reason, the official IPA chart is included in appendix A, and where relevant to the report, individual symbols and notations are explained with reference to said chart.

2.2 Human speech production

Human speech is the result of pushing air out of your lungs while shaping your vocal tract (see Figure 1) in patterns changing over time [1]. This produces an acoustic signal that a listener's brain - if they understand your language - can dissect into a meaning. To convey any information, this signal must also differ depending on how the vocal tract was shaped. It is therefore common to describe speech using the **source-filter model**, where a source sound is passed through the acoustic filter that is the vocal tract.

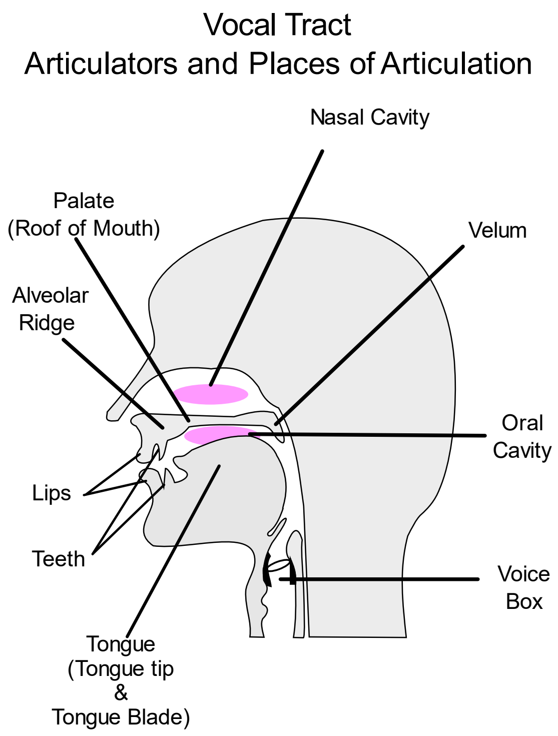


Figure 1: Human vocal tract with labels.
Source: Tavin, CC BY 3.0 [2], via Wikimedia Commons.

2.2.1 Source

The source sound of **unvoiced speech** is simply created by breathing out. For **voiced speech**, the air flow is modified the vocal cord (which is located in the *voice box* in fig. 1) to create its source sound. The vocal cord consists of two folds that vibrate by opening and closing periodically while in use. The result is a source sound with a **fundamental frequency** (F0) corresponding to the frequency of the vibrating folds, see Figure 2. F_0 is not constant for a single person, though the length of the person's vocal cords generally defines their F_0 range. Males, having longer vocal cords, are said to have a range of 80 to 200 Hz, and females 150 to 350 Hz [3]. These exact values vary depending on source material [2], and they are not a hard limit by any means.

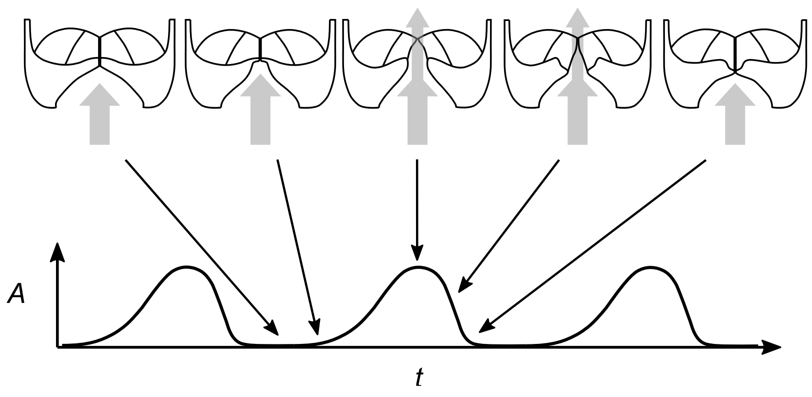


Figure 2: The opening and closing of the vocal cords, and the resulting acoustic signal.
Source: Gerazov, CC BY-SA 4.0 [3], via Wikimedia Commons.

2.2.2 Filter

All parts in Figure 1 other than the voice box are considered the filter in the source-filter model. By constricting airflow between various parts or changing the shape of the oral cavity, the source sound is modified. The former is the main component of pulmonic consonants, a table of which can be found in appendix A. Here they are listed with three dimensions: the location where the obstruction happens, the movement of the involved parts, and whether the consonant is voiced or unvoiced. As an example, both [p] and [b] are bilabial plosives. This means that the phones are formed by closing the lips (bilabial), which temporarily stops the airflow and lets pressure build up enough to break through the closure (plosive) [1]. This can be compared with another set of plosives, [k] and [g]. These two are velar instead of bilabial, meaning that the closure takes place between the tongue and the back of the mouth instead of between the lips, but the pattern of stopping and releasing the airflow remains the same. Similarly, [p] and [b] can be compared with the bilabial nasal [m]. Here the airflow is blocked by the lips closing, and the airflow is redirected through the nasal cavity as one continuous sound.

While continuous sounds are not exclusive to vowels, vowels do exclusively consist of continuous sounds. Vowels are characterized by minimal obstruction in the vocal tract and are also typically voiced. They are also listed with three dimensions: close/open, front/back, and rounded/unrounded. The vowel being open or close refers to the height of the tongue, with close meaning the tongue is close to the roof of the mouth, and open meaning it rests in the bottom of the mouth. Front and back again refer to the tongue, but, as the terms suggest, whether the tongue is towards the front or the back of the mouth. Finally rounded and unrounded refer to the shape of the lips [4]. Unlike what is the case for consonants, the categorization of vowels has more to do with their resonant frequencies, **formants**, than it has to do with a unique position and shape of the tongue and lips [4]. That is not to say that they do not matter, but rather that these categories have been defined based on their influence on the formants.

2.2.3 Formants

As mentioned in section 2.2.2, a formant is a resonant frequency, typically in the context of music or speech. Formants are, in other words, frequencies that have been amplified through resonance and become dominant in the acoustic signal. For speech, it is generally considered that the two lowest frequency formants, F_1 and F_2 are most important in distinguishing vowels [4] [3], corresponding roughly to tongue height and tongue backness respectively. The third formant, F_3, has been tied to lip rounding. Formants F_4 and above are not considered as important in the context of vowels [4]. Because of this, vowels are often visualized according to their first two formants, see Figure 3.

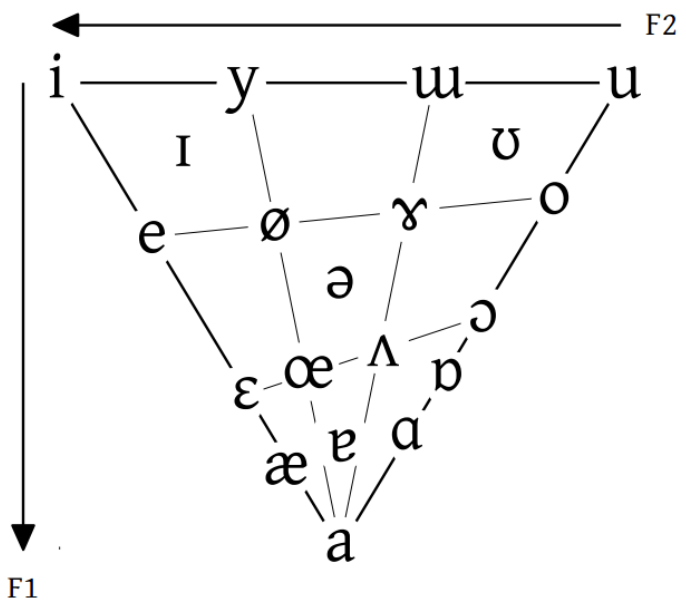


Figure 3: A selection of IPA vowels arranged in an ideal triangle according to their first two formants. Source: Kwamikagami, CC BY-SA 3.0 [3], via Wikimedia Commons.

The exact formants of e.g. [a] vary from person to person, however they generally exist in the same range. For human speech, it is a rule of thumb that a formant exists for every 1 kHz, meaning that F_1 is generally expected to be in the range of 0 to 1 kHz, F_2 in the range of 1 to 2 kHz, and so on. However, this is not a guarantee, as e.g. for some vowels F_2 and F_3 merge due to their proximity with each other.

