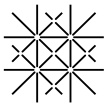


University of Basel

Word-Gesture Typing in Virtual Realty

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Databases and Information Systems Group
https://dbis.dmi.unibas.ch/

Examiner: Prof. Dr. Heiko Schuldt
Supervisor: Florian Spiess

Philipp Weber
phil.weber@stud.unibas.ch
19-051-697

20.06.2022

Acknowledgments

I would like to thank Prof. Dr. Heiko Schuldt for the opportunity to write this thesis in the Databases and Information Systems Groups. Further, I would especially like to thank my supervisor Florian Spiess for giving me great advice, helpful feedback and clear answers to the many questions I had. Finally, I would also like to thank my family, friends and fellow students for helping me with the evaluation and giving me motivation to give my best for this thesis.

Abstract

Text-entry is one of the most common forms of computer-human interaction and indispensable for many tasks such as word processing and some approaches to multimedia retrieval. The conventional keyboards everybody knows have been long established as the main text input method for desktop and laptop computers and even for touchscreen based devices they are very useful. But when it comes to virtual reality (VR) and augmented reality (AR), conventional keyboards might not be the best solution. As things stand now, these two technologies lack of tactile feedback and accurate finger tracking. As a result, text input for VR and AR is still an area of active research. In recent years, word-gesture keyboards (also called slide-to-type keyboards) have been introduced in most major smartphone operating systems. These keyboards do not work with tapping on single keys to input single characters but with gestures to input whole words. The question now is whether they can also perform well in VR and AR. In this thesis we implement a word-gesture keyboard for VR. We evaluate it by measuring the writing speed, some error rates and compare these with existing work. We show that a word-gesture keyboard can be useful as text input method for Virtual Reality.

Table of Contents

1 Introduction

1.1 Motivation

Lots of virtual reality applications need some kind of text input method. While most of the time this function is provided by a non-physical conventional keyboard implemented for VR, this might not be the most convenient solution. To use this kind of keyboard, a user has to tap on single letter keys to input single characters. Even though most of the text inputs in VR applications may not be very long, it can still be exhausting for the arms to input these. The conventional keyboard in VR applications often has a bigger scale than a physical one in reality. That said, a user has to move quite a distance with their arms and always move up and down to not accidentally hit a wrong key. Now, there is the idea to develop another type of keyboard, a so-called word-gesture keyboard, which is possibly more practical to use in VR. Such a keyboard looks more or less like a conventional one but instead of tapping on the different keys, words are written with gestures. With a word-gesture keyboard, the text input could become more comfortable. A user would not need to move their arms up and down, they could just move on a flat plane from one key to another. Such a keyboard could also be smaller because the precision is not as important as it is for the conventional keyboard. For example, if we tap in the middle of two keys on a conventional keyboard, we cannot tell which one to take for the input. But with a word-gesture keyboard, where we work with distances and graphs (more on this later), it does not have that much of an impact. Therefore, a smaller keyboard is possible, and we do not have to move our arms that much. Hence, it might be less exhausting to write with a word-gesture keyboard.
To show the power of such keyboards in a non-VR environment, we can make a comparison between the best possible performance on a conventional keyboard with the QWERTY layout and a word-gesture keyboard with the ATOMIK layout. MacKenzie and Zhang [6] found that after about 17 hours of practicing, the user of a conventional keyboard with QWERTY layout could input about 45 words per minute. On the other hand, Zhai and Kristensson [4] measured in their experiment with a word-gesture keyboard with the ATOMIK layout a record input speed of about 52 to 86 words per minute. This shows that the potential of such word-gesture keyboards is high and one could write really fast after some training. Therefore, we may ask ourselves if this could also be an efficient text input method for VR and AR.

1.2 Goals

For this thesis, we have two main goals. The first one is to develop a word-gesture keyboard. It has to work with vitrivr-VR and has to be available as open-source. It should also be available as a Unity package so that other developers can use it in their Unity projects as well.
The second goal is to evaluate said keyboard. The evaluation will be conducted according to current research standards with the usage of the MacKenzie phrase set. With the evaluation, we want to find out whether a word-gesture keyboard is a viable text input method for VR. To anaylze this, we need to take a look at the writing speed and different error rates.

2
Background

In this chapter we introduce the environment for which the word-gesture keyboard is mainly developed, some things about conventional and word-gesture keyboards in general and SHARK[2].

2.1 vitrivr-VR and UnityVR

vitrivr[1] is an "open source full stack content-based multimedia retrieval system"[2]. It supports video, image, audio and 3D collections. It also features a very broad set of query paradigms that are supported. vitrivr is developed by the Database and Information Systems group[3] (dbis) of the university of Basel. For our thesis, we use the VR part of vitrivr, namely vitrivr-VR, which is being developed in Unity[4].

Unity is a tool for developers where one can create projects in 2D, 3D and VR. To a certain degree Unity is free to use. Developers can provide assets and Unity packages. These can be either free to use or have to be bought. Another developer then can import and use these in their own Unity projects. The main language used in Unity is C#. A developer can write such C# scripts and if needed attach them to objects in a scene. These scripts can control the objects and what they are doing when a user interacts with them or something particular happens.

2.2 Conventional Keyboard

In this thesis, when we talk about a conventional keyboard, we do not look at its type of construction, at any special layouts or whether it is a mechanical keyboard or not. We define the term "conventional keyboard" as the most used keyboard type, the one where a user has to input every single letter by pressing or tapping a single key. There are two different kinds of conventional keyboards. One is the physical variant. This one is used for most desktop

[1] https://vitrivr.org/
[2] https://dbis.dmi.unibas.ch/research/projects/vitrivr-project/
[3] https://dbis.dmi.unibas.ch
[4] https://unity.com

and laptop computers. The other variant is a so-called soft keyboard. This is an on-screen conventional keyboard that is mostly used with phones, tablets and other touchscreen-based devices. It has the same functionality as the physical conventional keyboard but instead of pressing a physical key, one has to tap on the screen at the right place.

2.2.1 Disadvantages in VR/AR

When it comes to VR and AR, it seems that this is not the best method to input text. One reason for this is that tactile feedback and accurate finger tracking is currently lacking in VR and AR. While this could be improved during the next years, it is not really there yet. Another reason is the size of such keyboards in VR. A user has to tap on the keys with their controllers. If the keys are too close together, it might cause a problem in recognizing which one the user wants to press. Therefore, there needs to be either bigger keys or bigger spaces between two adjacent keys. This results in a bigger keyboard, which in turn ends in more needed movement with the arms. If the user has to move their arms a lot to input some text, this can quickly become exhausting.

2.3 Word-Gesture Keyboard

A word-gesture keyboard may look pretty much the same as a conventional keyboard but works quite different. First of all, it does not exist in a hardware version like the conventional keyboard does. It is more like the soft keyboard version on a screen. Independent of the details of the implementation, every word-gesture keyboard works with gestures. This means, instead of tapping on single keys, the user has to draw one line or a shape on the keyboard. This will then be evaluated by an algorithm. It determines the closest word, the one with the most similar shape seen from different aspects, from a lexicon. For example, to input the word “science”, the user has to put the finger on the screen, where the “s-key” is displayed. Then they have to move, with the finger still on the screen, to the respective adjacent key with the correct character. At the end, the user has to take away their finger from the screen at the “e-key”. If the gesture is more or less good, the algorithm behind should now be able to calculate that “science” is the word the user intended to write. But if the gesture is done bad, it can happen that a wrong word is being calculated.

2.3.1 SHARK^2

SHARK^2 is a “large vocabulary shorthand writing system for pen-based computers” [4] developed by Shumin Zhai and Per-Ola Kristensson. It can compare a user inputted graph with a perfect graph of any word in a given lexicon.

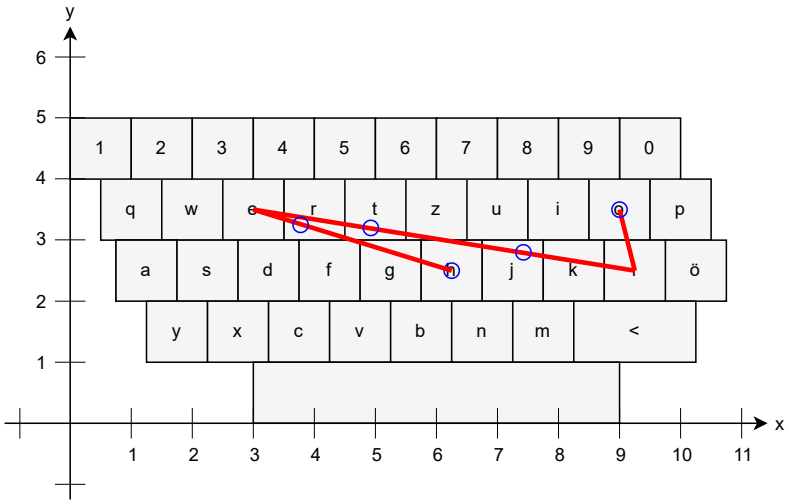


Figure 2.1: Perfect graph of the word "hello" with red lines. The 5 blue points show the sample points, if the graph was sampled with a number $N = 5$.

