Hannah Loukusa

Mechanical Engineering Undergraduate

Saint Cloud State University

Saint Cloud, MN, United States

[hlloukusa@stcloudstate.edu](mailto:hlloukusa@stcloudstate.edu)

hannahloukusa@gmail.com

**Abstract – In this paper we describe a method for extracting vocal features from an audio recording such that speech may be reconstructed from these vocal measurements while maintaining emphasis and stress with minimal annotation. We also present a script for the linguistic research tool Praat for performing this extraction and reconstruction via KlattGrid synthesis. [1] This technique contributes to the ongoing research of Dr. Ettien Koffi and Dr. Mark Petzold to preserve the Betine language, one of 10 critically endangered languages spoken in Côte d’Ivoire, West Africa.**

***Keywords–Speech Synthesis, Formant Synthesis, Formant Extraction, Formant Bandwidths, Endangered Languages, Betine, Klatt Synthesis***

**I. Introduction**

Many languages are rapidly going extinct, in that the number of native speakers has been or currently is rapidly decreasing [2]. Linguists are attempting to preserve these languages for their linguistic, historic and cultural value through study ofspeech samples. Part of this work involvesidentification of speech patterns and phonetic components.

Currently, these preservation techniques require large amounts of data and researcher time to model the language, normalize text, and reproduce these in software [3]. This paper details the implementation and use of Praat scripting to automate the extraction of acoustic correlates, namely pitch, formants, and intensity, from a sound file of the female Betine name /asamala/, and to generate a KlattGrid object that captures these measurements. Given sufficient Jacob Haapoja

Computer Science Undergraduate

Saint Cloud State University

Saint Cloud, MN, United States

[jlhaapoja@stcloudstate.edu](mailto:jlhaapoja@stcloudstate.edu)

Jhaapoja1997@gmail.com

formant, bandwidth, intensity, and duration data for each phoneme, it should be possible to reconstruct an approximation of any possible combination of voiced phonemes within the language, including simple fricatives.

For this implementation, a phoneme invariance approach was taken, as incorporating analysis of spectral tilt and allophones would greatly complicate the research. This study functions as a proof of concept for this approach to speech reproduction and language preservation. Further research is required to identify and catalogue allophones and determine the link between allophones, intonation, and word structure.

**The KlattGrid**

The KlattGrid is a speech synthesizing tool in Praat that uses a source-filter model based on the research of Dennis H. Klatt [4], [5]. The KlattGrid is time based, allowing the user to alter the characteristics of the sound at any desired time. This is different from other types of Klatt synthesizers, which are generally frame-based [5].

The KlattGrid has three sources, namely voicing, aspiration, and frication. Voicing is created using a sine wave, while aspiration and frication are created with turbulent air, or noise [4]. These mimic the voiced and unvoiced noises a person creates when they speak. These sources generate sounds that are passed through formant tiers to change the character of the sound. The formant tiers include oral formants, nasal formants, tracheal formants, and frication formants, which mimic the pharynx, oral, and nasal cavities [5].

Diagram

Description automatically generated

**Figure 1:** Schematic of the KlattGrid synthesizer [5]

Figure 1 displays a schematic of the KlattGrid. The KlattGrid was used in this project to generate phonemes for the Betine name /asamala/. The specific parameters used are mentioned later.

**II. Implementation**

**A. Manual KlattGrid Construction**

Authors 1 and 2 started the speech synthesizing process by manually annotating their given audio sample by hand, identifying phoneme boundaries and taking measurements with Praat’s graphical user interface (GUI).

Table

Description automatically generated

**Figure 2:** Annotated TextGrid for [a]

Figure 2 shows a sample of this process, synthesizing the phoneme [a]. The annotated tiers of the textgrid are as follows: word, phoneme, F0, F1, F2, F3, F4, intensity, duration, B1, B2, B3, B4, where F0-F4 and B1-B4 refer to formant and bandwidth, respectively.

