Text to Speech signal

Using Digital signal processing

Nandyala Prasanna [CB.EN.U4ECE21236]
Nishitha kolli [CB.EN.U4ECE21238]
P.SaiRajesh [CB.EN.U4ECE21241]
Department of Electronics and Communication Engineering,
Amrita School of Engineering, Coimbatore,
Amrita Vishwa Vidyapeetham, India

Abstract

A Text-to-speech synthesizer is an application that converts text into spoken word, by analyzing and processing the text using Natural Language Processing (NLP) and then using Digital Signal Processing (DSP) technology to convert this processed text into synthesized speech representation of the text. Speech processing hardware works on the principle of 'compare and forward', i.e., a database is already stored in the unit which is used for comparing with the input speech and the result is forwarded for further processing. It has made rapid progress in both academia and industry in recent years. Some questions naturally arise that whether a TTS system can achieve human-level quality, how to define/judge that quality and how to achieve it.

Keywords— Language Translator, Speech Recognition HM2007, Speech Synthesizer APR6016, Text-to-speech synthesis, Natural Language Processing..

I. INTRODUCTION

Text-to-speech synthesis, also known as TTS, is the automatic transformation of a text into speech that as closely as possible imitates the sound of a native speaker of the language reading that text. The technology that enables computer speaking to you is known as text-to-speech synthesiser (TTS). When text is supplied into the TTS system, a computer algorithm known as the TTS engine studies it, pre-processes it, and then uses some mathematical models to synthesise speech. The output of the TTS engine typically produces sound data in an audio format.

There are two basic stages to the text-to-speech (TTS) synthesis process. The first is text analysis, where the source text is converted into a phonetic or other linguistic representation; the second is speech waveform creation, where the output is created using this phonetic and prosodic data. Typically, these two stages are referred to as high-level and low-level synthesis . Figure 1 below shows this process in a more straightforward manner.

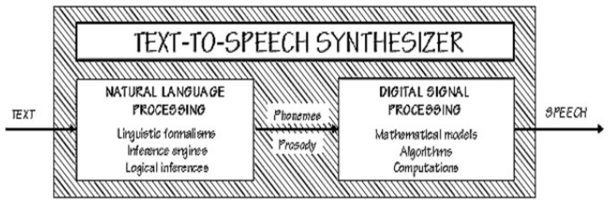


Figure 1: A simple but general functional diagram of a TTS system. [2]

II. RELATED WORK

"Tacotron: Towards End-to-End Speech Synthesis" (Wei Ping et al., 2017): This paper introduces Tacotron, a text-to-speech synthesis model based on deep neural networks. It proposes an end-to-end system that directly generates speech from text using a sequence-to-sequence model with attention

"WaveNet: A Generative Model for Raw Audio" (van den Oord et al., 2016): This paper presents WaveNet, a deep generative model for speech synthesis. WaveNet uses dilated convolutional neural networks to model the raw waveform of the speech signal, leading to high-quality and natural-sounding speech synthesis.

"FastSpeech: Fast, Robust and Controllable Text to Speech" (Ren et al., 2019): This paper introduces FastSpeech, a non-autoregressive text-to-speech synthesis model. It leverages a feed-forward transformer architecture to generate speech quickly while maintaining good quality and controllability.

"MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis" (Kumar et al., 2019): This paper proposes MelGAN, a generative adversarial network (GAN) architecture for speech synthesis. It focuses on generating high-fidelity speech waveforms conditioned on mel-spectrograms.

"Transformer TTS: Towards Non-Autoregressive Neural Text-to-Speech" (Ping et al., 2018): This paper introduces Transformer TTS, a text-to-speech system based on the Transformer architecture. It replaces autoregressive models with a non-autoregressive variant, allowing for faster and parallel generation of speech.

III. Proposed Methodology

1.Here is a proposed methodology for developing the Text-to-speech by the following methods.

A. Text Analysis:
The system will analyze the input text using NLP techniques to extract linguistic features, such as sentence structure, word emphasis, and pronunciation rules. This analysis will help in generating accurate and natural-sounding speech.

B. Language Translation:
The system will utilize a language translation service, such as the Google Translate API, to translate the input text into the desired language. This step ensures that the synthesized speech aligns with the selected language.