A perfect graph for a word is the graph that is produced if we start from the center of the word's first letter on the keyboard. Then we draw a straight line to the center of the next letter of the word and so on, until we reach the last letter. The resulting perfect graph for the word "hello" can be seen in Figure 2.1. A user inputted graph is the graph the user draws with their gesture on the keyboard. SHARK^2 then can find the word with the most similar perfect graph compared to the user input. To achieve this, it uses a multi-channel recognition system. The most important part are two core channels, a shape recognizer and a location recognizer, where different aspects of the graphs get looked at. There is also a channel that brings language information into the calculation. Additionally, the system uses some other tricks to achieve the best possible results.

2.3.1.1 Preconditions

The SHARK^2 system needs a lexicon that should be in the order of 10'000 words. Such a lexicon can be obtained through different methods. For example, the lexicon used to test SHARK^2 was mined from one of the authors' emails, but it could also just be a standard dictionary. For all the words in the lexicon, their perfect graphs have to be stored as well. Zhai and Kristensson [4] do not mention how they store these, but it would make sense, to only store N points for each graph, with N being the number of points that samples one. This can be justified by the fact that for later calculations with the graphs only these N points are needed per graph and not more.

2.3.1.2 Template Pruning

First of all, SHARK^2 uses template pruning. It compares the start and end positions of the perfect graph of each word in the lexicon with the input gesture from the user, both being normalized in shape and location. If either the start-to-start or end-to-end distance is

bigger than a given threshold, the checked word will be discarded and not further considered.

2.3.1.3 Shape Channel Recognition

The next step is to apply the shape recognizer. It compares the shapes of the perfect graph of each word in the lexicon with the user inputted graph. For this, an amount of N sampling points has to be calculated for every graph. These N points need to be equidistant. An example of this sampling can be seen in Figure 2.1.

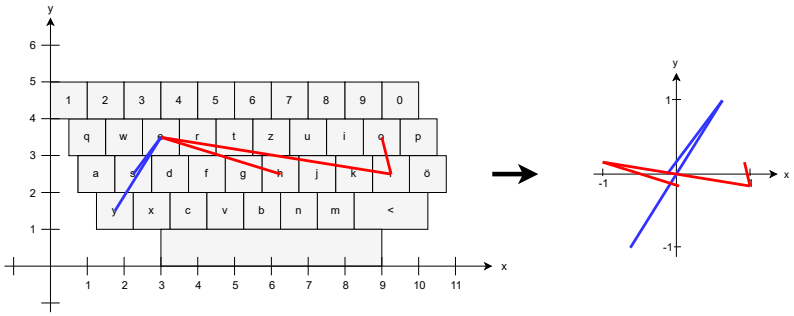


Figure 2.2: Normalization of the perfect graphs of the words “yes” and “hello” to a predetermined length of 2

Then they have to be normalized in scale and location as it is done in Figure 2.2. First of all, the middle point m of every graph's bounding box has to be calculated. Further, m has to be subtracted from every point, because this sets the middle point of the bounding box to $(0, 0)$, thus normalizes it in location. Then the graphs are all normalized in scale by scaling the largest side of the graph's bounding box to a predetermined length L:

$$s = \frac{L}{\max(W, H)} \tag{2.1}$$

W and H are the width and height of the graph's bounding box. Then, all points' positions have to be multiplied by s to get the normalized positions.

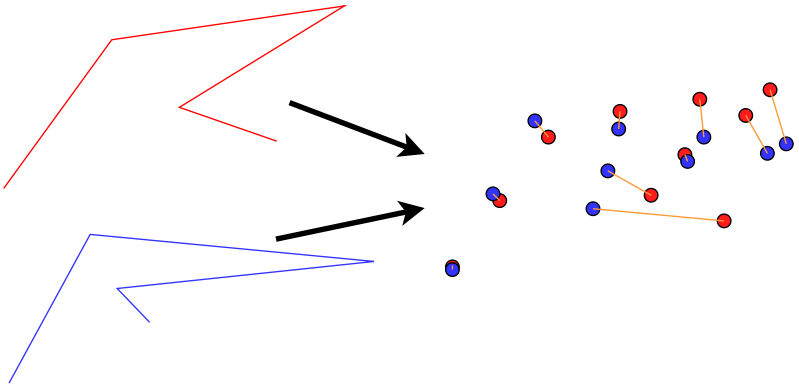


Figure 2.3: Calculating the distance of two random graphs. This is the summed up distance of all distances between the pairs of points. A pair consists of two i-th points, one from every graph, where $i \in \{1, N\}$.

Now, the distance between the normalized user inputted graph and every word's normalized perfect graph has to be calculated. To do so, we use the following formula:

$$x_S = \frac{1}{N} \sum_{i=1}^{N} \|u_i - t_i\|_2 \tag{2.2}$$

where u_i is the i-th point of the user inputted graph and t_i the i-th point of a word's perfect graph. The result x_S is the so-called "proportional shape matching distance" [4].

Now, one could think that this is enough and with the application of the template pruning and shape channel recognition the word is perfectly determined. This is not the case. The authors state that words can have a similar or even the same shape as other words. They call these word pairs "confusion pairs". They found that, for example, on an ATOMIK layout with a lexicon of 20'000 words, 1'117 confusion pairs occur, if the starting and ending positions are not considered. If these are also considered with the shape, there is still a total of 537 confusion pairs.

2.3.1.4 Location Channel Recognition

To avoid a lot of these confusion pairs, the authors are using a second channel, the location recognizer. For the following formulas and calculations, the normalization of the graphs is not needed anymore. As the name states, it is about the location, where the graph lies in a coordinate system.
They use an algorithm that computes the distance of the user inputted graph u to the perfect graph t of every word in the lexicon. The location channel distance is defined as:

$$x_L = \sum_{i=1}^{N} \alpha(i)\delta(i) \tag{2.3}$$

where N is the number of points used to sample a graph and $\alpha(i)$ with $i \in (1, N)$ are weights for the different point-to-point distances, such that $\sum_{i=1}^{N} \alpha(i) = 1$. These weights can be set in various ways. For SHARK^2 the authors use a function that gives the lowest weight to the middle point-to-point distance. The weight increases linearly towards the two ends for the other point-to-point distances. Another way could be through training with a large amount of data. Lastly, $\delta(i)$ is defined by the following formula:

$$\delta(i) = \begin{cases} 0, & D(u,t) = 0 \wedge D(t,u) = 0 \\ \|u_i - t_i\|_2, & \text{otherwise} \end{cases} \tag{2.4}$$

where u_i is the i-th point of u and t_i the i-th point of t. Further, D is defined as:

$$D(p,q) = \sum_{i=1}^{N} \max(d(p_i, q) - r, 0) \tag{2.5}$$

where r is the radius of a key on the keyboard and d is defined as:

$$d(p_i, q) = \min(\|p_i - q_1\|_2, \|p_i - q_2\|_2, \ldots, \|p_i - q_N\|_2) \tag{2.6}$$

For all these formulas N is the number of points used to sample a graph. The "trick", for which the authors use these formulas, is quite simple. They state that they form something like an "invisible" tunnel of one key width that contains all keys used to write a certain word. A perfect distance score of zero is given, when all the sampled points of the user inputted graph lie within the tunnel of t. If this is not the case, the distance score for t with respect to the user inputted graph u is set to the sum of the spatial point-to-point distances. This means N distances have to be calculated and summed up. These are the distances between the i-th point of u and the i-th point of t for $i \in \{1, N\}$, as seen in the example in Figure 2.3.

2.3.1.5 Channel Integration

With the two distances x_S and x_L, the most probable word, the user intended to write, can be calculated pretty well. The authors assume that the distance from a user inputted graph to the perfect graph of a word follows a Gaussian distribution. This means, if the user inputted gesture has distance x to a perfect graph of a word w, the probability that w is the intended word can be calculated using the Gaussian probability density function:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{2.7}$$

One important thing here is that this calculation has to be performed two times per word because it has to be done for x_S and x_L. They set $\mu = 0$ and σ can be obtained through training from large amount of data. Here, σ can be seen as the sensitivity of a channel. If, for example, σ is equal to one key radius, the words, whose perfect graphs have a greater distance to the user inputted graph than one key width, have practically zero probability of being the intended word. For SHARK^2 the authors use σ as parameter to adjust the weight of contribution of each channel. They also use σ for further pruning. They discard all candidate words, whose distance x is bigger than 2σ. For the candidate words $w \in W$, that have not been discarded until now, the marginalized probability to be the intended word is:

$$p'(w) = \frac{p(x)}{\sum\limits_{i \in W} p(i)} \tag{2.8}$$

Note that this calculation also needs to be performed twice per word, once for the location channel and once for the shape channel. The last step of the channel integration is to integrate the probabilities from the two channels using Bayes' rule:

$$c(w) = \frac{p'_S(w)p'_L(w)}{\sum\limits_{i \in W_S \cap W_L} p'_S(i)p'_L(i)} \tag{2.9}$$

The final result $c(w)$ is now the confidence score for the word w.