These values were used to construct a manual script that generated a KlattGrid based on those tier measurements. All data was hard coded, and points were added line by line. The following sample of code shows this process for the same example phoneme [a]. The values that are “0” indicate a start time of zero seconds, as this phoneme is at the beginning of the word. A slight amount of frication was added to produce a less robotic sounding vowel.

*# 'A'*

*Add pitch point: 0, 114*

*Add voicing amplitude point: 0, 80*

*Add frication amplitude point: 0, 5*

*Add frication bypass point: 0.5, 0*

*Add oral formant frequency point: 1, 0, 741*

*Add oral formant bandwidth point: 1, 0, 137*

*Add oral formant frequency point: 2, 0, 1283*

*Add oral formant bandwidth point: 2, 0, 147*

*Add oral formant frequency point: 3, 0, 2459*

*Add oral formant bandwidth point: 3, 0, 277*

*Add oral formant frequency point: 4, 0, 3906*

*Add oral formant bandwidth point: 4, 0, 751*

Once a suitably intelligible result was reached with the manual script, the authors began to develop an automated script for constructing such a KlattGrid from just an audio recording and a TextGrid object with minimal annotation.

**B. Initialization of Automated Script**

The automated method consists of two parts which are performed simultaneously in the script. The script has two initial requirements, and each part can be made to perform independently with adaptations to the script. The first of these requirements is a sample of speech audio, and the second is a Praat TextGrid with each phoneme to replicate marked with boundary intervals and labeled with a 1 or 0 to indicate frication. Any intervals without sound should be left unlabeled.

The script begins with an information dialogue prompting for clearly labeled analysis parameters that will be used in the extraction and subsequent reconstruction. After this it verifies that preselection requirements are met, prompting the user to do so if necessary.

*#CREATES FORM WITH INSTRUCTIONS*

*form Start up instructions*

*comment ------------------------------------------------*

*comment WELCOME!*

*comment INSTRUCTIONS*

*comment 1. Open the sound and textgrid files in the Praat Objects window.*

*comment 2. In the Praat Objects window, assure the sound and textgrid files are selected.*

*comment 3. Fill in the boxes below as instructed.*

*comment ------------------------------------------------*

*comment Enter the tier number that contains ones and zeros*

*integer get\_data\_from\_tier 3*

*comment Formant analysis parameters*

*integer maximum\_number\_of\_formants 5*

*positive maximum\_formant 5500*

*positive window\_length 0.025*

*real preemphasis\_from 50*

*comment Pitch analysis parameters*

*positive pitch\_time\_step 0.01*

*positive minimum\_pitch 75*

*positive maximum\_pitch 500*

*comment Intensity analysis parameters*

*positive maximum\_intensity 100*

*positive int\_time\_step 0.01*

*endform*

Next, the script scans through the TextGrid tier indicated in the initial dialogue. This can be any tier number but must be the tier in which the fricative specification resides. The following is an example of the verification code used in the script.

*#ASSURES FILES ARE SELECTED*

*if numberOfSelected ("Sound") <> 1 or numberOfSelected ("TextGrid") <> 1*

*exitScript: "Please select a Sound and a TextGrid first."*

*endif*

*sound = selected ("Sound")*

*textgrid = selected ("TextGrid")*

The script makes note of the start and end times of any unlabeled intervals at the start of the recording. This is to ensure that the “silent” part of the sound file before the utterance of interest begins is omitted from the analysis. This duration is stored in the *silence\_duration* variable, which is subtracted from the interval times to determine the times to set points at in the KlattGrid object. This duration is then printed in the text window.

*#EXTRACTS DURATION OF INTERVAL BEFORE WORD BEGINS, CALLED SILENCE DURATION*

*selectObject: textgrid*

*interval\_label$ = Get label of interval: get\_data\_from\_tier, 1*

*if interval\_label$ == ""*

*t1 = Get starting point: get\_data\_from\_tier, 1*

*t2 = Get end point: get\_data\_from\_tier, 1*

*silence\_duration = t2 - t1*

*else*

*silence\_duration = 0.0*

*endif*

The total duration of the utterance is determined by summing the durations of all successive labeled intervals. This is stored in the *total\_duration* variable, and is used when initializing the KlattGrid as the duration parameter, then it is similarly printed in the text window as a new line. This script is intended to be used on single word utterances, so any word breaks will cause the script to only replicate the first word in the utterance.

