MATLAB Speech Synthesizer

Final Report

Lachlan Dow

2021

**Abstract**

This paper will describe design, development and testing of a formant speech synthesizer built in MATLAB . The process took place between the 28th of October 2020 and the 27th of April 2021. The design process took place from the 28th of October to the 19 of February 2021 the implementation took place from the 20th of October to the 30th of March and the testing took place between the 2nd of April and the 24th of April.

**Background-**

People for hundreds of years have been trying to mechanically produce human speech, early models created by Christian Kratzenstein in 1779 (History and Development of Speech Synthesis, 2021) generated vowels by constructing models lie the human vocal tract as mentioned above by vibrating reeds like musical instruments.

Diagram

Description automatically generated

Figure . Katzenstein’s resonators (History and Development of Speech Synthesis, 2021)

In the subsequent century multiple models for mechanical speech synthesis were made

A large amount of work and research has gone into the field of speech synthesis through people using mechanical devices, whether in the form of “speaking heads” in 1003 AD or in 1779 when a Danish scientist-built models of the human vocal tract that could produce 5 long vowel sounds.

It was not until the late 1950s that early speech synthesis through electronic devices was created. It would not be another decade until the first full text to speech synthesis system was created in 1961 by John Larry Kelly.

There are many ways of approaching speech synthesis – these include concatenative synthesis -which include unit selection synthesis, diphone synthesis, domain specific synthesis, articulatory synthesis, HMM-based synthesis, sinewave synthesis, and formant synthesis. Due to the scope of this report, the only examples of these that will be explored in more detail is concatenative synthesis (including a couple of the sub-forms) and Formant synthesis. Information on the other forms can be found in ((Speech synthesis, 2021))

Concatenative synthesis consists of stringing together segments of recorded speech to produce the synthesis of inputted text. This form of synthesis is known to produce the most natural sounding speech, but because of the variances in how people speak, and the way segments of speech are extracted, there can be glitches when piecing together the different sections. This model of synthesis can be performed in multiple ways mentioned below:

* Unit selections synthesis, which uses a large database of recorded speech when each recorded utterance is segmented in different ways using a specially designed speech recognition system. These segments are indexed based off defining factors e.g., fundamental frequency, duration etc.
* Diphone synthesis, which uses a database containing all the diphones occurring in a language. Only one example of each diphone is contained in the speech database, then at runtime the prosody of a sentence is superimposed on the units, usually through a linear predictive coder (LPC).
* Domain-specific synthesis, which concatenates pre-recorded words and phrases to create complete utterances, is generally only used in speech variance as it will be very limited, for example a train announcement system or weather reports.

Formant synthesis on the other hand, does not use human speech samples but instead is created using a model to produce speech sounds. Different variables such as fundamental frequency and voicing noise levels are changed over time to create a waveform that represents speech. Formant based synthesis systems generally generate robotic sounding speech, for example the system that Stephen Hawking used. This results in the speech never really being mistakeable for human speech, but because of the way these synthesizers are made, they are very intelligible consistently without the glitches found in concatenative systems. Formant systems also have the advantage of being able to produce a wide variety of prosodies and intonations to be outputted, allowing them to produce questions, statements, emotions, and many voice tones.

**Design –**

The design process began with investigation into speech synthesizers, their history, and current state of the art synthesizers. This stage uncovered multiple different types of synthesis, namely:

* Unit selections synthesis, which uses a large database of recorded speech when each recorded utterance is segmented in different ways using a specially designed speech recognition system. These segments are indexed based off defining factors e.g., fundamental frequency, duration etc.
* Diphone synthesis, which uses a database containing all the diphones occurring in a language. Only one example of each diphone is contained in the speech database, then at runtime the prosody of a sentence is superimposed on the units, usually through a linear predictive coder (LPC).
* Domain-specific synthesis, which concatenates pre-recorded words and phrases to create complete utterances, is generally only used in speech variance as it will be very limited, for example a train announcement system or weather reports.
* Formant synthesis, this method uses an electronic speech production model where different parameters are varied to create speech sounds, these values can then be interpolated to create diphthongs or stopped to create plosive bursts. This method is said to create a sound that is very easily understood but sounds very robotic.

Until recently, this was the best option for synthesis due to its customisability and adaptability.

In the initial stages of design, creation of a unit selection synthesis system was investigated as a strong possible solution. In the experimentation stage, MATLAB was used to take speech segments and separate them into their respective sounds, and then reassemble them into the sounds that were needed to create other words.   
After more investigation into other options and reading speech synthesis and recognition (John and Wendy Holmes) and MITalk, this method had more information on the use of formant synthesis as a speech production method, and coupled with advice from the project advisor, a decision was made to transfer the project to a formant synthesizer.