2.3.1.6 Further Steps To Improve The Results

There are also some further steps that can be made but we do not implement in our algorithm. In this section, we discuss briefly about these.

Dynamic Channel Weighting by Gesturing Speed

The authors say that the user can draw a graph either on visual guidance from the keyboard (looking for the next letter of a word on the keyboard) or recall from memory. A graph drawn by visual guidance results in a higher location distance score than a graph drawn from memory recall. If a user draws a graph by memory recall, the location distance score will be poor and the focus lays more on the shape. Therefore, they suggest a dynamic weighting of the two channels. In general, graphs drawn by memory recall are faster than visual guided ones. Hence, the gesture completion time should tell how heavy each of the two channels should be weighted in the final selection. The time to complete a graph for a word obviously depends on its length and complexity. The authors use Fitts' law to calculate the normative writing time for a perfect graph. This law predicts that the time one needs to move to a target area is a function of the ratio between the distance to the target and the width of the target. They use this result together with the actual graph production time to modify σ used in formula 2.7.

Using Language Information

The authors achieved quite good performance with the two core channels, but there still might be conflicting words. To prevent these, the authors suggest to also use language context. For SHARK^2 smoothed bigrams are used as the language model, which is then used to rearrange the N-best list of words received before (the N words with the highest confidence score).

2.3.1.7 Other VR Input Methods

Later on, we will need the text input methods used by Boletsis and Kongsvik [2]. Therefore, we briefly introduce them here.

Raycasting keyboard: One of the most popular VR input method. A user casts a ray from their controller and has to aim for the keyboard. When aiming at a key, a button of the controller can be pressed to input a character. In this case, it is a two-handed one, which means a user can use both hands simultaneously.

Drum-like keyboard: The controllers are drumsticks. A user has to click on the keyboard keys with these sticks as they were their hands.

Head-directed input keybpard: The user controls a pointer with their head movement. They have to aim for a key and then press a certain button on the controller to select it.

Split keyboard: A virtual keyboard is split in two parts, where each is assigned to a controller. The keyboard keys can be selected by the touch-sensitive trackpad of the controllers (Vive controllers) and characters can be inputted by pressing the trackpad button.

All of these keyboards have in common that only single characters can be inputted and not whole words as one.

3
Implementation

In this section, we will introduce how we implement a word-gesture keyboard using Unity, a Python script and a simplified version of the word detecting algorithm used in SHARK^2.

3.1 Word Graph Generator

As previously discussed in Section 2.3.1, to make the word detecting algorithm used in SHARK^2 work, the perfect graphs for all words in a lexicon must be precomputed. Therefore, we need a script that either creates or overwrites a file for every available keyboard layout. It has to write the words included in the lexicon together with the corresponding N sampled points of their graphs. Per line, such a file contains a word, then a certain number N of points from the word's perfect graph followed by the same points but normalized. Normalized here means the same as mentioned in the background Section 2.3.1.3.
To run the script, the user has to provide the name of the layout that they want to create the perfect graphs for. Additionally, they also have to write the name of the text file containing all the words, which is the lexicon. The script then either creates a new file named "graph_*layout*.txt" or, if a file with this name already exists, it deletes its content. Then it fills the file line by line as mentioned above. In further sections, we will call a file of this type "layout file". The script can only be executed for one layout at a time. Hence, if there are multiple available layouts for our word-gesture keyboard, the user has to run the script for every single one. One thing the script pays attention to, is if a word can be written with used layout. If there are words in the lexicon that can not be written with the given layout, the script skips them. Hence, there will be no line in the file for said word.

3.2 Used Algorithm

For our word-gesture keyboard we use a simplified version of the algorithm used in SHARK^2. This means, we also work with two core channels, a location recognizer and a shape recognizer but do not implement everything from the algorithm used in SHARK^2. The shape recognizer is to calculate the deviation from the user inputted graph and a perfect graph from a word with respect to their shape. The location recognizer is for the same thing

but not with respect to the shape, but rather the position on the keyboard. When looking at the shape, we normalize the graphs in a specific way as explained in Section 2.3.1.3, so the position, where they exactly lie on the keyboard, does not matter. When looking at the location, we look at the graphs as they are without normalizing or changing anything.
As in the SHARK^2 system we also use the start and end positions of the graphs as a first pruning method. The difference is that for SHARK^2 the authors chose to normalize all the graphs in scale and translation before computing the start-to-start and end-to-end distances. In our algorithm, we do not normalize the graphs but just look at the start and end positions of a user inputted graph and a word's perfect graph. As threshold we set the width of a key. This means, if either the start-to-start or end-to-end distance is bigger than a key width, the word to which this applies, gets discarded.
Another thing we implement differently is σ. For the channel integration formula 2.7, they use σ in SHARK^2 as a parameter. They determine its value by the gesturing speed mentioned in Section 2.3.1.6. We do not use the gesturing speed in our algorithm. For the location channel (in the formula 2.7) we use the radius of a key on the keyboard as value for σ. For the shape channel (in the formula 2.7) we use a value that equals to the radius of a normalized key. That means, a small graph will have a bigger σ than a big graph because we normalize the graph's longer bounding box side to a fixed length. Therefore, a small graph gets stretched, whereby a big graph gets contracted.
As mentioned in the beginning of this section, we use a simplified form of the algorithm used in SHARK^2. That being said, we do currently not use any language information nor dynamic channel weighting by gesturing speed. We do not use any language information, because that would have gone beyond the scope of this project. The dynamic channel integration was not implemented by us because, in our opinion, the value of sigma, as we chose it, is fine for the purpose of our word-gesture keyboard in VR.

3.3 Functions

In this section, we will present the functions our word-gesture keyboard provides.

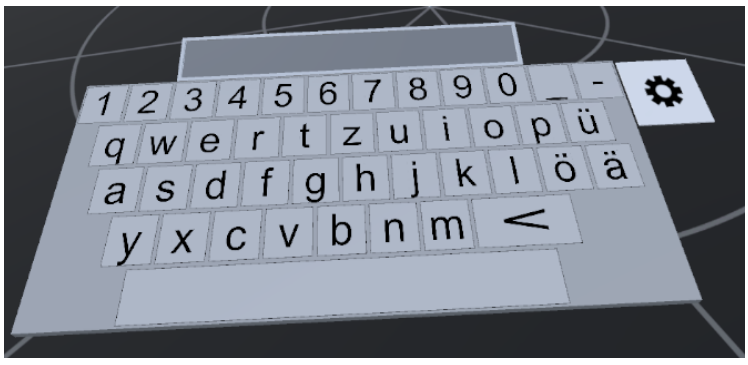
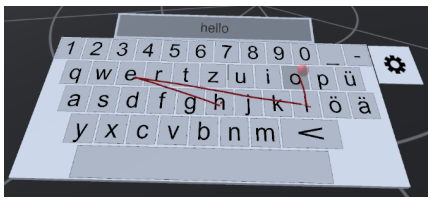


Figure 3.1: Word-gesture keyboard in its "normal" form without anything selected

3.3.1 Text input

There should be two text input methods for the user. One is the input of words with gestures, the other one is the input of single characters by tapping single keys.



(a) The user is writing the word "hello" with a gesture

(b) The user gets recommended words for "and"

Figure 3.2: In (a) a user is writing the word "hello" with a gesture, but did not release the trigger yet and can still see the red line, which indicates the path they draw. In (b) a user already wrote the word "and" with a gesture and can now see three word suggestions they could choose from.

3.3.1.1 With Gestures

The most important function our word-gesture keyboard provides, is the writing of words with gestures. A user can press and hold the trigger button of a VR controller inside the keyboard's hitbox and start making a gesture on it. The user will see a red line, as seen in Fig. 3.2(a), that is drawn on the keyboard where they gesture. This helps them to keep track of the line they drew. When the user wants to finish the gesture, they need to release the trigger of the controller. At this moment, our program starts to evaluate up to 5 words with the best match to the user inputted graph. The one with the highest confidence score will be written into a chosen text field. Up to 4 other words are displayed at the keyboard Fig. 3.2(b), such that the user can also choose between them by touching the right button with their controller. When they choose one of these word suggestions, the word that has been written into the text field before, is replaced by the chosen word and the button, where the chosen word was written, will then display the replaced word.

