Professional Bachelor Applied Information Science

Teach Robots to speak: Text 2 Speech Solutions for Robotics

Kenan Ekici

Promoters:

Tim Dupont	PXL AI & Robotics Lab
Niek Vandael	PXL Smart ICT

Professional Bachelor Applied Computer Science

Teach Robots to speak: Text 2 Speech Solutions for Robotics

Kenan Ekici

Promoters:

Tim Dupont	PXL AI & Robotics Lab
Niek Vandael	PXL Smart ICT

Bachelor paper Academic year 2017-2018

Acknowledgements

First of all, I would like to thank my promotor Mr. Tim Dupont. He has generously taken the time to help and support me through frustrating parts of my internship. Being in the presence of his expertise and passion for research and continuous improvement was a great advantage. My company promoter Mr. Niek Vandael has also helped to shape the final outcome of my internship. The business expertise that we have received from Niek is much appreciated and I want to thank him for being a part of this journey. I would also like to thank Mr. Steven Palmaers, the head of PXL Smart ICT, who has provided us the opportunity to conduct this research in the first place.

Not to mention my colleagues and fellow interns, Robin Goos, Sinasi Yilmaz, Niels Debrier and Toon Elewaut. I will certainly miss working and collaborating them and I want to thank them for helping and supporting me in times of need. Another caring words go out for two researchers of PXL Smart ICT, Maarten Bloemen and Arno Barzan. Their expertise and advice is appreciated as well. I would like to thank Ms. Greta Poelmans and Ms. Nathalie Fuchs for the coffee supply during the internship. It was definitely necessary. The countless teachers and junior colleagues that may or may not have participated the experiment are also appreciated for their advice, opinion and for representing a respectful working environment. Last but not least, I would like to thank my parents, family and close friends for always being there for me and showing me support.

Abstract

Robots have been given the ability to speak using speech synthesis technology commonly known as Text-to-Speech. Traditionally, these systems sounded rather robotic or unpleasant to the human ear. In his work, P. Taylor describes that the naturalness and intelligibility of the synthesized speech can be evaluated. Using certain evaluation techniques, an experiment on modern day systems can be conducted as a way to find the most intelligible, natural sounding, and practical Text-to-Speech system. Former projects within PXL-IT have described the Pepper humanoid robot's speaking ability to be lacking in naturalness. The findings of a comparative research can be implemented as an improvement of Pepper's Text-to-Speech ability.

The focus during the internship lies on implementing generative solutions. WaveNet, by the Google company DeepMind, is an example of a generative solution that uses deep neural networks. In the paper written by Aäron Van den Oord, WaveNet has been rated by human listeners to be more natural sounding than the best concatenative and parametric systems. Alternative papers and solutions such as API's and SDK's will be researched as well, for example Google Assistant, which can be used to improve Pepper's entire speaking and understanding ability. Because the field of Text-to-Speech practices more than one technology, other studies such as Artificial Intelligence and linguistics are included to achieve a complete understanding of the field of Text-to-Speech.

Table of contents

List of figures

List of tables

Code listings

List of abbreviations

TTS	Text-to-Speech
NLP	Natural Language Processing
POS	Part of Speech
GPS	Global Positioning System
MOS	Mean Opinion Score
SDK	Software Development Kit
ICT	Information and Communication Technology
SLAM	Simultaneous Localization and Mapping
SSML	Speech Synthesis Markup Language
HMM	Hidden Markov Model
LSTM	Long-Short-Term-Memory
RNN	Recurrent Neural Network
kHz	kilo Hertz
API	Application Programming Interface
ROS	Robot Operating System
DNN	Deep Neural Network
AI	Artificial Intelligence
REST	Representational state transfer
CMU	Carnegie Mellon University
RAM	Random Access Memory
URL	Uniform Resource Locator
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
AJAX	Asynchronous Javascript and XML
XML	Extensible Markup Language
JSON	Javascript Object Notation
LTS	Long Term Support

Introduction

This thesis has been greatly inspired by Paul Taylor's book on Text-to-Speech while also trying to present the technology in its state-of-the-art form. By introducing linguistics and technologies in regards to Text-to-Speech, any assumptions of background knowledge in a later stage of the thesis are eliminated. It can therefore serve as a starting point of Text-to-Speech for beginners or as a guideline for developers to solve a mentioned Text-to-Speech problem during the implementation phase.

First, the literature study will introduce Text-to-Speech. It describes what speech synthesis is and how Text-to-Speech serves its purpose. Afterwards, an introduction to linguistics is given. Paul Taylor describes this as a way to develop a model for understanding linguistics in relation to Text-to-Speech. The study of linguistics mostly focuses on the differences between the text and speech representation and how to interpret both in a given context. It is important for readers to understand fundamental terms and concepts within language and communication before diving into the details of Text-to-Speech.

An introduction of important pipelines and processes within Text-to-Speech follows afterwards. As opposed to discussing an end-to-end solution, the mentioned processes should provide a step-by-step overview of how the text representation is transformed to speech samples. However, this introduction also consists of real world end-to-end examples and papers that are based on statistics, signal processing, neural networks or a combination of these. The focus lies on generative solutions where speech is synthesized with what is referred to as "Deep Learning" techniques in the field of Artificial Intelligence. These are only mentioned and discussed in relation to the level of Artificial Intelligence that is presented during the thesis. For further consolidation, readers should refer to other mentioned works that provide a much deeper overview of Artificial Intelligence.

By presenting solutions, the thesis also mitigates a given Text-to-Speech problem. These solutions are alternative Text-to-Speech systems that will be evaluated during a comparative study. This study is conducted using measurements that are collected through research and experimentation. The evaluation of the solutions during the comparative study, is based on techniques that are mentioned during the study of the Text-to-Speech. After the comparative study, a certain amount of solutions are selected for implementation which are based on the findings of this study. The implementation phase describes how a certain solution can be implemented and which software or hardware is used in the process. In the end, a Proof of Concept is proposed for the Text-to-Speech problem and a given client.

I. Traineeship report

1 About the company

PXL Smart ICT is a research facility that is located within the campus of the PXL University College at the Elfde Linie in Hasselt. An entire building is dedicated to this department which is the center of expertise and research of PXL. The head of Smart ICT is Mr. Steven Palmaers, a researcher and lecturer of the PXL college. Steven and his team are responsible for a great amount of developments and research of emerging technologies in the field of IT and ICT. The mission and scope of these researchers are vast, from the Internet of Things to Virtual Reality. PXL Smart ICT also offers research consultancy of multiple IT services to companies. [1]

PXL SMART ICT

Figure 1: Logo of PXL Smart ICT. [2]

With the anticipation of Artificial Intelligence and Robotics, a new division has been established under PXL Smart ICT named PXL AI & Robotics Lab. This division as shown in Figure 2 and Figure 3, is located at iSpace on the Corda Campus and is led by Mr. Tim Dupont. Since the beginning of PXL Robotics Lab, Mr. Dupont has been collaborating with students in order to research and develop projects within Robotics and Artificial Intelligence. Because of this, many students were able to work with state-of-the-art robotics equipment and hardware for machine or deep learning purposes. [3]

Figure 2: The Corda Campus where PXL AI & Robotics Lab is located. [4]

Tim is a researcher and founder of many successful projects. In the year of 2018, several Robotics projects had been unearthed. One of them was Seek and Report, where students were challenged to autonomously navigate a Turtlebot Robot on a certain area such as the iSpace. The SLAM algorithm was particularly used for this purpose which allowed the robot to map and localize itself in a real or simulated environment. With a face recognition tool created by Maarten Bloemen, a former intern, the robot was then demanded to autonomously find a specific person within the mapped area. Some of the student participants that are visible on Figure 3, have continued their work with Artificial Intelligence and Robotics as interns at PXL AI & Robotics Lab.

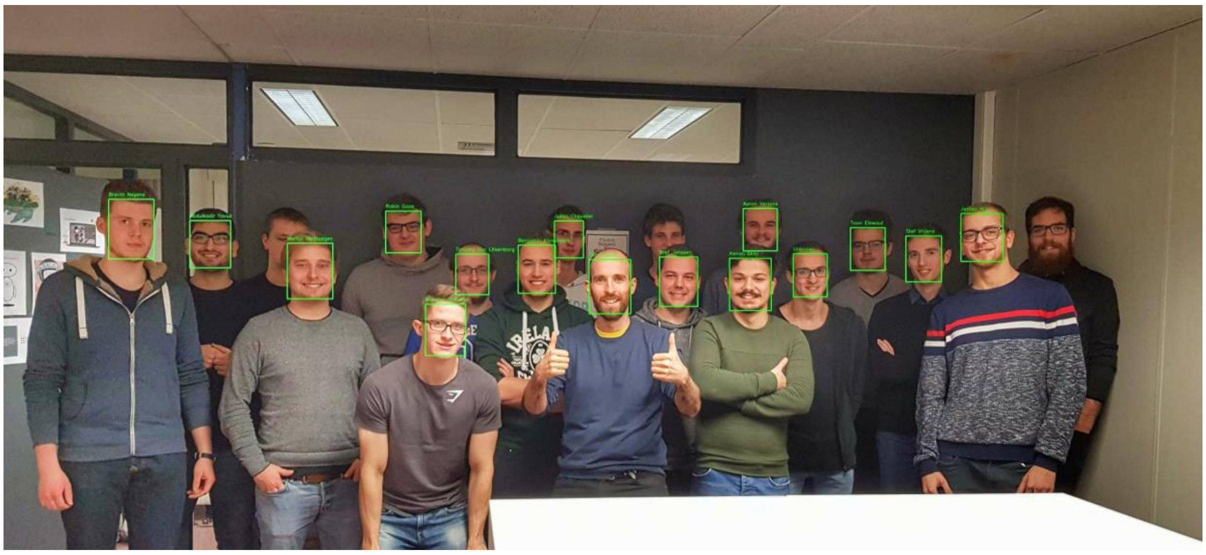


Figure 3: Tim Dupont and students during the IT project. [5]

II. Research topic

1 Speech synthesis

1.1 Speech technologies

It is not a surprise that speech and language technologies have had an huge impact on society. Mankind has been trying and has successfully achieved to transmit or generate speech in any way possible. Although its impact has not always been obvious, these technologies have massively influenced war, business and daily lives. [6] Starting as early as the revolutionary telephone invented by Alexander Graham Bell in 1876, and devices such as radio's and recorders. [7] These are all inventions made possible with speech technology. Another invention made history in the 1930's. The "Voder" founded by Homer Dudley, was able to generate speech by using a "Vocoder" to extract data from natural speech and the Voder itself to manipulate the data into actual speech. Using a keyboard as shown in Figure 4 and Figure 5, the Voder could only be controlled by a skilled operator. It was demonstrated at the 1940 New York World's Fair where it became very popular. [6] According to the press in 1939, "The Voder was one of several voice technologies to have a significant impact on radio and film production during the 1940s… ". [8]

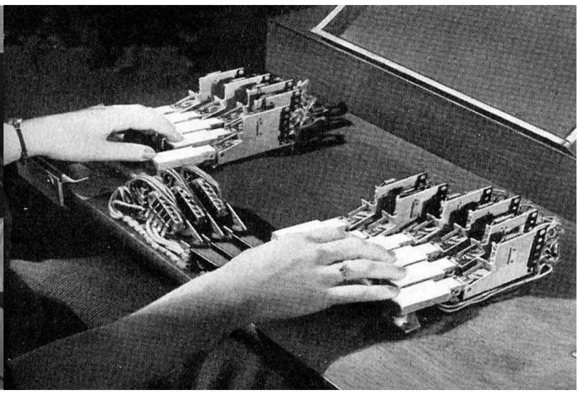


Figure 4: (right) An operator controlling the Voder. [9]

Figure 5: (left) The mechanical keyboard used to operate the Voder. [9]

All of these devices mentioned above can be referred to as "passive" forms of speech technology because its intention is to convert speech into a form that can be stored or transmitted only to be heard at a later time or different location. In other words, the technology itself is not responsible for the "interpretation or original generation of the speech or text". The systems that are actually responsible for this intent are called "active" forms of speech technology. This field of technology also contains speech recognition or voice recognition, speech synthesis, and speaker recognition. These domains enable computers to read, speak and distinguish a given language or person. The focus during this thesis lies on the field of speech synthesis which is the artificial generation of speech. Of course, forms like voice recognition can and will be mentioned in this thesis as well as a possible combination with speech synthesis. [6]

In science fiction films, robots or other futuristic devices can be seen talking effortlessly. They may not sound exactly like humans, but the idea of generating speech has been around nevertheless. In the film 2001: A Space Odyssey, a computer named "HAL" makes an attempt to interact in a humanly fashioned manner, almost indistinguishable from an actual person. This film, like many others, has been deemed successful in predicting the advancements of speech technology considering it was produced in 1968. [10]

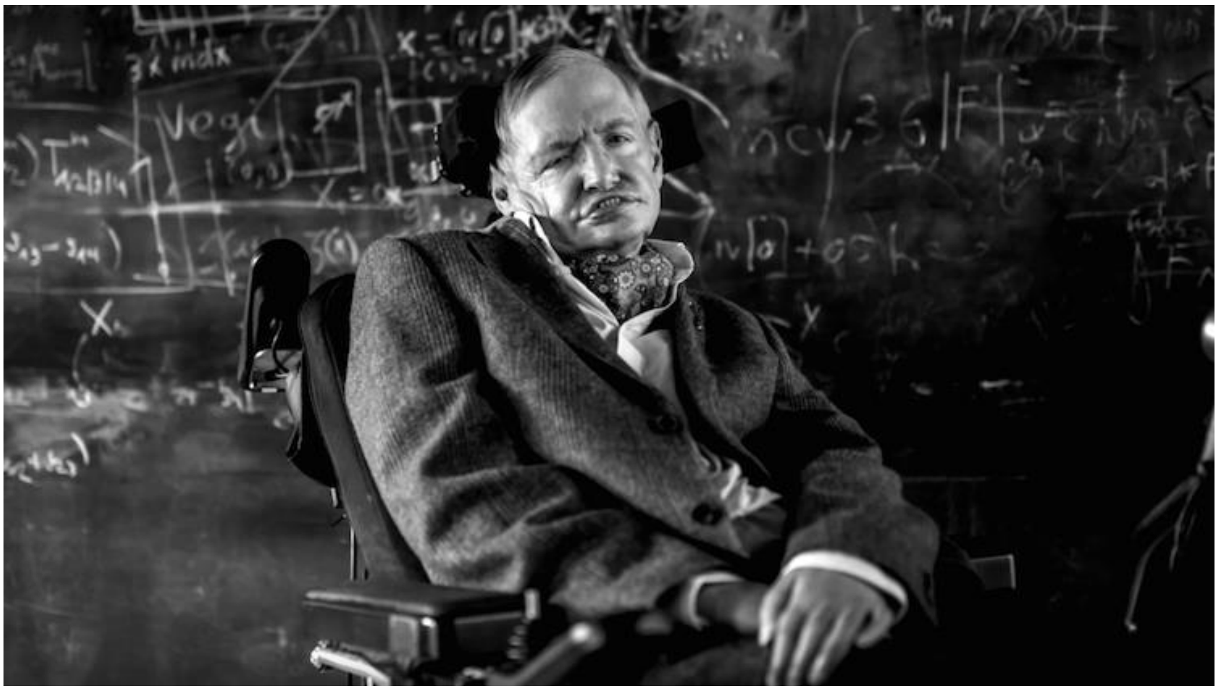


Figure 6: Stephen Hawking (1942 - 2018). [11]

Apart from its entertaining factor, there is also social value attached to speech synthesis technology. For example, allowing visually or hearing impaired people to "speak" without being able to generate speech themselves. Stephen Hawking, as shown in Figure 6, was best known more than anyone else contributing to the exposure of speech synthesis. The technology behind the different forms of speech synthesis is evolving in a very fast pace. People no longer have to worry about gender or personally tied speech synthesis with current adaptive methods allowing a quick generation of personalized speech. Changing the language of the generated speech could be a matter of training with the right data. [10]

When going through a typical day, speech synthesis is almost impossible not to encounter. From a speaking alarm clock, to a speaking GPS device in your car or mobile phone. Of course, depending on the application, different systems of speech synthesis can be found. Almost every mobile phone nowadays has some kind of an assistant to help you with, such as Google Assistant, Siri or Alexa. Take Google Duplex for example, which can start a phone conversation with an actual person. [12] And as technology advances, it becomes harder to distinguish the generated speech from actual human speech. Because of this advancement, systems no longer have to sound mechanical and monotone. Although sometimes it is easy to tell whether the speech was recorded or artificially generated, meaning that there is still progress to be made in the field of speech synthesis. [13]

1.2 Text-to-Speech

Text-to-Speech is a form of speech synthesis which uses text in order to synthesize speech. A set of voice recordings with associated transcripts form the basic requirement of synthesizing speech with a Text-to-Speech system. [14] As mentioned before, these systems are featured in a large range of applications. From mainstream use to a necessary feature for people with impairments. However, the reaction to typical Text-to-Speech systems has not always been equally positive apart from the impaired people who sometimes have no other choice. On first reaction, there might be some kind of surprise factor but in general, the anticipation definitely wears off after some time. [10] Especially if the generated speech is of lesser quality, resulting in monotone and robotic sentences forced after one another. Solving this matter required years of research facing all kind of technological and linguistic issues. The issue that strangled researchers for a long time was the conversion from a given text to a decent sequence of phonemes in combination with the associated acoustic information such as duration, amplitude, emphasis, pitch, pauses, and other acoustic features of speech. This study to convert text to a dictionary of acoustic and linguistic features, has almost taken an equal amount of time as the study to generate actual speech. [10]

There are many techniques in order to become Text-to-Speech. The traditional approach is by concatenating small samples of prerecorded speech that are selected based on the units of an input text. This approach is called the concatenative approach or more specifically “Unit Selection”. Unfortunately, the synthesized speech is not that impressive and can sound rather jumpy as further research will suggest. Is it really that important for the generated speech to sound like a human? After having to listen to robotic voices generated by systems, people can often feel uncomfortable and even irritated. People seem to care more about the way something is said, rather than the exact words, especially in commercial and mainstream use. Because the evaluation of the synthesized speech is rather subjective, it is evaluated using Mean Opinion Scores or MOS for short, and several other experiments. [10]

Before diving in to Text-to-Speech systems as listed on the table below, the fundamentals of linguistics must be covered as well as traditional and modern Text-to-Speech approaches. This is necessary because Text-to-Speech, as opposed to recording a simple message to play it back later, is a more advanced feature. It should for once be capable of playing back text that has not previously been recorded. [10]

1984	MacInTalk by Apple. [13]
2001	Natural Voices by AT&T. [13]
2002	Release of NeoSpeech, as used by Stephen Hawking. [15]
2013	Google releases Google Text-to-Speech or GTTS.
2016	DeepMind proposes WaveNet.
2016	Amazon releases Amazon Polly.
2017	IBM releases IBM Watson Text-to-Speech service.
2017	Deep Voice 3 proposed by Baidu Research.
2018	Google releases a Cloud Text-to-Speech service based on WaveNet.
2018	Google Duplex presented by Google. [12]

Table 1: A historical overview of Text-to-Speech applications.

2 Exploring linguistics

2.1 Introduction

Humans and animals use their senses to extract and exchange information. The process of interpreting information and exchanging it with other beings is called communication. In order to communicate, information has to be exchanged verbally. The ability to speak can be mimicked with technology which was earlier mentioned as speech synthesis. The study of Text-to-Speech within speech synthesis is interested in achieving a spoken signal from a written form such as text. This can be approached from different kinds of scientific aspects from one of which is linguistics. Studying the important parts of linguistics can help with developing a model for understanding problems and jargon within Text-to-Speech. [10]

2.2 Spoken and written forms

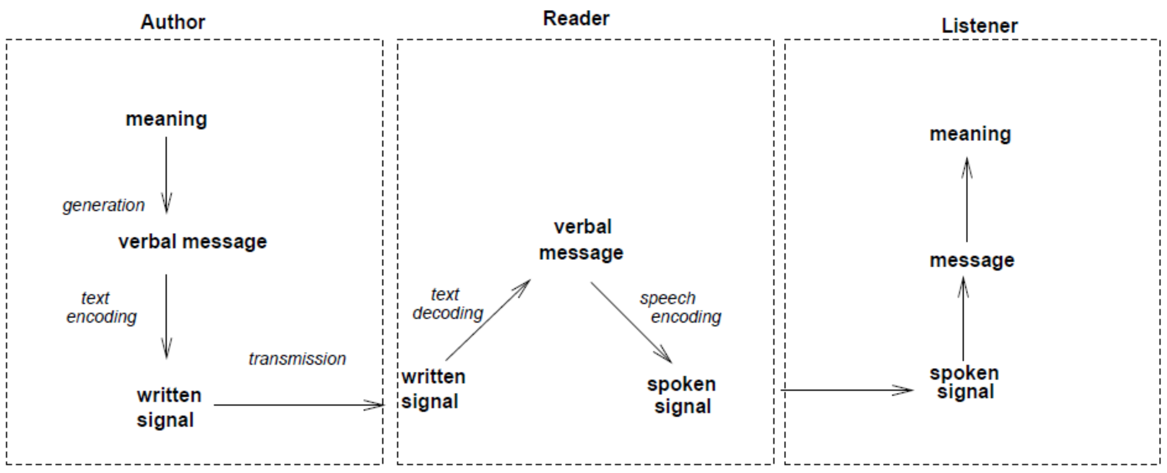


Figure 7: Process of reading a written form aloud to a listener. [10]

There is a difference between written and spoken forms of communication. A conversation, which is a spoken form of communication, can be considered as being spontaneous. As opposed to written communication, it requires no preparation. In a typical face to face conversation, the speaker can observe the reaction of what is being said. It allows the speaker or listener to change their behavior in response to the conversation. [10]

The scenario of Text-to-Speech can be considered as a combination of both communication forms. It represents a person who is reading aloud a form of written text as shown in Figure 7. The text may be written by the same or different person who may not be a part of the conversation. Nevertheless, the reader has to decode the written form of the author into a message. The message needs to be encoded into audio in the form of speech. The speech samples can then be exchanged with a listener, who at arrival decodes the speech into a message and extract the meaning. [10]

2.3 Speech encoding

The process of converting a message into signals is called speech encoding. In order to achieve this, a message has to be generated first. In Text-to-Speech, the message is provided in a written form as opposed to a conversation where it is generated spontaneously. Speech has two representations which are the message itself in a written form and a signal waveform. [10] Figure 8 shows the difference between the two representations in relation to the famous “Laurel vs. Yanny” debate.