*#CALCULATES DURATION OF WORD*

*total\_duration = 0.0*

*selectObject: textgrid*

*n = Get number of intervals: get\_data\_from\_tier*

*for i to n*

*interval\_label$ = Get label of interval: get\_data\_from\_tier, i*

*if interval\_label$ <> ""*

*t1 = Get starting point: get\_data\_from\_tier, i*

*t2 = Get end point: get\_data\_from\_tier, i*

*dur = t2-t1*

*total\_duration = total\_duration + dur*

*endif*

*endfor*

*appendInfoLine: total\_duration*

The KlattGrid object is then created with the relevant parameters.

*klattGrid = Create KlattGrid: "from your word", 0, total\_duration, 6, 1, 1, 6, 1, 1, 1*

This study focuses primarily on voiced phonemes and simple fricatives. Voiced phonemes directly take the measured values for the phoneme and maintain those levels throughout the duration with only minor frication added, whereas fricatives only maintain the voicing amplitude, or intensity as measured in Praat, for part of the duration. This is done as voicing from a preceding voiced phoneme will blend into the fricative, before abruptly beginning again with the next voiced phoneme. For other phoneme types such as stops, additional research is required.

**C. Automated Feature Extraction**

In this part, pitch, intensity, and formant objects are extracted using Praat’s built in algorithms. A count of intervals on the tier specified in the initial dialogue is taken, along with the start and end times of each interval from the TextGrid object. This count is stored in the variable labeled n. These are used to average the pitch, intensity, and formants over those intervals.

*#EXTRACTS AVERAGE PITCH, F1-F4, AND INTENSITY OF EACH INTERVAL THAT CONTAINS TEXT IN CHOSEN TIER*

*selectObject: sound*

*pitch = To Pitch: pitch\_time\_step, minimum\_pitch, maximum\_pitch*

*selectObject: sound*

*intensity = To Intensity: maximum\_intensity, int\_time\_step*

*selectObject: sound*

*formant = To Formant (burg): 0, maximum\_number\_of\_formants, maximum\_formant, window\_length, preemphasis\_from*

*selectObject: textgrid*

*n = Get number of intervals: get\_data\_from\_tier*

Measurements for frequency and bandwidth are taken in Hz while intensity is taken in dB. This is done with a programming structure called a for loop that counts through the number of intervals and extracts the relevant data into variables that are then passed to the next part. Unlabeled intervals are ignored during this process. It was found that setting the bandwidth value to 20% of the formant value yielded indistinguishable results in comparison to using the extracted bandwidth values, so the percentage method was used.

*for i to n*

*interval\_label$ = Get label of interval: get\_data\_from\_tier, i*

*if interval\_label$ <> ""*

*t1 = Get starting point: get\_data\_from\_tier, i*

*t1\_klatt = t1-silence\_duration*

*t2 = Get end point: get\_data\_from\_tier, i*

*t2\_klatt = t2-silence\_duration*

*selectObject: pitch*

*f0 = Get mean: t1, t2, "Hertz"*

*f0 = round(f0)*

*f0$ = fixed$(f0,0)*

*selectObject: formant*

*f1 = Get mean: 1, t1, t2, "Hertz"*

*f2 = Get mean: 2, t1, t2, "Hertz"*

*f3 = Get mean: 3, t1, t2, "Hertz"*

*f4 = Get mean: 4, t1, t2, "Hertz"*

*f1$ = fixed$(f1,0)*

*f2$ = fixed$(f2,0)*

*f3$ = fixed$(f3,0)*

*f4$ = fixed$(f4,0)*

*selectObject: intensity*

*int = Get mean: t1, t2, "energy"*

*int$ = fixed$(int,0)*

To verify the integrity of the automated correlate extraction process, a comparison of the correlate values for manually extracted phonemes and script-extracted phonemes was executed. The results are in Table 1.