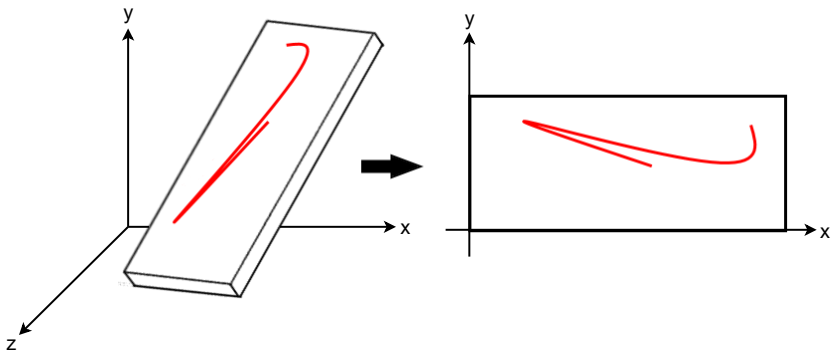


Figure 3.3: Transformation of a graph/gesture from the 3D space into the 2D space

There are some challenges in VR we have to master. First of all, in VR the user is in a three-dimensional space. Thus, when they make a gesture, they could move up and down with their controller and do not have to stay on the keyboard. For this we need to look that the line of a gesture still lies on the keyboard as if the user would draw on a piece of paper by using a pen. The next problem that follows is that a user can rotate the keyboard as they want in VR. Therefore, the drawn graph will most of the time not be lying perfectly on the x-z-plane even though it is drawn flat on the keyboard's surface. This means, we have to transform all the points into the x-z-plane to be able to further process these gestures. Another problem of drawing gestures is the amount of points. Technically seen, a new point is saved for each frame in which the user draws a gesture. Thus, we have to reduce this amount because it would be just too many points, and it would slow down the program massively. We implement a function that always keeps the new point. It either replaces the second-newest point with the new one or adds the new point to the point list. This is determined by the distance between the second-newest point and the third-newest one. If the distance between them is smaller than a minimum segment distance, there is a replacement otherwise there is an addition of the newest point.

3.3.1.2 As Single Characters

If the wanted word is not in the lexicon, there will not be any entry in any layout file. Therefore, our algorithm will not be able to get this wanted word as best match, hence it can not be written as gesture. For this case, we need a method to input single characters. Fortunately with our word-gesture keyboard this almost works without additional work. If the single letters are in the layout files, the user is technically seen able to write single letters with a gesture. This "gesture" would just be a point on the right key, hence a single click at the right position. But there might be a little inconvenience. This is caused by the fact, that we work with distances. The distance from one letter to another is not large. And if, for example, the user wants to write an "e" but presses the key with the "e" on it on its left side and not perfectly in the middle, our system will evaluate that aside from "e", also "w" and "we" are words the user might have intended to write (on a conventional QWERTZ or QWERTY layout). To avoid this, our system checks, if the user inputted graph's bounding box is smaller than the radius of a key on the keyboard. If so, it recognizes that the user wants to write a single letter, and then takes the best match. To get back to the example, "we" would be discarded and "e" would get a higher score than "w", because of a smaller location channel distance. Therefore, the written "word" in this case would be "e".

3.3.1.3 Spaces and Backspaces

Important when writing more than one word are spaces. The proper way to automatically set them would be that if a word of more than one character is written with a gesture, for the next input, no matter if it is a single character or a whole word, a space is put before it. Some exceptions where the space would be omitted is, for example, if after a word a ".", "," or "?" is written. Also, at the start of a sentence it should not set a space. If a single character was written, and the next input is a word consisting of more than one character

it should set a space before it, but if another single character follows, it should not put one. In our implementation, when the user writes a word that consists of more than one character, a space gets put behind the word. This means, if they write a second word or a single character after, they do not have to put a space manually. But if they write a single character, no space is put automatically after it. We do it this way because the keyboard does not know what was already inputted in a textfield. For example, if a user writes something into a textfield and then switches to another one, there could be problems because the keyboard itself does not know that the textfield has been changed and then might write a space where there should not be one.
When the user writes a word and then uses the backspace key, the whole word including the space gets deleted. After this first word, only single characters will be deleted afterwards.

3.3.2 Create New Layouts

Another function is the creation of custom layouts. While this would not be a necessary function to reach the core goal of inputting text, it can still be helpful for users to feel more confident using the keyboard. For example, if we only implement the QWERTZ layout, a user that is only using keyboards with the QWERTY, ATOMIK or any other layout than QWERTZ, might have problems using it. The problem is that we do not know every layout all users are using for non-VR text input. Therefore, it is good that a user can decide by themselves what kind of layout they prefer to use and simply create it for our word-gesture keyboard.
To create new layouts, a text file that contains all available layouts exists. The user can create as many new layouts as they want to. To create a new one, the user has to write the new layout's name on the first new line. On the following lines they have to write the characters in the order in which they want to have them on the keyboard. Then there have to follow two "$" and a number. This number will put a space from the left side. So if it is for example a 2, the line will be indented by two key widths. All the Unicode characters are working except for two. In the current implementation, one whitespace is used to declare the position of the spacebar and the "<" character is used for the backspace key. This file gets read at the start of the program, so it cannot be edited while the program is running, or to be precise, the changes will not be recognized during runtime.

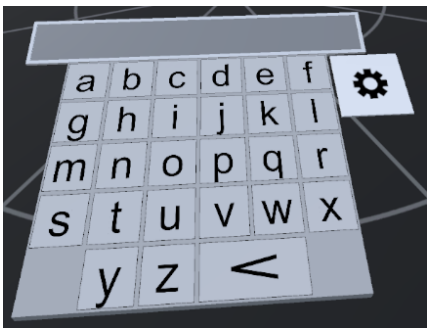
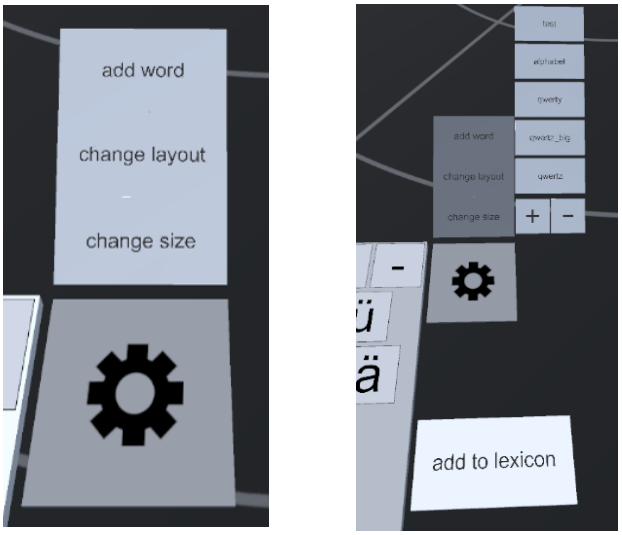


Figure 3.4: Experimental alphabet keyboard

3.3.3 Options



(a) The pressed options button with the three main options

(b) The pressed options button with all sub-buttons

Figure 3.5: The options button with the main buttons not activated in (a) and activated in (b)

To let the user use the other functions, we implement an options button for our keyboard, which is marked with a black gear as seen in Fig. 3.5(a). The user can click on it with the trigger button of the controller when being in its hitbox. Then, three new buttons appear, each for one currently available option. One is to add new words, one to change the currently selected layout and the last one to scale the keyboard.

3.3.3.1 Add Words

Figure 3.6: “vitrivr” written with single character inputs in the “add word mode”

The whole system works with a lexicon full of words and only these can be written with gestures. There will be words the user wants to write that are not yet in said lexicon, hence

they can not be written with gestures. For this case, we implement a function such that the user can add new words. They can access it via the options button. Then they have to press the button where it says “add word”. When this button is pressed, another button appears on the right side of the keyboard, the “add to lexicon” button. Now the user has to input their intended word with single characters. It will be displayed as seen in Figure 3.6. Then, when they press the “add to lexicon” button, the displayed word will be added to the lexicon text file, if it does not already exist and does not have a space in it. Additionally, for every available layout, the newly added word is also added to the corresponding layout file, so that the user can write the word with a gesture right after adding it. One additional thing we implement is that the word only gets added to the layout files if it can be written with the layout the file corresponds to. For example, if a user wanted to add the word “öffentlich” but they made a layout without the letter “ö”, this word could never be written with this layout, hence it would be unnecessary to have it in the corresponding layout file.

3.3.3.2 Change Scale and Layout

There are two other functions that can be found under the options button. One is to change the size of the keyboard. When the user clicks on the “change scale” button, a “+” and a “-” button appear. By pressing the “+” button, the keyboard gets bigger, by pressing the “-” button, the keyboard gets smaller. This can help the user to make the keyboard more handy for them. Depending on how they want to move their arm, they can make it slightly bigger or smaller. If it is bigger, the gestures can be made more precise, if it is smaller, gestures can be made faster. The “+” and “-” buttons can be seen in Fig. 3.5(b). The other function is the ability to change the layout. The user gets a list of all available ones when pressing the “change layout” button. They can then click on one of these buttons and the keyboard will change its appearance. The available layouts consist of the predefined ones and also of the newly added ones from the user. In Fig. 3.5(b) there is a long list of available layouts visible.

3.3.4 Moving, Rotating and Word Preview

Finally, we will briefly talk about the last two functions. One is the ability to grab and move the keyboard. This means, the user can grab the keyboard by being in its hitbox and pressing and holding the controller’s grip button. They can move it around in the room and rotate it as they want. If they release the grip button, the keyboard gets static and stays in the position.