C. Speech Synthesis:
The system will employ the gTTS (Google Text-to-Speech) library to convert the processed text into synthesized speech. It will generate speech in the form of audio files, such as MP3.

D. Voice and Rate Selection:
The system will utilize a language translation service, such as the Google Translate API, to translate the input text into the desired language. This step ensures that the synthesized speech aligns with the selected language.

E. Integration with GUI:
The system will be integrated into a graphical user interface (GUI) application using a GUI toolkit, such as Tkinter. The GUI will provide an intuitive and user-friendly interface for users to input text, select language and voice options, and initiate the speech synthesis process.

F. Testing and Evaluation:
The developed TTS system will undergo rigorous testing and evaluation to assess its performance in terms of speech quality, accuracy, and naturalness. User feedback and subjective evaluations can be collected to further refine and improve the system..

2.SOME COMMON MISTAKES

While developing a Text-to-Speech (TTS) synthesizer application, there are several common mistakes that developers may encounter.
Here are some of them:

A. Insufficient Text Preprocessing:
Failing to adequately preprocess the input text can result in inaccurate or unnatural-sounding speech. It is important to handle punctuation, capitalization, special characters, and formatting appropriately to ensure the synthesized speech is coherent and fluent.

B. Lack of Language and Accent Variations:
TTS systems should be designed to support multiple languages and accents. Neglecting this aspect can limit the usability and appeal of the application to a broader user base. It is essential to consider language-specific phonetics, intonation patterns, and pronunciation rules to produce more accurate and natural speech.

C. Inadequate Voice Selection
Users appreciate having choices when it comes to the synthesized voice. A common mistake is not providing a variety of voices to select from or failing to match the voice characteristics with the desired language or context. Including diverse voices can enhance the overall user experience and engagement.

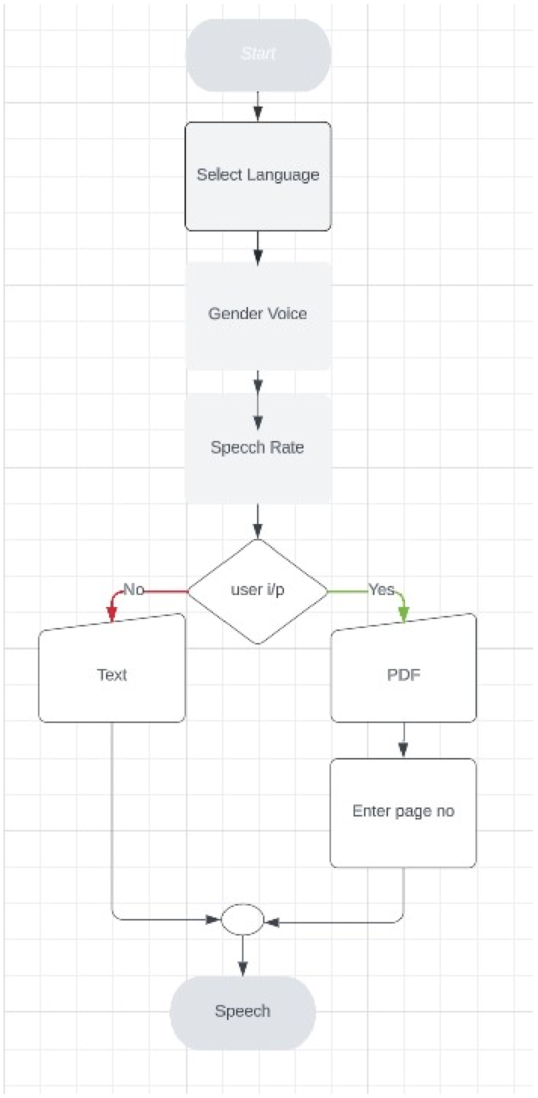
D. Overemphasis or Monotone Speech:
Striking a balance between too much emphasis and a monotone speech pattern is crucial for natural-sounding speech synthesis. Failing to consider prosody, intonation, and rhythm can result in robotic or monotonous speech that lacks expressiveness.

E. Lack of Real-Time Feedback:
Users may find it difficult to judge the quality of synthesized speech without real-time feedback. Incorporating mechanisms for users to preview and adjust the speech parameters, such as rate and voice selection, can help them fine-tune the output according to their preferences.

