Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Eduardo Sanz Martín

ARTICULATORY-TO-ACOUSTIC MAPPING USING 2D ULTRASOUND

SUPERVISOR

Dr. Tamás Gábor Csapó

EXTERNAL SUPERVISOR

Dr. Alexandra Markó

BUDAPEST, 2016

Contents

STUDENT DECLARATION

Budapest, 18 December 2016

...……………………………………………….

Eduardo Sanz Martín

Summary

Human speech is the result of the combined work of the larynx, which produces the voice as an air vibration, and the articulator organs such as tongue, lips, velum, etc. that modify this vibration signal to model the speech. The way these articulators act over the speech signal is of high interest for speech processing and artificial speech generation. Thus, the main task of this BSc thesis will be the analysis of the relation between the speech parameters and the articulators' movements which generated that speech.

We focus on the tongue movement as the main articulator of the speech generation process. Our study will be based on 2D ultrasound recordings of the tongue from that we will extract the contour movement while saying English sentences. In parallel, the audio signal corresponding to that ultrasound will be processed to obtain its spectral features in order to find the relation between both contour movements and audio signal.

The final idea is generating a functional module which receives the ultrasound images from a recorded sentence as input and process the interesting information for our study. The information extracted from these ultrasound images must be presented on a simple way and then compared with the one extracted from the audio signal of the sentence, showing how they are related to find some useful correlation between both domains.

The resulting module of the thesis will be tested on different objective and subjective ways. Applied to records made by the student or provided by the thesis tutor, the objective tests will include checking the right performance of the script processing the ultrasound and, as a subjective test, applying some classifier to the contour information and checking the resulting estimated values for some spectral coefficient of the audio.

This work will be accompanied by the corresponding documentation of all the processes and so by the previous studies of the different tongue tracking methods and different spectral estimation methods; besides, some review of papers about previous researches on the topic.

Resumen

El discurso humano es el resultado del trabajo combinado de la laringe, que produce la voz como una vibración del aire, y los órganos articuladores tales como lengua, labios, velo, etc. Los cuales modifican esa vibración modelando así el discurso. La manera en que actúan estos órganos sobre la señal de voz es de gran interés para abordar temas como el procesamiento de voz humana o generación de voz artificial. Por lo tanto, la principal tarea en esta tesis es el estudio de la relación entre las propiedades del habla humana y los movimientos de los órganos articulatorios que lo generan.

Nos centramos concretamente en el movimiento de la lengua como principal articulador en el proceso de generación del habla. El estudio se basa en grabaciones de ultrasonidos bidimensionales las cuales recogen el movimiento del contorno lateral de la lengua al pronunciar frases, principalmente en inglés. En paralelo la señal de audio generada se procesa para obtener sus características espectrales y finalmente se crea una relación entre señal de voz y movimientos de la lengua.

La idea final es crear un módulo funcional que reciba como entrada un bloque de imágenes de ultrasonido grabadas de una frase y que procese la información interesante para nuestro estudio. La información extraída de los ultrasonidos debe ser presentada de forma intuitiva y comparada con la información extraída del audio de la frase, mostrando cómo se relacionan para así encontrar alguna relación útil entre ambos dominios.

El módulo resultante será probado con métodos subjetivos y objetivos. Aplicándose a grabaciones hechas por el estudiante o proporcionadas por el tutor responsable del proyecto, las pruebas objetivas consistirán en comprobar el correcto funcionamiento al procesar los ultrasonidos mientras que como prueba subjetiva, se aplicarán clasificadores matemáticos a la información del contorno, comprobando los valores estimados resultantes de cierto coeficiente espectral del audio.

Todo el trabajo irá acompañado de la documentación del proceso llevado a cabo así como de estudios previos sobre métodos de reconocimiento de movimiento de lengua y sobre métodos de estimación espectral.

1 Introduction

The topic of this thesis is articulatory-to-acoustic mapping which studies how the articulatory organs generate human speech. As a first view of the topic, we develop a brief description of the speech generation process and the elements involved on it.

The organs of speech are classified as supraglottal and subglottal. This separation is delimited above and below the vocal folds, located in the larynx, whose separation is called glottis. The subglottal organs mainly belong to the respiratory system and provide the air flow needed to generate the speech signal while the supraglottal tract is in charge of modifying that air flow and producing the phonemes contained on it. These two stages of the speech production are called **phonation** and **modulation** respectively. Our interest will focus on the modulation phase which is performed by supraglottal articulators.

The basic structure of the supraglottal tract is composed by the pharynx, and both nasal and oral cavities, separated by the velum. The nasal cavity is used to generate so called 'nasal sounds' while the rest of the phonemes are generated by the oral cavity movements as a result of the tongue, teeth and lips work combined [1].

Is then the oral cavity on what we will focus, and more concretely on tongue movements since it is the main articulator organ into the supraglottal tract.

1.1 Task definition

The labor to be made on this thesis consists on studying the relation between the articulators and the speech signal. For this, both audio and articulator movement signals will be needed in order to find how they are related.

There are different parameters which need to be chosen before addressing the study, i.e. what articulator/s do we study, how to track its/their movements, what features of audio signal we want to analyze and how to obtain them, how to process audio and articulator's information and what models or schemes do we apply to find the relation between them.

1.2 Objectives

Within this study we look for a list of objectives, all related among them:

- Make a review of different articulatory-to-acoustic mapping methods used in previous studies about the topic.
- Compare different options to face up the topic in order to choose the one which fits best in our case.
- Obtain some relation between speech parameters and the corresponding articulator movements.
- Apply models for estimating those speech signal parameters from the articulator information associated to them.
- Check the result with different recorded sentences, including a set of them recorded by the own student.

1.3 Motivations

All this area of research leads to the target of artificial speech generation (also known as speech synthesis) based on non-acoustic information. This idea has interest for every communicative situation on which it is not possible or recommendable to transmit audio signal. Some of these practical applications are listed on [2]:

- Medical field, as aid for patients whose voice has been decreased or lost (e.g. laryngectomized patients or deaf and dumb people).
- 'Silent phone' device for confidential or furtive communications. It also might be useful for communications in silent places like theatres or concerts.
- Non-voiced speech generation in noisy environments where there is no 'audio contact' but visual contact between receptor and transmitter.

In general, what we want is to allow verbal communication without need to send sound signal between two points, using instead articulator information.

1.4 Structure of the thesis

The paper of this thesis is structured as follows: a general introduction of the topic included above, a deep explanation of the task to be done in this study fixing on it the main aspects to consider, a preliminaries analysis on which we do a review of the previous papers of different researches about the topic of articulator to acoustic mapping, a central section about the description of the work done in this study step by step and finally, the last section of results and critical assessment on which we expose the obtained outcome.

2 Analysis of the task

After introducing the main topic of the thesis on which we want to work, on this section we will define how we will approach the topic more concretely.

2.1 Articulators

First of all, this study is focused on phonetic modulation of the speech. This function is performed on supraglottal tract and into it, it is the oral cavity where most of the phonemes are generated. The oral cavity contains tongue, teeth, velum and palate as main articulator organs [1].

Concretely we will focus on **tongue movements** since it is the main modulator of speech. The superior contour of the tongue will be tracked while the subject of the recording pronounces different sentences. The result will be a profile of the tongue from which we want to extract some information about the speech signal generated by its movements.

2.2 Data acquisition

On section *Preliminaries analysis* we will introduce a review of previous researches and studies about the topic. Besides we will see different options to record articulator information and pros and cons of each one.

For our current task we choose **2D ultrasound based recognition**. Ultrasound system is based on sounds with frequencies above the audible range (audible range: 20 Hz - 20 kHz). Sound is mechanical energy that needs a medium to propagate and thus the transducer acts as a microphone in order to record the acoustic echoes generated by the tissue along the path of the emitted pulse. These echoes carry information about the internal tissues (the inside of the mouth in this case) along the path it traveled [3].

In Figure 2.1 we can see the image resulting of an ultrasound recording. What is shown on it is one of the frames of the sentence. One single sentence will be composed by several frames like this one in a row. The number of frames included into each sentence will depend on how long it is.

As we can see, the picture shows a shadowed image on grayscale where the tongue contour is represented as a white line which moves up, down and slightly sideways. It is a lateral contour of the tongue so the tongue tip is represented at the right extreme of the contour while the tongue root belongs to the left of the ultrasound.

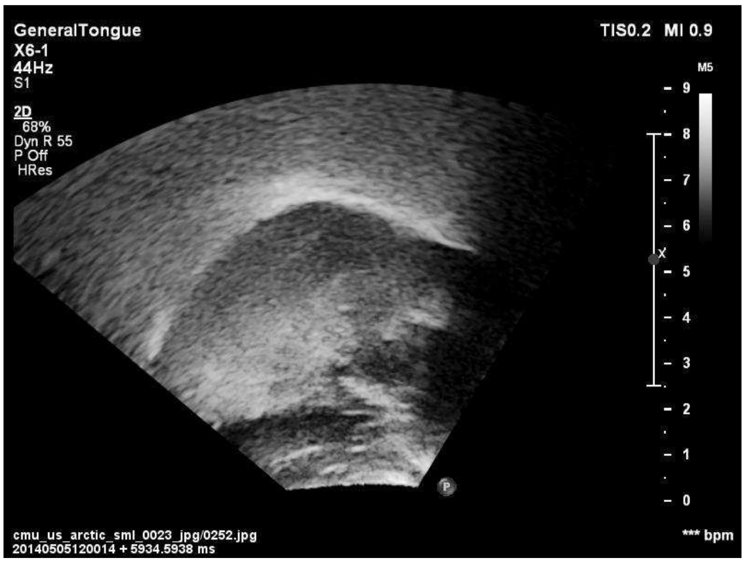


Figure 2.1. Example of a 2D ultrasound tongue image from a sentence recorded at the Speech Production Lab of Indiana University (IU)

2.3 Data processing

Once we have these 2D ultrasound images, some processing is needed to transform the information into numeric values, which can be later analyzed by computer software.

This processing will be divided into two steps:

- Using ***EdgeTrak*** software [4] we take ultrasound images associated to one full sentence and we obtain from it the coordinates of the contour's points on a text file.
- After this, various ***Python*** code scripts will be used to extract those coordinates. From them we will generate simple images of the contour combined with temporal and spectral representations of the audio signal. Besides this, contours coordinates will be written on other text file with the format needed for the next phase of the analysis.

These two processes will be explained more deeply on the main section of the document: Design process.

2.4 Audio data study

Audio data acquisition is made at the moment of the ultrasound record in order to obtain it synchronized with the ultrasound image. Acting this way we can easily obtain the audio fragment corresponding to each one of the ultrasound frames.