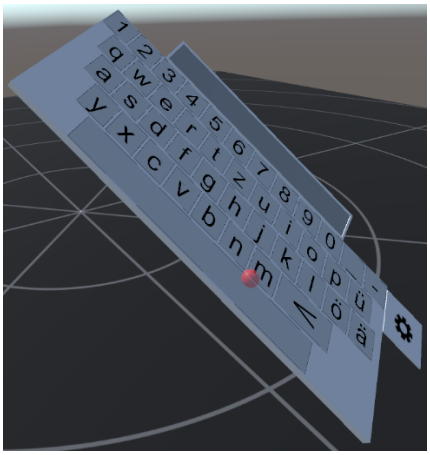


Figure 3.7: The user has the keyboard in the hand and moves and rotates it around.

This function can become handy, if the user wants to reposition themselves in the VR space. They can just take the keyboard with them. Also, they can choose to have it closer or further away, which can make it more convenient to use the keyboard. Rotating the keyboard can also help this purpose. Some users might prefer to have it more tilted than others.
The other function is that the keyboard shows a preview word in real time while a user makes a gesture. This means, it shows the word the algorithm would detect as best match, if the user released the trigger. This can be seen at the top of Fig. 3.2(a), where “hello” is written as “word preview”.

4
Evaluation

In this section we talk about the evaluation as a whole. We take a look at the phrase set we used, how we carried it out, the results that can be observed and compare it to existing results.

4.1 MacKenzie Phrase Set

For the evaluation we use the MacKenzie Phrase Set[5]. This is a set of 500 phrases. According to MacKenzie and Soukoreff [5], such a phrase set should use phrases of moderate length that are easy to remember and representative for the target language. The phrases of the MacKenzie Phrase Set do not contain any punctuation. Some of them use uppercase characters, but the authors mention, that participants can also be instructed to ignore the case of the characters.
Some statistics for the whole phrase set, also found in the original paper [5]: The MacKenzie phrase set consists of 500 phrases that have a minimum length of 16, a maximum length of 43 and an average length of 28.61 characters. On the whole, 2712 words were used, which consist of 1163 unique words. A phrase consists of a minimum of 1, a maximum of 13 and on average of 4.46 words.

4.2 Task of the Participants

The task of the participants is to copy 15 phrases from the MacKenzie Phrase Set. As they are not in a specific order, e.g. alphabetic order, we decide to take adjacent phrases and not random ones.

[5] http://www.yorku.ca/mack/PhraseSets.zip

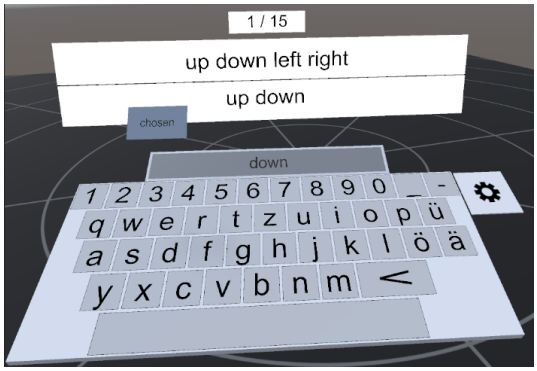


Figure 4.1: The textfield a participant sees. The phrase to copy is on top, the lower textfield displays the words the participant wrote.

The participants see two text fields as shown in Figure 4.1. On the top is the phrase to copy, on the bottom the words/phrase they write. If the given phrase matches the phrase the participant wrote, a tone sounds, such that the participants know when they finish one specific phrase. After that, a new phrase appears until 15 phrases are correctly inputted. If an incorrect word is entered, the participant either can use the word suggestions Fig. 3.2(b) or delete the wrong word and try to write it again. If a mistake is only noticed later on, the participants have to remove all words and characters up to and including the wrong word by using the backspace button. After this first step, in a second step, the participants get introduced to the scaling and the "add word" functions, which they could test afterwards. This is important, because we want to know if they find these functions useful and well implemented. The last step of the evaluation is to fill a questionnaire. First, it has some general questions about the participant's experience in VR. Then there is a block of twelve questions. The first ten are from the system usability scale, but the first one is changed a bit. The last two questions are for the previously mentioned functions. Per question, there are five possibilities to choose from, from 1 (strongly disagree) to 5 (strongly agree). The questions are structured in such a way that if the user is highly satisfied with everything, they would alternately make a cross at the 5 and 1.

4.3 Carry-out

To carry out the evaluation, we used two different VR systems. One was a setup with a HTC vive and HTC vive controllers. The other one included an Oculus Rift headset with corresponding controllers. Even though these are two different systems, it did not change much for the participants. In fact, only the controllers and their buttons differ a bit.
The participants consist of volunteers including computer science students, friends and family members. In total, eleven people got in touch with us and participated at our evaluation. Every participant got the same explanation to give everybody the same foundation of knowledge. Participants were provided the following information:
If they are close enough to the keyboard, then the color gets a bit brighter, and they are in the keyboard's hitbox. The keyboard is movable if they press and hold the controller's grip

button in the hitbox of the keyboard. If they release it, the keyboard gets static again and stays where it got put.
To write, they also have to be in the hitbox of the keyboard but not pressing and holding the grip button, but the trigger button. While pressing, they can make a gesture over the characters of the keyboard. They will see a red line where they move. If they release the trigger button, the system tries to find the best matching word and puts it in the text field. If they do a gesture and a word longer than one character is written, a space is automatically put behind the word. Single characters can be written by clicking on the key of the intended character. If they write a single character, no space is put behind it, and they have to do it on their own. In the English language, this is particularly important for the words "I" and "a". If they make a gesture and a word is written, there may be one to four other choosable words displayed in buttons. They can pick from them if the word written in the text field is not the intended one. The participants were warned about the word "the", because all the time "thee" would be written, therefore they would have to correct it every time.
If they use the backspace button after writing a word, the whole word gets deleted and afterwards only single characters get deleted.
They have enough time, and they should not hurry, but rather make sure that the inputted words are correct. Because if they are not correct, they have to use the backspace a lot of times, and they lose a lot of time.

In total for the eleven participants, 165 phrases were used. The average word length is 4.16 with the minimum length of 1 and a maximum length of 12 characters. In total, these phrases contain 856 words, 460 of which are unique. Before the carry-out we made sure that all the words in these phrases were in our word lexicon, such that the participants could write every word with a gesture.

4.4 Results

Now we talk about the results and some statistics we gained through the evaluation and compare them to exisiting results.

4.4.1 System Usability Scale

First, we begin with the results of the SUS questions. Note that we slightly changed the first question by appending "when I work in VR". These ten questions were:
1. I think that I would like to use this system frequently when I work in VR
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use

9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

Question	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
1	4	4	5	4	4	4	5	4	4	5	4
2	1	2	1	1	1	1	1	1	1	1	1
3	5	4	5	4	5	5	4	4	5	5	5
4	1	1	1	1	1	3	1	1	2	1	1
5	4	5	4	5	4	5	4	5	5	5	5
6	2	1	1	1	1	1	3	4	1	1	2
7	4	5	5	5	5	4	5	4	5	5	4
8	1	1	1	2	2	1	1	2	2	1	2
9	4	3	5	4	4	4	5	4	4	4	4
10	2	2	1	2	1	2	1	3	1	1	1
Score	85	85	97.5	87.5	90	85	90	75	90	97.5	87.5
	Excellent		Good								

	strongly agree for positive question, strongly disagree for negative questions
	agree for positive question, disagree for negative questions
	neutral
	disagree for positive question, agree for negative questions
	strongly disagree for positive question, strongly agree for negative questions

Figure 4.2: System Usability Scale results with all the given points per question from every participant

In Figure 4.2 green fields mark positive responses to questions, red ones negative responses and yellow ones neutral feedback. All in all, we can see a lot of green fields, which means the feedback to the SUS questions is quite good. Question 9 got the worst score but with it being a 4.09 out of 5, it is still fairly good. From the average values for each question, we can calculate the usability score. Every question with an even number is a negative one. This means, the highest possible score for them is 1 or "strongly disagree". For the other questions, a score of 5 or "strongly agree" is the best possible score. So, from the odd numbered questions 1 has to be subtracted from the average score. For the even numbered questions, the average score needs to be subtracted from 5. At the end, these ten newly calculated values have to be summed up and multiplied by 2.5. Our calculated usability score is 88.18 out of 100 possible points. This is a high score because from a score of 85.5 points one talks about an excellent system usability.

We also have two additional questions about the scale and the "add word" function:
11. The function to add words is well implemented and easy to use
12. The function to scale the keyboard is unnecessary

Question 11 got a score of 4.55 out of 5 and question 12 got a score of 2.18, whereby 1 would be ideal. We conclude from these two questions that the "add word" function makes a good

impression whereas the scale function does not perform so well.

4.4.2 Writing Speed

One important thing of our evaluation is to find out, how fast users can write with our word-gesture keyboard. As unit to measure these values, we take the "words per minute" WPM. We calculate the WPM with following formula:

$$WPM = \frac{|P|}{S} \times 60 \times \frac{1}{5} \tag{4.1}$$

where P is all the phrases a participant had to write. Therefore, $|P|$ is the number of characters they had to input. Finally, S is the time in seconds they used to write all 15 phrases.

