TECHNICAL UNIVERSITY OF CRETE
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT
TELECOMMUNICATIONS DIVISION

Gesture Recognition for Sentence Synthesis with Probabilistic Algorithms

by

Evangelos Giannelos

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DIPLOMA OF

ELECTRICAL AND COMPUTER ENGINEERING

July 2018

THESIS COMMITTEE

Associate Professor Aggelos Bletsas, *Thesis Supervisor*
Associate Professor Katerina Mania
Associate Professor Panagiotis Partsinevelos

Abstract

The purpose of this work was to develop a standalone system that predicts text and outputs speech synthesis for people with physical disabilities and reduced fine motor function. A low-cost, infrared hand tracking device was utilized, typically used in virtual/augmented reality applications. The implemented system offers two basic functionalities. First, a human-computer interaction (HCI) functionality that is customizable and offers enough features that can be tailored to a large number of different users. Second, a way to predict what the user intends to write in order to reduce the user's input text as much as possible, speeding up overall communication. The word prediction task was mainly based on n-grams (and relevant Markov Chains), abbreviation expansion (and lookup tables) and recurrent neural networks. Evaluation results confirmed that the simple prediction methods utilized accelerated word typing speed, saving input text by approximately 63%. Future work could utilize additional eye-tracking or hand-tracking sensors.

Thesis Supervisor: Associate Professor Aggelos Bletsas

Acknowledgements

First of all I would like to thank my supervisor Prof. Aggelos Bletsas for his continued guidance, support and understanding throughout his courses, this work and beyond that.

Mrs. Eleanna and Miss Katerina for their collaboration and inspiration regarding this work.

My friends and colleagues at the Telecommunications's lab G. Vougioukas, T. Topalis, K. Skivalakis, V. Karaterakis and E. Ouroutzoglou for all the help they provided during my thesis and for participating on my evaluations.

My family, my close friends and especially Ardit N., Antreas P., Antreas S. and Manos A. for their moral support, university related collaboration and most of all, all the fun we had throughout these years.

Contents

Table of Contents 4

List of Figures 6

1 Introduction 7
1.1 Problem Description 7
1.2 Prior Art 8

2 System Design 10
2.1 Sensor - Leap Motion 10
2.2 User Interface 13
2.2.1 Stages of Development 14
2.3 Speech Synthesis 24

3 Algorithms 26
3.0.1 Corpora 26
3.0.2 Tools 27
3.1 Approaches 28
3.1.1 Using Abbreviations to Minimize Input 28
3.1.2 Markov Chains and N-Grams 31
3.1.3 Neural Networks - LSTMs 37

4 Evaluation - Results 40
4.0.1 Introduction 40
4.1 Evaluation 42
4.1.1 UI 42
4.1.2 Leap Motion 43

4.1.3 Prediction 43
4.1.4 Text-To-Speech 44
4.1.5 Statistics 44
4.1.6 Overall 45
4.1.7 Results 46

5 Conclusions 47
5.1 Conclusion 47
5.2 Future Work 47

6 Appendix 49

Bibliography 52

List of Figures

2.1 The Leap Motion sensor, banknote for size reference 10
2.2 The Leap Motion sensor range. [1] 11
2.3 A snapshot captured from the Leap Motion sensor while in operation. 12
2.4 The initial version of the User Interface with the virtual keyboard 14
2.5 The second version of the User Interface with four selection options 15
2.6 The selection process of the character “Z” in UI version 2 . . . 16
2.7 Here some word predictions from the prefix “πορτοκ” can be seen 17
2.8 A visualization of the different areas in the process of option selection 18
2.9 Different Hands Models for Leap Motion 18
2.10 The rearranged Zones and adjustment options in version 3 . . 20
2.11 The rearranged Zones in version 3 20
2.12 The Final Look of the UI in Version 4 21
2.13 An example of word completion (a) and next word suggestion (c) in the final version of the UI 23
2.14 The process for the Text-To-Speech conversion and playback . 25

3.1 Long Short-Term Memory Network Structure 38

4.1 The Evaluation and Testing Environment 41

Chapter 1

Introduction

1.1 Problem Description

The task at hand is to give a working solution for text generation and speech synthesis to people that are unable to write or talk due to certain disabilities. This work is focused on individuals with physical and not mental disabilities. These people do not have the ability to talk or the fine motor skills required to write due to congenital or extrinsic conditions. The most common and known condition is ALS disease, which mainly causes necrosis of the nerve cells responsible for controlling voluntary muscle movements. For such an individual, additionally losing the ability to communicate can be devastating, it is the medium through which we connect with others, express our feelings and share our ideas. It is a primal need which our evolution and development is based on.

A person derived from social interactions, but conscious and fully aware of their environment, is isolated from human contact and along with being dependent on somebody else for basic human needs, can make the person feel imprisoned in their own body and lose interest in life itself. Obviously, this is catastrophic for the person themselves as well as for the people close to them. So, this work focuses on giving back to those in need a communication medium, the ability to write and share their ideas and feelings with others. All this in a relatively efficient and completely autonomous way.

1.2 Prior Art

As it will later be explained, the system consists of different elements for which research had to be done individually and cumulatively. Begining with the subject of Human-Computer Interaction (HCI) for people with physical disabilities, many studies and technologies have been developed that follow different approaches. The method that requires the least physical capabilities from that user and that has been proposed by many different researchers [2–5] is based on *ElectroEncephaloGraphy* (EEG) and their interfases are named Brain-Computer-Interfaces (BCI). While this is a viable method that usually requires no physical capabilities, it sometimes depends on intrusive procedures involving electrodes being implanted on the user's head. Non-intrucsive alternatives exist but also require head mounted devices like the Emotiv Epoc but its effectiveness and accuracy are reduced [6]. The other problem with this approach is that test have shown that there is a high variability between subjects. The classification accuracy of such systems is around 76% with an average communication speed of 46 bits/min, which is equivalent to writing 8.8 error-free letters per minute [7].

Another category of methods used for HCI, that is being increasingly researched recently due to Virtual-Reality/Augmented-Reality development, is motion tracking and gesture recognition [8]. According to [9], this method, along with others, has a big potential because of its similarity with the natural way humans interact with each other (sight, sound, touch). A specific method/device for hand-tracking and gesture-based HCI is the Leap Motion Controller [10]. This device shows potential in this field, is being used with many AR/VR applications and has been studied in [11–13] for its usage and capabilities. Other solutions that are gaining popularity are eye-tracking/gaze based products like some AR/VR headsets or the non-head-mounted Tobii eye tracker and others [14, 15]. A more low-cost approach is implemented using web cameras to track the movement of pupil and detect eye-blinks to make selections [16]. The tests conducted in this work concluded to an average writing rate of 19 seconds/character while its accuracy/success rate can reach 94.75% .The above prior art is indicative, and

by no means, complete.

Lastly, research has been conducted using Electromyography (EMG). *Sing2Talk* [17] is such a system and uses a wrist band with electrodes to detect signals produces by muscle movement. Using these signals, the software can detect certain gestures that are matched with sings of the American Sign Language (ASL). These signs correspond to a number of phonemic components and enable the user to compose words and sentences. The user's sentences are then converted to speech. It is important to note that this system offers a two way communication since it can listen to speech and display the signs that make up that sentence.

As for the NLP task of text prediction, it is a subject that has been greatly reasearched but still new advancements are made. The current state-of-the-art methods use Deep Neural Networks with word-level architectures [18]. Other NN architectures are also used and being researched, like character-level architectures [19] that excel in storage requirements and simplicity. Other methods face text prediction as a classification task [20, 21]. Another intresting method that requires a dictionary is abbreviation expansion [22]. A statistical approach expands the Markov Chains to language models creating the *N-grams* [23, 24] while incorporating Part-Of-Speech information created POS-grams [25]. Countless different approaches have been taken to resolve this task as well as variations or combinations of some of them [26].

Chapter 2

System Design

The system under design consisted of three different components: the Human-Computer Interaction system (sensor-display-UI), the text prediction engine and the Text-To-Speech synthesizer. The development stages and decision making for each individual component as well as the procedure of combining them together into one system is described in detail in this and the next chapter.

2.1 Sensor - Leap Motion

The sensor chosen for the user to interact with the system is the Leap Motion, a hand tracking device that is portable and accurate enough for our needs. Specifically, the whole devise measures at 7.9 cm in width, 3.0 cm in depth, 1.3 cm in height and weighs about 45 grams (Fig. 2.1).

Figure 2.1: The Leap Motion sensor, banknote for size reference

The specs claim that it has a sub-millimeter accuracy which for this application is more than enough. Indeed, the device proved to be very accurate and fast based on its use (a more dedicated investigation on the sensor's accuracy was conducted by [27]). The sensor is able to identify at least two hands in a conical area above it which spans at least 60 cm in width (120°-150°angle) and 60 cm in height as can be seen in Fig. 2.2.

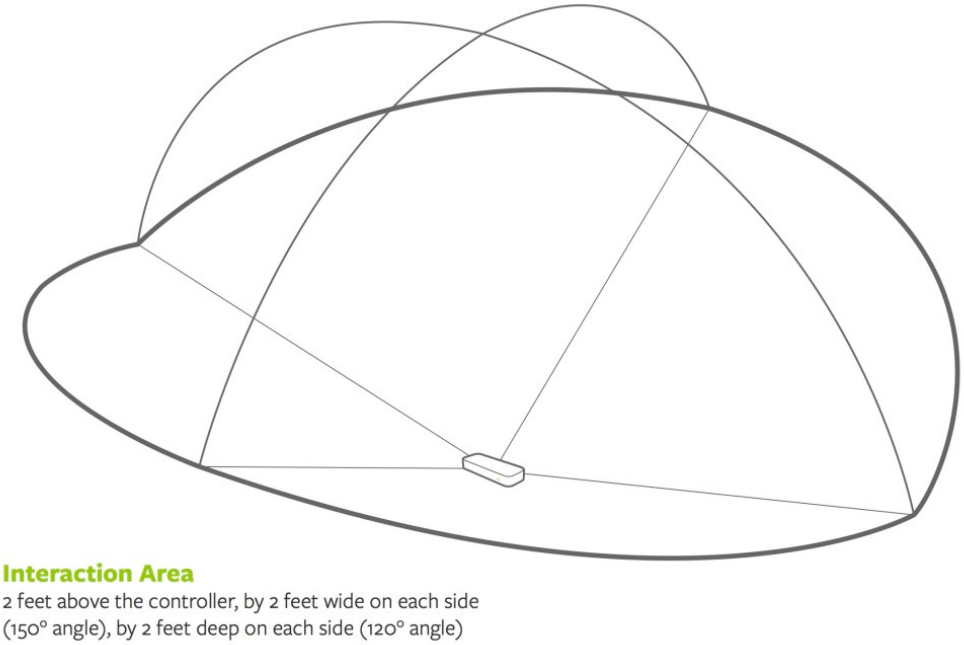


Figure 2.2: The Leap Motion sensor range. [1]

In this space, the sensor is able to specify the relative location and rotation of the hands and their parts in reference to its position. Also, it is capable of recognizing different gestures like swiping, making circles, taping motions (vertical or horizontal) and combinations of them. All of these information provided by the sensor can be used to customize and fit the user interaction environment to the specific capabilities of many different individuals.