What makes speech encoding such a difficult task is the fact that both representations are significantly different from each other. The written form is composed out of graphemes such as words. In order to encode it to a waveform, the grapheme model has to be converted to a phoneme model. The waveform representation can also contains other information that might not related to the message such as breathing, mood, speaker identity and emotion. Encoding such information from a written form is not an easy task. Needless to say, the speech encoding process is a difficult problem within Text-to-Speech and is tackled using several processing and encoding methods as described in a following chapter. [10]

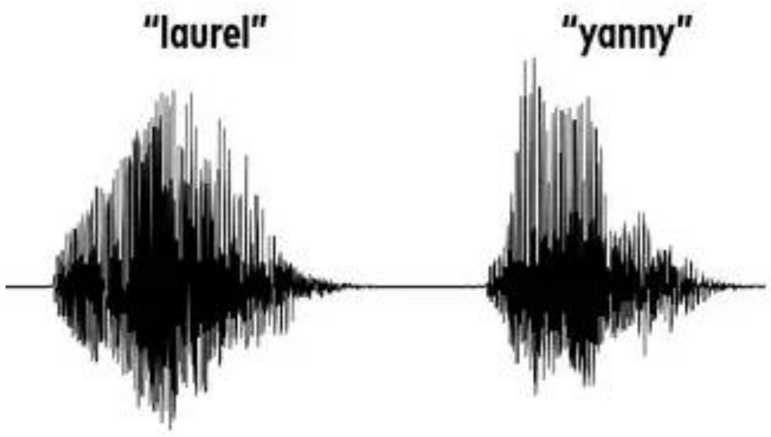


Figure 8: A written and waveform representation of “laurel” and “yanny”. [16]

2.4 Speech decoding

Speech decoding refers to converting a signal to a message which is the opposite of speech encoding. While encoding is a Text-to-Speech problem, speech decoding is tackled with technologies such as Speech Recognition. The challenge of speech decoding is the opposite of encoding which is extracting a small amount of information such as the message from a large amount of information being speech. A problem that is being faced with this task is ambiguity. It is hard to determine the exact message from a given speech because of its underlying factors and errors [10]

The solution to problems and certain matters within speech decoding, or more specifically Speech Recognition, have been briefly discussed and researched in “Teach robots to hear: Speech Recognition solutions for robotics” by Sinasi Yilmaz. Similar to this thesis, it proposes a certain software solution which combines both speech encoding and decoding technologies.

2.5 The verbal and prosodic component

Nonverbal communication refers to communicating and exchanging meaning without using words or any other linguistic structure for that matter. It is about exchanging messages using body language, tone and facial expressions. Verbal communication on the other hand, refers to the exchange of messages in the form of linguistical structures such as words. [17] These types of communication refer to important components in spoken language.

First off, the verbal component is responsible for arranging a set of words into sequences such as sentences. Then there is the prosodic component, referred to as prosody, which is responsible for expressing these sentences in a specific manner such as emotion or emphasis. Both these components can be studied separately but often interact with each other because they use the same signal as a representation, which is speech. If the sentence “I like that person” is expressed in a sarcastic tone, then the prosodic and verbal have interacted. In that case, the prosodic information likely wins. [10] Figure 9 presents both the verbal and prosodic components of a sample sentence in a tree like manner.

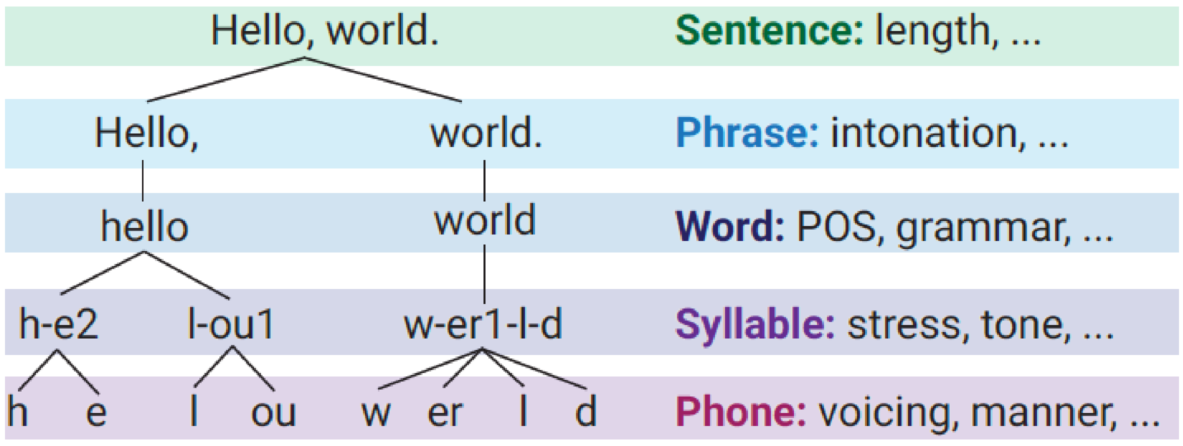


Figure 9: The structure of a sentence with the different verbal and prosodic components. [18]

2.5.1 The verbal component

The unit that is used within the verbal component is the phoneme. A phoneme is a set of sounds which combined with other phonemes, represents a word. These words can then be combined into sentences. The verbal component consists of these building blocks that construct words using phonemes, and sentences using words. The phonemes on their own do not have any meaning, they are simply meant to construct words. The constructed words do have a meaning. However, both the meaning and its associated form (sequence of phonemes) are arbitrary. The meaning and pronunciation of certain word needs to be learned in order to be understood and correctly pronounced. The arbitrary principle of the verbal component means that words with similar forms can mean entirely different things. For example, “pi” and “pie” sound similar but are completely different in meaning while “hound” and “dog” mean the same yet look very different. [10]

An endless amount of things can be said using the verbal component. This is known as the productive property of the verbal component. While some sentences exchanged during communication are conventional and reused such as “how are you doing?”, most of them are a unique combination of words. This is simply because of the great number of possible combinations that can be made using a vocabulary of words. This number is greater than the amount of galaxies in the observable universe. [10]

Another property within the verbal component is the discrete property. The previously mentioned phonemes and words are discrete because they form a finite set of units unlike speech, which is continuous because it moves in a certain timeline. In order to convert a set of phonemes into speech, which is yet another goal of Text-to-Speech, a discrete representation must be converted into a continuous one. This is a very complex process that also happens in reverse for Speech recognition. [10]

2.5.2 The prosodic component

The prosodic component also called prosody has two known properties, affective and augmentative. Affective prosody is used as a way to express primary emotions such as pain. This type of prosody is generally universal because it indicates the same meaning, in this case pain. Besides pain, other types of secondary emotions can be expressed as well such as sarcasm, question, etc., … However, these typically are language specific as opposed to primary emotion. Those yield the same arbitrary property as the verbal component because they need to be learned by the speaker. [10]

In English for example, an ascending pitch at the end of a sentence can indicate a question. In other languages such as Hungarian however, certain questions have a descending pitch. The main argument in the field of prosody is whether it is actually universal and non-arbitrary, or more like the arbitrary verbal component with rules and such. [10]

Then there is augmentative prosody meant to augment the verbal component. In a sentence, the speaker can choose to pause during the sentence and therefore change the meaning of it and bias the choice of interpretation of the listener. Aside from this phrasing purpose, this type of prosody also helps to put emphasis on certain words in a sentence. It is called augmentative because it augments the verbal aspect and helps the listener with understanding the meaning and form of a speech. [10]

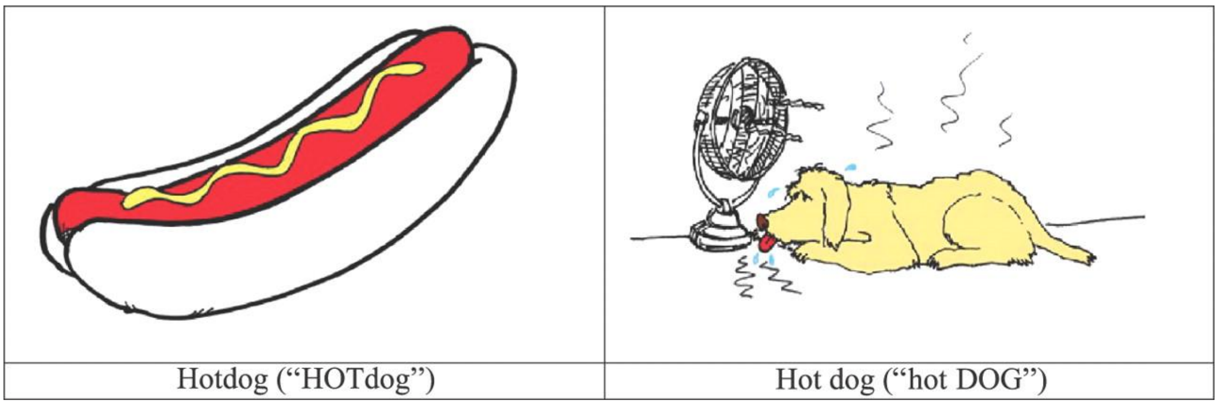


Figure 10: The meaning of “hotdog” changes according to its pronunciation. [10]

The difference between affective and augmentative prosody is that the latter does not generate any new meaning. It is only intended to help the existing meaning of the verbal component. Figure 10 shows an example of how augmentative prosody can be used to change meaning. An interesting point is the fact that nearly all different speakers will vary in the way they use augmentative prosody. Some speakers will pause in a specific sentence while others may not. This is mostly because speakers change the use of this augmentative prosody depending on the situation. The use of this kind of prosody can increase if the speaker fears the message might be ambiguous. In that case augmentative prosody is used in occasion. This augmentative property makes it impossible to predict it beforehand. However, the intensity of prosody can be predicted based on an intensity scale. A speaker is said to "speak with null or neutral prosody" if there is no prosody used at all. [10]

2.6 Linguistic levels

There are different kinds of levels that determine the verbal component. As previously mentioned, the first level consists of the primary words and phonemes as building blocks. The secondary level as shown in Figure 11, introduces a different set of way to look at verbal language. [10]

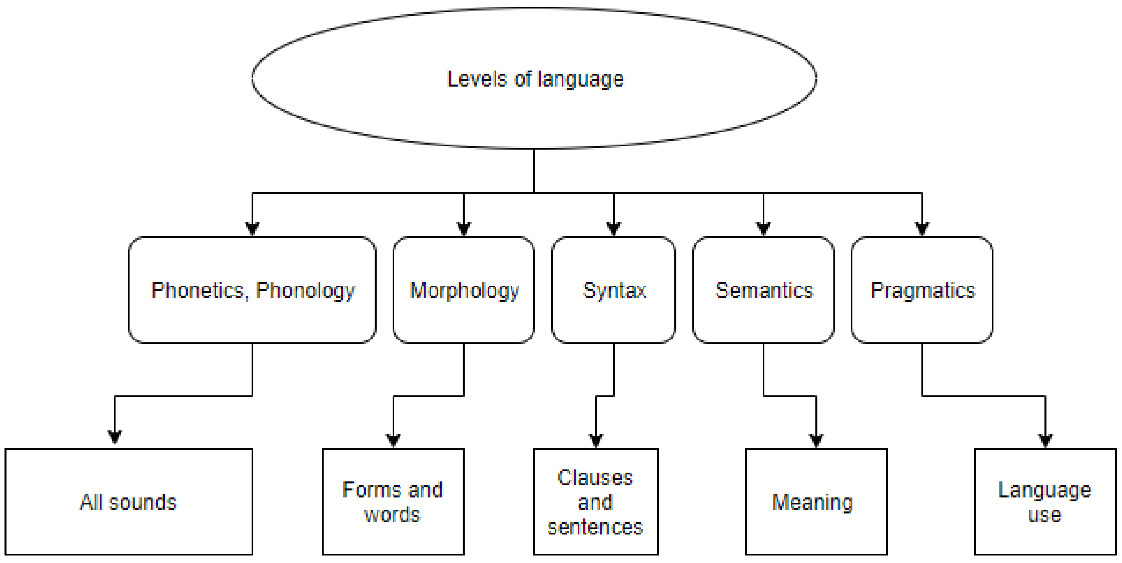


Figure 11: The different levels of language. [19]

First there are morphemes. These are the smallest units that words in a sentence are composed of and therefore cannot be further divided. The study of this is called morphology. [20] Next there is syntax, this is the study of the order of words in a sentence and its structure. Syntactic information intends to set the hierarchy of a given sentence into parts of speech such as nouns, verbs, etc., … and determine the phrase and sentence formation. It helps with understanding the grammatical connection between these units. [21] Then there are phonetics. This is the study of the acoustic representation of language which are signals of waveforms. [10]

Phonology is the study of phonemes and syllables that group these together. In general, it is concerned with everything that is associated with patterns of sound. While phonology and phonetics are related, they are not the same study. There are also semantics and pragmatics. Semantics concerns with the meaning of a verbal language. [10] It is interested in the intuition of words and phrases and their intentions at conscious and subconscious level. Semantics also looks at how certain words are related to each other such as synonyms, homonyms, etc., … and the relationship of sentences such as contradiction or paraphrasing. [22] Finally, pragmatics serves as a fallback for semantics, more specifically the use and meaning of language in general. [10]

2.7 Linguistic glossary

2.7.1 Orthography

Orthography defines the link between the symbols of the alphabet and the sounds of spoken language. Multiple speakers of a language can communicate by sharing the same orthographic written forms. It is a way to define the interaction between grapheme symbols such as words and the sound phonemes that they represent. Some languages can totally vary in orthographies. Take for example the word “goed” in Dutch and “good” in English. Knowing the orthographic representation of both help to determine that the letter “g” on each words has a different pronunciation. [23]

2.7.2 Corpora

Corpora is the plural of a corpus, which is a digital collection of a text in their written or spoken form. It is used to track and analyze the developments of language on their spelling, usage, trends, etc., … An example of a corpus is The Oxford English Corpus which contains over 2.5 billion words of 21st century English lexicon from magazines, newspapers or even blogs. It not only contains the English language from UK or the United States, but also from other parts of the world where English is used such as New Zealand or South Africa. [24]

3 Text-to-Speech

3.1 Architecture

Common Text-to-Speech systems mainly consist of two components: a linguistic analyzer which is called the front end, and a waveform generator which is often called the back end. The front end basically accepts any kind of input and turns this into a sequence of linguistic features. The back end takes this sequence and converts it into an appropriate waveform which then speech is generated of. [25]

This is how modern corpus-based systems generally turn text into speech. Corpus-based meaning they use a corpus as their training data for training both the front and back end components. [10] The architecture will generally be similar but can depend on the Text-to-Speech approach and the techniques (neural networks, decisions trees, other models) that are used within a certain system. [25] Figure 12 shows the architecture of a typical Text-to-Speech system.

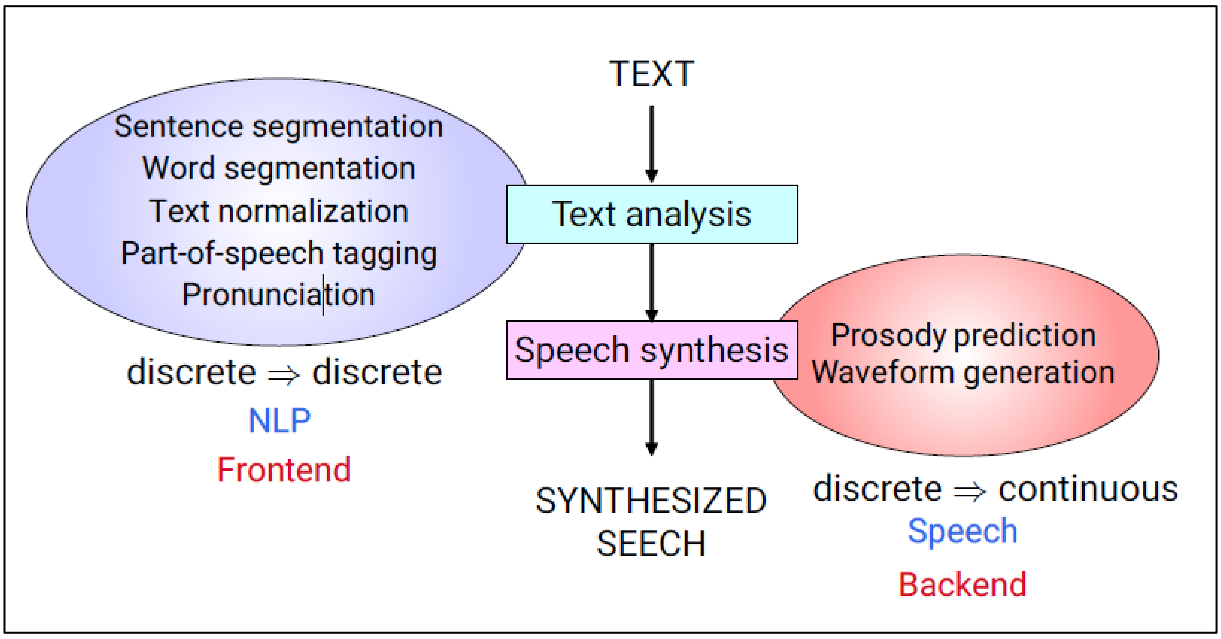


Figure 12: Architecture of common Text-to-Speech systems. [18]

3.2 Front end

The front end accepts plain text as input. It will parse the text through a pipeline in order to achieve a sequence of numbers which the back-end can use to generate speech. The process of converting this input into an appropriate sequence is called "feature extraction". Every type of Text-to-Speech system has its own method to achieve this. However, the most generic methods to achieve feature extraction are listed in the following subsections. [25]

3.2.1 Collecting input

The input text that will be used to generate speech from can be collected in multiple ways. One way is to use a terminal where users can input the text. A different way would be to send the input text to an API using a certain protocol. The method that will be used to provide the input text usually depends on the software environment of the Text-to-Speech system. The input text itself, which has been provided to the system, can consist of plain words in the form of sentences or more complex structures such as numbers and other symbols. In that case, these will be processed on a later stage. [26]

During this stage, several markup language tags such as Speech Synthesis Markup Language can be used to provide more information about the given text. The main reason for this is because it is difficult to achieve decent prosody on the generated speech. In other words, it is hard to express emotion or other effects with just plain text. That is why tags can be provided so that the front end can interpret and process this extra information in order to increase the accuracy of the generated speech. [26]

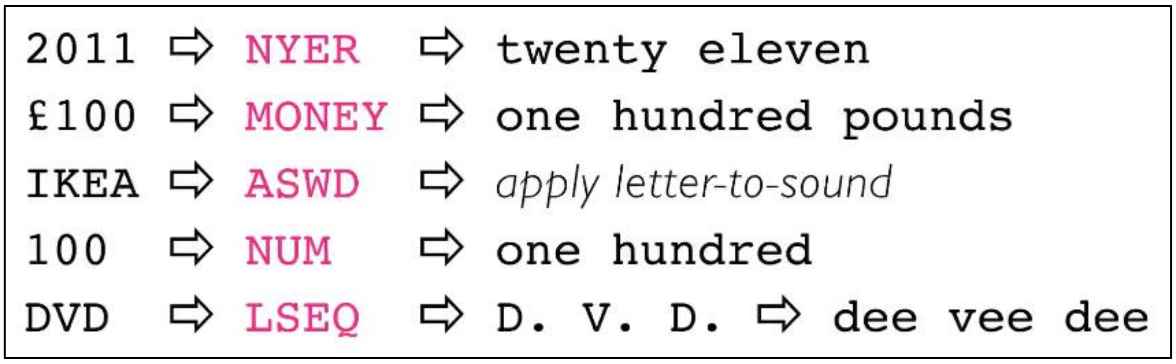
3.2.2 Segmentation

The input text that has been given can consist of anything. In order to ensure that the input eventually is composed out of sentences and words, segmentation for both structures needs to occur. By segmenting the text into sentences, it becomes easier to process by an algorithm. Each sentence can then be divided into tokens, such as words, in order to avoid analyzing a complete sentence for ambiguity or complexity. Both sentence and token segmentation can be tackled with algorithms while the approach itself is often "rule-based". The reason for this is because each language may have different rules for segmentation, while white spaces and punctuation marks are often used in English. [10]

3.2.3 Preprocessing

Once the input is segmented into tokens, it must be converted into a sequence of actual words which is required for a later stage. This process will be simple if tokens already consist of words. This is often not the case as these tokens can be complex. That is why this process is responsible for converting complex tokens into less ambiguous written out words. The tokens can consist of abbreviations, dates, numbers or any other structure that are not words. As shown in Figure 13, the preprocessor will turn the number "2011" to "twenty eleven" if it's a date or change "DVD" to "dee vee dee". [10]

This conversion can be approached with probabilistic statistic methods using Hidden Markov Models or neural networks. The complex tokens, which are composed of non-lexical items (as opposed to words which are lexical), should be able to parse through without any issues. In the worst case scenario, when the token is truly not processable, it should be spelled character per character. Again, the whole point of preprocessing is to turn any kind of token in to a word which can be used in the next stage. [25] [10].



```
2011  ⇨ NYER  ⇨ twenty eleven
£100  ⇨ MONEY ⇨ one hundred pounds
IKEA  ⇨ ASWD  ⇨ apply letter-to-sound
100   ⇨ NUM   ⇨ one hundred
DVD   ⇨ LSEQ  ⇨ D. V. D. ⇨ dee vee dee
```

Figure 13: Preprocessed characters into fully written out words. [25]

3.2.4 Part of Speech tagging

Having dealt with complex input and turned it into a sequence of words, there is one remaining problem and that is syntactic information. What about homographs? Homographs are words that are spelled the same but pronounced differently. Or even dates and numbers which also look similar and sometimes are pronounced different ways. [25]

In the front end, this syntactic information needs to be addressed as well. Because words in a later stage will be compared to a dictionary, they need to be tagged in order to be distinguished. This is achieved with Part-of-speech or POS tagging that can be approached with Hidden Markov Models or with neural networks. However, the data that is used for training such a model can be quiet expensive as it needs to be labeled. [25]. An example of POS tagging can be seen on Figure 14 where a sample sentence is divided into different parts of speech.