Once we have the audio signal, it is needed to think about what feature we will get out of it. The audio features observable in time domain depend on the subglottal tract. For example, the amplitude of the signal, which measures how loud the speech is, is related with how much air is provided by the lungs while speaking. For its part, fundamental frequency determines the tone of the audio signal and it represent the fundamental period of vocal folds vibration. It is up for each individual's characteristics like gender, complexion, etc. Male voices are usually lower while female's and children's tend to be higher [5].

The articulators' influence is shown then in frequency domain. All previous researches about the topic work with the spectrum of the signal. In our case we choose *Mel-Generalized Cepstrum* coefficients (MGC) as representation of the spectrum [6].

3 Preliminaries analysis

On this section we will introduce a review of previous studies about articulatory-to-acoustic mapping topic. First, we will develop a short explanation of the different acquisition techniques used to extract both articulator and audio information on each study. After that we go to the evolution of articulatory-to-acoustic mapping through different experiments, from Gaussian Mixture Static Models (GMM) to Hidden-Markov Models (HMM) considering dynamic properties.

3.1 Data acquisition techniques

We take an extract from [7] in which all the different options used on silent speech recognition studies are listed along with the corresponding authors:

"Several studies addressed the problem of silent speech recognition, i.e. the identification of a sequence of words from silent articulation: [8] for sEMG, [9] for NAM, [10] for PEMA and [11] for ultrasound."

Each one of these techniques involves some principles and characteristics. Here we introduce them more deeply.

3.1.1 Surface Electromyography (sEMG)

Electromyography consists of obtaining impulse energy generated on muscles while moving. In this case the electromyography works with various electrodes placed on the subject's face which track the activity of different muscles while speaking. On Figure 3.1 we can see an example of a person with these electrodes placed on.

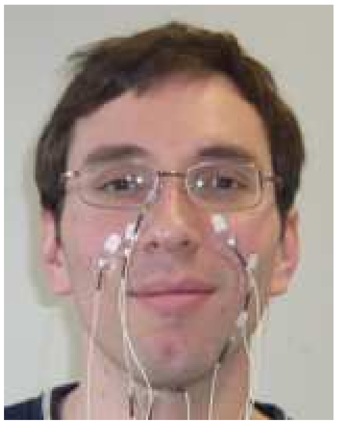


Figure 3.1. Overview of electrode positioning and captured facial muscles. Image extracted from [8]

Various disadvantages are mentioned in [8]: First, the impact of speaker dependencies, such as speaking style, speaking rate, and pronunciation. What is more, the EMG signal is affected by changes in electrode positioning and environmental conditions (temperature and humidity). These factors clearly favor the development of speaker-dependent and often session-dependent systems.

3.1.2 Non-Audible-Murmur recognition (NAM)

The idea here is collecting the sound vibrations using a stethoscope placed over the skin of the recording subject [9]. As its own name suggests, this option is applied for murmur or silent speech recognition which implies low sound vibrations.

3.1.3 Permanent Electromagnetic Articulography (PEMA)

In this case the movement of the articulators will be recorded by several magnets placed on key points such as tongue, lips or cheeks. The changes which these magnets produce in a magnetic field due to speaking movements carry the articulator information. Apart from the magnets some sensors are needed to capture those magnetic variations. We illustrate in Figure 3.2 an example of this kind of recording.

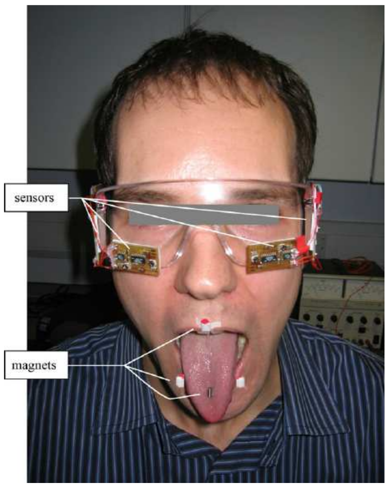


Figure 3.2. Example of PEMA record system placed in a subject. Image extracted from [10]

As we can see on Figure 3.2, this alternative is highly invasive and might result annoying for the subject. It could deal to variations on the way of articulating and thus to alter the results.

3.1.4 Ultrasound transducer

In ultrasound recordings a transducer is placed beyond the chin of the subject while talking. This transducer sends ultrasound signals, typically into 3-5MHz range, and collects the echo produced by them when contact with internal soft tissues (in this case the tongue). This way we obtain one grayscale image with the profile of the tongue represented on it as a white line moving as we saw on Figure 2.1, into section *Data acquisition*.

In some studies which apply this technique it is usual to record video from the front of the mouth simultaneous to ultrasound recording in order to obtain lips movement information. In Figure 3.3 we see an example of helmet for ultrasound recordings used to prevent the transducer from moving while contacting the chin. In the image are also included the camera and the microphone to record video and audio signals.

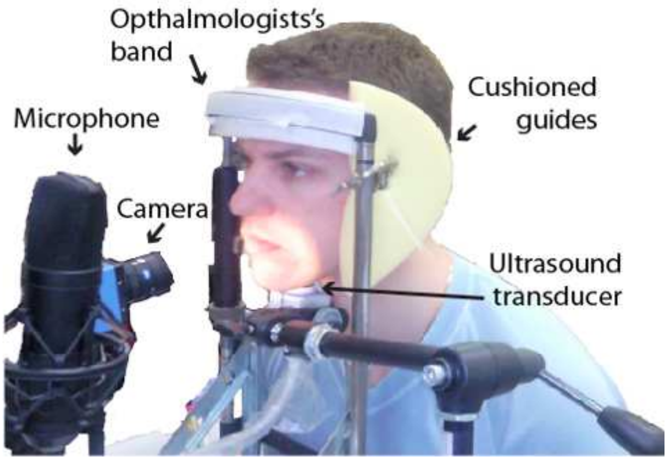


Figure 3.3. Ultrasound recording helmet setup. Image extracted from [13]

Ultrasound tracking is the option we choose for this thesis while it results in a set of ultrasound frames from which we can easily extract the tongue contour information with a tacking software as we will explain later in section *Tongue contour tracking: EdgeTrak*.

3.2 Articulatory-to-acoustic mapping techniques

Here we make a review of the development of different models applied in previous researches for mapping the articulator information into audio features. We will analyze four studies in order of publication so to show how more complex models are applied and each one tries to improve previous results.

The introduction of [13] presents their previous mapping techniques as starting point. These ones were based on two different stages: first, given the test sequence of visual features, a phonetic target sequence was predicted. A visual-acoustic relation dictionary is applied to this phonetic target sequence with a selection algorithm. All the obtained audio segments are concatenated to generate the final speech waveform.

Two disadvantages are mentioned for this process:

- The quality of the synthesis depends highly on the phonetic decoding since one single mistake on it is critical for the resulting audio speech.
- Audio and visual sequences are processed separately and thus, dependency between them is not taken into account.

After that, they introduce two static mapping techniques based on joint modeling of articulator and acoustic data: Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM).

3.2.1 Gaussian Mixture Models (GMM)

Defined $\mathbf{x}_t$ and $\mathbf{y}_t$ as source and target vectors observed in a specific moment *t* (in this case source is referred to visual stream and target is referred to audio stream), the implementation is based on the modeling of the joint probability density of them. Thus, the estimation of $\mathbf{y}_t$ at time *t* from the given $\mathbf{x}_t$ is calculated as a linear weighted combination of *M* components.

$$\hat{\mathbf{y}}_t = F(\mathbf{x}_t) = \sum_{m=1}^{M} (W_m \mathbf{x}_t + b_m) \cdot P(c_m \mid \mathbf{x}_t)$$

Equation 3.1. Estimated target vector in GMM implementation. Formula extracted from [13]

W_m and b_m are the transformation matrix and the bias vector of the *m*th component of the model while $P(c_m|\mathbf{x}_t)$ is the probability of the source vector to belong to that component.

3.2.2 HMM-based mapping

In this case the sequence of target vectors **y** is obtained from the sequence of source vectors **x** maximizing the probability of **y** conditioned to **x**:

$$\hat{\mathbf{y}} = \arg\max_{y} \{ p(\mathbf{y} \mid \mathbf{x}) \} \qquad p(\mathbf{y} \mid \mathbf{x}) = p(\mathbf{y} \mid \lambda, q) P(\lambda, q \mid \mathbf{x})$$

Equation 3.2. HMM based mapped audio vector. Formula extracted from [13]

λ expresses the set of parameters of the HMM model while q is the state sequence of the model. The equation on the right illustrates the two stages involved on the process: first finding the optimal state sequence for a given source sequence and then inferring the target sequence from it.

That state sequence consists of the set of phonemes which best fits with the source vector. Since this kind of mapping is achieved at phonetic level instead of frame level, linguistic constraints can be included like limited vocabulary or language models. At the conclusion of [13] this constraints are said to improve the quality of the mapping.

Apart from these two mapping techniques, this paper explains how the voiced/unvoiced parameter and the pitch value for the voiced segments are predicted.

A voiced segment is a sound produced as an uniform vibration of the vocal folds by normal exhalation of air, while unvoiced segments correspond to sounds generated with constriction in the way out of the air, for instance 's' or 'p' sounds [5].

Artificial Neural Networks (ANN) are used for this classification. An ANN is a networked module which simulates the human neural system in order to solve a task. "Inspired by the sophisticated functionality of human brains where hundreds of billions of interconnected neurons process information in parallel, (...) an artificial neural network consists of an input layer of neurons (or nodes, units), one or two hidden layers of neurons and a final layer of output neurons" [14].

The research presented in [13] is followed by [2], in which they try to improve the previous mapping technique. On that previous technique the sequence of target vectors **y** (spectrum of the speech vector) was predicted by two stages, first estimating the phonetic state sequence from the source vectors and then obtaining the target sequence from that estimated phonemes.

This process is called **Baseline mapping technique** and it implied that spectral trajectories, **y**, were estimated directly from the phonetic decoded sequence and it did not take into account the observations of the articulator. Then the quality of mapping depended completely on the phonetic decoding.

The new approach is to predict the spectral trajectories from both phonetic and articulator sequences instead of only phonetic information. This improved technique is defined as **Continuous HMM-based mapping** and it allows considering local correlations between articulator and spectral features modeling their joint probability density function by a single Gaussian distribution.

$$\hat{\mathbf{y}} = \underset{y}{\arg\max} \left\{ p(\mathbf{y} \mid \mathbf{x}, \hat{\lambda}, \hat{q}) \right\}$$

Equation 3.3. Continuous HMM-based mapping expression. Formula extracted from [2]

As we can see, estimating HMM state is still the first step, but now the predicted spectrum depends directly on the articulator sequence besides the decoded phonetic sequence. This method still allows to include linguistic restrictions for phonetics which leads to better mapping quality.