In order to detect all these useful information, the sensor uses a quite simple method, infrared emitters and cameras are placed inside the device that track light with a wavelength of 850 nanometers. This makes the device perfect for low light conditions or indoor environments, where the ambient

light does not interfere with the sensors. The device can also be used outdoor but depending on the conditions the IR noise from other sources may affect its accuracy and functionality. Fig. 2.3 is a snapshot from the device while in use indoors; at the top part of the picture, small flickers are shown caused by the laboratory's fluorescent lights that can sometimes emit IR light. As it can be seen though, the detection of the hands is not affected at all for such low interference.

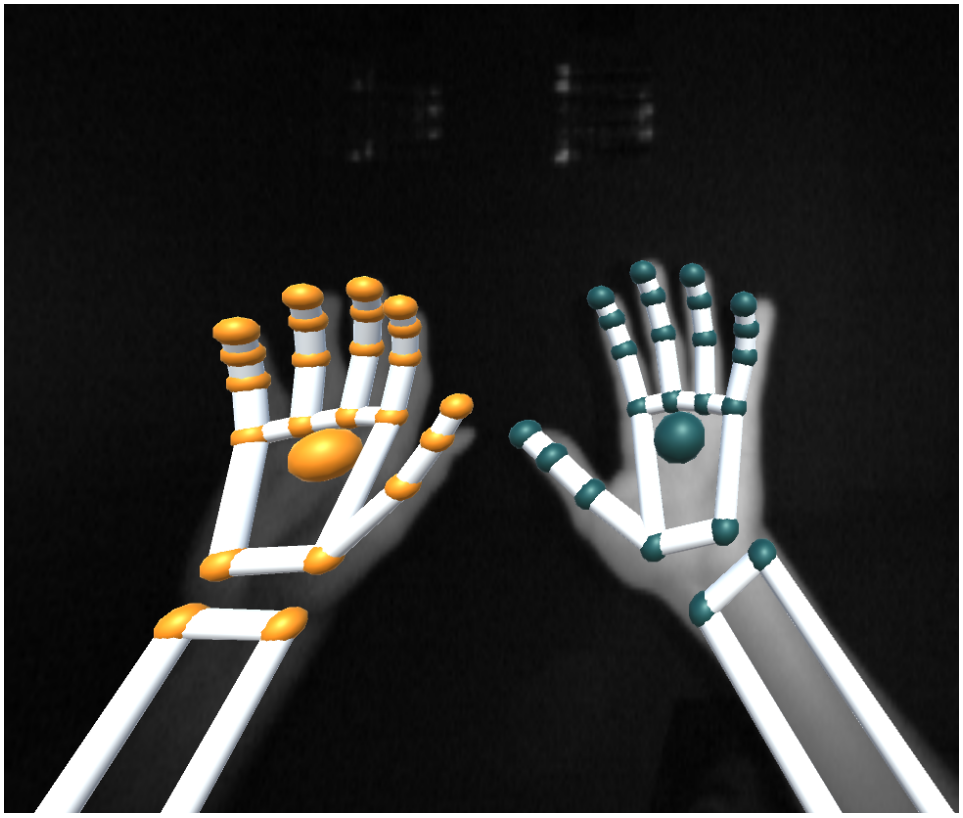


Figure 2.3: A snapshot captured from the Leap Motion sensor while in operation.

Overall, the use of the Leap Motion sensor and its Unity API was very pleasing because of its stable performance and (for the most part) its comprehensive and well-organized documentation. There was some conflict though with some of the features that are mentioned in the documentation and the actual features in the API; this is issue is thought to be associated with bad versioning.

2.2 User Interface

The user interface for this application was developed in *Unity 2017-2018* using C# and *Leap Motion's Unity Assets* for the *Orion Beta*. Unity is a cross-platform game engine that can be used to create 2D or 3D games and applications. It was chosen to be the development platform due to Leap Motion's extensive API and integration support, as well as its ease of use when creating 3D graphical user interfaces.

Before jumping into developing the UI for this application it was essential and very helpful to check out some of the applications developed by the community of Leap Motion in order to learn about any specific features and uses of the device, as well as to get familiar with its use and behavior. Such applications can be found through the "Leap Motion App Home" store application as well as on Leap Motion's online app store [28].

The next step was to brush up on our Unity skills since development in the Unity environment had been done before for other applications. In order to do this the *Unity User Manual* [29], the *Unity Scripting Manual* [30] as well as the *Unity Answers Forum* [31] were a great help throughout the process of development thanks to their comprehensive descriptions of the inner workings of Unity and the answers provided for a number of issues that arose.

The final step was to get familiar with the Leap Motion API and test out the device's performance and capabilities as well as its limitations. In order to do this, the Leap Motion's intergraded visualizer was used (Fig. 2.3 was captured with this software) along with a testing application to fiddle around with the sensor gesture detection and general usage. In this step, the API's documentation provided was very useful [32], as well as support from the community forum [33].

2.2.1 Stages of Development

Version 1

The first stage and version of the UI was more of a test to the Leap Motion's accuracy and capabilities but it differed from the previous tests as it was used to actually type text. It consisted of just a virtual keyboard that the user could tap the keys using their fingers or palms to tap the different keys in order to write sentences.

Another input method was also implemented that was inspired by Google's swipe keyboard feature; the user could point with their index finger in order to start a trace originating from the tip of the finger. The user could then swipe over the characters that their word consisted of and then open their palm again to signal the end of the word. The system then would try to guess what the word is according to the trace created by the user.

Both of these methods though are not very useful since the user that this work is aimed for does not possess the fine motor skill required to accurately tap or swipe over the right characters. Using such input methods would just frustrate the users and not yield the efficiency and result intended.

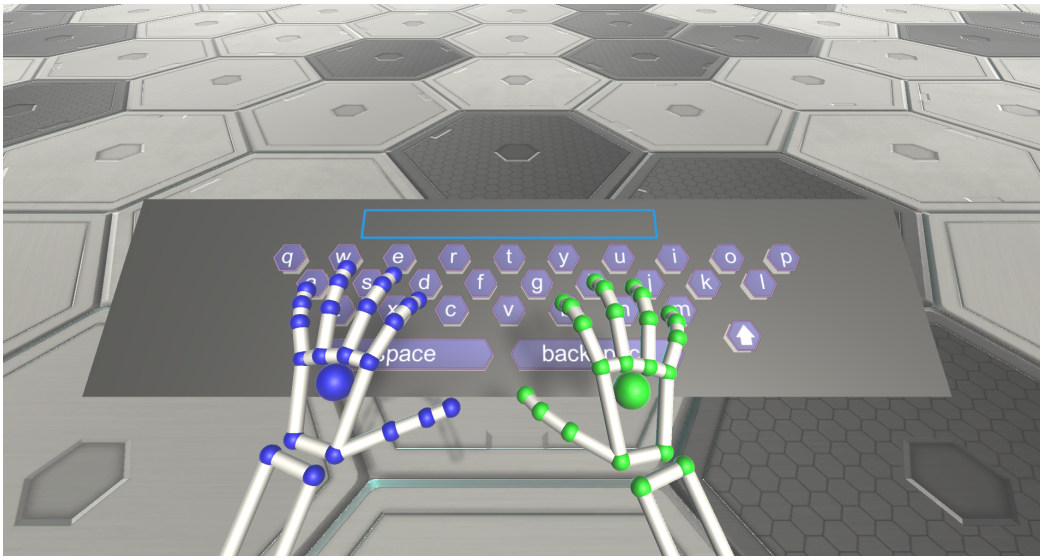


Figure 2.4: The initial version of the User Interface with the virtual keyboard

Version 2

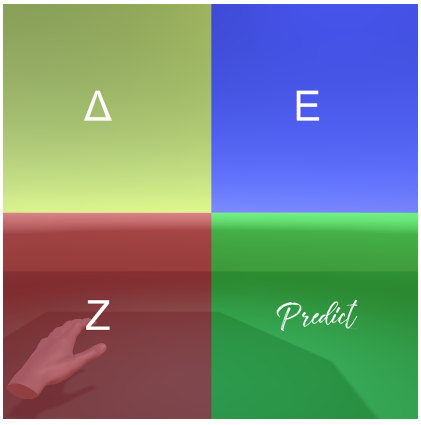
The second version of the UI (Fig. 2.5) was based more on users with physical movement difficulties and adopted a four options interface. The environment was divided in four quadrants, each one of which consisted of a dynamic choice according to previous selections. The user could use these quadrants in order to input a character with a very simple method. Each of the 24 of the Greek alphabet could be selected in a three-step process. While this method requires more input per character, it is way easier to use because of its reduced need of accuracy.

Figure 2.5: The second version of the User Interface with four selection options

At the beginning, three out of four choices contained the whole alphabet divided into three groups of 8 letters each, {A-Θ} for the first quadrant, {I-Π} for the second and {P-Ω} for the third one. The user could then select one of the three groups desired in order to get closer to her desired character. The contents of the three quadrants would then change into groups of three, three and two characters from the group selected in the first step and the user would then make their next selection. At the final stage there would be left at most a character per quadrant and the user can then select her desired character, the character is appended to the current word and passed to the prediction engine. An example of this process can be seen at Fig. 2.6.

(a) First Stage

(b) Second Stage



(c) Third Stage

Figure 2.6: The selection process of the character “Z” in UI version 2

The fourth quadrant was created in order to provide the user with word suggestions; as they type their words, possible word suggestions are displayed in this area as they are generated from the prediction engine. A number of up to four words can be displayed here for the user to choose from. If at any point the user sees the word they want to type in this quadrant, they can select this option and the four words will be spread out to the four quadrants for the user to choose from. After the user selects one of the possible suggestions, the corresponding word is then appended to the current sentence and the process starts from the beginning again.

Figure 2.7: Here some word predictions from the prefix "πορτοκ" can be seen

The selection process in all of the stages is quite simple and consists of a two-step procedure. As the user places their hands above the sensor, a visual representation of them appears in the virtual environment and their movements are tracked in real-time. The sensor's effective range was divided in two by a horizontal plane, perpendicular to the user, creating two separate areas.