What about words that are not recognized or simply not included in the training phase and the lexicon of the tags? That is why POS guessers are used. By analyzing the features of a given word, such a system can figure out in what POS category it belongs. A popular strategy is called "ending guessing" which tokenizes a word with a POS tag entirely based on its ending characters. [27]

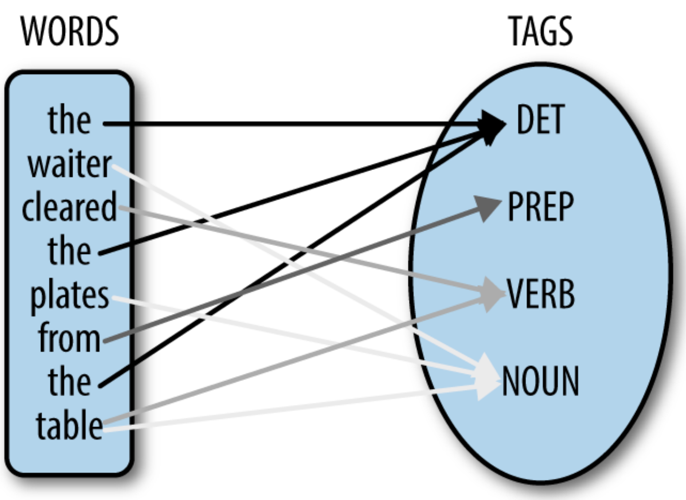


Figure 14: An example of POS tagging. [25]

3.2.5 Grapheme-to-phoneme

Once a sequence of words is acquired and all of the ambiguity is taken care of, the system has to figure out how each word is pronounced. This can be achieved by extracting a unit of the word, for example letters or syllables and determining its phonemes which indicate the pronunciation. As shown in Figure 15, "author" is represented as "ao-th-er" and "of" as "ah-f". This process of converting words to phonemes is called "grapheme-to-phoneme". [10]

The way this process approaches the conversion is by looking up the specific word in a dictionary and retrieving the phonemes based on each letter or syllable, depending on the system. This dictionary is a very expensive set of data because it requires a human expert to write a dictionary of lexical units and their phonetic transcript. [25] And when the word does not occur in the dictionary, because of a spelling error for example, the phonemes need to be predicted or determined by again, using a statistical model or a neural network. [10]

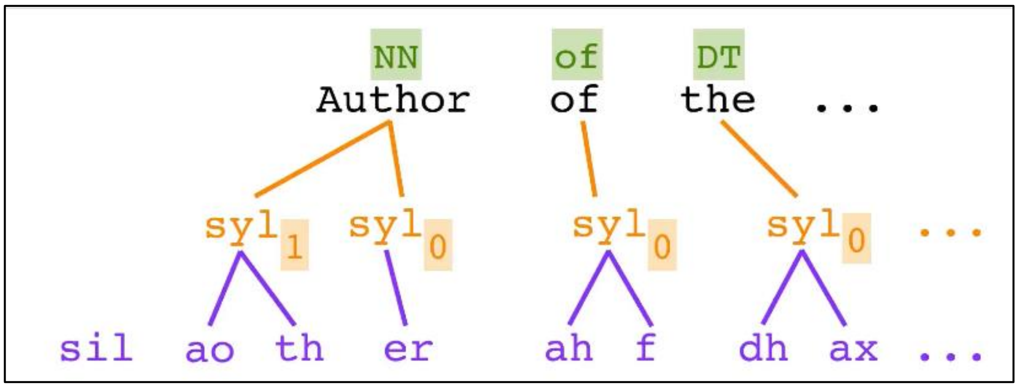


Figure 15: Example of words and their tree-like linguistic and phonetic structure. [25]

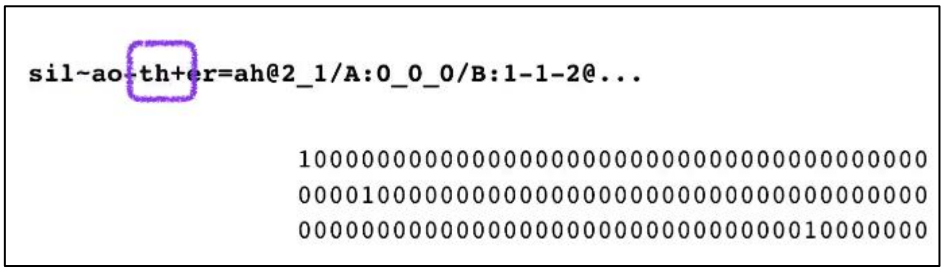
3.3 Back end

The sequence of phonemes and their specifications that have been acquired from the front end are not enough to generate speech. This is mainly because there is a great difference between this linguistic representation of the input and an acoustic representation which is necessary to generate audio. In order to generate speech from the linguistic representation, it needs to be matched with an acoustic representation using a regression model. In order for this to work, the linguistic representation has to go through another pipeline which is called "feature engineering". The reason for this pipeline is because linguistic representation is discrete as opposed to the acoustic representation which is constantly moving in a frame rate. [25]

Same as the linguistic representation, the acoustic one has to be feature engineered. Finally, after both the engineered linguistic features and acoustic features are matched using regression, audio waves can be synthesized from the output sequence, frame per frame. [10] This whole process is what the back end will be responsible for and depending on the type of system, it will have different components and techniques to handle these processes most often by neural networks. [25]

3.3.1 Encoding

As represented on Figure 15, the sequence of phonemes coming from the front end can be visualized as a tree. In order to process this structure, it needs to be flattened into a flat structure such as a vector. To become what is called a "hot encoded" vector, each phoneme in a particular sequence needs to be encoded into a context dependent phoneme as shown in Figure 16. This type of phoneme contains all the contextual information as previously represented by the tree. The contextual information can be the type of syllables, the position of the syllable in a letter/word, stress, POS,. ... In order to push the encoded phonemes in a vector, it need to be converted into binary data using a binary code table where each encoded phoneme represents a binary code. Afterwards, each binary sequence is pushed sequentially into a vector. The result is a hot encoded vector. [25]



```
sil~ao-th+er=ah@2_1/A:0_0_0/B:1-1-2@...

                 1000000000000000000000000000000000000000
                 0000100000000000000000000000000000000000
                 0000000000000000000000000000000010000000
```

Figure 16: Flattened representation of a sequence of phonemes with a part of its binary data. [25]

3.3.2 Up sampling

Before speech can be generated using the vector of flattened phonemes, it needs to be up sampled to an acoustic time frame. As opposed to an acoustic representation, a linguistic representation has no time information. There is no way of telling how long the phonemes will take just by looking at it. In order to up sample the vector of phonemes, a regression model has to be used to match and align the vector of linguistic features with a vector of acoustic features. Similar to linguistic features, the sample of acoustic features need to be feature engineered as well. [25]

3.3.3 Vocoding

The acoustic representation of existing speech samples is required in order to generate new audio samples. These are matched with their corresponding linguistic representation during regression as a way to construct new waveforms. The audio waveform of the existing speech samples however, cannot be used as the acoustic representation. Instead, a spectrogram is extracted from the existing audio waveforms. This is because a spectrogram, as shown in Figure 17, is a more compressed representation of the audio and its frequencies. To extract the spectrogram from a raw audio sample, a signal processor such as a vocoder is typically used. Aside from extracting, the vocoder can also generate a waveform from an acoustic representation such as a spectrogram. [25]

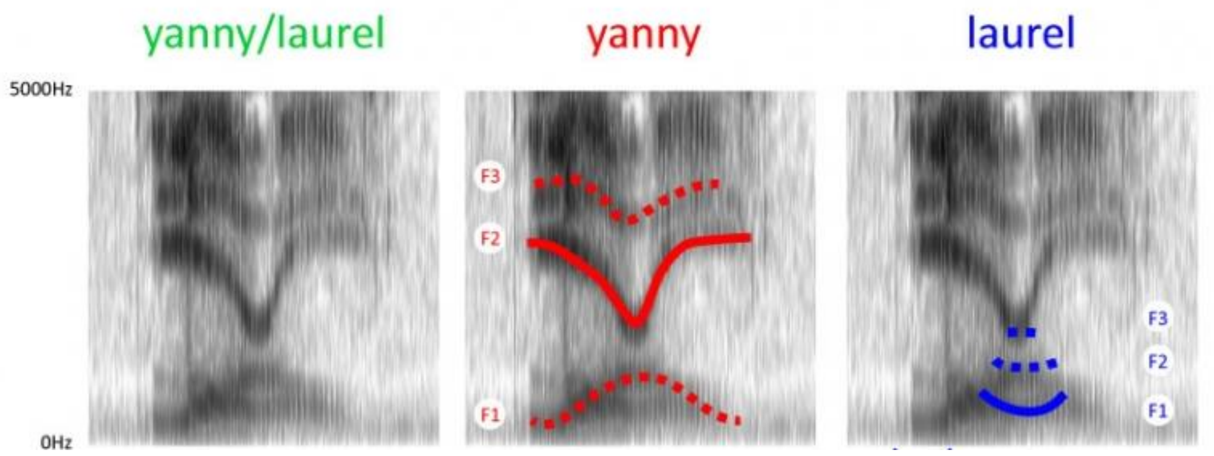


Figure 17: The spectrograms of the "laurel and yanny" audio waveforms. [28]

3.3.4 Regression

In order generate new output samples, the linguistic and acoustic vectors that have been engineered previously need to be matched using a regression model. Linear regression is an algorithm used for both statistical and machine learning purposes. It is studied to understand the relation between input (linguistic vector) and output (acoustic vector) as shown in Figure 18. [29] A recurrent neural network can be used as a regression model where the linguistic vector is fed as an input. By passing the output of the neural network back to the input, the next acoustic sample can be predicted.

Because these recurrent neural networks essentially learn to remember previously generated samples layer by layer, a great amount of layers created over time introduces a certain problem. This issue can be solved by using what is called a Long Short Term Memory or LSTM, which basically knows when to remember and forget certain information. [25] Refer to the next section for a more detailed overview of these types of neural networks.

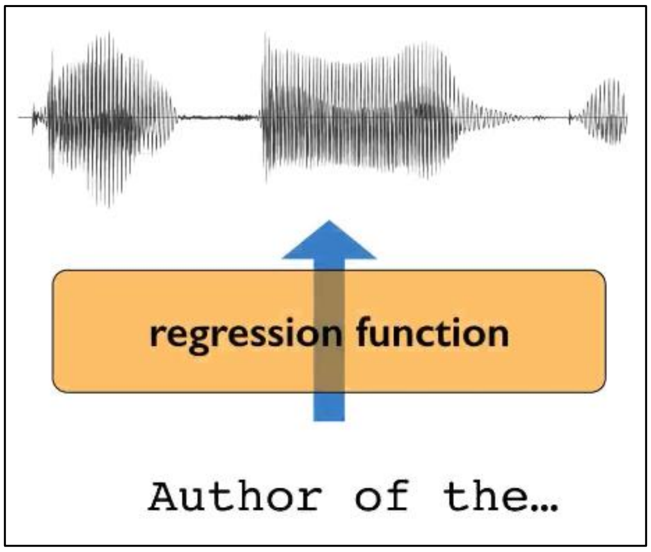


Figure 18: The regression function pictured as a black box. [25]

3.4 Approaches

When a regression model is trained with both the linguistic and acoustic vectors, new acoustic features can be predicted. These features can then be used to generate speech waveforms. This process is called synthesizing. In the next sections, different approaches and techniques are discussed to synthesize speech samples. Regardless of the approach, the definition of synthesizing in the context of Text-to-Speech remains the same. The goal of synthesis is to simply achieve speech waveforms that are understandable to a listener. [25]

The first approach is concatenation, where speech is synthesized using a dictionary of prerecorded speech samples. The second approach extracts the parameters of existing speech waveforms to reconstruct and predict new speech waveforms using statistics. The final approach trains neural networks on text and speech samples in order construct new speech samples.

3.4.1 Concatenative

Concatenative systems use actual recorded speech samples from a corpus to synthesize speech samples. This approach of synthesizing uses an index of small prerecorded speech samples with its corresponding label. During synthesis, the necessary speech samples are looked up in the index, and simply concatenated to each other to generate speech samples. The index that is being used during synthesis is like a dictionary of labeled speech sounds. If the exact specification at synthesis time is not found, then the best sequence must be selected for concatenation. The units that are selected for concatenation can consist of entire words, phones, syllables and so on. The final concatenated speech samples can be stored as waveforms. [27]

The difference between the concatenative approach and other approaches is the fact that with concatenation, real speech samples are used to form new speech signals. With other approaches, the signal is constructed from scratch using certain trained parameters or neural network models. Concatenative systems have been around for a long time, resulting into different generations and extensions. In these traditional systems, techniques in signal processing were used to match the pitch and timing of the original speech waveform with the selected sequence. Several assumptions of signal processing techniques in first and second generation of concatenative systems became limiting factors to the quality of the synthesized audio. Those last systems were typically applied for “pragmatic and engineering” use cases. The weakness of these systems lead to the development of other techniques in the concatenative approach also called “unit-based” selection. [10]

A unit based system uses a database that contain speech units from a corpus. This database aims to cover all of the possible phonetic and prosodic units. These units consist of feature vectors which contain a discrete or continuous value that determine the unit and their context. These features are necessary for the selecting process during synthesis so that the exact requirement can be found with respect to the context of the required unit. So at synthesis time, the best matching unit has to be found for a specific speech target unit. The requirements that are needed within the selecting process are defined using cost functions which should be as low as possible in order for the best possible unit selection. In unit based systems two cost functions can be defined, one for determining how close the target is to the selected unit and another one for describing the connection between the selected units. [30]

3.4.2 Statistical parametric

As shown in Figure 19, the statistical parametric approach uses the vocoder as a way to extract acoustic features from existing speech samples. These features are parametrized and trained by a statistical model. This model can then predict new acoustic parameters from a given input text. Finally, the predicted parameters are passed through a vocoder in order to generate audio waves. [31]

The Hidden Markov Model is often used as a statistical model. It can be visualized as a network of states which is responsible for the parameterization of given speech samples. The parameterized form of these samples consists of the spectrum, the phoneme duration and frequency pattern. Various algorithms inside the Hidden Markov Model are used to generate these parameters. [32] By using a decision tree, the parameters can be mapped with linguistic features so that new samples can be created from a new input. [18] Deep Neural Networks can also be used as a replacement for traditional parametric components. This has been done by Merlin for example, which is discussed in the next subsection. Merlin is a statistical parametric Text-to-Speech system based on neural networks. [25]

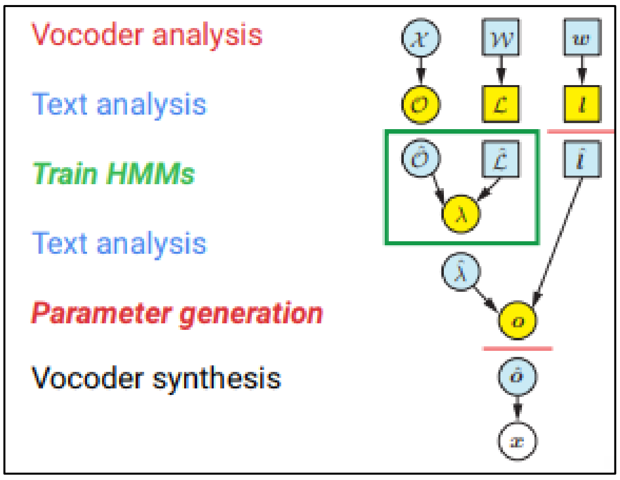


Figure 19: A typical training and synthesizing process with statistical and parametric speech synthesis. [25]

3.4.2.1 Merlin

Merlin is an example of a statistical parametric Text-to-Speech system. It uses neural networks in order to predict acoustic features using linguistic features extracted from an input text. The predicted and parameterized acoustic features are then used to generate waveforms by using a vocoder. [25]

Merlin describes the use of an acoustic model that is necessary to learn the relation between the linguistic and acoustic features. It is mentioned as “a complex and non-linear regression problem” which is a limiting factor for the synthesized speech. Hidden Markov Models were used as acoustic models for a long time, but nowadays a more powerful model such as neural networks are considered instead. Because of the lack of a toolkit to build such systems, Merlin presented itself as an Open Source toolkit to build speech synthesis systems. It allows the combination of different front and back end components. [25]

3.4.3 Generative

This approach works directly with the raw waveform of an audio sample and is able to generate new waveforms by using autoregressive methods. Recurrent or diluted neural network models are used for this which allow the model to grow exponentially as audio samples are generated over time. Training such a model is done with prerecorded audio samples of humans. New audio samples can be generated after the training phase. These samples are actually predicted by using probabilistic methods. It is then fed back into the input so that the next sample can be predicted. [33]

Because this approach works well with raw waveforms, it can create any kind of sound including music. Some implementations of generative Text-to-Speech models have acquired state-of-the-art results. One of these systems is WaveNet, a generative model based on Deep Neural Networks. It uses dilated neural networks for learning and generating speech samples. Because it relies on probabilistic methods to generate samples, the output will be predicted based on its recurrent and exponentially growing architecture. Conditioning must be applied on a model to predict features that are implied. That is why conditioning is mentioned frequently during this approach as a way to train the entire neural network model, or a part of it on a given parameter such as text. By providing input text, speech can be generated based on the given text which is necessary for Text-to-Speech. [33]

3.4.3.1 Neural networks

A neural network in machine or deep learning is a paradigm that is based on the human brain. Similar to the brain, it consists of interconnected neurons that can be trained on data. The trained network can then be used on new data to classify or recognize a pattern. This paradigm works in a different way than traditional computer algorithms. The difference lies within the approach of problem solving. Traditional algorithms use a set of programmed instructions to solve a certain problem and can only solve the problem if the instructions are known. This restricts the algorithm from solving more problems than intended. A neural network however, solves problems by learning. It is unpredictable because unlike traditional algorithms, it is not programmed on instructions. [34]

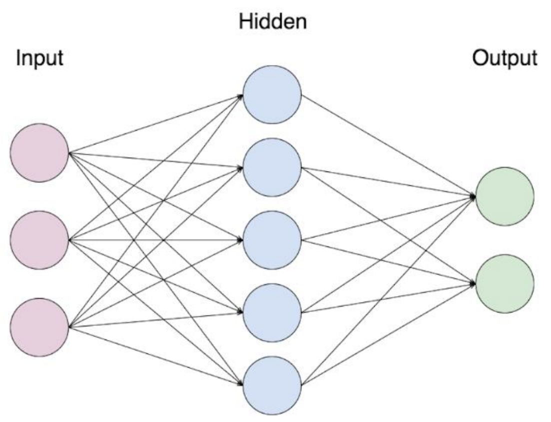


Figure 20: A neural network with an input, hidden and output layer. [35]

A neuron within a neural network consists of multiple inputs and only one output. A neural network is essentially a network of such interconnected neurons. A neuron can be activated based on the weight of the received inputs. If the sum of these weights exceeds a given threshold, then the output of the neuron is activated. The output of the entire neural network is based on the activations of its interconnected neurons. However, a neuron can adapt and "correct" its weights or threshold to become the desired output. This is done to become maximum efficiency and is also known as "Gradient Descent" which uses the "Back Propagation" algorithm. [34] For an explanation on how neural networks work in detail and the algorithms associated with it, refer to "Smart Shops & Face Recognition as a service" by Maarten Bloemen.

As shown in Figure 20, neural networks consist of layers such as the input, hidden and output layer. Different types of neurons can be arranged accordingly to achieve a desired neural network architecture. Each architecture of neural network can serve a different purpose and solve a specific problem. In a "feed-forward" network the signals are passed straight forward, from input to output which is typically used for pattern recognition. A "feed-back" or "recurrent" network also exists where signals can move in both directions by using loops within the network. [34]

3.4.3.2 RNN's and LSTM's

A recurrent neural network is used for its ability to memorize data. These are frequently used in Natural Language Processing where for example, a new sentence must be formed with words. The words are predicted based on the sequence of words that were already predicted. A recurrent neural network can be used to memorize this sequence. Basically, each time a neuron outputs a signal, it is stored in a hidden layer and passed to its own input the next time. This is also referred to as a timestep. In this way, a new output can be predicted by combining the previous timestep output with the current timestep input. The current state as it moves through time depends on the previous states thereby forming "dependencies" linearly also referred to as "memory". For gradient descent, an extended Backpropagation algorithm called "Backpropagation through time" is used. [36]

The problem with recurrent neural networks however is the fact that it is difficult for learning long range dependencies. This is because of the great variable amount of layers that are created in recurrent neural networks as shown in Figure 21. The data that passes through the functions during gradient descent becomes too small after a certain amount of layers as it finally vanishes. This is also known as the vanishing gradient problem. A solution to this problem has been proposed called LSTM's which stand for Long Short-Term Memory units. [36] It allows a recurrent neural networks to learn through more time steps. This is made possible with gated cells, where information can be written to or read from. The gates learn to decide whether to block or accept information that passes through based on its weight. The cell also has a forget gate in case information has to be forgotten. [36]

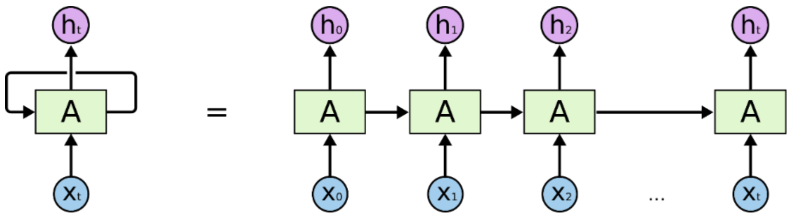


Figure 21: The architecture of a recurrent neural network. [37]

3.4.3.3 WaveNet

In September 2016, Google's DeepMind has introduced WaveNet, a Text-to-Speech model which uses deep neural networks and probabilistic methods to generate speech. By using dilated convolutional neural networks, the generated speech is fed back to the input in order to retrain it, resulting into a state-of-the-art performance. The paper showcases ratings proving their system to sound "more natural" than any parametric or concatenative systems during that timeline. WaveNet is also said to be able to switch between speaker identities to generate different voices, generate music from scratch and even recognize speech apart from generating it. These last features are very promising and worth looking into at a later chapter. [33]

By introducing the use of "dilated causal convolutions", a different form of convolutional neural networks, the entire model is forced to predict the generated speech sequentially. This is to ensure that the speech is predicted in the same manner as human voice which can reach frequencies up to 16 kHz. This could not be possible with only recurrent neural networks because these are not able to generate speech that fast. And even if such neural networks were to be used, the linear design would cause very great number of layers that needs to represent the generated speech. The dilated structure solves this by introducing "exponential growth", rather than a linear one with RNN's. This way, the receptive field of the model also grows exponentially ensuring that the model "remembers" previously generated samples in a tree-like manner as shown in Figure 22. However, the exponential structure makes it harder to feed the predicted sample back to the input. [38]

Training the dilated causal model is quick because during training time, the timestep of each training voice sample is already known and all of the samples can be trained in parallel. However, generating new audio samples will take very long because of the exponential structure of the trained model. The paper mentions conditioning as a way of providing extra parameters during training resulting into a given characteristic of the model. With global conditioning, the output across all timesteps is influenced such as the speaker voice. Local conditioning influences smaller time steps such as linguistic features. [38]

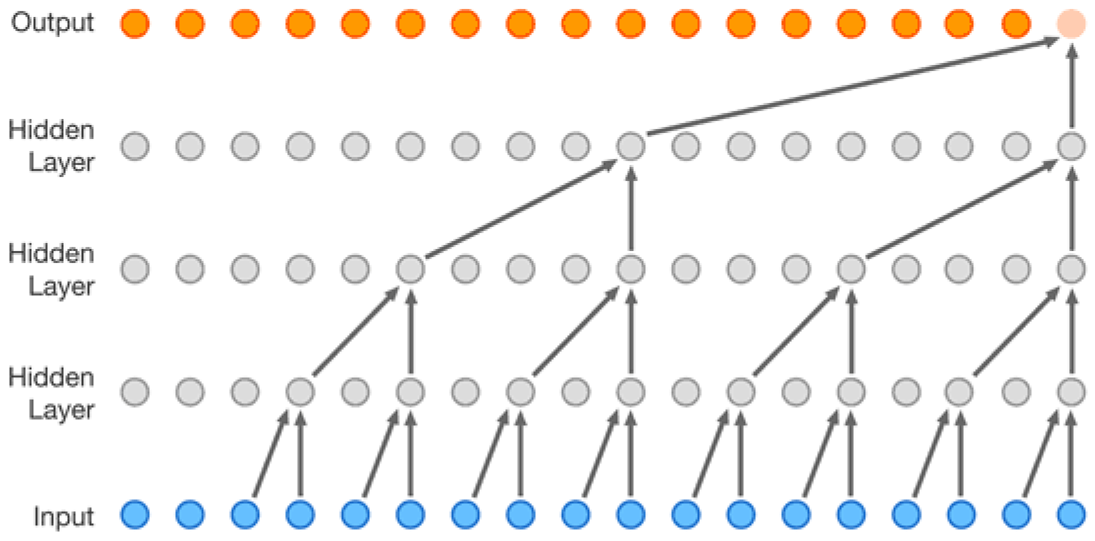


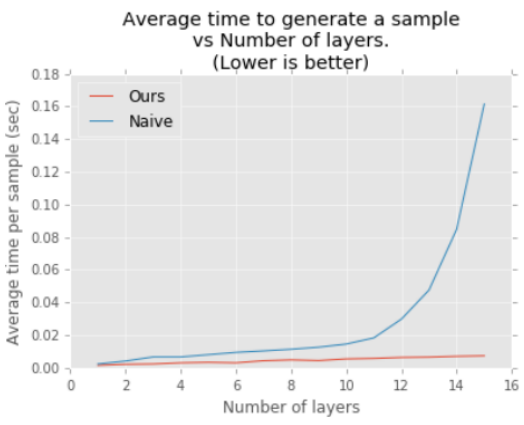
Figure 22: A dilated causal convolution model. [33]

3.4.3.4 Fast and Parallel WaveNet

After the collaboration of the University of Illinois and IBM Watson research center, Fast WaveNet was proposed as an improvement on the current WaveNet Text-to-Speech model. The reason for this improvement is mainly because the original WaveNet was slow in generating speech. This model could not be used in a production environment because of its disadvantage in speed. The difference between Fast WaveNet and WaveNet is that the latter omits redundant convolutional calculations by caching previous ones. It reduces the time complexity from $O(2L)$ to $O(L)$, where L is the number of layer.