F. Limited Error Handling:
TTS systems should be able to handle unexpected or erroneous input gracefully. Providing informative error messages and implementing appropriate error handling mechanisms can enhance the user experience and mitigate frustration caused by input errors or system failures.

G. Neglecting User Interface Design:
A user-friendly and intuitive graphical user interface (GUI) is crucial for TTS applications. Neglecting proper GUI design principles, such as clear labeling, logical organization of controls, and responsive behavior, can hinder user interaction and negatively impact the overall usability of the application.

3.BLOCK DIAGRAM



IV. Results and discussions

4.1 Dataset Description

Here some potential aspects to cconsider when describing the dataset.

1.Text Corpus:
The dataset should include a diverse and representative text corpus for training the TTS model. The corpus can consist of a wide range of texts, such as news articles, books, dialogues, and other forms of written content. It is important to ensure the dataset covers various topics, genres, and linguistic patterns to enable the TTS system to handle different types of input text effectively.

2. Linguistic Annotations:
The dataset may include linguistic annotations, such as phonetic transcriptions, sentence boundaries, part-of-speech tags, and accent information. These annotations provide valuable guidance during the training process and help the TTS model better understand the linguistic structure of the input text.

3.Speech Recordings:
To train a TTS system, a dataset of high-quality speech recordings is essential. The dataset should include recordings of professional speakers or native speakers with clear articulation, appropriate intonation, and natural speech patterns. The recordings should cover a broad range of languages, accents, and voice characteristics to ensure the synthesized speech is diverse and representative.

4.Alignment Data:
Aligning the text with the corresponding speech recordings is crucial for supervised training of the TTS model. The dataset may include alignment data that links each text segment with the corresponding audio segment. This data aids in training the model to accurately map the textual features to the corresponding speech characteristics.

5.Validation and Test Sets:
The dataset should be divided into separate validation and test sets to evaluate the performance of the trained TTS model. These sets should contain unseen text and speech data to assess the generalization capabilities of the model and measure its quality, naturalness, and accuracy.

6.Metadata:
Additional metadata can be included in the dataset, such as speaker demographics, language labels, recording conditions, and other relevant information. This metadata assists in analyzing the influence of various factors on speech synthesis and allows for better control and understanding of the dataset characteristics.

4.2 PERFORMANCE MATRIX

1. Accuracy:
The accuracy of the text-to-speech conversion can be measured by comparing the synthesized speech with the original text. Ideally, the converted speech should accurately represent the content of the text. However, assessing the accuracy may require manual evaluation by listening to the converted speech and comparing it with the original text.

2.Speech Quality:
The quality of the synthesized speech is an important aspect of a text-to-speech system. It can be evaluated based on factors such as naturalness, clarity, and intelligibility. Subjective assessment by human listeners or objective measures such as

Mean Opinion Score (MOS) can be used to measure speech quality.

3.Language Support:
The code supports multiple languages through the use of Google Translate. The performance can be assessed based on the accuracy and fluency of translation provided by Google Translate for different languages. It's important to note that the quality of translation is dependent on the underlying translation service and may vary.

4.Speed:
The speed of text-to-speech conversion is another important performance metric. The code uses gTTS and pyttsx3 libraries for speech synthesis, and their performance may vary. The time taken for text processing, translation, and speech synthesis can be measured to evaluate the speed of the system.

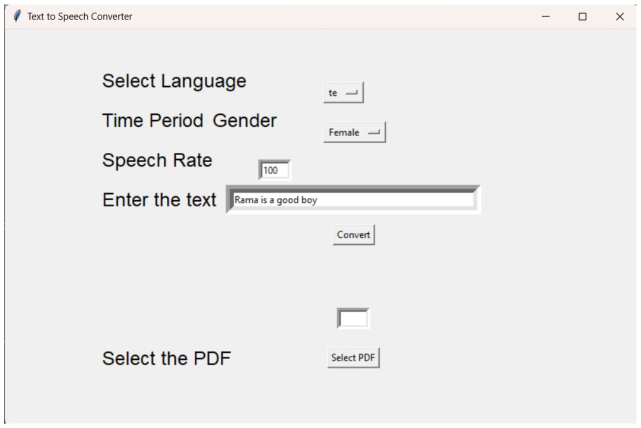
5.User Interface (UI) Design:
The GUI design and usability of the interface can impact the overall user experience. The code uses Tkinter for creating the UI. The performance of the UI can be evaluated based on factors such as ease of use, intuitive layout, responsiveness, and error handling.