Table 4.1: Average WPM, lowest WPM and highest WPM per participant. For the first three, we failed to record all data.

participant	average WPM	lowest WPM	highest WPM
1	11.46	-	-
2	12.19	-	-
3	13.06	-	-
4	11.61	5.30	25.50
5	12.58	6.83	21.65
6	10.29	5.27	19.00
7	12.42	6.10	24.41
8	16.06	8.28	30.74
9	13.36	7.96	24.15
10	17.12	7.98	24.45
11	10.07	4.71	14.55
average	12.75	6.55	23.06

In Table 4.1 we can see how fast on average the participants were able to write their 15 phrases. We do also list the lowest and highest value. To understand the values of Table 4.1 a bit better, we make a so-called boxplot for every participant, for whom we have the necessary data.

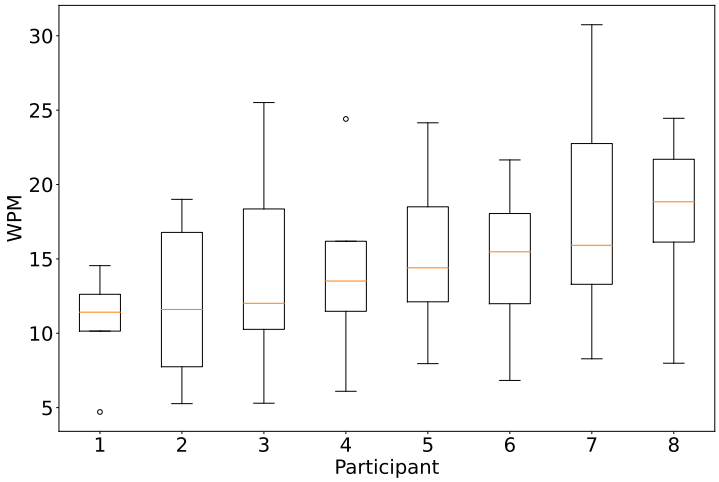


Figure 4.3: for participants 4-11: The upper and lower whiskers (or the outliers if the values are too extreme) show the maximum and minimum, the boxes show the 25% and 75% quartiles and the orange lines mark the median values. Each box corresponds to one participant. They are sorted by the average WPM value.

In Figure 4.3 we can see the higher the participant's median, most of the time, the higher the lowest WPM value. The lowest WPM values mostly come about because a participant made a mistake and had to delete a lot and basically write the phrase two times. On the other hand, most of the highest WPM values come about because a participant made no mistake in writing the phrase. The rectangle in the middle of the two bars shows how consistent or inconsistent a participant's writing speed was. The lower bound is the 25% quartile, the upper bound the 75% quartile. This means, if the rectangle is shorter, the writing speed was more consistent. We cannot find any clear trend that combines writing speed and consistency by looking at our measurements.
As mentioned above, if the participants only recognized an error at the beginning of the phrase when they almost finished it, they had to use a lot of backspaces and more or less had to write the phrase a second time. These can pull down the average WPM a lot. Therefore, we calculate the average WPM for the participants 4-11, where we only look at "perfect" phrases that were written without using a backspace. First of all, the average WPM for these eight participants over all phrases is 12.94. The average WPM of these "perfect" phrases is 16.53, which is about 28% higher. It is so much higher because the participants lost a lot of time using backspaces and correcting errors.

Now, these values alone do not tell us much about whether these are good results. To get a better view on this we compare our results with existing results from Boletsis and Kongsvik [2]. In their paper they evaluate four different VR input methods, a raycasting, a drum-like, a head-directed input and a split keyboard. More about these can be read in Section 2.3.1.7. Another paper we want to compare the WPM with is from Chen et al. [3]. One of the keyboards they evaluate is a word-gesture keyboard that works with rays. A

user has to point with a ray from the controller to the keyboard and make gestures on it.

Table 4.2: WPM for different VR text input methods

text input method	WPM
Drum-like keyboard	21.01
Raycasting keyboard	16.65
Word-gesture keyboard (ours) with "perfect" phrases only	16.53
Word-gesture raycast keyboard	16.43
Word-gesture keyboard (ours)	12.75
Head-directed input keyboard	10.83
Split keyboard	10.17

In Table 4.2 we listed the results of Boletsis and Kongsvik [2], Chen et al. [3] and our two measurements of the WPM value. Both other papers had a similar approach to the evaluation as we did, with two differences. Firstly, they used the same ten phrases for every participant and keyboard type. Secondly, they did not force the participants to write the phrases correctly but allowed errors. As we can see, our keyboard lines up in the lower half whereas the average value of only the "perfect" phrases lines up better. This means, if we did not force the participants to write phrases perfectly correct and chose another evaluation approach, our keyboard could keep up with the others regarding the writing speed. All in all, we can say that our keyboard is not the slowest one but there are faster ones especially for beginners.

The next thing we want to find out is if the writing speed of the participants has something to do with their experience with VR writing on one hand and experience with word-gesture keyboards on the other. Our prediction is that the participants that are experienced with both, are the fastest and the ones not experienced with neither of them are the slowest.

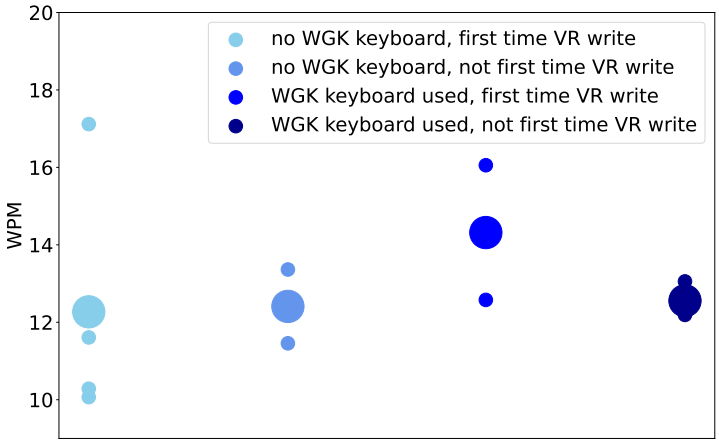


Figure 4.4: Each small dot shows the average WPM for a user. The big dots represent the average WPM per group. The colors are for the different groups of experience in VR writing and word-gesture keyboards

In Figure 4.4 we can see that against our predictions the prior knowledge, some participants have, did not really help them to write faster. In fact, the fastest group was the one that is experienced with word-gesture keyboards but not with writing in VR. We can also observe that the participant with the highest average WPM is in the group that has no experience with any of these two things. But we think these results might not be too representative due to the small sizes of the groups.

Another thing we analyze is the writing speed compared between different age groups.

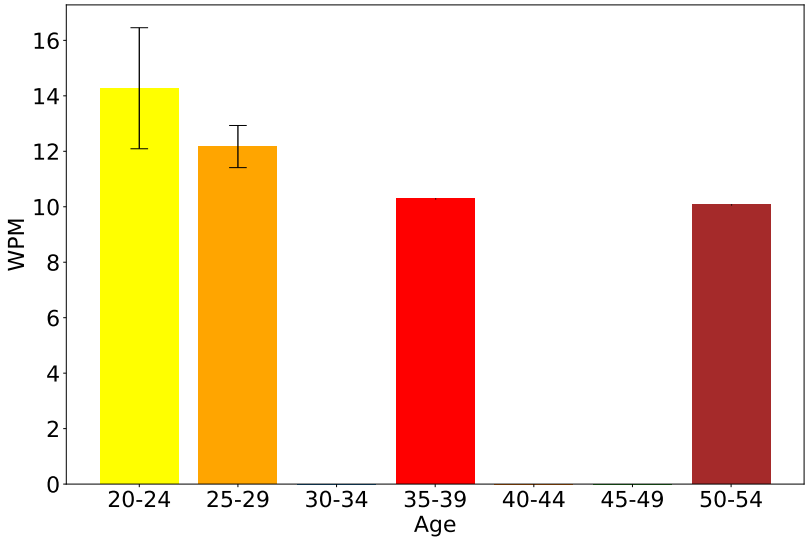


Figure 4.5: WPM values sorted by age groups of five years. The 20-24 and 25-29 groups also have a standard deviation error bar, because they consist of five and four participants. The 35-39 and 50-54 groups do not have an error bar since each of them only contains of one participant.

We might expect that older people are a bit slower in writing in VR than younger ones. Actually, this is what we can observe in Figure 4.5. At least among our participants, the WPM value decreases with the increase in age. This means, the older participants were a bit slower than the younger ones. But again, due to the low amount of participants in the older age groups, the results are not too representative and must be viewed with caution.

To end this section about writing speed, we check what is the fastest possible WPM we could reach. Zhai and Kristensson [4] made this experiment with four different phrases. We take the same phrases and compare their results with the ones we get. The four phrases are:

1. The quick brown fox jumps over the lazy dog
2. Ask not what your country can do for you
3. East west north south
4. Up down left right

Table 4.3: Fastest achieved WPM value per sentence for each person

Phrase	SHARK^2 Author A	SHARK^2 Author B	Author of this thesis
1	69.0	70.3	47.78
2	51.6	60.0	48.48
3	74.4	72.9	70.22
4	74.1	85.6	58.04

We can see that we can not reach the same maximal WPM but especially for phrase 3 we can get close to their value. The word-gesture keyboard is obviously faster if it is on a small screen and a finger is used. The distances that have to be covered are much smaller than in VR, hence it makes it easier to write fast. Nevertheless, we think we still reached a really high wpm with our keyboard and it shows the potential it has.