The paper briefly describes WaveNet and mentions its state-of-the-art performances. While WaveNet can predict the audio in parallel during training time, it must predict waves sequentially during the generation of speech. This is caused because of WaveNet's architecture which parses the predicted output back to the input to predict the next sample. Because of its time complexity of $O(2L)$ where L is the number of layers, it becomes obvious that a time problem will be caused with large amount of layers. WaveNet as opposed to Fast WaveNet seems to be missing the main ingredient which is the storage of necessary computations (caching). Because of WaveNet's recurrent design, an algorithm can be used to cache recurrent states instead of computing them from scratch for each audio sample.

Two algorithms named the Generation Model and Convolution Queues as shown in Figure 23 and Figure 24, are mentioned in the paper in order to make this possible. The generation model is responsible for accepting an input of recurrent states and producing an output of new recurrent states. The states are "popped" when taken from its queue and new states are "pushed" onto the Convolution Queue (First-in-First-out). [39] Figure 23a shows how Fast WaveNet achieves a more constant synthesis time in proportion to the amount of layers as opposed to the naive way.



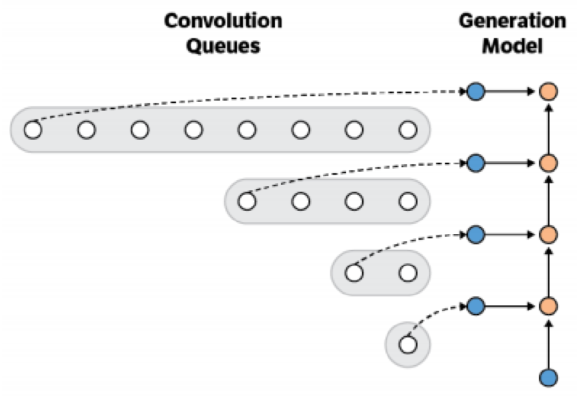


Figure 23: (left) A graph plotting the results of Fast WaveNet compared to alternatives. [39]

Figure 24: (right) The Convolution Queues next to the Generation Model. [39]

Fast WaveNet was not the only improvement upon WaveNet. Another paper presents Parallel WaveNet, which is an improvement on the original WaveNet by its original contributors. As previously described, WaveNet is rather slow to generate speech with. The main cause for this issue is that generating speech happens sequentially because each output needs to be pushed back into the input.

Parallel WaveNet solves this issue by introducing "Probability Density Distillation" which is described as a new method for training a parallel network from an already trained WaveNet. This way, the speech samples are generated 1000 times faster than the original WaveNet. By introducing this new algorithm, millions of users will be able to generate high quality speech. [39] WaveNet was fast to train using autoregressive methods but slow to generate speech with. Parallel WaveNet introduces Inverse Autoregressive Flows, where generating speech can be done parallel while it is still rather slow to train. By combining the best features of both Parallel WaveNet (fast speech) and WaveNet (fast training), a new architecture is discovered. This new architecture as shown in Figure 25, works with a "teacher" model which is an already trained WaveNet model and a smaller WaveNet student that learns from the teacher model. Over time the student tries to match the probability by tuning its own generated speech with backpropagation under the distribution of the teacher. The paper refers this process as "Probability Density Distillation". [40]

Training with this model only might not be enough for the student to generate high quality speech because losses may occur resulting into bad pronunciation. That is why extra loss functions have been introduced such as the power loss. This makes sure that all of the sorts of frequency bands are used averagely just like in human speech. [40]

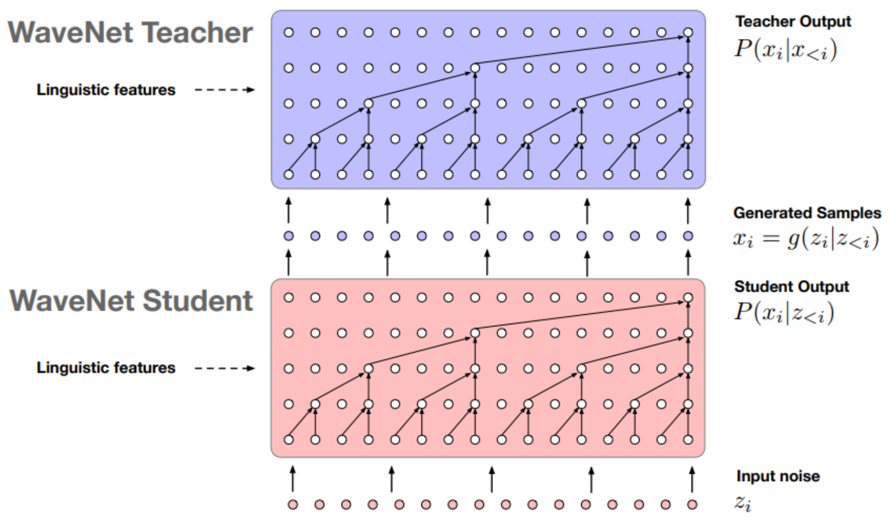


Figure 25: Probability Density Distillation. [40]

3.4.3.5 Tacotron

Various components and pipelines of the front and back end of Text-to-Speech systems were discussed earlier. Most of these components are complex to design and are trained independently which requires certain expertise. An end-to-end model however can mitigate the complexity by allowing to train a single model on simple text and audio pairs. It is also easier to condition on given parameters because it can achieve this from the beginning, rather than for each component. The goal of Tacotron is to design an end-to-end generative model that is easier to train from scratch and is adaptive to new and various real world data. WaveNet has been discussed as a state-of-the-art generative model. Unfortunately, it is rather slow because it is autoregressive. And unlike Tacotron, WaveNet is not an actual end-to-end model because it reuses components from other existing TTS solutions. [41]

Tacotron’s architecture as shown in Figure 26, is based on a sequence to sequence model that is composed of an encoder, a decoder and a post-processing net. The encoder is used to extract the features of an input text by taking a sequence of its characters in a hot encoded vector form and embedding it using a “pre-net”. The embedded vectors are parsed to a decoder. The decoder is responsible for the alignment of the encoded linguistic frames and acoustic feature frames from the speech samples which are raw spectrograms. The output of a decoder is a predicted raw spectrogram frame. The process is frame-based and consumes one frame at a time. This is not the case with WaveNet which uses an autoregressive model. Tacotron being frame-based is subsequently faster because a variable amount of frames can be predicted at once. In order to synthesize waveforms of the spectrogram frames, it has to be converted using a “post-processing net”. Finally, the Griffin-Lim algorithm is used to construct the waveforms from the post processing output. [41]

In February 2018, Berkeley and Google have introduced Tacotron 2. The main difference with the original Tacotron is the model design. Earlier in Tacotron, the Griffin-Lim algorithm was used to synthesize waveforms from raw spectrograms. However, this approach can be replaced with neural network style approaches for better audio quality because it is significantly lower with the Griffin-Lim approach. As a result, Tacotron 2 replaces the Griffin-Lim with a modified version of the WaveNet vocoder. Tacotron is essentially a combination of the Tacotron sequence-to-sequence model and a modified WaveNet vocoder. Both components are bridged using mel-frequence spectrograms. This is a better and lower level acoustic representation than raw waveforms because it can be computed easily from a given speech sample and is easier to train with. [41]

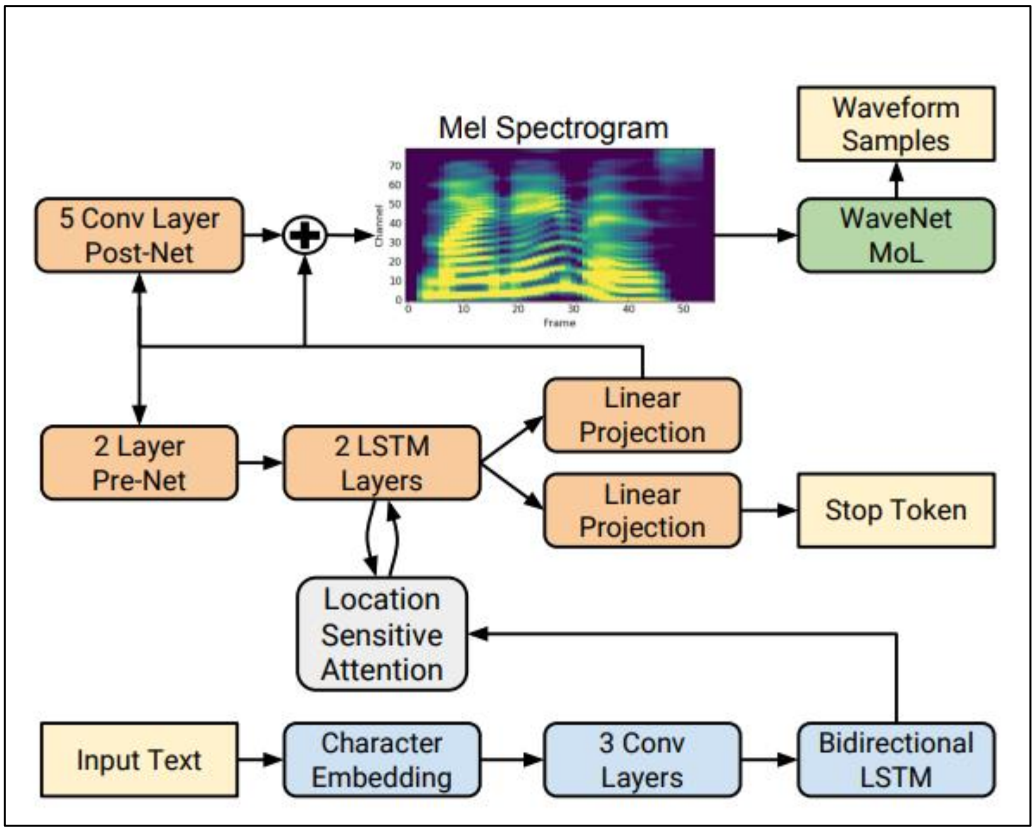


Figure 26: Tacotron 2's architecture. [41]

Tacotron 2 is similar in its sequence-to-sequence model. The encoder accepts a hot encoded vector of characters which the decoder uses to predict spectrograms from. However, Tacotron 2 as opposed to Tacotron, uses vanilla LSTM and convolutional layers in the encoding and decoding process. The result of this was a more simple to build architecture. For a more detailed explanation of how LSTM and convolutional networks work, refer to “Smart Shops & Face Recognition as a service” by Maarten Bloemen. The predicted mel-frequence spectrograms are afterwards used by the WaveNet vocoder to synthesize waveform samples. [42]

One month after Tacotron 2, an extension on Tacotron had been published. The extension proposes a way to embed certain prosody in a model using a desired sample of prosody. It basically means that the pronunciation style of existing speech samples can be “learned” and applied on new generated speech samples even if the synthesized text has never been learned before. Also, the “space of prosody” that is being referred to in the paper, can even be applied if the voice of the learned speaker and synthesized speaker are different. [43]

3.5 Evaluating Text-to-Speech

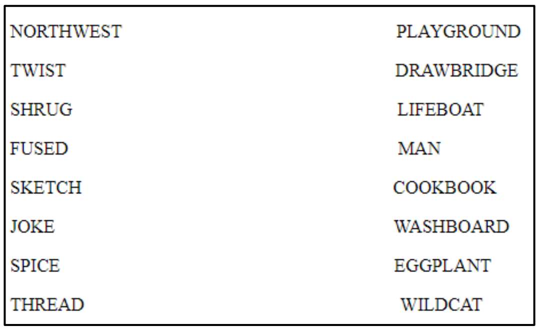
The quality of the speech output is determined by its intelligibility and its similarity to human voice. This is can be important especially if the listener has impairments or disabilities and relies on the assistance of the generated speech. Either way, there are a few ways to determine the quality of the generated speech as listed below. [10]

3.5.1 Intelligibility

There are a few approaches to test intelligibility on Text-to-Speech systems. One of them is a word recognition test. In this type of test, a subject has to listen to a playback of words coming from a Text-to-Speech system. The words can be played back apart or in an example sentence. The subject is then asked which words have been recognized. Some words can be played back multiple times so that the subject has to distinguish the words. With these types of tests a group of subjects can be handed a multiple choice sheet and played back a specific set of grouped words. For testing on sentences rather than words, existing tests can be used such as “The Harvard sentences”. As shown in Figure 28, these contain a set of sentences “with a natural distribution of phonemes in English”. [10]

However, with these test sentences, a subject could be able to guess the arrangement of word. In order to avoid this, unpredictable sentences should be used rather than predictable ones. These test sentences are not supposed to make sense anyway. Such an example are “The Haskins sentences”. While the unpredictable tests are sufficient and commonly used, they are simply not recommended for real world applications. A reason for this is that in a real life application, the sentences to be synthesized may be predictable to the listener. In such a case, it is a bad idea to test a system on unpredictable words and sentences where the syntax does not make any sense. That is why these types of tests are rather done with sentences with normal syntax and low predictability. [10]

Another way to test the intelligibility would be to conduct Tikofsky's Test as shown in Figure 27. These are a set of 50 words that can be played back to subjects. These participants should not be familiar with the pattern of the words. Eventually, the percentage of recognized words is used as a way to rate the intelligibility. [44]

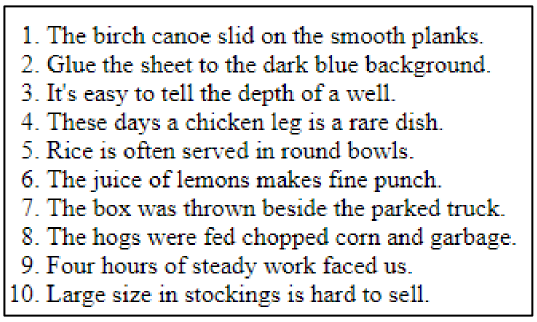


NORTHWEST	PLAYGROUND
TWIST	DRAWBRIDGE
SHRUG	LIFEBOAT
FUSED	MAN
SKETCH	COOKBOOK
JOKE	WASHBOARD
SPICE	EGGPLANT
THREAD	WILDCAT

Figure 27: A set of words that are used during Tikofsky's Intelligibility Test.

3.5.2 Naturalness

In order to test the naturalness of the generated speech, subjects can be asked to rate generated speech based on a scale. For this, Mean Opinion Scores or MOS are often used as a scale between 1 and 5. This type of test is simple and can provide a decent representation of the systems naturalness. The harder part of this test lies within the interpretation of the results. While the test is meant for naturalness, factors such as likeability can also come to play. This becomes a problem when real human speech is used in the test as well.



1. The birch canoe slid on the smooth planks.
2. Glue the sheet to the dark blue background.
3. It's easy to tell the depth of a well.
4. These days a chicken leg is a rare dish.
5. Rice is often served in round bowls.
6. The juice of lemons makes fine punch.
7. The box was thrown beside the parked truck.
8. The hogs were fed chopped corn and garbage.
9. Four hours of steady work faced us.
10. Large size in stockings is hard to sell.

Figure 28: One of the few lists of the Harvard Sentences. [45]

In theory, human speech should receive maximum score on naturalness, in reality that is not the case. This is again because people prefer "pleasant" sounding voices meaning it is not only naturalness that amounts to a higher score. Also, comparing different tests can also become a difficult task because two conditions may not always receive lead to the same score. In most conditions, natural speech in tests is used as an anchor to future results. [10] For example, if actual human speech scores 4.6 on a MOS test, a test score of 4 can be judged as more natural sounding than a test score of 3.5 because on the scale, it is closer to human speech.

4 Research case

The purpose of this thesis is not only to get better understanding of what Text-to-Speech is, but also to solve a given Text-to-Speech problem. That is why this research case is presented. Earlier in the abstract, the Text-to-Speech problem with the Pepper robot was described to be lacking in naturalness. This problem can be solved by replacing it with alternative Text-to-Speech solutions that are more natural and more efficient. An exploration phase however, is necessary to search for alternative solutions. After this phase, a comparative study will be conducted using a set of measurements. The purpose of this study is to compare the found solutions and to propose a conclusion. The measurements of the comparative study are based on techniques to evaluate Text-to-Speech. It also includes other requirements concerning the hardware or software environment of the client, which is the Pepper robot. However, the client is not restricted to this type of robot. It is assumed that the Text-to-Speech systems should be applicable for multiple software and hardware environments.

An important note is that generative Text-to-Speech systems will be the primary focus during the exploration phase. However, the exploration phase also includes a statistical parametric system. Earlier, a few example papers were discussed in the generative and parametric category. Some of these papers fail to provide an available open source implementation on the internet. This does not mean that the discussed papers are any less relevant. The concepts discussed within these papers are reoccurring in Text-to-Speech according to previous chapters. However, if a paper cannot provide an actual working example, then it is not further evaluated. [46]

The description of the systems in question during the exploration phase serve as a mere overview of the Text-to-Speech capabilities. Some of the features that are included with Text-to-Speech such as multi language or multiple voice are extra features. These are "nice to have", but not essential for the study. The comparative study includes measurements and specifications necessary for evaluating these systems. [47] Earlier, these systems were mentioned to be evaluated on objective or subjective scores such as MOS scores for naturalness or scores on intelligibility. It is possible that certain numbers or samples necessary for the comparative study are inaudible or simply unmeasured for a specific Text-to-Speech system. In these cases, the subject system will be marked as "unmeasured" or "inaudible" in the specific category of testing. Some of the numbers however are necessary as it is impossible evaluate without. If that is the case, then an experiment has to be conducted to compensate for the unmeasured numbers. That can be done using test subjects and evaluation sheets.

4.1 Measurements

There are minimal requirements that have to be met for a Text-to-Speech system to be considered adequate. These requirements are based on measurements that can be conducted in Text-to-Speech such as naturalness and intelligibility, and include problems that are being faced with the Pepper humanoid robot. [10] Conducting a search on the internet has resulted into a collection of various Text-to-Speech solutions. Not every solution is applicable and that is why a comparative study is done using the measurements on which the solutions can be compared. It is possible that some measurements can outweigh others. In the next section, a feature comparison matrix is created using these measurements. Each category of Text-to-Speech systems has a designated matrix. The Text-to-Speech systems are compared using the matrix in a specific category.

Section 5 of the previous chapter shows that naturalness and intelligibility are factors that are used to evaluate Text-to-Speech system. It would be convenient if most of the Text-to-Speech systems included these test numbers by default. Unfortunately, as the comparative research will reveal, this is not always done. Aside from these key factors, there are a few other requirements that emerge from the software environment or the architecture behind the system. Some systems are based on neural networks and could require training and training data. If the target systems fails to provide the requirements that are necessary for training or synthesis, it will be excluded of the evaluation by default. Or even if the target system has other unappealing disadvantages such as intolerable long training or synthesis time.

Other requirements also come to play that are based on software or hardware environments. The Pepper humanoid robot is typically programmed with the Python or C++ programming language using a Software Development Kit. [48] In this case, the SDK could be a limiting factor. By taking the hardware or software requirements in to consideration, possible issues with compatibility can be avoided. From a client perspective, it is expected that the implementation of choice must be compatible in as many environments as possible, especially a Linux environment as that is the desired environment of work.

4.1.1 Primary measurements

The primary measurements that are listed below, are the factors on which Text-to-Speech systems are commonly evaluated, such as the intelligibility and naturalness. It is also important for the source code of a found solution to be open source. This emerges from a programming standpoint in which a license should grant certain rights to the programmer. Also, it is desirable for the solution to work correctly, meaning that it should be free of errors. Unfortunately, the possibility of a working solution can depend on the target software/hardware environment or even the specific version of the solution. Once a found solution fails to provide a working example, it is not further evaluated.

- Intelligible: the understandability of the generated speech by listeners.
- Naturalness: the significance of the generated speech to actual human voice.
- Open source: Licensed and working correctly.

4.1.2 Neural Network based measurements

There are certain measurements for solutions that use neural networks in their architecture. These include training time, synthesis time and the availability of training data. The synthesis time is similar to the time that is used to evaluate a Text-to-Speech system on how fast it can generate speech. Both training and synthesis times should be efficient, meaning that faster is better. Some papers also refer to synthesis time as inference and mention that long inference times are not feasible. [40]

It is also important for the training data to be available. However, preference lies with pretrained model. These are neural network models that are trained beforehand with the necessary training data. This eliminates the need for hardware and data that is necessary for training such a neural network.

4.1.3 Measurements for API’s and SDK’s

There are also measurements that can be done on commercial API solutions such as latency. The latency includes the response time of the API. This is the time it takes for a server to process the request and generate a response. [49] The list below also includes cost as an important measurement. The cost of synthesis using an API is evaluated per million characters. If a Software Development Kit is used rather than an API only, then compatibility of the SDK becomes an important factor.