The next reviewed paper [7] keeps going on the research about silent speech interface and for this it offers two variations of the previous mapping options. Both GMM and HMM based mappings are tested considering dynamic features of source and target vectors. These dynamic features are considered simply by extending source and target vector with their first derivatives. Derivative vector at time *t* is calculated as $\Delta\mathbf{y}_t = 0.5\mathbf{y}_{t+1} - 0.5\mathbf{y}_{t-1}$ and then added as an additional column of the data.

The result of applying this modification into GMM based mapping is what is called **GMM-based mapping considering dynamic features** (GMM + dyn) and it was first defined in [15] for voice conversion (VC) and later in [16] applied to articulatory-to-acoustic mapping.

$$\hat{\mathbf{y}}_t = \sum_{m=1}^{M} (W_m\mathbf{x}_t + b_m) \cdot P(c_m \mid \mathbf{x}_t) \qquad \hat{\mathbf{y}}_{seq} = (W^T D^{-1} W)^{-1} W^T D^{-1} E$$

Equation 3.4. GMM based conventional mapping and considering dynamic features. Formulas extracted from [13] and [7]

As we can see on these expressions, conventional GMM based mapping estimates each individual $\mathbf{y}_t$ vector depending only on the corresponding $\mathbf{x}_t$ vector. On the other hand, when dynamic features are considered, the resulting estimation refers to the whole sequence of target vectors.

The same idea is applied to HMM based mapping technique and again the result is a whole target sequence obtained as function of the whole sequence of source vectors and the whole sequence of HMM model states.

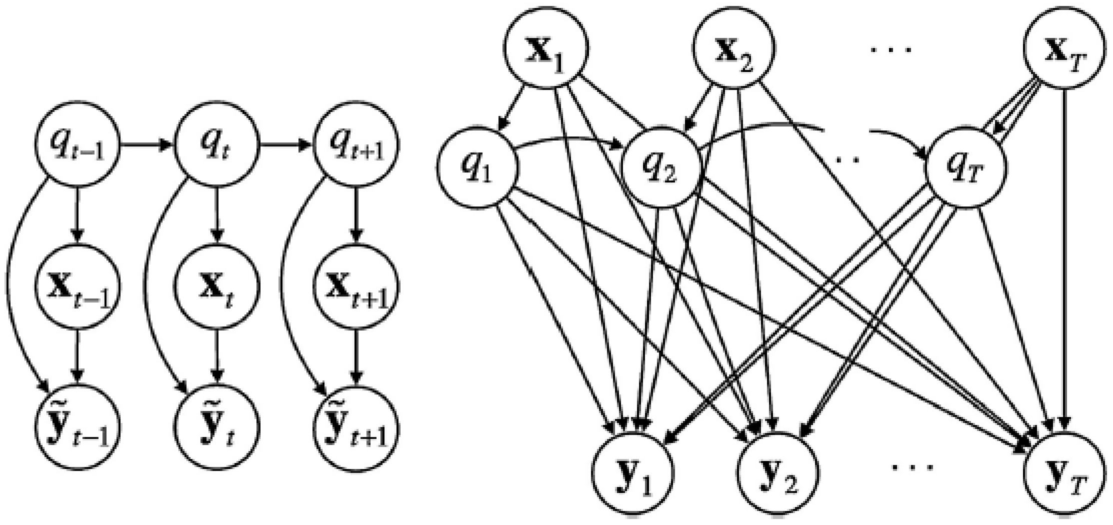


Figure 3.4. Graphic of dependencies in HMM+dyn mapping. Image extracted from [7]

On Figure 3.4 $\mathbf{q}_t$ is the phonetic sequence obtained from the corresponding articulator vector $\mathbf{x}_t$. On the figure of the left the resulting target vector depends only on the corresponding source vector ($\mathbf{x}_t$) and the corresponding phonetic sequence ($\mathbf{q}_t$). On the right figure we see how each $\mathbf{x}_t$ and $\mathbf{q}_t$ vector affects each $\mathbf{y}_t$ in this new approach.

As a result of the study, the paper indicates that both approaches outperformed their previous versions. However, the word accuracy on transcription tests were 75% and 60% using HMM+dyn model and GMM+dyn model respectively. Although HMM+dyn results are better in terms of spectral distortion, GMM+dyn is said to reach better responses from listeners in perceptual tests.

"Nevertheless (...) transcription tests showed that none of these techniques was yet able to synthesize perfectly intelligible speech, in a systematic manner." [7]

After this pessimistic perspective included in the conclusion of [7], a parallel study gives other view about the feasibility of the speech synthesizer from articulator data only one year later. A newer paper from that study is [17] and on it the long-term goal is said to be also developing a SSI (Silent Speech Interface).

For this process PEMA was used to extract articulator information. Some drawbacks of this technique are mentioned in the paper as "some limitations for detecting certain aspects of speech articulation (e.g. the manner of articulation, voicing and the phones articulated at the back of the mouth)".

Two different databases of speech were used: TIDigits (names of numbers from 0 to 9 and 'oh' sound) and CV database (48 syllables composed by 12 consonants and 4 vowels). Different results of listening tests were reached from both databases, the better ones performed using TIDigits. The reasons are said to be the context influence on speech recognition using TIDigits (is easier to recognize a sequence of numbers) against isolated non-sense syllables using CV and the higher complexity of CV design which includes more variety of sound combinations.

Two conversion algorithms were also considered. They are MMSE (Minimum Mean Square Error estimator) and MLE (Maximum Likelihood Estimator). MLE needs more computational load and it takes into account the temporal dynamics of the speech parameters. The conclusion about this choice is that the differences between the results of both techniques do not worth the extra complexity of MLE. Then, MMSE estimation is said to be the simplest and best option.

As conclusion of [17], the text gives a good perspective of the results: "We have successfully demonstrated that the proposed technique is able to generate speech of sufficient intelligibility and quality for some vocabularies. This is a big step in our long-term goal of developing a discrete and reliable SSI that will ultimately allow laryngectomees to recover their voice."

4 Design process

Once all these different visions have been presented, this section will explain the research made to find correlation between articulator and audio data, i.e. how we deal with articulatory-to-acoustic mapping topic.

As explained in section *Analysis of the task*, our approach focuses on tongue function and how its movements modulate the corresponding audio signal. We know that tongue belongs to supraglottal tract and it only affects voice modulation process. Since tongue does not interfere on phonation, we will put our emphasis on spectral features of the audio signal.

Following we define step by step all the different stages done in this work: recoding articulator data, extracting the information we are interested in, processing that information to work with it, analyze the resulting values and finally extract conclusions from the results.

4.1 Data acquisition: Microphone and Ultrasound transducer

From the list of data acquisition options listed above (Section *Data acquisition* techniques) we use ultrasound tracking for this research.

The result of recording one sentence by ultrasound technique is a set of images similar to the one showed in Figure 2.1. The tongue contour is represented as a white curved line in an also curved grayscale image. The shape of the image is curved because the transducer sends ultrasound rays upwards from the chin in a radial way. When it extracts the echo produced by these rays touching internal tissues, the final shape is curved like the wave front of the ultrasounds. The interesting information to be extracted from these images is the tongue contour movement.

4.1.1 Available synchronized speech and ultrasound data

BME-TMIT provided available speech and ultrasound data that was recorded in 2014 at the Speech Production Laboratory of Indiana University (IU_ULTRASOUND).

From the IU_ULTRASOUND dataset (info available: http://smartlab.tmit.bme.hu/csapo/IU_ULTRASOUND/) , we were using speaker SML (adult male, native speaker of American English). The full speech material was the first 27 sentences of the CMU-ARCTIC database. In order to avoid buffer overflow of the ultrasound device, sentences were recorded in groups of three. Before and after each group, the nonsense word /tata/ was uttered for alignment purposes.

The tongue movement was recorded using a Philips EpiQ-7G ultrasound system with an xMatrix 6-1 MHz transducer. The recordings were made in a soundproof booth in the Speech Production Laboratory at Indiana University. Simultaneous speech signals were recorded with a Shure professional microphone and a pre-amplifier. The ultrasound system was placed outside the soundproof booth in order to reduce background noise coming from the device, and only the ultrasound transducer with a long cable was in the booth. The speech signal was digitized at 48 kHz with a National Instruments card. For the ultrasound recordings, a helmet (Ultrasound Stabilisation Headset, Articulate Instruments Ltd) was used which fixed the ultrasound transducer relative to the skull of the subject. The subject was sitting on a chair while uttering the speech material.

The ultrasound image frame rates were between 42-48 fps for all of the recordings. The ultrasound data were saved in DICOM format with 800x600 resolution and converted to JPG image sequences using Image-J (http://imagej.nih.gov/ij).

In order to align ultrasound and audio data with each other, the subject uttered the /tata/ nonsense word before and after each sentence group. In the speech recording the locations of the stop burst for the four /t/ consonants were measured. In the ultrasound recording, the same location was measured by finding the place where the tongue tip started moving from the neutral position. The ultrasound images were synchronized to match the audio based on this.

In the later analyses, the sentences identified with numbers 005, 010, 017, 019, 022 were used from speaker SML. In **Error! Reference source not found.** we show the content of these recorded sentences.

#	**SENTENCE**
005	*Will we ever forget it*
010	*I'm playing a single hand in what looks like a losing game.*
017	*From that moment his friendship for Belize turns to hatred and jealousy.*
019	*I followed the line of the proposed railroad, looking for chances.*
022	*Hardly were our plans made public before we were met by powerful opposition.*

Table 4.1. Sentences used on the study. They were extracted from the CMU-ARCTIC database and recorded at the Speech Production Lab of Indiana University (IU)

4.1.2 Recording session of ultrasound data

As explained in section *Available synchronized speech and ultrasound data*, for this project one set of 27 ultrasound recorded sentences with each synchronized audio signal were provided by BME TMIT at the beginning of it. The development of the code for processing articulator data and first tests were made using this database. After that, some sentences were also recorded by the student, Eduardo Sanz Martín, in both Spanish (mother language) and English languages. The sentences belong to 'The Grandfather passage' and 'Rainbow passage', two typical texts frequently used in speech studies. Apart from this, some non-sense words which combined different vocal and consonant sounds were included as well: "apapa", "atata", "epepe", etc. (all these with Spanish pronunciation)

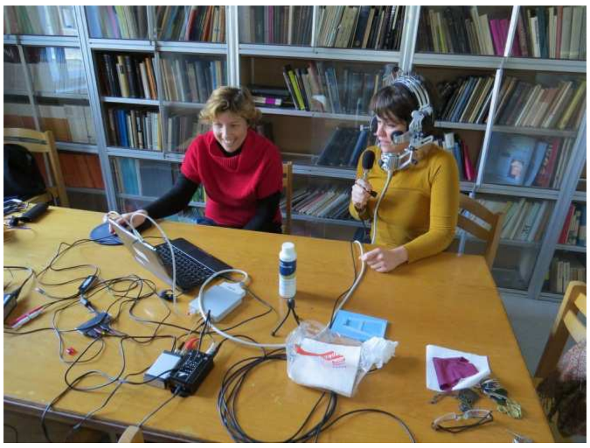


Figure 4.1. Example of ultrasound recording session using helmet to fix ultrasound transducer to the chin. Image provided by the MTA-ELTE Lendület Lingual Articulation Research Group

Some specifications of the recording process are detailed in [18]: The tongue movement was recorded using a SonoSpeech ultrasound system (Articulate Instruments Ltd.) with a 2-4 MHz / 64 element 20mm radius convex ultrasound transducer at 96 fps. During the recordings, the transducer was fixed using an ultrasound stabilization headset (Articulate Instruments Ltd.). The speech was recorded with an Audio-Technica ATR 3350 omnidirectional condenser microphone at a distance of approximately 30cm from the lips. The ultrasound and the audio signals were digitized using an M-Audio – MTRACK PLUS audio interface at 22050 Hz sampling frequency. The ultrasound and speech recordings were synchronized applying the frame synchronization output of the equipment with the Articulate Assistant Advanced software (Articulate Instruments Ltd.). After this, the ultrasound frames were extracted as raw scan line data and converted to JPG images.