The area closer to the user, named the Hover-Aim Zone, is used in order to visualize the user's hand in the virtual environment so they can prepare and aim their selection better and not having to train and memorize which areas in the real world correspond to the different choices in the UI.

After the user has placed their hand in front of the desired option in the Hover Zone they can then move their hand forwards in the zone further from them, named the Touch-Select Zone, in order to verify their selection. If the user's hand remains in the Touch Zone for more than a predetermined amount of time, named the *Selection Time*, the selection is made.

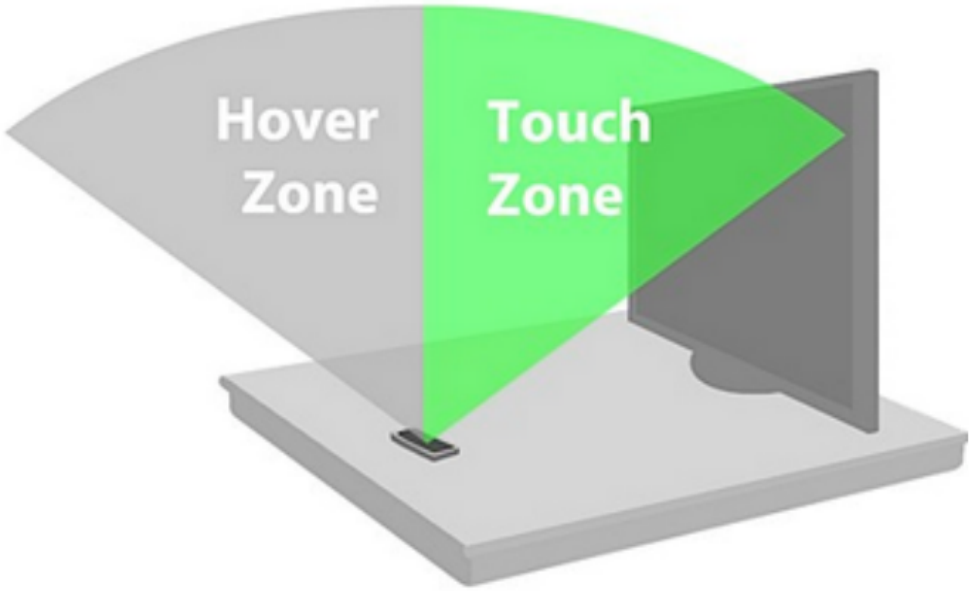
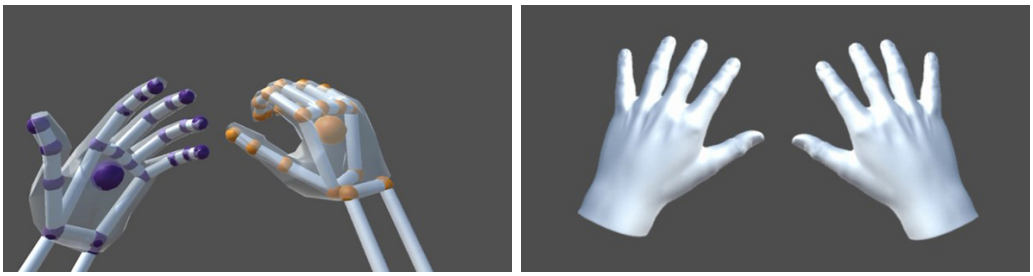


Figure 2.8: A visualization of the different areas in the process of option selection

Lastly, the default mechanical-looking hand model was replaced with a natural looking model to make the system more human-like for the user.



(a) Default Hands Model (b) Natural Hands Model

Figure 2.9: Different Hands Models for Leap Motion

At this stage of the system only the abbreviation prediction method (explained in section 3.1.1) was integrated into the system.

Version 3

This version focused more on the specific user this work was intended for and brought some minor improvements to the system in general. More specifically, after some feedback from the user's interpreter, the character selection method was tailored more closely to their specific abilities and their currently used physical method of communication.

The first change aimed at the layout of the options; the quadrants approach was dropped due to the user's difficulty of choosing between different height options. A new layout was designed; the cubes were placed in a line perpendicular to the user (Fig. 2.10). Also the Hover Zone expanded over the top part of the Touch Zone, making it possible to aim for the option desired in both the areas in front and above the cubes (Fig. 2.11).

Also, some visual and audio feedback was added. While the user's hand remains in one of the cubes, the cube's color would become less transparent by the time and a sound would be played when the option was finally selected. Both, the time needed to make a selection (*Selection Time*) as well as the volume of the sound, can be adjusted from within the UI during execution using two sliders at the bottom right edge of the screen. This further enhances customizability for each user and they can adjust the speed according to their abilities.

Until now the user did not have the option to write a word outside of the vocabulary of the system and had to choose a suggested prediction in order to move to the next word. This changed by adding the space character to the available set so the user can now enter a new word by selecting three times the third option (which was blank until now). Then, the currently typed word is appended to the sentence. Also, in this version the abbreviations prediction method was dropped and the Unigram model (explained in section 3.1.2) was adopted.

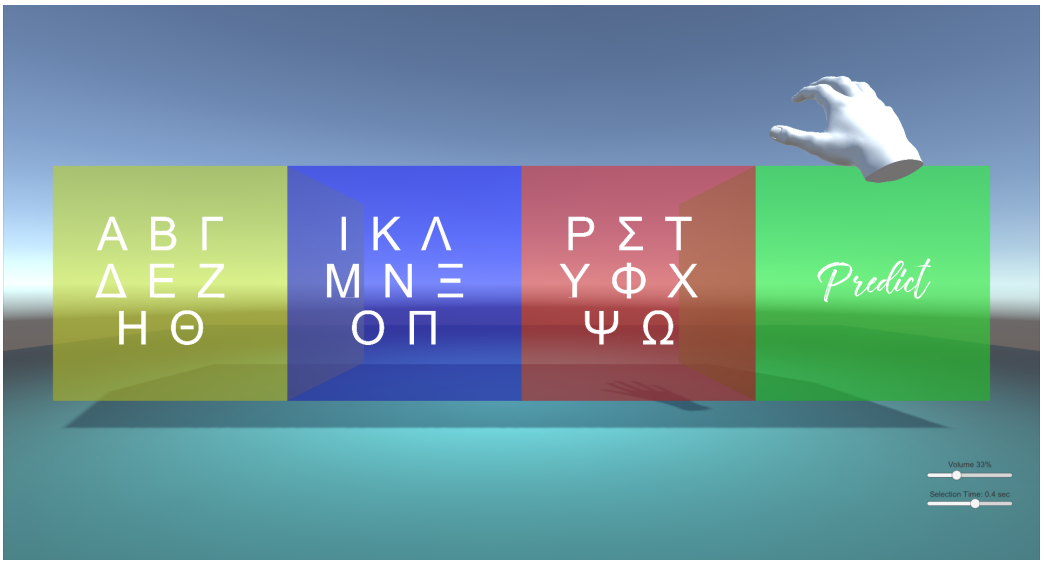


Figure 2.10: The rearranged Zones and adjustment options in version 3

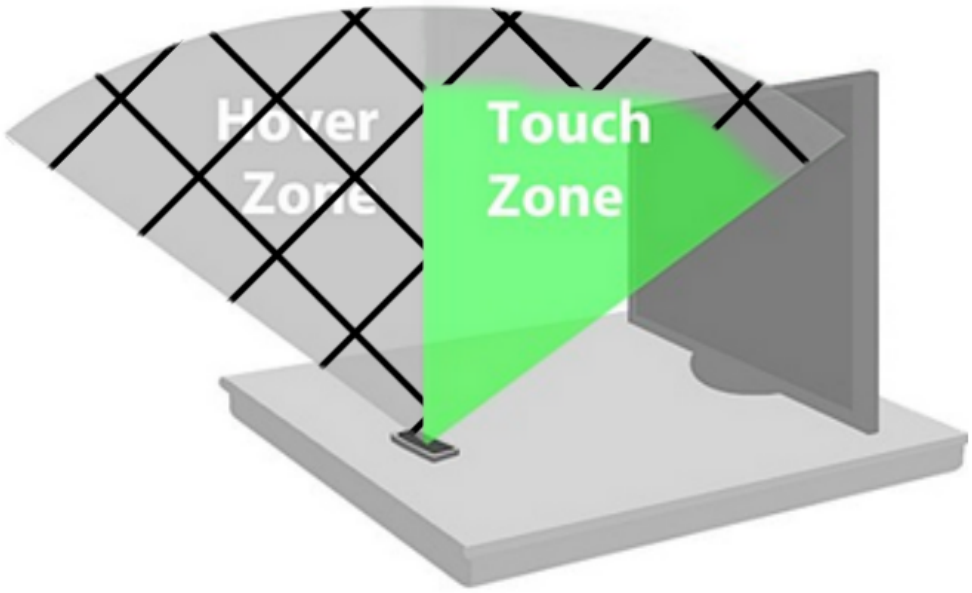


Figure 2.11: The rearranged Zones in version 3

Version 4

For the last and final version of the User Interface in the context of this thesis some extra cubes/options were added to better control the user's input as well as to be able to correct mistakes done by the user. Most importantly the speech synthesizer was added and the user is provided for an option to playback the text they generated. The final layout of the UI in this version can be seen in Fig. 2.12

Figure 2.12: The Final Look of the UI in Version 4

In regards to the input controls, three more cubes were added to the top of the screen, a *Back/Delete* option, a *Capitalize/Tone* option and a *Play* option. The *Back/Delete* option enables the user to go back one step in the selection stages (Fig. 2.6) as well as delete characters or words already typed. The *Capitalize/Tone* option allows the user to capitalize the first letter of the last word, capitalize or de-capitalize the whole word. Also, this option enables the user to add accent to the appropriate letters.

The *Back* option is enabled when the user is in the second, third, or prediction step of the selection process and when chosen, the user drops back to the previous stage. If the user is already in the first stage of selection and there are characters in the current word he is typing, the *Delete* option

is enabled. If the user selects it, the last character of the word is deleted. If there are no characters in the current word, then the first word of the sentence is deleted.

The *Tone* option is enabled when there are characters in the current word and the last character of the current word is able to receive an accent or umlaut (e.g. ï, í). If these conditions are met, then selecting the *Tone* option multiple times will cycle through adding accent, umlaut, umlaut with accent (if applicable) and no punctuation again. If there are no characters in the current word and there are words in the sentence, then the *Capitalization* option is enabled. Selecting this option multiple times will cycle through first letter capitalization, full capitalization and full de-capitalization of the last word in the sentence.

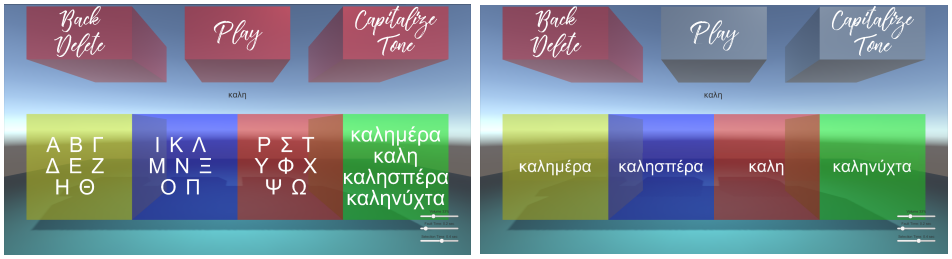
At any stage, the user can select the *Play* option in order to playback their text in speech using a male or female voice. Although this process is not done by the Unity engine, it is not required to leave the UI, since communication between the systems has been implemented. The speech synthesizer and the communication between the systems will be discussed in more detail in section 2.3.

In addition to these option, extra measures were taking in order to reduce false input. The introduction of a *Fault Time* was added to the UI; this timer makes sure that if the user selects an option and accidentally move their hand in and out of that option within the *Fault Time* specified, the option will not be activated twice. This timer is also used to avoid accidental activation of neighboring options. As with the *Selection Time*, the *Fault Time* can be adjusted through a slider added to the same spot.

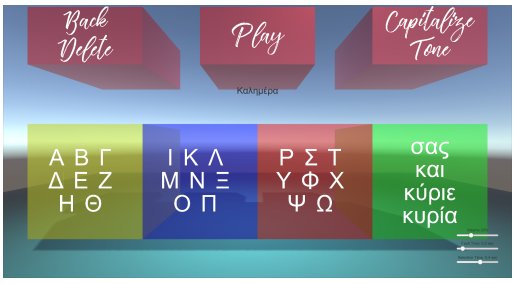
Another measure taken, while trying to increase accuracy and decrease false input for users that can and prefer to use both hands, was the ability to chose which hand (left, right or both) can activate each option. This is a very important feature as it decreases the number of steps required to make a selection up to half the amount, possibly making the whole text synthesis process twice as fast! This speedup can be achieved by splitting the letters in groups half the original size and assigning them to the same box but making it possible to select a different subgroup depending on the hand that is used.

Finally, a menu was added in order to be able to choose from the different prediction options available so the user can experiment with them and chose the one that fits her the most. The main motivation behind this menu was actually for a possible future version of the system that would enable the user to select a mode like conversation, formal, etc, and the prediction engine would offer more relevant results.

An example of word completion (a) and next word prediction (c) can be seen in Fig. 2.13. Having as input the characters "καλη", the system provides them with four word completion suggestions (a), the user can then select the bottom far right option in order to complete their desired word "καλησπέρα" that is in the list. They are then brought to the suggestion selection screen (b), where they can chose the blue option to append that word in the sentence. As soon as the word is appended, the user is provided with four possible next word suggestions "σας, και, κύριε, κυρία". If one of these words are the desired next word, the user can select the bottom far right option to be brought to screen (b) again and choose which word they want to append to the sentence.



(a) Word Completion Suggestions (b) Word Suggestion Selection



(c) Next Word Prediction Suggestions

Figure 2.13: An example of word completion (a) and next word suggestion (c) in the final version of the UI

2.3 Speech Synthesis

The speech synthesis engine was a little tricky to integrate into the main system since free to use, even for non-commercial applications, Text-To-Speech (TTS) systems with support for the Greek language are not widely available. Some projects were found across the web but, for most of them, the speech output was too unnatural or they could not function in cooperation with other systems or language support was not provided.

In order to overcome this problem, a JavaScript API was found that is mainly used to voice-enable websites and is free for non-commercial use by *ResponsiveVoice* [34]. Using the commands found in the API, a website developer can dynamically convert text into speech and playback the output through the browser. The only drawbacks of this method are the required internet connection, since the TTS process is done at the provider's end, and that it is not completely integrated in the system, but an external service has to be run at the start of the system.

To use this API, a local website had to be created that would act as medium for the text typed in the application to be played as speech. In order to exchange data between the system and the website, a text file was created and was modified when needed by the system. On the system's end, when the user selects the *Play* option through the UI, the current sentence is appended to the text file along with its time-stamp. On the website's end, the text file is ready once per second and if the file contains a sentence with a new time-stamp, the sentence will be converted to speech and played by the browser.

Because of security policies, it is not allowed to automatically read a local file directly without using a browser prompt. In order to bypass this problem, a local HTTP-server was created locally and the file can then be read without prompt from the server's directory. Since this directory is local, the system can read, write and modify its files. The server was created using Node.js [35] JavaScript runtime and the http-server package from NPM [36], with the option of disabling caching.

The whole process can be see at Fig. 2.14.

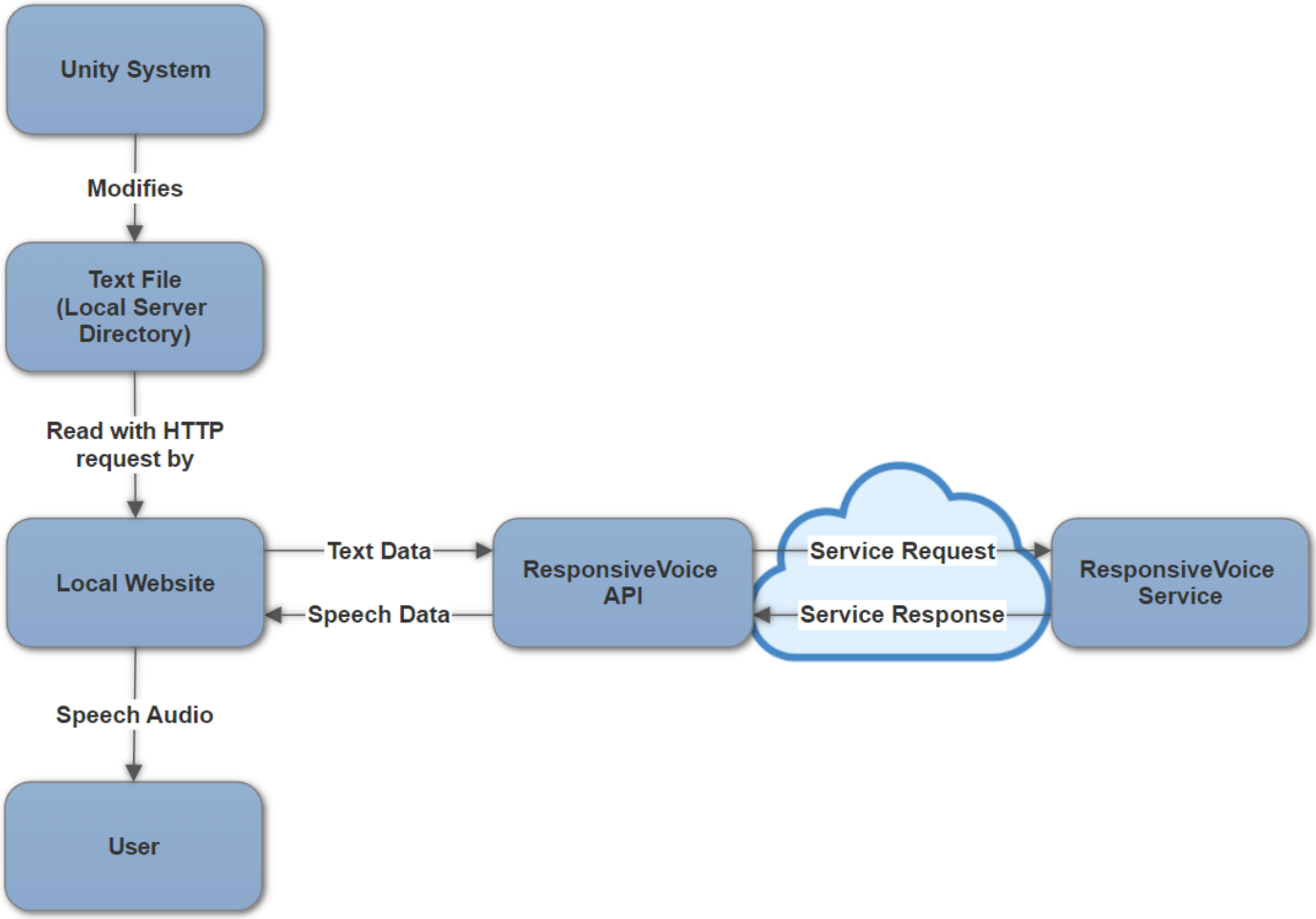


Figure 2.14: The process for the Text-To-Speech conversion and playback

Chapter 3

Algorithms

This chapter describes the different methods used to provide predictive suggestions to the user about the current word they are typing (word completion) and the words they most likely to type next (word prediction). This is done to minimize the need for user input in order to relieve them of having to input every single character/word and also to speed up the text composition process. This task is most commonly known as word prediction and be tackled using a wide variety of approaches.

In this work three main methods were implemented, the first two of which were relatively simple ones used as a baseline, with the last one being relatively more complex and not as established in this field.

3.0.1 Corpora

The corpora and dictionaries that are required for the methods used were taken from the Leipzig Corpora Collection [37]. The sources of the corpora used were texts randomly collected from the web as this was the most suiting option compared to the other sources. These sources are less than ideal for this case, but these corpora were used due to being well structured and the lack of reliable and big enough data for the Greek language form other sources. The corpora are automatically collected from carefully selected public sources without considering in detail the content of the text. The texts are processed and most of the foreign material is removed. Then the text is split into sentences and stored in a file along with each sentence's unique ID. Some other useful data are provided like a file that contains a list of all the word forms in the corpus and their IDs, ordered by their number of occurrences. Another useful file is the neighborhood co-occurrences file that contains information about how often two words occurred directly next to

each other and the number of those occurrences. All the files used were encoded in the UTF-8 format in order to support different languages. While this information was useful to start with, additional info has to be extracted from the corpus in order to make more accurate predictions.

3.0.2 Tools

The sentences of the corpus contain elements that are not useful and can interfere with the analysis. These elements can be punctuation marks, numbers, dates or foreign characters. In order to remove these elements a tool was created in Java that takes as input a text document and removes such elements using regular expressions to find them. This tool can also extract information like the vocabulary, the count of each word and N-grams but regarding the latest task our tool was not optimized for large files. For this task, a tool named "N-Gram Extraction Tools" and provided by the School of Informatics of the University of Edinburgh [38] was used. This tool implements Nagao '94 arbitrary N-gram extracting algorithm and can provide N-grams of higher orders with better efficiency. The output of this process is then passed through our tool to be brought to a suitable form and to be organized according to our needs. Even though the tools and algorithms were implemented based on these corpora, their input can easily be changed if a more fitting resource is found.

3.1 Approaches

3.1.1 Using Abbreviations to Minimize Input

The first approach that was implemented is based on [22] and it uses word abbreviations in order to minimize the characters required as input in order to type a word. Obviously, this method is not very effective for small words but can be very effective when it is used for big words or phrases. The Greek language has many words with prefixes and compound words, so the average length of words is quite high; thus, making this method suitable for our case.

This method requires a dictionary of all the different the words the user might want to write; this dictionary was extracted using a corpus of 1.000.000 sentences and consisted of 575.000 words. As the user is typing the characters, the prediction engine tries to match the input sequence to the whole dictionary using the longest common subsequence algorithm (LCS) in order to produce word completion suggestions. From the set of words that matches the input sequence, the user is displayed four words with priority to smaller words in length and with matching prefix. If the first criterion was not in place, relatively smaller words could be left out of the possible suggestions so the user would have to type the whole word character by character. The priority given to the small words does not completely count out bigger words from suggestions, as the user can just keep on typing parts of the words that will exclude the small words due to the length or the matching differences. An example of this is shown below:

Let's say that the user wants to input the word "καλημέρα". The prediction engine starts working after the first three characters are typed. The selection of the characters that will be skipped or not is entirely up to the user. The ideal case would be inputting characters that are characteristic of the word and can differentiate it from other possible words, but just giving the consonants of the word and possibly the vowel with the accent works really well. Especially the latter can reduce the possible suggestion substantially because of the morphology of the Greek language.

In this case the consonants are "κ - λ - μ - ρ" and as the user inputs the first three characters "κλμ" the four words suggested are:

κάλμα, κλάμα, κλάμπ, κλήμα.

After the final consonant "ρ" is inputted the suggestions become:

κάλμαρα, κάλμαρε, καλμάρω, κομπλέρ.

Since the desired word still does not show up in the suggestions the user can input the last remaining character of the word, which is "α", getting as the final suggestions:

κάλμαρα, καλαμαρά, καλημέρα, καλήμερα

So the user can then select the third suggestion and append it in the sentence he is typing. In this example we can also see how the accent can help differentiate between words with identical possible abbreviations. If the user wanted to input the word "Καλαμαριά" he could add the accent to the last character of the abbreviation, making it an "κλμρά", thus limiting the possible suggestions to a more desirable set:

καλαμαρά, καλαμαράς, Καλαμαριά, καλαμαριά.

As it can be observed from these examples, there are incorrect forms of words in our dictionary, since its source is the web, and this can affect the performance of the system. Increasing the number of suggestions can partly solve this problem since the words matching the abbreviation and being incorrect are very few. A more complete solution would be filtering the dictionary or use a more formal source.

In addition to this problem affecting the performance, there is another flaw in this method. It is not uncommon for users to type place names, new words that do not exist in the dictionary or make spelling mistakes. In these cases, the user might not know that the word they want to type is not in the dictionary so he might try to input an abbreviation of the word. That will result in failed prediction attempts and the user will have to delete the

abbreviation and enter the complete word character by character. Also, the prediction engine will try to match the abbreviation to the dictionary and since there will probably be no matches, the system will have to go through the whole dictionary. Scanning the whole dictionary has some computational requirements and would make the system sluggish.

The first part of the problem, about the users' unawareness of the contents of the dictionary, cannot be solved. To solve the second part of problem, three measures were put into place. First of all, because the engine is searching for the smallest words to match, the dictionary was organized based on word length to make searching faster. As soon as four words were found (they are the smallest thanks to the sorting) the search was interrupted. Second, if the input does not match any words from the dictionary, further searches will not be started if the user keeps on typing characters. Finally, the prediction engine process was placed in a different thread than the UI system and the suggestions were provided to the user as soon as they were found. This avoided delays cause by the searches that would influence the rest of the system.

3.1.2 Markov Chains and N-Grams

When trying to make predictions for word completion or next word suggestions, it is important to have a model to base those prediction. In natural language processing applications like this, a category of such models is called statistical language models. In general, given a sequence of words $w_1, w_2, ..., w_n$ they try to assign a probability $P(w_1, w_2, ..., w_n)$ to this sequence. Having a way to estimate those probabilities for different sequences is useful in a range of natural language tasks such as speech recognition, machine translation, word prediction etc. The models used for this approach fall under this category.

The second approach implemented was rooted on [23] and it uses Markovian language models based on frequencies. Sentences are seen as a discrete stochastic process in which the words (or sets of words) are the different states. Using the Markov assumption, hence the name Markovian models, future states are only dependent of the current state and completely independent from past states. This means that we can make a prediction for future words without using all the preceding words but just the last state. This model is also known as a Markov chain.

Formally a Markov chain is a sequence of random variables $X_1, X_2, ..., X_n$ such that:

$$P(X_{t+1} = i_{t+1}|X_t = i_t, X_{t-1} = i_{t-1}, ..., X_1 = i_1) = P(X_{t+1} = i_{t+1}|X_t = i_t) \tag{3.1}$$

In general, a Markov chain of order n is a discrete stochastic process in which future states depend only on the current and the last $m - 1$ states ($\forall n \geq 2$). Formally:

$$\begin{aligned} P(X_{t+1} = i_{t+1}|X_t = i_t, X_{t-1} = i_{t-1}, ..., X_1 = i_1) = \\ P(X_{t+1} = i_{t+1}|X_t = i_t, X_{t-1} = i_{t-1}, ..., X_{t-(m-1)} = i_{t-(m-1)}) \end{aligned} \tag{3.2}$$

Our states' probabilities are independent from time, such a Markov chain is said to be stationary and it is formally written as:

$$P(X_{t+1} = j|X_t = i_0, X_{t-1} = i_1, ..., X_{t-(m-1)} = i_{m-1}) = p_{j,i_0,...i_{m-1}} \ \forall t \quad (3.3)$$

In order to use these models for word prediction, the Markov Chains are applied to sequences of words. If we replace the random variables $X_1, X_2, ..., X_n$ with the words of the sentence $W_1, W_2, ..., W_n$ then the probability of next word W_{t+1} can be calculated given the values of the last $N-1$ words $W_t, W_{t-1}, ..., W_{t-(N-2)}$. This is called an N-gram, formally a stationary Markov chair of order $N-1$ ($\forall N \geq 2$).

Therefore the probability of the next word W_k in a sentence can be given from an N-gram using only the last N-1 words $W_{k-1}, W_{k-2}, ..., W_{k-(N-1)}$ instead of the whole preceding sentence $W_{k-1}, W_{k-2}, ..., W_1$ and can be written as:

$$P(w_k|w_{k-1}, w_{k-2}, ..., w_1) = P(w_k|w_{k-1}, w_{k-2}, ..., w_{k-(N-1)}) \ \ \forall N \geq 2 \quad (3.4)$$

N-grams with $N = 1$ exist and are called Unigrams but they are a special degenerative case. With Unigrams the probability of the next word is completely independent from the previous words therefore they do not really constitute a Markov chain.

The most commonly used N-grams are those with $N = 2$ and $N = 3$, known as Bigrams and Trigrams. Let's look at an example to better understand how N-grams work. The sentence in question will be "Αυτό είναι ένα παράδειγμα πρόβλεψης κειμένου για κατανόηση". As mentioned before, the probability of any word would be dependent on all the preceding words, so the probability of "κειμένου" being the next word would be:

$$P(\text{κειμένου}|\text{πρόβλεψης,παράδειγμα,ένα,είναι,Αυτό})$$

Using the Trigram model the probability of the next word in the sentence "Αυτή είναι μια δοκιμή για πρόβλεψη" being "κειμένου" would be:

$$P(\text{κειμένου}|\text{πρόβλεψης,παράδειγμα})$$

Using the Bigram model would further limit the dependency on previous words:

$$P(\text{χειμένου}|\text{πρόβλεψης})$$

Knowing that simplifies things and results in reducing the number of probabilities we need to learn in order to predict the next word. Instead of having to calculate the probability of every sequence of words (including their subsequences), calculating the probability of pairs or triplets, depending on the order of N-gram model we want to use, is enough. This in turn vastly reduces the computation time and, while trying to predict the next word, the search time. Also decreased are the requirements for storage space. As a result, this makes the N-grams a feasible language model for prediction.

In this work implementations using a combination Unigrams, Bigrams and Trigrams were mainly used, but experimentation with higher order N-grams was also done. In order to calculate the probabilities of the N-grams the Maximum Likelihood Estimation (MLE) was used.

For example, calculating the probability of a word w_k in the Bigram model given the previous word w_{k-1}, was done by counting the occurrences, $C(w_k, w_{k-1})$, of the pair w_{k-1}, w_k in a corpus and normalizing by the sum of all the pairs that have w_k as the first word:

$$P(w_k|w_{k-1}) = \frac{C(w_k, w_{k-1})}{\sum_{i=1}^{n} C(w_k, w_i)}, \tag{3.5}$$

where n is the number of unique words in the corpus.

This can be simplified because of the fact the number of occurrences of the pairs w_k, w_i is similar or equal to the number of occurrences of the word w_{k-1} [1], so 3.5 can be written as:

$$P(w_k|w_{k-1}) \simeq \frac{C(w_k, w_{k-1})}{C(w_{k-1})} \tag{3.6}$$

[1] because if w_{k-1} is in the end of a sentence $C(w_{k-1}) > C(w_i, w_{k-1})$

In general for any order of N-gram the ML estimation equivalent of 3.6 can be formally written as:

$$P(w_k|w_{k-1},...,w_{k-(N-1)}) \simeq \frac{C(w_k,w_{k-1},...,w_{k-(N-1)})}{C(w_{k-1},w_{k-2},...,w_{k-(N-1)})} \tag{3.7}$$

In theory a model with N tending to infinite would be a very good model, but in practice this is impractical due to the large number of possible N-grams and an issue called the *zero-frequency problem*. This problem arises because of the fact that the size of the corpus would also have to be infinite. Even for relatively small orders of N-grams the transition probability vector is sparse due to the fact that the corpus does not and cannot contain every possible N-gram. With a fixed size of vocabulary n there are n^N different N-grams. While most of the N-grams would not be valid in terms of meaning, some of them just do not exist in the corpus because of their vast number. To solve this problem different smoothing techniques have been proposed that try to move some probability from the high frequencies to the zero frequencies elements of the matrix.

As mentioned before, in this work we used a combination of Unigrams, Bigrams and Trigrams. Since smoothing was not used, sometimes matching Bigrams or Trigrams could not be found. In this case Unigrams were used that were extracted from the corpus in a process almost identical to the method used for obtaining the dictionary for the abbreviations. Using a big enough corpus, the ML estimate of the word's w_k probability can be calculated by counting its occurrences in the corpus and dividing by the total number of words in the corpus:

$$P(w_k) = \frac{C(w_k)}{\sum_{i=1}^{n} C(w_i)} \tag{3.8}$$

Considering that the sum of the occurrences of all the words is the same for any word, the previous expression can be reformed as:

$$P(w_k) \propto C(w_k) \tag{3.9}$$

Since we are not interested in the absolute value of the probabilities and

rather in which word is the most probable to follow we can use Eq. 3.9 and just count the occurrences of each word.

So, a list of all the Unigrams with the number of occurrences for each one was obtained using our tools. The Java tool was used to remove from the corpus as much of the unwanted elements as possible and shape the text to a suitable form for the next process. Then, the N-gram Extraction tool was used in order to obtain a list with unique words and their number of occurrences. Finally, the list was organized alphabetically and by occurrences in order to speed up searching while the prediction engine is in use.

When the user starts typing the first word, there are no preceding words and Bigrams or Trigrams cannot be used. So as the first character is inputted the prediction engine searches the list of Unigrams for the four words beginning with that character that are the most probable. When and if the words are found they are displayed as prediction suggestions for the user to choose from. This goes on as the user keeps on inputting characters until he selects a word from the suggestions or inputs a blank space. If a suggestion is selected then the word is appended to the sentence and the user can start typing the next word. After the user has entered the first word in the sentence the Bigram model is activated. The Unigram model is also used when there are no Trigrams or Bigrams for the words preceding.

Similarly to the way the Unigrams' probabilities can be simplified to 3.9, the probabilities of Bigrams and Trigrams can also be simplified. As already said before the Bigram probability for w_{k-1}, w_k is:

$$P(w_k|w_{k-1}) \simeq \frac{C(w_k, w_{k-1})}{C(w_{k-1})} \tag{3.6}$$

Because of the fact that w_{k-1} is stable and given for all the probabilities of different words w_k, the count of w_{k-1} is independent and same for every w_k. So the expression above can be reformed into:

$$P(w_k|w_{k-1}) \propto C(w_k, w_{k-1}) \tag{3.10}$$

This can also be expanded and the equation for the general case of N-grams 3.7, to:

$$P(w_k|w_{k-1}, ..., w_{k-(N-1)}) \propto C(w_k, w_{k-1}, ..., w_{k-(N-1)}) \tag{3.11}$$

According to this, the lists of Bigrams and Trigrams were generated through the Java and the N-gram Extraction tool and then was organized alphabetically and by count of occurrences.

So, after the first word is entered, the prediction engine searches the list of Bigrams for the four of them that have the word w_{k-1} as their first part and that are the most probable. Then the words w_k of those four Bigrams are displayed to the user as prediction suggestions for the next word they want to type. If none of these matches what the user wants to type, he can start typing the characters of the word he wants. While he is doing that, the engine starts searching those Bigrams that their first word is w_{k-1} and their second word starts from the character(s) the user has typed. If at a moment there are no matches for those criteria, the Bigram model is dropped and the engine falls back to the Unigram model. The Bigram model is also used if the preceding words do not match a Trigram.

Lastly the processes described before were used in order to create the Trigrams list and use it in the prediction engine in the same way as with the Bigrams.

3.1.3 Neural Networks - LSTMs

The topics of AI and Neural Networks (NNs) have recently attracted renewed intrest. These techniques are being applied to all sorts of fields with their performance being at least to say interesting and results that are sometimes groundbreaking. This success can be partly accounted to the increasing computation power of the modern systems and to some innovative techniques that make deep neural networks and faster training more viable. This method was inspired by and mainly based on Andrej Karpathy's post about Recurrent Neural Networks [39] and his related project *char-rnn* [40].

Recurrent Neural Networks - LSTM

Unlike traditional NNs, Recurrent Neural Networks (RNNs) introduce the concept of sequences of data. This is a crucial factor in natural language processing because the actual meaning of a sentence does not reside in the words as individual elements but rather depends in the context, the previous sentences, their order of the words and in their intermediate relations.

The NN model that was decided to be used is a specific variation of a RNN and its called Long Short-Term Memory (LSTM). LSTMs are very similar to regular RNNs but with the difference that they have logic to control the flow of information from state to state. They can forget parts of the previous state, selectively update state values and control which parts of the current state will be outputted. Another advantage of LSTMs is that because of their structure, when using back-propagation, dependencies are based on addition instead of creating a huge chain rule product (*Vanishing Gradient Problem*).

The language model chosen for this work was based on [19] and it is a character-level architecture. Even though state-of-the-art models use word level prediction, it has been proven that this model can provide state-of-the-art performance despite having approximately 60% fewer parameters (based on the work cited). Another advantage of this model is, because the input and output are just characters and not words, there are no words that are out of its vocabulary.

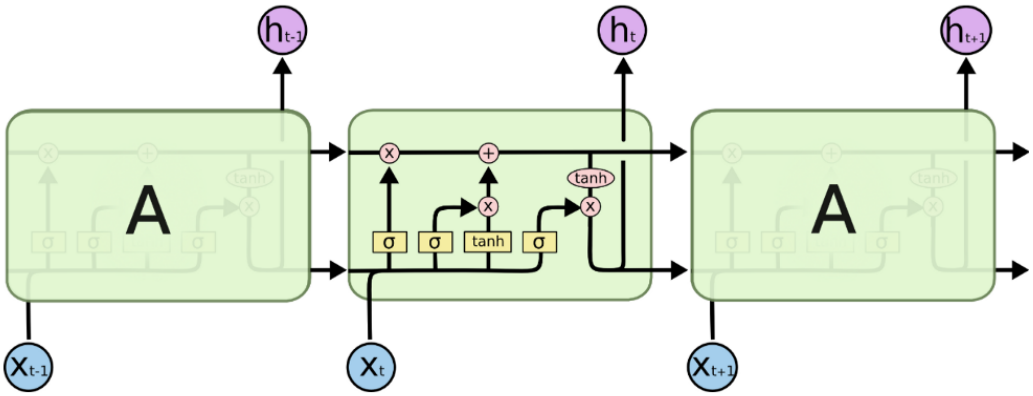


Figure 3.1: Long Short-Term Memory Network Structure

To create the model, the implementation of [41] *char-rnn-tensorflow* was chosen, which is basically a replica of Andrej Karpathy's project but implemented in Tensorflow instead of PyTorch. Tensorflow was chosen because of its increased popularity as well as its strong visualization tool *Tensorboard*, which is useful for debugging and visualization while the model is being trained.

In Fig. 3.1 we can see a visual representation of the basic structure of an LSTM. First of all, the recurrency/feedback of the model can be seen as the output of the previous cell/state h_{t-1} is used as input for the current cell/state. Also visible, in this figure, is the inner structure of a cell that gives the RNNs the ability to forget or remember information of the past states.

The training was done using the same corpus as the N-grams. Tuning the model and its parameters is more of an experimental process; multiple models were trained using different tuning settings for the values: network size, number of layers, length of the characters window used as input, input and output dropout. Dropout refers to a technique used during training to reduce over-fitting to the input data-set by randomly ignoring elements of the model[2]. In order to make these decisions some general guidelines provided

[2]Elements of the model are ignored or not according to a user-defined probability (0 always ignored, 1 always used). Input and output dropout refer to ignoring a certain input or output of the model during training

with the library were followed. Some of the main settings were:

Model #	Layers	Size	Input Dropout	Output Dropout	Sequence Length
1	2	128	1.0	1.0	50
2	3	500	1.0	0.8	50
3	5	500	0.8	0.5	50
4	3	128	0.8	0.5	20

Table 3.1: Different settings used for the training process

The training was done on the laboratory computer's GPUs (2 x NVIDIA GeForce GTX 780 Ti) and took from 8 hours for the simplest mode, to 24 hours for the most complex one. In general, due to these long times, tuning the models is a long and difficult process that requires time and experience.

The models trained were not actually used in the finished system since there were some difficulties, but some conclusions can be extracted by the experimentation conducted. First of all, advanced NN models have very recently started becoming popular and available for the most people. Naturally, there are not many tutorials available with detailed instructions and explanation of how these techniques work. Also, the tools needed to create and train such models are not refined to be used by inexperienced users yet. A decent amount of work has to be done in order to install, use and customize the tools for certain needs and application. Despite these difficulties, it is possible to create models that yield respectable performance and if they are fine-tuned and trained properly, they can provide state-of-the-art results.

The major disadvantage of NNs is that, although the model is designed by the user, the process through which they produce a result is mostly a mystery. More specifically, even though the models can provide impressive results, we cannot understand why or how these results were produced.

Lastly, there are many different types and variants of NNs. On the one hand, that gives us the capability to use the variant that is most suitable for our case. On the other hand, the amount of different possibilities and settings available makes the design process more complicated, experience requiring and, along with training, very time consuming.

Chapter 4

Evaluation - Results

4.0.1 Introduction

The evaluation of the system was done on version 4 of the UI and used the combined N-gram prediction model since it was deemed more appropriate for inexperienced users than the abbreviation method. The LSTM approach was not included in the trials with the users since there were difficulties integrating it into the system, but more importantly there was more work needed in order to fine tune the models and get the execution environment to behave as desired.

The trials included five members of the Telecommunications' Lab and myself. Four of them had never used the Leap Motion sensor before or any version of the UI. Two of them had very basic experience with the sensor and the UI but without the prediction engine. Only myself was experienced with the sensor and the UI but had not used the system extensively besides testing, so my results will be listed separately from the others.

The participants were given a brief explanation of how the system works and a time period of up to five minutes in which they could get familiar with the UI and the sensor and could adjust the positioning of the sensor and monitor. After that they were given three sentences to reproduce using the system as well as one of their own. All of the sentences were chosen randomly and had never been tested on the system. While they were using the system, their mistakes, time needed per sentence and comments were recorded. Since this was a first experience with the system, questions asked about the usage of the system were answered but there was no outside interference with the system. Finally there was a brief conversation with the participants and they were asked to write a short review of their overall experience. A sample of the evaluation sheet can be found in the appendix (Chap. 6)

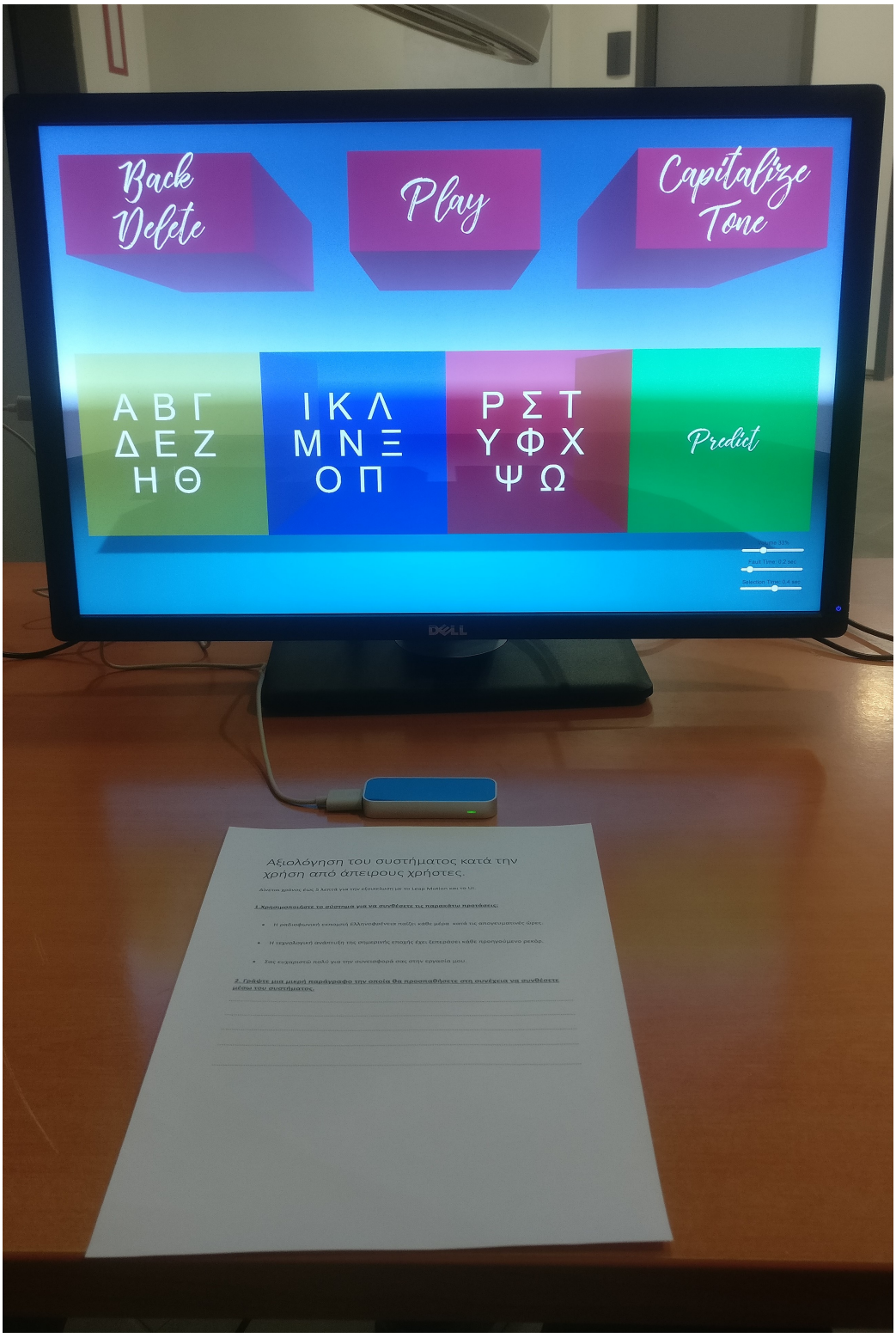


Figure 4.1: The Evaluation and Testing Environment

4.1 Evaluation

4.1.1 UI

In terms of the UI, what was greatly noted was its simplicity. That was expected since this work is intended for people with limited physical capabilities but left regular users wanting more. It was also noted though that it did not lack in any way in terms of functionality and speed. Positive comments were noted for the audio feedback and for the ability to change the selection time because it provided the user with confidence of moving their hand around without accidentally selecting something they did not want. In general the UI was described easy/fast to learn, easy-to-use, user-friendly, fast, foolproof, not tiring to the eyes, not lacking functionality and not demanding on the user.

On the negative side there was a comment concerning editing words already appended in the sentence in order to change their endings. This comment came up due to the many grammatical forms being based on the same linguistic root in the Greek language. In some cases, the desired grammatical case or gender of a word would not appear in the suggestions. That is because there were other words having the same prefix (root) with higher probability, making the user type many character before the desired word appears. A possible solution to this matter is addressed in 5.2. Another participant noted a similar comment for word endings and suggested that the system could display more possible predictions.

Another issue mentioned was about punctuation marks and their effect in the speech synthesis process. A participant with a long enough sentence noticed that the absence of punctuation marks makes the Speech produced a little unnatural for those cases. This is true and is more related to the UI since punctuation is treated accordingly by the TTS system if provided, but the UI does not provide this option in its latest version. This issue is also addressed in 5.2.

The last issue with the UI was that sometimes the participants would not see that the desired word was suggested by the prediction engine (sugges-

tion neglection) and they would keep on typing characters. This issue was appointed to the fact that the participants were changing their focus from the system to the paper the sentences were written and were more focused trying to find the next character without even looking at the suggestions. Although this might be the only cause of the issue, further investigation has to be made regarding the display of the suggestions.

4.1.2 Leap Motion

The Leap Motion sensor and its usage received mixed comments. On the positive side, the sensor was commented for its seamless response time and accuracy that provided a very natural feel. On the negative side, the participants noted some discomfort from the continuous hovering of the arm, although some of them adjusted their positioning and movements accordingly making the process less tiring. Finally, observing the capabilities of the device, the participants noted and gave some suggestion about the customization of the interaction process which was partially implemented in previous version of the UI.

4.1.3 Prediction

Regarding the prediction engine the participants noted that it was fast and that in most cases they were more than satisfied with the suggestions provided and sometimes they were surprised with its abilities. What was negatively noted for the prediction system was its difficulty to provide predictions for the first and second grammatical person and some grammatical cases. Another comment was about the number of possible suggestions which is not really a restriction of the prediction engine but of the UI.

The issue with the grammatical forms was clearly observed during the trials for the first and second grammatical person (occurred 1.4 times per participant), this issue was clearly attributed to the nature of the training corpus, which consisted mostly articles found on the web were the third person is principally used. Some solutions for this particular problem are discussed in 5.2.

Another issue observed was about the accent system. If the user forgets to add the accent to the right character, then the desired word would not end up in the predictions. But the use of the accent helps in the prediction process significantly and the users got used to it very fast. Specifically, the average number of accent omissions was 1.8 in 48 words (3.75%). These mistakes were observed mainly on the first time the participant had to enter an accent and they got accustomed to this process very quickly. This issue could be avoided by matching words independently of the accent but it would have a negative effect in the performance of the system. It was decided to keep the original implementation since the problem can easily be dealt with by doing very little training. This way, the desired word is predicted faster and does not stress the user as much since accents require only one step for insertion.

Finally the participants gave the suggestion for a prediction system that learns from its user as it is being used.

4.1.4 Text-To-Speech

The Text-To-Speech system's performance was spot on. Its response time was unnoticeable and the quality of the audio generated was very natural and human-like. The only negative comment was about the lack of pauses during the sentences in places where there should be a punctuation mark. This issue is actually not related to the TTS system itself, but it has to do with the lack of punctuation capabilities of the UI, as it was mentioned in the relative subsection 4.1.1.

4.1.5 Statistics

As it was mentioned in the introduction, these statistics were derived from the evaluation of inexperienced users. My results are listed separately for the sake of comparison between a total beginner and a more experience user.

The average time needed to complete the evaluation was 16 minutes, 100 characters were inputted directly and 171 characters were skipped thanks to the prediction system, reducing the time needed per word by 34 seconds. This

translates to a 63% reduction in characters/time and brings the time needed per actual character to 3.53 seconds. The participants generated on average 48 words resulting to 20 seconds needed per word with an average length of 5.66 characters. There were 2.1 characters per word required before the right prediction was made skipping 3.56 characters per word. In total the amount of time skipped was 27 minutes and 10 seconds. There were on average 2.6 miss-selections, 1.8 accent omissions and 3.0 prediction neglections per participant in the whole evaluation with them occurring at the start of the process.

For reference, my results, as a more experienced user without having done this particular test, showed that the time needed to complete the test was less than half that of an untrained user. Specifically, it took me 7 minutes and 24 seconds (54% decrease), typing at 6.00 seconds per character, needing 1.82 seconds per actual character, making 1 miss-selection, 0 accent omissions and 0 prediction neglections.

4.1.6 Overall

The system as a whole was generally described as very functional with fast and seamless performance. The UI was simple enough without lacking any core features but needing some extra additions and tweaks. The prediction engine was described more than sufficient and unexpectedly capable in some cases but with some complaints that were actually related to the UI. Lastly, the TTS system was flawless with the exception of the missing punctuation pause issue.

4.1.7 Results

#	Time (sec)	Chars Typed	Chars Skipped	Words	Miss Selections	Grammatical Misses	Accent Omissions	Prediction Neglections
1	960	103	164	48	1	2	1	3
2	1057	118	185	48	2	3	4	1
3	924	109	183	54	2	0	0	1
4	893	91	153	45	5	1	3	4
5	960	82	170	48	3	1	1	6
AVG	958.8	100.6	171	48	2.6	1.4	1.8	3

Table 4.1: The Results of the Inexperienced Participants' Evaluation

#	Time (sec)	Chars Typed	Chars Skipped	Words	Miss Selections	Grammatical Misses	Accent Omissions	Prediction Neglections
1	444	74	170	44	1	0	0	0

Table 4.2: Experienced User's Evaluation Results

Chapter 5

Conclusions

5.1 Conclusion

This work provided a working and viable system, simple enough to be controlled by physically impaired individuals, that relies only in the rough movements of one or both hands, for the purpose of text and speech synthesis, aided by a word-completion and word-prediction system, without requiring any expensive/dedicated hardware or any intrusive/obscure equipment. Although having achieved these goals, after the evaluation, some simple and some complex improvements can be done in order to improve its performance and expand its capabilities.

5.2 Future Work

As it has already been said, the main goal of this work has been achieved, but some modifications and investigation can be made in order to improve its performance. More specifically:

- Further experimentation with the actual user to customize and tailor the system to their needs and capabilities.
- A way to benchmark and resolve issues concerning the abbreviations completion method and how it can be incorporated and combined with the currently used prediction methods.
- Further investigation and work with the character-lever LSTMs in order to fine tune the models and make use of their increasing capabilities and potential. Also different kinds of architectures can be studied like word-level models or Convolutional Neural Networks.

- Research how having additional knowledge of the vocabulary, like a Part-Of-Speech tagged dictionary or knowing the root of each word, can improve the performance of the prediction engine.
- Investigate the use of different kind of sensors for the interface of the user with the system like, eye-tracking and electroencephalography technologies technologies or porting the system to a tablet to use a simple touch interface.
- Incorporate machine learning techniques, so that the prediction engine can be adjusted to better fit each individual user and their dictionary.
- Introduce big or common phrases set by the user that can be easily accessed as well as different text synthesis modes like conversation, formal, math, etc.

Another section of future work that is already being investigated includes changes to the UI inspired by the evaluation. This includes:

- The possibility of excluding words that were already suggested if the users keep typing characters. This will provide more possible suggestions also greatly helping with the cases that many words have the same linguistic root.
- The issue with the roots can also be resolved by modifying the UI to enable the user to hold their hand inside a word suggestion option in order to display more words have the same root.
- A way to be able to modify words already appended into the sentence to provide more flexibility.
- Add capability for punctuation marks. The available character options in the current 3-step method are 27 ($3 * 3 * 3$) with 24 of them being the letters of the alphabet and one being the space character. The two remaining options can be used to add a dot and a comma. Additional functionality has to be implemented for other punctuation marks.
- Search for better corpora that better fit the use cases desired.

Chapter 6

Appendix

Αξιολόγηση του συστήματος κατά την χρήση από άπειρους χρήστες.

Δίνεται χρόνος έως 5 λεπτά για την εξοικείωση με το Leap Motion και το UI.

1.Χρησιμοποιήστε το σύστημα για να συνθέσετε τις παρακάτω προτάσεις:

- Η ραδιοφωνική εκπομπή Ελληνοφρένεια παίζει κάθε μέρα κατά τις απογευματινές ώρες.
- Η τεχνολογική ανάπτυξη της σημερινής εποχής έχει ξεπεράσει κάθε προηγούμενο ρεκόρ.
- Σας ευχαριστώ πολύ για την συνεισφορά σας στην εργασία μου.

2. Γράψτε μια μικρή παράγραφο την οποία θα προσπαθήσετε στη συνέχεια να συνθέσετε μέσω του συστήματος.

---

3. Παρακαλώ γράψτε τα σχόλια σας για το UI, την χρήση του Leap Motion και το σύστημα πρόβλεψης (θετικά ή/και αρνητικά).

Bibliography

[1] A. Colgan, "How does the leap motion controller work," Leap Motion Blog, Leap Motion Inc. [Online]. Available: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

[2] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "Bci2000: a general-purpose brain-computer interface (bci) system," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 1034–1043, 2004.

[3] E. Buch, C. Weber, L. G. Cohen, C. Braun, M. A. Dimyan, T. Ard, J. Mellinger, A. Caria, S. Soekadar, A. Fourkas, and N. Birbaumer, "Think to move: a neuromagnetic brain-computer interface (bci) system for chronic stroke," *Stroke*, vol. 39, no. 3, pp. 910–917, 2008. [Online]. Available: http://stroke.ahajournals.org/content/39/3/910

[4] C. Guger, S. Daban, E. Sellers, C. Holzner, G. Krausz, R. Carabalona, F. Gramatica, and G. Edlinger, "How many people are able to control a p300-based brain–computer interface (bci)?" *Neuroscience Letters*, vol. 462, no. 1, pp. 94 – 98, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304394009008192

[5] S. Vaid, P. Singh, and C. Kaur, "Eeg signal analysis for bci interface: A review," in *2015 Fifth International Conference on Advanced Computing Communication Technologies*, Feb. 2015, pp. 143–147.

[6] M. Duvinage, T. Castermans, M. Petieau, T. Hoellinger, G. Cheron, and T. Dutoit, "Performance of the emotiv epoc headset for p300-based applications," *BioMedical Engineering OnLine*, vol. 12, no. 1, p. 56, Jun 2013. [Online]. Available: https://doi.org/10.1186/1475-925X-12-56

[7] M. Spüler, "A high-speed brain-computer interface (bci) using dry eeg electrodes," *PLOS ONE*, vol. 12, no. 2, pp. 1–12, 02 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0172400

[8] G. Welch and E. Foxlin, "Motion tracking: no silver bullet, but a respectable arsenal," *IEEE Computer Graphics and Applications*, vol. 22, no. 6, pp. 24–38, Nov 2002.

[9] N. Sebe, M. S. Lew, and T. S. Huang, "The state-of-the-art in human-computer interaction," in *Computer Vision in Human-Computer Interaction*, N. Sebe, M. Lew, and T. S. Huang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–6.

[10] "Leap Motion Controller," Leap Motion, Inc., 2015. [Online]. Available: hyyps://www.leapmotion.com

[11] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the leap motion controller," *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013. [Online]. Available: http://www.mdpi.com/1424-8220/13/5/6380

[12] J. Guna, G. Jakus, M. Pogačnik, S. Tomažič, and J. Sodnik, "An analysis of the precision and reliability of the leap motion sensor and its suitability for static and dynamic tracking," *Sensors*, vol. 14, no. 2, pp. 3702–3720, 2014. [Online]. Available: http://www.mdpi.com/1424-8220/14/2/3702

[13] L. E. Potter, J. Araullo, and L. Carter, "The leap motion controller: A view on sign language," in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, ser. OzCHI '13. New York, NY, USA: ACM, 2013, pp. 175–178. [Online]. Available: http://doi.acm.org/10.1145/2541016.2541072

[14] E. Dalmaijer, "Is the low-cost eyetribe eye tracker any good for research?" *PeerJ PrePrints*, vol. 2, p. e585v1, Nov. 2014. [Online]. Available: https://doi.org/10.7287/peerj.preprints.585v1

[15] W. Quevedo, A. Santana, G. Vayas, M. Navas, and M. Huerta, "System of evaluation for reading based on eye tracking," in *Emerging Technologies for Education: Second International Symposium, SETE 2017, Held in Conjunction with ICWL 2017, Cape Town, South Africa, September 20–22, 2017, Revised Selected Papers*, vol. 10676. Springer, 2017, p. 234.

[16] M. Su, C. Yeh, S. Lin, P. Wang, and S. Hou, "An implementation of an eye-blink-based communication aid for people with severe disabilities," in *2008 International Conference on Audio, Language and Image Processing*, July 2008, pp. 351–356.

[17] A. Dikaiou, V. Kosmidou, F. Tzima, A. Valsamidis, and L. J. Hadjileontiadis, "Sign2talk: a wearable sign language translation system for deaf or hearing-impaired people," *Microsoft Imagine Cup 2005 Final Report*, pp. 1–10, 2005. [Online]. Available: http://www.microsoftsrb.rs/download/obrazovanje/academic/imaginecup2005/ImagineCup2005_EE_finale.pdf

[18] S. Ghosh, O. Vinyals, B. Strope, S. Roy, T. Dean, and L. P. Heck, "Contextual LSTM (CLSTM) models for large scale NLP tasks," *CoRR*, vol. abs/1602.06291, 2016. [Online]. Available: http://arxiv.org/abs/1602.06291

[19] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-Aware Neural Language Models," *ArXiv e-prints*, Aug. 2015. [Online]. Available: http://adsabs.harvard.edu/abs/2015arXiv150806615K

[20] H. Al-Mubaid, "A learning-classification based approach for word prediction." *Int. Arab J. Inf. Technol.*, vol. 4, no. 3, pp. 264–271, 2007.

[21] Y. Even-Zohar and D. Roth, "A classification approach to word prediction," in *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, ser. NAACL 2000. Stroudsburg, PA, USA: Association for Computational

Linguistics, 2000, pp. 124–131. [Online]. Available: http://dl.acm.org/citation.cfm?id=974305.974322

[22] F. E. Sandnes, "Reflective text entry: a simple low effort predictive input method based on flexible abbreviations," *Procedia Computer Science*, vol. 67, pp. 105–112, 2015.

[23] T. D'albis, R. Blatt, R. Tedesco, L. Sbattella, and M. Matteucci, "A predictive speller controlled by a brain-computer interface based on motor imagery," *ACM Trans. Comput.-Hum. Interact.*, vol. 19, no. 3, pp. 20:1–20:25, Oct. 2012. [Online]. Available: http://doi.acm.org/10.1145/2362364.2362368

[24] S. Bickel, P. Haider, and T. Scheffer, "Predicting sentences using n-gram language models," in *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, ser. HLT '05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, pp. 193–200. [Online]. Available: https://doi.org/10.3115/1220575.1220600

[25] C. Spiccia, A. Augello, and G. Pilato, "A word prediction methodology based on posgrams," in *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, A. Fred, J. L. Dietz, D. Aveiro, K. Liu, and J. Filipe, Eds. Cham: Springer International Publishing, 2016, pp. 139–154.

[26] J. Matiasek, M. Baroni, and H. Trost, "Fasty — a multi-lingual approach to text prediction," in *Computers Helping People with Special Needs*, K. Miesenberger, J. Klaus, and W. Zagler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 243–250.

[27] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the leap motion controller," *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3690061/

[28] "Leap Motion App Store," Leap Motion, Inc. [Online]. Available: https://gallery.leapmotion.com/category/app-store/

[29] *Unity User Manual*, 2017th ed., Unity Technologies. [Online]. Available: https://docs.unity3d.com/Manual/index.html

[30] *Unity Scripting API Manual*, 2017th ed., Unity Technologies. [Online]. Available: https://docs.unity3d.com/ScriptReference/index.html

[31] "Unity Answers," Unity Technologies. [Online]. Available: https://answers.unity.com/index.html

[32] *Leap Motion Unity SDK Manual*, 4.4.0 ed., Leap Motion, Inc. [Online]. Available: https://leapmotion.github.io/UnityModules/

[33] "Leap Motion Community Forum," Leap Motion, Inc. [Online]. Available: https://forums.leapmotion.com/

[34] "ResponsiveVoice Text To Speech API," ResponsiveVoice, 2018. [Online]. Available: https://responsivevoice.org/api/

[35] R. Dahl, "Node.js," Joyent, Version 8.11.1. [Online]. Available: https://nodejs.ord/en/

[36] I. Z. Schlueter, "npm," Verion 5.6.0. [Online]. Available: https://www.npmjs.com/

[37] D. Goldhahn, T. Eckart, and U. Quasthoff, "Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages." in *Proceedings of the 8th International Language Ressources and Evaluation (LREC'12), 2012.*

[38] School of Informatics, University of Edinburgh, "N-gram extraction tool," 2004. [Online]. Available: https://homepages.inf.ed.ac.uk/lzhang10/ngram.html

[39] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," Andrej Karpathy Blog, 2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[40] ——, "Multi-layer recurrent neural networks (lstm, gru, rnn) for character-level language models in torch," Karpathy's Github Repository, 2016. [Online]. Available: https://github.com/karpathy/char-rnn

[41] S. Ozair, "Multi-layer recurrent neural networks (lstm, rnn) for character-level language models in python using tensorflow," Sherjil Ozair's Github Repository, 2018. [Online]. Available: https://github.com/sherjilozair/char-rnn-tensorflow