- Latency: the time it takes for a request to a server to complete, faster is better.
- Cost: The price of the API or SDK service, less is better.
- Compatibility with the mentioned software specification.

4.1.4 Software specifications

The software specifications emerge from the target software environment of the client. The generic client is described as a Linux based environment that supports any Robot Operating System version, by preference the latest one. However, the client can differ from the generic one for example the Pepper robot. In that case, the solution has to be compatible with the NAO SDK. This is the SDK for controlling and programming the robot. These are the main software environments for the target solution to be compatible with. [50]

- Compatible with a Linux system that supports Robot Operating System (ROS).
- Compatibility with the NAO Software Development Kit is a plus.

4.1.5 Text-to-Speech features

There are features that can play a role during the comparative study. However, these are more of a preference rather than a necessity. For example, if the target solution includes Dutch as an extra language for synthesis or whether certain markup language is supported or not.

- Multi speaker: availability on multiple voices is a plus.
- Multi language: the language of the synthesized speech. English and Dutch are sufficient.
- Markup language: tags used in the text to emphasize prosody.

4.2 Exploration phase

During this phase, open source Text-to-Speech systems are explored based on references from the internet. These systems are based on papers and have been discussed earlier or are made accessible through an API or a Software Development Kit. After the exploration phase, the systems will be analyzed and compared based on the requirements in the comparative study.

4.2.1 Paper based implementations

In the course of a few years, papers have been written on generative Text-to-Speech solutions such as WaveNet and Tacotron. Unfortunately, these papers are unusable in the sense that it may not provide an actual Text-to-Speech program that is being strived for. However, the source code of an implementation of such a paper can be made open source by its original contributors. If that is not the case, other researchers or academics may attempt to implement the papers and provide an open source solution on the internet. [46] There are many open source solutions available on GitHub. When looking for repositories of a specific paper, popularity was taken into consideration as well as the status of the author or company responsible for the source code.

4.2.1.1 WaveNet and extensions

WaveNet was earlier described as a neural network model for synthesizing speech. It has been rated very high in terms of naturalness. The model itself can also be conditioned globally on different speakers in order to become multiple voices, and locally on text to become Text-to-Speech. WaveNet wishes to provide a framework for other audio generation techniques than Text-to-Speech as well, such as music generation or voice conversion. [38] Several improvements upon WaveNet have been attempted in order to deal with problems like the slow speed of speech generation. This speed problem caused WaveNet to be unsuitable for production environments where speech has be generated instantly. With a combination of both Fast, Parallel WaveNet and other papers, the speed problem along with other issues have been effectively solved. Parallel WaveNet is known to be thousand times faster than the original WaveNet while maintaining its naturalness. [51]

Unfortunately, an open source implementation of Parallel WaveNet is not available as of April 2018. However, it is already implemented by Google in their Cloud Text-to-Speech and Google Assistant services. Users that wish to use Parallel WaveNet would have to consume Google's services instead. A list of open source implementations on the original WaveNet is available. All of the available implementations are programmed in Python and some of the repositories have a high amount of stars. This mostly indicates the popularity of a given repository. [52] Most of the found WaveNet implementations unfortunately lacked local conditioning, meaning that Text-to-Speech was not possible and instead human-like speech would be generated in an incomprehensible form. [38] For testing purposes it would be sufficient but the Text-to-Speech characteristics would have been impossible to compare with other solutions. That is why the focus shifts towards Tacotron, which is based on WaveNet and does actually have a Text-to-Speech implementation available on Github.

4.2.1.2 Tacotron and extensions

Tacotron is similarly to WaveNet, a generative Text-to-Speech model based on deep neural networks. However, it is an end-to-end system meaning it is composed of a single model that can be trained as a whole. Tacotron proposes an easier to train Text-to-Speech model with only text and audio pairs. It allows for easier conditioning of features such as speaker identity or language. As opposed to WaveNet, Tacotron is also based on sequence-to-sequence models. [41] In Tacotron 2, both Tacotron's sequence-to-sequence model are combined with the WaveNet vocoder. In combination with other improvements, the naturalness has improved significantly outperforming traditional systems. [42] Several other are papers proposed on Tacotron afterwards as well to improve prosody prediction. [43]

There are a dozen open source implementations available of Tacotron and a few of Tacotron 2 on Github. A working implementation is sufficient as a proof of concept as long as the repository reflects the paper and its results. Considering that, a few implementations were selected based on the amount of stars, the sample results and the background of the author.

4.2.1.3 Merlin

Merlin serves as a toolkit to build statistical parametric Text-to-Speech systems using neural networks. It uses a front end tool for feature engineering the input text and neural networks in the back end for predicting acoustic features. A vocoder is used to synthesize speech samples using the predicted acoustic features. The toolkit is written in Python and is fully open source. Merlin is not an end-to-end system. It allows combining external front end and vocoder tools with Merlin's core back end functionality. It is possible to change the language as well as the identity of the speaker by training the model on a specific set of training data. [25]

4.2.1.4 Deep voice and extensions

Deep Voice is a generative model that is entirely composed of deep neural networks which uses a modified WaveNet model that is faster to train. Deep Voice replaces every component in Text-to-Speech with neural networks in order to become a more flexible and simple system. It also introduces WaveNet inference kernels that are 400 times faster than the original WaveNet implementation. Deep Voice is proposed as a standalone real-time production system meaning no pre-existing system is required and speech has to be synthesized real time. Similar to Tacotron, it allows training on simple audio and text samples. [53]

Deep Voice 2 is the successor of Deep Voice. It successfully embeds multiple speakers using a single model and introduces an improved sequence-to-sequence Tacotron model. As a result the audio of the synthesized speech improves significantly. [54] Deep Voice 3 is introduced months later and achieves parallel computation for even faster training and synthesizing speeds. With this model, a different architecture is presented as well as a kernel that can "serve up to ten million queries per day on one single-GPU server". [55]

4.2.2 API's

4.2.2.1 IBM Watson

IBM Watson is a platform that offers multiple AI services to clients. There are multiple services to choose from such as Text-to-Speech. This feature can be used freely up to 10,000 characters per month but costs 0.02$ per thousand characters on a paid plan. It comes with different speakers, a mobile SDK and a custom API where users can add their own dictionary of words. There are no MOS scores available on the naturalness, meaning it will be harder to compare with other solutions. [56]

4.2.2.2 Amazon Polly

Amazon Polly is the Text-to-Speech service API of Amazon. As opposed to Alexa the Smart Assistant, which is also an Amazon product, Polly is a Text-to-Speech service only and can be embedded in multiple devices. This service supports multiple languages, SSML tags and Speech Marks metadata to adjust acoustic parameters and add effects to the generated speech sound. The API is "low latency", meaning it will be fast to generate speech with and previously generated speech can be cached. During the first year at sign-up, 5 million characters per month are free to use and "pay-as-you-go" pricing is settled after exceeding the limits. [57]

4.2.2.3 GTTS

Google Text-to-Speech is the TTS API by Google that was primarily used by Google Services such as Google Translate and still is being used today. The API is free and simple to use. With only few lines of code as shown in the code snippet below, speech is generated from a given input text and downloaded as an MP3 file. It is unknown which type of Text-to-Speech system is behind the system. It is assumed to be a concatenative system because it sounds rather robotic and jumpy. However, it is a fast, free and simple Text-to-Speech solution that serves its Text-to-Speech purpose. [58]

```python
tts = gTTS(text = 'Hello world!', lang = 'en')
tts.save("test.mp3")
```

Code snippet 1: Text-to-Speech sample code with GTTS in Python.

4.2.2.4 Google Text-To-Speech Cloud

The Google Cloud Text-to-Speech is cloud service that uses WaveNet for synthesizing speech. It is a real-time service that supports over 32 voices and 12 languages. The API can be implemented in several devices using REST requests. It is also possible to control the pitch, gain and speed of the synthesized speech. SSML tags are supported as well. The service costs 4 or 16 dollars per million characters depending on the desired pricing plan. The 16 dollar pricing plan contains a voice package with voices that sound more natural than the ones in the standard package. [59]

4.2.3 Integrated

4.2.3.1 Google Assistant SDK

Google Assistant SDK can be used to integrate Google Assistant in own software. Google unveiled this product in May 2016 and it is supported on a variety of devices such as mobile phones, cars and their latest Google Home product as shown in Figure 29. Google Assistant is a voice-controlled assistant and is the successor of Google Now, which was rather unknown as opposed to its competitors. Google competes with smart assistants such as Amazon's Alexa. The Google Assistants speech synthesis and voice recognition capabilities have been tested during the project and the evaluations they have received for the outcome are justified.

The main reason the Google Assistant SDK is being discussed, is because Parallel WaveNet is implemented as the Text-to-Speech system in the back end. Unfortunately, Google Assistant does not allow users to use the Text-to-Speech functionality. It is only meant to serve as an assistant although the assistant can be asked to repeat sentences. That does not count as Text-to-Speech however. With a small experimentation, a free workaround was built using the Google Assistant SDK and several other tools. The workaround is not discussed in this thesis. Refer to the Google TTS Cloud for a correct way to use Parallel WaveNet as a Text-to-Speech system. [60]

Figure 29: The Google Home. [61]

4.2.3.2 Alexa SDK

Alexa is the cloud-based Smart Assistant of Amazon that was originally shipped with Amazon Echo. With the SDK, Alexa can be integrated into any supported device. While Alexa uses a Text-to-Speech engine to speak, it is unclear whether it uses Amazon's Text-to-Speech service named Polly or has its own integrated Text-to-Speech implementation. Anyhow, assuming it is a service, the Text-to-Speech service is connected with the Alexa service and similarly to Google Assistant, it is impossible to use as a standalone feature. Google Assistant and Alexa provide similar features and are obvious competitors in the Smart Assistant race. [62] This is primarily the reason why Alexa is discussed in this chapter at all.

It can be interesting to see the differences of the generated speech between these two assistants and how well they perform on a scale. Unfortunately Amazon has not provided benchmarks of their Text-to-Speech performance, which can be used to distinguish the most natural sounding system. Because these scores are commonly subjective, a personal test can provide similar yet inaccurate results. The inaccuracy will most likely be due the fact that actual professional tests are often performed and organized with multiple people and organizations. [10]

4.3 Comparative study

The comparative study includes solutions of the exploration phase and uses feature comparison matrices as a way to compare these solutions. [47] The features on the upper columns are based on measurements that were discussed earlier. An experiment is also conducted to measure the numbers that are necessary for the evaluation during the comparative study.

4.3.1 Comparison matrices

Name	Open source	Naturalness (1-5)	Intelligibility (%)	Paper (1-5)	Pre-trained	Synthesis time (s)	Training time (min)
Tacotron	Yes	2.04	75	3.82	Yes	123	Unmeasured
Tacotron 2	No	Unmeasured	Unmeasured	4.53	/	/	/
WaveNet	Yes	Inaudible	Inaudible	4.21	Yes	65	Unmeasured
Fast WaveNet	Yes	Inaudible	Inaudible	4.21	Yes	65	Unmeasured
Parallel WaveNet	No	Unmeasured	Unmeasured	4.41	/	/	/
Deep Voice 3	Yes	2.42	25	3.78	Yes	7	Unmeasured
Merlin	Yes	2.42	Unmeasured	/	No	11	388

Table 2: Comparison matrix for paper based systems

Name	Link to Github	Pretrained on
Tacotron	https://github.com/Kyubyong/tacotron	LJ Speech Dataset 200K steps
WaveNet & Fast WaveNet	https://github.com/ibab/tensorflow-wavenet	VCTK corpus 72K steps
Deep Voice 3	https://github.com/r9y9/deepvoice3_pytorch	Nyanko 500K steps
Merlin	https://github.com/CSTR-Edinburgh/merlin	CMU_ARCTIC dataset

Table 3: The repositories of the solutions that were used in the previous table.

	Average synthesis latency (ms)	Naturalness (1-5)	Intelligibility (%)	Cost	Output	Multi speaker	Multi language
Amazon Polly [2]	Unmeasured	/	/	4.80$ / million characters	MP3, Audio stream	Yes	Yes
IBM Watson [2]	534,8	2.4	86	20$ / million characters	WAV, FLAC, MP3, PCM,…	Yes	Yes
GTTS	141,1	3.08	60	Free	MP3	Yes	Yes
Google Cloud TTS [2]	819,3	3.45	78	4$ / million characters	MP3, OGG,…	Yes	Yes
Pepper TTS	Unmeasured	3.06	52	Free	Audio stream	No	Yes

[2] Demo only

Table 4: Comparison matrix for API's

4.3.2 Experiment and subjects

The naturalness and intelligibility of the Text-to-Speech solutions in the comparison matrices are determined with an experiment. Unless the solution includes these numbers, the results of the experiment are used instead. For paper based solutions, the experiments may have already been conducted by researchers of the corresponding papers. However, the scores mentioned on these papers have little to no value as the results of an open source solution that is merely based on the paper can never correspond to results of the original paper. For the results to be similar, the implementation has to similar to the original implementation. Every implementation in the matrix is only based on the paper and is not the exact implementation as tested in a specific paper. Also, the factors that come in to play during the original training of neural networks of the papers are impossible to reproduce. In conclusion to the previous sentence, this would require the same neural network architecture as used in the reference paper and the same conditions and parameters of training that have been conducted by the researchers.

The subjects who have been used for the experiment are primarily students or teachers of PXL-IT. The average age of the subjects is 24 years old with the youngest subject being 20 years old and the oldest 35 years old. The experiment has been conducted using a consistent batch amount of subjects. The speech fragments however remained the same for each batch. An HP Z220 workstation has been used to generate the samples. The exact specifications of this workstation are discussed in the following chapter. A Trust Remo 2.0 speaker set or the speakers of the Pepper have been used to playback the sound fragments to the subjects and a closed workspace office was chosen to conduct this experiment.

4.3.3 Techniques

A set of techniques is required for testing the Text-to-Speech systems. Mean Opinion Scores are used based on List 1 of synthesized Harvard Sentences or alternatives [45]. The intelligibility is tested by conducting a modified Tikofsky's Test. [44] Both tests are done using subjects and an evaluation form for the results. The results are then copied onto the matrices. This is done for both paper based implementations as API based systems. For naturalness, subjects are unknowingly asked to rate real human speech first. This is to determine the accurateness of the ratings. The synthesis time is measured on List 1 of the Harvard Sentences. If local conditioning is not applied on a certain system, then the resulting speech will be inaudible. In that case, both tests are omitted and marked as unmeasured.

The API's are also tested on naturalness and intelligibility with the same methods as before with the same subjects. The API's however are evaluated on additional requirements such as latency and cost. Unfortunately, the latency is harder to measure due the fact that some demo's do not allow Text-to-Speech and only have sound samples available. The latency is an average of multiple measurements on a sentence of the Harvard Sentences with a consistent connection. For intelligibility, a modified Tikofsky's Test was used again with its corresponding 50 words. However, only 10 words were used during the experiment. If the words could not be synthesized, then other available sample words were used. Naturalness was tested on List 1 of Harvard Sentences or based on other available sound samples. In order to avoid that subjects could memorize the given word samples, different samples were randomized in order.

4.3.4 Progress

Unfortunately during the first batch, some of the subjects reacted poorly to the experiment mainly because the results of Text-to-Speech samples may have sounded "funny" or "weird". This can be considered as a form of response bias. Also, when asked to rate a solution based on scale, some subjects seemed to have misinterpreted the scale. [63] Questions such as "how can speech sound more human" were asked during the experiment indicating that subjects did not really know how to rate speech in the first place. That is why an actual human speech sample was asked to rate beforehand without the subjects knowing it was an actual human. This was done in order to judge the accurateness of the ratings and to restrict the scale to an average of this score.

4.3.5 Conclusions

4.3.5.1 Naturalness and intelligibility

The MOS scores which the paper based solutions have received during the experiment were dramatically lower than the score of the original paper. Even when considering that the average MOS score that human speech received during the experiment was 4.5, which is a realistic score. [40]

The MOS score of paper based solutions were also lower than the API based solutions. This is because the API solutions are products of well-trained neural network models as opposed to the use of pretrained models during the experiment. Some of the subjects have reported that noise and other unpleasant acoustic features were the cause of a low MOS score.

The Google Cloud Text-to-Speech solution has received the highest MOS score as shown in Table 4. It certainly scores higher than the Text-to-Speech solution of the Pepper humanoid robot and not only on naturalness but also on intelligibility. IBM Watson however, has received the highest intelligibility score while only scoring a 2.4 on naturalness. Some of the solutions such as WaveNet were impossible to evaluate due to the synthesized speech being inaudible. Amazon Polly simply did not provide a way to synthesize speech without purchase meaning it was also not further evaluated.

4.3.5.2 Synthesis and training time

The training time of certain neural network based solutions has been omitted considering that a pretrained model was used for a majority these solutions. Deep Voice 3 has the fastest synthesis time in seconds of all the open source paper based solutions. The paper of Deep Voice 3 mentions the techniques that account for the low inference time. [55] The synthesis time of API-based solutions is the latency. This includes the response time and the actual latency of the API requests. [49] The solution with the least latency is the Google Text-to-Speech API. The average latency of this API was around 141 milliseconds. This is fast considering that the more advanced Cloud Text-to-Speech service of Google that is based on Parallel WaveNet, takes at least 819 milliseconds to finish an average request. [40] The synthesis time of the Pepper Text-to-Speech solution is unmeasured mainly due to the fact that the API or the Text-to-Speech module works locally. It can therefore not be compared to the latency of remote API's or cloud services.

4.3.5.3 Cost and licensing

The cost in the context of Text-to-Speech does not include costs such as hardware pricing or costs for training a neural network model. It strictly refers to the costs for synthesizing speech. The open source solutions are free to clone from Github but may be licensed and therefore include legal restrictions. The open source implementation of WaveNet and Python GTTS are licensed under MIT, meaning they can be used commercially and are open for modification and distribution. [64] The Alexa SDK, Google Assistant SDK, Tacotron and code samples of Google Cloud TTS fall under the Apache License 2.0 and can be also be used commercially. [65] The Pepper TTS, which uses the NAOQI SDK, is licensed under the copyright of SoftBank Robotics Europe for commercial use. [66] The PXL University College or PXL AI & Robotics Lab for that matter, has no intention of using the SDK commercially.

The open source implementations on Github of GTTS and Pepper TTS are free to use. The remaining solutions charge for the amount of characters that are synthesized with IBM Watson charging the most per million characters. The basic pricing plan of Google Cloud Text-to-Speech which only includes basic voices is also the least priciest under its listed competitors. However, an extended version of Google Cloud TTS costs up to 16$ per million characters. [59]

4.3.5.4 Software environment

The paper based solutions were cloned directly from Github on a Linux operating system. Most of these solutions were originally developed in Python 2 or 3 and required certain packages to be installed. Installing these requirements was straightforward and often did not require any other extra configuration. However, installing multiple solutions to a single client caused a versioning problem. Virtual environments can be used to avoid this problem by isolating the packages. [67] Some of the repositories also do not mention how to work with a pretrained model or how to synthesize using one. Because of the general lack of proper documentation, installing and configuring neural network based solutions can be time consuming if not frustrating.

API based solutions however, reduce the complexity of a given solution by providing an endpoint to synthesize speech. Because of this, the software environment could be restricted to any system that can send requests to the API in question, and process its response which ideally, would be an audio file.

4.3.5.5 Final conclusion

The exploration phase has introduced this research to existing Text-to-Speech solutions. A comparative study was conducted afterwards in order to evaluate these solutions. During this study, an experimentation was required as a way to collect measurements that were necessary for evaluation. Tests on naturalness and intelligibility were held using subjects which resulted into MOS scores and percentages. These numbers were successfully implemented in the comparative study.

The research case presented earlier, was interested in finding an alternative Text-to-Speech solution for the Pepper robot. By experimentation and comparison, the Google Cloud Text-to-Speech service was found to score the highest on naturalness and second highest on intelligibility. Being a cloud service, it also avoids the complexity that is involved in installing, configuring or even training a traditional cloned solution. The Google Cloud Text-to-Speech service, in regards to its additional Text-to-Speech features, can therefore be implemented as an alternative.

The next chapter will focus on the implementation of the Google Cloud solution on the Pepper humanoid robot. The research case however, does not restrict the implementation to one particular solution. In addition to the Google Cloud solution, other solutions will be implemented as well such as Merlin and Tacotron.

5 Implementing Text-to-Speech solutions

During the comparative research, a list of Text-to-Speech systems have been evaluated based on a variety of measurements such as naturalness, performance, speed and usage cost. In this chapter, some of these systems are implemented on a client of choice. Depending on the goals of the Text-to-Speech system, the client can differ from a notebook, a workstation, a robot or a combination of these.

Earlier it was assumed that the Text-to-Speech system should at least be compatible with a recent version of Linux. However, most of the Text-to-Speech systems may not have been designed to work on Linux or any other kind of software environment for that matter. This means compatibility issues can emerge during the implementation. The following sections of this chapter will focus on the implementation of a Text-to-Speech system on a given client along with the encountered issues and solutions. The replacement of Pepper's entire smart assistant functionality with the Google Assistant solution can be found in the next chapter.

5.1 Clients

The clients on which the Text-to-Speech systems will be implemented consist of a workstation and the Pepper robot. The workstation is an HP Z220. It consists of an i7-3770 processor, an NVIDIA Quadro K2000 graphics card and 8 GB of RAM. The operating system is a Linux Ubuntu 16.04 LTS. The workstation is a property of PXL University College. During the time of implementation, other workstations were also available to use. But due to convenience, the Z220 was used instead. The client workstation supports both versions of Python 2 and 3. And if necessary, ROS is also supported.

The Pepper robot can be controlled with the Python SDK. This SDK is called PYNAOQI and is compatible with Python 2.7. The 2.1.2.17 version of the PYNAOQI SDK is used. With the SDK, multiple modules of the Pepper can be addressed by using a proxy. The Text-to-Speech module of the Pepper is called “ALTextToSpeech”. By issuing the proxy using the workstation, the ALTextToSpeech module of the pepper can be used. The code snippet below shows how to use this service with only three lines of code, assuming the PYNAOQI SDK on the workstation is working correctly. [68]

```python
from naoqi import ALProxy
tts = ALProxy("ALTextToSpeech", "<IP of your robot>", 9559)
tts.say("Hello, world!")
```

Code snippet 2: Using Pepper's Text-to-Speech module.

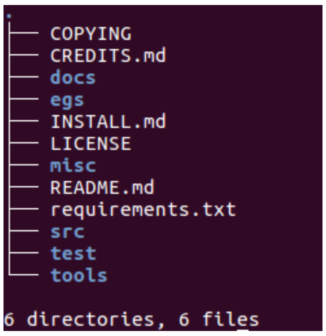
The combination of the workstation and Pepper will occur during implementation. This is because the SDK is typically installed on the workstation and will execute service calls to the proxy of the Pepper robot as mentioned before. In that case the Pepper robot can be referred to as the lightweight client. The actual Text-to-Speech processing or API networking will occur on the workstation back end and the resulting speech samples will be played back on the Pepper robot. The connectivity between the workstation and the Pepper robot is only possible in the same network.