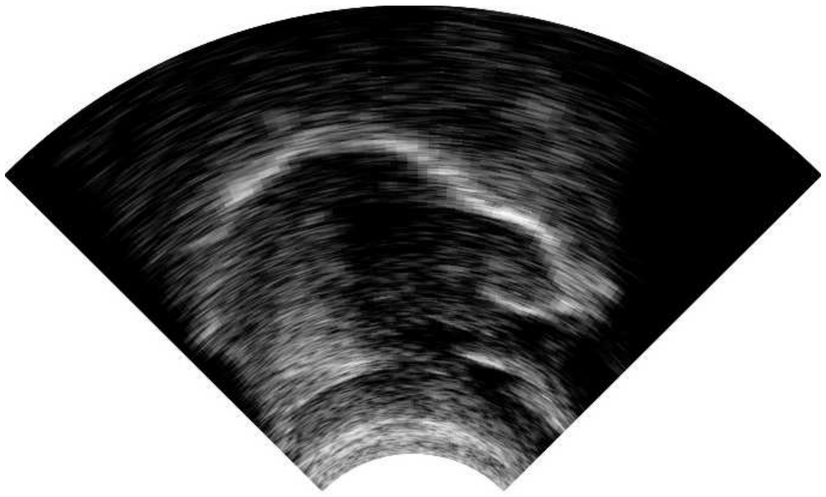


Figure 4.2. Example of the result of the recorded sentences by the student for this same research.

Comparing Figure 2.1 and Figure 4.2 the difference of resolution is clear. The images obtained by our recording system were more fuzzy and thus, harder to process and to analyze.

4.2 Tongue contour tracking: *EdgeTrak*

Our ultrasound produces a set of images in .jpg format, one for each frame of the sentence so the number of images depends on the frames-per-second rate of the ultrasound recording. Nevertheless, this representation is not useful for our interest and some preprocessing is needed to extract what we want. We will use for this purpose the tool called ***EdgeTrak***.

EdgeTrak is a simple software used to track sequences of ultrasound images. It receives a set of images, which are supposed to belong to a continuous sequence, and outputs one file on .ts format which contains the position on each frame of all points of the contour.

When the images are loaded, the user has to select the area of interest for the tool to optimize it. Then the gradients of each image texture are calculated by the tool to identify where the most abrupt changes into the color are and place there the contour points when they move from one frame to other. After this the user has to draw manually the contour on the first image. This contour is composed by as many dots as the user wants and once it is drawn, *EdgeTrak* can start to track the following next images of the sequence automatically from the first one. The software draws each frame's contour from the previous' one [19].

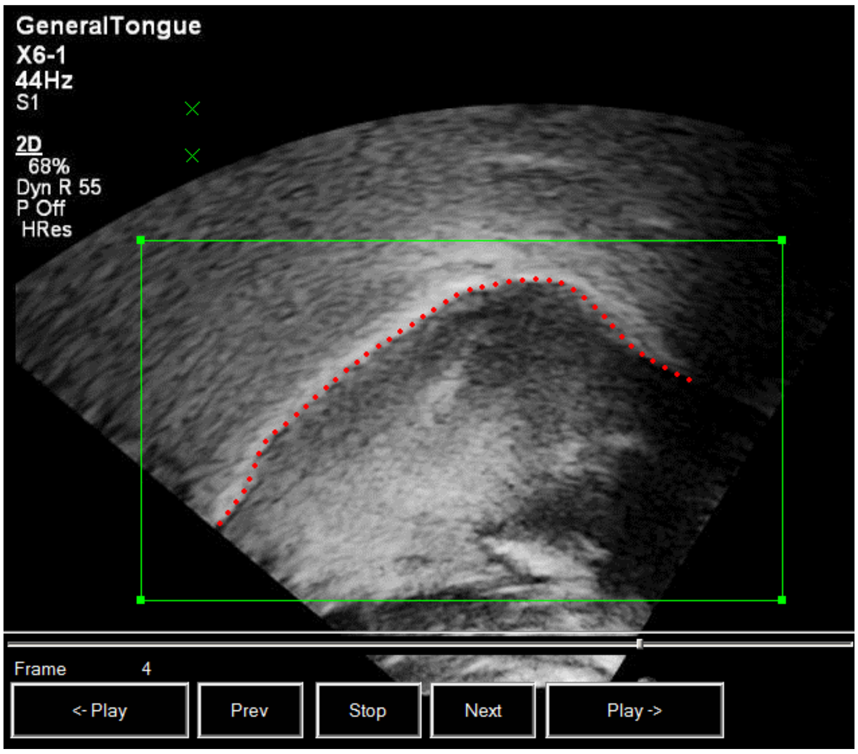
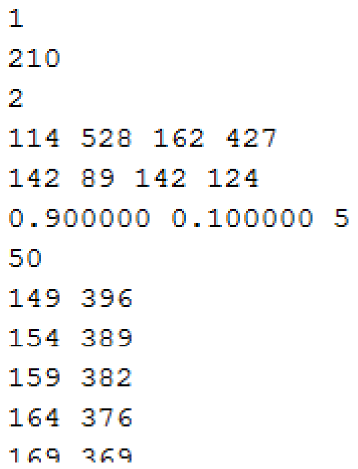


Figure 4.3. *EdgeTrak* doted-line tracking sample.

Although *EdgeTrak* has an automatic tracking function, some losses during the contour tracking are usual, mainly when the image seems to lose the clarity of the white line. This happens either when the tongue goes too high and the ultrasound signal does not reach it so easily or when the transducer is moved from its optimal position while speaking. Because of this, the user must check the result after every sequence tracked, looking for leaks and errors on the contour and correcting them manually. After correcting one error all the following frames from it have to be tracked again because they were obtained from the wrong one.

4.3 Data processing: *Python 3*

The output of *EdgeTrak* for each sentence is one file which contains the vertical and horizontal position of each point from each contour organized by frames along with more information about the tracking.



```
1
210
2
114 528 162 427
142 89 142 124
0.900000 0.100000 5
50
149 396
154 389
159 382
164 376
169 369
```

Figure 4.4. Content of the output file from *EdgeTrak* software

As shown in Figure 4.4, the .ts file contains some metadata about the tracking procedure such as number of frames, maximums and minimums or number of points of each frame's contour. After that, a vertical list of pairs of data corresponds with the two coordinates of each point into the contour. This coordinate list is divided by frames, for example, it is divided in groups of 50 pairs of coordinates in case that the user drew contours of that number points.

Once we have this file available, it is time to process it to extract those coordinates and then work with them. We choose ***Python 3*** as programming language for this task which includes different phases explained below in more detail. Here we have the main script to process the .ts file:

```python
def dataProcessing(*filename):
    if len(filename)==0:
        filename = input("Type the name or the ts file to read: ")
    else:
        filename=filename[0]
    #------------------------READ CONTOUR-----------------------#
    matrixCoord = readCoord (filename)
    (numFrames, numPoints) = np.shape(matrixCoord)
    numFrames = int(numFrames/2)
    #-----------------OBTAIN MGC COEFFICIENTS-------------------#
    MGC = np.zeros(numFrames)
    pitch = np.zeros(numFrames)
    (MGC, pitch, duration) = getMGC(filename, numFrames)
    #----------------GENERATE IMAGES AND VIDEO------------------#
    contourImagesVideo(filename, matrixCoord, numFrames, duration)
    audioImagesVideo(filename, numFrames, matrixCoord)
    spectrumImagesVideo(filename, numFrames, matrixCoord, MGC)
    #----------------INTERPOLATE Y-AXIS VALUES------------------#
    matrixData = interpolate(matrixCoord)
    #-------------------TEMPORAL DERIVATIVE---------------------#
    matrixData = spatialDerivative(matrixData)
    #---------GENERATE ARFF FILE FOR WEKA PROCESSING-----------#
    createArff(matrixData, MGC)
    #extendArff(matrixData, MGC)
    return
```

The script `dataProcessing` receives only the name of the .ts file we want to use and operates different tasks over it. The user can choose what of these tasks are applied by commenting or not the corresponding line of code. The name of the file can be provided passing it as a parameter, but if it is not specified the script asks for it through the standard input.

4.3.1 Read contour

The first labor to do is obtaining all the coordinates from the file. This is done by `readCoord` script, which receives the name of the .ts file, opens it, takes the meta data needed (number of frames and number of points per contour) and reads all the coordinates organizing them into a variable type list called `matrixCoord`. This list is composed by vectors: 0th and 1st are x and y coordinates of the 1st frame's contour, 2nd and 3rd belong to the 2nd frame and so on.

`MatrixCoord` is the only output of the script and after extracting it, the main script calculates number of frames and number of points per contour from the length of the list (which is `numFrames`) and from the length of one vector of the list (which is `numPoints`).

4.3.2 Obtain MGC coefficients

This script is the one which works with audio data in order to extract spectral information from it. We choose MGC coefficients as representation of the spectral features and they are calculated by the script `getMGC`.

This script receives the name of the file (name of the audio file and .ts file have to be the same for the correct function of the code) and the number of frames of the sentence. With that data it opens the audio signal and calculates its duration, which will be returned by the function. The number of frames is used to obtain the coefficients in order to extract one set of MGC coefficients for each one of the frames of the sentence.

Finally, the output is composed by the sets of coefficients calculated into a list of vectors called `mgc_coeff`, one list called `pitch` containing the fundamental frequency of each frame and the duration of the audio useful for following synchronism tasks.

4.3.3 Generate images and video

As an extra function added to the script, we decided to include this three options which produce three simple visual representations of the extracted contours. The preferred option can be chosen again by uncomment only one of the three scripts on this section. These three scripts are `contourImagesVideo`, `audioImagesVideo` and `spectrumImagesVideo`.

`ContourImagesVideo` produces a sequence of images of the sentence's contours (as many images as frames the sentence has got). The duration of the audio is needed here as an input to create a video which has the same duration as the audio, resulting in a real-time representation of the tongue movement.

`AudioImagesVideo` generates a set of images composed by two graphs. On the first one the contour of each frame is represented the same way as in the previous script while on the second one we plot the segment of the audio signal corresponding to each contour. We can see then the audio signal produced by every single position of the tongue in the sentence.

This script offers an interesting tool to check any possible direct relation between tongue and audio signals, but we do not expect the relation to be easy enough to notice it with the naked eye. In this case the rate of frames per second is lower than real time in order to make the video clearer to watch in detail.

Finally, `spectrumImagesVideo` follows the same idea as the previous script. In this case the second graph contains the spectral representation that we are working with; this is the MGC coefficients. Looking again for making the resulting video as easy to watch as possible, we consider the option of excluding the first MGC coefficient (which contains the amplitude of the audio signal on each frame). We do this because this value is always one order of magnitude above the rest of the coefficients and if we represent them all together the scale of the y axis does not allow us to difference the smaller values.

As we commented about `audioImagesVideo`, the relation tongue-MGC coefficients is not supposed to be clearly perceptible, but this representation could be useful in following studies or in order to check some possible result found by deeper analysis.

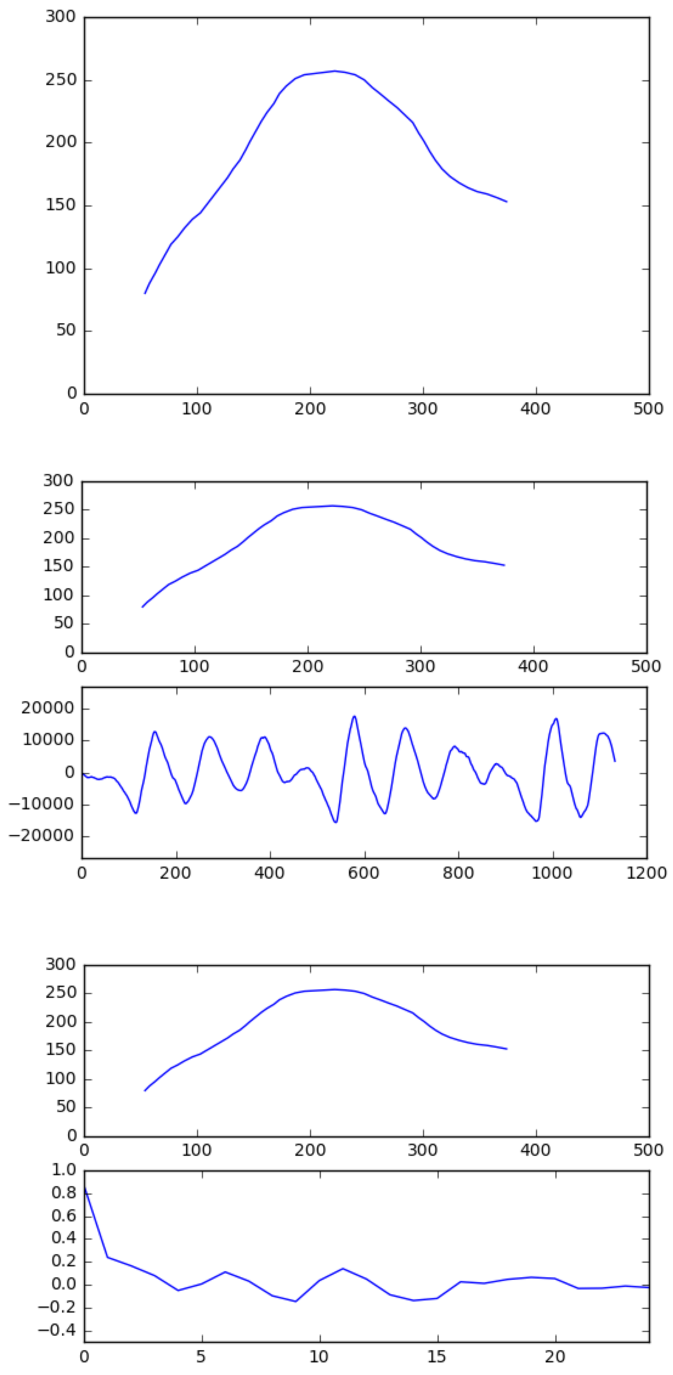


Figure 4.5. Sample of one frame representation result of the scripts `contourImagesVideo`, `audioImagesVideo` and `spectrumImagesVideo` respectively. Each one of these scripts produces also one video of the sequence of frames included in one sentence.

As this representation of the data is very simple and graphic we do not expect to find important relations only by watching these images and videos. Thus, more specialized analysis of the data will be needed, so we will introduce them on following sections.

4.3.4 Interpolate Y-axis values

The coordinates of the points obtained from *EdgeTrak* are useful for representing the contours as the previous scripts do. However, when we want to analyze those positions some more preprocessing is needed.

In *EdgeTrak* tracking we draw a certain number of points on the contour and the tool follows tongue movement automatically. While doing this, each point moves both vertically and horizontally which is disturbing for our interest because it makes us lose the reference of the position of each point. One way to solve this issue is considering both coordinates of each point in our analysis but that implies working with two properties of each point, which is hard to manage.

Since we want to work with a single property which describes the position of each point, the solution is fixing a set of values for the abscissa axis and then obtaining the corresponding arguments for those values on each contour. Acting this way we are sampling the profile of the tongue and extracting the height of each sample. This is made by the script `interpolate`.

For the interpolation we use an abscissa axis of 50 samples and therefore the resulting contours will contain that number of points. Since we are fixing the X axis, is it possible that some contours have no images for all the samples that we are considering if the contour is shorter than the interpolating vector. The defined behavior in this case is padding those unknown values with the symbol '?'. We will take this into account in following stages (*Generate .arff file for Weka processing*).

4.3.5 Temporal Derivative

By now we were only considering information about the position of the points of the tongue profile, nonetheless we are not sure about the existence of that relation between tongue position and spectral features. That is why we will cover different options and we will also consider the slope on each point of the contour apart from their position. With `spatialDerivative` script the first derivative is calculated on every point of the fixed abscissa axis on a very simple way: we only need the difference of height between samples to obtain the slope because they are all equally spaced so for each point of the contour we subtract the previous point's value from the next one's value.

Once again the usage of this function is up for the user and can be omitted by commenting the line in the code. The script receives `matrixData` and uploads its content with its spatial derivative. This allows us to calculate even higher order derivatives by executing more than one calls to the function in a row, for example if we want to test the curvature of the contour (second derivative) we will execute two calls to the function.

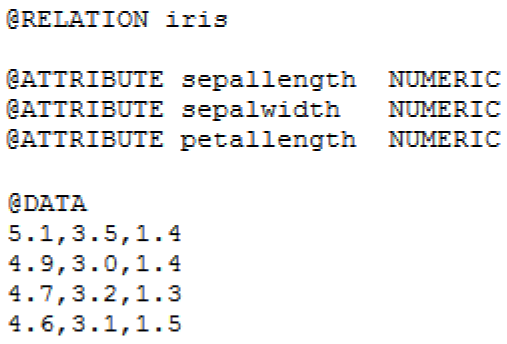
4.3.6 Generate .arff file for Weka processing

On section *Generate images and video* is said that we do not expect to find any direct relation between contours and audio visually. That is why we need a more complex tool to analyze our data. **Weka 3** will be the software chosen for this task:

As explained in [20] "Weka is a collection of machine learning algorithms for data mining tasks. (...) Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization."

We will use the classifier tool (explained in *Subjective results*) which allows us to predict one attribute of an instance as a function of the rest of its attributes. Since we are trying to relate tongue's profile information and MGC coefficients, we will create a structure on which each instance contains all the points of one frame followed by its MGC coefficients.

`CreateArff` and `extendArff` scripts are used to translate our data into that structure, making it manageable by Weka. The format on which Weka reads data is .arff (*Atribute-Relation File Format*), which content is explained in [21].



```
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC

@DATA
5.1,3.5,1.4
4.9,3.0,1.4
4.7,3.2,1.3
4.6,3.1,1.5
```

Figure 4.6. Example of .arff file content. Image extracted from [21]

Weka work with information in table format, thus the .arff file contains the definition of one table on which first all attributes are defined and then all the values corresponding to each instance. For example, in Figure 4.6 we see a simple table on which three numeric attributes are defined in `@ATRIBUTE` section and then four instances are declared in `@DATA` section, each one with its three corresponding values.

One detail should be considered in this point: Weka allows us to use unknown values for some instance writing them as '?' symbol in the .arff file. That is why in section *Interpolate Y-axis* *values* is said that we write that symbol in cases on which the contour had not image for a certain value.

Following this format, `createArff` and `extendArff` scripts receive the contours matrix and the MGC coefficients matrix (`matrixData` and `MGC`) as input and produce the corresponding .arff file as output. `CreateArff` is used for a first execution of the program since it creates the file or rewrites it in case it already existed. Once the file is created, the user can add more data from different sentences executing `extendArff`. On this kind of prediction analysis the more data we have available, the better the results will be and that is why we implemented this last option.

Both functions are complementary to each other so there is no point on executing both simultaneously. The user has to know whether he wants to create the file or extend its content and then uncomment the corresponding line of code for it.

The complete code of all these scripts previously explained is available on *Annex* section, at the end of this thesis.

5 Results and critical assessment

After all the processing, we finally have articulator and spectral data on the needed format and we can analyze the relation between them using Weka tool. For this Weka offers a powerful visual function on which all instances of a certain attribute can be represented in a graph as function of other attribute. For example, we can plot all values which a certain MGC coefficient takes in a sentence in function of each one of the corresponding contour's points.

From the resulting clouds of points we could obtain some conclusion like lineal relations or patterns into the distributions of the points.

We will divide the results of the analysis by the articulator information used: We will consider contour's points positions and first spatial derivative of them, which is the slope of the contour in each considered point.

5.1 Position of the tongue

As we said, our first analysis works over the positions of the tongue contour directly obtained from the tracking tool.

One of the recorded sentences supplied at the beginning of the thesis is processed by *EdgeTrak* and by our *Python* program. The resulting .arff file is represented on the visual tool of Weka and here we show the graphs obtained:

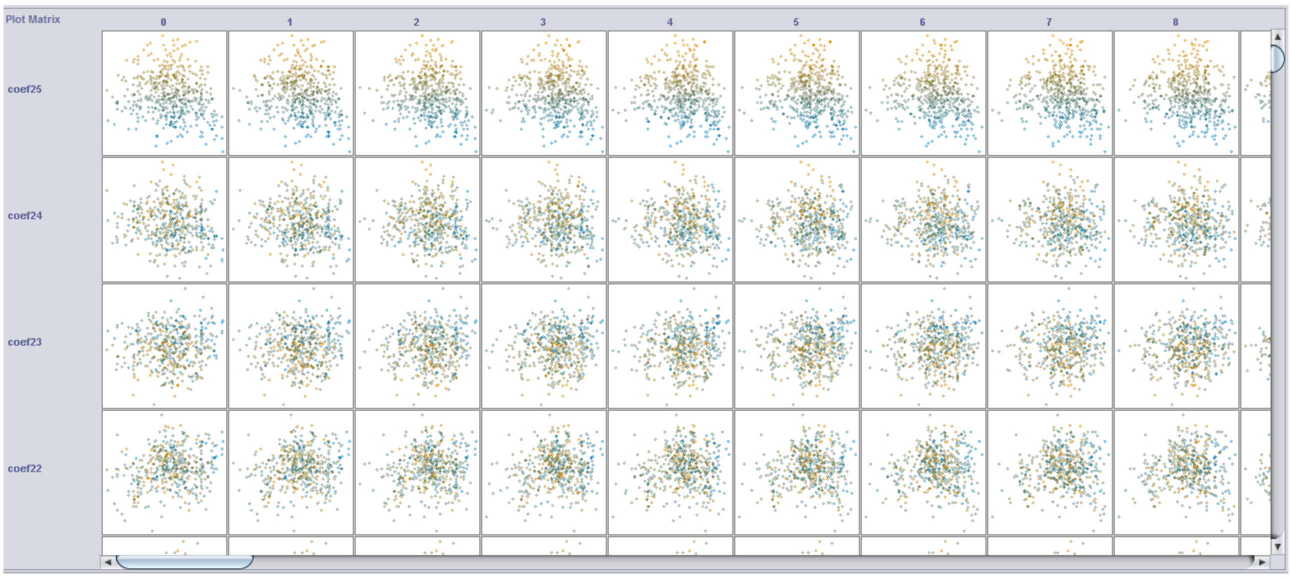


Figure 5.1. Plot of data representing contour points against MGC coefficients on Weka software

As we can see on Figure 5.1, the results seem to be clouds of points with no clear relation between any pair of contour point-coefficient. However, by checking deeper, we find one case on which some slight linear relation is displayed. That is coefficient number 11, whose results are shown below.

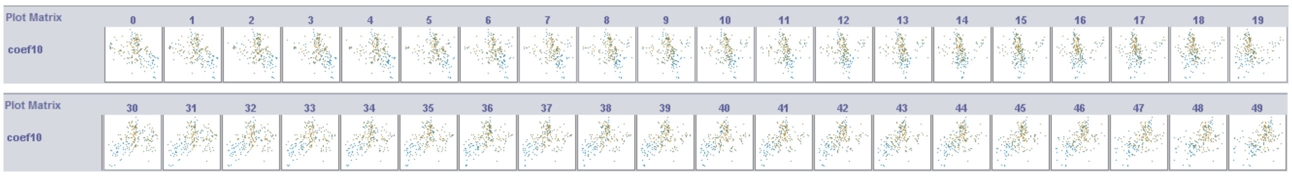


Figure 5.2. Distribution of 11th MGC coefficient values

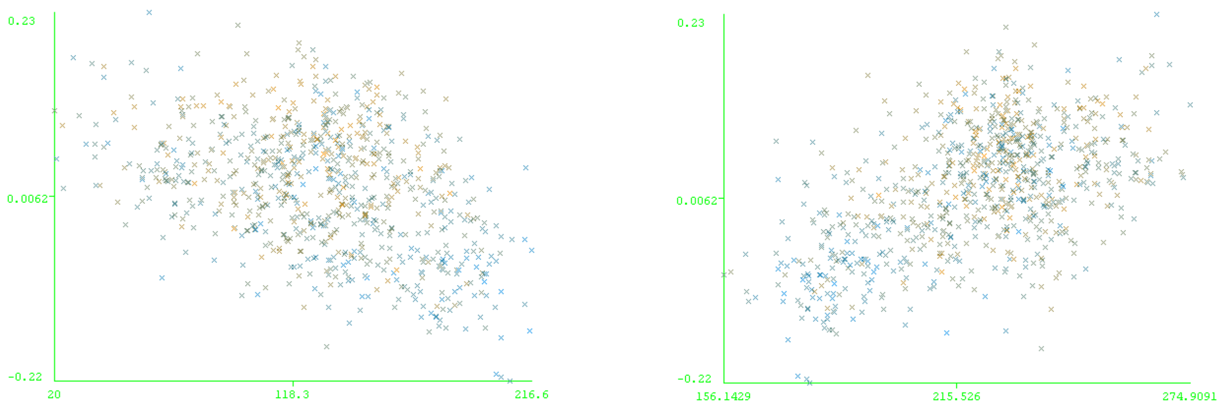


Figure 5.3. Detail of the slight linear relation of 11th MGC coefficient with contour points. Graphs represent its values in function of points number 1 (left) and number 40 (right) from a contour of 50 points

Figure 5.2 and Figure 5.3 illustrate how the 11th MGC coefficient takes values related with the position of the contour points. This linear relation evolves from the beginning to the end of the contour: with the first points of the contour (the root of the tongue) the relation is inverse, i.e. the higher the root of the tongue, the lower the value of the coefficient. Towards the tip of the tongue the relation becomes direct, so the higher the frontal points of the tip of the tongue, the higher the value of the coefficient.

5.2 First derivative

We apply now the same analysis for the slope of the contour by passing the .arff file to Weka after call once `spatialDerivative` script.

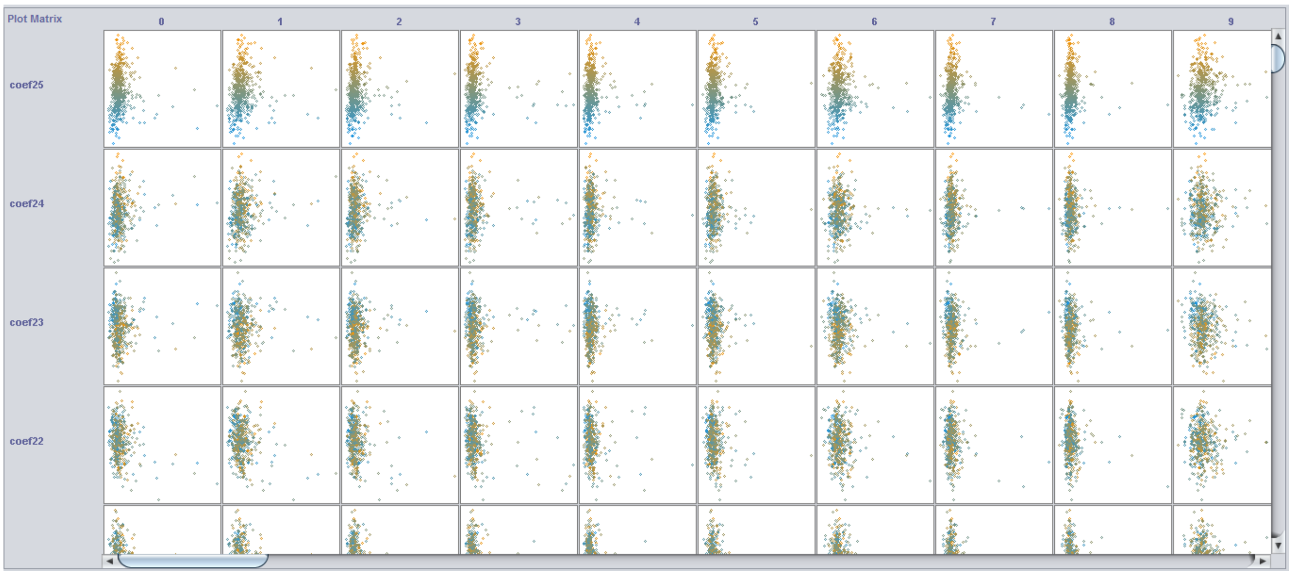


Figure 5.4. Plot of data representing contour slope against MGC coefficients on Weka software

The result of this execution is similar to the previous one: in general we obtain clouds of points with no dependencies between variables but for coefficient number 11, which has linear distribution of its values.

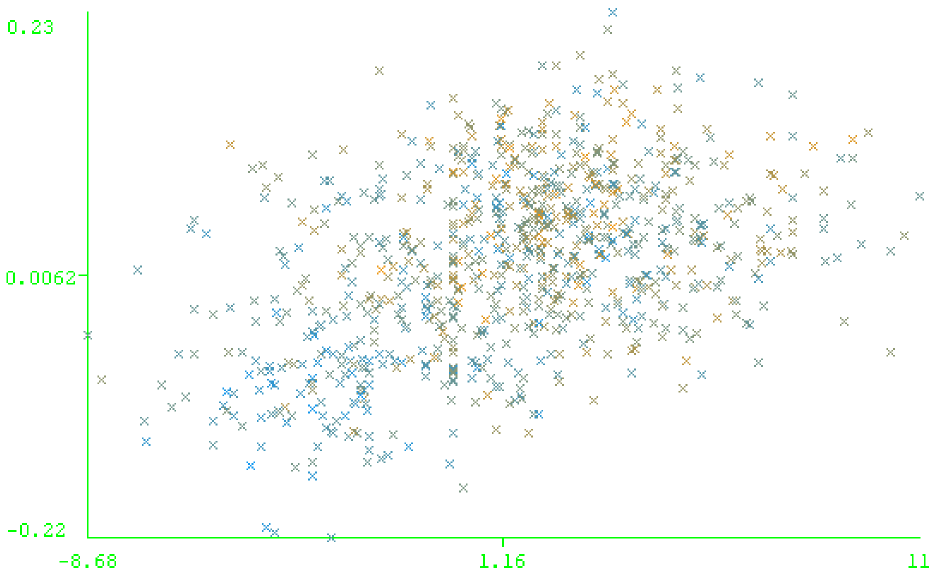


Figure 5.5. Detail of the linear relation of MGC coefficient with slope values. Graph represents its values in function of the slope in point number 24 from a contour of 50 points

In this case the relation is more clear on central points of the contour since the points of the sides have their values plotted highly compressed due to very separated outliers. These outliers are more common in both extremes of the contour because it is harder for *EdgeTrak* to follow the movement on these points and more abrupt slopes appear.

5.3 Subjective results

After this visual checking of the resulting relation, we will go further to performance one last test. On it the idea is using that relation obtained between spectrum and contour points to estimate the MGC coefficients via Weka classifiers.

Weka classify tool allows to estimate the value of an attribute as function of the rest of them in two steps: first, a model is created using some percentage of the available data (training set) and once this model is optimized, it is applied to the rest of the data (testing set) [22]. In our case we want an estimated version of all the MGC coefficients and thus the model obtained after the training stage will be applied to all the data (training and testing sets).

Weka offers different kinds of classifiers. The one that we will use is **M5P decision tree** classifier, which gave the best results on estimating tests (around 0.45 of correlation coefficient).

"A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called “root” that has no incoming edges. (...)In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values." [23].

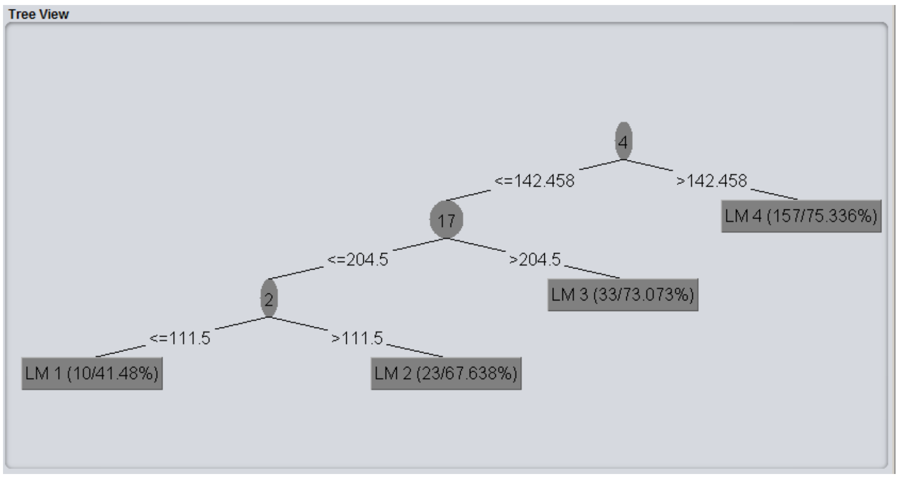


Figure 5.6. Example of decision tree obtained with Weka software

After classifying all MGC coefficients for every frame of one of the sentences and saving all them in .arff files generated by Weka, we expand the code included in section *Data processing: Python 3* to synthesize a new audio signal with the new coefficients:

```
#-----TEST AUDIO SYNTHESISED FROM THE ESTIMATED COEFF'S-----#
MGCestimated = MGC

for i in range (len(MGC[0])):
    MGCestimated[:,i] = readMGC (filename, numFrames, i)

for i in range (len(pitch)):
    pitch[i] = pitch [40]

audioSynthesised = synthesise (filename, MGCestimated, pitch)
```

The code above shows how the MGC values are saved successively on `MGCestimated` variable. After that all values of pitch are set to a fixed one. We do this to obtain the resulting synthesized audio that contains only the information provided by the MGC coefficients.

The script `readMGC` takes the values of the MGC coefficient number `i` for all the frames of the sentence and, after substituting all the coefficients by the estimated ones, the script `synthesize` reconstructs the audio signal using the values obtained from the classification. The code content of both scripts in included in last section *Annex*.

The result of the execution of the script `synthesize` is a sample of how the audio signal can be reproduced with the spectral information extracted from the articulatory information. We want to measure the quality and understandability of these sentences and for this we will perform the following listening test.

We work with the five sentences mentioned in section *Available synchronized speech and ultrasound data* and we create four audio files with four different versions of each sentence: one with the original audio (a), one with the signal after coded in MGC coefficients and pitch information and decoded from that same information (b), one with the signal decoded using our estimated MGC coefficients instead of the original ones (c) and the last one with the signal decoded using the MGC coefficients of one frame repeated in all frames like a uniform spectrogram (d).

Five subjects (including the author and the tutor of the thesis) were asked to fill a web-based listening test (available http://leszped.tmit.bme.hu/mapping2016/) about how natural and easy to understand each one of the sentences was, marking them with a number from 0 to 100. For it each one of the sentences was presented to the subject in random order. They had the original audio as reference along with the four commented versions of it, which were also displayed in random order. The subject could play each sentence as many times as he wanted in order to rate them in a scale from 0 to 100 (100 the best, 0 the worst) using a scroll control. After that the data where automatically stored and processed to obtain the average value of all marks.

The results are placed in Table 5.1 below. For each version of each sentence, the table includes the average value of the rates given by the subjects of the test.

SENTENCE	a	b	c	d
005	90	54	27	13
010	92	53	27	4
017	91	53	21	3
019	93	54	17	2
022	85	47	34	1

Table 5.1. Results of subjective experiment about synthesized sentences' quality

In Table 5.1 we can see the same pattern for all the sentences like a logical order of quality among versions: always the most preferred was the original audio followed by the decoded with the original MGC coefficients one. The third best is our version with the ultrasound-based estimated MGC coefficients and in last place the one with continuous spectra along the time.

There is some interesting conclusion to extract from these results. As we can see the value of the estimated coefficients' sentence is always in a middle point between the second and the fourth versions with little variations depending on the sentence (some sounds are easier to recognize than others). What we can extract from this is that our resulting spectra estimated from the articulatory information via classifiers is not random values and it follows the structure of the original audio enough to recognize some segments of the sentence and even understand some parts when they are simple.

6 Summary and conclusion

The idea of this thesis was treating the articulatory-to-acoustic mapping topic from a practical view. It is an interesting research topic and therefore a lot of studies have been done about it. Those papers were our starting point and we used them as basics to develop our own study.

We decided to focus that study on the relation between the movements of the human tongue that produce the phonemes when we talk and the characteristics of the voice signal which is generated from those movements. More concretely, we worked with the spectral features of the voice signal represented by the MGC coefficients of it.

For reaching that relation different tools were used such as the tongue tracking software *EdgeTrak*, different scripts made with *Python 3* to process the data that we wanted to analyze and *Weka* tool for data mining work, which was used to extract the results from the processed information.

After all the process we had to check the results obtained. We did it separated in two different kind of tests: objective ones and subjective ones.

The objective test was about representing the values of the positions of the tongue together with the MGC values. We perceived a slight linear relation between one of the MGC coefficients (the 11th value out of 26) and the positions of the points of the tongue contour. This does not give much information as a result and that is why we include some subjective tests.

The idea of the subjective experiment was to generate an audio signal from an estimated version of the coefficients obtained and test whether the result was comprehensible or at least similar to the original audio. That estimation of the coefficients was done from the articulator data using a decision tree classifier. This kind of test offers a more practical result closer to the idea that we are chasing with this study (development of synthesizer of voice or silent speech interface from articulatory information). In this case the results showed that there is a relation between articulator movements and speech signal since the estimated sentences were quite similar to the original ones although still far from being understandable.

This result gives a point to investigate possible improvements of the process and keep searching alternatives which could give better results on synthesized speech. Those improvements might be applied in every step, from using more efficient tracking tools to testing more complex classifiers for estimation. There is much to improve but also many options to explore.

References

[1] J. Clark and C. Yallop, *An introduction to phonetics and phonology*, 1st ed. Oxford, UK: B. Blackwell, 1990, pp. 13-52.

[2] T. Hueber, G. Bailly and B. Denby, "Continuous Articulatory-to-Acoustic Mapping using Phone-based Trajectory HMM for a Silent Speech Interface" in *Proc. Interspeech*, 2012, pp. 723–726.

[3] J. Wilhjelm, A. Illum, M. Kristensson and O. Andersen, *Medical diagnostic ultrasound - physical principles and imaging*, Oxford: Technical University of Denmark, 2013.

[4] M. Li, C. Kambhamettu and M. Stone, "Automatic contour tracking in ultrasound images", *Clinical Linguistics & Phonetics*, vol. 19, no. 6-7, pp. 545-554, 2005.

[5] T. G. Csapó, Speech coding - laboratory measurement, https://github.com/csapot/SpeechCoding/ (download date: Nov 11, 2016).

[6] K. Tokuda, T. Kobayashi, T. Masuko, and S. Imai, "Mel-generalized cepstral analysis - a unified approach to speech spectral estimation," in *Proc. ICSLP*, 1994, pp. 1043–1046.

[7] T. Hueber and G. Bailly, "Statistical conversion of silent articulation into audible speech using full-covariance HMM", *Computer Speech & Language*, vol. 36, pp. 274-293, 2015.

[8] T. Schultz and M. Wand, "Modeling coarticulation in EMG-based continuous speech recognition", *Speech Communication*, vol. 52, no. 4, pp. 341-353, 2010.

[9] Y. NAKAJIMA, "Non-Audible Murmur (NAM) Recognition", *IEICE Transactions on Information and Systems*, vol. 89-, no. 1, pp. 1-4, 2006.

[10] J. Gilbert, S. Rybchenko, R. Hofe, S. Ell, M. Fagan, R. Moore and P. Green, "Isolated word recognition of silent speech using magnetic implants and sensors", *Medical Engineering & Physics*, vol. 32, no. 10, pp. 1189-1197, 2010.

[11] T. Hueber, G. Chollet, B. Denby, G. Dreyfus, M. Stone, "Visuo-phonetic decoding using multri-stream and context-dependent models for an ultrasound-based silent speech interface", *Proceddings of Interspeech*, Brighton, pp. 365-369, 2009.

[12] M. Fagan, S. Ell, J. Gilbert, E. Sarrazin and P. Chapman, "Development of a (silent) speech recognition system for patients following laryngectomy", *Medical Engineering & Physics*, vol. 30, no. 4, pp. 419-425, 2008.

[13] T. Hueber, E. Benaroya, B. Denby and G. Chollet, "Statistical Mapping between Articulator and Acoustic Data for an Ultrasound-based Silent Speech Interface" in *Proc. Interspeech*, 2011, pp. 593–596.

[14] S. Wang, *Interdisciplinary computing in Java programming*, 1st ed. Boston: Kluwer Academic, 2003.

[15] T. Toda, A. Black and K. Tokuda, "Voice Conversion Based on Maximum-Likelihood Estimation of Spectral Parameter Trajectory", *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 8, pp. 2222-2235, 2007.

[16] T. Toda, A. Black and K. Tokuda, "Statistical mapping between articulatory movements and acoustic spectrum using a Gaussian mixture model", *Speech Communication*, vol. 50, no. 3, pp. 215-227, 2008.

[17] J. Gonzalez, L. Cheah, J. Gilbert, J. Bai, S. Ell, P. Green and R. Moore, "A silent speech system based on permanent magnet articulography and direct synthesis", *Computer Speech & Language*, vol. 39, pp. 67-87, 2016.

[18] T. G. Csapó, A. Deme, T. E. Gráczi, A. Markó, and G. Varjasi, “Szinkronizált beszéd- és nyelvultrahang-felvételek a SonoSpeech rendszerrel”, in MSZNY [Conference on Hungarian Computational Linguistics], 2017.

[19] U. Dentistry, 2016. [Online]. Available: https://www.dental.umaryland.edu/speech/research/ultrasound/edge-detection/. [Accessed: 08- Dec- 2016].

[20] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java", *Cs.waikato.ac.nz*, 2016. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/index.html. [Accessed: 01- Dec- 2016].

[21] "Attribute-Relation File Format (ARFF)", *Cs.waikato.ac.nz*, 2016. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/arff.html. [Accessed: 01- Dec- 2016].

[22] "Data Mining with Weka (2.2: Training and testing)", *YouTube*, 2016. [Online]. Available: https://www.youtube.com/watch?v=FMiCOx95lAc&list=PLm4W7_iX_v4NqPUjceOGd-OKNVO4c_cPD&index=9. [Accessed: 14- Dec- 2016].

[23] H. Dahan, S. Cohen, L. Rokach and O. Maimon, "Chapter 9.Decision trees", *Proactive Data Mining with Decision Trees*, 1st ed. Dordrecht: Springer, 2014.

Annex

Here we include all the codes of the scriptsthat we talked about on section *Data processing: Python 3*.

readCoord

```
def readCoord(filename):
    with open("ts/"+filename+".ts", "r") as tsFile:
        for i in range(4):
            tsFile.readline()
        numFrames=int(tsFile.readline())
        for i in range(4):
            tsFile.readline()
        numPoints=int(tsFile.readline())
        matrixCoord = []
        for i in range(numFrames):
            x_axis=[]
            y_axis=[]
            for j in range(numPoints):
                    coordenates=tsFile.readline().split()
                    x_axis.append(int(coordenates[0])-100)
                    y_axis.append(-int(coordenates[1])+450)
                    j+=1
            matrixCoord.append(x_axis)
            matrixCoord.append(y_axis)
            tsFile.readline()
        tsFile.close()
  return matrixCoord
```

getMGC

```
def getMGC(filename, numFrames):
    (x, Fs) = wavread("Audio/"+filename+".wav")
    duration = len(x)/Fs
    frshft = len(x)//numFrames
    frlen = 512
    mgc_coeff = np.zeros(numFrames) # list of zeros
    pitch = np.zeros(numFrames) # list of zeros
    order = 25
    alpha = 0.41
    stage = 5
    gamma = -1.0 / stage
    pitch = pysptk.swipe(x.astype(np.float64), fs=Fs, hopsize=frshft, min=60, max=260,
otype="pitch")
    frames = librosa.util.frame(x, frame_length=frlen,
hop_length=frshft).astype(np.float64).T
    frames *= pysptk.blackman(frlen)
    mgc_coeff = np.apply_along_axis(pysptk.mgcep, 1, frames, order, alpha, gamma)
    return (mgc_coeff, pitch, duration)
```

audioImagesVideo

```
def audioImagesVideo(filename, numFrames, matrixCoord):
    (x, Fs) = wavread("Audio/"+filename+".wav")
    frameLen = len(x)//numFrames
    segments = [type('', (), {})() for c in range(numFrames)]
    for i in range(numFrames):
        segments[i] = x[i*frameLen : (i+1)*frameLen]
    FFMpegWriter = manimation.writers['ffmpeg']
    metadata = dict(title='Togue contour '+filename, artist='Matplotlib')
    writer = FFMpegWriter(fps=5, metadata=metadata)
    fig = plt.figure()
    with writer.saving(fig, "ImagesAudio/VideoA_"+filename+".mp4", numFrames):
        for j in range(numFrames):
            plt.cla()
            plt.subplot(211)
            plt.cla()
            plt.plot(matrixCoord[2*j], matrixCoord[2*j+1])
            plt.axis([0, 500, 0, 300])
            plt.subplot(212)
            plt.plot(segments[j])
            plt.axis([0, 1200, -27000, 27000])
            plt.savefig('ImagesAudio/' + str(j+1))
            writer.grab_frame()
    return
```

contourImagesVideo

```
def contourImagesVideo(filename, matrixCoord, numFrames, duration):
    fps = numFrames//duration
    FFMpegWriter = manimation.writers['ffmpeg']
    metadata = dict(title='Togue contour '+filename, artist='Matplotlib')
    writer = FFMpegWriter(fps, metadata=metadata)
    fig = plt.figure()
    with writer.saving(fig, "ImagesContour/VideoC_"+filename+".mp4", numFrames):
        for i in range(numFrames):
            plt.plot(matrixCoord[2*i], matrixCoord[2*i+1])
            plt.axis([0, 500, 0, 300])
            plt.savefig('ImagesContour/' + str(i+1))
            writer.grab_frame()
            plt.cla()
    return
```

spectrumImagesVideo

```
def spectrumImagesVideo(filename, numFrames, matrixCoord, MGC):
    FFMpegWriter = manimation.writers['ffmpeg']
    metadata = dict(title='Togue contour '+filename, artist='Matplotlib')
    writer = FFMpegWriter(fps=5, metadata=metadata)
    fig = plt.figure()
    with writer.saving(fig, "ImagesSpectrum/VideoA_"+filename+".mp4", numFrames):
        for j in range(numFrames):
            plt.cla()
            plt.subplot(211)
            plt.cla()
            plt.plot(matrixCoord[2*j], matrixCoord[2*j+1])
            plt.axis([0, 500, 0, 300])
            plt.subplot(212)
            plt.plot(MGC[j][1:])
            plt.axis([0, 24, -0.5, 1])
            plt.savefig('ImagesSpectrum/' + str(j+1))
            writer.grab_frame()
    return
```

interpolate

```
def interpolate (matrixCoord):
    length = len(matrixCoord)//2
    matrixData = []
    maxim = 350
    minim = 100
    spaced = 5
    for i in range (length):
        f=interp1d(matrixCoord[2*i], matrixCoord[2*i+1])
        numZerosPrev = 0
        numZerosPost = 0
        if (max(matrixCoord[2*i]) < 350 and min(matrixCoord[2*i])>100):
            init= min(matrixCoord[2*i])
            final= max(matrixCoord[2*i])
            x= np.arange(init, final, spaced)
            interpolated = f(x)
            numZerosPrev = (init - minim)//spaced
            numZerosPost = ((maxim - final)//spaced)+1
            zerosPrev = []
            zerosPost = []
            for j in range(numZerosPrev):
                zerosPrev.append('?')
            for j in range(numZerosPost):
                zerosPost.append('?')
            interpolated = zerosPrev + list(interpolated) + zerosPost
        elif min(matrixCoord[2*i]) > 100:
            init= min(matrixCoord[2*i])
            while (init%5) != 0:
                init = init+1
            x= np.arange(init, maxim, spaced)
            interpolated = f(x)
            numZerosPrev = (init - minim)//spaced
            zerosPrev = []
            for j in range(numZerosPrev):
                zerosPrev.append('?')
            interpolated = zerosPrev + list(interpolated)
        elif max(matrixCoord[2*i]) < 350:
            final= max(matrixCoord[2*i])
            x= np.arange(minim, final, spaced)
            interpolated = f(x)
            numZerosPost = ((maxim - final)//spaced)+1
            zerosPost = []
            for j in range(numZerosPost):
                zerosPost.append('?')
            interpolated = list(interpolated) + zerosPost
        else :
            x = np.arange(minim, maxim, spaced)
            interpolated = f(x)
        matrixData.append(interpolated)
    return matrixData
```

spatialDerivative

```python
def spatialDerivative (matrixData):
    numFrames = len(matrixData)
    numPoints = len(matrixData[0])
    matrixDerivate = []
    for i in range (numFrames):
        vector = []
        for j in np.arange(1,numPoints-1):
            if (matrixData[i][j+1]=='?') or (matrixData[i][j-1]=='?'):
                vector.append('?')
            else:
                vector.append(matrixData[i][j+1] - matrixData[i][j-1])
        matrixDerivate.append(vector)
    return matrixDerivate
```

createArff

```python
def createArff(matrixData, MGC):
    numFrames = len(matrixData)
    numSamples = len(matrixData[0])
    numCooefs = len(MGC[0])
    with open("Weka/contours.arff", "w") as arffFile:
        arffFile.write("@RELATION contour\n\n")
        for i in range(numSamples):
            arffFile.write("@ATTRIBUTE "+str(i)+"  NUMERIC\n")
        for i in range(numCooefs):
            arffFile.write("@ATTRIBUTE coef"+str(i)+" NUMERIC\n")
        arffFile.write("\n@DATA\n")
        for i in range (numFrames):
            frame = ""
            for j in range (numSamples):
                frame = (frame+str(matrixData[i][j])+",")
            for j in range (numCooefs-1):
                frame = (frame+str(MGC[i][j])+",")
            frame = (frame+str(MGC[i][25])+"\n")
            arffFile.write(frame)
        arffFile.close()
    return
```

extendArff

```python
def extendArff(matrixData, MGC):
    numFrames = len(matrixData)
    numSamples = len(matrixData[0])
    numCooefs = len(MGC[0])
    with open("Weka/contours.arff", "a") as arffFile:
        for i in range (numFrames):
            frame=""
            for j in range (numSamples):
                frame=(frame+str(matrixData[i][j])+",")
            for j in range (numCooefs-1):
                frame=(frame+str(MGC[i][j])+",")
            frame = (frame+str(MGC[i][25])+"\n")
            arffFile.write(frame)
        arffFile.close()
    return
```

Below we show the content of the scripts mentioned in section *Subjective result*:

readMGC

```
def readMGC (filename, numFrames, MGCnum):
    with open("Weka/"+filename+"_"+str(MGCnum)+".arff", "r") as arffFile:
        vector = []
        for i in range (5):
            arffFile.readline()
        for i in range (numFrames):
            vector.append(arffFile.readline())
        arffFile.close()
        return vector
```

synthesize

```
def synthesize(filename, MGC, pitch):
    (x, Fs) = wavread("Audio/"+filename+".wav")
    frlen = round(0.025 * Fs)
    frshft = len(x)//len(MGC)
    nframes = len(pitch)
    order = 25
    alpha = 0.41
    stage = 5
    gamma = -1.0 / stage
    source_excitation = pysptk.excite(pitch, frshft)
    mglsadf_coeff = np.apply_along_axis(pysptk.mgc2b, 1, MGC, alpha, gamma);
    synthesizer = Synthesizer(MGLSADF(order=order, alpha=alpha, stage=stage),
frshft)
    audioSynthesised = synthesizer.synthesis(source_excitation, mglsadf_coeff)
    return (audioSynthesised)
```