2.2.4 Phone transition

It is easy to think of phones as discrete units. This is how they are represented with IPA and with alphabets in general. However, for the acoustic signal that is speech, this is not the case, as the transition from one vocal tract shape to another is not instant, and the speech continues while the change is taking place, resulting in **coarticulation** [7] [8]. Even if attempting to speak slowly and transition sharply from one phone to another, it is difficult to truly separate them from each other. In regular speech, this becomes near impossible. One example is short vowels trending towards the neutral vowel [ə] (also known as *schwa*) as the mouth quickly has to prepare for the next phone [4]. Another is with words of Latin origin with the prefix *in-*, often being a negation to the word that follows. The word *inpossible* has morphed into *impossible*, because in preparation for the bilabial [p], the [n] first has to transition into [m], and over time it has been entirely replaced [4] [8].

2.3 Speech in signal processing

Speech is an analog signal, and in the modern world digital computers are by far dominant over their analog counterparts. Because of this, using a computer for signal processing most often demands that the speech is converted into a digital signal first, turning both time and amplitude discrete rather than continuous. Speech travels through air forming pressure, which, when passed through a microphone, is translated into voltage. For the time aspect, the microphone is sampled at a regular interval also known as the **sample rate**. When the microphone is sampled, the analog voltage is run through an analog-to-digital converter (ADC) and quantized into a discrete value, so that it can be stored and processed digitally. This is known as **pulse-code modulation** (PCM), and it is the most straightforward method of storing audio in digital form [1].

2.3.1 PCM storage

A classic file format for PCM is WAVE, with the extension `.wav`. It is a variation of Microsoft's Resource Interchange File Format (RIFF), which is also the basis for the video format AVI among others. A WAVE file consists of three *chunks*. The first chunk contains the letters `RIFF`, the remaining size of the file in bytes, and the letters `WAVE`, which specifies which chunks the file is expected to contain. Following that is a chunk containing the format information of the file. This includes specifying if the data is stored as PCM or compressed, how many audio channels the data consists of, as well as bits per sample, sample rate, and other information regarding how to interpret the data. The final chunk is for data and includes the size of the data itself in bytes.

PCM, while it has the benefit of being lossless, i.e. the data stored is exactly what was recorded, unsurprisingly requires a large amount of storage space. The data can be compressed in ways that e.g. take advantage of repetitive data (lossless) or reduce the number of bits per sample (lossy). Ultimately, in order to be played back through a speaker or analyzed, the data will have to be uncompressed and restored to its waveform.

2.3.2 Short-time analysis and framing

Due to the waveform of speech changing over time, in other words being dynamic, analyzing an entire speech recording at once will not yield much information about the characteristics of the speech. Reducing the length of the audio to analyze makes the signal quasi stationary. This means that, while the waveform is likely still changing from the beginning of the window to the end, the changes are relatively small, and the data within the window is approximately stationary [4]. This allows for the data to be split up into frames consisting of consecutive windows. A commonly recommended window length for speech analysis is one corresponding to 20 [5] or 25 ms, with some sources going as far as suggesting anywhere between 15 [4] and 40 ms. One thing to consider when choosing window length is whether the fundamental frequency can fit within one frame. If analyzing a series of frames, having some frames with F_0 and some without may cause inconsistencies [4]. Going by the frequency ranges listed in section 2.2.1, the period of F_0 has a length of up to 12.5 ms, meaning that 20 ms windows should be sufficient.

In speech analysis, the window function used when framing the data is typically Hamming [4] or Hann [6]. They both cause the windowed data to slope towards the edges [5], and if the windows are placed one after another, this means that the data in the transition between frames is attenuated and ultimately lost. To counter this, it is recommended to use overlap between frames, such that a frame shares a percentage of its data with the previous frame, and the next. A commonly suggested amount of overlap is 50 % or 10 ms [5].

2.3.3 Distinguishing voiced and unvoiced speech

With the description of voiced sounds having a fundamental frequency, while unvoiced sounds do not, it is not far-fetched that much of voiced speech's energy is contained in the lower frequencies, compared to unvoiced speech's higher frequencies. This is even visible in the waveform, where the high energy low frequency appears to carry the higher frequencies, so that the number of times the waveform crosses 0 more closely resembles the fundamental frequency, see Figure 4.

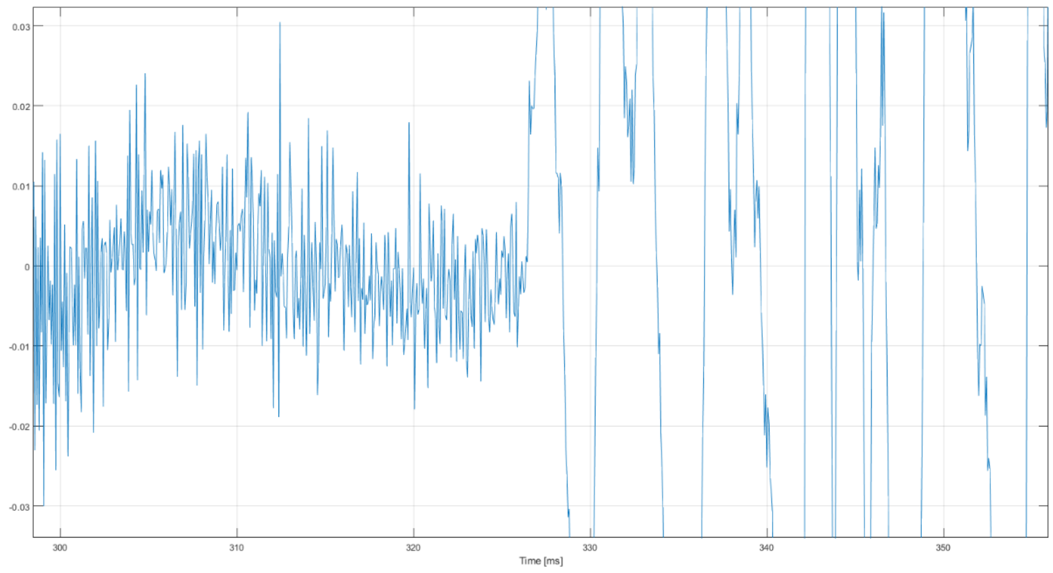


Figure 4: Close up of a speech waveform transitioning from unvoiced [s] to voiced [a].

Because of this, the **zero-crossing rate**, i.e. the frequency of the waveform's sign change, carries information about whether the phone is voiced or unvoiced. By counting the number of changes N within a frame, the zero-crossing rate can be calculated:

[9] Equation 1

$$ZCR = \frac{N \cdot f_s}{2 \cdot W_L}$$

Where f_s is the sample rate, and W_L is the window length.

If the calculated zero-crossing rate is sufficiently low to not be unvoiced, the energy at this frequency can then be examined using a Fourier transform to ensure that the zero-crossing rate is not the result of random noise in between voiced segments. If the energy is above a selected threshold, it can be assumed that the frame contains voiced speech [9] [10] [1].

Nasal consonants are like vowels in that they also have a fundamental frequency and formants. Because of that, this method will tend to lump nasals and vowels together, when it may be of interest to separate vowels from consonants on top of voiced from unvoiced. Nasal consonants tend to have a lower zero-crossing rate than vowels, and so instead of only setting an upper frequency boundary, a lower one can be set as well.

2.3.4 Formant estimation

As a human, it is not unreasonable to look at a frequency spectrum containing formants and being able to estimate them by the spectral envelope, see Figure 5. For a speech recognition algorithm analyzing multiple frames per second, this would obviously not suffice, but thankfully there are ways to calculate a formant estimate.

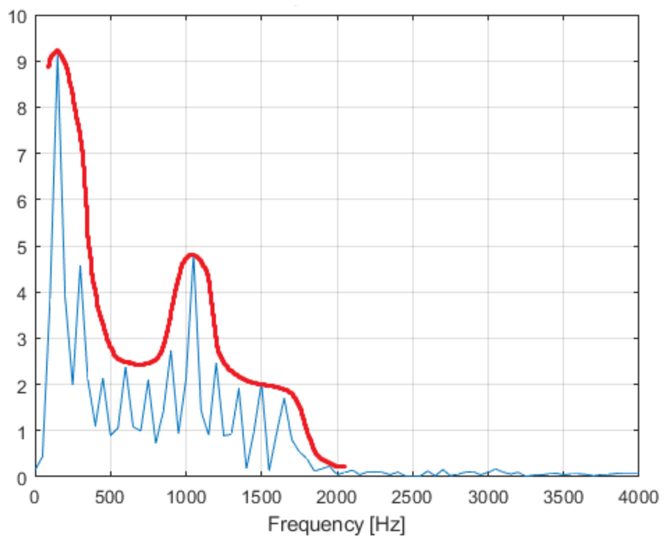


Figure 5: An audio frame's frequency spectrum with a sketched envelope.