5.2 Merlin Text-to-Speech

As described before, Merlin serves as a free and licensed toolkit to build statistical parametric systems with Deep Neural Networks. The toolkit uses Python and certain dependencies in order to work with neural networks, statistical and audio libraries such as Keras, Theano and Scipy. The Github repository of Merlin is maintained by its developers and contains a fair amount of stars and contributors. [25] A first glance of the repository reveals an organized file structure. The source code is present in the `src` directory with the corresponding tests in a `test` directory at root level. The repository also contains a folder for front end tools, if the default front end tool were not to be used, and the scripts that are necessary to build them.

5.2.1 Overview repository

Documentation is present either on the main README file, in a markdown document at each directory, or in a specific `docs` folder. Refer to Figure 30 for an overview of the root file system. Besides these documentation files, there are several blogposts mentioned to help users with installing Merlin and synthesizing speech. The blogpost used as a guide in this section is the "Getting started with the Merlin Speech Synthesis toolkit" post written by Josh Meyer. It is a lengthy guide that contains a detailed walkthrough of the Github repository. First, a clone of the Github repository is needed. [69]



```
.
├── COPYING
├── CREDITS.md
├── docs
├── egs
├── INSTALL.md
├── LICENSE
├── misc
├── README.md
├── requirements.txt
├── src
├── test
└── tools

6 directories, 6 files
```

Figure 30: An overview of the Merlin repository.

This repository has been cloned and issued on April 4th of 2018. The system that is being used to train and test this repository is the workstation client with the Nvidia Quadro K2000 video card containing 384 CUDA cores [70]. As the training process will reveal, this setup is more than adequate especially with Merlin's compatibility of CUDA. Software wise, the install guide mentions compatibility of Python 2 and 3 versions. The exact version that is currently being used in the training and testing process is Python 2.7.12. The `requirements.txt` file in the repository is available to help with installing the Python dependencies using the pip utility. If pip is not present on the Ubuntu system then it can be installed with: [69] **`$ sudo apt-get install python-pip`**

It might be recommended to use virtualenv during the installation process. It will assure the dependencies are installed on an isolated virtual environment, rather than the whole system. However, in this case it is omitted. [67]

5.2.2 Dependencies

At root, where `requirements.txt` is located, the dependencies are installed using:

```
$ pip -r install requirements.txt
```

This installs all the necessary Python dependencies. There are also a couple of tools provided in the `tools` folder that will need to be compiled. Several scripts are provided in order to compile these tools. The one that is mentioned during the tutorial is the `compile_tools` shell scripts. Executing this script in the directory resulted in an error that advised to install a few packages. This problem was solved by simply doing a "**`$ sudo apt-get install`**" of the mentioned packages. After resolving the issue and executing the script again, the "All tools successfully compiled!" line at the end of the compiling process confirms a successful build. [69]

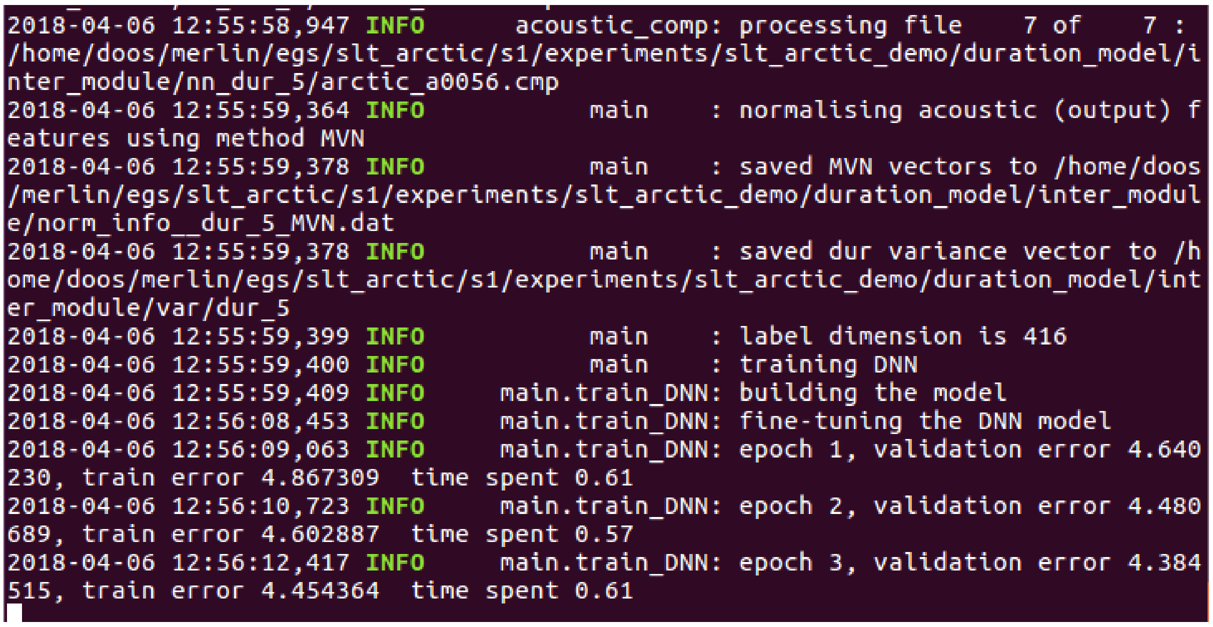
5.2.3 Training and demo synthesis

After installing and compiling the necessary tools and dependencies, a specific demo can be trained and tested. These demos are located in the `egs` directory at root level. The name of this directory stands for “examples”. The `slt_arctic` demo is going to be tested. This demo is trained with the SLT Arctic corpus. Under the `slt_arctic` directory there are two directories. One is called `S1` and uses a WORLD vocoder and the other `S2` which uses a MagPhase vocoder. The guide advises to use the first one so `S1` is chosen to be tested. A tree view the `S1` directory shows a list of multiple directories and scripts. The `conf` directory contains configuration files that define information of the directories, the training data and the architecture of the deep neural networks that will be trained and tested.

Then there is the `scripts` folder in the main `S1` directory which contains scripts for executing the necessary training and testing work of Merlin. There is also the `testrefs` directory. This contains all the training log files done by the development team so that it can be compared if errors were to occur. [69] The tutorial describes two ways to train the neural network model. The script `run_demo.sh` will train the model on 50 utterances and `run_full_voice.sh` trains on 1132 utterances. The one with the most utterances will take the most amount of time to train but result in a better trained model. For the sake of testing, it does not matter which one is used but for a quick training, the `run_demo.sh` is executed.

This script will:

1. Download the `slt_arctic_demo_data.zip` training data.
2. Create the directories for the training data and move the data into this directory.
3. Prepare config files for the training process
4. Train the duration neural network
5. Train the acoustic neural network
6. Synthesize speech using the trained models

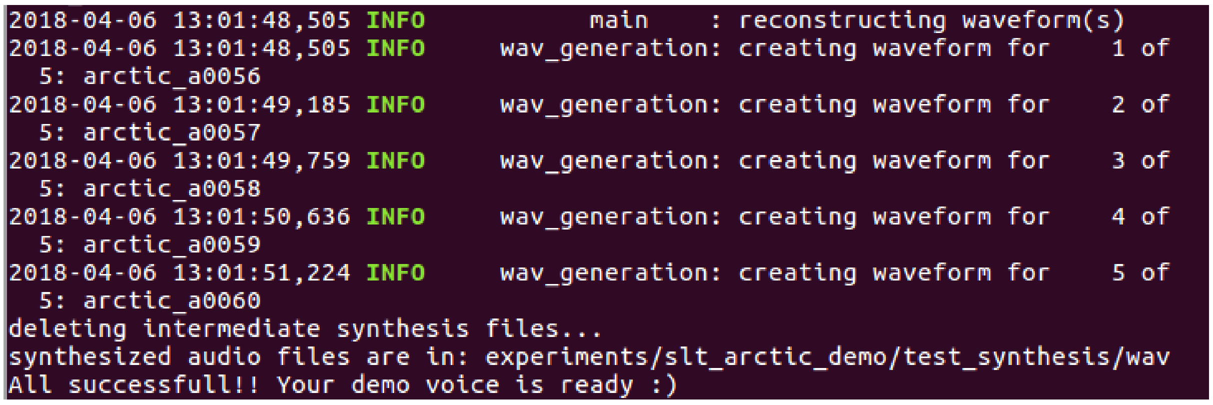


```
2018-04-06 12:55:58,947 INFO      acoustic_comp: processing file    7 of    7 :
/home/doos/merlin/egs/slt_arctic/s1/experiments/slt_arctic_demo/duration_model/i
nter_module/nn_dur_5/arctic_a0056.cmp
2018-04-06 12:55:59,364 INFO           main    : normalising acoustic (output) f
eatures using method MVN
2018-04-06 12:55:59,378 INFO           main    : saved MVN vectors to /home/doos
/merlin/egs/slt_arctic/s1/experiments/slt_arctic_demo/duration_model/inter_modul
e/norm_info__dur_5_MVN.dat
2018-04-06 12:55:59,378 INFO           main    : saved dur variance vector to /h
ome/doos/merlin/egs/slt_arctic/s1/experiments/slt_arctic_demo/duration_model/int
er_module/var/dur_5
2018-04-06 12:55:59,399 INFO           main    : label dimension is 416
2018-04-06 12:55:59,400 INFO           main    : training DNN
2018-04-06 12:55:59,409 INFO     main.train_DNN: building the model
2018-04-06 12:56:08,453 INFO     main.train_DNN: fine-tuning the DNN model
2018-04-06 12:56:09,063 INFO     main.train_DNN: epoch 1, validation error 4.640
230, train error 4.867309  time spent 0.61
2018-04-06 12:56:10,723 INFO     main.train_DNN: epoch 2, validation error 4.480
689, train error 4.602887  time spent 0.57
2018-04-06 12:56:12,417 INFO     main.train_DNN: epoch 3, validation error 4.384
515, train error 4.454364  time spent 0.61
```

Figure 31: Terminal view of the training process.

Merlin trains two models. One is a duration model as shown in Figure 31, in order to predict the phoneme duration and an acoustic model for training on the speech samples. So when input text needs to be synthesized into speech, its duration is predicted using the duration model and used as an input by the acoustic model in order to predict the parameters that are needed for the vocoder to generate waveforms. [25]

Executing `run_demo.sh` will do the necessary training and synthesizing and create a directory `experiments` which houses the training data, the neural network files and synthesized WAV files. A global config file is also created which contains all the directory information of Merlin that is needed during the training and testing phase. [25] This whole process took approximately 5 minutes and as a result generated 5 demo speech samples as shown in Figure 32. These are located in the experiments directory under `slt_arctic_demo/test_synthesis/wav`.



```
2018-04-06 13:01:48,505 INFO           main    : reconstructing waveform(s)
2018-04-06 13:01:48,505 INFO     wav_generation: creating waveform for    1 of
  5: arctic_a0056
2018-04-06 13:01:49,185 INFO     wav_generation: creating waveform for    2 of
  5: arctic_a0057
2018-04-06 13:01:49,759 INFO     wav_generation: creating waveform for    3 of
  5: arctic_a0058
2018-04-06 13:01:50,636 INFO     wav_generation: creating waveform for    4 of
  5: arctic_a0059
2018-04-06 13:01:51,224 INFO     wav_generation: creating waveform for    5 of
  5: arctic_a0060
deleting intermediate synthesis files...
synthesized audio files are in: experiments/slt_arctic_demo/test_synthesis/wav
All successfull!! Your demo voice is ready :)
```

Figure 32: Terminal view of the synthesizing process.

The synthesized wav samples are short and generated based on labels that had been preprocessed by text using a front end tool. The result samples acoustically represent and sound human but are almost impossible to understand. This is because the model has been trained on only 50 utterances which is not enough to generate decent sounding and understandable speech. That is why there is a `run_full_voice.sh` script that will train the models on 1132 utterances. [69]

5.2.4 Custom synthesis

Some configuration is necessary in order to generate speech samples based on custom text. First of all, a front end tool such as Festival has to be compiled. This can be achieved the same way as the tools were compiled but this time using `compile_other_speech_tools` script in the `tools` directory. After executing this script, the festival folder will be created under the tools directory. After building the necessary tools, a folder named `txt` has to be created under the directory `slt_arctic_demo/test_synthesis/` containing text files for each speech sample that should be generated. In order to actually synthesize speech from the text files, the `merlin_synthesis` script has to be executed. This will preprocess the custom text files in the `txt` folder using the Festival front end and generate speech using the trained models. After executing this script the generated speech will be available in the directory `/slt_arctic_demo/test_synthesis/wav`. The WAV file will be named after the name that had been given to the corresponding text file. [69]

However, the festival front end that was compiled earlier seemed to be having broken binaries because it was not built correctly. When compiling the custom text input with festival during custom synthesis, an error was produced that prohibited merlin from synthesizing speech from this input. The issues on Github had mentioned this problem several times but there was no answer suggesting to recompile festival. Some answers suggested to retrain Merlin or install certain dependencies but that clearly was not a solution. Luckily, a different guide found on Github did provide an actual solution to the festival problem. It provides links to the tar packages of festival and the corresponding dependencies. Its dependencies have to be extracted in the mentioned folders within festival.

After recompiling the custom synthesis was reattempted using the `merlin_synthesis` shell script. The sentence in the text file under the text directory was converted to labels after being processed by festival. The speech audio files were generated with the trained neural network and moved to the `WAV` directory. To guarantee that this solution can be reused in the future, it was compressed into a tar file and saved as a backup. Any user can simply extract the tar file, install the dependencies and continue with synthesis at any given time. However, it would be advised to install Merlin from scratch and follow the necessary steps to synthesize speech in order to guarantee the comprehension of the toolkit and Text-to-Speech in general.

5.3 Tacotron

During research, the Tacotron 2 paper introduced an end-to-end Text-to-Speech model that is based on deep neural networks. A correctly working implementation of Tacotron 2 was unfortunately nowhere to be found, meaning the search during this section had to be shifted to Tacotron which as mentioned before is the former of Tacotron 2. There are a dozen implementations available on Github of this paper yet only a select few could be classified as stable working or well maintained. One of few originates from a person called Kyubyong Park. His LinkedIn Profile on the internet states that he is an author of natural language who wrote several books on Korean. Kyubyong is also a researcher of Natural Language Processing, Machine and Deep Learning. [71] It is certain that the mentioned author as shown in Figure 33, possesses all of the necessary skills in order to implement the Tacotron paper.

5.3.1 Overview repository

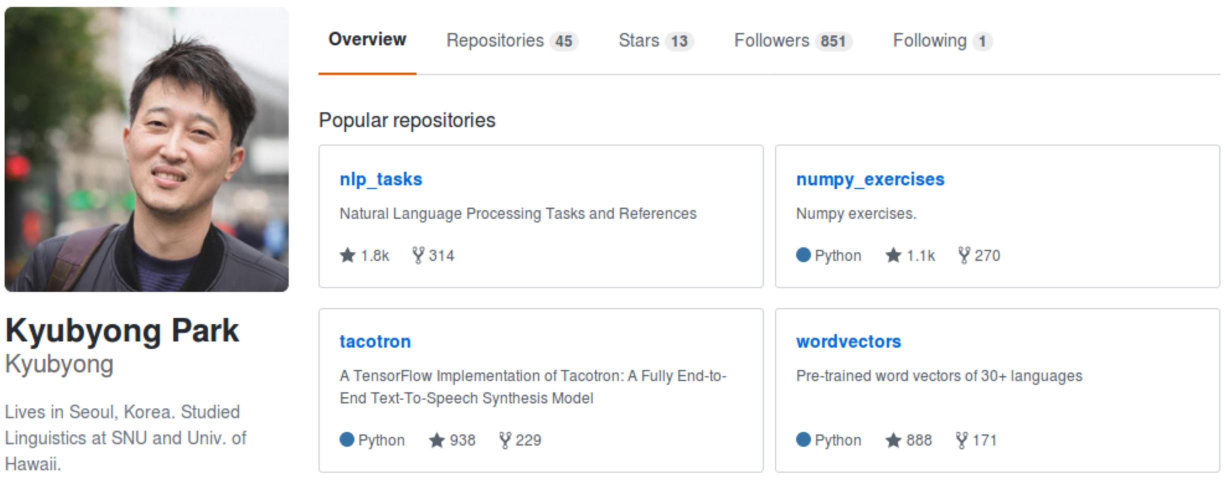

Overview
Repositories 45
Stars 13
Followers 851
Following 1
Popular repositories
Kyubyong Park
Kyubyong
Lives in Seoul, Korea. Studied Linguistics at SNU and Univ. of Hawaii.
nlp_tasks
Natural Language Processing Tasks and References
★ 1.8k 314
numpy_exercises
Numpy exercises.
● Python ★ 1.1k 270
tacotron
A TensorFlow Implementation of Tacotron: A Fully End-to-End Text-To-Speech Synthesis Model
● Python ★ 938 229
wordvectors
Pre-trained word vectors of 30+ languages
● Python ★ 888 171

Figure 33: A Github overview of Kyubyong Park. [72]

Kyubyong's motive to implement Tacotron has saved this research a lot of time that would otherwise be wasted with testing possibly broken Tacotron systems. Also, the Github repository of Kyubyong's Tacotron implementation has received (932 stars as of 3 April 2018). This mainly indicates a work to be superior and a lot more favorable compared to other works with a lot less stars. The work is also well documented which is crucial for implementation and has even been referenced by a few other papers in the field of Text-to-Speech (Tachibana et al., 2017). Kyubyong's Tacotron repository is therefore definitely worth a try.

In order to synthesize speech, the repository has to be cloned. Cloning of the repository is required in order to acquire the necessary source files. This is done with the following command in the Linux terminal, assuming the git tool is installed:

```
$ git clone https://github.com/Kyubyong/tacotron
```

Compared to Merlin, Tacotron's repository consists of a more basic file structure with only one directory containing all of the needed source files. As of training data, the Github tutorial advises three available sources. First there is the LJ Speech Dataset which is publicly available and mentioned to be widely used. It contains of 24 hours of samples. Then there is Nick Offerman's Audiobook which is 18 hours long and at last The World English Bible which is 72 hours in total. [73]

5.3.2 Requirements and dependencies

The Github `README` mentions a description for each file in the root directory as follows: [73]

- **`Hyperparams.py`:** includes all parameters that are needed within the whole training or synthetization process.
- **`Prepro.py`:** Used for preprocessing the training data.
- **`Data_load`:** For loading the training data using queues and mini-batches in parallel.
- **`Utils.py`:** Contains custom functions.
- **`Modules.py`:** Contains functions for encoding and decoding networks.
- **`Networks.py`:** Contains the network functions.
- **`Train.py`:** For the training process.
- **`Eval.py`:** For sample synthesis.
- **`Synthesize.py`:** For synthesizing samples using the "harvard_sentences.txt"

Also because of the lack of a `requirements.txt` file, the following dependencies had to be installed manually with the pip utility [73]:

- NumPy >= 1.11.1
- TensorFlow >= 1.3
- librosa
- tqdm
- matplotlib
- scipy

5.3.3 Pretrained model

In order to train the network, training data of one of the three advised sources has to be downloaded. After downloading the training data, the parameters in hyperparams.py have to be configured if necessary. The hyperparams.py also contains paths which need to be configured depending on the path of the downloaded training data. The train.py file can be executed in order to start training the neural network model. [73] Unfortunately, because the training process of such generative models is intensive as mentioned during the research of generative models, training Tacotron could take a lot of time. By using a pretrained model of someone else, the whole training process can be skipped. However, the pretrained model forces to synthesize speech based on the data that was used training the model. For testing purposes this is not an issue and any pretrained model is welcome.

The `README` file on the Github repository of Kyubyong mentions two pretrained models that can be used with the Tacotron configuration. Both are trained on 200K iterations which may not be enough to generate high quality synthesis. For testing purposes however, this is not an issue.

- LJ: [74]
- World English Bible: [75]

5.3.4 Synthesis

By downloading one of the pretrained models, extracting it and executing the `synthesize.py` file both in the root directory, speech samples can be generated. Note that the pretrained model should be extracted in a directory called `logdir` at root level. This is because the default hyperparameters are configured for that specific directory. A different path would require an edit of the hyperparameters.

After executing `synthesize.py`, 20 samples were generated using the `hardvard_sentences` text file. By editing this file, the generated speech samples can also be customized. The sound samples sounded similar to the Tacotron demo samples in terms of naturalness. However, noise can be heard on the synthesized speech samples. According to similar Github issues, this is because of the quality of the pretrained model.

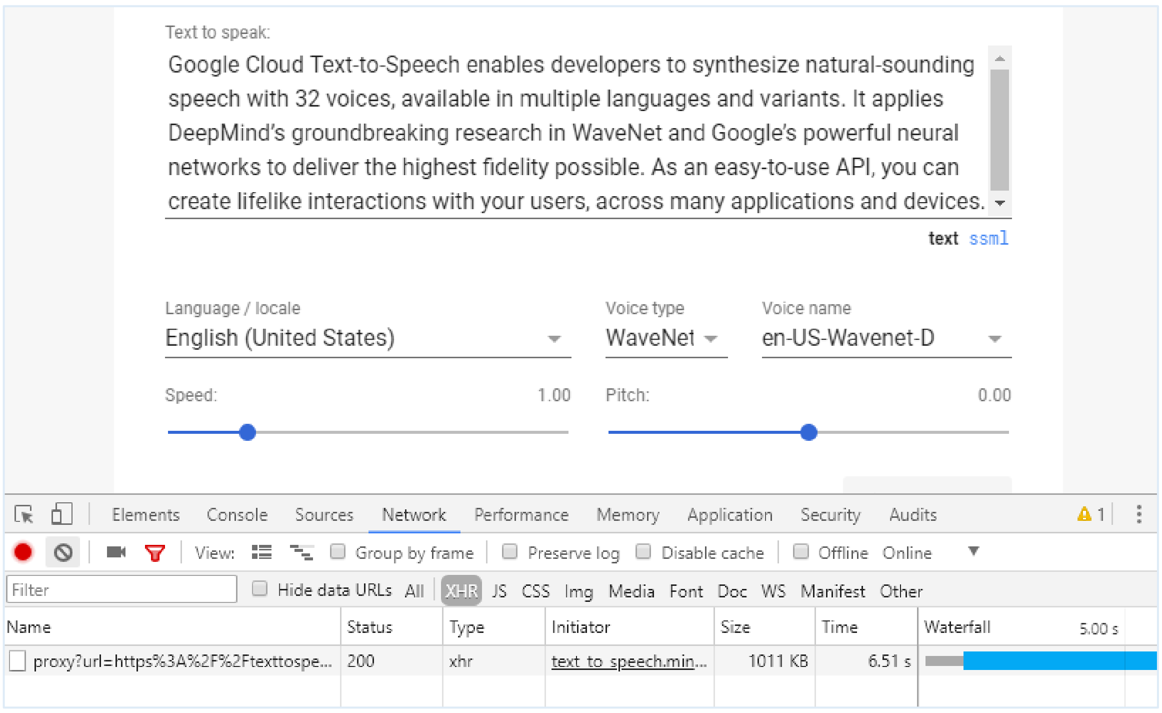
5.4 Google Cloud TTS

The Google Cloud Text-to-Speech is a paid cloud service for synthesizing speech. A Google Cloud Platform account is necessary to try out a costless demo of The Google Cloud Text-to-Speech. However, billing and payment information is required in order to create such an account. This information is necessary to upgrade to a paid account later on. [59] A workaround is required to bypass this step. The webpage where this service is featured contains a demo that does not require any authentication. By reverse engineering the process behind the demo, the API network requests can be mimicked. This is similar to the original demo but wrapping a different context around it rather than Google's webpage. This way, the Text-to-Speech demo of the cloud service can still be showcased in any other software environment that supports REST requests while remaining costless. To verify that this was not a security hole or unlicensed use of the service, an email was sent to Google's Security Bug Report. A reply of the staff confirmed that it was intended for the demo API to be used that way.

5.4.1 Reverse engineering

The reverse engineering process is straight forward. By inspecting the front end source code and switching over to the network tab, outgoing network requests can be monitored. This is shown in Figure 34. When a sample has to be synthesized using several front end controls, a JSON body of the input text is created in combination with prosody information that can be adjusted with the controls. The body will then be sent as a POST request to a certain URL.

The request will return a base64 encoded string that represents the file of the synthesized speech. This is then decoded in the front end and played back with HTML5. However, before the request can be sent on the demo page, a RECAPTCHA appears to verify that the user is not a robot. This is merely a front end utility and can be bypassed by simply not executing the request with the demo Google Cloud Text-to-Speech webpage. If the API request is executed using any other method to execute REST calls, then the RECAPTCHA will not appear.


Text to speak:
Google Cloud Text-to-Speech enables developers to synthesize natural-sounding speech with 32 voices, available in multiple languages and variants. It applies DeepMind's groundbreaking research in WaveNet and Google's powerful neural networks to deliver the highest fidelity possible. As an easy-to-use API, you can create lifelike interactions with your users, across many applications and devices.

text ssml

Language / locale: English (United States)
Voice type: WaveNet
Voice name: en-US-Wavenet-D
Speed: 1.00
Pitch: 0.00

Name	Status	Type	Initiator	Size	Time	Waterfall 5.00 s
proxy?url=https%3A%2F%2Ftexttospe...	200	xhr	text_to_speech.min...	1011 KB	6.51 s	

Figure 34: Reverse engineering Google Cloud Text-to-Speech demo.

5.4.2 Audio playback

Earlier it was mentioned that the Pepper robot features a Python SDK. This SDK can be used to control modules of the Pepper. The API module that is necessary for audio control is the ALAudioDevice. This API is used for several audio operations such as playing back or recording audio using Pepper's hardware. Unfortunately, the documentation of Aldebaran's NAO SDK features C++ only. The SDK however is cross-platform and cross-language, meaning API remains the same for Python. [76] In the documentation, there are two functions mentioned to playback sound as shown below. [77]

```
bool ALAudioDeviceProxy::sendLocalBufferToOutput(const int & nbOfFrames,const
int & buffer)
bool ALAudioDeviceProxy::sendRemoteBufferToOutput(const int & nbOfFrames,const
AL::ALValue & buffer)
```

Code snippet 3: C++ code snippets for playing back audio buffers.

The first function can be used to send a buffer containing audio data to Pepper's audio module. But this can only be done locally on Pepper's operating system assuming the buffered data is available on the Pepper. In most cases, the generation and processing of the audio file of the synthesized speech will occur on a remote device. This means that the data is only available remotely. This has been discussed before. That is why the second function exists. It serves as a way to send buffered audio data to the Pepper on any remote device. The Pepper will then process this data and play it back on its speakers. [77]

To paint the whole picture, a proxy object of the ALAudioDevice module has to be created first. The `sendRemoteBufferToOutput` function of the object can then be called, assuming the settings of the proxy object are correctly configured. These settings are preferences such as the sample rate of the played back audio or the amount of channels of Pepper's speakers to be used. The audio file of the synthesized speech can be buffered in a certain amount of frames. This amount should not exceed 16384 according to the documentation. [77] Each buffer can be provided with the function in combination with the amount of frames that the audio file has been buffered in. Depending on the response of the function and the latency of the network, each buffer of the audio file can now be played back on the Pepper in iteration. [77]

This is one way to playback an audio file on the Pepper. Another way would be to use the ALAudioPlayer module of the SDK. With this module, it is possible to playback a certain audio stream. This is only possible if the audio stream can be accessed with the HTTP protocol. In order to achieve this, the audio fragment must be made accessible with an URL by using a webserver such as Flask. The `playWebStream` method of the ALAudioPlayer module can then be used with the URL of the audio stream as an argument. [78] Refer to the code snippet below for the `playWebStream` method.

```cpp
void ALAudioPlayerProxy::playWebStream(const std::string & streamName, const
float & volume, const float & pan)
```

Code snippet 4: C++ code snippet for playing back an audio stream.

5.4.3 Proof of Concept

5.4.3.1 Flask webserver

Flask is considered as the framework to stream the audio that is necessary for the playback method of the NAO SDK. It is also used for several other server side services such as consuming cloud Text-to-Speech services or processing data. Flask is a web framework for Python. It is implemented as the webserver because it is a RESTFUL microframework that fits well in the software environment of the Pepper SDK or other Google services.

The webserver is installed on Linux by using the pip utility. Afterwards, a Python file including the source code to initialize the Flask webserver is created and executed. This will start and run the Flask webserver on a given IP address and port. [79]

5.4.3.2 Synthesis

First, a back end is created with the Flask webserver. This will serve as an entry point for synthesizing speech samples, streaming these samples and playing back the streamed samples on the Pepper robot. To synthesize speech, a route end point is created. This route accepts a POST request with input text and other parameters as a JSON body. In the background it will send a request to the Google cloud with the reverse engineered request data. The response of this request is an encoded string file. The encoded string is then decoded and written to an MP3 file. This file will be overridden whenever this end point is called. This will make sure that only the last synthesized file exists at all times.

5.4.3.3 Streaming and playback

The second and final goal is to stream the synthesized audio file and play it back on the Pepper robot. To achieve this, a GET end point is created for the audio stream that simply returns the audio file. The full URI of this route is provided as an argument to the method of the ALAudioPlayer module. When this method is executed, the Pepper will play the audio back. For convenience, the method to playback audio with the provided URI stream is also provided with an endpoint.

5.4.3.4 Front end dashboard

In order to provide users an interface to input text, a front end is required. This can be achieved by creating an end point which returns an HTML file with the necessary CSS or Javascript files. In the Javascript source code, AJAX requests of the input text and other Google Cloud Text-to-Speech parameters are sent to the Flask back end in a JSON format. The back end will execute the necessary requests for synthesis. The front end for the demo is built with an HTML file that uses Bootstrap 4 for layout. The rest of the front end business logic and AJAX requests are executed within Javascript.

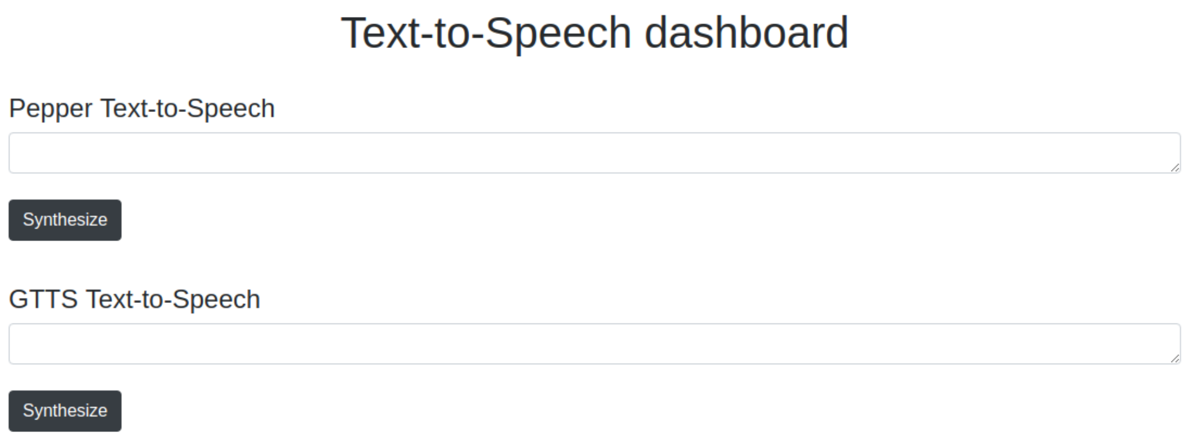
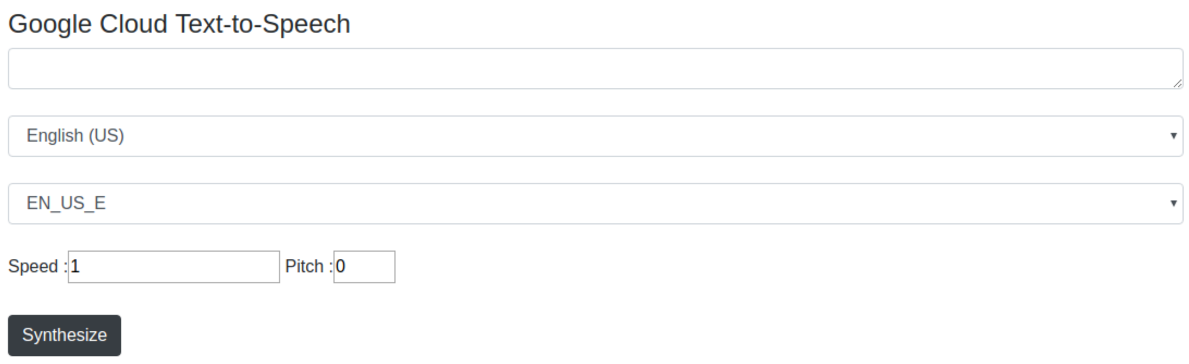


Figure 35: Dashboard for users to synthesize speech with Pepper and GTTS.

Users can synthesize using three different API's as shown in Figure 35 and 36. The first one is Pepper's Text-to-Speech. This will at synthesis use the NOAQI SDK in the back end to create a proxy for the Text-to-Speech module. The input text which is received in the back end as a JSON body, is provided to this module as an argument. The Text-to-Speech module will in result synthesize the input text and play it back on the Pepper automatically. The second way to synthesize speech using the front end is with GTTS. The synthesized audio file is hosted and played back with the Pepper robot in the back end.


Google Cloud Text-to-Speech

English (US)

EN_US_E

Speed : 1 Pitch : 0

Figure 36: Form to synthesize speech using Google Cloud TTS .

Finally, users can synthesize speech using Google Cloud Text-to-Speech. Refer to Figure 36 for the dashboard of this functionality. It accepts more parameters and therefore needs more controls in the front end. Aside from the input text, users can choose what language the synthesized speech should be. For the demo, users can only choose between options English and Dutch even though the API accepts more languages. In combination with the language users can also provide the type of voice of the synthesized speech. This will mostly determine the gender and the dialect of the generated voice. The speed and pitch of the generated speech can also be controlled by providing the desired value in the input box. At synthesis, the back end will work in the same manner as described in previous sections. The synthesized speech will also be played back on the Pepper robot automatically after synthesis.

5.4.3.5 ROS solution

Robot Operating System or ROS is an open source framework that contains libraries and tools for building Robotics applications. It is widely used in the PXL Robotics Lab for several applications and projects. What makes ROS so valuable is that it is modular and allows users to create and distribute functionality across multiple systems in a network. It also provides existing algorithms and drivers that are frequently used in Robotics meaning they can be reused at all times. ROS has a large community composed of researchers, developers and enthusiasts who account for the continuous improvement and development of ROS. [80] It can be useful to wrap the current functionality of this implementation in a ROS package so that it can be used in future Robotics applications. The version of ROS that was originally installed on the workstation client was Kinetic Kame. It is a Long Term Support version that will be supported until 2021. However, a more recent LTS version of ROS has been released called Melodic Morenia in the month May of 2018 which will be supported until 2023. [81]

The LTS version has been considered for this ROS implementation. A package must be created in order to share and organize source code within ROS. Assuming the right version of ROS has been installed, this can be achieved by using the command line tool catkin. These packages are created in a `catkin` workspace which is a workspace created with the catkin tool. [82] The source code within the package is also built using catkin. As previously mentioned, ROS is known for being distributed. It uses “nodes” that are able to compute loosely coupled functionality. This means that each node is responsible for a function and can be executed on separate systems therefore spreading a greater workload over a graph of interconnected nodes. These nodes are able to communicate with each other using a master node that is running at all times. The master node also helps nodes exchange data. For example, a “microphone node” is used for recording data and a “speaker node” for playing back the recorded data, each running on a separate system and interchanging data over the network. Both nodes are able to communicate because a master node is running in the same network. [83]

Nodes use a publisher/subscriber architecture for sending and receiving data. However, sometimes a request/reply interaction can be required. That is why “services” can be used in ROS. Luckily, a node is sufficient for the current ROS implementation as users will be able to publish a given text to the node. The node will then compute the text and playback the synthesized speech on the Pepper robot. In order for this node to receive the input text, it must have a topic. Topics in ROS are necessary for nodes to receive or publish data to each other. In this case, the node will need a topic which it then will subscribe onto. Any incoming data on this topic can be processed within the node. Whenever a user publishes data to the topic, a callback will be executed. The callback will trigger a function that will use the incoming data of the topic to synthesize the speech. Afterwards the synthesized speech will be streamed and played back on the Pepper robot in relation to its hosted URI on Flask. A thread for the Flask server is created and started because otherwise it will block the main thread meaning the node will not start. The source code for this implementation is placed in the `src` folder within the ROS package. [83]

```python
def main():
    rospy.init_node('tts_listener', anonymous=True)
    pepper = rospy.get_param('~pepper_ip')
    host = rospy.get_param('~host_ip')
    port = rospy.get_param('~port')
    play_pepper = rospy.get_param('~play_pepper')
    engine = rospy.get_param('~tts_engine')
    rospy.Subscriber("tts", String, synthesize)
    thread.start_new_thread(start_server, ())
    rospy.spin()
```

Code snippet 5: The main function which starts the node and Flask server

A node in Python is written using the Rospy Python package. In order for python to recognize the package, the PYTHONPATH must contain the ROS `dist_packages` path. Along with the node there are parameters that must be provided such as the IP address of the client computer, the IP address and port of the Pepper robot. Users can also decide whether the synthesized speech should be played back on the robot or locally on the client. The node allows users to synthesize speech using the Google Cloud Text-to-Speech API or the GTTS API. This decision can be made by providing the Boolean as a parameter.

These parameters can be provided with the launch file which simultaneously launches the node. This file as shown in the code snipper below, is created with the `.launch` extension and contains XML. When the package has been built with catkin, users can launch the node using the "**`$ roslaunch`**" commando. [84]

```
<launch>
    <node name="tts_listener" pkg="tts4ros"  output="screen" type="tts.py">
    <param name="host_ip" value="192.168.3.197" />
    <param name="port" value="3000" />
    <param name="pepper_ip" value="192.168.3.146" />
    <param name="play_pepper" value="True" />
    <param name="tts_engine" value="gcloud" />
    </node>
</launch>
```

Code snippet 6: The launch file of the TTS node.

A Github repository for this ROS implementation has been provided by a superior. It is located in the repository of similar Robotics projects with ROS. The repository can be cloned built using the catkin tool. After sourcing the `setup.bash` file in the workspace and configuring the IP addresses and ports in the launch file, users can launch the node with the "**`$ roslaunch`**" commando tool. The topic can be found with "**`$ rostopic list`**" considering the master node has been configured correctly.

6 Google Assistant implementation

The exploration phase has led to the discovery of Google Assistant. In this chapter, it is presented as a solution that incorporates both Text-to-Speech and Speech Recognition. Google Assistant is able to process and reply to personal requests and commands such as “What time is it?”. This solution is presented in order to discover if Pepper’s entire speaking and understanding ability can be replaced with an alternative. A collaboration with Sinasi Yilmaz was necessary in order to develop a Proof of Concept. The final solution uses both Google Assistant SDK and a neural network based hotword detector. Snowboy is used for the latter in order to activate Google Assistant whenever the hotword “Pepper” is heard.

6.1 Installing the SDK

The original guide of Google mentions the necessary steps for installing Google Assistant on a client. In order to work with the SDK, a Google Cloud Platform project has to be created with an existing Google Account. In this project, the Google Assistant API can be enabled and the necessary controls can be activated. This is done in the developers console of the created project. Users can monitor valuable information using this console. Another necessary step for working with the SDK is registering the device on which the assistant will run. This is referred to as the device model which is necessary in later steps. The name “peppermodel” has been given to the device model mainly because the assistant is meant for the Pepper. A different name can be given depending on the choice of the user. [85]

The project contains a corresponding credentials JSON file. This has to be downloaded and moved to the directory that is mentioned during the guide. [86] This file has been moved to the workstation as that is the device which the assistant will run on. Why are these steps executed on the workstation, instead of the Pepper itself? Would that not be more efficient? The reason is because the developers of the Pepper robot do not allow root access on the robot. Certain tools and requirements that are necessary for the Google Assistant SDK can simply not be installed with the lack of root permission. That is one of the reasons why the workstation is used as the client for the Google Assistant.

The last step consists of installing the necessary requirements and tools of Google Assistant. The Python requirements, assuming that Python is the SDK environment of choice, can be installed with pip. Along with the requirements a set of tools are also installed. [87] These Python tools can be used to run Google Assistant on the device and in this case the workstation. However, a different method is used that omits installing the tools as mentioned by the guide. Instead, the source code of the SDK samples are cloned from a Github repository of Google. After cloning the Python samples, the Python requirements are installed manually with pip. This repository has been issued and cloned on May 15th of 2018.

The samples in the repository contain three different modes of assistance. The first one is “hotword”. This will listen for the “OK Google” word and will respond to requests after the hotword has been recognized. This mode will require the hotword at every request unless the conversation requires a follow up answer. In that case the conversation will be ongoing until the assistant runs out of responses. Another mode is the “pushtotalk”. This will require the user to push enter whenever the user wants to input a request for the assistant. [88]

6.2 Running Google Assistant

After the necessary configuration, Google Assistant can be started on the device. This is done by locating the cloned Github repository and executing the Python file of the desired mode of assistance. The repository samples contain a file for each corresponding mode. The file, which is a Python file in this case, has to be executed along with parameters. The amount of parameters that are required depend on the mode. For “pushtotalk”, the user has to provide the “device_model_id” and “device_id” parameter. Earlier, the name “peppermodel” was given to device model. The device ID is created the first Google Assistant is ran on the device. The stream of input and output audio for Google Assistant is provided by the soundcard assuming it is correctly configured. The configuration was not necessary on the workstation as it was working correctly before. However, the install guide mentions methods for configuring the soundcard on Linux or Raspbian operating systems if necessary. Apart from streams, users can also provide file names of the files to be used as input or output. A user can record a certain request beforehand and provide this as the input for the assistant. The output can be written to a file of choice which can then be played back on a different time or location. The last feature is very useful because it enables the output to be played back on the Pepper robot. Refer to the previous chapter where the Pepper was used as a way to playback synthesized speech. In this chapter, the methods for playing back audio on the Pepper remain the same.

6.3 Pushtotalk mode

In hotword mode, the Google Assistant reacts to “OK Google” only. In reality, users are talking directly to the Pepper robot meaning that the original greeting would seem confusing. The hotword must therefore be changed to “Pepper”. To achieve this, the “pushtotalk” module has to be used instead. This allows the user to manually trigger the Google Assistant to start listening for requests meaning the hotword is no longer required. Originally it waits for the user to press the enter button. After the user presses the button, Google Assistant will start listening for requests. The result of the method is a Boolean that determines whether the response of Google Assistant is a follow up conversation or not. If it is a follow up question then the loop will continue and Google Assistant will wait for a follow up request or answer. The developers have provided a variable outside the loop to decide whether Google should run continuously, regardless if the conversations are a follow up or not. The default state of this variable is set to “not once” meaning that Google Assistant will loop until the entire session is killed.

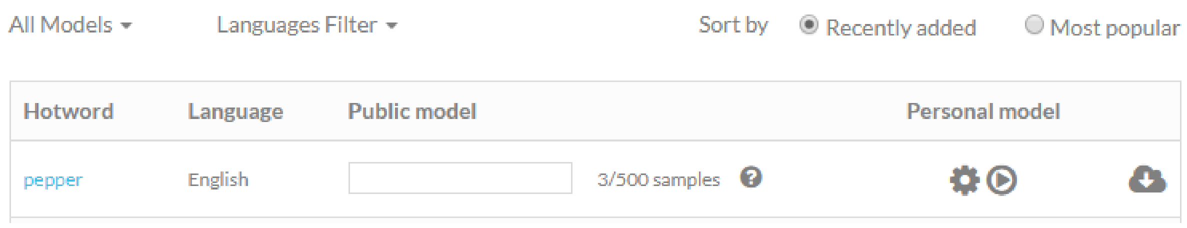
```python
wait_for_user_trigger = not once
    while True:
        if wait_for_user_trigger:
            click.pause(info='Press Enter to send a new request...')
            continue_conversation = assistant.assist()
            wait_for_user_trigger = not continue_conversation
            if once and (not continue_conversation):
                break
```

Code snippet 7: Google Assistant pushtotalk loop.

The previous code snippet is only relevant if audio streams are used. Earlier, it was mentioned that an output file can be provided to write the response audio to as opposed to audio streams. If this option is chosen then Google Assistant will have to generate an audio file each time it has generated a response. This means that the previous code snippet is skipped and can therefore be removed from the pushtotalk.py file. The reason a file is chosen as the output for the response is because it must be hosted on the Flask webserver. That way it can be played back on the Pepper robot similarly to the method used in the previous implementation. The name and extension of the file to which the response will be written to can be provided as a parameter. However in this case, it is hardcoded in the pushtotalk.py code.

6.4 Snowboy hotword detection

As a result, the pushtotalk.py at startup will now listen to a request using the available microphone, generate an audio file of the response and then exit. This is the intended behavior. A tool can be used to trigger this behavior by listening to a custom hotword. Snowboy is used for this purpose. It is a hotword detection engine based on neural networks. Most important of all, it is free for educational purposes and is well supported by community of developers. Snowboy can be trained on a custom hotword. In fact, multiple users can train on one custom hotword to improve its accuracy. For the Pepper robot, the “Pepper” hotword is created. As shown in Figure 37, it was trained on three voice samples. Other users can download the trained model only if it has been trained by 500 users in total. [89]



Hotword	Language	Public model		Personal model
pepper	English		3/500 samples	

Figure 37: The trained "Pepper" hotword model.