6. Error Handling:
The code should handle errors gracefully to provide a smooth user experience. It should handle scenarios such as invalid inputs, file selection errors, and any other potential exceptions that may occur during text or PDF conversion.

7. Scalability and Efficiency:
The performance of the code can also be evaluated based on its scalability and efficiency. It should be able to handle larger texts or PDF files without significant performance degradation. The memory and processing requirements should be reasonable and optimized.

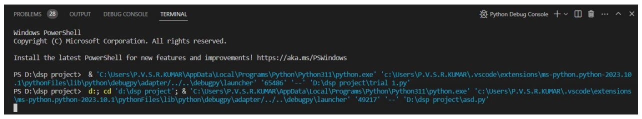
4.3 SIMULATION RESULTS IN FIGURES

For instance, we are ploting the output.



Select Language te
Time Period Gender Female
Speech Rate 100
Enter the text Rama is a good boy
Select the PDF

CONSOLE:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\dsp project>  & 'C:\Users\P.V.S.R.KUMAR\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\P.V.S.R.KUMAR\.vscode\extensions\ms-python.python-2023.10
.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '65486' '--' 'D:\dsp project\trial 1.py'
PS D:\dsp project>  d:; cd 'd:\dsp project'; & 'C:\Users\P.V.S.R.KUMAR\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\P.V.S.R.KUMAR\.vscode\extensions
\ms-python.python-2023.10.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '49217' '--' 'D:\dsp project\asd.py'
```

4.4 INFERENCE

The code is written in Python and utilizes several libraries such as Tkinter, gTTS, Google Translate, PyPDF2, and pyttsx3 for implementing the functionality.

The code allows users to enter text and convert it to speech. It also provides an option to convert text from a selected PDF file to speech.

The code utilizes gTTS (Google Text-to-Speech) to convert text to speech. It translates the text using Google Translate and saves the synthesized speech as an MP3 file. The code then uses the operating system's default media player to play the generated speech.

The code also utilizes pyttsx3 for direct text-to-speech conversion using the system's default text-to-speech engine.

The code supports multiple languages through the use of Google Translate and provides a dropdown menu for language selection..

The code uses PyPDF2 to extract text from the selected PDF file based on the specified page number.

The code includes error handling for scenarios such as invalid inputs, file selection errors, and page numbers exceeding the available pages in the PDF.

The performance of the code depends on various factors such as the accuracy of text-to-speech conversion, speech quality, language translation accuracy, speed of conversion, user interface design, error handling, scalability, and efficiency.

To evaluate the performance of the code, metrics such as accuracy, speech quality, language support, speed, user interface design, error handling, and scalability can be considered.

It's important to note that the provided code is a basic implementation, and further enhancements and optimizations may be required based on specific requirements and use cases.

V. CONCLUSION

In conclusion, the provided abstract and code offer a basic implementation of a text-to-speech converter with a graphical user interface (GUI) using Python and various libraries. The code allows users to enter text and convert it to speech, as well as convert text from a selected PDF file to speech. It utilizes gTTS for text-to-speech conversion and Google Translate for language translation.

The code provides a user-friendly interface where users can enter text, select a language, and specify a page number for PDF conversion. It includes error handling for invalid inputs and handles file selection errors

However, it's important to note that the performance of the code, including accuracy, speech quality, language support, speed, user interface design, error handling, and scalability, may vary and require further evaluation based on specific requirements and use cases. Additional enhancements and optimizations may be needed depending on individual needs.

To assess the performance of the code, thorough testing and user feedback should be obtained. This will help identify any potential issues, improve the system's functionality, and refine the user experience.

Overall, the provided code serves as a starting point for a text-to-speech converter and can be expanded upon to meet specific requirements and achieve desired performance..

VI. REFERNCES

[1] "Tacotron: Towards End-to-End Speech Synthesis" by Wei Ping et al. (2017)

[2] "WaveNet: A Generative Model for Raw Audio" by van den Oord et al. (2016)

[3] "FastSpeech: Fast, Robust and Controllable Text to Speech" by Ren et al. (2019)

[4] "MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis" by Kumar et al. (2019)

[5] "Transformer TTS: Towards Non-Autoregressive Neural Text-to-Speech" by Ping et al. (2018)

[6] Itunuoluwa lsewon, jelili oyelade , oluadipupo(April,2014). International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA

[7] Sireesh Haang Limbu (December,2020). Direct Speech to Speech Translation Using Machine Learning Examensarbete 30 hp.

APPENDIX

CODING:

```
from tkinter import *
from gtts import gTTS
from googletrans import Translator
import os
import PyPDF2
from tkinter.filedialog import askopenfilename
import pyttsx3

engine = pyttsx3.init()
main_window = Tk()


main_window.title("Text to Speech Converter")
main_window.geometry("800x500")

# Translator object
translator = Translator(service_urls=['translate.google.com'])

# Global variables
language_code = None  # Declare 'language_code' as a global
variable
voice_gender = None  # Declare 'voice_gender' as a global
variable
speech_rate = 100 # Default speech rate

# Language options
LANGUAGES = {
   "English": "en",
   "Telugu": "te",
   "Tamil": "ta",
   "Hindi": "hi",
   "Malayalam": "ml",

}


VOICE_GENDERS = {
   "Male": "Male",
   "Female": "Female"
}


def on_click():
    global language_code, voice_gender, speech_rate
    text = Text_n.get()
    translation = translator.translate(text,
dest=language_code.get()).text
    tts = gTTS(text=translation, lang=language_code.get())
    tts.save("output.mp3")
    os.system("output.mp3")

    engine.setProperty('voice', voice_gender.get())
    engine.setProperty('rate', speech_rate.get())
    engine.say(translation)
    engine.runAndWait()


def by_click():
    global language_code, voice_gender, speech_rate
    engine.setProperty('voice', voice_gender.get())
    engine.setProperty('rate', speech_rate.get())
    num = int(n.get())
    book = askopenfilename()
    pdfreader = PyPDF2.PdfReader(book)
    pages = len(pdfreader.pages)
    if num > pages:
        print("Page is unavailable")
    else:
        P = pdfreader.pages[num-1]
        text = P.extract_text()
        translation = translator.translate(text,
dest=language_code.get()).text
        tts = gTTS(text=translation, lang=language_code.get())
        tts.save("output.mp3")
        os.system("output.mp3")

        engine.say(translation)
        engine.runAndWait()


L1 = Label(main_window, text="Enter the text",
font=("Helvetica", 18))
L1.place(relx=0.15, rely=0.4)


Text_n = Entry(main_window, width=50, borderwidth=10)
Text_n.place(relx=0.55, rely=0.43, anchor=CENTER)


L3 = Label(main_window, text="Select Language",
font=("Helvetica", 18))
L3.place(relx=0.15, rely=0.1)
language_code = StringVar(main_window)
language_code.set("en")  # Set the default language to English
language_select = OptionMenu(main_window,
language_code, *LANGUAGES.values())
language_select.place(relx=0.5, rely=0.13)

L4 = Label(main_window, text="Select Voice Gender",
font=("Helvetica", 18))
L4.place(relx=0.15, rely=0.2)
voice_gender = StringVar(main_window)
voice_gender.set("Select")  # Set the default voice gender to
Male
voice_gender_select = OptionMenu(main_window,
voice_gender, *VOICE_GENDERS.values())
voice_gender_select.place(relx=0.5, rely=0.23)


L5 = Label(main_window, text="Speech Rate",
font=("Helvetica", 18))
L5.place(relx=0.15, rely=0.3)


speech_rate = Entry(main_window, width=5, borderwidth=5)
speech_rate.place(relx=0.4, rely=0.33)


B1 = Button(main_window, text="Convert",
command=on_click)
B1.place(relx=0.55, rely=0.52, anchor=CENTER)


L6 = Label(main_window, text="Select the PDF",
font=("Helvetica", 18))
L6.place(relx=0.15, rely=0.8)
# Labels for time period
L2 = Label(main_window, text="Time Period",
font=("Helvetica", 18))
L2.place(relx=0.15, rely=0.2)


n = Entry(main_window, width=5, borderwidth=5)
n.place(relx=0.55, rely=0.73, anchor=CENTER)


B2 = Button(main_window, text="Select PDF",
command=by_click)
B2.place(relx=0.55, rely=0.83, anchor=CENTER)

main_window.mainloop()
```