**Table 1:** Comparison of correlate values for manually extracted phonemes (black) and script-extracted phonemes (red).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | [a] | [s] | [a] | [m] | [a] | [l] | [a] |
| F0 | 114  114 | 117  117 | 124  124 | 125  125 | 130  131 | 127  127 | 122  122 |
| F1 | 741  741 | 831  832 | 675  676 | 647  647 | 683  684 | 567  568 | 690  691 |
| F2 | 1283  1283 | 1744  1745 | 1220  1220 | 1085  1089 | 1185  1184 | 1110  1111 | 1172  1172 |
| F3 | 2459  2459 | 2757  2757 | 2542  2544 | 2747  2751 | 2610  2611 | 2618  2617 | 2490  2491 |
| F4 | 3906  3906 | 3830  3828 | 3705  3706 | 3742  3679 | 3364  3365 | 3831  3827 | 3679  3679 |
| Int | 80  81 | 74  74 | 83  83 | 78  79 | 82  83 | 82  83 | 82  83 |

These results indicate that the extracted measurements have a variance of less than the “just noticeable difference” threshold for each value, and so will produce sounds indistinguishable from those produced using the manual measurements.

**D. Automated KlattGrid Construction**

This part is performed after the data extraction for each interval as the script loops through the intervals. At each interval, the data extracted is used to add a new grid point in the KlattGrid object at the start time of that interval. If a particular interval is marked as fricative by having a 1 for the interval label, the intensity of that region is added as a frication amplitude point which is scaled down to 90%. This is done to soften the raw frication generated by the Klatt synthesizer. Conversely, if the interval is marked as non-fricative with a label of “0,” the intensity is still added as a fricative amplitude point, but instead scaled down to 10%. This adds some slight variation to the produced sound which results in a smoother and slightly less robotic sounding reconstruction.

*selectObject: klattGrid*

*Add pitch point: t1\_klatt, f0*

*Add voicing amplitude point: t1\_klatt, int*

*if interval\_label$ == "0"*

*Add frication amplitude point: t1\_klatt, 5*

*Add frication bypass point: 0.5, t1\_klatt*

*elsif interval\_label$ == "1"*

*Add frication amplitude point: t1\_klatt, 60*

*Add frication bypass point: 0.5, t1\_klatt*

*Add voicing amplitude point: t1\_klatt+0.103, 0*

*else*

*exitScript: "Please place zeros and ones in the*

*frication tier."*

*endif*

*Add oral formant frequency point: 1, t1\_klatt, f1*

*Add oral formant bandwidth point: 1, t1\_klatt, f1\*0.2*

*Add oral formant frequency point: 2, t1\_klatt, f2*

*Add oral formant bandwidth point: 2, t1\_klatt, f2\*0.2*

*Add oral formant frequency point: 3, t1\_klatt, f3*

*Add oral formant bandwidth point: 3, t1\_klatt, f3\*0.2*

*Add oral formant frequency point: 4, t1\_klatt, f4*

*Add oral formant bandwidth point: 4, t1\_klatt, f4\*0.2*

**III. Results & Analysis**

The final synthesized version of /asamala/, which was synthesized using the automated Praat script, bears a strong resemblance to the original sound file. The rhythm was preserved, as well as the general sound of each phoneme. However, the consonants are not as articulate as the original version, and the whole sound file sounds muffled. The [s] also sounds more like noise rather than a focused fricative. It is possible that more research into the parameters of the KlattGrid could yield clearer consonants and reduce muffling.

As seen in Figures 3 and 4, visualization of the spectrograms and waveforms indicate that the synthesized version very closely matches the general structure of the original file.

A picture containing chart

Description automatically generated

**Fig 3:** spectrograms of the original (top) and synthesized file (bottom)

Graphical user interface, chart

Description automatically generated

**Fig 4:** waveforms of the original (top) and synthesized file (bottom)

**A. Visualization**

The visuals included in this paper were generated in python using the Parselmouth[6], Numpy[7], Matplotlib[8], and OS libraries. This script uses 2 functions titled *generate\_waveforms()* and *generate\_spectrograms().* Each of these functions takes two string parameters titled *orig* and *auto* which hold the paths to or names of the wave files to generate figures for. Parselmouth was used to import the wave files and extract waveform and spectrogram data for display. A pyplot was used to graph the waveforms, while a colormesh was used to graph the spectrograms, both of these are tools of the Matplotlib library.

**IV. Conclusion**

Current language preservation techniques require large amounts of data that must be processed by hand to form a corpus of speech data. The goal of this study was to identify a method for automating this information gathering, and to use this extracted data to reconstruct sounds and words in these languages from these simple measurements.

A script was created that implements Praat’s KlattGrid interface to synthesize speech from manually extracted data, which was used as a foundation and reference for a further evolution of this script. The revised script was designed in two parts to automatically extract the data used to generate the initial script, then apply that data to a newly generated KlattGrid object. This method requires only minimal annotation by the researcher to identify fricatives and phoneme boundaries, greatly reducing the work required to transcribe, preserve, and reproduce audible approximations for these endangered languages.

This work has the potential to aid researchers in the cataloguing and preservation of languages from minimal recorded data, and functions as a proof of concept for a phonetic invariance approach to language reconstruction.

**V. Future Work**

This project demonstrated that speech reconstruction via the KlattGrid in Praat is possible. Future efforts may focus on improved fricative synthesis, generation of all categories of phonemes, automatic phoneme separation, and building a repository of Betine allophones for arbitrary word generation.

The fricative generated in this project made use of white noise, which is the pure output of the frication source in the Klatt synthesizer. Experimentation with the frication formant parameters had little effect on the produced sound. Since fricatives generally focus around a specific frequency, applying an equalizer to reduce noise around other frequencies had a significant effect on the clarity of the [s] produced. This was not implemented in this project as an external tool was used for this equalization, however future research may involve focusing the produced noise around different frequencies to form other fricatives.

The method of speech reconstruction used in this project worked well for [a], [s], [m], and [l]. However, if other phonemes are to be reconstructed, the authors suspect that the process may need to be altered. For example, a stop has characteristics that make it very different from a liquid or a nasal. It may be that only the parameters of the formant tiers need to be altered, but other changes may need to be made in addition to this.

The Praat script used in this project eliminated the need for a user to manually gather the correlate values for each phoneme. However, the user still needs to manually add boundaries between phonemes and insert a “1” or “0” to differentiate between fricatives and non-fricatives, with intervals lacking sound left unlabeled. An automatic method to separate phonemes would greatly speed up the process and reduce dependence on the user.

A proposed method to accomplish this is to analyze the waveforms of words and find patterns that correlate to the beginning and end of phonemes. The waveform rather than pitch, intensity, or formants is suggested due to the rapidly changing nature of the waveform. The pitch, intensity, and formants within a word change gradually between phonemes, so using them to delimitate the boundaries between phonemes is less reliable. Indeed, when drawing the boundaries manually between phonemes for /asamala/, the authors found themselves using the waveform to most strongly inform their decision of where to place the boundaries. Once clear patterns are found that correlate to the beginning and end of phonemes, these can be implemented into or in addition to the Praat script to automatically separate phonemes.

The method of speech parsing may also be reconsidered for future speech synthesis. The intended direction for this project was to collect average values for the correlates for each phoneme and use those values to synthesize a KlattGrid for a phoneme whenever it appeared in a word. This is the phonemic invariant approach. However, this approach does not account for the many allophones of a single phoneme. For example, a generic [o] phoneme would not include nasalization when places after an [n]. An allophonic approach would allow for more accurate speech synthesis. Its realization would require the researcher to collect data for every allophone in the language, which would take a considerable amount of time. However, the authors argue that the resulting increase in linguistic accuracy would be worth the time investment.

Given a linguistic transcription of each phoneme in an utterance, it may be possible to automatically determine distinct allophones of those phonemes from an audio file by their differences to the pure phoneme itself, along with their adjacent phonemes. This would alert the researcher of new allophones not already in the repository.

**VI. References**

**[1]** D. H. Klatt & L. C. Klatt. “Analysis, synthesis, and perception of voice quality variations among female and male talkers.” *The Journal of the Acoustical Society of America*. vol. 87, no. 2, pp. 820-857, 1990.

**[2]** K.Hale. “Language Endangerment and the Human Value of Linguistic Diversity.” *Language* vol. 68, no. 1, pp. 35-42, 1992.

**[3]** K. Ettien and M. Petzold. **“**A Tutorial on Formant-Based Speech Synthesis for the Documentation of Critically Endangered Languages.” *Linguistic Portfolios.* vol 11, no 3, pp 1-31, 2022.

**[4]** D. H. Klatt. “Software for a cascade/parallel formant synthesizer.” *The Journal of the Acoustical Society of America*. vol. 67, no. 3, pp. 971–995, Mar. 1980, doi: 10.1121/1.383940

[5] D. Weenink. “The KlattGrid Speech Synthesizer.” *INTERSPEECH 2009, 10th Annual Conference of the International Speech Communication Association*, Brighton, United Kingdom, September 2009, doi: 10.21437/Interspeech.2009-591

[6] Y. Jadoul, B. Thompson, and B. de Boer. “Introducing Parselmouth: A Python interface to Praat.” *Journal of Phonetics*, vol. 71, pp. 1-15, 2018. https://doi.org/10.1016/j.wocn.2018.07.001

[7] C. R. Harris et al. ”Array programming with NumPy.” *Nature*, vol. 585, pp. 357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[8] M. Droettboom, et. al. “matplotlib: v1.4.3 (v1.4.3).” 2015. *Zenodo*.

https://doi.org/10.5281/zenodo.15423

**Appendix**

**A. Program Code**

**(i) Praat Script**

# automated\_word\_klattGrid.praat  
  
# DESCRIPTION OF PROGRAM  
 # This script creates a klattGrid from a sound file in Praat.  
 # You will need a sound file with a textgrid, in which one tier contains boundaries around each  
 # segment and each segment in that tier has a "1" or "0" if it is or is not a fricative, respectively.  
 # Choose this tier number when the script asks for a tier number.  
 # To run the script, click Run > Run on the menu at the top of this window.  
  
# CREDITS  
 # This script was created by Hannah Loukusa and Jacob Haapoja in 2023 in Praat, Version 6.1.38  
 # The following sources were used in the scripting process:  
 # We can list sources here...  
  
  
#----------------------------------------------------------------------------------------------------------  
  
  
#CREATES FORM WITH INSTRUCTIONS  
form Start up instructions  
 comment ----------------------------------------------------------------------------------------------------------------------------  
 comment WELCOME!  
 comment INSTRUCTIONS  
 comment 1. Open the sound and texgrid files in the Praat Objects window.  
 comment 2. In the Praat Objects window, assure the sound and textgrid files are selected.  
 comment 3. Fill in the boxes below as instructed.  
 comment ----------------------------------------------------------------------------------------------------------------------------  
 comment Enter the tier number that contains ones and zeros  
 integer get\_data\_from\_tier 3  
 comment Formant analysis parameters  
 integer maximum\_number\_of\_formants 5  
 positive maximum\_formant 5500  
 positive window\_length 0.025  
 real preemphasis\_from 50  
 comment Pitch analysis parameters  
 positive pitch\_time\_step 0.01  
 positive minimum\_pitch 75  
 positive maximum\_pitch 500  
 comment Intensity analysis parameters  
 positive maximum\_intensity 100  
 positive int\_time\_step 0.01  
   
endform  
  
#ASSURES FILES ARE SELECTED  
 if numberOfSelected ("Sound") <> 1 or numberOfSelected ("TextGrid") <> 1  
 exitScript: "Please select a Sound and a TextGrid first."  
 endif  
 sound = selected ("Sound")  
 textgrid = selected ("TextGrid")  
  
#EXTRACTS DURATION OF INTERVAL BEFORE WORD BEGINS, CALLED SILENCE DURATION  
 selectObject: textgrid  
 interval\_label$ = Get label of interval: get\_data\_from\_tier, 1  
 if interval\_label$ == ""  
 t1 = Get starting point: get\_data\_from\_tier, 1  
 t2 = Get end point: get\_data\_from\_tier, 1  
 silence\_duration = t2 - t1  
 else  
 silence\_duration = 0.0  
 endif  
  
appendInfoLine: silence\_duration  
  
#CALCULATES DURATION OF WORD  
 total\_duration = 0.0  
 selectObject: textgrid  
 n = Get number of intervals: get\_data\_from\_tier  
 for i to n  
 interval\_label$ = Get label of interval: get\_data\_from\_tier, i  
 if interval\_label$ <> ""  
 t1 = Get starting point: get\_data\_from\_tier, i  
 t2 = Get end point: get\_data\_from\_tier, i  
 dur = t2-t1  
 total\_duration = total\_duration + dur  
 endif  
 endfor  
  
appendInfoLine: total\_duration  
  
klattGrid = Create KlattGrid: "from your word", 0, total\_duration, 6, 1, 1, 6, 1, 1, 1  
  
#EXTRACTS AVERAGE PITCH, F1-F4, AND INTENSITY OF EACH INTERVAL THAT CONTAINS TEXT IN CHOSEN TIER  
#(AN INTERVAL IS A SECTION BETWEEN BOUNDARIES)  
# t1 AND t2 REFER TO START AND END TIME OF AN INTERVAL. "Hertz" AND "energy" REFER TO SETTINGS FOR F0, F1-F4, and INTENSITY.  
 selectObject: sound  
 pitch = To Pitch: pitch\_time\_step, minimum\_pitch, maximum\_pitch  
 selectObject: sound  
 intensity = To Intensity: maximum\_intensity, int\_time\_step  
 selectObject: sound  
 formant = To Formant (burg): 0, maximum\_number\_of\_formants, maximum\_formant, window\_length, preemphasis\_from  
 selectObject: textgrid  
 n = Get number of intervals: get\_data\_from\_tier  
 for i to n  
 interval\_label$ = Get label of interval: get\_data\_from\_tier, i  
 if interval\_label$ <> ""  
 t1 = Get starting point: get\_data\_from\_tier, i  
 t1\_klatt = t1-silence\_duration  
 t2 = Get end point: get\_data\_from\_tier, i  
 t2\_klatt = t2-silence\_duration  
 selectObject: pitch  
 f0 = Get mean: t1, t2, "Hertz"  
 f0 = round(f0)  
 f0$ = fixed$(f0,0)  
 selectObject: formant  
 f1 = Get mean: 1, t1, t2, "Hertz"  
 f2 = Get mean: 2, t1, t2, "Hertz"  
 f3 = Get mean: 3, t1, t2, "Hertz"  
 f4 = Get mean: 4, t1, t2, "Hertz"  
 f1$ = fixed$(f1,0)  
 f2$ = fixed$(f2,0)  
 f3$ = fixed$(f3,0)  
 f4$ = fixed$(f4,0)  
 selectObject: intensity  
 int = Get mean: t1, t2, "energy"  
 int$ = fixed$(int,0)  
  
 selectObject: klattGrid  
  
 Add pitch point: t1\_klatt, f0  
 Add voicing amplitude point: t1\_klatt, int  
  
 if interval\_label$ == "0"  
 Add frication amplitude point: t1\_klatt, 5  
 Add frication bypass point: 0.5, t1\_klatt  
 elsif interval\_label$ == "1"  
 Add frication amplitude point: t1\_klatt, 60  
 Add frication bypass point: 0.5, t1\_klatt  
 Add voicing amplitude point: t1\_klatt+(t1\_klatt \* 0.6), 0  
 else  
 exitScript: "Please place zeros and ones in the frication tier."  
 endif  
  
 Add oral formant frequency point: 1, t1\_klatt, f1  
 Add oral formant bandwidth point: 1, t1\_klatt, f1\*0.2  
  
 Add oral formant frequency point: 2, t1\_klatt, f2  
 Add oral formant bandwidth point: 2, t1\_klatt, f2\*0.2  
  
 Add oral formant frequency point: 3, t1\_klatt, f3  
 Add oral formant bandwidth point: 3, t1\_klatt, f3\*0.2  
  
 Add oral formant frequency point: 4, t1\_klatt, f4  
 Add oral formant bandwidth point: 4, t1\_klatt, f4\*0.2  
  
 selectObject: textgrid  
  
 endif  
 endfor   
selectObject: sound, textgrid

**(ii) Visualization Script**

#############################################################################  
#  
# analysis.py  
#  
# Desc: A script meant for visualizing audio data from .wav files  
#  
# Pre: sounds labeled original.wav and automated.wav exist  
# in the same folder as this .py script  
#  
# post: two .png format images are generated, one showing the waveforms  
# and the other showing the spectrograms for each sound  
# stacked vertically  
#  
#############################################################################  
  
import parselmouth as pm  
import numpy as np  
import matplotlib.pyplot as plt  
import os.path as path  
  
def generate\_waveforms(orig: str, auto: str):  
 sounds = [pm.Sound(orig),  
 pm.Sound(auto)]  
 plt.figure(figsize=(38.4, 21.6))  
  
 for i, snd in enumerate(sounds):  
 plt.subplot(3, 1, i+1)  
 plt.plot(snd.xs(), snd.values.T)  
 plt.xlim([snd.xmin, snd.xmax])  
 if i == 0:  
 plt.title(orig.replace(".wav", ""), fontsize=40)  
 else:  
 plt.title(auto.replace(".wav", ""), fontsize=40)  
 plt.xlabel("time [s]", fontsize=40)  
 plt.xticks(fontsize=20)  
 plt.ylabel("amplitude [Hz]", fontsize=40)  
 plt.yticks(fontsize=20)  
 plt.tight\_layout()  
 plt.savefig("waveforms.png", bbox\_inches='tight')  
 return True

def generate\_spectrograms(orig: str, auto: str, d\_r = 70):  
 sounds = [pm.Sound(orig),  
 pm.Sound(auto)]  
  
 plt.figure(figsize=(38.4, 21.6))  
  
 for i, snd in enumerate(sounds):  
 plt.subplot(3, 1, i+1)  
 spect = sounds[i].to\_spectrogram()  
 X, Y = spect.x\_grid(), spect.y\_grid()  
 sg\_db = 10 \* np.log10(spect.values)  
 plt.pcolormesh(X, Y, sg\_db, vmin=sg\_db.max() - d\_r, label="spect", cmap='magma', alpha=0.7)  
 plt.ylim([spect.ymin, 5000])  
 if i == 0:  
 plt.title(orig.replace(".wav", ""), fontsize=40)  
 else:  
 plt.title(auto.replace(".wav", ""), fontsize=40)  
 plt.xlabel("time [s]", fontsize=40)  
 plt.xticks(fontsize=20)  
 plt.ylabel("frequency [Hz]", fontsize=40)  
 plt.yticks(fontsize=20)  
 # end for  
 plt.tight\_layout()  
 plt.savefig("spectrograms.png", bbox\_inches='tight')  
  
def main():  
 orig = "original.wav"  
 auto = "automated.wav"  
  
 exist = path.exists(orig) \  
 and path.exists(auto)  
 corr\_format = orig.endswith(".wav") \  
 and

auto.endswith(".wav")  
  
 if exist and corr\_format:  
 generate\_spectrograms(orig, auto)  
 generate\_waveforms(orig, auto)  
 return True  
 else:  
 print("Files not found.\n"  
 "Make sure that all three wave files below exist in the source directory:\n"  
 "original.wav\n"  
 "automatic.wav\n"  
 "manual.wav\n"  
 "\n"  
 "Exiting...")  
 return False  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Complete project files available at:   
https://github.com/jakadake/CSCI475-Final-Project