Continuing with the source-filter model covered in section 2.2, if the waveform being analyzed is the result of an impulse train being passed through a filter, an inverse filter would instead be applied to the waveform to obtain the source signal. If the coefficients of the inverse filter are approximated, it can then be inversed to form as estimate of the original vocal tract filter. The transfer function of said filter roughly corresponds to, but is not equal to, the spectral envelope [4], see Figure 6, and the peaks are the result of the inverse filter's complex conjugate roots. Such an inverse filter can be obtained through **linear predictive coding** (LPC) [6] [9].

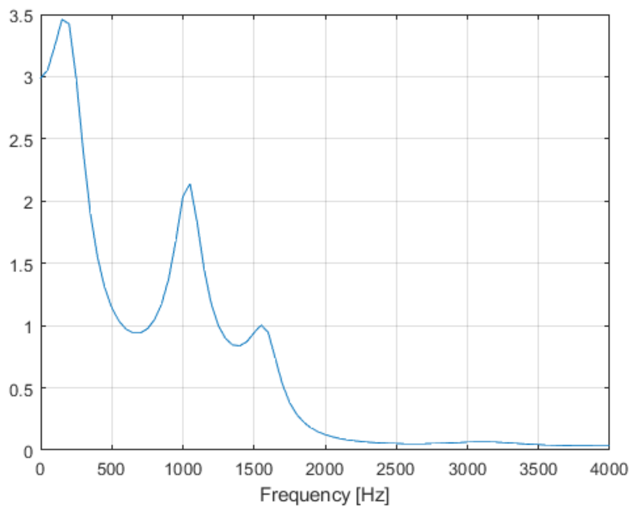


Figure 6: The LPC estimated transfer function corresponding to Figure 5.

2.3.4.1 Linear predictive coding (LPC)

LPC takes advantage of the waveform of speech being somewhat predictable based on a handful of previous samples. This section will not go in-depth with the individual methods that exist for calculating the LPC coefficients, but rather attempt to cover what the methods accomplish. With the assumption that every sample is the result of a linear combination of p previous samples, it is possible to set up an equation for every single sample in the frame. A rule of thumb is that each formant found between 0 Hz and the Nyquist frequency corresponds to one complex root pair. Expecting a formant for roughly every 1 kHz [3] results in $p = f_s/1000$, sometimes adding a few more to account for LPC picking up on e.g. the fundamental frequency [5]. The coefficients a_k are then the solution to all these equations that results in the smallest error, *e*, between the sampled value and the value predicted by the coefficients [4]:

Equation 2

$$x(n) = \sum_{k=1}^{p} a_k \cdot x(n-k) + e(n)$$

Where *x(n)* is the predicted sample.

What various methods of solving this system have in common is that they ultimately require multiplying the inverse of a matrix with a vector to obtain the coefficients [9]. Once the set of coefficients has been calculated, the prediction polynomial can be formed [2]:

Equation 3

$$P(z) = 1 - \sum_{k=1}^{p} a_k \cdot z^{-k}$$

Solving for complex roots results in a number of roots equal to the LPC order *p*. Only the complex conjugate root pairs correspond to formants, with their frequencies being located between 0 Hz and the Nyquist frequency. The frequencies and bandwidths of each complex root pair is calculated by:

[6] Equation 4

$$F_i = \frac{f_s}{2\pi} \cdot \theta_i$$

Where F is a formant, and θ is the angle of the root in radians.

[8] Equation 5

$$B_i = -\frac{1}{2} \cdot \frac{f_s}{2\pi} \cdot \ln r_i$$

Where *B* is the formant's bandwidth, and *r* is the size of the root.

Formants with a frequency close to zero, e.g. below 90 Hz [9] [6], are discarded, as they should be higher than the fundamental frequency [6]. Likewise, if the formant's bandwidth is too large, e.g. above 400 Hz [4], it is a sign that the root is too spread out to represent the narrow peak of a formant [5]. The remaining formants are sorted according to size, giving F_1 the lowest frequency [1]. For the frame presented in both Figure 5 and Figure 6, the first three estimated formants are 202, 1040 and 1590 Hz. Comparing with the peaks in both figures, this appears to be a reasonable estimate, although F_1 is low enough that it could be F_0.

3 Requirements

To be able to judge whether the vowel transcriber performs well, it is a good idea to determine a set of minimum requirements prior to testing. As there are no ISO standards for phonetic transcription, the requirements are formulated with the intended use of the algorithm in mind.

Real time

For live feedback regarding the user's pronunciation of a vowel, such as in speech therapy or language training, it is necessary that the algorithm can run in real time. This means that the audio cannot be enhanced or analyzed through preprocessing, and analysis should mainly rely on current and past frames. To use future frames, the algorithm would need to delay the output to match the number of future frames needed, up to a maximum of 0.1 seconds. For automated IPA transcription purposes, the algorithm does not need to run in real time, however it is likely easier to change an algorithm away from real time than it is to change it towards.

Platform

The intended distribution is as a standalone application on a personal computer, be it a desktop computer or a smartphone. Proof of concept and testing should be done using a math tool such as MATLAB. It should be programmed in such a way that it can be implemented in another coding language without MATLAB's specialized functions and tools.

Input noise

It is assumed that only one person is speaking at a time, and that the recording takes place under quiet circumstances with no music and minimal background noise such as traffic.

Framing

The window length of each frame should be 20 ms, with a frame step of 10 ms. The algorithm should be capable of fully analyzing a single frame before continuing to the next.

Formants

The algorithm should be able to estimate formants using LPC. The order of LPC should be equal to $f_s/1000$, rounded to an integer. A minimum of three formants should be kept for use in vowel estimation.

Vowel detection

Using zero-crossing rate and energy, the algorithm should be able to determine if a frame contains a vowel.

Vowel estimation

With formants and vowel detection, the algorithm should estimate which vowel was spoken during vowel segments and provide the corresponding IPA symbol. The algorithm should be designed to distinguish between at least five different vowels.

3.1 Testing

The test will be performed with the vowels [a], [ɘ], [i], [ɑ] and [u]. In contexts where IPA symbols are not available, they will be written as **a**, **e**, **i**, **o** and **u** respectively.

All speech clips will be selected from the linguistic resource Sound Comparisons [6], where a large collection of sound clips is available for download along with their IPA transcriptions. For each vowel to be tested, 20 clips will be selected where the vowel is present in the transcription, preferring long vowels and discarding clips where the vowel to be tested is part of a diphthong (combined vowel).

For each speech clip, the vowel segments will be manually determined by listening. The first and last frame of each vowel segment will be noted along with which vowel to test.

For the sake of uniformity in analysis and testing, each speech clip will be resampled to 16 kHz prior to analysis.

Vowel detection

For each frame manually marked as containing a vowel, at least 90 % should be correctly detected.

For each frame not marked as containing a vowel, at least 90 % should be correctly detected.

The total number of vowel segments detected should match the number of manually determined vowel segments by ±10 %.

Vowel estimation

For each vowel segment to be tested for vowel estimation, the correct vowel should be the highest match for at least 90 % of the segments.

For each vowel segment to be tested for vowel estimation, the correct vowel should be in the top two matches for at least 95 % of the segments.

4 Algorithm

This chapter's purpose is to describe the vowel transcriber that has been developed based on the knowledge presented in chapter 2, as well as the requirements defined in chapter 3. As a proof of concept, the algorithm was coded in MATLAB, using several of the software's specialized functions to analyze one or more audio files of variable length.

4.1 Preparation

File analysis is prepared by storing values that cannot be made discrete until the sample rate of the input has been discovered, as well as the complete paths to the audio files that will be analyzed.

4.1.1 Frame length and overlap

The window length of each frame is stored, as well as the time overlap between consecutive frames.

```
FrameLengthMillis = 20
FrameStepMillis = 10
```

4.1.2 List of input audio files

To allow for bulk analysis of multiple files, an input directory is chosen, and each compatible audio file contained within said directory, excluding subfolders, is added to a string array of input filenames. The chosen input directory is the subfolder `Input` in the current work folder, and the audio file formats provided on line 2 are the ones supported by MATLAB [6].

```
1.  InputPath = pwd+"\Input"
2.  InputFormats = ["*.aifc", "*.aiff", "*.aif", "*.au", "*.flac", "*.ogg", "*.opus",
    "*.wav", "*.mp3", "*.m4a", "*.mp4"]
3.  for FormatIndex = 1:length(InputFormats)
4.      TempFiles = dir(fullfile(InputPath, InputFormats(FormatIndex)))
5.      for FileIndex = 1:height(TempFiles)
6.          InputFiles                                                             =
    [InputFilesconvertCharsToStrings(TempFiles(FileIndex).name)]
7.      end
8.  end
```

4.2 File analysis

The file analysis takes place in a loop going through each filename stored in the `InputFiles` array. The index of the file currently being analyzed is stored in the variable `FileIndex`. This variable is used for accessing the appropriate filename in the array, as well as identifying the file in question when storing test results.

4.2.1 Audio import

The contents of the audio file are first imported into a struct containing the uncompressed waveform data and the file's sample rate. If the data consists of multiple channels, they are merged into one by storing their mean. The sample rate in hertz, given by the file's metadata, is then stored as its own variable, allowing the struct to be cleared from memory.

```
FileContent = importdata(fullfile(InputPath+'\'+InputFiles(FileIndex)))
RawAudio = zeros(height(FileContent.data), 1)
for SampleIndex = 1:height(FileContent.data)
```

```
    RawAudio(SampleIndex) = mean(FileContent.data(SampleIndex, :))
end
SampleRate = FileContent.fs
```

4.2.2 Sample rate dependent variables

The frame length and frame step provided in section 4.1.1 are continuous variables and will need to be made discrete to apply them to the likewise discrete waveform. For this, the sample rate is used. The same goes for the linear prediction order, which is covered in section 2.3.4.1. Since the number of samples the data consists of is known, the number of frames that it can be divided into is calculated in advance and used to establish the loop containing frame analysis. Lastly, the window function is calculated for the given frame length.

```
FrameLength = floor(SampleRate*FrameLengthMillis/1000)
FrameStep = floor(SampleRate*FrameStepMillis/1000)
LinPredOrder = floor(SampleRate/1000)
NumFrames = floor((length(RawAudio)-FrameLength+FrameStep)/FrameStep)
Window = hamming(FrameLength)
```

4.3 Frame analysis

The frame analysis loop goes through every integer from 1 to the number of full frames possible, with the index of the frame currently being analyzed stored in the variable `FrameIndex`. This index is used to calculate the corresponding indexes in `RawAudio`, as well as for storing information about detected vowels for future frames.

4.3.1 Windowing

First the `RawAudio` index to start reading from is calculated, compensating for the fact that MATLAB arrays always begin with an index of 1 instead of 0. Then the window function is applied to the samples starting at this index.

```
DataOffset = (FrameIndex-1)*FrameStep
RawFrame = RawAudio(DataOffset+1:DataOffset+FrameLength)
WinFrame = Window.*RawFrame
```

4.3.2 Fourier transform

The frequency components of the window are calculated and stored.

```
FreqFrame = fft(WinFrame)
```

4.3.3 Zero-crossing rate

The sign of each sample in `WinFrame` is stored in a new array, with 1 corresponding to a signed value, and 0 to an unsigned value. Going through each sample, a counter is incremented every time a sample differs from the previous one, meaning that a zero-crossing has taken place. Using the number of zero-crossings as well as the number of samples and sample rate, the zero-crossing rate in hertz is calculated. The amount of energy found at this frequency is then stored for use in vowel detection.

```
SignFrame = sign(WinFrame)
ZeroCrossingCount = 0
for SampleIndex = 2:FrameLength
    if abs(SignFrame(SampleIndex)-SignFrame(SampleIndex-1))
        ZeroCrossingCount = ZeroCrossingCount+1
    end
```

```
end
ZeroCrossingRate = SampleRate*ZeroCrossingCount/(2*FrameLength)
ZeroCrossingEnergy                                                  =
abs(FreqFrame(round(ZeroCrossingRate/(SampleRate/FrameLength))+1))
```

4.3.4 Formants

In order to estimate the formants of a frame, first the linear prediction coefficients are calculated. After that, the complex roots of a polynomium containing said cofficients are estimated, and the roots are translated to formants, as covered in section 2.3.4.1. Included is also protection against coefficients that are *not a number*, as sometimes a solution does not exist.

```
LinPredCoeffs = lpc(WinFrame, LinPredOrder)
Formants = zeros(3, 1)
if sum(isnan(LinPredCoeffs)) == 0
    Roots = roots(LinPredCoeffs)
    Roots = Roots(imag(Roots) > 0)
    Rads = atan2(imag(Roots), real(Roots))
    Freqs = Rads.*(SampleRate/(2*pi))
    Band = (-1/2*(SampleRate/(2*pi))).*log(abs(Roots))
    FormantIndex = 0
    Formants = zeros(NumFormants, 1)
    for RootIndex = 1:length(Roots)
        if Freqs(RootIndex) > 90 && Band(RootIndex) < 400
            FormantIndex = FormantIndex+1
            Formants(FormantIndex) = Freqs(RootIndex)
        end
    end
    Formants = sort(Formants)
end
```

4.3.5 Vowel detection

To detect vowels, first the zero-crossing rate is examined, as covered in section 2.3.3. The zero-crossing rate is divided into three possible states: high (1), medium (0), and low (-1). This state is initiated as medium and stored between frames, and an uneven threshold has been put in place to ensure that small fluctuations exceeding the boundary between each state do not cause the state to rapidly change back and forth. If the state is medium, the energy level at the zero-crossing frequency is then checked, and if it is sufficiently large, it is estimated that the frame contains a vowel.

```
if FrameIndex == 1
    Vowel = zeros(1, NumFrames)
    State = 0
end
if State < 1 && ZeroCrossingRate > 5000
    State = 1
elseif State < 0 && ZeroCrossingRate > 400
    State = 0
elseif State > -1 && ZeroCrossingRate < 350
    State = -1
elseif State > 0 && ZeroCrossingRate < 4000
    State = 1
end
if State == 0 && ZeroCrossingEnergy > 1
    Vowel(FrameIndex) = 1
end
```

4.3.5.1 Experimental smoothing of vowel detection

While developing the algorithm, it was found that spikes appear in the vowel detection. As an example, a vowel consisting of 20 frames might have a single frame within it that, according to the set thresholds, does not contain a vowel, and vice versa. To examine how this can be smoothed out, an extra algorithm is executed after all frames have been processed. This algorithm goes through every frame of the `Vowel` array to summarize where each vowel segments begins and ends. Following that, every transition between vowel and non-vowel is examined, and if a sufficiently small gap is detected between the current transition and the previous one, said gap is erased. As this algorithm only requires a set number of previous frame outputs, it could run on a frame-to-frame basis, but for use in real time, it would require either a delay or a running correction of previous vowel detection output.

```
VowelCount = 0
for FrameIndex = 1:NumFrames
    Change = Vowel(FrameIndex)-Vowel(FrameIndex-1)
    % Note start of a vowel segment (1st col):
    if Change == 1
        VowelCount = VowelCount+1
        VowelSegments(VowelCount, 1) = FrameIndex
    % Note end of a vowel segment (2nd col):
    elseif Change == -1
        VowelSegments(VowelCount, 2) = FrameIndex
    end
    % If final vowel segment is not closed off, close it:
    if FrameIndex == NumFrames && VowelCount > 0
        if VowelSegments(VowelCount, 2) == 0
            VowelSegments(VowelCount, 2) = FrameIndex
        end
    end
end
% Smooth the gaps:
if VowelCount > 0
    TempVowelCount = 0
    for VowelIndex = 1:VowelCount
        TempVowelCount = TempVowelCount+1
        TempVowelSegments(TempVowelCount, :) = VowelSegments(VowelIndex, :)
        if VowelIndex > 1
            % Erase gaps of 1 frame between vowel segments:
            if VowelSegments(VowelIndex, 1)-VowelSegments(VowelIndex-1, 2) < 2
                Vowel(1, VowelSegments(VowelIndex-1, 2):VowelSegments(VowelIndex, 1)) = ones(1, VowelSegments(VowelIndex, 1)-VowelSegments(VowelIndex-1, 2)+1)
                TempVowelSegments(TempVowelCount-1, 2) = TempVowelSegments(TempVowelCount, 2)
                TempVowelSegments(TempVowelCount, :) = []
                TempVowelCount = TempVowelCount-1
            end
        end
    end
    VowelCount = 0
    VowelSegments = []
    for VowelIndex = 1:TempVowelCount
        % Only include vowel segments longer than 2 frames:
        if TempVowelSegments(VowelIndex, 2)-TempVowelSegments(VowelIndex, 1) > 2
            VowelCount = VowelCount+1
            VowelSegments(VowelCount, :) = TempVowelSegments(VowelIndex, :)
        % Otherwise erase the vowel segment from the array:
        else
            Vowel(1, TempVowelSegments(VowelIndex, 1):TempVowelSegments(VowelIndex, 2)) = zeros(1, TempVowelSegments(VowelIndex, 2)-TempVowelSegments(VowelIndex, 1)+1)
        end
    end
end
```

4.4 Vowel estimation

This part of the algorithm has not yet been integrated into the main code. First, a set of baseline F_1 and F_2 values is stored for each vowel [a], [ə], [i], [ɑ] and [u]. The vowels are sorted in the order that they are listed here, starting at [a] = 1 and ending with [u] = 5.

```
Ref = [690, 1500; 450, 1420; 270, 2320; 640, 1100; 310, 830]
```

For every frame of the vowel segment being analyzed, the calculated F_1 and F_2 values are stored as a row in the matrix `VowelData`. Then, for each of the five possible vowels, a loop is initiated with the index `VowelIndex` corresponding to their row in the `Ref` matrix. The mean and the 50th percentile of the squared error between the data and the reference is stored, then square rooted, and their size relative to the reference is calculated. What remains are four scalars, and the smaller each of them is, the more said data matches the current vowel. Different combinations of these scalars were tried out before deciding on what is shown below.

```
VowelEst = zeros(1, height(Ref))
for VowelIndex = 1:height(Ref)
    ResF1 = (VowelData(:, 1)-Ref(VowelIndex, 1)).^2
    ResF2 = (VowelData(:, 2)-Ref(VowelIndex, 2)).^2
    PercF1 = sqrt(prctile(ResF1, 50))/Ref(VowelIndex, 1)
    PercF2 = sqrt(prctile(ResF2, 50))/Ref(VowelIndex, 2)
    MeanF1 = sqrt(mean(ResF1))/Ref(VowelIndex, 1)
    MeanF2 = sqrt(mean(ResF2))/Ref(VowelIndex, 2)
    VowelEst(VowelIndex) = PercF1*MeanF1+PercF2*MeanF2
    if isnan(VowelEst(VowelIndex))
        VowelEst(VowelIndex) = 10
    end
end
[Value, Index] = sort(VowelEst)
```

With `VowelEst` being sorted from lowest to highest, the first element of `Index` is the index of the vowel with the highest match, i.e. the best guess. Similarly the second element corresponds to the next best guess.

5 Test results

The results of the test described in section 3.1 are presented here. For the full data tables, see Appendix B.

5.1 Vowel detection

Total number of frames: 10464
Vowel frames expected: 3232
Non-vowel frames expected: 7232
Vowel segments expected: 150

Without smoothing

	Detected vowel frame	Detected non-vowel frame
Expected vowel frame	1513 (46.8 %)	1719 (53.2 %)
Expected non-vowel frame	374 (5.17 %)	6858 (94.83 %)

Vowel segments detected: 273 (182 %)
Number of clips with no vowel frames detected: 10 (10 %)

With smoothing

	Detected vowel frame	Detected non-vowel frame
Expected vowel frame	1519 (47.0 %)	1713 (53.0 %)
Expected non-vowel frame	322 (4.5 %)	6910 (95.5 %)

Vowel segments detected: 129 (86 %)
Number of clips with no vowel frames detected: 18 (18 %)

5.2 Vowel estimation

Number of speech clips for each vowel: 20

For each expected vowel, the columns count how many times each detected vowel appeared in the top 1 and top 2 respectively. For example, for all the clips expecting [a], [ə] appeared as the best guess twice and as the next best guess 7 times, for a total of 9 appearances in the top 2.

Count

Detected	a		ə		i		ɑ		u	
Expected	Top 1	Top 2	Top 1	Top 2	Top 1	Top 2	Top 1	Top 2	Top 1	Top 2
a	9	10	2	9	0	0	1	4	8	8
ə	3	3	9	19	0	0	0	1	8	8
i	1	1	0	4	15	16	0	0	4	4
ɑ	4	4	0	2	0	0	14	18	2	2
u	0	0	1	1	0	1	0	0	19	19

Percent

Detected	a		ə		i		ɑ		u	
Expected	Top 1	Top 2	Top 1	Top 2	Top 1	Top 2	Top 1	Top 2	Top 1	Top 2
a	45 %	50 %	10 %	45 %	0 %	0 %	5 %	20 %	40 %	40 %
ə	15 %	15 %	45 %	95 %	0 %	0 %	0 %	5 %	40 %	40 %
i	5 %	5 %	0 %	20 %	75 %	80 %	0 %	0 %	20 %	20 %
ɑ	20 %	20 %	0 %	10 %	0 %	0 %	70 %	90 %	10 %	10 %
u	0 %	0 %	5 %	5 %	0 %	5 %	0 %	0 %	95 %	95 %

6 Discussion

Vowel detection

The vowel detection aspect of the algorithm appears to be on the conservative side when it comes to marking frames as vowels. This is not necessarily a bad thing, as it is possible that more frames were manually marked as being vowels. This is because the vowel signal blends into the surrounding phones, as covered in section 2.2.4. While manually marking the vowels, the intent was to include as much of each vowel as possible, rather than settling for the halfway point between phones, but it is uncertain whether that was the right call.

The number of frames marked as vowels does not change significantly with the smoothing. Instead the substantial change comes with the number of individual vowel segments that are detected, which makes sense, as the purpose of the smoothing is to bridge the gaps between nearby vowel segments that are too close to be separate vowels. However, the downside with the smoothing appears to lie in the number of speech clips with no vowels detected at all. Looking through the test data in Appendix B.3 shows that the affected clips in some cases contained many vowel segments, but each very short and spread out. A possible solution could be to only apply the smoothing in favor of vowel segments, so that short vowel segments are not erased.

The verdict here is that the algorithm **does not** detect enough vowel frames, as the percentage is not even close to exceeding 90 %. The algorithm **does** avoid marking many non-vowel frames as vowels, as the value is around 95 % both before and after smoothing. Changing the thresholds for vowel detection might help with detection of vowel frames, but in return it is likely to cause more false positives as well, bringing the non-vowel frame success down.

Vowel estimation

The vowel estimation clearly works well in favor of [u], which is the vowel with the lowest F_1 and F_2 of the ones used. The estimation appears to be biased in direction of [u], which might explain why the estimation of exactly this vowel is so successful. One possible explanation is something that was mentioned in section 2.3.4.1: the frame that the formant calculations were based on contained the vowel [a], which generally has F_1 and F_2 around 900 Hz and 1.6 kHz respectively. These values roughly corresponded to the 2nd and 3rd estimated formants, leaving F_1 at around 200 Hz. It is possible that this is happening across many frames and many speech clips, which is supported by the fact that a majority of the wrong-guessed vowels are estimated to be [u]. It would be necessary to manually adjust the formant frequency threshold, or include an estimation of F_0 in order to dynamically choose the threshold.

The verdict is that top 2 guess for [ə] fulfills the requirement, and so does the top 1 guess for [u]. Unfortunately this is not necessarily a success. The reasoning behind [u] not being an obvious success has been discussed in the paragraph above. As for [ə], it is considered the neutral vowel that other vowels naturally trend towards. As such, these two vowels fulfill the requirement not necessarily because the model is working well, but rather due to model bias. Ensuring that F_0 is not unintentionally included in the list of formants will be the first step towards bettering the model, as it may keep the vowels from trending towards the lower frequencies.

Real time

While the real time aspect was not included in the testing, it bears saying that the requirement for the algorithm to work in real time might be detrimental to the goal of the algorithm. This is because it excludes pre- and postprocessing, which could help provide a more precise analysis of the speech data. It is not necessarily a lost cause, as speech therapy tools working in real time do exist, even if they are not designed to cover a wide array of phones at once.

7 Conclusion

Zero-crossing rate shows some potential in automatically determining the vowel segments of speech.

Linear predictive coding has proved useful in estimating the formants of a voiced segment of speech.

The developed algorithm utilizing both zero-crossing rate and linear predictive coding is able to detect and estimate vowels to some extent, but it does not fulfill the requirements set forth in section 3.1. It is highly likely that with some refinement, the algorithm will perform better in terms of estimating vowels, though the vowel detection aspect might not improve much as long as the real time compatibility requirement remains.

Bibliography

[1] D. G. Childers, Speech processing and synthesis toolboxes, John Wiley & Sons, Inc., 2000.

[2] J. N. Holmes, Speech synthesis and recognition, Van Nostrand Reinhold (UK), 1988.

[3] Creative Commons, "Attribution 3.0 Unported (CC BY 3.0)," [Online]. Available: https://creativecommons.org/licenses/by/3.0/. [Accessed 1 June 2022].

[4] N. Peleg, "Linear Prediction Coding," March 2009. [Online]. Available: http://cs.haifa.ac.il/~nimrod/Compression/Speech/S4LinearPredictionCoding2009.pdf. [Accessed 2 June 2022].

[5] Creative Commons, "Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)," [Online]. Available: https://creativecommons.org/licenses/by-sa/4.0/. [Accessed 1 June 2022].

[6] M. R. Schroeder, Computer speech: recognition, compression, synthesis, Springer, 1999.

[7] H.-S. Kim, "Linear Predictive Coding is," [Online]. Available: https://ccrma.stanford.edu/~hskim08/lpc/. [Accessed 2 June 2022].

[8] Creative Commons, "Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)," Creative Commons, [Online]. Available: https://creativecommons.org/licenses/by-sa/3.0/. [Accessed 1 June 2022].

[9] K. K. Paliwal, J. G. Lyons and K. K. Wojcicki, "Preference for 20-40 ms window duration in speech analysis," 2010.

[10] K. Rao, V. R. Reddy and S. Maity, "Appendix," in *Language Identification Using Spectral and Prosodic Features*, Springer, 2015, pp. 87-92.

[11] T. M. Bőhm and G. Németh, "Algorithm for formant tracking, modification and synthesis," BME Department of Telecommunications and Media Informatics, 2007.

[12] S. Goswami, P. Deka, B. Bardoloi, D. Dutta and D. Sarma, "ZCR Based Identification of Voiced, Unvoiced and Silent Parts of Speech Signal in Presence of Background Noise," in *International Conference on Computation and Communication Advancement*, Kalyani, West Bengal, India, 2013.

[13] R. G. Bachu, S. Kopparthi, B. Adapa and B. D. Barkana, "Voiced/Unvoiced Decision for Speech Signals Based on Zero-Crossing Rate and Energy," in *Advanced Techniques in Computing Sciences and Software Engineering*, 2010.

[14] R. C. Snell and F. Milinazzo, "Formant Location From LPC Analysis Data," *IEEE Transactions on Speech and Audio Processing*, April 1993.

[15] T. Bäckström, "Linear prediction," 10 April 2019. [Online]. Available: https://wiki.aalto.fi/display/ITSP/Linear+prediction. [Accessed 2 June 2022].

[16] The MathWorks, Inc., "Formant Estimation with LPC Coefficients," MathWorks, [Online]. Available: https://www.mathworks.com/help/signal/ug/formant-estimation-with-lpc-coefficients.html. [Accessed 1 June 2022].

[17] P. Heggarty, A. Shimelman, G. Abete, C. Anderson, S. Sadowsky, L. Paschen, W. Maguire, L. Jocz, M. J. Aninao, L. Wägerle, D. Dërmaku-Appelganz, A. P. d. C. e. Silva, L. C. Lawyer, J. Michalsky, A. S. A. C. Cabral, M. Walworth, E. Koile, J. Runge and H.-J. Bibiko, "Exploring

Diversity in Phonetics across Language Families," 2019. [Online]. Available: https://soundcomparisons.com/. [Accessed 1 June 2022].

[18] The MathWorks, Inc., "Supported File Formats for Import and Export," [Online]. Available: https://www.mathworks.com/help/matlab/import_export/supported-file-formats-for-import-and-export.html. [Accessed 1 June 2022].

[19] L. Goldstein, "Formant Analysis using LPC," [Online]. Available: https://sail.usc.edu/~lgoldste/Ling582/Week%209/LPC%20Analysis.pdf. [Accessed 2 June 2022].

[20] C. S. Sapp, "WAVE PCM soundfile format," [Online]. Available: http://soundfile.sapp.org/doc/WaveFormat/. [Accessed 2 June 2022].

[21] J. D. Cook, "Don’t invert that matrix," 19 January 2010. [Online]. Available: https://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix/. [Accessed 2 June 2022].

A International Phonetic Alphabet

IPA Chart[*] available under a Creative Commons Attribution-Sharealike 3.0 Unported License. Copyright © 2018 International Phonetic Association.

CONSONANTS (PULMONIC)

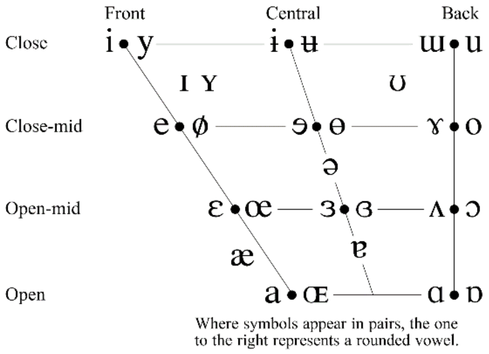
	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k ɡ	q ɢ	▨	ʔ ▨
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ	▨	▨
Trill	ʙ			r				▨	ʀ		▨
Tap or Flap		ⱱ		ɾ		ɽ		▨			▨
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative	▨	▨		ɬ ɮ						▨	▨
Approximant		ʋ		ɹ		ɻ	j	ɰ			▨
Lateral approximant	▨	▨		l		ɭ	ʎ	ʟ		▨	▨

Symbols to the right in a cell are voiced, to the left are voiceless. Shaded areas denote articulations judged impossible.

CONSONANTS (NON-PULMONIC)

Clicks		Voiced implosives		Ejectives	
ʘ	Bilabial	ɓ	Bilabial	ʼ	Examples:
ǀ	Dental	ɗ	Dental/alveolar	pʼ	Bilabial
ǃ	(Post)alveolar	ʄ	Palatal	tʼ	Dental/alveolar
ǂ	Palatoalveolar	ɠ	Velar	kʼ	Velar
ǁ	Alveolar lateral	ʛ	Uvular	sʼ	Alveolar fricative

VOWELS



Where symbols appear in pairs, the one to the right represents a rounded vowel.

OTHER SYMBOLS

ʍ	Voiceless labial-velar fricative	ɕ ʑ	Alveolo-palatal fricatives
w	Voiced labial-velar approximant	ɺ	Voiced alveolar lateral flap
ɥ	Voiced labial-palatal approximant	ɧ	Simultaneous ʃ and x
ʜ	Voiceless epiglottal fricative		
ʢ	Voiced epiglottal fricative		
ʡ	Epiglottal plosive		

Affricates and double articulations can be represented by two symbols joined by a tie bar if necessary. t͜s k͡p

[*] http://www.internationalphoneticassociation.org/content/ipa-chart

SUPRASEGMENTALS

ˈ	Primary stress	ˌfoʊnəˈtɪʃən
ˌ	Secondary stress	
ː	Long	eː
ˑ	Half-long	eˑ
̆	Extra-short	ĕ
|	Minor (foot) group	
‖	Major (intonation) group	
.	Syllable break	ɹi.ækt
‿	Linking (absence of a break)	

TONES AND WORD ACCENTS

LEVEL			CONTOUR		
e̋	˥	Extra high	ě	˩˥	Rising
é	˦	High	ê	˥˩	Falling
ē	˧	Mid	e᷄	˦˥	High rising
è	˨	Low	e᷅	˩˨	Low rising
ȅ	˩	Extra low	e᷈	˧˦˧	Rising-falling
↓	Downstep		↗	Global rise	
↑	Upstep		↘	Global fall	

DIACRITICS

̥	Voiceless	n̥ d̥	̤	Breathy voiced	b̤ a̤	̪	Dental	t̪ d̪
̬	Voiced	s̬ t̬	̰	Creaky voiced	b̰ a̰	̺	Apical	t̺ d̺
ʰ	Aspirated	tʰ dʰ	̼	Linguolabial	t̼ d̼	̻	Laminal	t̻ d̻
̹	More rounded	ɔ̹	ʷ	Labialized	tʷ dʷ	̃	Nasalized	ẽ
̜	Less rounded	ɔ̜	ʲ	Palatalized	tʲ dʲ	ⁿ	Nasal release	dⁿ
̟	Advanced	u̟	ˠ	Velarized	tˠ dˠ	ˡ	Lateral release	dˡ
̠	Retracted	e̠	ˤ	Pharyngealized	tˤ dˤ	̚	No audible release	d̚
̈	Centralized	ë	̴	Velarized or pharyngealized	ɫ			
̽	Mid-centralized	e̽	̝	Raised	e̝	(ɹ̝ = voiced alveolar fricative)		
̩	Syllabic	n̩	̞	Lowered	e̞	(β̞ = voiced bilabial approximant)		
̯	Non-syllabic	e̯	̘	Advanced Tongue Root	e̘			
˞	Rhoticity	ɚ a˞	̙	Retracted Tongue Root	e̙			

Some diacritics may be placed above a symbol with a descender, e.g. ŋ̊

B Test data

B.1 Sound clips

All clips are available at Sound Comparisons [6].

Table 1: List of speech clips used for testing, along with the vowel to be tested.

#	vowel	IPA	language
1	a	[gɹaːn]	Canada: Iona
2	a	[naːm]	Dutch: Std in NL
3	a	[vaːf]	‘Flemish’: Antwerp
4	a	[saːlʦ]	Franc. E.: Altersbach
5	a	[ʦβ̥aː]	Franc. E.: Nuremberg
6	a	[saːm]	Fris. W.: Hylpen
7	a	[gɾaː]	Gascon: Val d’Aran
8	a	[naːn]	Liverpool
9	a	[aːχtə]	‘Low Saxon’: Veenkoloniën
10	a	[ˈβaːɾə]	‘Low Saxon’: Veenkoloniën
11	a	[ˈcçaːnə]	Molise: Guglionesi
12	a	[tʰɹaː]	Molise: Guglionesi
13	a	[ʃaː]	Norman: Les Pieux
14	a	[haː]	Norman: Les Pieux
15	a	[luŋˈgaːtɕ]	Romansh: Surmiran
16	a	[ˈsaːɫɪ]	Sicily E.: Catania
17	a	[gaː]	Suth.: Melness
18	a	[ʦʰaːnə]	Thur. E.: Altenburg
19	a	[saːl]	Trieste (city)
20	a	[vaː]	Welche: Labaroche
21	ə	[ˈtsʰʊŋən]	Austria: Vienna
22	ə	[fʏnvə]	Brandenburg: Berlin
23	ə	[ˈtʃaŋkə]	Canada: Inverness
24	ə	[ʦʰaːnə]	Erzgebirge W.: Aue
25	ə	[ˈwettʰə]	FrPrv: Aosta: Roisan
26	ə	[ˈnɔːvə]	Italy: N. in S.: Rivello
27	ə	[siːvə]	Lechrain: Heinrichshofen
28	ə	[ˈɸo̝lə]	Liecht.: Walser
29	ə	[eːnə]	‘Low Saxon’: Achterhoek
30	ə	[ˈtʃiˑn̪d̪ə]	Lucano: Castelmezzano
31	ə	[salˠən]	Meath: Rathcarran
32	ə	[ˈɔ̝ˑttə]	Naples (city)
33	ə	[dəː]	Norman: Orglandes
34	ə	[sɪvə]	Penn. Ger: Hartville, OH
35	ə	[ˈs̠eːz̠ə]	Piedmont: Pianezza
36	ə	[ˈlɜˑŋgə]	Provençal: Lagnes
37	ə	[ˈtʃɛgə]	Ross: Opinan
38	ə	[ˈnäːmə]	Std. German (Ctl Germany acc.)
39	ə	[ˈtsʊŋˑə]	Switz.: Biel
40	ə	[ˈz̥ɪvən]	Translv.: Schäßburg
41	i	[siː]	Bologna (city)
42	i	[ˈviːtrə]	Czech: Std (Plzeň)
43	i	[θɾiː]	‘Doric’ Scots: Buckie
44	i	[ziːbɛn]	Eastphalian: Biere
45	i	[z̥iːm]	Franc. E.: Berching
46	i	[ˈdiːvi̽]	Latvian: Std (Cēsis)
47	i	[tɾiːs]	Latvian: Std
48	i	[ˈfiːfə]	‘Low Saxon’: Achterhoek

#	vowel	IPA	language
49	i	[tiːnə]	‘Low Saxon’: Achterhoek
50	i	[fiːvə]	Lower Saxony: Barterode
51	i	[nˠiː]	Meath: Rathcarran
52	i	[ˈtʃiːndə]	Molise: Guglionesi
53	i	[ˈdiːtʃə]	Molise: Guglionesi
54	i	[iːn]	Patagonia: Esquel
55	i	[diːɕ]	Romansh: Surmiran
56	i	[tɕiː]	Ross: Laide
57	i	[niː]	Stavanger
58	i	[iːn]	Sth-E: Fochriw
59	i	[siːbə]	Swabia: Dinkelscherben
60	i	[diːç]	Walloon: Clavier
61	ɑ	[nɑːm]	Afrikaans: Kroonstad
62	ɑ	[nɑːmə̩]	Bavarian N.: Hausen
63	ɑ	[ˈsɑːlẹ]	Campania: Montella
64	ɑ	[ɫɑːn]	Canada: Inverness
65	ɑ	[ɑːk]	Canada: Iona
66	ɑ	[gɹɑːn]	C’mara N.: Cornamona
67	ɑ	[dɑː]	Dgl: Annagry
68	ɑ	[dɑː]	Dgl: Glencolumbkille
69	ɑ	[ɑːg]	Dgl: Glencolumbkille
70	ɑ	[dɑː]	Dgl: Tory Island
71	ɑ	[vɑ̝ːɾts]	Latvian: Std
72	ɑ	[sɑː]	Lombardy: Barlassina
73	ɑ	[lɑːn]	Mayo: Achill Island
74	ɑ	[lɑːn]	Mayo: Mullet Peninsula
75	ɑ	[lˠɑːn]	Meath: Rathcarran
76	ɑ	[sɑːɫ]	Port.: Std Lisbon acc.
77	ɑ	[ˈpɑːt̪ɾʊ]	Romanian: N.: Moldovenesc
78	ɑ	[ˈsɑːɾɪ]	Romanian: N.: Moldovenesc
79	ɑ	[sɑːɫ]	Romansh: Sursilvan
80	ɑ	[drɑː]	Translv.: Schäßburg
81	u	[huːnətʰ]	Bavarian Ctl: Thierhaupten
82	u	[suːl]	Czech: Std (Plzeň)
83	u	[suːlˠ]	Czech: Std
84	u	[fuː]	‘Doric’ Scots: Buckie
85	u	[duː]	‘Flemish’: France
86	u	[ˈuːɾa]	Franco-Provençal: Valaisan
87	u	[huːnɛ̈tʰ]	Hessen: Frankfurterisch
88	u	[uːr]	Istro-Romanian: Žejane
89	u	[ˈuːnɜ]	Italy: N. in S.: Rivello
90	u	[ˈuːnö]	Italy: N. in S.: Tito
91	u	[nuː]	Limousin: Hautefort
92	u	[ˈzuːwen]	Lombardy: Gardone
93	u	[ˈʒuːne]	Romanian: N.: Moldovenesc
94	u	[duːs]	Romansh: Surmiran
95	u	[tɕuːn]	Romansh: Tuatschin
96	u	[kɐˈɾuːsʊ]	Salento S.: Ruffano
97	u	[ˈduːz̥ʊ]	Sardinian CS: Gonnesa
98	u	[ˈuːnʊ]	Sardinian NW.: Nuoro
99	u	[nuːf]	Walloon: Clavier
100	u	[huːnət]	Wisc. Pom.: Green Bay

B.2 Vowel estimation

Table 2: Test data comparing the vowel to the tested with the two best matches according to the algorithm.

	ref.	test	
#		1st	2nd
1	a	a	ə
2	a	ɑ	ɑ
3	a	u	ə
4	a	ə	ɑ
5	a	ɑ	a
6	a	u	ɑ
7	a	u	ə
8	a	u	ə
9	a	a	ɑ
10	a	a	ɑ
11	a	a	ɑ
12	a	a	ɑ
13	a	u	ə
14	a	u	ə
15	a	u	ə
16	a	a	ɑ
17	a	a	ɑ
18	a	a	ɑ
19	a	u	ə
20	a	ə	ɑ
21	ə	u	ə
22	ə	ə	ɑ
23	ə	a	ə
24	ə	ə	u
25	ə	u	ə
26	ə	ə	ɑ
27	ə	ə	ɑ
28	ə	ə	ɑ
29	ə	a	ə
30	ə	ə	i
31	ə	u	ə
32	ə	a	ɑ
33	ə	u	ə
34	ə	ə	ɑ
35	ə	ə	i
36	ə	u	ə
37	ə	ə	a
38	ə	u	ə
39	ə	u	ə
40	ə	u	ə
41	i	i	ə
42	i	i	u
43	i	i	ə
44	i	i	ə
45	i	u	i
46	i	i	ə
47	i	i	ə
48	i	i	ə
49	i	i	ə
50	i	i	ə
51	i	u	ə
52	i	i	ə

#	ref.	test	
		1st	2nd
53	i	a	ə
54	i	i	ə
55	i	i	ə
56	i	i	ə
57	i	u	ə
58	i	u	ə
59	i	i	ə
60	i	i	ə
61	ɑ	u	ə
62	ɑ	ɑ	a
63	ɑ	ɑ	a
64	ɑ	a	ɑ
65	ɑ	ɑ	a
66	ɑ	ɑ	ə
67	ɑ	ɑ	ə
68	ɑ	ɑ	a
69	ɑ	ɑ	a
70	ɑ	ɑ	ə
71	ɑ	ɑ	u
72	ɑ	a	ɑ
73	ɑ	u	ə
74	ɑ	ɑ	ə
75	ɑ	ɑ	a
76	ɑ	ɑ	a
77	ɑ	a	ɑ
78	ɑ	a	ɑ
79	ɑ	ɑ	a
80	ɑ	ɑ	a
81	u	u	ə
82	u	u	ə
83	u	u	i
84	u	u	ə
85	u	u	ə
86	u	u	ə
87	u	u	ə
88	u	u	ə
89	u	u	ə
90	u	u	ə
91	u	u	ə
92	u	ə	i
93	u	u	ə
94	u	u	ə
95	u	u	ə
96	u	u	ə
97	u	u	ə
98	u	u	ə
99	u	u	ə
100	u	u	ə

B.3 Vowel detection

Table 3: Test data comparing the manually determined vowel segments (ref.) with the automated ones (test). The numbers signify the first:last frame of a vowel segment.

	ref.			test									
#				w/o smoothing					w/ smoothing				
1	25:60			20:22	23:27	28:42	43:48	51:56	20:48	51:56			
2	37:55			36:56					36:56				
3	39:61			41:43	49:52	56:58			49:52				
4	33:69			76:78									
5	30:67			29:56	57:63				29:63				
6	38:64			29:30	31:35	37:57			29:35	37:57			
7	45:68			42:43	46:65				46:65				
8	39:66			38:39	40:42	43:48	49:50	51:56	38:62	66:69			
9	64:91	115:126		64:65	70:86	87:88	94:95		70:88				
10	50:84	86:109		56:66	68:73	75:78	79:83		56:66	68:73	75:83		
11	26:53	66:78		19:21	24:43	44:46	48:51	52:53	24:46	48:53			
12	32:58			25:26	29:34	35:48	49:52		29:52				
13	23:51			11:12	13:21	22:23	44:46		11:23				
14	18:39			13:14	15:16	26:27	28:32		13:16	26:32			
15	44:57	68:87		66:77	78:79	80:81	82:83	103:115	66:83	103:117			
16	34:55	62:70		63:66	74:75				63:66				
17	42:83			38:73	74:79				38:79				
18	22:39	50:60		19:20	22:23	25:41			25:41				
19	48:75			46:47	49:81				49:81				
20	46:68			45:48	49:58	59:60	61:62	63:64	45:69				
21	25:37	45:53		20:22	28:36				28:36				
22	24:32	56:83		24:28					24:28				
23	24:36	54:76		52:55	56:59	60:74			52:74				
24	22:44	52:64											
25	24:33	59:72		23:26	27:31	59:68			23:31	59:68			
26	36:55	67:75		35:54	55:56				35:56				
27	15:28	40:59		36:52					36:52				
28	32:44	63:76		63:71	72:76				63:76				
29	18:34	42:49		44:48	50:52				44:48				
30	48:61	85:95											
31	33:50	56:66		57:67					57:67				
32	29:47	70:78		29:38	39:42	43:48	71:72	74:75	29:48				
33	20:65			27:30	32:34	40:44			27:30	40:44			
34	32:38	48:62		36:38	39:40	42:43			36:40				
35	36:68	81:92		27:29	30:35	44:45	62:63	79:89	27:35				
36	21:31	48:61		20:30					20:30				
37	29:44	54:71		30:35	36:45	56:67	68:70		30:45	56:70			
38	41:55	65:88		37:38	39:41	44:47	49:51	52:53	37:41	44:47	49:57	67:81	
39	27:41	60:74											
40	35:47	57:68		36:47	59:61	63:65	66:70		36:47	63:70			
41	51:71			58:61	70:71				58:61				
42	31:46	71:85		64:68	69:79				64:79				
43	40:72			37:39									
44	25:46	57:69		28:30	56:61	62:64	65:67	68:69	56:71				
45	36:49												
46	38:54	61:73		37:38	39:40	41:42	53:55	59:63	37:42	59:63			
47	22:43			13:14	20:22	29:31	42:43	64:65					
48	33:61	73:86		71:73	74:75				71:75				
49	22:40	49:63		18:20	36:38	40:41	49:50	64:65					
50	22:39	50:60											
51	29:52			40:41	45:47	48:51			45:51				

#	ref.			test									
				w/o smoothing					w/ smoothing				
52	36:58	84:96		24:31	32:35	37:39	41:42	46:47	24:35	46:51	54:58	85:92	
53	32:31	80:94		33:39	50:51	54:59	61:63	68:74	33:39	54:59	68:78	81:84	90:94
54	16:36			30:32	34:37	44:45			34:37				
55	32:56			58:71	72:77	79:86	87:88		58:77	79:88			
56	34:72			31:32									
57	46:81			49:53	64:67	70:71	73:74		49:53	64:67			
58	14:33			15:16	17:18	20:25	27:30		15:18	20:25	27:30		
59	29:44	58:78		37:38	58:72				58:72				
60	24:50			22:23	32:33	35:36	42:43	60:61					
61	60:88			62:81	82:88				62:88				
62	25:41	52:75		25:30	31:41	54:57	59:64		25:41	54:57	59:64		
63	32:62	70:86		33:65	70:82	84:85			33:65	70:82			
64	27:74			24:56	57:61	62:63	64:65	66:67	24:70	73:77			
65	14:50			15:51	52:53	65:66			15:53				
66	25:53			14:15	16:22	26:56	57:59		14:22	26:59			
67	24:58			23:57					23:57				
68	38:70			39:66	67:69				39:69				
69	25:71			30:31	32:71				30:71				
70	31:78			32:71					32:71				
71	29:50			28:53					28:53				
72	40:81												
73	31:61			33:39	40:60	61:64			33:64				
74	29:55			30:50					30:50				
75	38:70			32:36	37:76				32:76				
76	46:67												
77	32:55	75:87		32:37	39:46	48:56	59:60	61:62	32:37	39:46	48:56	59:62	72:85
78	47:84	89:113		48:85	88:95	97:98			48:85	88:95			
79	54:88			54:95					54:95				
80	46:85			37:39	40:46	47:48	49:56	59:61	37:56	59:74	76:79		
81	16:25	39:54		42:49	50:53	54:55			42:55				
82	39:70			21:22	79:82				79:82				
83	34:54			14:18	19:22	23:26	27:28	29:32	14:32				
84	50:72												
85	56:85												
86	21:53	66:79		23:25									
87	16:34	43:66		43:50	53:55	56:58	60:62	65:66	43:50	53:58			
88	31:60			58:63	65:67	69:71			58:63				
89	25:45	55:73		54:68					54:68				
90	20:38	54:71		54:61	62:65				54:65				
91	36:59			37:44					37:44				
92	47:62	67:80		33:34	38:39	46:48	67:77		67:77				
93	62:81	92:118		40:44	46:47	61:62	63:84	100:102	40:44	61:84	107:114		
94	30:58			63:66	67:68				63:68				
95	32:52			24:30	32:33				24:30				
96	83:91	96:115	128:139	82:91	92:93	96:111	128:139	140:141	82:93	96:111	128:141		
97	21:52	63:79		21:51	52:53	63:70	71:72		21:53	63:72			
98	9:31	38:55		9:10									
99	28:53												
100	16:29	36:42		37:43	45:46	47:48			37:43	45:48			