4.4.3 Error Rate

In this section, we will look at different error rates. First of all, we want to find out, which words caused most of the errors. Later, we will examine different errors rates caused by the participants.

4.4.3.1 Most Common Error Words

We look at all the words that were not the best match. We do not differentiate between the ones that were in the keyboard suggestions and the ones that were not. We call such a word “error word”. The top ten such words are listed in the following table:

Table 4.4: Most common error words

word	times wrong
the	53
is	17
to	14
of	12
in	9
for	8
do	5
all	3
more	3
see	2

As we can see in Table 4.4, the most common error word is “the”, which has a count of 53. In this list, we can also see other words whose errors could be avoided by a better implementation of the word detecting algorithm. For example “in” and “more” are such words. The best matches, when a participant wanted to write these words, were “thee”, “inn” and “moore”, which are words that do not appear that much in plain language. For other words like “to” or “of”, whose best matches were “too” and “off”, it is maybe not as

easy to get the right word because all of them appear fairly often in plain language.

4.4.3.2 Backspace Error Rate

The erroneous keystroke error rate (EKS ER) [1] measures the ratio of the total number of erroneous keystrokes to the length of the phrase that has to be written. In another paper, Chen et al. [3] use a metric derived from the EKS ER. Instead of the erroneous keystrokes, they take the number of times the backspace key was used. For our calculations, we will use the same metric as they do. We call this the "backspace error rate" and calculate it with following formula:

$$\text{backspace error rate} = \frac{\#\text{backspaces used}}{|P|} \times 100\% \tag{4.2}$$

where $|P|$ is the number of characters a participant had to write.

Table 4.5: Backspace error rate per participant and average over all

participant	backspace ER
1	20.95%
2	23.38%
3	9.71%
4	18.97%
5	12.40%
6	15.48%
7	15.48%
8	12.57%
9	6.54%
10	3.45%
11	0.24%
average	12.65%

In Table 4.5 the average backspace error rate of 12.65% is listed. This means, for every input of 100 correct characters about 13 backspaces were used. Again, as for the WPM previously, we compare our calculated backspace error rate with another existing work. As mentioned above, we calculate the same metric as Chen et al. did in their work [3]. Therefore, we compare it to their results:

Table 4.6: Comparison of backspace error rates

text input method	backspace error rate
Word-gesture keyboard (ours)	12.65%
Word-gesture keyboard raycasting	15.64%

In Table 4.6 we can see that the two error rates are pretty much the same, although our rate is a bit better. A reason for this might be that they did not implement that a word can be deleted as a whole. For example, if the newest written word is wrong the user can just press the backspace key once and delete the whole word when using our system. Now, they possibly implemented that no matter the last input every character must be deleted

individually, which would result in more needed backspaces. It is still interesting that our value is lower because we forced the participants to correct all errors but Chen et al. did not. So, either their participants also corrected the errors when they were almost finished with a phrase or they did just have to use it more during the whole process of writing the phrases.

4.4.3.3 Total ER

The next error rate we calculate is the Total ER according to Soukoreff and MacKenzie [7]. For this we use following formula:

$$Total\ ER = \frac{INF + IF}{C + INF + IF} \times 100\% \tag{4.3}$$

where C is the number of correct characters in the transcribed text. The transcribed text consists of all phrases a participant "sends" before beginning with a new one. Next, INF denotes the minimum string distance between the transcribed text and the given phrases. The MSD value is the number of single character insertions, deletions or substitutions to get from the wrong string to the right one. For example, to get from "thee" to "the", we need one deletion, therefore the MSD value would be 1. Finally, IF denotes the keystrokes in the input stream (everything a participant inputted during their evaluation) that are not in the transcribed text and not editing keys.
In our evaluation, a participant could only go to the next phrase if the last one had no errors left. This means C is the number of characters a participant had to write and INF is 0. It is 0 because, as mentioned right before, the transcribed text had to be equal to the text they had to write. So, these two values are taken as mentioned in [7]. However, for IF we have to use a bit another value than suggested. Due to insufficient evaluation records, we do not have the full input stream. Therefore, we can not just count the characters that were written too much. Soukoreff and MacKenzie also mention a fourth value F that counts the number of editing keys (backspaces, delete, ...) used. They mention that $F = IF$. Therefore, we will assign F the number of backspaces used and set IF equal to F. Important to mention is that if counted correctly IF would be a bit larger because if one backspace is used, sometimes a whole word gets deleted. But this should not have happened too many times. This means it is not a bad approximation of the real IF value.

Table 4.7: Total ER per participant and average over all

participant	total ER
1	17.32%
2	18.95%
3	8.85%
4	15.94%
5	11.03%
6	13.40%
7	13.41%
8	11.17%
9	6.14%
10	3.33%
11	0.24%
average	10.89%

As we can see in Table 4.7, the average is 10.89%. The standard deviation for the total ER is 5.57. We can not really say if this is a high error rate or rather a low one. Therefore, we make another comparison with the results of Boletsis and Kongsvik [2]:

Table 4.8: Comparison of the total ER for different VR text input methods

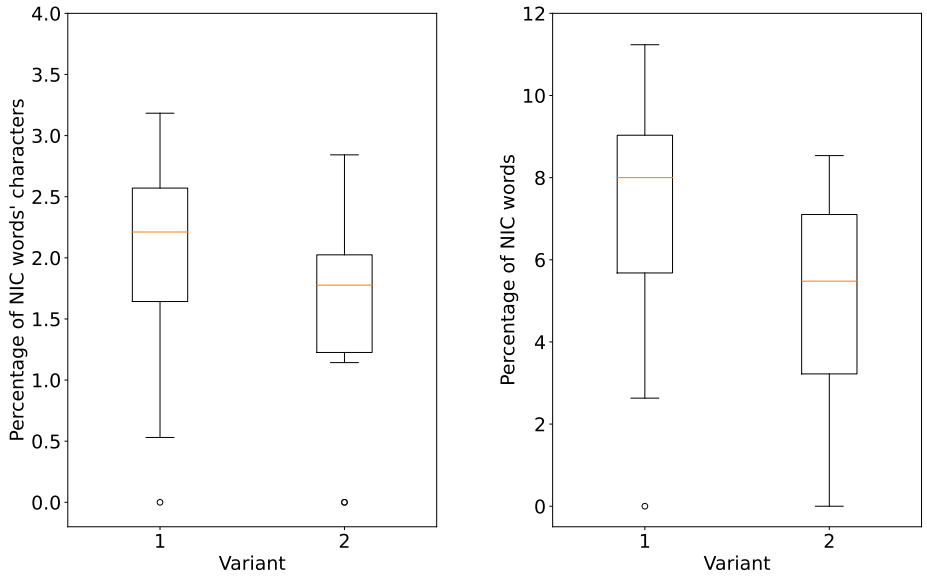
text input method	total ER
Split keyboard	8.11%
Head-directed input keyboard	10.15%
Word-gesture keyboard (ours)	10.89%
Raycasting keyboard	11.05%
Drum-like keyboard	12.11%

The total ER for our keyboard seems also to be ranked in the middle as the WPM value before. An explanation for this could be that we forced the participants to correct errors because they could only continue with the next phrase, if the last one was correct. Thus, a lot of participants saw an error only when they thought they had ended a phrase. Then, they had to delete half of the phrase to be able to correct the error. Boletsis and Kongsvik [2] on the other hand did allow transcribed texts that are not correct. If we had also done this, the IF value would be much lower. Obviously the INF value would increase but not as much as IF would decrease. While only some characters would be added from wrong words in the transcribed text to he INF value, the IF value would sometimes be only a fraction of itself. This is because a participant would not have deleted half of a phrase as many times as they did now because they could just continue with the next phrase. This means that our total ER value could be lowered if we had taken another evaluation approach. Nevertheless, the result is still not too bad.

4.4.3.4 Participant Conscientiousness

For the next calculation, we investigate the percentage of not immediately corrected words. Not immediately corrected (NIC) words are the ones, where the best match is

not the intended word but the word would be in the keyboard's word suggestions but the participant does not choose it. Firstly, we will do this with characters and then with the whole words. For the calculation with the characters of a NIC word we take the MSD between the wrong word and the intended one. Then we want to compare these two values with the same approach, but without considering the errors happened because of "the", "in" and "more". We decide to remove these three words because they could be avoided by a small change in the best match calculations, and because we are interested if there are any differences worth mentioning.



(a) Not immediately corrected words' characters (b) Not immediately corrected words

Figure 4.6: (a) shows the ratio of NIC words' characters to all inputted characters. (b) shows the ratio of NIC words to all inputted words. The left boxplots in (a) and (b) are with all NIC words, the right ones are without considering "the", "in" and "more".