Predesign for the MATLAB program was split into two parts-

1. The formant synthesizer, based off the parallel/cascade design(Klatt. D, 1980)
2. The text pre-processor for input into the synthesizer

**The Formant Synthesizer – Design and Development**

The design of the formant synthesizer was adapted from the design found in the paper by Klatt himself and also other adapted versions that were more suited for modern systems, most notably the system developed by d'Heureuse (2019). The synthesizer was **Diagram

Description automatically generated**broken down into different sections as objects in Matlab, and each object would have step functions to replicate a signal passing through them and the effect that it would have on that signal.

Figure 2. Synthesizer class diagram

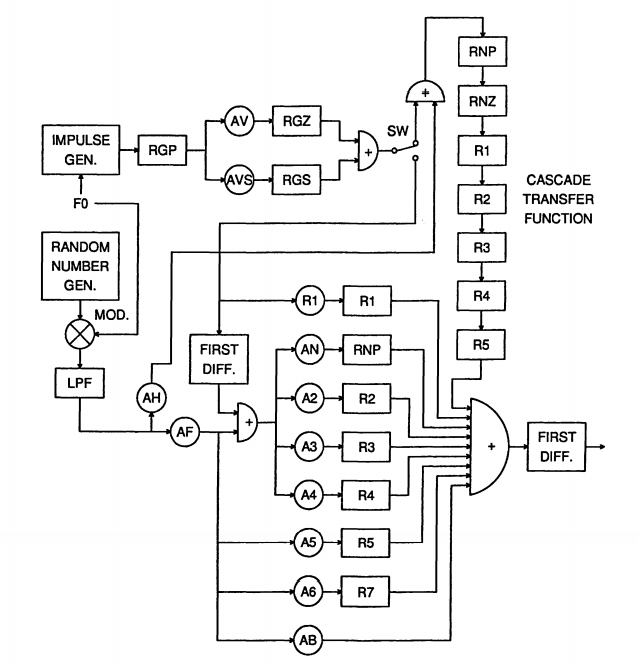


Figure 3. Block diagram of Klatt cascade/parallel formant synthesizer.

Figure 2 shows the model of the Klatt formant synthesizer proposed in his paper (Klatt, D., 1980.)

Pictured in figure 1 is the class diagram for the synthesizer in Matlab. This has been built as an object-oriented application. The main function for this system is the generator object, which is responsible for creating the sound waveforms from the parameters inputted. The main function is the generateSound() function. The function takes a data frame as input as well as the main parameters for the synthesizer, and then will process the values for the sound through each branch to output an array of doubles that can be used to produce a sound using the sound values.

**Difficulties in creation –**

When constructing the synthesizer, a waterfall method of production was used. Generally, to have a system that produced any sound at all, the whole base structure of the synthesizer would be needed. Because of this method, after laying out the base structure of the synthesizer roughly 4 weeks was spent debugging; stepping through the program and ensuring variables were correct at each stage as well as fixing errors that were caused by putting all the different classes together. In the future, it would be good to construct a predefined testing plan for each function and class to ensure that they would work correctly with the synthesizer. This would positively impact the production of the generator class, as that was where a large portion of the errors were found.

**Pre-Processing unit**

The pre-processing unit was developed using an agile approach. Each part was developed and functional in a deployable way. This section of the system was broken down as:

1. Text Reformatting
2. Text-to-phoneme conversion
3. Phoneme-to-sound frame conversion
4. Sound duration computation

More detail on these sections is provided below.

**Text Reformatting**

In this section, numbers were broken down into their word equivalent with a series of functions and acronyms were translated into individual letter segments to get the right sound from the system, and some abbreviations were translated into their full words.

Pseudocode:

1. findAbbreviations()
2. split array into words
3. loop for each word
   1. use regular expression to find acronyms
   2. regular expression to check for numbers
   3. if the word is a acronym
      1. replace the acronym in the array
   4. end if
   5. if the word is a number
      1. replace the number with the word equivalent
   6. end if
4. end for loop
5. change all words into the lowercase versions for the next stage of pre-processing

**Text-to-Phoneme Conversion**

This function’s purpose was to break down a word into its different phoneme segments. In an overarching method, the function word2Phone will take a word input, a map of key-value pairs for the different letters, and their position in an array of letter-to-phoneme rules. The function then loops through every letter of the word, referencing the word conversion with the rule list, and creates a new array with the phoneme conversion.

Pseudocode:

1. while loop for every letter
   1. get index position of rule in array
   2. loop until match found
      1. extract the character matches for rule segment
      2. check if the segment matches the section of text from current letter
      3. if this section matches extract the conditional rules for match before letter rules
      4. check that the string preceding the rule set matches the extracted rules
      5. extract conditional rules after the matched letter set
      6. check the string after the rule set matches the extracted rules
      7. assign Boolean value to matchFound
      8. if match is found push phonemes equivalent from the rule to array of phonemes
      9. increase rule index value by one and return to start of array
   3. end loop
2. end while

**Phoneme-to-Sound Frame Conversion**

This function will convert the phonemes into their sound frame equivalents. The function will loop through the array of phonemes and then convert then use a map of the phoneme symbol to an index representing the sound frame for that phoneme.

Pseudocode:

1. loop for the length of the array of phonemes
2. if sound doesn’t contains dipthong
   1. if sound is a \_ for HH sound
      1. get next phoneme and convert to HH equivalent
      2. add sound to sound array
      3. move onto phoneme after adjusted one
   2. end else
      1. convert to sound
      2. move onto next phoneme
   3. end if
3. if sound does contain a diphthong
   1. split diphthong into segments
   2. convert phoneme into sound frames
   3. assign diphthong attribute to these sounds
   4. more onto next sound in the array
   5. end if
4. end loop

**Sound Duration Computation**

This function is used to calculate the duration of sounds and how to blend between them. The inputs for this function is a list of sound objects to be converted. The function will then process these sounds into 5ms sound frames from durations based of a map of phonemes and duration characteristics. And output these sounds as an array to be passed into the generateSound() function in the Generator class.

Pseudocode:

1. check if array length is one
2. if the array length is one
   1. assume that the item is a letter and adjust the interpolation accordingly
   2. function return
3. end if
4. loop for length of sound array
5. if sound is last in array or plosive
   1. generate sound frames with no blending
6. else
   1. generate frames for current sound frame generate blended sound for next frame
7. end if

After these functions are run the text is ready to be sent to the synthesizer.

Pictured in figure 3 is the diagram that was used to construct these functions and how thy fit together to be input into the synthesizer

**Functionality testing of the system-**

In order to test the different sections of the system it was required to use test cases

**Diagram

Description automatically generated**

Figure . Diagram of text pre-processing to then be passed to the synthesizer

**Personal reflection-**

The project has been challenging, largely because speech synthesis is such a broad subject, that knowing where to start was a huge task and it has taken a large amount of time feel like progress has been heading in the right direction.

Since a great amount of time was lost due to the initial focus on the diphone synthesizer, work on the redesign and production of a formant synthesizer will take time but progress has been quick and testing has been positive.

**Plans for the Remainder of the Project**

Finishing the construction of the Klatt formant synthesizer in MATLAB, the main building

blocks of this system are in place but need to be combined into a fully functioning system.

Creating the input text pre-processor:  
Input text will need to be processed so that it can be used by the synthesizer. There are 39 parameter values, and on any given step, 20 may need to be changed to produce the formant. The values for this will need to be stored and referenced when required, based on the pre-processed data obtained from the text.

Going forward, an interface will also be constructed for any user to use the synthesizer. Designs for this will be made and then implemented in MATLAB.

Testing of the system will also need to be conducted to ensure that the speech emitted from the system is understandable. These tests include but are not limited to – MOS, Rhyme, AB test based on testing standards document (2021).

A progress plan has also been created to approach the tasks that need to be completed to ensure that the project is delivered on time:

Figure : Gantt Chart for the remainder of the project

Working Klatt formant synthesizer – 12/2/21

Text Pre-processor – 26/2/2021

Creation of the user interface linking of Text pre-processor and formant synthesizer– 5/3/2021.

Testing and refinements – 15/4/2021

Report finalisation- 27/4/2021

With this schedule in mind the creation and completion of the synthesizer should be completed on time with all the main features mentioned before implemented.

**References**

Cs.mcgill.ca. 2021. *Speech Synthesis*. [online] Available at: <https://www.cs.mcgill.ca/~rwest/wikispeedia/wpcd/wp/s/Speech\_synthesis.htm#:~:text=The%20first%20computer%2Dbased%20speech,the%20history%20of%20Bell%20Labs.> [Accessed 26 January 2021].

Klatt, D., 1980. *Software For A Cascade/Parallel Formant Synthesizer*. [online] Fon.hum.uva.nl. Available at: <https://www.fon.hum.uva.nl/david/ma\_ssp/doc/Klatt-1980-JAS000971.pdf> [Accessed 20 January 2021].

Wiki.inf.ed.ac.uk. 2021. [online] Available at: <https://wiki.inf.ed.ac.uk/twiki/pub/CSTR/Speak14To15/evaluation.pdf> [Accessed 27 January 2021].

Jonathan Allen, M. Sharon Hunnicutt, Dennis H. Klatt, Robert C. Armstrong, and David B. Pisoni. 1987. *From text to speech: the MITalk system*. Cambridge University Press, USA.

d'Heureuse, Christian. "Chdh/Klatt-Syn". *Github*, 2019, [https://github.com/chdh/klatt-syn. Accessed 4 Feb 2021](https://github.com/chdh/klatt-syn.%20Accessed%204%20Feb%202021).

Research.spa.aalto.fi. 2021. *History and Development of Speech Synthesis*. [online] Available at: <http://research.spa.aalto.fi/publications/theses/lemmetty\_mst/chap2.html> [Accessed 12 April 2021].

|  |
| --- |
| . |
|  |