Snowboy provides and SDK for multiple software environments. The Python SDK is a C++ wrapper. The Github repository of Snowboy can be cloned and compiled with SWIG to become a working solution. However, precompiled solutions are also available on the internet if compilation results into failure. The SDK also provides a few demo’s as shown in Code snippet 8. The demo requires the trained hotword model. During the demo, Snowboy will continuously listen for the hotword. In the demo, a callback method can be provided that will be executed when the hotword is recognized by Snowboy.

Normally, the method that activates Google Assistant can be provided as the callback method. This is not possible because then Google Assistant will only run once after Snowboy has executed the callback method. This can be solved by running both in iteration sequentially. However, this is only possible by modifying the Snowboy code so that the thread that is created for Snowboy is finished whenever the hotword is recognized. The multithreading solution is presented in the next section.

```
def snowboy():
    global detector
    model = "pepper.pmdl"
    detector = snowboydecoder.HotwordDetector(model, sensitivity=0.5)
    detector.start()
    detector.terminate()
```

Code snippet 8: Modified Snowboy demo with the trained hotword.

6.5 Multithreading solution

In this section, a multithreading solution is presented. This is achieved with the multithreading package in Python. There are two methods that are executed sequentially. One is for activating Google Assistant and the other one is for running Snowboy. The Snowboy thread is started first and by using a .join on this thread, the main thread will wait for this thread to finish. The thread will finish when Snowboy has recognized the custom hotword. After this thread finishes, the Google Assistant thread starts and will start listening for requests. After the assistant has generated a response, the thread finishes and the Snowboy thread will be started again. This is the main loop. However, another loop must be included because conversations within Google Assistant can follow up on each other. This can be determined with the response of the thread. If the response is "True", then the Google Assistant thread should continue listening for requests and generating responses. Otherwise, the thread will finish and Snowboy will be activated back in the main loop.

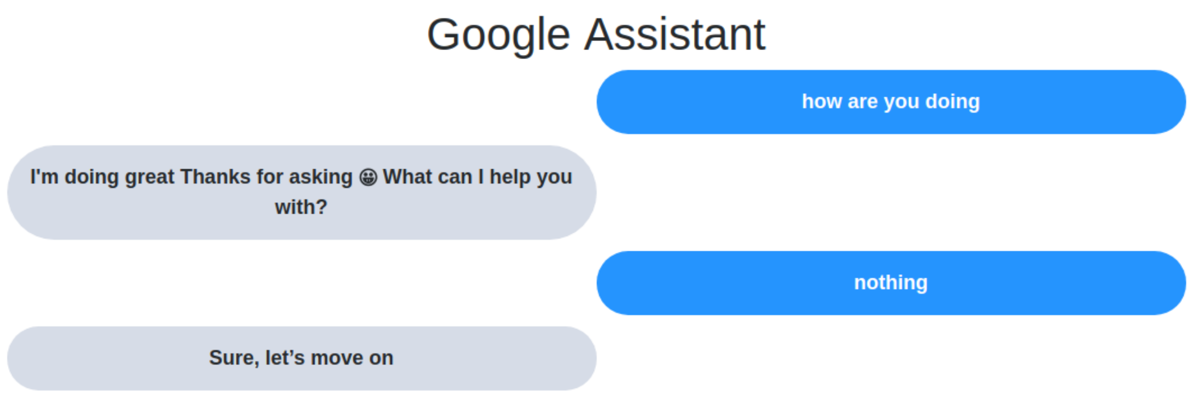
```
while True:
    t1 = threading.Thread(target=snowboy)
    t1.start()
    t1.join()
    while True:
        t2 = threading.Thread(target=assistant)
        t2.start()
        t2.join()
        play_response()
        if not CONT:
            break
```

Code snippet 9: Multithreading iterative solution with Snowboy and Google Assistant.

In the secondary loop, the main threads waits for the Google Assistant thread to finish. When finished, the generated response file will be already be hosted on the Flask webserver. The webserver will simply return the generated response file. The method afterwards will stream the hosted response file on the Pepper robot. Refer to the code snippet above which represents a part of the found solution.

6.6 Front end solution

A front end application was also developed by Sinasi Yilmaz for the Google Assistant solution. The purpose of this application was for users to visualize the dialog flow between Google Assistant and the user requests. Figure 38 is a screenshot of the front end application.


Google Assistant

how are you doing

I'm doing great Thanks for asking 😀 What can I help you with?

nothing

Sure, let's move on

Figure 38: Web page with the dialog of the Google Assistant solution.

6.7 Containerizing with Docker

Docker is a container platform that allows users to containerize new or existing applications. These containers have many benefits such as keeping dependencies isolated. This increases security and portability meaning that containers can be migrated easily to different infrastructures. Docker guarantees that an application will work consistently. This is especially important for automating development pipelines where environments can change constantly. Docker enables separation of concerns and makes sure that application conflicts are eliminated. [90]

An existing Ubuntu 16.04 image is used to containerize the Google Assistant application. In order to achieve this, the image has to be pulled or built using a Dockerfile. The existing Ubuntu image can be pulled with "**`$ docker pull ubuntu:16.04`**". After loading this image, the image must be run in a container. However, a few additional parameters are required as the standard way of running is not sufficient. First of all, the host IP address of the client, which the docker container is running on, must be used in order for Flask to host the necessary audio files. Also, some of the audio drivers and configurations are necessary for the docker container to be able to record using the microphone of the host client. As a result, the container is started using "**`$ sudo docker run --name assistant --device /dev/snd --network host -it ubuntu:16.04`**". This will run the Ubuntu image with the name "assistant".

First, the Google Assistant and Snowboy solution for the Pepper are cloned from Github. Afterwards, the Python dependencies of the solution are installed using pip. However, some additional packages must be installed on the container for Snowboy to work such as libatlas. [89] PYNAOQI is installed as previously done during the implementation.

The configuration file that is provided with the solution in the running container must also be configured with the correct IP addresses in order for the solution to work. After installing the solution on the container and assuring that it works correctly, the container can be saved as an image and exported as a tar file. Other users can simply load the tar file into any existing Docker configured client and run the image in a container. Within the container, users will still have to configure the IP settings and run the Python application manually. After running the application in the container, clients can access the front end application using the configured host IP address.

7 Conclusion

One of the objectives of this thesis was to conduct a literature study of Text-to-Speech. This study introduced the thesis to fundamental subjects such as speech synthesis, computational linguistics, signal processing and even Artificial Intelligence. Another objective was to find an alternative Text-to-Speech solution for the Pepper robot, which was formerly lacking in naturalness. This was achieved by conducting a comparative study and through experimentation using certain evaluation techniques.

The literature study started with an introduction and historical overview of speech synthesis and Text-to-Speech. The fundamental parts of linguistics were studied afterwards, before diving in to the details of the Text-to-Speech architecture. This was done in order to achieve a better understanding of both the text and speech representation. Also, it was essential to know about the verbal and prosodic components of communication. Eventually, the architecture of a typical Text-to-Speech system was discussed, along with a few approaches on how to synthesize speech from a given text. The focus during the thesis lied on the generative approach, in which speech was synthesized with the use of neural networks.

During the exploration phase, multiple generative Text-to-Speech solutions were found and discussed. These solutions have been compared by conducting a comparative study using measurements such as naturalness and intelligibility. The purpose of this study was to find an alternative which could replace Pepper's original Text-to-Speech solution and therefore improve its naturalness. The findings of the comparative study have led to the implementation of the Google Cloud Text-to-Speech service. Both research and experimentation have proven this solution to be superior. In the end, a final solution was proposed as a Proof of Concept which utilizes both the cloud service and the SDK of the Pepper robot.

Bibliographical references

8 Bibliography

[1] „Smart ICT," [Online]. Available: https://www.pxl.be/SmartICT.html. [Accessed 6 May 2018].

[2] „PXL Smart ICT," [Online]. Available: https://italent.samycoenen.be/img/logo/logo_pxl_Smart-ICT.png. [Accessed 7 June 2018].

[3] T. Dupont, „PXL Robotics Lab," [Online]. Available: https://prl.pxl.be/. [Accessed 6 May 2018].

[4] „Corda Campus," [Online]. Available: https://www.cordacampus.com/sites/all/themes/cordacampus/images/ogimage.jpg. [Accessed 7 June 2018].

[5] „PXL Robotics Lab," [Online]. Available: https://prl.pxl.be/web/image/465/received_10208666934789509-01.jpeg. [Accessed 15 May 2018].

[6] R. Mannell, „Introduction to Speech and Language Technology," [Online]. Available: http://clas.mq.edu.au/speech/synthesis/history_slp/index.html. [Accessed 21 April 2018].

[7] V. U. Brussel, „Alexander Graham Bell," [Online]. Available: http://educinno.intec.ugent.be/geluiddruk/verkl_Alexander_Graham_Bell.htm. [Accessed 7 June 2018].

[8] K. Eschner, „Meet Pedro the "Voder," the First Electronic Machine to Talk," [Online]. Available: https://www.smithsonianmag.com/smart-news/meet-pedro-voder-first-electronic-machine-talk-180963516/. [Accessed 7 June 2018].

[9] „The 'Voder' & 'Vocoder' Homer Dudley, USA,1940," [Online]. Available: http://120years.net/the-voder-vocoderhomer-dudleyusa1940/. [Accessed 21 April 2018].

[10] P. Taylor, Text-to-Speech Synthesis, University of Cambridge: University of Cambridge, 2009.

[11] R. Vaughan, „WATCH Stephen Hawking announce the Global Teacher Prize shortlist," [Online]. Available: https://www.tes.com/us/news/breaking-news/watch-stephen-hawking-announce-global-teacher-prize-shortlist. [Accessed 21 April 2018].

[12] Y. M. Yaniv Leviathan, „Google AI Blog | Google Duplex," Google, [Online]. Available: https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html. [Accessed 8 June 2018].

[13] C. Woodford, „How speech synthesis works," [Online]. Available: http://www.explainthatstuff.com/how-speech-synthesis-works.html. [Accessed 21 April 2018].

[14] O. Watts, „Unsupervised Learning for Text-to-Speech Synthesis," The University of Edinburgh, The University of Edinburgh, 2012.

[15] M. Hanlon, „Stephen Hawking chooses a new voice," [Online]. Available: https://newatlas.com/go/2708/. [Accessed 8 June 2018].

[16] „Laurel vs. Yanny," [Online]. Available: https://www.popsci.com/sites/popsci.com/files/styles/655_1x_/public/images/2018/05/yanny_laurel-100.jpg?itok=acjkMde2&fc=50,50. [Accessed 26 May 2018].

[17] „Communication and Types of Communication," [Online]. Available: http://www.notesdesk.com/notes/business-communications/types-of-communication/. [Accessed 21 April 2018].

[18] H. Zen, „Generative Model-Based Text-to-Speech Synthesis.," MIT, Londen, 2017.

[19] „Levels Of Language," [Online]. Available: https://www.uni-due.de/SHE/LevelsOfLanguage-Graph.gif. [Accessed 21 April 2018].

[20] T. E. o. E. Britannica, „Morpheme | linguistics | Britannica.com," [Online]. Available: https://www.britannica.com/topic/morpheme. [Accessed 21 April 2018].

[21] E. J. Vajda, „Linguistics 201," [Online]. Available: http://pandora.cii.wwu.edu/vajda/ling201/test1materials/syntax.htm. [Accessed 21 April 2018].

[22] „What does semantics study?," [Online]. Available: http://all-about-linguistics.group.shef.ac.uk/branches-of-linguistics/semantics/what-does-semantics-study/. [Accessed 21 April 2018].

[23] „Why do we need orthography?," [Online]. Available: http://www.mpi.nl/q-a/questions-and-answers/why-do-we-need-orthography. [Accessed 21 April 2018].

[24] „The corpus and Oxford Dictionaries," [Online]. Available: https://en.oxforddictionaries.com/explore/the-oxford-english-corpus. [Accessed 21 April 2018].

[25] O. W. S. K. Zhizheng Wu, „Merlin: An Open Source Neural Network Speech Synthesis System," Sunnyvale, CA, USA, 2016.

[26] R. Mannell, „TTS Input: Plain text and other input formats," [Online]. Available: http://clas.mq.edu.au/speech/synthesis/tts_input/index.html. [Accessed 21 April 2018].

[27] A. Mikheev, „Learning Part-of-Speech Guessing Rules from Lexicon:," HCRC Language Technology Group, University of Edinburgh, Edinburgh EH8 9LW, Scotland, UK, 1996.

[28] „The Yanny or Laurel Debate," [Online]. Available: https://www.newstatesman.com/2018/05/yanny-or-laurel-debate-reveals-more-about-our-brains-our-ears. [Accessed 8 June 2018].

[29] J. Brownlee, „Want help with algorithms? Take the FREE Mini-Course.," [Online]. Available: https://machinelearningmastery.com/linear-regression-for-machine-learning/. [Accessed 21 April 2018].

[30] K.-T. L. H. L. Minghui Dong, „A Unit Selection-based Speech Synthesis Approach," Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore 119613.

[31] S. King, „A beginners' guide to statistical parametric speech synthesis," University of Edinburgh, UK, 2010.

[32] J. Yamagishi, „An Introduction to HMM-Based Speech Synthesis," 2006.

[33] S. D. H. Z. Aäron van den Oord, „WaveNet: A Generative Model for Raw Audio," Google DeepMind, [Online]. Available: https://deepmind.com/blog/wavenet-generative-model-raw-audio/. [Accessed 21 April 2018].

[34] C. S. a. D. Siganos, „Neural Networks," [Online]. Available: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html. [Accessed 14 May 2018].

[35] „Neural Networks," [Online]. Available: https://blog.webkid.io/content/images/old/neural-networks-in-javascript/nn_blog.png. [Accessed 9 June 2018].

[36] „A Beginner's Guide to RNN's and LSTM's," DL4J, [Online]. Available: https://deeplearning4j.org/lstm.html. [Accessed 14 May 2018].

[37] „Recurrent Neural Networks," [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png. [Accessed 15 May 2018].

[38] A. v. d. Oord, „WAVENET: A GENERATIVE MODEL FOR RAW AUDIO," Google, London, UK, 2016.

[39] T. L. Paine, „FAST WAVENET GENERATION ALGORITHM," University of Illinois, IBM Thomas J. Watson Research Center, 2016.

[40] A. v. d. Oord, „Parallel WaveNet: Fast High-Fidelity Speech Synthesis," 2017.

[41] Y. Wang, „TACOTRON: TOWARDS END-TO-END SPEECH SYNTHESIS," Google, Inc, 2017.

[42] J. Shen, „NATURAL TTS SYNTHESIS BY CONDITIONING WAVENET ON MEL SPECTROGRAM," Google, Inc., University of California, Berkeley, 2018.

[43] R. Skerry-Ryan, „Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron," 2018.

[44] R. Tikofsky, „Tikofsky's 50-word Intelligibility Test," [Online]. Available: http://www.ucs.louisiana.edu/~ncr3025/roussel/codi555/tikofsky.htm. [Accessed 2 May 2018].

[45] „Hardvard Sentences," [Online]. Available: http://www.cs.columbia.edu/~hgs/audio/harvard.html. [Accessed 2 May 2018].

[46] N. Barnes, „Science Code Manifesto," [Online]. Available: http://sciencecodemanifesto.org/. [Accessed 2 May 2018].

[47] K. Walk, „How to Write a Comparative Analysis," 1998. [Online]. Available: https://writingcenter.fas.harvard.edu/pages/how-write-comparative-analysis. [Accessed 2 May 2018].

[48] „Python SDK," Aldebaran, [Online]. Available: http://doc.aldebaran.com/1-14/dev/python/index.html. [Accessed 21 April 2018].

[49] S. Rajak, „API Latency vs Response Time," [Online]. Available: https://medium.com/@sanjay.rajak/api-latency-vs-response-time-fe87ef71b2f2. [Accessed 9 June 2018].

[50] „Python SDK," Aldebaran, [Online]. Available: http://doc.aldebaran.com/1-14/dev/python/install_guide.html#python-install-guide. [Accessed 21 April 2018].

[51] Y. L. I. B. Aäron van den Oord, „High-fidelity speech synthesis with WaveNet," Google DeepMind, [Online]. Available: https://deepmind.com/blog/high-fidelity-speech-synthesis-wavenet/. [Accessed 21 April 2018].

[52] „About stars," Github, [Online]. Available: https://help.github.com/articles/about-stars/. [Accessed 21 April 2018].

[53] S. O. Arık, „Deep Voice: Real-time Neural Text-to-Speech," Baidu Silicon Valley Artificial Intelligence Lab, 1195 Bordeaux Dr. Sunnyvale, CA 94089, 2017.

[54] S. Ö. Arık, „Deep Voice 2: Multi-Speaker Neural Text-to-Speech," Baidu Silicon Valley Artificial Intelligence Lab, 1195 Bordeaux Dr. Sunnyvale, CA 94089, 2017.

[55] W. Ping, „DEEP VOICE 3: SCALING TEXT-TO-SPEECH WITH CONVOLUTIONAL SEQUENCE LEARNING," Baidu Research, 2018.

[56] IBM, „Text-to-Speech - IBM Cloud," IBM, [Online]. Available: https://console.bluemix.net/catalog/services/text_to_speech. [Accessed 21 April 2018].

[57] Amazon, „Amazon Polly," Amazon, [Online]. Available: https://aws.amazon.com/polly/. [Accessed 21 April 2018].

[58] deparkes, „Python Text To Speech," [Online]. Available: https://deparkes.co.uk/2017/06/30/python-text-speech/. [Accessed 21 April 2018].

[59] „Cloud Text-to-Speech," Google, [Online]. Available: https://cloud.google.com/text-to-speech/. [Accessed 8 May 2018].

[60] E. Betters, „What is Google Assistant, how does it work, and which devices offer it?," January 2018. [Online]. Available: https://www.pocket-lint.com/apps/news/google/137722-what-is-google-assistant-how-does-it-work-and-which-devices-offer-it. [Accessed 21 April 2018].

[61] „Google Home," [Online]. Available: https://sitecdn.tvpage.com/live/users/1758166/canvas/320/179/785552264-01320180.jpg. [Accessed 21 April 2018].

[62] „Alexa Voice Service Device SDK," Amazon, [Online]. Available: https://developer.amazon.com/alexa-voice-service/sdk. [Accessed 21 April 2018].

[63] „Response Bias," [Online]. Available: http://www.statisticshowto.com/response-bias/. [Accessed 10 June 2018].

[64] „MIT License," [Online]. Available: https://choosealicense.com/licenses/mit/. [Accessed 16 May 2018].

[65] „Apache License 2.0," [Online]. Available: https://choosealicense.com/licenses/apache-2.0/. [Accessed 16 May 2018].

[66] „Legal notices," Aldebaran, [Online]. Available: http://doc.aldebaran.com/2-1/legal/notice.html. [Accessed 16 May 2018].

[67] „Installing Python Modules," [Online]. Available: https://docs.python.org/2/installing/index.html. [Accessed 21 April 2018].

[68] „Using the API," Aldebaran, [Online]. Available: http://doc.aldebaran.com/1-14/dev/python/making_nao_speak.html. [Accessed 8 May 2018].

[69] J. Meyer, „Getting started with the Merlin Speech Synthesis Toolkit," [Online]. Available: http://jrmeyer.github.io/tts/2017/02/14/Installing-Merlin.html. [Accessed 21 April 2018].

[70] „Datasheet Quadro K2000," NVIDIA, [Online]. Available: https://www.nvidia.com/content/PDF/data-sheet/DS_NV_Quadro_K2000_OCT13_NV_US_LR.pdf. [Accessed 21 April 2018].

[71] „Kyubyong Park," LinkedIn, [Online]. Available: https://www.linkedin.com/in/kyubyongpark/. [Accessed 7 June 2018].

[72] „Kyubyong Github," [Online]. Available: https://github.com/Kyubyong/. [Accessed 21 April 2018].

[73] T. M. Kyubyong Park, „A TensorFlow Implementation of Tacotron: A Fully End-to-End Text-To-Speech Synthesis Model," [Online]. Available: https://github.com/Kyubyong/tacotron. [Accessed 21 April 2018].

[74] „LJ_logdir.zip," [Online]. Available: https://www.dropbox.com/s/8kxa3xh2vfna3s9/LJ_logdir.zip?dl=0. [Accessed 21 April 2018].

[75] „WEB_logdir.zip," [Online]. Available: • : https://www.dropbox.com/s/g7m6xhd350ozkz7/WEB_logdir.zip?dl=0. [Accessed 21 April 2018].

[76] „Naoqi Framework," Aldebaran, [Online]. Available: http://doc.aldebaran.com/1-14/dev/naoqi/index.html#remote-modules. [Accessed 9 May 2018].

[77] „ALAudioDevice API," Aldebaran, [Online]. Available: http://doc.aldebaran.com/2-1/naoqi/audio/alaudiodevice-api.html. [Accessed 9 May 2018].

[78] „ALAudioPlayer API," Aldebaran, [Online]. Available: http://doc.aldebaran.com/2-1/naoqi/audio/alaudioplayer-api.html. [Accessed 9 May 2018].

[79] „Flask," [Online]. Available: http://flask.pocoo.org/. [Accessed 9 May 2018].

[80] „ROS.org," [Online]. Available: http://www.ros.org/is-ros-for-me/. [Accessed 23 May 2018].

[81] „ROS/Installation," [Online]. Available: http://wiki.ros.org/ROS/Installation. [Accessed 23 May 2018].

[82] „create_a_workspace," [Online]. Available: http://wiki.ros.org/catkin/Tutorials/create_a_workspace. [Accessed 23 May 2018].

[83] „ROS/Concepts," ROS, [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed 26 May 2018].

[84] „Roslaunch," [Online]. Available: http://wiki.ros.org/roslaunch. [Accessed 26 May 2018].

[85] „Configure a Developer Project," Google, [Online]. Available: https://developers.google.com/assistant/sdk/guides/library/python/embed/config-dev-project-and-account. [Accessed 15 May 2018].

[86] „Registering the Device Model," Google, [Online]. Available: https://developers.google.com/assistant/sdk/guides/library/python/embed/register-device. [Accessed 15 May 2018].

[87] „Install the SDK and Sample Code," Google, [Online]. Available: https://developers.google.com/assistant/sdk/guides/library/python/embed/install-sample. [Accessed 15 May 2018].

[88] „Run the Sample Code," Google, [Online]. Available: https://developers.google.com/assistant/sdk/guides/library/python/embed/run-sample. [Accessed 15 May 2018].

[89] „Snowboy, a Customizable Hotword Detection Engine," Kitt-ai, [Online]. Available: http://docs.kitt.ai/snowboy/. [Accessed 17 May 2018].

[90] „Docker," [Online]. Available: https://www.docker.com/. [Accessed 30 May 2018].