In Fig. 4.6(a) we can see that the percentage of NIC words' characters lies between about 0-3.5%. We can observe that the percentage decreases a bit from the left to the right box. In Fig. 4.6(b) the percentage is between about 0-11.5%. It also drops if we do not consider the three mentioned words. Overall, the percentages are much higher when working with words instead of the characters. This is because we use MSD. If a word is wrong, most of the time, our keyboard will not write a totally wrong word, if the gesture is right to some extent. It will consist of similar letters as the correct word. Therefore, not the whole word's characters will be counted into the MSD.

To take another look at the participants' attention, we want to calculate the value of participant conscientiousness. As for the total ER, we also take the formula for the

participant conscientiousness value from the paper of Soukoreff and MacKenzie [7]:

$$Participant\ Conscientiousness = \frac{IF}{INF + IF} \tag{4.4}$$

Normally, they would take the values of IF and INF as mentioned in Section 4.4.3.3 (not as we took them but as Soukoreff and MacKenzie [7] defined them). We think for this calculation, we should not use characters but words. Thus, for INF we count all the words not immediately corrected by a participant and for IF the words that were corrected by using the keyboard's suggestions. We think that the calculation makes more sense this way when looking at our keyboard, which writes whole words and not single characters most of the time. It shows us, if the participants were attentive and used the keyboard's word suggestion when needed. A score of 100% would be perfect. This would mean that a participant has corrected every word with the word suggestions, if it was possible and needed. The average value we get here is 66.28%, which means about two third of the words that could be corrected with the suggestions, were corrected this way. The worst value is 33.33% and the best value a participant achieved is 100%. In our opinion, this score is quite good. A lot of the participants are not used to word-gesture-keyboards, word suggestions, do not work in VR or do not use VR that much. We think if they used the keyboard over a longer period of time, they would get used to the words that are sometimes harder for the keyboard to get as best match and would pay more attention to use the word suggestions then. It is interesting that if we ignore the words "the", "in" and "more", the average value decreases to 63.73%. This is an indication that the participants have already gotten used to correct these words by using the word suggestions.

4.4.3.5 Word Not Found Error Rate

Here, we do not want to see if a participant was paying attention or not but rather if our algorithm finds the right words most of the time. We count all the words that were neither the best match nor in the word suggestions, such that a participant had no chance to get this word right the first time. If this happened multiple times to a word, we count it in every time. We decide to name this error rate the word not found (WNF) error rate because our algorithm was not able to find the right word. We use following formula:

$$word\ not\ found\ error\ rate = \frac{\#\text{ words not found}}{\#\text{ words in phrases}} \times 100\% \tag{4.5}$$

Table 4.9: Word not found error rate for each participant and the overall average

participant	WNF error rate
1	0.00%
2	3.37%
3	13.43%
4	3.90%
5	5.00%
6	2.67%
7	6.10%
8	5.48%
9	1.30%
10	3.95%
11	2.44%
average	4.33%

In Table 4.9 we can see that the average WNF error rate is 4.33%, which means that about every 23rd word could not be written in the first attempt. While the average length of a word of the phrases we used is 4.16, the average length of a word the system has not found in the first try is 7.03. This seems to indicate that words with a length way bigger than the average one are more error-prone. We think this makes sense because the longer the word, most of the time, the larger the gesture. Deviations from the perfect graph are harder to prevent if one draws a large gesture instead of a small one and then the system might not find the right word.

4.4.4 Feedback

The full list including all verbatim feedback from every participant can be found in the appendix. Here, we just want to highlight the most frequently addressed points.
Some participants find the best matches and suggestions sometimes confusing. The most mentioned example is that “thee” is preferred over “the”. The visualization of spaces or the current position are another thing that is frequently addressed. Some participants find it unclear where the cursor is, hence if a space is missing or not. They suggest to implement some kind of visual indication that indicates where the cursor currently is. We can say that this problem existed in the evaluation environment but vitrivr-VR, the program that our keyboard is manly developed for, has these kinds of cursor visualizations. We also got a lot of praise and most of the time we saw a cheerful face when the VR headset got taken off. Some find it surprisingly intuitive and others have a good feeling about using it.

4.4.5 Improvements

Because of the feedback and some observations, we decide to do one more implementation. It is not a big change in the code, but can still be fairly useful. We have the problem, that for example, “thee” is prioritized before “the”, which does not make much sense, because “the” is used far more in plain language. Because the word list we are currently using is ordered by the frequency of the words, we let this influence the best match and word

suggestions. Now, if two or more words have the same highest score, the one that is highest in the word list will be taken, followed by the second highest and so on, until a word with a lower score comes next. This will prevent the problem with some words listed in Table 4.4.

5 Conclusion

In this last section, we will briefly talk about what we have done in this thesis, the results and what can be implemented in the future to improve the keyboard developed during this thesis.

5.1 Conclusion

We developed a word-gesture keyboard for VR. We used the algorithm of the SHARK^2 system [4] as a basis, which we implemented in a simplified version in Unity. Besides the basic function to write words with gestures, there are some other functionalities our word-gesture keyboard provides. A user can add new words, create new layouts, change the size of the keyboard and choose from up to four other words, if the best one is not the intended one. From the results, we think that the text input method we implemented might not be the fastest way to input words in VR. At least this can be said for beginners. When we tried to write some phrases after some training, we got a fairly high WPM score, which shows that one could possibly write quickly after some training. Further, it seems to have an average error rate compared to other text input methods. This also means that from this point of view, it works fine. All in all, we think it is a nice and useful text input method for the VR space.

5.2 Future Work

First of all, the word detecting algorithm of the keyboard can be extended with all the improvements used in the SHARK^2 algorithm that we did not implement. The most important of these improvements is the usage of language information. If this was implemented, the word suggestions could possibly improve a lot. The dynamic channel weighting by gesturing speed could also be useful if the keyboard was used a lot by some people. Because if they used it a lot, they might write more from memory recall than visually guided (Section 2.3.1.6), and it might make sense to weight one channel more than the other one.
In the future, one could also improve the available layouts. Right now, only the standard layout is modifiable, but no hexagonal layouts are implemented. So, a future work could

be to implement hexagonal layouts and a possibilty to switch between the standard layouts and them.
Another thing for more user-friendliness would be the use of error messages. For example, if a layout is added incorrectly to the file containing all layouts, the user could be informed about it in the VR space via some kind of text bubble.
Finally, another important implementation would be to be able to change the language. Right now, English is the only available one. There could be some new options button to let the user change the language.

Bibliography

[1] Ahmed Arif and Wolfgang Stuerzlinger. Analysis of text entry performance metrics. pages 100 – 105, 2009.

[2] Costas Boletsis and Stian Kongsvik. Controller-based text-input techniques for virtual reality: An empirical comparison. *Int. J. Virtual Real.*, 19:2–15, 2019.

[3] Sibo Chen, Junce Wang, Santiago Guerra, Neha Mittal, and Soravis Prakkamakul. Exploring word-gesture text entry techniques in virtual reality. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA '19, page 1–6, 2019.

[4] Per Ola Kristensson and Shumin Zhai. Shark2: a large vocabulary shorthand writing system for pen-based computers. In *UIST '04*, pages 43—52, 2004.

[5] I. Scott MacKenzie and R. William Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, pages 754—-755, 2003.

[6] I. Scott MacKenzie and Shawn X. Zhang. The design and evaluation of a high-performance soft keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 25—-31, 1999.

[7] R. William Soukoreff and I. Scott MacKenzie. Metrics for text entry research: An evaluation of msd and kspc, and a new unified error metric. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, page 113–120, 2003.

A
Appendix

Feedback

Thanks, it works & feels very good!
Personally, I'd like to see the automatically added space (e.g. visual indication in form of a color?)
A setting to disable/prioritise certain words (e.g. make "the" a higher priority than "thee"). When I want the word "the" the recommended word was "thee". Since "the" is more common, I would have preferred if it would suggest "the" by default instead of "thee".
Maybe two separate buttons for deleting a word and deleting only a character might be useful.
If the last typed character is a space, maybe have a placeholder symbol that visually shows that.
Some additional nice looking things in the 3D VR room! It looks very experimental now :-)
To question 1 in evaluation: if it supports the langauge I need (strongly agree, 5)
Hitbox could be larger (of the keyboard).
Writing single-letter words works differently (no added space) to >2-letter words.
The automatically created words when using slide-to-type are somtimes quite obscure, especially when compared to the other suggestions. E.g. thee instead of the.
Overall it was surprisingly intuitive to use.
I really like the visual feedback being provided, I could tell a lot of thought went into user friendliness.
Sometimes it was unclear where the current cursor is.
the → thee
Needs a better differentiation for very similar words or words with a double vowel.
long words are hard to write.

University of Basel

Faculty of Science

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis:

Name Assesor: _____

Name Student: _____

Matriculation No.: _____

With my signature I declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Place, Date: _____ Student: _____

Will this work be published?

☐ No

☐ Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: _____

Place, Date: _____ Student: _____

Place, Date: _____ Assessor: _____

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .