UCLouvain

epl

École polytechnique de Louvain

Composition of Strokes Gestures into System Instructions

Author: **Adrien Moinnet, Benoît Gosselin**
Supervisor: **Jean Vanderdonckt**
Readers: **Charles Pecheur, Arthur Sluÿters**
Academic year 2021–2022
Master [120] in Computer Science

Abstract

For a long time, the main technique for executing a command has been point and click. However, due to the increasing need to execute commands quickly, shortcuts are becoming more and more popular, as is the case with point and click. Many studies have improved the shortcut system by developing 2D shortcuts. Unfortunately, keyboard shortcuts and 2D shortcuts involve a one-to-one gesture-command mapping. This leads to many memorability problems which are mainly because it is necessary to assign a new gesture to each new command and also, after a certain number of shortcuts, some gestures become more and more abstract. To address this issue, we decided to explore the concept of combining 2D gestures when interacting in smart environments. To achieve this, we have created an application allowing to realize shortcuts using macro-commands or a combination of 2D gestures which are congruent and hierarchic. This application will aim to visualize interactions with connected objects in a smart environment named EMERITI. To better understand the EMERITI environment, we will detail its grammatical structure, its functioning, and its advantages.

Acknowledgments

First and foremost, we would like to express our deepest gratitude to our thesis supervisor, Prof. Jean Vanderdonckt, for his precious guidance. This thesis would not have been possible without his insightful advice and feedback.

Special thanks to Arthur Sluÿters for his much appreciated help and support in some crucial parts of our thesis.

We would also like to thank Nathan Magrofuoco for his much appreciated help at the beginning of our thesis.

Thanks to our thesis reader, Prof. Charles Pecheur.

Finally, we would like to express our deep and sincere gratitude to our friends, our family, and especially our life partner Noémi Galdos and Coline Metzler for putting up with our stress and for their support throughout this year. We could not have done it without them.

Contents

1 Introduction **4**
1.1 Context of The Problem . . . 4
1.2 Mission statement . . . 5
1.2.1 Statement . . . 5
1.2.2 Working Hypotheses . . . 5
1.3 Overview of the Thesis . . . 5

2 Related Work **7**
2.1 Using Strokes as Command Shortcuts: Cognitive Benefits and Toolkit Support . . . 7
2.1.1 Introduction . . . 7
2.1.2 Hypothesis . . . 8
2.1.3 Experimentation and Results . . . 8
2.1.4 Usefulness for our Work . . . 10
2.2 On Gesture Combination: An Exploration of a Solution to Augment Gesture Interaction . . . 11
2.2.1 Introduction . . . 11
2.2.2 Smart Environment . . . 12
2.2.3 Gestures Combination . . . 13
2.2.4 The Procedure of the Experimentation . . . 13
2.2.5 Result of the Experimentation and Conclusion . . . 13
2.2.6 Common Point with our Project . . . 14
2.3 Congruent Gestures . . . 14
2.3.1 Unistrokes Gesture . . . 14
2.3.2 Multistroke Gesture . . . 15
2.3.3 Problem . . . 15
2.3.4 Non-congruent and Non-hierarchical Gestures . . . 15
2.4 Recognizers . . . 17
2.4.1 Paradigms . . . 17
2.4.2 2D Recognizers . . . 17
2.4.3 Overview of the $P+ Algorithm . . . 20
2.4.4 Selecting a 3D Recognizer . . . 22
2.5 QuantumLeap . . . 23
2.5.1 Introduction . . . 23
2.5.2 Definition . . . 23
2.5.3 QuantumLeap vs OpenNI . . . 23
2.5.4 The Leap Motion Controller . . . 23
2.5.5 Pipeline Architecture . . . 24

2.5.6 UI . . . 25
2.5.7 Limitations . . . 26
2.6 Address and Command . . . 28
2.6.1 Definitions . . . 28
2.6.2 Advantages of Combination of Gestures . . . 29
2.6.3 Challenges . . . 29
2.6.4 Result of the Study . . . 30

3 Composition of Stroke Gestures into Instructions in EMERITI 32
3.1 General Format . . . 32
3.1.1 Definitions . . . 32
3.1.2 Instructions . . . 33
3.1.3 Elements . . . 34
3.1.4 Combination of Gestures . . . 35
3.1.5 Nassi-Schneiderman Diagram . . . 35
3.1.6 Explanation of 2D Stroke Gestures by Congruence and Hierarchy 36
3.1.7 Examples . . . 37
3.1.8 Parser Generator . . . 39
3.1.9 Process of the creation of the grammar . . . 42
3.1.10 Advantages Compared to A&C . . . 43
3.2 Exploratory Analysis . . . 44
3.2.1 Scenario . . . 44
3.2.2 Process Modeling . . . 45

4 Creation process and explanation of EMERITI 47
4.1 Process of Work . . . 47
4.1.1 Discovery of the Tools . . . 47
4.1.2 Creation of a Canvas . . . 48
4.1.3 Connection with QuantumLeap . . . 48
4.1.4 Realization of our Virtual Simulator . . . 50
4.1.5 Creation of the Table of Gestures . . . 51
4.1.6 Add a Timer for Combination of Gestures . . . 51
4.1.7 Steps to Realize All Buttons . . . 52
4.1.8 Improvement of the Visual . . . 53
4.1.9 Start the Implementation of Common Instructions . . . 55
4.1.10 Implementation of Macro Instructions . . . 55
4.2 Final Version . . . 56
4.2.1 Sequence Diagram . . . 56
4.2.2 Architecture . . . 59
4.2.3 How to run EMERITI . . . 60
4.2.4 Code Structure . . . 61
4.2.5 User Manual . . . 64

5 Final Conclusion 70
5.1 Summary of the Contributions . . . 70
5.2 Benefits and Limitations . . . 71
5.3 Future Work . . . 72

A Analysis Work **77**
A.1 Task Modeling 82
A.1.1 Task with Low Complexity: Task 1 – Launch Application 82
A.1.2 Task with Medium Complexity: Task 17 - Create Gesture 83
A.1.3 Task with High Complexity: Task 11 - Connect Profile 83
A.2 Organization Modeling 84
A.3 Workflow Modeling 86
A.4 Workflow Analysis 90

B ANTLR Code **91**
B.1 ANTLR Code 91

Chapter 1

Introduction

1.1 Context of The Problem

Nowadays, there are two main ways to invoke a command in a graphical user interface: (1) point and click the label in a menu or its icon; (2) activate a keyboard shortcut [1]. The technique mostly used nowadays is the point and click. Although this technique is easy to use and intuitive, it makes the realization of tasks much slower than the use of shortcuts. Moreover, the use of devices such as a computer has become something indispensable in everyday life. As a result, the use of keyboard shortcuts has become a way to make our various tasks faster.

Unfortunately, the classic use of keyboard shortcuts also reaches its limits in many situations. Indeed, studies show that users have some difficulty in making the transition from menu selection to keyboard shortcuts [2]. Moreover, for each command we want to execute, we have to execute a specific shortcut. From this fact we can easily deduce that it is difficult to remember all the possible shortcuts in case there are many commands with keyboard shortcuts. The reason is that once you have a certain number of commands, it is difficult to remember them all and the shortcuts become more and more abstract [1, 3]. Indeed, having a unique shortcut for each command leads designers to have to invent shortcuts that become less and less intuitive.

To improve the various shortcomings of keyboard shortcuts, many studies have investigated the use of stroke gestures as shortcuts to menu selection. Indeed, stroke shortcuts have been shown to improve recall by making shortcuts as analogous or mnemonic to the command name or meaning as possible [1]. However, despite these advantages, there was still a problem concerning the number of shortcuts (keyboard or stroke) to be memorized. A large number of shortcuts to be created made some shortcuts abstract and therefore difficult to remember [3].

Following this problem, we decided to explore the concept of combining 2D gestures allowing us to create hierarchical gestures. The advantage of hierarchical gestures is that users can create sets of gestures with fully optimized combinations to reuse the gestures. The combination of gestures, therefore, improves recall capacity compared to single gestures. Moreover, gesture combinations are often preferred to single gestures when they have an abstract [4] gesture-command correspondence [3].

To implement our research work, we have created an application that uses a smart environment called EMERITI (EnvironMent gEstuRe InTo Instruction). The objective behind this application is to be able to demonstrate the advantage of the combination of 2D gestures with simple gestures, whether it is in terms of memorization capacity, ease of use, or speed of execution of the various commands.

1.2 Mission statement

1.2.1 Statement

The central purpose of our dissertation is to create an application allowing to realize shortcuts using macro instructions or a combination of gestures that are congruent and hierarchic to interact with connected devices. This application allows us to explore the concept of combining 2D gestures when interacting in smart environments.

1.2.2 Working Hypotheses

During this thesis, we assume the following working hypothesis:

1. Some subjects being not of our competence, we made the hypothesis that we should not solve the problems concerning recognition. Indeed, our work does not have to deal with the accuracy of the recognizer since it is not in the scope of our thesis.
   It can happen that the gesture realized by the user is not correctly recognized. This could be a problem and that is why we make the assumption that the recognition of gestures is perfect.

2. It is not in the objective of our thesis to have an application that aims at the perfection of the usability criterion.

1.3 Overview of the Thesis

To achieve the central objective assigned to this thesis, the manuscript is structured as follows.

In chapter 2, we will discuss all the related work that allowed us to do our work by explaining the use of stroke gestures as shortcuts for menu selection, exploring the design of 3D gestures combination in a smart environment, explaining what congruent gestures are, the choice of our recognizer, explaining the operation and use of QuantumLeap and finally, discuss about 'Address and command'.

In chapter 3, we will explain our work. First, we will explain the general format of our grammar to handle hierarchical and congruent gestures. After that, we performed an exploratory analysis in order to explain, through a scenario, the objective that we would like to achieve with EMERITI.

In chapter 4, we will talk about the process to arrive at the final result of EMERITI (our smart environment developed in our application) while talking about the problems encountered and their resolutions. Finally, we will present the final version of our work with the help of sequence diagrams, architecture diagrams, and a user manual.

Finally, chapter 5 overviews the contributions of this thesis and suggests some future avenues to this work, given its existing shortcomings.

Chapter 2

Related Work

After talking about "Using Strokes as Command Shortcuts" in section 2.1, we will present the article on gesture combination 2.2.
In section 2.3, we will talk about congruent gestures followed by an explanation of the existing recognizer 2.4.
After that, we will talk about the framework we used to create EMERITI which is QuantumLeap 2.5.
In the end, we will talk about Address and command which is a similar work compared to EMERITI.

2.1 Using Strokes as Command Shortcuts: Cognitive Benefits and Toolkit Support

2.1.1 Introduction

This paper [1] investigates using stroke gestures as shortcuts to menu selection. The reason is that traditional keyboard shortcuts quickly reach their limits in many situations. First, studies show that users have more difficulty in making the transition from menu selection to keyboard shortcuts [5]. Second, the use of shortcuts from traditional keyboards is very restrictive for many devices such as smartphones and pen-based Tablet PCs. These different devices require manipulation of the screen by the users.

To use shortcuts based on stroke gestures, they experimentally measured the performance and ease of learning of stroke shortcuts in comparison to keyboard shortcuts when there is no mnemonic link between the shortcut and the command (see related work on icons [6]). While both types of shortcuts had the same level of performance with enough practice, stroke shortcuts had substantial cognitive advantages in learning and recalling. With the same amount of practice, users could successfully recall more shortcuts and make fewer errors with stroke shortcuts than with keyboard shortcuts. The second half of the paper focuses on UI development support and articulates guidelines for toolkits to implement stroke shortcuts in a wide range of software applications.

Indeed, they formulate and investigate the following hypothesis: stroke shortcuts may have a cognitive advantage in that they are easier to memorize than keyboard shortcuts.

2.1.2 Hypothesis

As mentioned in the introduction, they hypothesize that an arbitrary association between a command and a shortcut is more learnable when this shortcut is a stroke rather than a combination of keys. This hypothesis is based on two arguments, one in favor of strokes and one against key combinations:

- Based on several articles, it has been postulated that strokes (also known as gestures or marks) have multiple advantages including being iconic [7]. Nowadays, many contemporary software applications use icons. As a result, it can be deduced that many users are able to establish arbitrary correspondences between commands and icons. For example, a compass icon is used for launching the Safari browser, a curved arrow is for reversing the last action (undo), and a floppy disk, now an obsolete concept is used as an icon for saving the current file on the hard drive.

  From a theoretical perspective, research supports that the spatial and iconic information of a stroke allows for the better association of the stroke and corresponding commands thus promoting memory. Indeed, human memory suggests that deeper or more levels of encoding and processing promote memory [8].

- Letters are special symbols that are strongly linked to the words they are related to. Therefore, it is very difficult to associate a letter with a command that does not begin with that same letter (such as Ctrl+V to paste).

2.1.3 Experimentation and Results

In order to highlight the advantage of shortcuts using strokes than keyboard shortcuts or via the selection menu, experimentation has been conducted. The experimentation is divided into three phases [9]. First, a command stimulus appears as an icon on the screen. The participant must click on the icon to make it semi-transparent. Once this step is done, the participant has the option to perform the command using either the menu or a stroke shortcut.

To establish analyses, the authors of the article used three measures:

- Time: the total time interval (in ms) between the presentation of the command icon and the correct execution of the command. The total time includes the recall time and the execution.

- Errors: the number of times the participant makes a mistake before executing the correct shortcut.

- Recall: a binary value that is equal to 1 when the user is able to execute the correct gesture without error and without the help of the recall menu, 0 otherwise.

Based on the results of the experiment, it was found that participants switched from menu selection to stroke shortcuts more often or earlier than keyboard shortcuts [2]. In order to compare the learning speed of each type of shortcut (stroke or keyboard), graphs were made measuring the average time and recall performance as a function of the number of times an item was tested since the beginning of the experiment.

The results obtained confirmed the hypotheses (Figure 2.1), namely:

- The time required to perform the command decreased more rapidly with stroke shortcuts than with keyboard shortcuts.
- The precision following a recall increased more rapidly with the stroke shortcuts than with the keyboard shortcuts.
- The number of errors following the execution of a command is much lower with stroke shortcuts than with keyboard shortcuts.

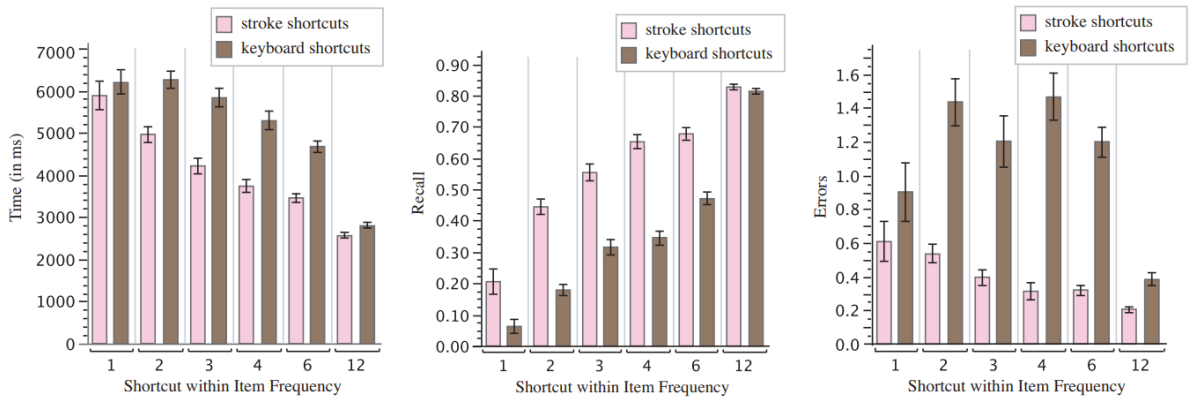

Figure 2.1: Time, Recall, and Error by Shortcut x Frequency.
Source: Figure taken from [1].

It should be noted, however, that the difference in performance between the two types of shortcuts is essentially cognitive (learning and recall of shortcuts). Consequently, when a user performs a large number of repetitions (around 25), the difference in time and recall between the two types of shortcuts becomes indistinguishable (Figure 2.2).

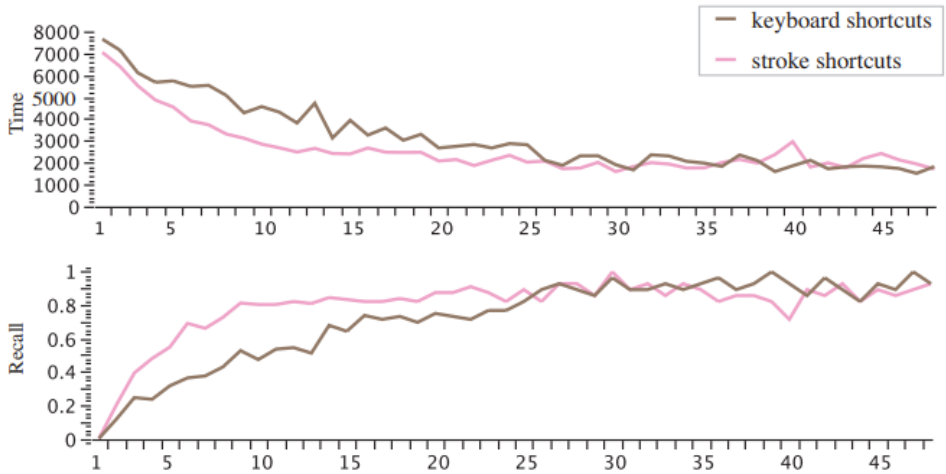

Figure 2.2: Time and Recall performance according to the number of times a command has been seen.
Source: Figure taken from [1].

On the second day of testing, the main objective was to evaluate the memory retention of the learned shortcuts by the users and thus find out which type of shortcuts they

preferred. Thus, based on the data collected, they were able to show the percentage of use of each individual participant for each technique (Keyboard, Line, and Menu). On average, stroke shortcuts are used much more than keyboard and menu shortcuts (77.7% stroke, 20.3% keyboard, and 2% menu) (Figure 2.3).

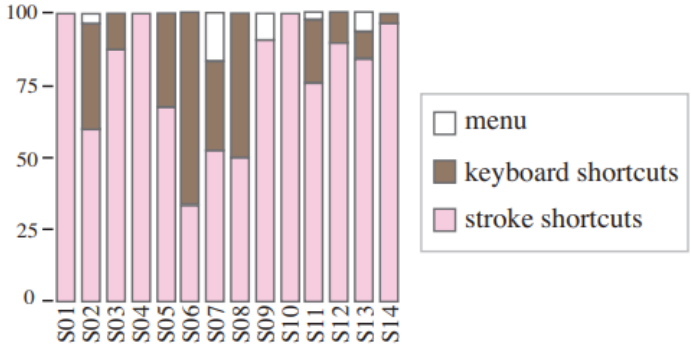

Figure 2.3: Percentage of use of each technique in retest on the second day (by the participant)
Source: Figure taken from [1]

In summary, although the aim of the experiment is not to compete with or replace menu selection or keyboard shortcuts, the experiment clearly shows that stroke shortcuts can be as effective or more advantageous than keyboard shortcuts. After sufficient practice, the total command execution times were indistinguishable between the different shortcuts. However, with the same amount of practice, participants tended to remember more gestures (about 3 times more), to remember gestures more easily and quickly, and finally, to make fewer errors with stroke shortcuts than with other shortcuts.

2.1.4 Usefulness for our Work

Based on this article, we can establish a solid foundation for the usefulness of our project. Indeed, as we discussed earlier, our work aims to facilitate the use, memorization, and creation of shortcuts. Therefore, this paper supports our idea that memorizing and executing stroke shortcuts is more efficient than keyboard shortcuts.

However, despite the fact that the experiment allows users to replace keyboard shortcuts with stroke shortcuts, there is still an important problem with memorizing each command. Indeed, for each command, a unique gesture is associated with it, thus causing a large number of gestures to be memorized. Although this is a big step in the use of shortcuts, our work aims to facilitate this principle by allowing users to execute a command by performing a set of gestures (section 2.3).

2.2 On Gesture Combination: An Exploration of a Solution to Augment Gesture Interaction

2.2.1 Introduction

The current gesture interaction paradigm mainly involves a one-to-one gesture-command mapping. This means that for each command we wish to record, it is necessary to associate it with a new gesture. Just like keyboard shortcuts, this leads to memory problems concerning (1) the mapping - as each new command requires a new gesture, and (2) the specificities of the gestures - which can be complex to take advantage of the recognition of several gestures.

In order to address this issue, this paper [3] explores the design of 3D gesture combinations in a smart environment such as EMERITI. Based on three user studies, several results were found. The first study reveals that end-users can create gesture sets with combinations fully optimized to reuse gestures. The second study demonstrates that gesture combinations can improve recall compared to single gestures. The third study shows a preference for combined gestures when single gestures have an abstract gesture-command mapping.

Following the use of mid-air hand gestures being more and more interesting, a huge gain of interest was born towards the interaction with augmented physical appliances [10, 11, 12]. Indeed, the ability to control objects at a distance has led researchers to propose solutions to improve gesture recognition [13, 14], gesture registration [15], gesture set designs [16, 17], gesture guidance [18, 19] or toward easing the deployment [20] and the adoption [21] of gesture interaction to name a few.

Previous studies involved mainly actions performed by one hand, or sometimes coordination between the two hands to perform a single gesture-command mapping.

This paper proposes to expand the investigation of 3D mid-air gesture combinations [22]. The idea would be to execute a first gesture - a signifier gesture - representing for example a context (e.g., 'Light') while a second gesture - an operator gesture - could trigger a command (e.g., 'Switch on') (Figure 2.4).

The main benefit of this practice was in the re-usability of gesture definitions. Thus, a user who would select an action such as 'increase/decrease', could apply it to the brightness of the lamps, the volume of the sound, the temperature, etc. In this way, the gesture combination can easily extend a set of commands without increasing the number of gestures to memorize.

Another goal of the study is to demonstrate the interest in gesture combination towards single gestures and single abstract gestures (Figure 2.5). The interest behind including abstract gestures among single gestures is to observe the behavior of users with respect to these gestures. In this way, if we notice that abstract gestures are more difficult to memorize and execute, we could deduce that combined gestures would be a possible solution to this problem. Indeed, since we would limit the number of gestures to remember, these gestures would have meaning with respect to the actions to which they are linked and would limit the errors of execution and memorization.

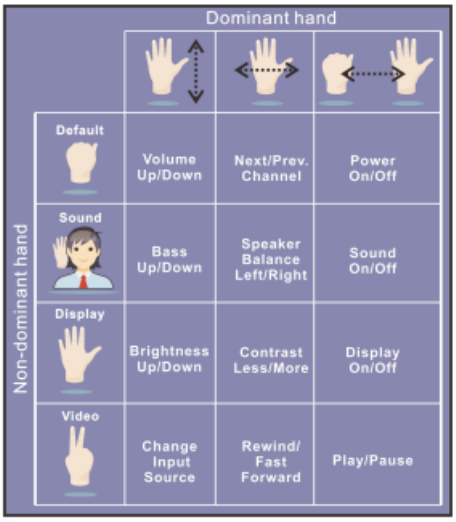

Figure 2.4: Gesture set while combining gestures. Gestures are depicted with rows showing the signifiers and columns showing the operators. **Source:** Figure taken from [3]

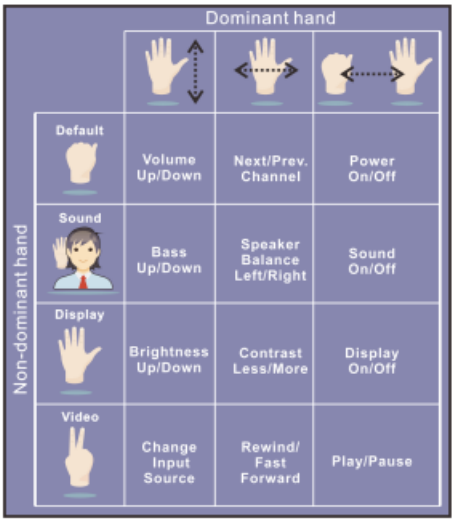

Figure 2.5: Gestures set with single gestures. We distinguish between abstract (light orange background) and non-abstract (dark purple background) gestures. Note that the two-handed gesture used for "Display On/Off" is a single gesture-command mapping. **Source:** Figure taken from [3]

2.2.2 Smart Environment

Gestural interaction with smart environments [10, 11, 23] mainly requires dynamic semaphoric gestures [24, 25]. Semaphore gestures are used to define a specific vocabulary to trigger discrete commands. For example, in order to trigger the "Save" command, the user can draw a circle in the air [11].

Among the different devices that can be part of a smart environment, the television has been used in several studies [26, 11, 27]. Indeed, gesture interaction has the potential to be a good complement or to completely replace the classical remote control. Moreover, the television is a device allowing several functionalities (brightness, volume, channel, etc.) and several commands (up, next, play, etc.). All these characteristics make the television a good candidate to explore interaction paradigms such as speech recognition [28] or freehand gesture interaction [29, 16, 30, 23, 27].

Most of the previous work reports the use of one-to-one gesture/command mappings. This means that to execute a command, a new unique gesture must be invented by the designers. For example, even though 2 commands may intuitively share similar gesture preferences such as raising the hand for "Increase volume" and "Increase brightness", designers must create different gestures for each command. And in order to avoid the user having to remember all the gestures assigned to each command, this study aims to explore the possibilities offered by the combination of gestures.

2.2.3 Gestures Combination

The combination of gestures allows the different available commands to be divided into hierarchical sets [31, 32, 33, 22].

For example, instead of having to remember 36 gestures where each one performs a command, users could perform a gesture to select one of 6 commands in the first level and the same in the second level. This would allow remembering 12 gestures instead of 36. Hierarchies also allow gestures to be reused since a context is specified in the definition of the hierarchy. For example, with the hierarchical marking menu, one could very well perform a left stroke and it would mean different things depending on the levels previously selected [33]. Moreover, hierarchical structures can improve learning ability [22].

Indeed, in the case of selecting a device, a common gesture to turn it on could be executed and thus, limit the number of gestures to remember and allow to remember it more easily since it would be more intuitive.

However, few works have been interested in the subject of gesture combination, and have mainly focused on sequential gesture combinations, in 2D [33, 22, 34, 35] or in 3D [31, 32]. Thus, the objective of this study is to extend the concept of sequential gestures (i.e., "the chaining of primitives" [22]) to the broader notion of gesture combination.

2.2.4 The Procedure of the Experimentation

In order to demonstrate the interest in the combination of gestures, experimentation in several stages was carried out. The experimentation takes place in several steps. First, the participant sits in front of a Smart TV and is presented with the two gesture sets (single gestures and gesture combinations) and their design rationales. Once the participants have taken the time to memorize the different gesture sets, the experimenter reads a sequence of scenarios (e.g., "Your friend is calling you, please input the command 'Sound off'.") and the participants have to execute the command related to the scenario. When the participant performs a correct gesture, the corresponding command is displayed on the TV screen. In case of an error, a sound signal was emitted. This whole process is carried out on the two gesture sets.

After getting familiar with the two gesture sets, the experimenter asks the participants to perform the same tasks as before but this time the participants are free to choose either single gestures or a combination of gestures.

2.2.5 Result of the Experimentation and Conclusion

The results show that based on the three types of gestures (combination, single abstract, and single non-abstract), there was a significant difference in recall rate between the gesture combinations and the single abstract gestures, but not between the other pairs. Therefore, combining gestures is a viable solution to prevent designing too many single gestures, potentially leading to abstract gestures. Indeed, the day after the experiment,

the average recall rate was 98% for the single non-abstract gestures and 69% for the abstract ones.

As expected, abstract single gestures have a negative impact on memorability. Participants struggled to remember the mapping of abstract gestures. Moreover, this experiment involves only 5 abstract gestures out of 12. We could therefore expect that users would have even more difficulty if they were to add new gestures. As a result, the study also showed that users' preferences go toward gesture combinations when the single gesture counterparts are abstract.

Finally, the results show that end-users mostly focus on temporal aspects of the combination and manage to optimize the reuse of gestures.

2.2.6 Common Point with our Project

As you may have noticed, the problem at the origin of this study as well as the solutions developed are partially identical to ours. Indeed, this problem is that as the number of objects and their interactive functionalities increase, it becomes difficult to define new gestures without including memorability issues.

To address this issue, they will develop the concept of gesture combination in 3D and, in order to use it to control augmented home appliances in smart environments. In our case, we also have a smart environment (EMERITI) where we can control connected devices, but we have opted for the execution of gesture combinations in 2D.

Despite the fact that we execute gesture combinations, the way of combining them is totally different. In the case of this study, the principle is to execute the command by performing gestures with both hands. The non-dominant hand for the "signifier" gestures and the dominant hand for the "operator" gestures. In our case, we propose to the users execute their 2D gestures one after the other.

2.3 Congruent Gestures

First of all, you need to know that there are two basic types of gestures to understand what congruent gestures are.

2.3.1 Unistrokes Gesture

The first is a unistroke gesture, i.e. a set of x and y points created in a single stroke with a stylus, a finger, or a computer mouse on a surface. The sample points draw the shape of the gesture which is also called a stroke. This can represent characters, symbols, or commands. You can see some examples in figure 2.6.

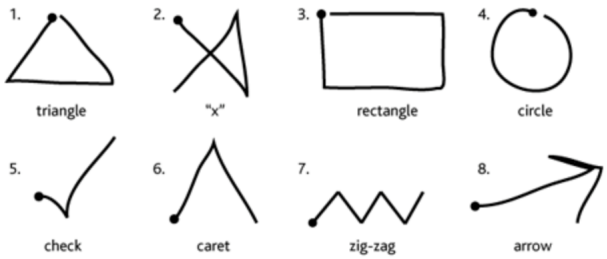

Figure 2.6: Some examples of unistrokes gesture.
Source: Figure taken from [36].

2.3.2 Multistroke Gesture

The second is the multistroke gesture. This gesture is composed of multiple unistrokes which means that we have to realize more than one stroke. You can see some examples of multistrokes gestures in figure 2.7.

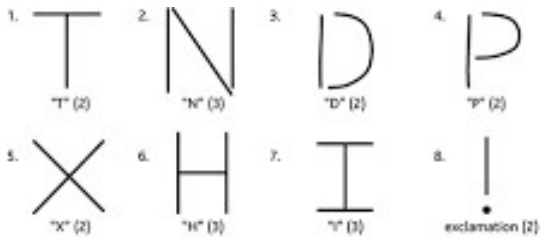

Figure 2.7: Some examples of multistrokes gesture.
Figure taken from [37].

2.3.3 Problem

Once we understand what are strokes gestures [38], we need to understand the problem that can arise if we use only one gesture to carry out a command in a menu or in an application. Indeed, assigning a 2D gesture to commands is an attractive interaction technique. However, when the set of commands or menu items subject to this assignment increases in size, these advantages tend to be reduced.

Indeed, it was shown in the article [39] that during the gesture elicitation phase, participants run out of imagination and creativity as the number of gestures produced increases. After a while, they can no longer find an appropriate gesture to correctly establish an assignment. They fall back on previously known gestures or create unnatural or unnecessarily complex gestures just to complete the process.

This is why we will distinguish these gestures as non-congruent and non-hierarchical.

2.3.4 Non-congruent and Non-hierarchical Gestures

Non-congruent means that related commands do not necessarily produce related gestures. For example, when 2 commands are opposite or symmetrical such as "Turn on

the TV" and "Turn off the TV", participants tend to propose unrelated gestures such as "Draw a circle" and "Draw a cross". If it was congruent, the 2 gestures would be a "Draw a circle in the clockwise direction" and "Draw a circle counter-clockwise direction"

Non-hierarchical means assigning a gesture to a sentence instead of assigning every single action in the sentence to gestures. Because a single gesture will be assigned for each individual command, resulting in a very large gesture vocabulary that no longer offers the desired benefits. As you can see in figure 3.9, an example of a non-hierarchical gesture is that we just have a "Circle" to represent "Turn TV on". Whereas if it had been hierarchical, we would have had 2 or 3 gestures. "Circle" represents "Turn", "Rectangle" which is for TV, and "V" for "on".

Finally, using two elicitation studies, the paper found a highly significant difference between non-congruent and non-hierarchical gestures proposed for the first set of referents and congruent and hierarchical gestures proposed for the second set of referents. In fact, the average agreement rate increased by 30%, suggesting that participants found agreement more easily and frequently when the gestures were congruent and hierarchical than with non-congruent and non-hierarchical gestures.

Therefore, in our application, we have used the results of this article [39] to implement a way to achieve congruent and hierarchical gestures. More details are given in Chapter 3.

2.4 Recognizers

One of the most important steps in projects based on gesture recognition is the development or use of a recognizer. Indeed, to be able to recognize the gestures that the user makes, it is important to choose a good recognizer. There are many different types of recognizers, each with its own specificity. There are many 2D recognizers such as R, $1, $N, $P, $P+, $Q, μV, etc. But there are also many 3D recognizers such as $P^3, $P^3+, $Q^3, Rubine-Sheng, Jackknife, etc.

In order to determine the most efficient and relevant recognizer for our use, it is essential to define the preferred fields of application of these different recognizers. Indeed, some tools are more relevant for 2D recognition than 3D and vice versa, but above all, they have certain criteria that define their mode of use and the gestures they are likely to recognize best.

2.4.1 Paradigms

To characterize flat stroke gestures, it is essential to understand the Rotation-Scaling-Translation (RST) paradigm [40]. From two (or more) reference points captured, meaningful gesture parameters, also called features, are computed to establish rotation (relying on angular data), scaling (based on length ratio), and translation (linked to the barycenter of the translation). These three fundamental operations can be effectively performed on any stroke gesture. To obtain a recognizer that is as polyvalent and insensitive as possible to gesture variations, it is important that it be invariant to these operations (rotation, scaling, and translation). Finally, there is also the last invariance which is the articulation-invariance. The articulation-invariance for short, accommodates gesture variations in terms of how strokes are produced in time, how many (unistroke vs. multistroke), and in which direction [41, 42].

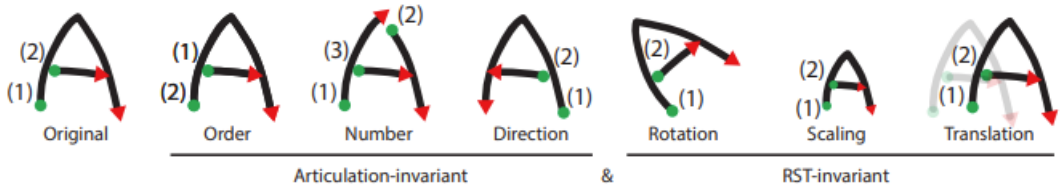

Figure 2.8: Stroke gestures can be articulated in multiple ways owed to the order, number, and direction in which the strokes can be drawn (Articulation-invariance). Stroke gestures can be performed at different rotations, scales, and/or translations (RST-invariance); hence the need for an ARST-invariant multistroke recognizer. **Source:** Taken from [42].

2.4.2 2D Recognizers

To determine the most efficient 2D recognizer for our project, we based ourselves on several studies comparing numerous recognizers. Among the large families of recognizers, we have the $-family.

$1 and $N Recognizers

This family has many efficient recognizers such as $1 [43] and $N [44] but each one has its limitation. In the case of the $1 algorithm, it satisfies RST-invariance for unistroke gestures. As for the $N algorithm, it satisfies articulation-invariance.
In this way, the $N algorithm can accurately classify multistrokes gestures, the order, the number, and direction in which the strokes are produced can be ignored [43]. This improvement thus allows multistrokes to be treated as unistrokes by connecting individual strokes "in the air".

However, as the order and direction of strokes may differ between users drawing the same symbol, $N must generate all possible permutations of a multistroke gesture. This problem causes an explosion in both memory and execution time. As a result, one can end up with a symbol comprising 9 strokes, which means that the system must represent $9!x2^9 = 185,794,560$ permutations [45]. This extremely high number is a challenge for modern desktop computers, let alone mobile platforms. Although $N uses runtime optimizations that reduce the number of comparisons with stored permutations, the cost of storing these permutations is currently its main limiting factor.

$P, $P+ and $Q recognizers

Following extensive research, new recognizers have been developed to overcome the problems encountered. Among the new recognizers, we find $P [46], $P+ [47] and $Q [48]. These algorithms implement cloud matching to fasten the classification of multi-strokes, they consider gestures as unordered sets of points. These quick, highly accurate recognizers have dropped support for rotation invariance as it requires comparing two gestures at different angular adjustments, which is a resource- and time-consuming task [42].

These recognizers, therefore, extend the $-family with fast, accurate, and low-cost recognizers for rapid interface prototyping. Indeed, $-family has initiated several applications and recognizers [49, 50, 51, 52, 53, 54, 55]. The performance of these new algorithms is similar to that of the $1 recognizer for single strokes and better than that of the $N recognizer for multiple strokes, making them much better successors [45].

Strokes	Invariant to...	Modalities	Recognizers
Uni-	Sampling, Translation	Mouse	GRANDMA
	Sampling, Scaling, Translation	Pen	xstroke,SHARK$^{(2)}$, SED-NE, Gestimator, Jackknife
		Pen, Finger	Penny Pincher
	Sampling, Scaling, Translation, Rotation	Pen	$1, Protractor, !FTL
Multi-	Sampling, Scaling, Translation, Articulation	Pen	$P+, G-Gene
		Pen, Finger	$P, $Q
	Sampling, Scaling, Translation, Rotation, Articulation	Pen	$N

Figure 2.9: Decision table for UI prototypers, designers, and researchers. The recognizers are highlighted in red when they require training with user-dependent samples, in green when they proved accurate when trained with user-independent samples, and in blue when the designer must input the formal definition of all gesture classes instead of providing samples.
Source: Taken from [56]

μV recognizer

Among the latest recognizers developed, there is one that we also found relevant and extremely effective, namely μV. μV is a multistroke gesture recognizer that addresses the properties of articulation, rotation, scaling, and translation invariance by combining $P+'s cloud-matching for articulation invariance with !FTL's local shape distance for RST-invariance. The advantage of the recognizer μV is that it is significantly more accurate than its predecessors when rotation invariance is required and not significantly inferior when it is not. μV is also significantly faster than others with many samples and not significantly slower with few samples [42].

Due to its very interesting qualities (speed, reliability, flexibility, accuracy), we consider that it would have been a very good recognizer for our project. Unfortunately, since we learned about this recognizer late, we did not have the time to test it in our project, which would have allowed for greater accuracy when using our application on portable hardware such as smartphones or tablets. The reason for this is that on handheld devices, users may be prone to drawing gestures with different rotations. Therefore, we focused on the recognizer that we considered to be the most relevant based on all the information we acquired, namely $P+.

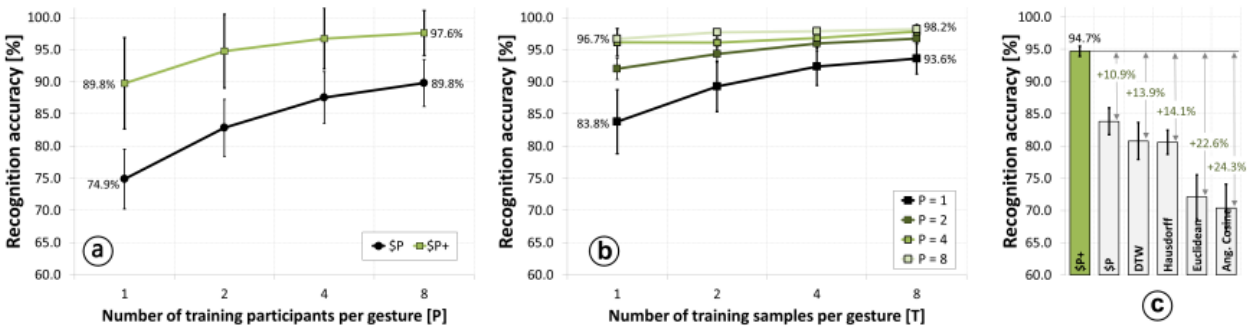

Figure 2.10: Recognition accuracy rates (user-independent) for gestures articulated by people with low vision: (a) direct comparison between $P+ and $P; (b) combined effect of the number of training participants P and the number of training samples per gesture type T on the recognition accuracy of $P+; (c) gain in recognition accuracy enabled by $P+ over the other gesture dissimilarity measures.
Source: Taken from [42].

2.4.3 Overview of the $P+ Algorithm

Since $P+ is the recognizer that we consider to be the most relevant for our project, an explanation of its operation is considered appropriate.

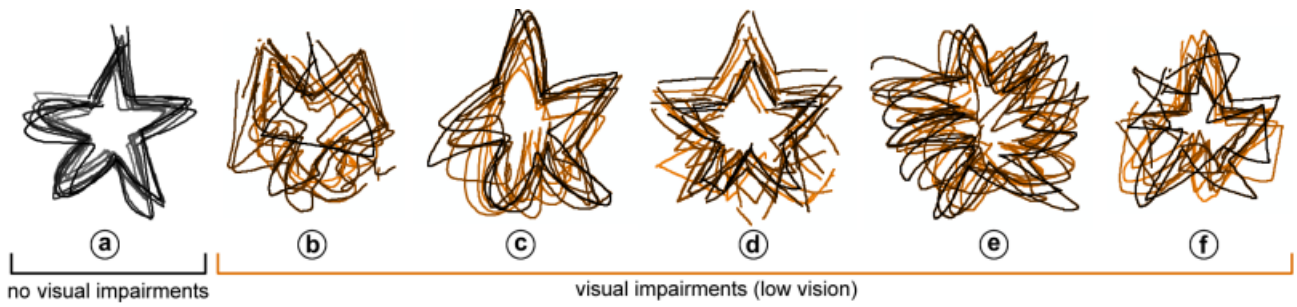

Figure 2.11: Gesture articulations produced by people with low vision present more variation than gestures produced by people without visual impairments, which negatively affects recognizers' accuracy rates.
Source: Taken from [57]

The $P+ Point-Cloud Recognizer is a 2-D gesture recognizer designed for rapid prototyping of gesture-based user interfaces, especially for people with low vision. Indeed, the recognition accuracy of gesture recognizers based on the Nearest-Neighbor approach for gestures produced by people with low vision, we find that there is an error rate of 22.5%. By carefully analyzing the gesture articulations, several observations led to significant algorithmic improvements to the basic design of the $P gesture recognizer. As a result of these modifications, the new recognizer iteration, $P+, increased accuracy for gestures produced by people with low vision up to 98.2% (error rate 1.8%) [57].

$P+ improves the accuracy of the $P Point-Cloud Recognizer and was, as explained previously, developed by carefully studying the stroke-gesture performance of people

with low vision, which generated insights for how to improve $P for all users.

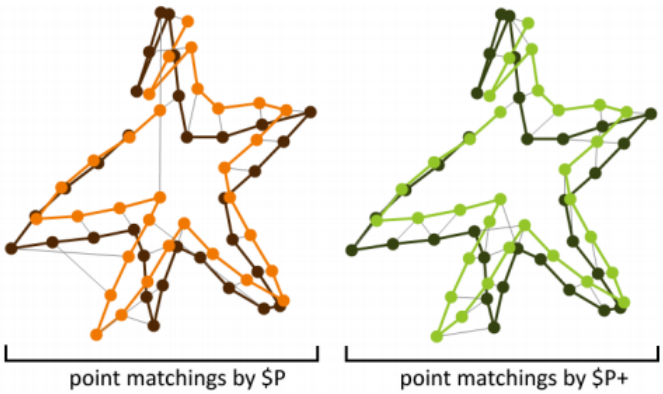

Figure 2.12: Point matchings produced by $P (left) and $P+ (right) for two "star" gestures produced by a participant with low vision. Note how the one-to-many matchings of $P+ help reduce undesirable point matchings of $P (see the long connecting lines in the figure on the left).
Source: Taken from [57]

This recognizer allows the use of multistrokes and preserves the properties of $P, such as scale, translation, and joint invariance. Moreover, the matching of the $P+ clouds is faster than its predecessor, which allows to obtain a time complexity in $O(N^2)$ instead of $O(N^{2,5})$ with N resampling points [56].

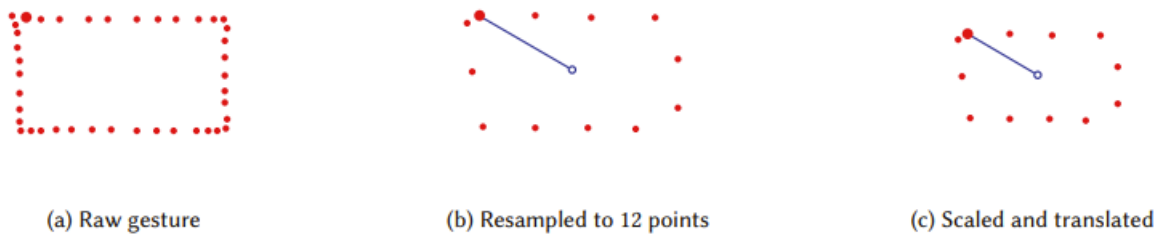

Figure 2.13: Three-step preprocessing applied by the $P+ recognizer on a rectangle.
Source: Taken from [56]

To determine the template that is closest to the gesture being performed, it is necessary to compute its between-points Euclidean distance. To do this, $P+ pre-processes the templates and any candidate to "normalize" them in terms of sampling, scaling, and translation:

1. The gestures are resampled to a fixed number of equidistant points to avoid any variation in sampling due to the hardware and software.
2. The resampled points are non-uniformly scaled to remain insensitive to scaling.
3. Finally, points are translated so that their centroid is at the origin (0, 0) to remain invariant to translation.

$P+ computes the Euclidean distance of a candidate point C_i and a template point T_j along with their difference in turning angles ($C_{i.\theta} - T_{j.\theta}$).

$$||C_i - T_j|| = \sqrt{(C_{i.x} - T_{j.x})^2 + (C_{i.y} - T_{j.y})^2 + (C_{i.\theta} - T_{j.\theta})^2}$$

The sum of these distances determines the dissimilarity score from the candidate cloud to the template cloud. This calculation is performed in both directions: from the candidate cloud to the template cloud, and from the template cloud to the candidate cloud. The minimal score of both directions is considered as the similarity score of the two gestures [42].

Finally, after testing the new $P+ algorithm with a new dataset acquired from visually impaired subjects and subjects without visual impairments, the results show that $P+ has a 94.7% recognition rate for visually impaired users in a user-independent scenario. Even for non-visually impaired users, $P+ proved to be more accurate than $P (98.3% vs. 95.6%). In terms of execution time, $P+ was 3 times faster than the original $P in the same scenario [56].

2.4.4 Selecting a 3D Recognizer

Our initial project is mainly based on 2D gestures. However, we also created this project to be able to work with 3D gestures. Therefore, it is also important for us to determine the most efficient 3D recognizer.

After some research, we were able to find several sources of information to answer our question. Indeed, to evaluate the most relevant recognizer for 3D, we based ourselves on an article whose goal is to recognize 3D gestures using 2D recognizers. Indeed, the trajectories, with their coordinates (x, y, z), can be interpreted as three 2D stroke gestures projected onto three planes, i.e. XY, YZ, and ZX, which makes them eligible for 2D stroke gesture recognition software.
To investigate whether these 3D trajectories can be effectively and efficiently recognized by this recognition software, four 2D recognizers, namely $P, $P+, $Q, and Rubine, will be extended to the third dimension: $P^3, $P+^{3}, $Q^3, and Rubine-Sheng. Two new variations are also introduced: $F for flexible cloud matching and FreeHandUni for uni-path recognition. Rubine3D, another extension of Rubine for 3D that projects the 3D gesture on three orthogonal planes, is also included. Following the selection of these 7 recognizers, the researchers were able to evaluate them in the traditional user-dependent and user-independent procedures with 3D trajectories from six datasets to show the efficiency of these recognizers depending on the conditions.

In summary, the results are unambiguous, $P+^{3} is more accurate than all other recognizers with a very highly significant difference, the effect size ranging from none to medium. $P+^{3} remains the most effective and efficient recognizer in terms of recognition rate, loss, and execution time [49].

2.5 QuantumLeap

2.5.1 Introduction

In the context of our thesis, we have based ourselves on the QuantumLeap [66] framework to manage gesture recognition. This framework allowed us not to start from scratch and this is why we will present this tool in this section.

QuantumLeap is a framework for engineering gestural user interfaces based on the Leap Motion Controller.

2.5.2 Definition

The QuantumLeap pipeline software architecture can be parameterized to define a workflow among modules for acquiring gestures from the Leap Motion Controller, segmenting them, recognizing them, and managing their mapping to the functions of the application.

2.5.3 QuantumLeap vs OpenNI

Compared to other existing applications such as OpenNI [58], QuantumLeap allows certain flexibility thanks to its modules. Indeed, our application initially had to accept 2D but also 3D. Therefore, being able to choose from the different existing modules to create our pipelines was more interesting. In addition, QuantumLeap allows us great flexibility as we could bring in any sensor or recognizer without having to make many changes.

Moreover, OpenNI was created based on the Kinect. As a result, doing the work in 2D seemed more complicated. Whereas QuantumLeap works with the Leap Motion Controller, which is the most accurate and robust tool for mid-air gesture recognition in 3D today.

Finally, QuantumLeap allows the use of different recognizers. You just have to choose the recognizer you want to use. This allows you to choose the best corrector for 2D but also 3D. Contrary to OpenNI whose recognition is done on data based only on the Kinect.

So, we decided to choose QuantumLeap as it allowed more modularity in the choice of module and recognizer. Moreover, the Leap Motion Controller implementation was already in place.

2.5.4 The Leap Motion Controller

The Leap Motion Controller [59] is an optical hand tracking module that captures the movements of your hands with unparalleled accuracy. It works with 2 cameras and some infrared LEDs. This allows for a wide area of interaction. This area can be represented as an inverted pyramid. Furthermore, the further away the hands are from the device, the less accurate the detections will be, as the distance influences the

accuracy. Then, once the Leap Motion Controller has detected the hands, it reads the data from the sensor into its local memory and then sends all the required data via USB.

Moreover, in addition to its affordability and wide availability, the Leap Motion Controller comes with an application programming interface (API) for acquiring raw gesture data that will be transformed into meaningful gestures for the developer as well as system-defined gestures.
However, when these system-defined gestures need to be modified or when new gestures need to be incorporated, the corresponding code is no longer appropriate and needs to be significantly updated. This leads to a significant programming effort requiring a mastery of oriented programming and machine learning.
Today, no software engineering environment facilitates the development of gesture-based user interfaces by offering certain independence from a Leap Motion Controller.

This is why QuantumLeap was created to fully deploy Leap Motion Controller-based interactive applications.

2.5.5 Pipeline Architecture

In this section, we will talk about the architecture of the pipeline. QuantumLeap is designed as an intermediate layer between gesture-based applications and Leap Motion Controller. It allows developers to associate gestures with actions without focusing on gesture acquisition, segmentation, and recognition. The QuantumLeap framework is divided into frozen spots and hot spots. A frozen spot is a "fixed" part that remains constant for all its instantiations, and a hot spot is a generic part that can be modified according to the application. As you can see in figure 2.14, there are 8 hot spots or modules:

- **Sensor interface(s)**: It's the interface between QuantumLeap and a sensor such as Leap Motion Controller, Microsoft Kinect, etc. It transforms the raw, sensor-specific data into a standardized format that can be used throughout the frame.

- **Filter**: Before a frame can be used by the other modules, filtering must take place to deal with noise that can have a negative impact on the experience. This module will therefore carefully filter each image to calculate a new image with reduced noise while not creating a time lag.

- **Static and Dynamic Datasets**: These modules transform a gesture dataset into a standardized format compatible with the framework. The datasets are then used to train static and dynamic recognition.

- **Static Recognizer**: The purpose of this module is to calculate a confidence score ranging from 0 to 1 to recognize static gestures in each image. A static gesture is a particular configuration of the body, a part of the body, or a stroke at a given moment in time. If the score of the gesture is below a certain threshold, the gesture is rejected, whereas a score of 1 indicates that the gesture is fully identified.

- **Dynamic Recognizer**: The difference with the static recognizer is that the Dynamic recognizer analyses image sequences rather than individual images. They are

therefore only called when the segmenter detects an intentional gesture. In the end, a confidence score is returned for each recognized gesture.

- **Analyzer**: These modules extract additional data from the sensor images, which can then be used inside applications. They can be used to offload heavy computation from the front-end to the QuantumLeap pipeline, implement more advanced feature extraction techniques that would otherwise clutter the front-end application, or remove sensor-dependent code from the application.

- **Segmenter**: The segmenter analyses the images one by one until it identifies a gesture. Once the gesture is recognized, it sends one or more lists of images back to the dynamic gesture recognizer for recognition. The aim of this module is therefore to identify intentional dynamic gestures from a data stream.

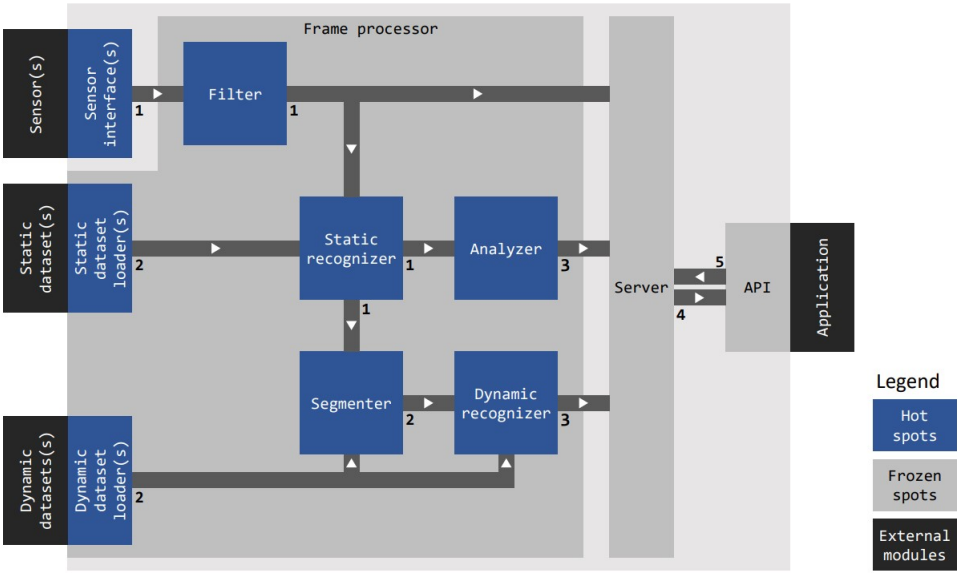

Figure 2.14: QuantumLeap Framework software architecture.
Source: Taken from [66].

2.5.6 UI

One of the advantages of QuantumLeap is its UI configuration which allows developers to quickly adapt it to their needs. It is divided into a series of pages. An overview page (Figure 2.15) that shows the complete architecture of the pipeline and allows navigation to the page for each module, a page with the complete API documentation (Figure 2.17), and pages dedicated to each module allowing the selection of a specific implementation and configuration (Figure 2.16).

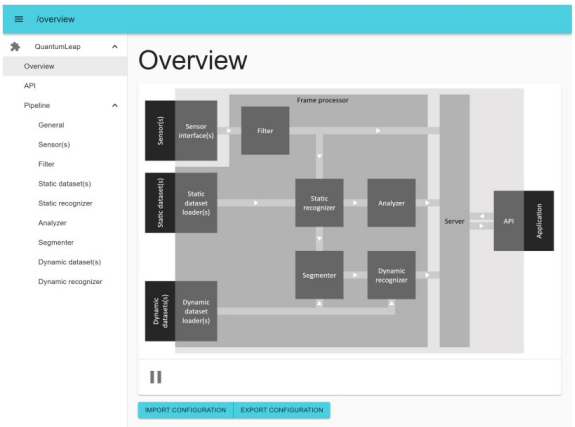

Figure 2.15: Overview page.
Source: Taken from [66].

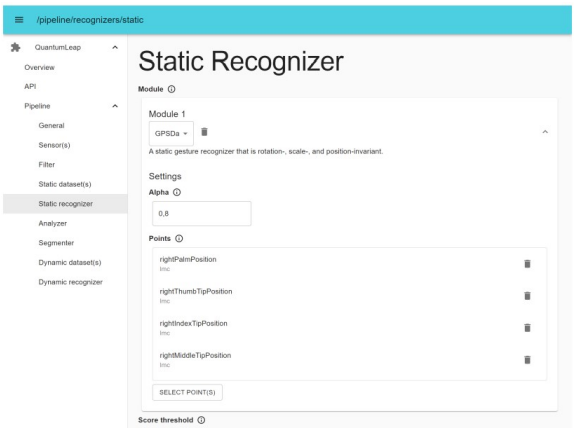

Figure 2.16: module page.
Source: Taken from [66].

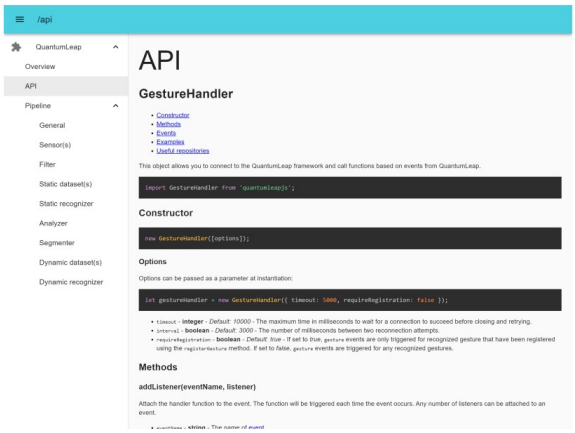

Figure 2.17: API page.
Source: Taken from [66].

2.5.7 Limitations

Firstly, the document provided to introduce QuantumLeap does not have any documentation for understanding how to configure the gesture recognition pipeline. Therefore, even if the effort to develop a gesture-based application is limited, knowledge of the modules is required to build an efficient pipeline.
In the case of our thesis, we encountered this problem. When we had to configure our pipeline, we had difficulty in understanding and modifying the code due to its complexity.

Secondly, although QuantumLeap advocates modularity, there is some rigidity such as when adding new module types or changing their order in the pipeline. It is difficult to perform such actions without modifying the framework source code. This rigidity makes QuantumLeap more accessible to less experienced developers but may prevent some developers from experimenting with more advanced gesture recognition techniques. The configuration process becomes simpler with a rigid architecture because

many choices are already made for the developer (for example, the order between the different modules). All that remains afterward is to choose the type of each module (which recognizer, which sensor, etc.). On the contrary, if the architecture was more modular, for example, by letting the developers arrange the modules as they want, it would be much more powerful. However, it would require more work and knowledge to create a good pipeline, so it is less accessible.

Furthermore, the evaluation of QuantumLeap was based on two applications (DICOM and Large User Interface) which demonstrate that complex gesture interfaces can be built in relatively few steps in two different contexts. However, these two examples only cover a limited number of possible gesture applications including connected objects.

Finally, even though QuantumLeap allowed us to start our work by not having to create the recognition part ourselves. It is implemented taking into account only the Leap Motion Controller and therefore 3D. However, no functionality exists to allow the use of 2D. Even if the recognizers used to allow to recognize 2D gestures for example $P+ (see section 2.4.3), no pipeline or sensor is planned for the recognition of 2D gestures. So we decided to create our pipeline.

Concerning datasets, there is no way to create a dataset used to train our recognizers other than by implementing it in the code. Indeed, the only way to add new gestures in datasets was to create a JSON with the right format and to add it by hand in the code at the right place. This is not very practical in the case of our application as we wanted to allow users to be able to create their gestures.

We will discuss these limitations and how we dealt with these problems more in detail in chapter 3.3.

2.6 Address and Command

2.6.1 Definitions

In this section, we will discuss 'Address and command' [60]. This is a model of mid-air interactions using multiple remote devices, mid-air interaction being a distinct paradigm from natural human-computer interaction where users apply non-contact gestures to interact with distant displays or devices.
The A&C model requires users to use both hands. A&C only looks at the upper body. The dominant hand performs commands (see figure 2.18) and the non-dominant hand addresses a device (see figure 2.19). This allows for interactions that simultaneously address and control multiple devices with the help of Kinect in ubiquitous environments such as smart homes, which respond to mid-air inputs.

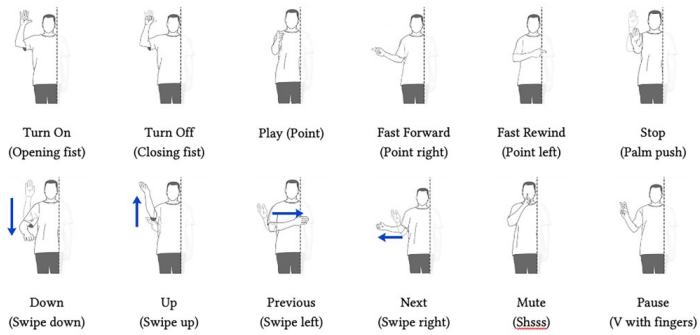

Figure 2.18: Commands and the corresponding gestures performed with the dominant hand.
Source: Taken from [60].

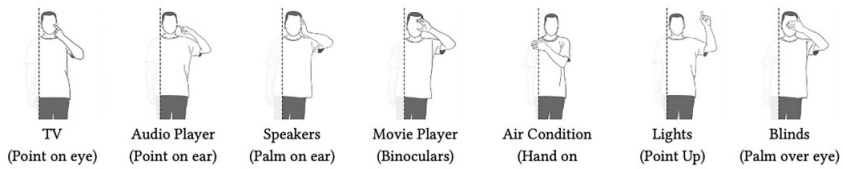

Figure 2.19: Commands and the corresponding gestures performed with the dominant hand.
Source: Taken from [60].

The A&C design follows five principles:

1. An address gesture is applied with the non-dominant hand (the left for most people).
2. Address gestures are unique for each device (target). All of them are static (postures).
3. While a device is addressed, a command gesture can be applied to that device.
4. Command gesture is applied with the dominant hand.

5. Command gestures are consistent among devices.

These A&C interactions allow the design of a single gesture for the same command when applicable to multiple devices. This reduces the total number of gestures to be memorized and facilitates the learning of the gesture set.

2.6.2 Advantages of Combination of Gestures

Indeed, the study showed that the use of common gesture commands and their association with simple gestures helps to limit the size of the set of gestures. This makes it easier for users to memorize the set of gestures. This has been demonstrated because if we assume that there are 5 devices in the ecosystem that share 15 command gestures for common commands. The set of gestures will consist of $15 + 5 = 20$ distinct gestures, the combination of these gestures will be used to perform $15 \times 5 = 75$ tasks. In other words, the user will have to combine 5 postures with one (non-dominant) hand and 15 gestures with the other (dominant) hand to perform 75 tasks. Therefore, there is a significant saving in terms of the number of gestures that need to be remembered, even with the addition of other devices.

However, compared to our application, A&C is more limited. There is a certain ambiguity since one cannot realize commands with more than 2 gestures. Let's imagine that we want to turn on the television and set the volume to 5, it is not possible to perform this action with one command with the A&C model. While with our application, we can manage an Address with Devices and the Command with Action. In addition to this, we also have parameters such as setting the volume to 5. We can also choose the environment in which we want to perform a command. We can decide to turn on the TV in the kitchen even if we are not in the room. You will have more details about our grammar in chapter 3.

2.6.3 Challenges

Then, since A&C takes place in ubiquitous environments, many design challenges for control have arisen.

- Having multiple interaction targets in the same environment increases the complexity of the various steps from design to gesture evaluation.
- As the number of devices increases, the potential controls can be numerous even though the model reduces the set of gestures to be remembered.
- The Midas touch can be a problem. When a user interacts with a device, no other device should be accidentally activated. Therefore, the user must address the device before (or during) the interaction [61]. During the interaction, the discussion between the user and the device must be isolated from other devices.
- Prototyping mid-air interactions with multiple targets in physical space is more demanding, as there are no readily available platforms for this type of research. As a result, most current research is limited to the results produced by gesture elicitation studies, which most often involve a single home device.

- In the smart home, users interact several times a day with multiple devices, possibly without wearing special sensors or accessories, so it is desirable to design interactions without accessories.

Now that you know all the challenges the study faced, let's talk about the results of the observations. For this experiment, a Kinect was used to recognize gestures made by people and also to interact with a virtual environment created with the help of Unity representing a smart home(see figure 2.20).

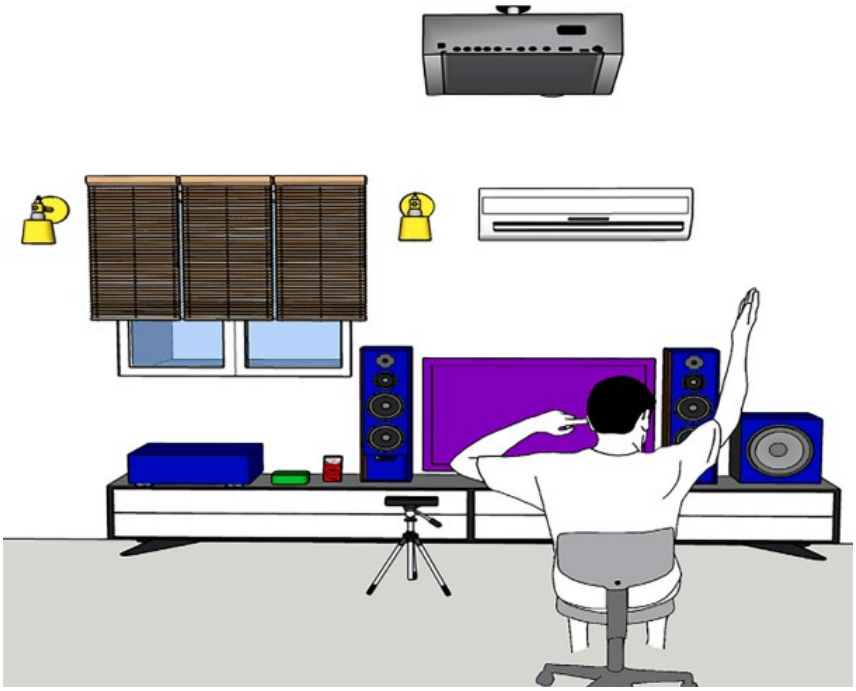

Figure 2.20: Virtual environment created with the help of Unity.
Source: Taken from [60].

2.6.4 Result of the Study

The evaluation of A&C model was evaluated in a two-phase laboratory including a learning session and a testing session. A task was considered successful when the user successes to execute a command before 4 repeated attempts, unsuccessful otherwise. It was also necessary to be careful with the false negatives which are an error that occurs when a user performs correctly the gestures but the system does not respond. there is also the false positive where an error occurs when a task is performed even if the participant makes the wrong gesture.

As you can see in figure 2.21 the results of both approaches are very satisfactory. There are no significant differences observed between the 2 sessions. Concerning the time, we can see that for most tasks, the completion time showed some improvements between the 2 sessions. Finally, The A&C model presented considerably fewer false-negatives errors than the two-step approach and both approaches have very low false positive errors.

In conclusion, thanks to this study on the A&C model for mid-air interactions with several home devices, we could see that the results were encouraging. These results

are encouraging for our application because the study proved that using separate gestures allows us to remember more commands with a small number of gestures. This means that our application allows remembering more commands since it has more categories of gestures namely Action, Device, Parameters, and Environment while keeping the number of gestures to a minimum. Moreover, A&C is more limited in terms of modularity. Indeed, since Address is assigned to the non-dominant hand and Command to the dominant hand, there can be no congruence between the gestures. Thanks to our grammar (see chapter 3), we can perform gestures in any order. This is why another advantage of our work is that it is more modular.

Table 3
. Quantitative results from the evaluation process. (L.S.= Learning Session, T.S.= Testing Session, C.I.=Confidence Interval ($a = 0.05$)).

	Task Success		Task Completion time (*sec*)				False Negatives (Errors)				False Positives (Errors)			
	LS	TS	LS	CI	TS	CI	LS	CI	TS	CI.	LS	CI	TS	CI
TV Turn on	100%	100%	6.4	1.8	6.0	1.8	1.1	0	1.3	0	0.1	0	0.1	0
TV Turn off	94%	88%	5.5	1.3	3.7	0.5	1.6	1	1.1	0	0.1	0	0.0	0
TV Next channel	82%	94%	5.6	1.2	4.8	1.1	1.8	1	1.2	0	0.1	0	0.1	0
TV Previous channel	100%	76%	4.6	0.9	4.6	0.7	0.9	0	1.5	0	0.2	0	0.0	0
TV Volume up	100%	100%	5.3	0.9	4.5	0.4	0.5	0	0.2	0	0.0	0	0.0	0
TV Volume down	100%	100%	5.2	0.7	5.1	1.0	0.8	0	0.6	0	0.0	0	0.0	0
TV Mute	76%	94%	5.6	1.5	4.7	0.9	2.2	1	1.7	1	0.0	0	0.0	0
Blinds up	100%	76%	6.4	1.6	5.3	1.8	1.2	0	1.9	0	0.0	0	0.1	0
Blinds Down	82%	71%	5.4	1.4	4.5	0.5	2.2	1	1.1	0	0.0	0	0.0	0
Blinds Stop	76%	65%	3.7	0.4	4.1	1.4	1.5	1	2.2	3	0.0	0	0.0	0
Lights Turn on	100%	100%	4.2	0.6	4.1	0.6	0.1	0	0.4	0	0.0	0	0.0	0
Lights Turn off	94%	100%	4.4	1.0	3.6	0.6	0.6	0	0.5	0	0.0	0	0.1	0
Lights Dim Up	100%	100%	4.9	0.6	4.5	1.0	0.4	0	0.6	0	0.0	0	0.0	0
Lights Dim Down	100%	100%	5.5	1.2	4.6	0.8	0.6	0	0.8	0	0.0	0	0.0	0
Air Cond. Turn on	94%	100%	4.2	0.5	4.6	0.9	0.8	0	0.8	0	0.0	0	0.0	0
Air Cond. Turn off	82%	94%	4.6	1.0	3.9	1.0	1.7	0	1.0	0	0.0	0	0.0	0
Air Cond. increase temp.	94%	94%	4.4	0.8	3.8	0.3	0.7	0	0.4	0	0.0	0	0.0	0
Air Cond. decrease temp.	94%	100%	5.1	1.2	4.1	0.6	1.0	0	0.4	0	0.0	0	0.0	0
Audio Player Turn on	100%	100%	4.6	1.2	5.1	1.6	0.8	0	1.1	0	0.1	0	0.2	0
Audio Player Turn off	88%	94%	4.9	0.8	5.1	1.2	1.5	0	1.2	1	0.1	0	0.0	0
Audio Player Next	94%	94%	5.2	1.5	3.9	0.6	1.4	0	0.6	0	0.1	0	0.0	0
Audio Player FF	100%	100%	4.3	0.6	4.8	0.8	0.5	0	0.6	0	0.1	0	0.2	0
Audio Player FR	100%	94%	4.8	0.8	4.1	0.5	0.5	0	0.6	0	0.2	0	0.1	0
Audio Player Play	100%	100%	3.6	0.3	2.9	0.3	0.1	0	0.0	0	0.0	0	0.0	0
Video Player Turn on	100%	100%	6.7	1.6	5.1	1.3	1.5	1	0.9	0	0.6	0	0.5	0
Video Player Turn off	82%	88%	4.9	1.1	3.8	0.3	1.8	0	0.9	0	0.1	0	0.1	0
Video Player FF	94%	100%	6.2	1.1	4.2	0.6	1.2	1	0.6	0	0.2	0	0.2	0
Video Player FR	100%	94%	4.8	0.5	4.8	0.7	0.5	0	1.1	0	0.1	0	0.1	0
Video Player Play	100%	100%	4.3	0.6	3.8	0.9	0.5	0	0.5	0	0.2	0	0.2	0
Video Player Stop	94%	94%	4.9	1.1	5.4	1.6	1.2	0	1.3	0	0.1	0	0.3	0
Video Player Pause	100%	100%	3.4	0.3	3.6	0.7	0.0	0	0.4	0	0.0	0	0.0	0
Speakers Turn on	100%	100%	4.8	0.6	4.8	1.6	0.4	0	0.8	0	0.0	0	0.0	0
Speakers Turn off	100%	100%	5.0	1.2	3.8	0.6	0.8	0	0.2	0	0.0	0	0.0	0
Speakers Mute	100%	100%	5.1	0.8	3.7	0.5	0.6	0	0.0	0	0.0	0	0.0	0
Average values	95%	94%	5.0		4.4		1.0		0.8		0.1		0.1	

Figure 2.21: result from the evaluation process
Source: Taken from [60]

Chapter 3

Composition of Stroke Gestures into Instructions in EMERITI

In this chapter, we will tell you about the work we have accomplished.
First, in section 3.1, we'll talk about the grammar we created to allow us to have a syntactically correct language, some examples to be able to understand better, the steps we realized to arrive at our final result as well as the advantages of our grammar compared to the one of 'Address and Command'.

Then, in section 3.2, you will discover an exploratory analysis to allow you to imagine what our application could be in the future.

3.1 General Format

Always intending to improve memorization and the use of gestures to execute commands, we have created an environment allowing to test our different formats. Indeed, our application allows the combination of several instructions but also realizes macro instructions by merging several instructions. Moreover, our application being very permissive, we also allow the user to create simple elements and use them together in the recognizer to obtain instruction afterward. Finally, the application also allows the merging of these simple elements in any order.

3.1.1 Definitions

In this section, we will discuss a general format for expressing the instruction of a user performing tasks. First, we defined some terms to facilitate the understanding of our format.

- Action: an action to be performed on one or more devices, i.e. turn on, turn off, etc.
- Device: a device on which functionality will be executed, i.e. television, radio, etc.
- Parameter: value whose choice allows to adapt the functionality of a device, i.e. channel on 5, volume on 3, etc.

- Environment: a place where the action will be executed, i.e. kitchen, living room, etc.
- Macro: word or sentence which includes one or more previously created instructions, i.e. all(contains all the created instructions), macro_lights (contains all the created instructions which include modifications on the lights), etc.

3.1.2 Instructions

With these 4 definitions, we have created an Extended Backus Naur Form (EBNF) expression:

Instruction : = Action {Device} [{Parameter}] [{Environment}] | {Macro }

We have formulated this definition like this because it is expressed in a logical order. But we can make permutations between the different elements of this statement. It's opportunistic programming.

Moreover, this instruction is decomposed into other instructions to make it easier to understand. There are other definitions:

1. Basic Instruction = Action Device [{Parameter}][{Environment}]
   Basic instruction is an action that can execute 0 or more parameters on a Device. That action should be executed in 0 or more environments.

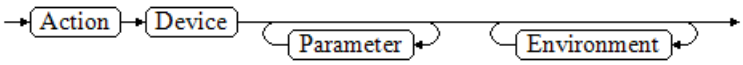

Figure 3.1: Visualization of a basic instruction

Because of the EBNF visualizer, we could not represent all permutations. Therefore, don't forget that we can also have a permutation, so we can have other sequences of elements in the statement for example :

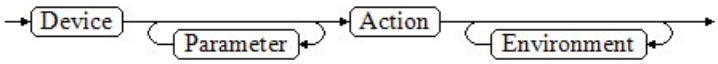

Figure 3.2: Permutation of a basic instruction

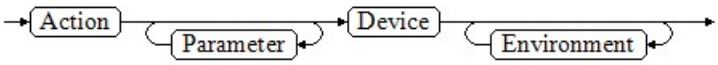

Figure 3.3: Permutation of a basic instruction

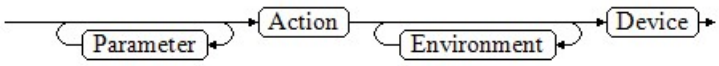

Figure 3.4: Permutation of a basic instruction

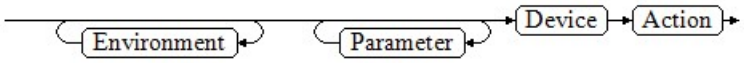

Figure 3.5: Permutation of a basic instruction

2. Common Instruction = Action {Device} [{Parameter}] [{Environment}]
   A common instruction is an action that can execute zero or more parameters on two or more Devices. That action should be executed in zero or more environments.

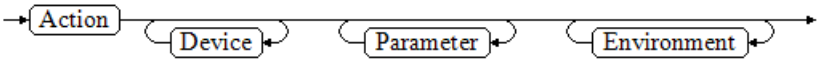

Figure 3.6: Visualization of a common instruction

3. Macro Instruction = Macro
   A macro instruction is a shortcut. If you want to carry out several instructions without having to make a gesture for each one, you can create a macro to combine the different instructions in a single gesture. For example, the macro instruction "Macro living room" allows to combine the instructions "Turn on the lights in the living room" and "Turn on the television on channel 3".

4. Composed Instruction = {Common instruction} | {Basic instruction} | {Macro instruction}.
   A composed instruction is one or more common and/or basic and/or macro instructions.

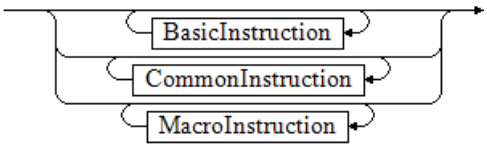

Figure 3.7: Visualization of a composed instruction

3.1.3 Elements

In order to obtain the most permissive application possible, we decided to define a new term: "element". The fusion of these elements will be used later to create the combinations of gestures that will allow later to obtain instructions.

1. Element = Action | Device | Parameter | Environment
   An element is a unique action, device, parameter, or environment.

2. Composed Element = ([Action] | [{ Device }]) [{Parameter}] [{Environment}]
   A composed element is a composition of elements. We can have either zero or one action, either zero or several devices followed by potential parameters and/or environments. The only rule is that a composed element can not have an action with a device. Otherwise, this would be a basic or common instruction.

3.1.4 Combination of Gestures

Now that we have been able to define what we mean by "instruction" and "element", we will be able to explain the use of our application concerning the combination of gestures. Indeed, the combination of gestures is essential to allow hierarchical grammar (see Section 3.1.6).

To be clear on the use of the term "gesture" and "combination of gestures", it is important to define them clearly.

1. Gesture = Basic instruction | Common instruction | Macro instruction | Composed element

2. Combination of gestures = {Gesture}

A gesture is therefore either a Basic/Common/Macro instruction or a Composed element.

These gestures can therefore be combined to create a combination of gestures and therefore new instructions.

End users can thus create sets of gestures with fully optimized combinations to reuse the gestures. In addition, gesture combinations can improve recall compared to simple gestures. And finally, combination gestures are preferred when simple gestures have an abstract gesture-command correspondence [3].

3.1.5 Nassi-Schneiderman Diagram

We realized a Nassi-Schneiderman diagram to represent an instruction. We choose this graph because it allows having a global view of instruction.

As you can see in figure 3.8, we have several steps to have correct instructions. Until we have an instruction to analyze, we check the case where we are in a composed instruction. In a composed instruction, we can have basic, common, or macro instruction.

If we have a basic instruction, we look until there is no more element in the basic instruction. If we still have an element, we look at whether an action or device has not been taken because a basic instruction accepts only one action and one device.

If we have a parameter or an environment element, then we accept it. The same procedure is used for a common instruction, except that this time we can accept several devices.
We find the same way of doing things for a common instruction except that this time we can accept several devices. Finally, the command macro only accepts macros.

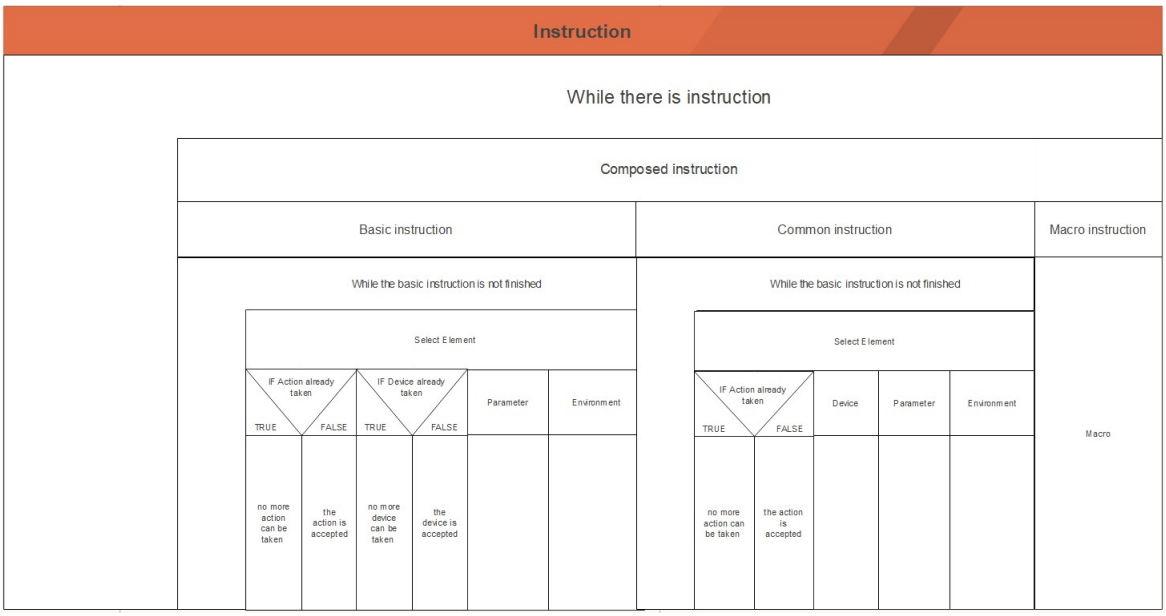

Figure 3.8: Nassi-Schneiderman's Graph

3.1.6 Explanation of 2D Stroke Gestures by Congruence and Hierarchy

As explained in section 2.3, there are congruent and hierarchical gestures. We have already explained the principle but we will explain more concretely what they represent. Congruent means that related commands produce related gestures. For example, as you can see in figure 3.9, the 2 gestures for "Turn TV on" and "Turn TV off" would be "Draw a circle in the clockwise direction" and "Draw a circle counter-clockwise direction" in congruent and non-hierarchical grammar.

On the other hand, hierarchical gestures are gestures in which we assign every single action in the sentence to gestures. For instance in figure 3.9, "Circle" represents "Turn", "Rectangle" which is for "TV", and "V" for "on".

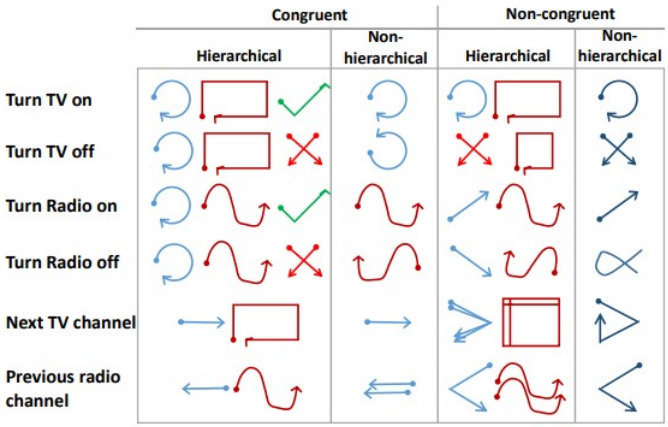

Figure 3.9: Comparison of 2D stroke gestures by congruence and hierarchy
Source: Based on [39]

3.1.7 Examples

For example, we created a series of actions, devices, parameters, environment, and macro. We realized some possibles combination with our format.

Action	Device	Environment	Parameter	Macro
Turn on	Television	Kitchen	Volume on NUMBER	MacroLiving
Turn off	Washing machine	Living room	Time in NUMBER	
Increase	Light	Office	Channel NUMBER	
Decrease	Radio	Parent bedroom	Program NUMBER	
Pause	Air Conditioner	Children bedroom	Brightness on NUMBER	
Play	Computer	Bathroom		
Mute	Fan	Dining room		
Unmute	Microwave	Laundry room		
Next				

1. Basic Instruction

   - Turn on the washing machine
   - Turn on the light in the dining room with brightness on 5
   - Turn on the light in the laundry room
   - Turn off the fan in 5 minutes
   - Increase the television volume on 2
   - Mute the television in 9 minutes in the living room

   Don't forget that you can swap all the elements of the instructions. You can also have :

   - In 9 minutes the television in the living room mute
   - The television in 9 minutes mute in the living room
   - In 9 minutes mute the television in the living room
   - In the living room in 9 minutes the television mute

2. Common Instruction

   - Turn on the microwaves and the computer in the office
   - Turn off the fan and the lights in 5 minutes
   - Turn off the lights and the air conditioner in the children bedroom in 5 minutes
   - Increase the television and the radio in the parent bedroom with the volume on 3
   - Pause the television and the air conditioner

3. Macro Instruction

   - MacroLiving (Turn on the light in the living room and turn on the television on channel 3)

- MacroLightsOffBedroom (Turn off the light in the parent bedroom and children bedroom and turn off the radio)

4. Composed Instruction

   - Decrease the volume of the radio to 1 and turn off the air conditioner in the children bedroom
   - Play the television and the radio in 2 minutes and macroLiving
   - Play the television and decrease the light in the kitchen with the brightness to 6
   - Play the radio and pause the television
   - Play the radio and turn off the television
   - Unmute the television and the radio in 2 minutes and macroLiving

   If you want to realize a permutation in a composed instruction, you always have to start with an action. If you don't do that, there can be some problems in the separation of instructions. For example, if you have Play the radio and the television turn off, the grammar will understand that you want to play the radio and the television instead of playing the radio and turn off the television.

5. Element

   - Unmute
   - Television
   - Radio
   - Kitchen
   - Office
   - Volume on 3
   - Brightness on 2

6. Composed Element

   - Unmute
   - Television
   - Radio, washing machine, and fan
   - Kitchen, dining room, and laundry room
   - Office
   - Brightness on 2 and Volume on 3
   - Play children bedroom and parent bedroom
   - Air conditioner and fan program 3 in 5 minutes
   - Next in the laundry room program

7. Gesture

   - Turn on the washing machine
   - Turn off the fan in 5 minutes
   - Turn off the fan and the lights in 5 minutes
   - Turn off the lights and the air conditioner in the children bedroom in 5 minutes
   - MacroKitchen (Turn on the microwave and the light in the kitchen in 2 minutes and turn off the television and the light in the living room)
   - Turn on
   - Television
   - Light
   - Play children bedroom and parent bedroom
   - Air conditioner and fan program 3 in 5 minutes

8. Composition of gestures

   - Turn on + Television + Air conditioner and fan program 3 in 5 minutes= Turn on the television, air conditioner, and fan to program 3 in 5 minutes
   - Increase + Light children bedroom brightness to 8 + Light parent bedroom = Increase light in children bedroom and parent bedroom brightness to 8
   - Increase + Light children bedroom brightness to 8 + Turn off the light in the kitchen =
     Increase the light in children bedroom brightness to 8 and Turn off the light in the kitchen

   As explained above, the purpose of the composition of gestures is to execute several gestures one after the other to perform a more complex instruction. This step can only be performed from the drawing area and in no case is there any question of creating a shortcut. However, if you want to create a shortcut to this set of gestures, it is better to create a macro instruction instead.

3.1.8 Parser Generator

Once we have created our grammar, we have to prove that it is correct. To do this, we looked at different parser generators. We made our rules on Xtext and ANTLR. At first, we noticed that the Xtext framework did not allow permutations. Indeed, it does not accept the case where 2 permutations start with the same element but the end changes. For example, when we define "Action Device Environment Parameter " and "Action Device Parameter Environment ", then Xtext returns an error. Therefore, we could not use Xtext to perform the proof of our grammar.

After that, we realized our grammar on ANTLR [62]. ANother Tool for Language Recognition is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It is widely used to build languages, tools, and frameworks. From grammar, ANTLR generates a parser that can build and

traverse parse trees. This allows us to know if our grammar is syntactically correct or not.

To implement our code, we created a rule to define an instruction. As explained earlier, an instruction can be a basic, common, macro, or composed instruction. The composed instruction itself can contain basic instruction, common instruction, and macro instruction.
So we defined the basic instruction as the set of possible permutations including one action, one device, 0 or more parameters, and 0 or more environments. You can see below the result of 2 basic instructions.

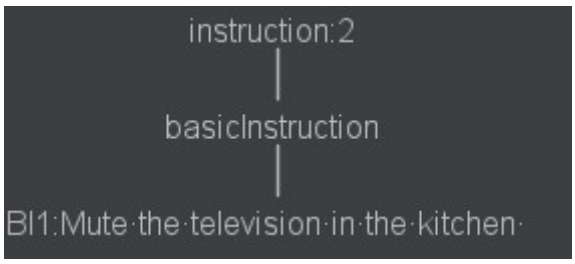

Figure 3.10: Syntax tree of a logical basic instruction

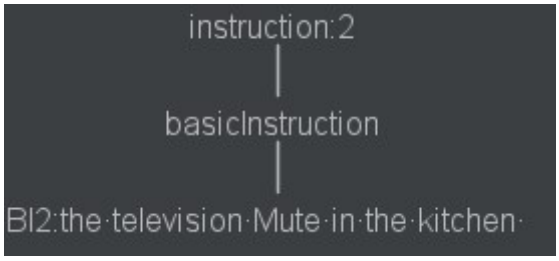

Figure 3.11: Syntax tree of a permuted basic instruction

The first (3.10) is the syntax tree of basic instruction in a logical sequence: "Mute the television in the kitchen ". The other (3.16) is a permutation of the first instruction "the television Mute in the kitchen ". In both cases, we can see that the syntax tree is formed correctly.

Then, the common instruction is similar to the basic instruction except that you can have 2 or more Devices.
We can also see with the syntax trees below(3.15 that the common instructions are also correctly managed. The first tree is for the sentence "Turn off the lights and the air conditioner in the kitchen in 5 minutes " and the second (3.13) for " in the kitchen the air conditioner the lights Turn off in 5 minutes ".

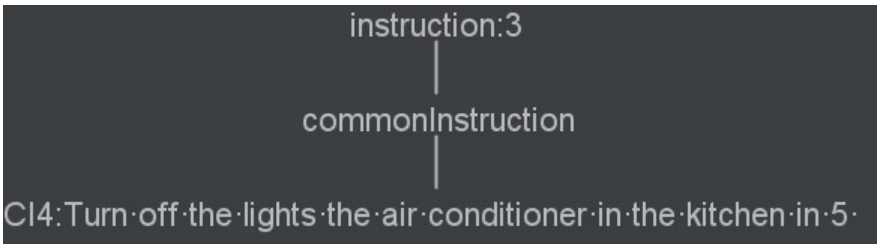

Figure 3.12: Syntax tree of a logical common instruction

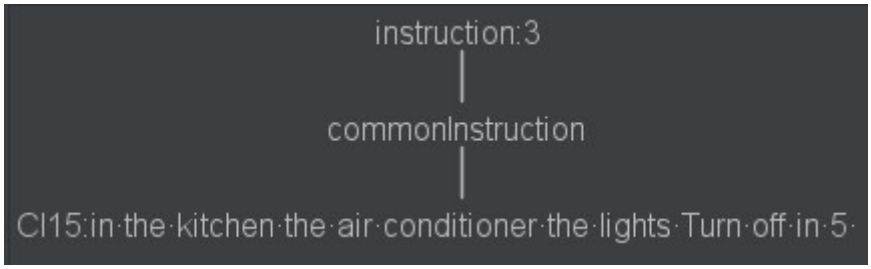

Figure 3.13: Syntax tree of a permuted basic instruction

Next, a macro instruction is represented by a macro, and a correct tree (3.14) is found when a macro instruction arrives.

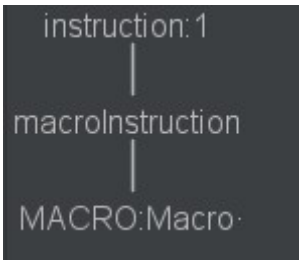

Figure 3.14: Syntax tree of a macro instruction

Finally, the greatest challenge was composed instruction. Indeed, it was necessary to be sure that all the different types of instructions were well identified. To make sure, we made two sentences. One in a logical sense and the other being a permutation.
" Mute the television in the kitchen || Turn off the lights and the air conditioner in the kitchen in 5 minutes || Macro "
and " Mute in the kitchen the television|| Macro || Turn off in the kitchen the air conditioner the lights in 5".
You can see that the syntax tree is correctly realized and so we correctly decomposed the composed instructions.

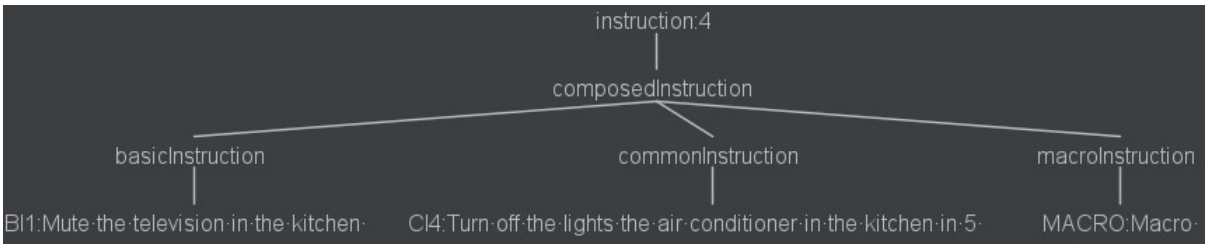

Figure 3.15: Syntax tree of a logical composed instruction

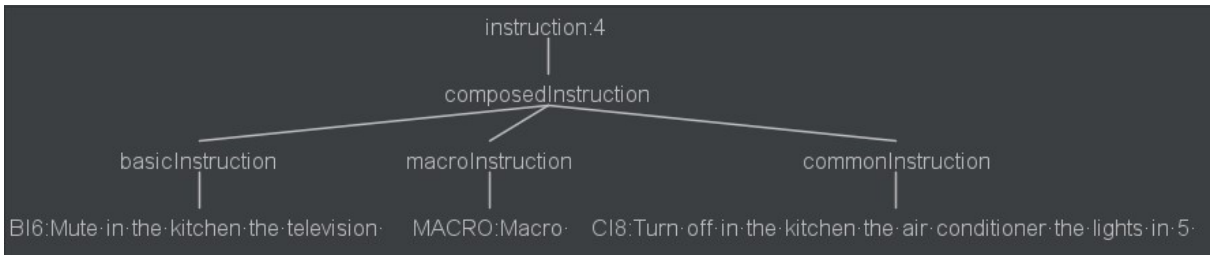

Figure 3.16: Syntax tree of a permuted composed instruction

In conclusion, we proved that the syntax is correct for basic, common, macro, and composed instructions. This is why we can say that our grammar is syntactically correct.

3.1.9 Process of the creation of the grammar

In this section, we will explain the whole process of arriving at our final version of the EBNF grammar.

Initially, we created our grammar from only 3 elements namely Action Device Parameter. Action and Device were correctly formed from the start but we had some problems with Parameter. Indeed, we confused Environment and Parameter. Therefore the separation between the different environments (kitchen, room, etc.) and the real parameters (volume, etc.) became very complicated. This is why we decided to create a 4Th element in our grammar which is Environment.

After that, another problem came up, which is macro instructions. Initially, a macro instruction was represented by an Action. However, when we made our grammar on ANTLR, we realized that the grammar was not syntactically correct. Indeed, when creating a composed instruction, we detected a problem. The parsing worked very well when the instruction was formed logically but when permutations occurred, the macro could become another type of instruction. You can see this for yourself in Figure 3.17. This is why we decided to create a 5Th element in our grammar, namely Macro.

Then, in the case of composed instructions, we realized that when an instruction did not begin with an action, it did not work. For example, if a user wants to perform the composed instruction "Turn on the television Turn off the radio" in a permuted sense such as "Turn on the television the radio Turn off ", the grammar would include "Turn on the television the radio", " Turn off" when this is not what the user wanted. That's why we added the rule of having to start with an action when a basic or common statement is used inside a composed instruction.

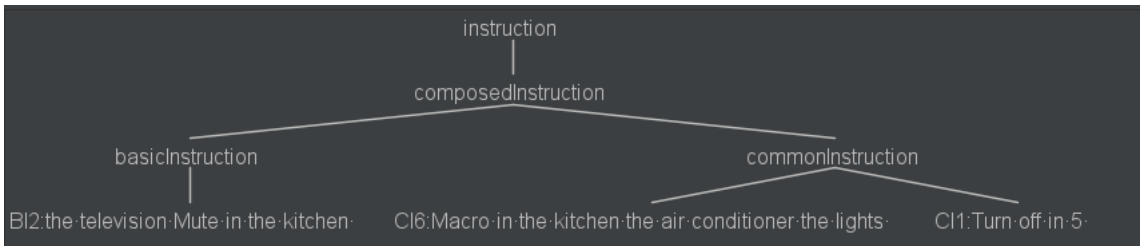

Figure 3.17: Visualization of the error in composed instruction

In the end, the whole process takes us 3 weeks to realize the final version of the grammar. It took a lot of thought to create valid grammar and also to find viable examples. As explained above the process was iterative and each time we had to add and/or modify the work already done.

Then, in addition to creating our grammar, we had to find ways to visualize it. We made a Nassi-Schneiderman diagram so that the EBNF could be visualized and we used an EBNF visualizer.

Finally, the creation of grammar is a good thing, but it must also be possible to prove that it is syntactically correct. We have done this with ANTLR and XText.
This is why the entire process takes 3 weeks to be achieved.

3.1.10 Advantages Compared to A&C

Thanks to all these examples, we can see that our grammar is better than "Address and Command" [60] because we allow very long combinations of gestures whereas, for A&C, we can only make 2 gestures each time to carry out a command.

Then, one of the advantages of A&C was that the application managed several devices. However, the grammar of our application manages multi-devices but it also allows to manage different environments. Indeed, we can choose to turn on a light in the kitchen or to turn on a light in the living room.

Another advantage of our grammar is that it also allows us to be more precise about the parameters. Indeed, the command in A&C allows making basic actions like "Next", "Pause", "Up", etc. Our grammar allows us to set the volume to 5 or to turn on the lights with a certain brightness. This increases the possibilities of instructions that can be carried out.

Finally, our grammar allows the permutation between its different elements. This means that you don't have to worry about the order in which you perform the gestures except for composed instructions for which the action must come first. Contrary to A&C which only allows to carry out an "Address" then a "Command" compared to this Address.

3.2 Exploratory Analysis

In this section, we will explain to you a work realized before the implementation of our application. This is an exploratory analysis in which we imagined a scenario. This scenario presents a family using our ideal vision of EMERITI. This means that our application replaces the keyboard and the mouse and uses 3D. The family members live in a smart home with various connected devices in different rooms where they can interact with.

3.2.1 Scenario

Mr. Marty McFly is married to Mrs. Lorraine McFly and they adopted little Robert Brown 12 years ago. The whole family loves everything about technology and has recently acquired a LeapMotion which is a 3D gesture sensing device. It can be connected to a PC. They have also bought a television, a Prodigio (the first connected Nespresso machine), and a connected thermostat. In order to experiment with their new purchases, Mr. McFly downloaded the QuantumLeap application, a software that allows 3D gestures to be linked to connected devices.

First of all, Mr. Mcfly launches the application. He sees that the family can create one or more accounts and add several family members to the same account. He can also access a list of connected devices to which he can add or remove an object.

Mr. McFly wishes to create a profile, he is asked to enter a gestword, i.e. a password created from one or more gestures, his personal information, the different devices he wishes to use as well as to choose or create an order of priority concerning the other members of the family. Mr. McFly can assign himself the role of administrator or parent. It is also possible to create a "child" profile for little Robert Brown. This will allow him to limit certain functionalities such as the use of certain devices and/or define a time slot during which he will not be able to use certain objects.

After the creation of his profile, Mr. McFly can connect to it. Once in his personal space, several options are available to him.

He has access to a catalog of gestures that correspond to all the gestures most frequently used by the community to perform certain actions (e.g. turn on, open, increase, etc.).

To use his connected objects, Mr. McFly must create a new combination of gestures and assign it to action. Wishing to use the Prodigio device, he decides to choose it from the list of connected devices. After that, he can choose one of the functionalities available on the device. Wanting to drink a ristretto, he decides to assign a sequence of gestures to this action, consisting of a gesture for "drink" and another gesture for "ristretto". To do so, the application asks him to perform these gestures 5 times to have a good recognition of them.
After creating his gesture, Mr. McFly goes back to the main menu and tries the gesture sequence to verify that Prodigio is indeed making a ristretto. As if it was magic, the Prodigio starts preparing a ristretto. To see the progress of his coffee, QuantumLeap

displays a loading bar on the screen.

Wishing to turn on the television, he performs the same steps as before. Having no idea how to choose his gestures, he decides to access the recommendation system based on the movements of other people to advise him. He chooses a gesture for "switch on", another gesture for "television" and one for "in the kitchen".
After creating his gestures, he performs them and sees the television turn on.
To view all his previously created gestures, he decides to consult his personal catalog. Noticing that his gesture for "turn on" does not correspond to him, he has the choice of modifying it or deleting it. Mr. McFly decides to modify it.
It is also possible to test the different gestures in the list to verify their correct functioning. He can also improve the recognition by performing the gesture several times to add new "templates".

Intrigued by the novelty, little Robert decides to create his child account. As mentioned above, the account will limit access to certain features. Mr. McFly uses his administrator account to prevent his son from using the thermostat and to limit access to the television from 8 am to 8 pm. After setting the limitations, Mr. McFly logs out of his account.

Like his father, Robert follows all the steps to create an account. To save time, he also decides to copy some of his actions to use them on his account.
Wishing to test the creation of a new gesture, he decides to find one to make hot chocolate with Prodigio. Unfortunately, his gesture sequence is identical to the one for creating a ristretto coffee that he had copied just before. An error appears on the screen telling him that the gestures have already been assigned to an action. He can either modify the previously created gesture or modify the gesture he has just created. He decides to choose another one and returns to the main menu.

Clumsily, Robert tries to make himself a hot chocolate but, in his haste, he makes the wrong combination, and nothing happens. Therefore, he decides to re-do the combination correctly and hot chocolate is launched.
Finding himself in the time slot to watch television, he decides to turn it on using the gesture copied from his father and settles down to watch his favorite series.

Finally, the day ends and Robert logs out of his profile and leaves the application.

3.2.2 Process Modeling

With this scenario, we created 26 different tasks that you can see in A.1. Based on this, we created a process modeling and we decided to divide the different tasks according to their nature in the application. It was difficult for us to divide the different tasks according to the different jobs and users due to the limited number and lack of interest in our case.

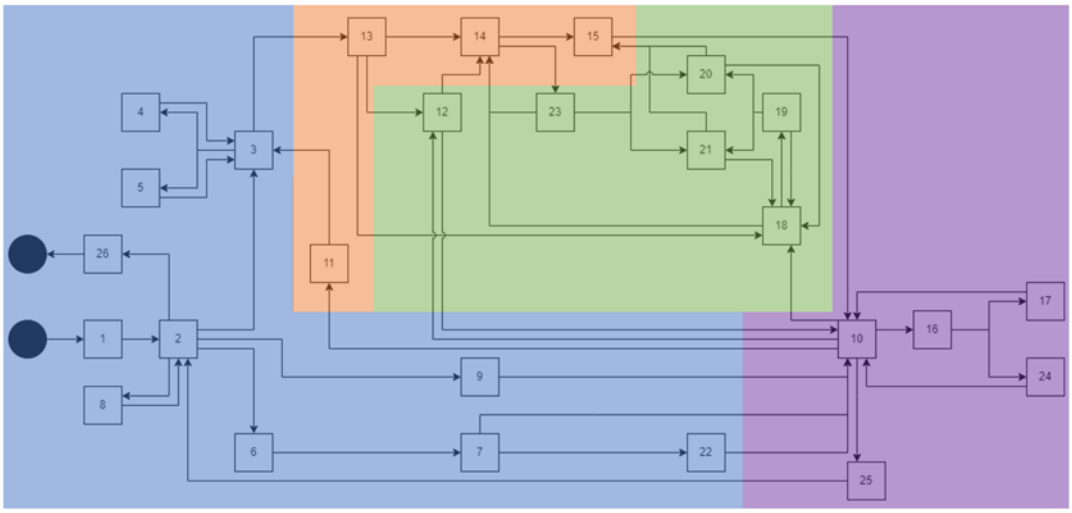

Figure 3.18: process modeling of our scenario

Therefore, as you can see, we divided the tasks into 4 sections:

- Start-up and home menu: This section is about starting the software and the different features that can be used in the main menu.

- Creating a gesture: This section is dedicated to the creation of a gesture and the different tasks necessary or possible when creating it.

- Access and management of catalogs: This section allows the management of catalogs to be separated from the other tasks. As there are a good number of tasks related to this section, it seemed relevant to separate them from the others.

- Profile menu and gesture execution: In this section, we decided to put the tasks that are related to the profile menu. Consequently, we find the execution of a gesture and its interaction with the software as well as other tasks such as the possibility of disconnecting from the profile.

Chapter 4

Creation process and explanation of EMERITI

In this chapter, we will explain all the work we have done throughout the year. In section 4.1, you'll be able to understand the whole process, all the problems we encountered, and how we solved these problems.
Finally, we will explain the final version of EMERITI in section 4.2. We will talk about how to use EMERITI, how the interactions between Quantumleap and EMERITI are made as well as the architecture and we will finish by making a manual on the utilization of EMERITI.

4.1 Process of Work

In this section, we will describe all the steps we undertook to obtain our final project. We used the Agile method throughout our work. This method is an iterative approach to project management and software development that helps teams deliver work in small but consumable increments. Requirements, plans, and results are continuously evaluated. Indeed, we had weekly meetings with Professor Vanderdonckt to discuss the progress of the project. These meetings allowed us to move forward in our project since we did each time a review of the work done in the previous sprint and we define the objectives for the next sprint. That's why in this chapter, we are going to present all the sprints we have done throughout the year with their duration, the problems we encountered, and the way we solved them.

4.1.1 Discovery of the Tools

First of all, before starting any coding session, it was essential for us to learn about the different specificity of the tool we were going to use. Therefore, we analyzed in depth the different modules used in QuantumLeap, the different recognizers that exist, how it would be possible to perform a 2D drawing, and finally, how we should manipulate the different modules of QuantumLeap to perform our pipeline.

This step was quite long (almost 2 weeks) because of the analysis of QuantumLeap. Although it is readable, it took us some time to understand how the modules fit together.

The help of Arthur Sluÿters was then very useful for us to better understand all the subtilities.

4.1.2 Creation of a Canvas

After this analysis stage, we were able to start creating a draft of our application. To do so, we relied on several video sources allowing us to create a drawing app using React and the canvas API. We were thus able to obtain relatively quickly (3 days) an application allowing us to draw figures and integrate a means of obtaining the coordinates of the points traversed by each feature. Although our application works well, it is not yet connected to QuantumLeap and is still only a prototype.

4.1.3 Connection with QuantumLeap

After having succeeded in creating a convincing prototype, we wanted to test EMERITI by connecting it to QuantumLeap. This step was without a doubt the part that took us the longest (more than a month) and where we got stuck many times.

Problem with the importation of QuantumLeap

First of all, we had a problem with the import of QuantumLeap into our application. The reason is that our EMERITI developed with React did not allow to import QuantumLeap without making most of the features malfunction. To solve the problem, we had to change a big part of our code to extend React.Component. In addition to allowing to import QuantumLeap, React.Component allows defining components as classes or functions. Components defined by classes provide more functionality for the moment.

The solution to extend our project to React.Component was however found only at the very end after several meetings. Before that, we tried to recreate several prototypes in the hope that they would accept the QuantumLeap import. Unfortunately, these prototypes were in turn blocked due to similar errors or limitations in their code.

Creation of JSONs for the datasets

Another difficulty we had to face was the creation of a dataset and consequently, the creation of a JSON file related to the coordinates of the points of each gesture.

The problem in creating a JSON file is that it was necessary to create a functionality both to record the gestures in the QuantumLeap dataset and also to recognize the gestures using the recognizer. For this part, it was relatively easier since the main difficulty in this problem was the very strict format of the different JSON files. Below you can see the JSON format sent to both the dataset and the recognizer.

JSON sent to the dataset :


```
{"name":"example","subjet":"1","paths":[{"label":"point","strokes":[{"id
    ":0,"points":[{"x":174,"y":310,"t":1659470311441},{"x":174,"y":311,"t
    ":1659470311450},{"x":174,"y":312,"t":1659470311458},{"x":175,"y":313,"t
    ":1659470311465}]},{"id":1,"points":[{"x":257,"y":305,"t
    ":1659470312089},{"x":258,"y":310,"t":1659470312097},{"x":258,"y":311,"t
```

```
    ":1659470312105},{"x":260,"y":312,"t":1659470312114},{"x":261,"y":314,"t
    ":1659470312122},{"x":261,"y":315,"t":1659470312130},{"x":262,"y":315,"t
    ":1659470312138}]}]}]}
```


JSON sent to the recognizer :


```
{"name":"","subjet":"1","paths":{"point_lmc":{"label":"point_lmc","strokes
    ":[{"id":0,"points":[{"x":156,"y":324,"t":1659470088914},{"x":156,"y
    ":331,"t":1659470088927},{"x":156,"y":334,"t":1659470088931},{"x":156,"y
    ":337,"t":1659470088938},{"x":156,"y":342,"t":1659470088946},{"x":156,"y
    ":343,"t":1659470088954},{"x":157,"y":345,"t":1659470088962},{"x":157,"y
    ":346,"t":1659470088970}]},{"id":1,"points":[{"x":261,"y":316,"t
    ":1659470089533},{"x":261,"y":320,"t":1659470089540},{"x":260,"y":327,"t
    ":1659470089545},{"x":260,"y":331,"t":1659470089554},{"x":260,"y":334,"t
    ":1659470089562},{"x":260,"y":337,"t":1659470089570},{"x":261,"y":338,"t
    ":1659470089586}]}]}}}
```


Creation of a new pipeline to manage 2D gestures

Finally, the last problem we had to solve was the creation of a pipeline connecting EMERITI to QuantumLeap. Given the difficulty of understanding all the subtleties of the code, we spent several weeks trying to create it. And it was finally with the help of Arthur Sluÿters, who was able to clarify the few areas of the code that were still a bit unclear, such as the config.json file, the FrameProcessor constructor, and the functioning of some important modules.

After that, all we needed to do was to link EMERITI with QuantumLeap using quantumleapjs. The GestureHandler object from quantumleapjs allows us to connect to the QuantumLeap framework and calling functions based on events. Thanks to the meeting with Arthur, we were able to find a way to create our pipeline in only a few hours, fix the last errors in our application, and finish creating the JSON files. To verify that our pipeline was working well, we decided to manually add the different JSON files to the dataset to see the results. Following this test, we also had to modify some functionalities so that the dataset could train itself for the recognizer to recognize the gestures that were sent to it.

In order to better understand how our pipeline works, it is important to understand its global functioning. First of all, our application behaves in the same way as a "sensor". Indeed, the principle of a sensor is to send to QuantumLeap the information related to the gestures performed. Since we want to recognize 2D gestures, a drawing app seemed relevant to send these data.

After EMERITI has sent the data to the dataset, it is important to train it so that the recognizer can base itself on the recorded gestures to be able to compare with the drawn gesture. We decided to use the dynamic recognizer for the sake of ease of use. Indeed, following our study to choose the most relevant recognizer for our project (CFR Chapter 2.3), we determined that $P+ was the most efficient recognizer. Since it was already installed among the dynamic recognizers, we decided to use this module. Although $P+ does not need to have a way to determine the order of the points in time since it consists of a point cloud (CFR Chapter 2.3), it is nevertheless necessary to calculate the time needed for the recognizer to find the most similar gesture.

Finally, after the recognition of the gesture by the recognizer, the result is sent to our application via quantumleapjs.

Link between EMERITI and QuantumLeap

After solving the big installation problems, setting up our pipeline, and getting a better grasp of the pipeline features, we could spend our time improving our application. Therefore, the first important step was to send a gesture from EMERITI to QuantumLeap, so that it recognizes it correctly and sends the response back to our application. Despite some minor issues, we managed to get EMERITI working for the first time, which was the first real small victory in our project.

4.1.4 Realization of our Virtual Simulator

Once we got our application to receive messages from QuantumLeap correctly, we moved on to the simulator part. It took us a while to get the first prototype. Indeed, it took us 3 weeks to have an interaction between the drawing area and the virtual simulator.

Research about connected objects

Indeed, we wondered how to use the data received from QuantumLeap. First, we wanted to make our application interact with connected objects. This would have allowed us to see the effectiveness of EMERITI in a real situation. However, we realized that the work to be done was laborious. Indeed, after a week of research on connected objects, we realized that they often had a predefined application and that there was no API in Open Source that would have allowed us to interact with the objects. In addition, one of the goals of our project is that our application is multi-device. This means that it must be able to use several devices. Unfortunately, there is more than one protocol that allows using connected devices (Wi-Fi [63], Bluetooth [64], Zigbee [65], etc.). Having no knowledge in this field, we would have spent a lot of time understanding all the protocols and implementing all the interactions between EMERITI and all the different devices.

Moreover, the main goal of our thesis is mainly to realize an application allowing to manage hierarchical and congruent gestures, we decided to focus on the management of these gestures instead of spending a lot of time on the use of real connected objects.

Advantages of the virtual simulator

Even so, we needed to find a way to demonstrate that our job works. That's why we decided to make a virtual simulator after reading the article on "Address and Command" [60]. Indeed, having our virtual simulator allows us to reproduce a smart home virtually while saving time. It also allows using a large number of connected objects while keeping a certain simplicity of implementation.

Moreover, if we had had to buy all the connected devices we needed, it would have been a waste of money and the budget is limited, we would not have as many devices as in the final version of our application.

Another advantage is that we created our grammar so that we can interact with connected objects in various environments. For example, in the context of a smart home, it is necessary to be able to interact between the different devices located in different rooms. The simulator allowed us to respect this criterion.

This being the case, we started the realization of our simulator. Not being used to animations made in javascript, we started by creating a button allowing to swap 2 images. We had to try different alternatives to get the correct result. Our final solution was to play with the opacity. This allows us to play between the foreground and the background with overlapping images. Once we succeeded in creating this animation, we used the recognition feedback values to make a change in our simulator. This takes us 3 days. From this point on, we started to have a working simulator.

4.1.5 Creation of the Table of Gestures

At the same time, we created the "Table of gestures". This is a table that contains some data about all the gestures created before. Indeed, after having successfully linked EMERITI and QuantumLeap, we were able to save the gestures created in QuantumLeap but also their names and elements in EMERITI. The table contained at first a column with the name of the gesture asked by the user and the instruction. Indeed, at that time, we did not have all the elements of the final grammar. We had only the 2 macros "Turn all off" and "Turn all on". These allowed us to turn on or off all the devices in the house.

Later on, we added all the devices. However, we had not yet managed the environments. Therefore, the device was tied to the environment which means that we had for example "Turn on the lights in the kitchen" and not "Turn on", "The lights", and "In the kitchen".

4.1.6 Add a Timer for Combination of Gestures

Furthermore, it is important to know that at that time, gesture recognition was only performed on one gesture at a time. You could not choose to realize a sequence of gestures to an instruction. We had to create a single gesture for the command "Turn on the bathroom light". So we were managing non-hierarchical gestures. This is why we decided to create a timer.

This timer allows us to separate 2 gestures in a sequence of gestures. Indeed, if the user does not create a gesture during a certain time, the next stroke will be considered as a new gesture. This allows managing the problem with multistrokes gestures. Since in a multi-stroke gesture we can have several strokes, we could not simply consider that a gesture is created as soon as the user lifts his stylus, and stops his gesture. If we assume this hypothesis to be true, it means that we can only realize unistroke gesture sequences. Except that in the context of EMERITI, we also had to manage multistrokes gestures. This modification take us a week because we had to change the format of the JSON so that it correctly records the different gestures.
Moreover, as said before, the timer allows making a combination of gestures to perform an instruction. So we went from non-hierarchical gestures to hierarchical gestures to

perform basic instructions.

After that, the table of gestures will evolve by adding a column for each element of instruction. So we finally have 5 columns in this table (Name of Gesture -Actions - Devices - Environments - Parameters) as you can see in figure 4.1.

NAME OF GESTURE	ACTIONS	DEVICES	ENVIRONMENTS	PARAMETERS
TURN ON TV	Turn On	Television	-	-
TURN OFF TV	Turn Off	Television	-	-
TURN ON COMPUTER WITH VOLUME ON 5	Turn On	Computer	Office	Volume, 5

Figure 4.1: Table of gestures

Addition of other drop-down lists

Then, as said before, we made sure that our gesture table could accommodate the 4 elements of our grammar as you can see in figure 4.3. However, in the beginning, we could just choose with the help of a drop-down list of the action related to the environment or a macro.

That's why we decided to make a drop-down list for each element of the grammar. We took the opportunity to add all the possibilities. We had to be able to interact with any device in any room regardless of the setting. Once all these examples were created, we made sure that animation was done for each device. We could then actually perform basic instructions with hierarchical gestures that could modify our virtual simulator.

Creation of dynamic lists

Then, we made sure that these lists are dynamic. Dynamic lists are lists that adapt themselves according to the choices of the users. Let's take an example. If the user chooses the action "Pause", he should not be able to choose the washing machine. If he chooses the television, let's imagine that there is only one in the living room, he should not be able to choose other rooms. Nor should he be able to choose any other setting. This is why we made a dynamic list. After the first realization of the dynamic list, we added other conditions or elements so we modified the dynamic of all lists two times. This is why we realize this step in 2 days in total.

4.1.7 Steps to Realize All Buttons

At the same time, we also created several buttons to allow us to perform certain actions. These buttons take us some time (more than 3 weeks) because we had to manipulate the code of QuantumLeap, quantumleapjs, and EMERITI.

We had a button for "Download". This one allowed us to download the result of the gestures under the JSON format. It is important to know that to make the recognition work properly, we had to introduce it ourselves in the code in the datasets folder. Indeed, QuantumLeap did not offer an automatic way to create datasets. However, we wanted to allow users to use EMERITI without having to touch the code. We wanted to create a

"Black Box".
That's why we decided to automate the creation of Datasets. So we changed the name of the button to "Record". We created a new folder "Basic2D" in the dynamic Datasets module. In this folder, we create a folder for each gesture created in the application thanks to the "Drawing Area".
The name of the folder is determined according to the name of the assigned instruction i.e. "Turn on, A, name of the gesture, etc.". This folder will contain all the JSON collected. These JSON will allow training the recognizer to improve the chances of detecting the right gesture. From that moment, we could use EMERITI without having to make changes in QuantumLeap.

Another button was "Recognize". This one allows to recognize the gesture made in the "Drawing Area" and to show the result for the first time after modifying the devices in the smart home. The gesture cannot be recognized if it has not been recorded before.

The "Clear" button deletes the data collected for the existing gestures in the Drawing Area and resets it in case the user makes a mistake when making a gesture.

Then we had the "Clear Dataset" and "Clear Everythink" buttons. These allow you to delete all the existing gestures. The only difference is that "Clear Dataset" only deletes the data in the QuantumLeap dataset. Local storage is not affected by this button. The local storage allows keeping in memory in the application all the names of the created gestures. This allows checking if the gesture that is recognized corresponds to a created gesture. The problem with this one is that the deleted gestures still existed in the table of gestures in the application but could not be recognized since they no longer existed in the dataset. That's why in the end, we deleted the button and we created the "Clear Every Think" button to delete the data in the Dataset and the local storage. We then renamed it "ClearDataset" for more logic.

4.1.8 Improvement of the Visual

Overlay of Images

Once we managed to create all these buttons, we continued the development of our smart home. We found a smart home image. We created additional images using editing software to allow us to create one image per device and have multiple overlays. You can find an example in figure 4.2. This allowed us freedom in the interaction of the connected objects.

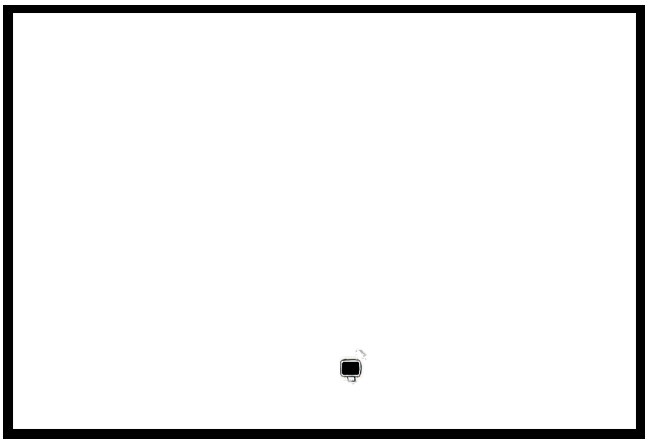

Figure 4.2: An example of an image of one device

The creation of all the images did not take so much time but the implementation in the code to animate the simulator with the result of the recognizer take us 4 days because we had to create a correct flow in the code. Thus, we had all the cards in hand to succeed in achieving our goal.

Reorganization of the Visual

Afterward, following a meeting with Prof. Vanderdonckt, we had to make several visual modifications to the application. So we increased the image of the smart home to allow to see the image changes more clearly. We also had to define the different zones of EMERITI.

Indeed, the different components were scattered and were not put optimally. That's why we added a title to the drawing area and the smart home.

We also rearranged the edit field and the different lists so that the user respects a certain order. So we put first the area in which the user writes the name of his gesture, then the list of actions followed by those of devices, environments, and finally the list of parameters. We also put the record button. Then, we gathered all the components that concern the Gesture Table. We have made a "Table of gestures" area with a title and a table of gestures. This area can be very large as we create gestures, so we decided to also create a button to hide or show this area if needed.

Finally, we moved the Drawing area. We had to play with the coordinates sent by this one since when we move the Drawing area, the line can shift according to the direction the gesture takes and it may not be at the place where we click with our mouse. You can see all these modifications in figure 4.3.

On the other hand, we could not yet handle numbers in the parameters. That's why we also made a hidden field that appears only when you choose a parameter that needs a number.

4.1.9 Start the Implementation of Common Instructions

Thanks to all these changes, our application has become more pleasant. However, we could not manage all the instructions. Indeed, we could only manage the basic instructions. That's why we changed the drop-down lists to allow multiple choices in them as you can see in figure 4.3. Until now, it was only possible to choose one item per drop-down list. For example, we could only have "Turn on the television in the kitchen in 5 minutes". In addition, we were forced to choose one item for each drop-down list. Therefore, we made it so that we could choose 0 or more items in each list. This is why in 2days, we were able to realize common instructions.

4.1.10 Implementation of Macro Instructions

Afterward, we understood that the command macros should not be actions anymore. Indeed, as mentioned in section 3.1.8, there could be errors when the user made permutations in the instructions. Therefore, we added a drop-down list named macro-commands which includes all the previous creations. With this list, you can choose any instruction, and thanks to the creation of the add instruction button, you can combine 2 or more instructions. The "add instruction" button acts like an "and".

In the beginning, it was only possible to create composed macro instructions via previously created basic macro instructions. To do this, you had to choose the instructions and then press the "create macro instruction" button. Unfortunately, this limited the macro instruction system. That's why we made it possible to use the instructions from the drop-down lists to create a composed/macro instruction.

However, it was either one or the other, you couldn't make a macro instruction of a composed instruction including a macro of instruction and another one that was an instruction created with the drop-down list.

That's why, in the end, we decided to make sure that we could create a composed instruction from the drop-down lists and the macro instructions already existing. We also decided to delete the button "Create macro instructions" to use only the record button to make it possible. This step was very difficult since we had to manage all the possibilities when pressing the record button.
Indeed, we had to know if it was an element, a composed element, a basic/common instruction, or a composed instruction. In addition, in the case of composed instructions, we had to know if the instructions present in the composed instruction were macros or instructions in the instructions composing the list. Knowing that the composed instructions cannot contain an element or a composed element. That's why we took 5 more days to create the final version of the record button.

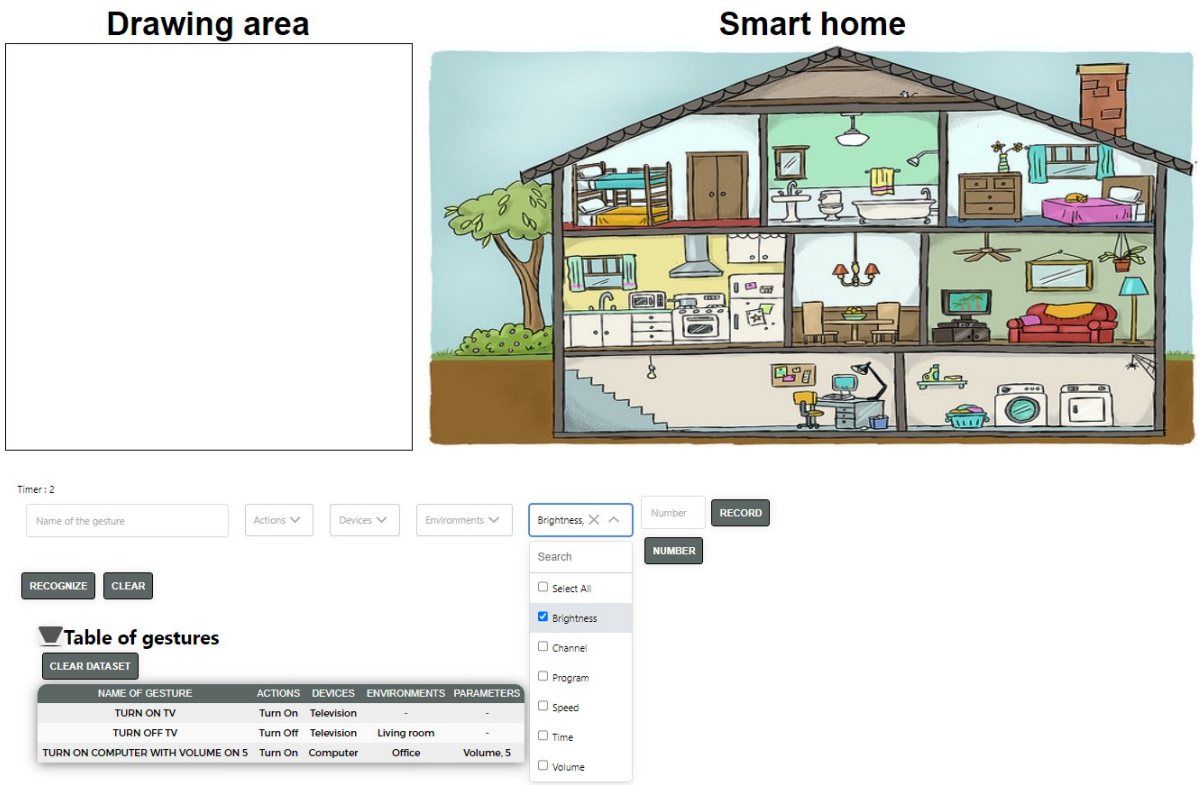

Figure 4.3: Screenshot of the application

4.2 Final Version

4.2.1 Sequence Diagram

In this section, we will explain the graphical representation of the interactions between EMERITI and QuantumLeap via quantumleapjs in chronological order.

The Sequence Diagram of Record

As you can see in figure 4.4, there are several steps to achieve the record of a gesture.

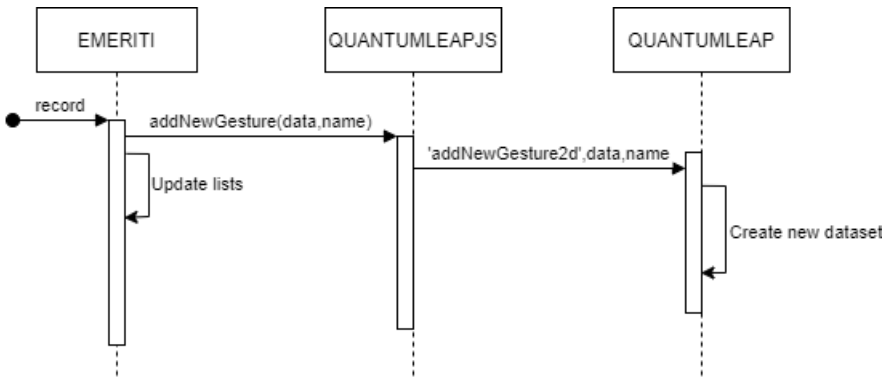

Figure 4.4: The sequence diagram of the process of recording

First, the user presses the record button in our EMERITI environment. Several scenarios are possible at this time. The user could realize a basic instruction, a composed

instruction with drop-down lists and the "add instruction" button, a composed list with existing macro instructions, or a composed instruction with macro instructions and drop-down lists. You have to know that all the instructions inside a composed instruction must have an action and a device at least even if it's a macro instruction.

Then, in all cases, EMERITI will send a request to QuantumLeapJS containing the name of the gesture that the user has chosen and the data collected in the drawing area. It will also update the lists of the localStorage. There are 2 lists for instructions with drop-down lists. The first one is checkList which contains all the names recorded and checklistAssign which contains an Object. This is the list that we show in the table of gestures. This one will know which actions, devices, environment, and/or parameters correspond to the gesture name.
For example, the user has decided to create a gesture that he names "Turn on" when the user asks to recognize this gesture, EMERITI must know that it must be turned on. We can find all the gestures and their correspondence in the table of gestures.

If the instruction to record is a composed instruction, then we update 2 other lists. CheckMacroList and checkMacroListAssign realize the same thing as checkList and checkList assign but for composed instructions with the table of macro/composed instructions.

After sending a request to quantumleapjs, it will send a message to the QuantumLeap framework. This message contains

- 'addNewGesture2d': which lets QuantumLeap know that a new gesture should be created.
- the data of the gesture allows creating a JSON that is use to train the recognizer.
- the name of the gesture that will allow creating a new folder in the dataset that can store the created JSON.

With all these data, QuantumLeap will be able to create a new dataset and recognize the gesture.

The Sequence Diagram of Recognize

Once our gesture has been correctly registered. The user can use it to recognize it to execute the command linked to the gesture.
This recognition starts when we click on the recognize button. At this moment, EMERITI sends a message registerGestures with a type and a name. quantumleapjs will transmit this call with the message "'operation', data". Once arrived at QuantumLeap, this one will update the set of gestures that can be recognized by the recognizer. After some thought, we found a possible improvement in this call. Indeed, if we called this method at the time of record and not at the time of the recognize function, we would avoid unnecessary calls.
Then, EMERITI sends the recorded data via the sendGesture function to quantumleapjs. The latter will then send to QuantumLeap the message 'recognized2d' with the data. Thanks to this, the framework will be able to recognize the gesture and return the name

of the recognized gesture to its API which will send it to EMERITI. This will allow it to look in the list of gestures or composed instructions.
If one of these lists contains the recognized name, then EMERITI will execute the command to modify the Smart Home. You can see the process in figure 4.5.

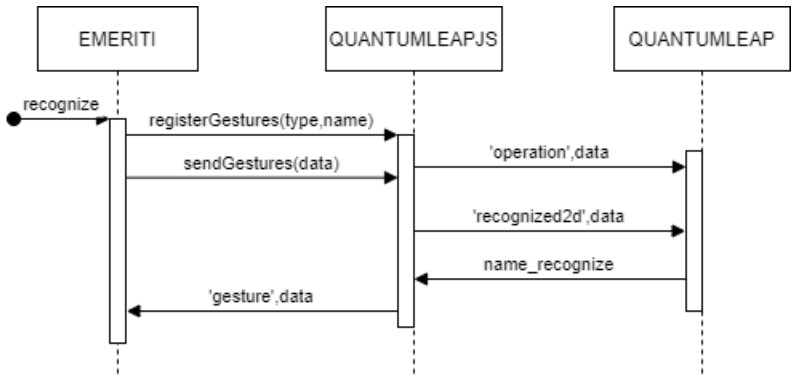

Figure 4.5: The sequence diagram of the process of recognizing

The Sequence Diagram of Delete

If the user is no longer happy with a gesture and wishes to delete it, he will have to press the "Delete" buttons in the list of gestures in the good row. Depending on whether the name is in the table of gestures or the table of composed instructions, the lists EMERITI has modified will be checkList and checkListAssign or checkMacroList and checkMacroListAssign.
Then, once the deleteGesture message with the gesture data is received by QuantumLeap, it will delete the correct folder in the dataset so that it can no longer be recognized 4.6.

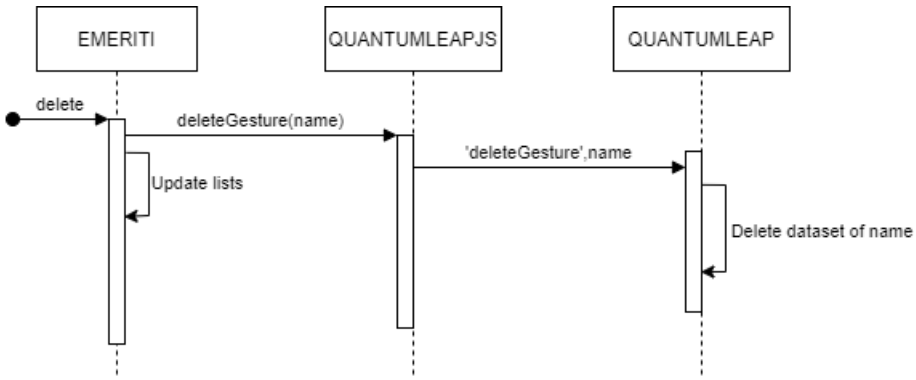

Figure 4.6: The sequence diagram of the process of delete

The Sequence Diagram of ClearDataset

The purpose of the clearDataset button is to delete all the saved gestures. To do this we will do as shown in figure 4.7. EMERITI will send the clearDataset request to quantumleapjs which will transmit it to QuantumLeap. This one will delete all the existing datasets.

In addition to sending a request to quantumleapjs, EMERITI will also reinitialize its localStorage.

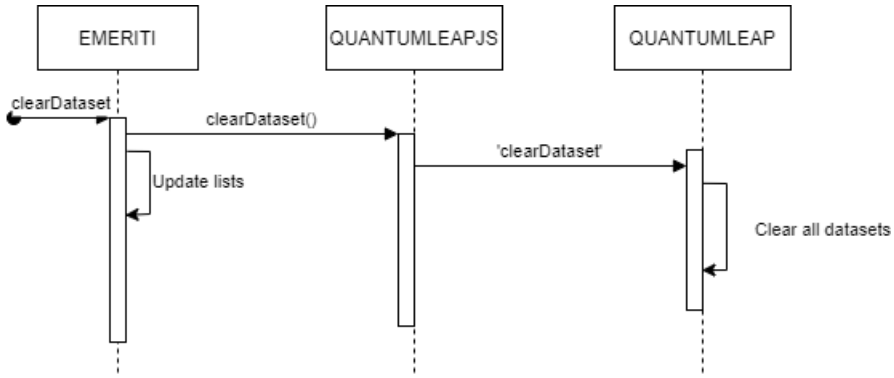

Figure 4.7: The sequence diagram of the process of clearing all the datasets

4.2.2 Architecture

In this section, we will discuss the architecture of our work. Indeed, we had to modify the QuantumLeap pipeline as well as its API to realize the implementation of our work.

To allow users to be more autonomous, we have made sure that EMERITI can create datasets themselves to train the recognizer. This makes it possible to not have to modify the code. Moreover, it also makes it possible to use gesture recognition even for those who do not have the necessary implementation knowledge.
Then, if we allow users to create their dataset and thus their gesture, we must also allow them to delete it in case they don't want to use it anymore or if they made a mistake. These 2 interactions between EMERITI and QuantumLeap are represented by the arrow number 1 and 2 in figure 4.8. EMERITI will communicate with the QuantumLeap API. Then the information will be sent to the server (arrow 1). After that, according to the message received from EMERITI, the server will transmit the request to add a gesture, delete a gesture, or delete all the gestures to the "dynamic dataset writer" (arrow 2). This one will thus modify the datasets according to the received message.

Then, let's imagine that the user does not remember the gesture he made for instruction. He can display it via the display button. This functionality does not exist in the old version of QuantumLeap, so we also had to create a path in QuantumLeap to realize it. In the beginning, it follows the same path as creating or deleting a gesture. However, once arrived at the server this one will send the message to the new module "dynamic dataset reader" (arrow 3). This one will extract the data of the required gesture and send it to the server (arrow 4) which will transmit it to EMERITI via quantumleapjs (arrow 9).

Once a gesture has been created, EMERITI can also ask QuantumLeap to recognize it. Thanks to the API, EMERITI will send a request to quantumleapjs. The API will send a message to the server. The server will first send a message to the "Dynamic dataset loader" allowing it to update the set of gestures used for the recognizer and the new templates added in the gestures already known (arrow 5). If this step is not done, the recognizer list is not updated, and therefore it cannot recognize the new gestures. As explained in section 4.2.1, there is a better way to realize this step. Instead of realizing this in the recognition process, we could have done it in the record part.
Then the dynamic dataset loader will also extract the required dataset and send it to the dynamic recognizer (arrow 6). Once the recognizer has received the dataset

to train it, the server will ask it to recognize the gesture (arrow 7). Once the recognizer has recognized the gesture, it sends the result to the server (arrow 8) which will transmit it to the API (arrow 9) and thus to EMERITI. EMERITI will make changes to images on the virtual simulator. All this process can be seen with the help of the figure 4.8.

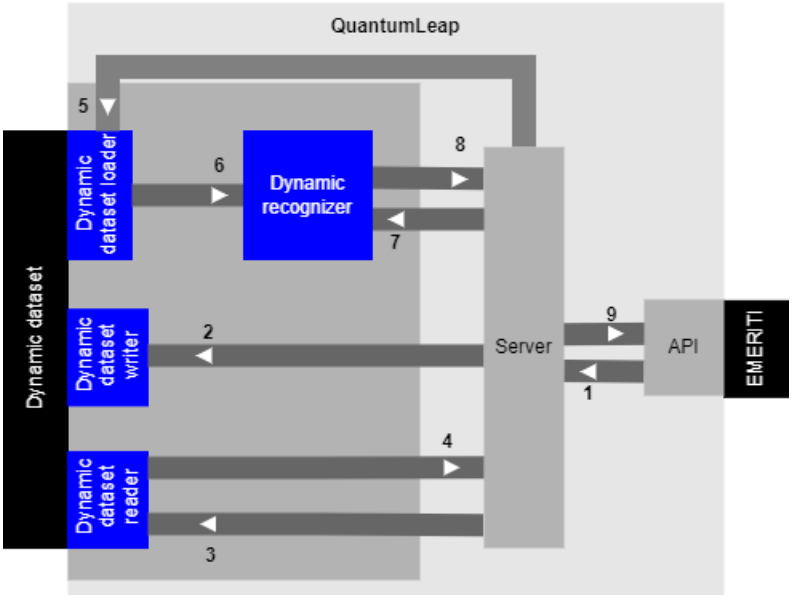

Figure 4.8: QuantumLeap configuration for EMERITI
Source: Adapted on [66]

4.2.3 How to run EMERITI

To launch the EMERITI application, it is necessary to install 2 repositories (QuantumLeap and EMERITI).

After the installation of QuantumLeap, you will have to install Node.js.
Once this is done, you will have to install the dependencies in the backend folder.

```
$ cd backend
$ npm install
```

Then start the dependencies also in the backend folder to launch the framework.

```
$ npm start
```

Following the installation of QuantumLeap, you will also have to install EMERITI. Like QuantumLeap, you must have installed Node.js beforehand to be able to launch the dependencies. Unlike QuantumLeap, you can install and start the dependencies directly from the main EMERITI folder.

To make the application work properly, it is necessary to launch both QuantumLeap and EMERITI via two different terminals. Once the installation is finished, it will not be necessary to reinstall the dependencies each time you use them. You will only need to start the frameworks via the "npm start" command.

Once these steps are completed, the application should automatically launch on localhost.

4.2.4 Code Structure

To better understand how our application works and how it is articulated, we think that an explanation of its structure is relevant. To do this, we will explain the different important files in both the EMERITI and QuantumLeap repositories.

EMERITI's code structure

In the EMERITI project, we find two very important folders which are "quantumleapjs" and "src" (Figure 4.9).

Indeed, in the first one (quantumleapjs), we can find a file named "index.js" which allows connecting to the QuantumLeap framework and call functions based on events from QuantumLeap.

In the second folder (src), we will also find important files. We will find "App.js" which is the main file of our application and where there are all the features of the application. Then we have "index.css" which will tell the browser how to display the existing HTML from "App.js". Next to that, we also have an images folder that contains all the images that we overlay in the application to display and hide the different connected devices.

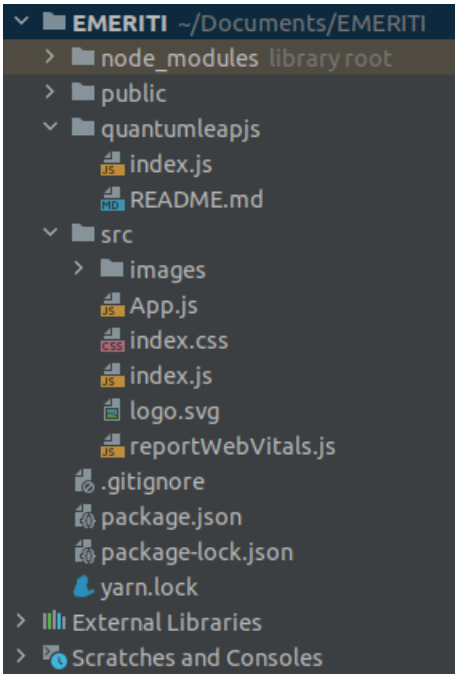

Figure 4.9: EMERITI's code structure

QuantumLeap's code structure

Concerning QuantumLeap, we will focus on the parts that are important to us for our application.

First of all, we can a file named "config.json" (`QuantumLeap/backend/src/config/quantumleap/config.json`, Figure 4.10). This is a config file allowing QuantumLeap

to know the parameters selected by the user, such as the recognizer used, the dataset to be trained, etc.

Next, we have probably the most important file in our case, namely "quantumleap.js" (`QuantumLeap/backend/framework/quantumleap.js`, Figure 4.10). This file contains our pipeline allowing to interact with EMERITI. Indeed, it is in this file that are called all the modules that we use as well as the dataset with which we interact.

We can also find the different recognizers installed in QuantumLeap and more particularly the folder "pdollarplus" which is the recognizer we use by default (`QuantumLeap/backend/src/implementation/recognizers/dynamic/pdollarplus/pdollarplus`, Figure 4.10). The different files that make it up allow the recognizer to work. This file does not require any modification to work.

Finally, the last folder we are interested in is "basic2D" (`QuantumLeap/backend/src/datasets/dynamic/basic2D`, Figure 4.11) which is our dataset. This folder contains all the gestures created by the user and at each creation of a new gesture, a folder is created as well as a JSON file with the name given by the user. If the gesture already existed, a new JSON file would be added to the folder relative to the name given by the user.

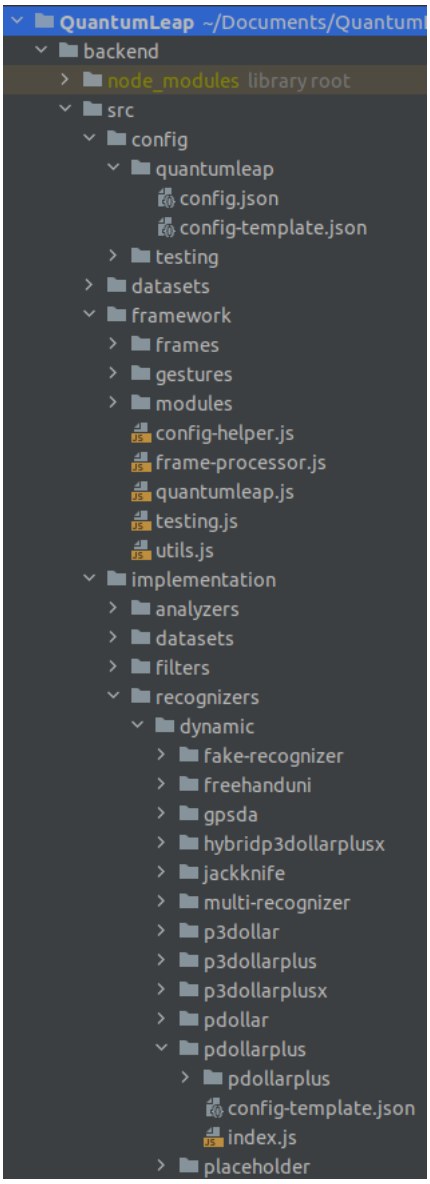

Figure 4.10: QuantumLeap's code structure

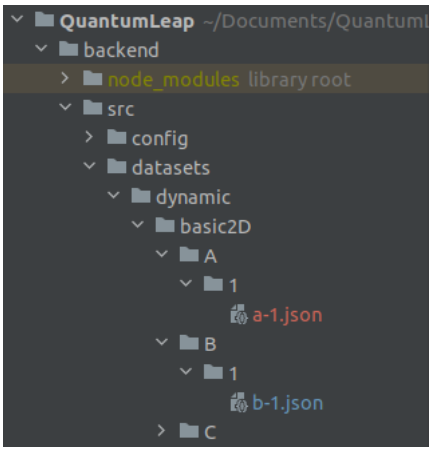

Figure 4.11: QuantumLeap's code structure (dataset)

4.2.5 User Manual

As you can see in figure 4.12 when you enter for the first time in the EMERITI interface, no gestures have been recorded yet, all the devices are turned off, the two tables are empty and therefore the recognition and the display function are not yet feasible. This is why the user must, at first, record some gestures.

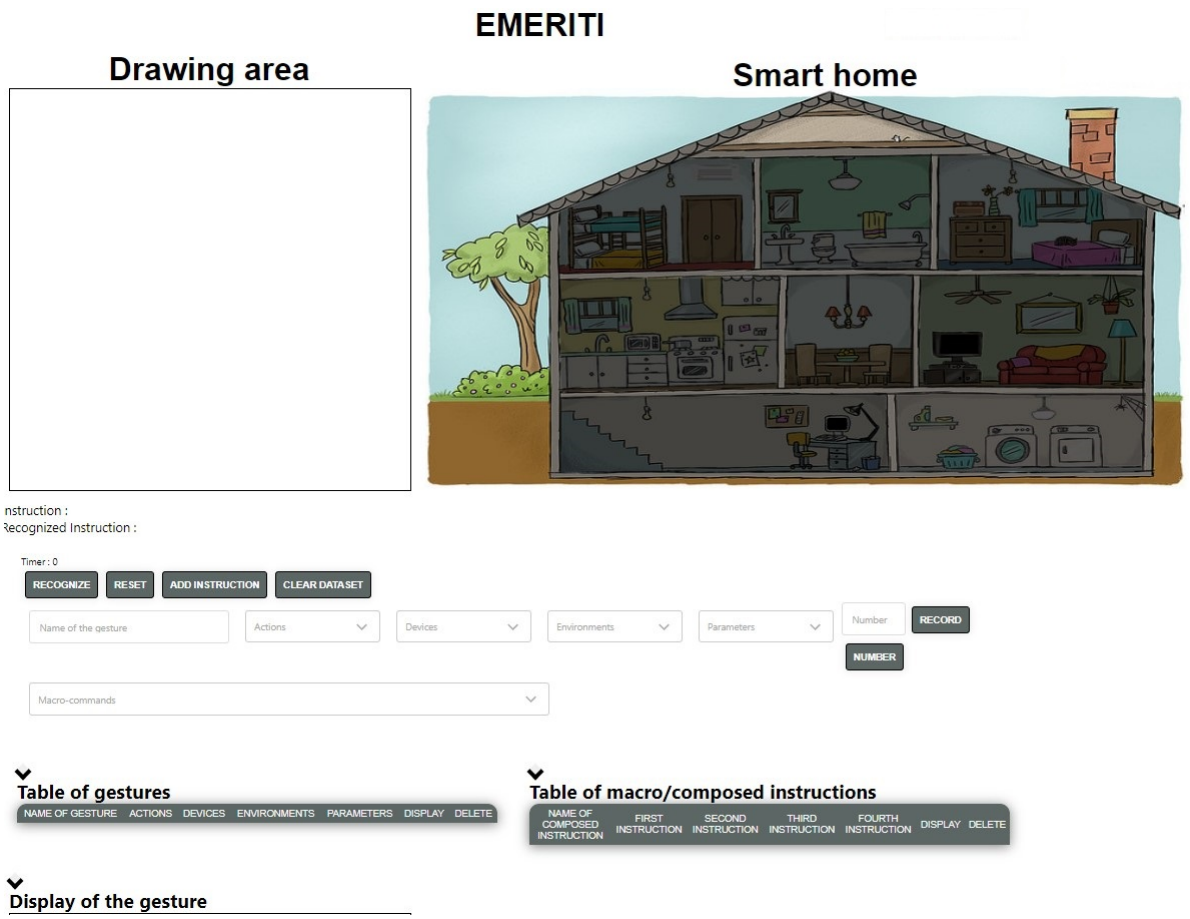

Figure 4.12: The environment EMERITI.

The only possibility at the beginning is to record a gesture thanks to the drop-down lists. As previously explained, they are dynamic and each of them offers all possible functionalities. All you have to do is to choose which one or ones you are interested in, indicate a name for the desired instruction, draw the gestures you have chosen for this instruction and finally click on the "record" button to record the gesture. For example, in figure 4.13, we can see that the user realized a triangle as a gesture with the name "first". Then, as you can see in the drop-down lists or at the level of "Instruction" the user has chosen to create this gesture for "Turn On". He will just have to press the record button to create his gesture.

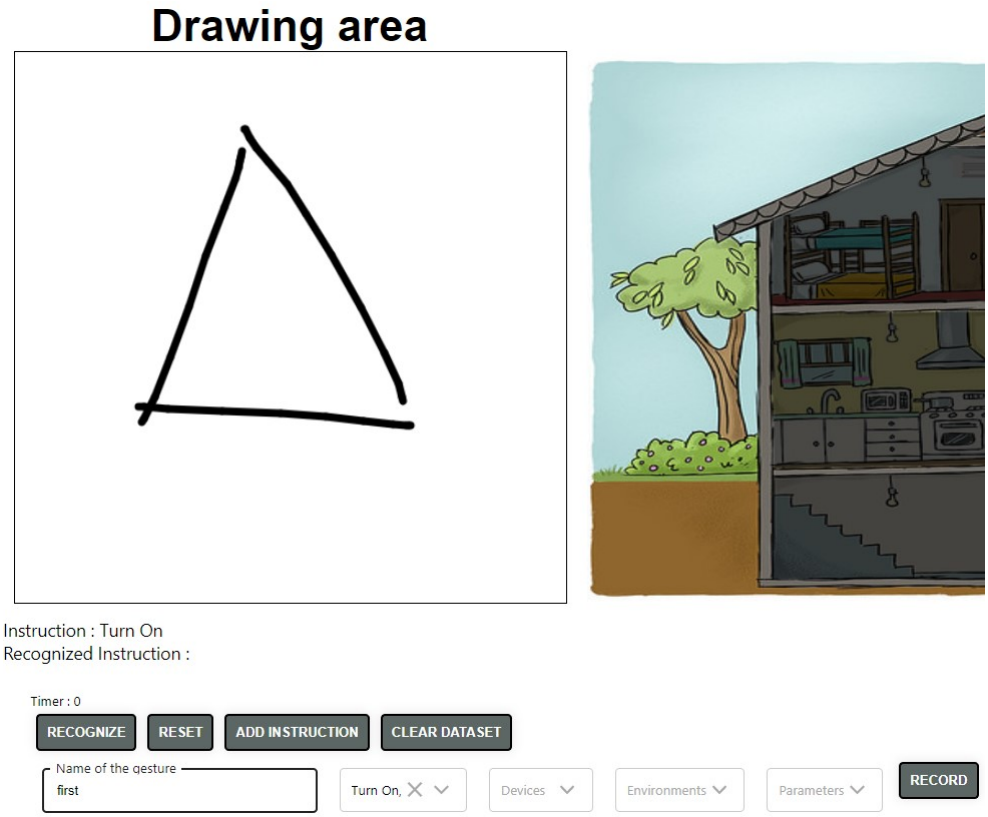

Figure 4.13: A basic record with the drop-down list.

Once the user has created gestures to, at least, be able to execute an instruction, he just has to draw the desired gestures in the "Drawing area". After that, he has to wait that the timer ends between each gesture. When he finished the realization of all the gestures, he can press the "Recognize" button and operate a change on the smart home. For example, if a user has decided to draw a gesture for "Turn on" and another one for "computer", and if the gesture recognition is correct, then the computer will be turned on.

Now that the user knows how to create gestures, he also can create macro instructions or composed instructions.

Several choices are available to the user at this time. He can choose to create a composed instruction thanks to the list of already existing gestures. In this case, he just has to choose all the ones he wants to select from the list of macro commands and press the "record" button. Note that all the selected macros must contain an action and a device. Moreover, it is necessary to take into account the order in which they are taken so that the execution order is correct. You can see in figure 4.14 the process described above.

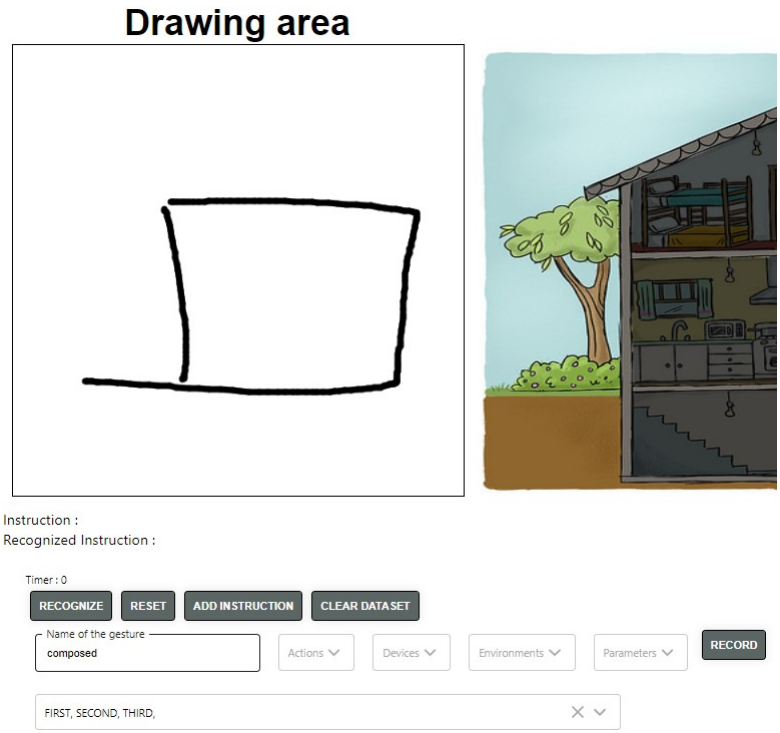

Figure 4.14: A macro record with the drop-down list.

He can also use the "add instruction" button. To do this, you still need to have a name and a drawing for the gesture. Then the user chooses from the drop-down lists the first execution he wants to perform, presses the "add instruction" button, and can choose his second instruction, etc. You can see this process in figure 4.15. It is important to know that for visual reasons, we have decided that we should not have a composed instruction with more than 4 other instructions.

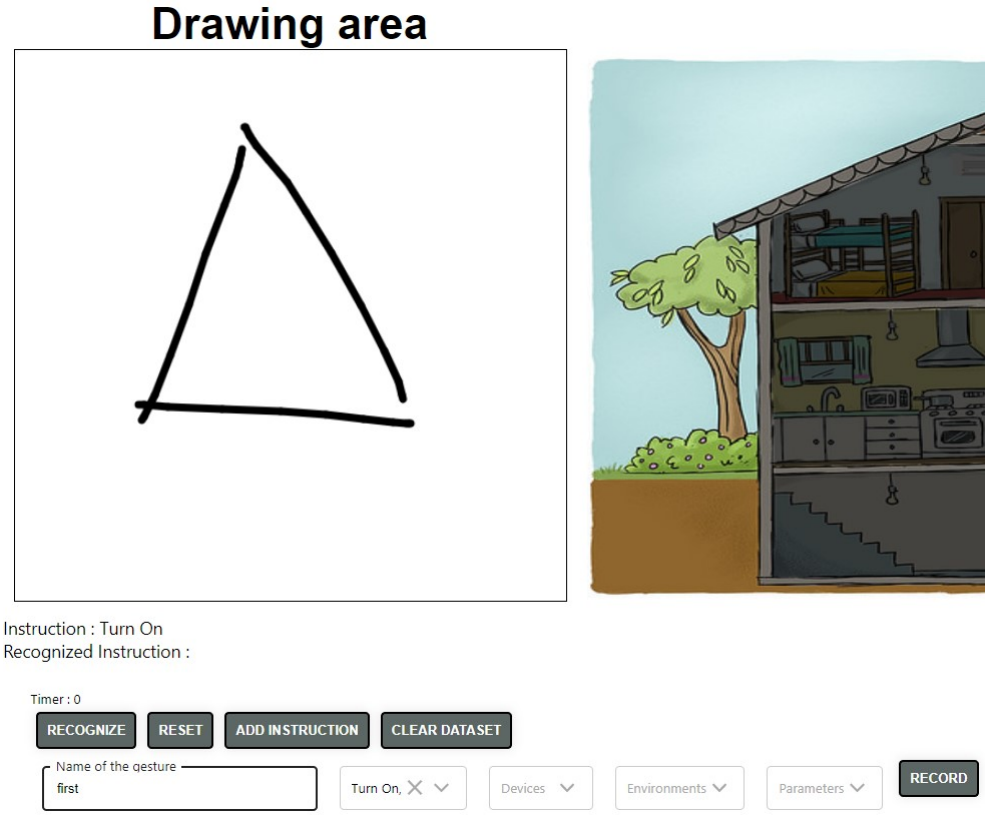

Figure 4.15: A composed record with the drop-down list and the button "add instruction".

At this point, the user can create a simple instruction, to create a macro or composed instruction via the drop-down lists as well as the drop-down list of macros. However, the user can also decide to combine the instructions in the drop-down lists with already existing instructions as you can see in figure 4.16. This offers greater fluidity and freedom.

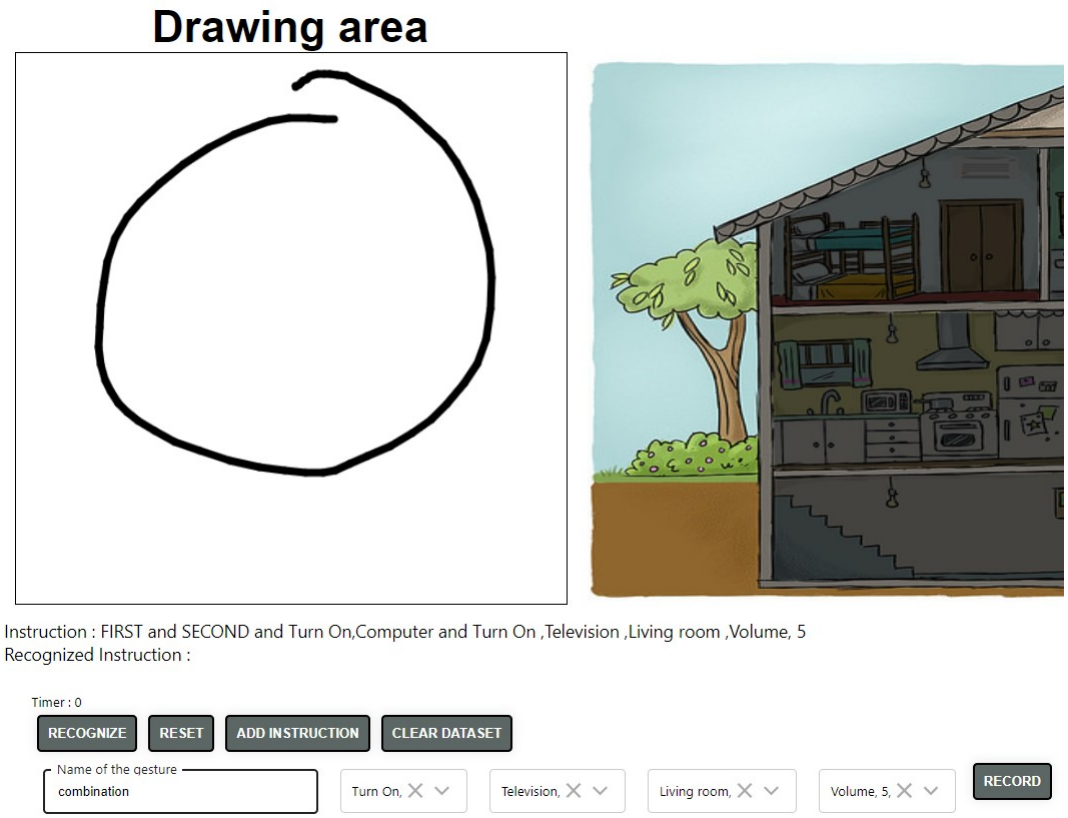

Figure 4.16: A composed record with the drop-down list, the macro-commands list, and the button "add instruction".

Once the gestures are created, we can use the 2 tables "Table of gestures" and "Table of macro/composed instructions" to know all the existing gestures. In addition to that, as you can see in figure 4.17, there is the possibility to delete a gesture or display it. If we choose the display option, EMERITI will give the user the possibility to see the gesture he had chosen for this instruction. You just have to go to the "Display of the gesture" part to see the gesture drawing. You can see an example of a display in figure 4.18.

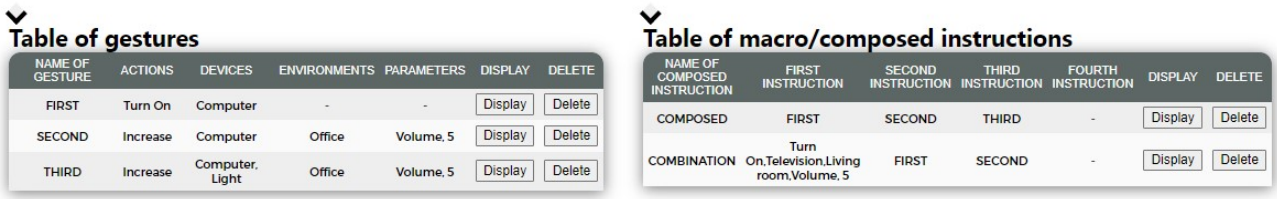

Table of gestures

NAME OF GESTURE	ACTIONS	DEVICES	ENVIRONMENTS	PARAMETERS	DISPLAY	DELETE
FIRST	Turn On	Computer	-	-	Display	Delete
SECOND	Increase	Computer	Office	Volume, 5	Display	Delete
THIRD	Increase	Computer, Light	Office	Volume, 5	Display	Delete

Table of macro/composed instructions

NAME OF COMPOSED INSTRUCTION	FIRST INSTRUCTION	SECOND INSTRUCTION	THIRD INSTRUCTION	FOURTH INSTRUCTION	DISPLAY	DELETE
COMPOSED	FIRST	SECOND	THIRD	-	Display	Delete
COMBINATION	Turn On,Television,Living room,Volume, 5	FIRST	SECOND	-	Display	Delete

Figure 4.17: The table of gestures and the table of macro/composed instruction.

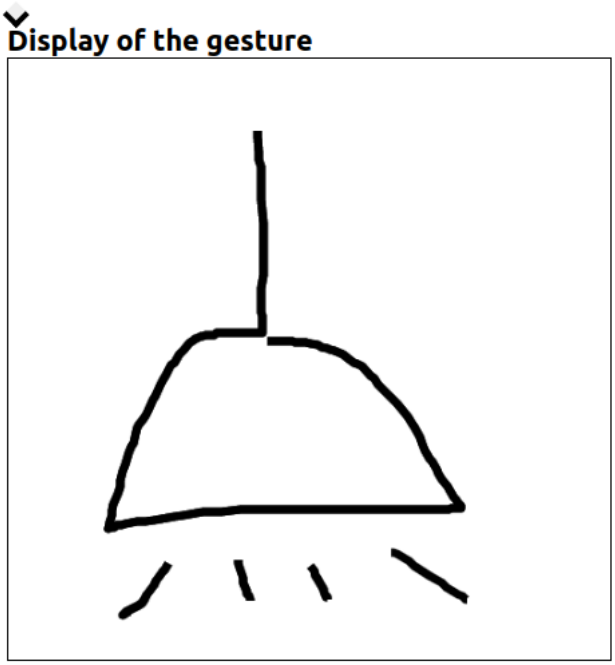

Figure 4.18: A display of a gesture.

Now that you have read this part, you know how to create gestures with the help of the drop-down lists, the macro-commands drop-down list, the use of the "add instruction", "record", and the "recognize" button. You have all the expertise you need to exploit EMERITI's environment to the maximum.

Chapter 5

Final Conclusion

5.1 Summary of the Contributions

After completing this work, we can provide an overview of the various contributions brought throughout this manuscript.

In Chapter 1, we analyzed the context of the problem: how to improve the various shortcomings of keyboard shortcuts by exploring the concept of combining 2D gestures. Following that, we described the main objective of our thesis which is to create an application allowing to realize shortcuts employing macro-commands or a combination of gestures that are congruent and hierarchic in order to interact with connected devices.

In Chapter 2, we performed a targeted literature review allowing us to obtain multiple pieces of information for the rest of our thesis.

Concerning the article based on the use of strokes as command shortcuts, we could highlight that stroke shortcuts can be as effective or more advantageous than keyboard shortcuts. However, with the same amount of practice, participants tended to remember more gestures (about 3 times more), to remember gestures more easily and quickly, and finally, to make fewer errors with stroke shortcuts than with keyboard shortcuts.
Concerning the article based on 3D gesture combinations, we could notice that users' preferences go toward gesture combinations when the single gesture counterparts are abstract. And that abstract single gestures have a negative impact on memorability.

In the article on congruent and hierarchical gestures, we have seen that participants found agreement more easily and frequently when the gestures were congruent and hierarchical than with non-congruent and non-hierarchical gestures.

In the section devoted to different recognizers, we could highlight several powerful 2D recognizers such as $P+, $Q+, and μV. Following the different articles based on this topic, we have chosen $P+ as the main recognizer of our application given the very efficient recognition rates in the context of our application.

We were also able to explain in detail how QuantumLeap works and how to use it, which is the framework on which we based our application.

Finally, regarding the article based on "Address and Command", this study proved that using separate gestures allows us to remember more commands with a small number of gestures. This means that our application allows remembering more commands since it has more categories of gestures namely Action, Device, Parameters, and Environment.

In Chapter 3, we have explained all the work we have done in order to answer the problematic of this thesis.

First of all, we talked about the grammar we created to allow us to have syntactically correct language. We also gave some examples to be able to understand better, the steps we realized to arrive at our final result as well as the advantages of our grammar compared to the one of 'Address and Command'.

After that, we made an exploratory analysis to imagine what our application could be in the future.

In Chapter 4, we explained the whole process, all the problems we encountered, and how we solved these problems.

Then, we explained in detail our application. Indeed, in order to realize our research work, we had to create an application involving a smart environment called EMERITI (EnvironMent gEstuRe InTo Instruction). In order to make our application as permissive as possible, we decided to allow users to create their shortcuts by selecting the command they want to execute and drawing the associated gesture. Among the particularities of our application, it is also interesting to note that it is possible to merge several previously created gestures into one macro instruction. This feature increases the permissiveness of the application, as each user will have the possibility to create their own set of personalized shortcuts and combine them.

Finally, at the end of this chapter, we will explain the final version of EMERITI and we will explain how to use it, how the interactions between Quantumleap and EMERITI are made as well as the architecture.

5.2 Benefits and Limitations

Our application allows putting forward many advantages. Indeed, following our work, we could notice that the use of 2D gesture combination was a very effective technique allowing users to create a limited number of gestures. The fact that they can decide the gesture also allows users to have a mnemonic link, thus improving the memorization of the gestures. EMERITI being very permissive, it offers the user a great deal of freedom regarding the use and customization of shortcuts.

Regarding the connection with QuantumLeap, we managed to create a personal pipeline allowing us to easily integrate our project with the framework. Due to the way we created our pipeline in QuantumLeap and our project, we can easily modulate it to integrate new sensors such as LeapMotion

Our application is also very simple to use and can be easily transposed to different

devices such as a smartphone or a tablet.

However, concerning the modularity (degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components) of our application, we encounter a limit. Indeed, in our application, we have created a controlled environment. This one allows us to visualize how the use of our application influences the environment. Unfortunately, this environment is not very modular. The slightest change of elements in the environment requires a relatively important modification in the code.

5.3 Future Work

Our application has many qualities that can be used in future research. Indeed, it would be quite possible to extend EMERITI to other environments and take advantage of it to make the application more modular.

EMERITI is an application implementing a smart environment. A relevant objective that could be considered would be to integrate real connected objects into the application. This could show concretely the usefulness of our memory.

Among the possible improvements, an improvement of the application to make it more user-friendly could be interesting. Indeed, a tutorial to show the user the different functionalities of the application could be useful.

We could also modify the code to have a similar structure to QuantumLeap. Indeed, the creation of modules to separate the different parts of our code would make it more understandable for developers wishing to modify EMERITI.

We could also imagine extending the application to other devices such as smartphones or tablets. Given that it is enough to make signs on a screen, it would be even more relevant.

Among other future works, it would also be possible to change the type of sensor to another one (i.e. a 3D sensor) or to merge 2D and 3D. This modification would not require much change since the most complex task in this idea would be to retrieve the gesture data and send it to QuantumLeap. Overall, our code would require almost no change to integrate or change a new sensor.

Finally, the future work that we consider to be the most interesting to develop, would be an adaptation of our project in order to help disabled and/or visually impaired people. Indeed, EMERITI has the advantage of allowing users to interact with objects at a distance by performing one or more gestures on a graphical interface. This would improve their comfort in life in many aspects. It would be quite possible to modify the type of environment to assist people with disabilities in their lives. Our application has a recognizer ($P+) create at the beginning to help the recognition of gestures made by visually impaired people (with a recognition rate higher than 98%).

List of Figures

2.1 Time, Recall, and Error by Shortcut x Frequency. 9
2.2 Time and Recall performance according to the number of times a command has been seen. 9
2.3 Percentage of use of each technique in retest on the second day (by the participant) 10
2.4 Gesture set while combining gestures. Gestures are depicted with rows showing the signifiers and columns showing the operators. 12
2.5 Gestures set with single gestures. We distinguish between abstract (light orange background) and non-abstract (dark purple background) gestures. Note that the two-handed gesture used for "Display On/Off" is a single gesture-command mapping. 12
2.6 Some examples of unistrokes gesture. 15
2.7 Some examples of multistrokes gesture. 15
2.8 Stroke gestures can be articulated in multiple ways owed to the order, number, and direction in which the strokes can be drawn (Articulation-invariance). Stroke gestures can be performed at different rotations, scales, and/or translations (RST-invariance); hence the need for an ARST-invariant multistroke recognizer. 17
2.9 Decision table for UI prototypers, designers, and researchers. The recognizers are highlighted in red when they require training with user-dependent samples, in green when they proved accurate when trained with user-independent samples, and in blue when the designer must input the formal definition of all gesture classes instead of providing samples. 19
2.10 Recognition accuracy rates (user-independent) for gestures articulated by people with low vision: (a) direct comparison between $P+ and $P; (b) combined effect of the number of training participants P and the number of training samples per gesture type T on the recognition accuracy of $P+; (c) gain in recognition accuracy enabled by $P+ over the other gesture dissimilarity measures. 20
2.11 Gesture articulations produced by people with low vision present more variation than gestures produced by people without visual impairments, which negatively affects recognizers' accuracy rates. 20
2.12 Point matchings produced by $P (left) and $P+ (right) for two [Pleaseinsertintopreamble]star" gestures produced by a participant with low vision. Note how the one-to-many matchings of $P+ help reduce undesirable point matchings of $P (see the long connecting lines in the figure on the left). 21
2.13 Three-step preprocessing applied by the $P+ recognizer on a rectangle. . 21

2.14 QuantumLeap Framework software architecture. 25
2.15 Overview page. 26
2.16 module page. 26
2.17 API page. 26
2.18 Commands and the corresponding gestures performed with the dominant hand. 28
2.19 Commands and the corresponding gestures performed with the dominant hand. 28
2.20 Virtual environment created with the help of Unity. 30
2.21 result from the evaluation process 31

3.1 Visualization of a basic instruction 33
3.2 Permutation of a basic instruction 33
3.3 Permutation of a basic instruction 33
3.4 Permutation of a basic instruction 33
3.5 Permutation of a basic instruction 34
3.6 Visualization of a common instruction 34
3.7 Visualization of a composed instruction 34
3.8 Nassi-Schneiderman's Graph 36
3.9 Comparison of 2D stroke gestures by congruence and hierarchy 36
3.10 Syntax tree of a logical basic instruction 40
3.11 Syntax tree of a permuted basic instruction 40
3.12 Syntax tree of a logical common instruction 41
3.13 Syntax tree of a permuted basic instruction 41
3.14 Syntax tree of a macro instruction 41
3.15 Syntax tree of a logical composed instruction 42
3.16 Syntax tree of a permuted composed instruction 42
3.17 Visualization of the error in composed instruction 43
3.18 process modeling of our scenario 46

4.1 Table of gestures 52
4.2 An example of an image of one device 54
4.3 Screenshot of the application 56
4.4 The sequence diagram of the process of recording 56
4.5 The sequence diagram of the process of recognizing 58
4.6 The sequence diagram of the process of delete 58
4.7 The sequence diagram of the process of clearing all the datasets 59
4.8 QuantumLeap configuration for EMERITI 60
4.9 EMERITI's code structure 61
4.10 QuantumLeap's code structure 63
4.11 QuantumLeap's code structure (dataset) 63
4.12 The environment EMERITI. 64
4.13 A basic record with the drop-down list. 65
4.14 A macro record with the drop-down list. 66
4.15 A composed record with the drop-down list and the button "add instruction". 67
4.16 A composed record with the drop-down list, the macro-commands list, and the button "add instruction". 68
4.17 The table of gestures and the table of macro/composed instruction. . . . 68
4.18 A display of a gesture. 69

A.1 Identification of tasks 78
A.2 The process of our scenario 82
A.3 Task with low complexity 82
A.4 Task with medium complexity 83
A.5 Task with high complexity 84
A.6 Organization modeling 84
A.7 Job and user model 85
A.8 Workflow model 86

Appendix A

Analysis Work

Figure A.1: Identification of tasks

ID	Task name	Pred.	Definition	Rationale	Nature
1	LaunchApplication	/	Launch the application	/	Interactive
2	MainMenu	1,3,8, 25	A user arrives to the main menu	- %1: Change of resource of type software: +QuantumLeap Waiting point of type decision: "If a user decides to launch the application" - %3: Change of resource of type information: -LD Waiting point of type decision: "If a user decides to leave the list of devices" - %Global: Change of resource of type information: +LP Difference of execution frequency: F(2) != F(1),F(3),F(8),F(25)	Interactive
3	AccessListDevices	2,4,5, 10,11	A user access to the list of the connected devices	- %2 and %10: Waiting point of type decision: "If a user decides to access to the list of connected devices" - %Global: Change of resource of type information: +LD Difference of execution frequency: F(3)!=F(2),F(4),F(5),F(10),F(11)	Interactive
4	AddDevice	3	A user can add a device to the list	- Change of resource of type information: +Device - Change of resource of type information: LD->LD' - Waiting point of type decision: "If a user decides to add a device in the list of connected devices" - Difference of execution frequency: F(4)!= F(3)	Interactive
5	RemoveDevice	3	A user can remove a device to the list	- Change of resource of type information: -Device - Change of resource of type information: LD->LD' - Waiting point of type decision: "If a user decides to remove a device in the list of connected devices" - Difference of execution frequency: F(5)!= F(3)	Interactive

6	CreateProfile	2	A user creates his profile. To realize the creation, he has to create a gestword, i.e., a password created from one or more gestures and enters all his personal information	- Change of resource of type information: +Profile - Change of resource of type information: LP ->LP' - Waiting point of type decision: "If a user decides to create a profile" - Difference of execution frequency: F(6)!= F(2)	Interactive
7	SelectDevices	6	A user selects the different devices he wishes to use as well as to choose or create an order of priority with respect to the other members of the family	- Change of resource of type information: +DevicesSelected - Change of resource of type information: LD->LD' - Difference of execution frequency: F(7)!= F(6)	Interactive
8	DeleteProfile	2	An administrator enters his gestword and decides to delete a profile	- Change of resource of type information: -Profile - Change of resource of type information: LP ->LP' - Waiting point of type decision: "If a user decides to delete a profile" - Difference of execution frequency: F(8)!= F(2)	
9	ConnectProfile	2	A user decides to enter his gestword and to connect to his profile	- Change of resource of type information: +ConnectedProfile - Waiting point of type decision: "If a user decides to connect to a profile" - Difference of execution frequency: F(9)!= F(2)	Interactive
10	ProfileMenu	3,7,9, 12,15, 17,22, 24	A user arrives to profile menu	- %3: Change of resource of type information: -LD Waiting point of type decision: "If a user decides to go to the profile menu" - %12: Change of resource of type information: -CG Waiting point of type decision: "If a user decides to go to the profile menu" - %17 and %24: Change of resource of type person: + User - %Global: Difference of execution frequency: F(10)!= F(3), F(7),F(9), F(12), F(15), F(17), F(22), F(24)	Interactive
11	AddNewGesture	10	A user decides to create a new gesture assigned to a new device	- Change of resource of type information: +Gesture - Waiting point of type decision: "If a user decides to create a new gesture" - Difference of execution frequency: F(11)!= F(10)	Interactive
12	AccessCatalogueOfGestures	10,13	A user has access to a catalogue of gestures that correspond to all	- %Global: Change of resource of type information: +CG	Interactive

			the gestures most frequently used by the community to perform certain actions	Waiting point of type decision: "If a user decides to access to the catalogue of gestures" Difference of execution frequency: F(12)!= F(10),F(13)	
13	ChooseFunctionality	3	A user chooses a functionality from all the possibilities of a chosen device	- Change of resource of type information: +ChosenFunctionality - Difference of execution frequency: F(14)!= F(13)	Interactive
14	CreateGesture	12,13, 18,23	After choosing a device and a functionality from all the possibilities of the chosen device, a user creates and assign a sequence of gestures to an action	- %12 and %18 : Waiting point of type decision: "If a user decides to create a gesture" - %23 : Change of resource of type person: + User - %Global: Change of resource of type information: +CreatedRepeatedGesture Difference of execution frequency: F(14)!= F(12),F(13), F(18),F(23)	Interactive
15	RepeatGesture	14,20, 21	A user must repeat 5 times the gesture to have a good recognition	- %Global: Change of resource of type information: PCG->PCG' Change of resource of type information: +PG Existence of a waiting point of type accumulation: after 5 gestures correctly realized Difference of execution frequency: F(15)!= F(14),F(20),F(21)	Interactive
16	PerformGesture	10	A user performs a created sequence of gestures to realize an action	- Change of resource of type information: +GesturePerformed - Waiting point of type decision: "If a user decides to perform a created sequence of gestures" - Difference of execution frequency: F(16)!= F(10)	Interactive
17	DisplayLoadingBar	17	QuantumLeap displays a loading bar on the screen to see the progression of the action	- Change of resource of type person: - User - Change of resource of type information: -GesturePerformed - Frequency of execution: F(17)!= F(16)	Automatic
18	AccessPersonalCatalogue	10,13, 19,20, 21	To view all his previously created gestures, a user decides to consult his personal catalogue	- %10: Waiting point of type decision: "If a user decides to access to the personal catalogue" - %19 : Change of resource of type information: -PG - %Global: Change of resource of type information: +PCG Waiting point of type decision: "If a user decides to access to the personal catalogue" Difference of execution frequency: F(18)!= F(10),F(13),F(19),F(20),F(21)	Interactive

19	ChoosePersonalGesture	18	A user chooses a gesture in his personal catalogue	- Waiting point of type decision: "If a user decides to choose a gesture in his personal catalogue" - Change of resource of type information: +PG - Difference of execution frequency: F(19)!= F(18)	Interactive
20	DeletePersonalGesture	19,23	He has the possibility to delete a gesture in his personal catalogue	- %19 : Waiting point of type decision: "If a user decides to delete a gesture in this personal catalogue" - %23: Change of resource of type person: + User - %Global: Change of resource of type information: -PG Change of resource of type information: PCG -> PCG' Difference of execution frequency: F(20)!= F(19),F(23)	Interactive
21	ModifyPersonalGesture	19,23	He has the possibility to modify a gesture in his personal catalogue. It means that he can replace a gesture by an another one	- %19 : Waiting point of type decision: "If a user decides to modify a gesture in this personal catalogue" - %23: Change of resource of type person: + User - %Global: Change of resource of type information: PG->PG' Change of resource of type information: PCG -> PCG' Difference of execution frequency: F(21)!= F(19),F(23)	Interactive
22	CopyActions	7	He can copy some gestures of actions from other users to use them on his account	- Change of resource of type information: PCG -> PCG' - Waiting point of type decision: "If a user decides to copy actions from other profiles" - Difference of execution frequency: F(22)!= F(7)	Interactive
23	WarningSameGesture	14	QuantumLeap displays an error about the fact that the user has created a gesture who was already assigned in his personal catalogue of gestures	- Change of resource of type person: - User - Change of resource of type information: -Gesture - Difference of execution frequency: F(23)!= F(14)	Automatic
24	ErrorUnauthorisedGesture	16	A message appears saying that the child does not have permission to perform this action	- Change of resource of type person: - User - Change of resource of type information: -GesturePerformed - Difference of execution frequency: F(24)!= F(16)	Automatic
25	DisconnectProfil	10	A user can disconnect to a profile	- Change of resource of type information: - ConnectedProfile	Interactive

				- Waiting point of type decision: "If a user decides to disconnect to a profile" - Difference of execution frequency: F(25)!= F(10)	
26	LogOut	2	Log out the application	- Change of resource of type information: - QuantumLeap - Waiting point of type decision: "If a user decides to leave the application" - Difference of execution frequency: F(26)!= F(2)	Interactive

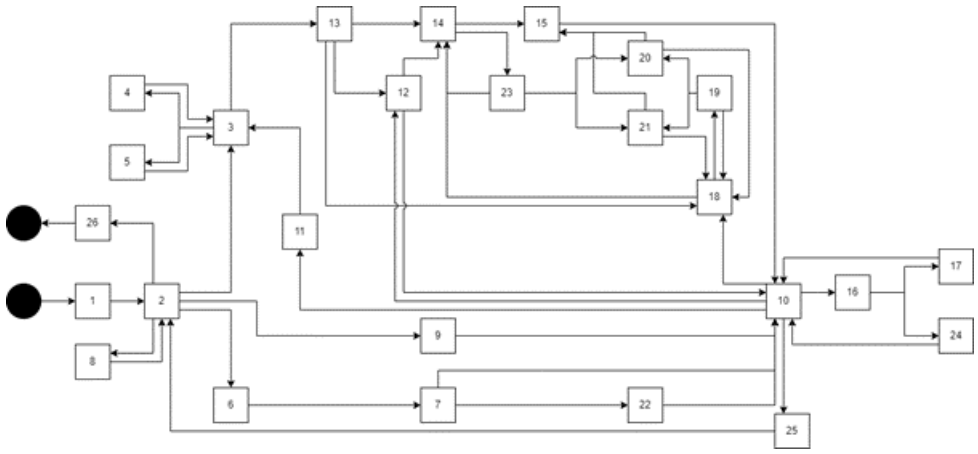

Figure A.2: The process of our scenario

A.1 Task Modeling

A.1.1 Task with Low Complexity: Task 1 – Launch Application

- **Description**: In order to try the QuantumLeap application, the user must first turn on their computer and their 3D recognition device (LeapMotion).

- **Precondition**: Have a computer and the LeapMotion device

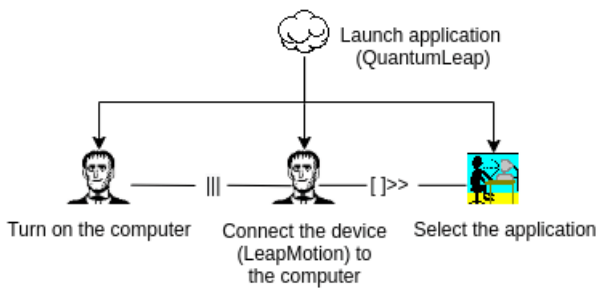

Figure A.3: Task with low complexity

A.1.2 Task with Medium Complexity: Task 17 - Create Gesture

- **Description**: A user creates and assign a sequence of gestures to an action. In order to do this, he has to come up with an idea for a gesture, perform it and repeat it 5 times. In case the new gesture is already in the saved gestures, the user will have to modify or delete a gesture.

- **Precondition**: In order to be able to create a gesture, the user must have gone to their profile (task 12)

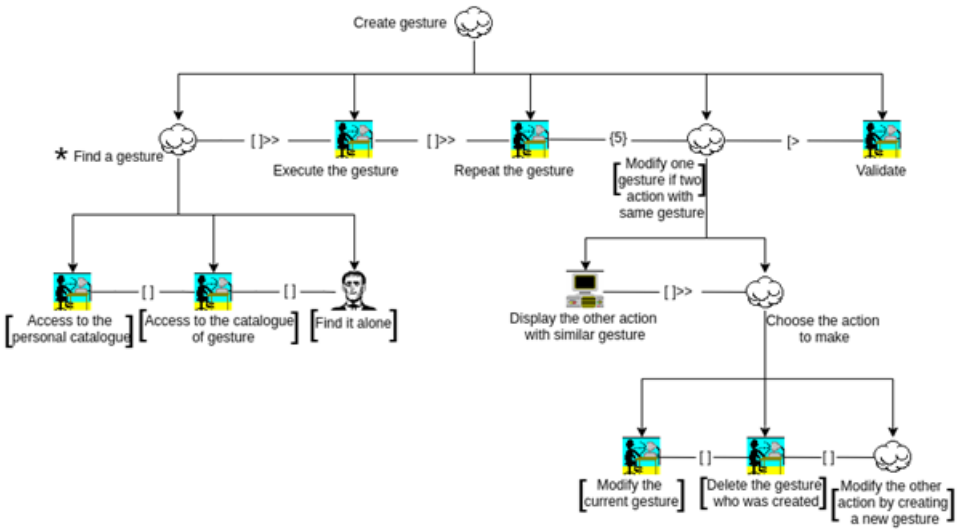

Figure A.4: Task with medium complexity

A.1.3 Task with High Complexity: Task 11 - Connect Profile

- **Description**: In order to use the QuantumLeap application, the user must log in to an ac-count. If the user does not have an account, he must create a new one. To do so, he will have to give his/her information, choose the devices he wants to use and finally, the potential parents who will limit his/her actions.

- **Precondition**: In order to log in (or create a new profile), the user must first have arrived at the main menu of the application (task 2).

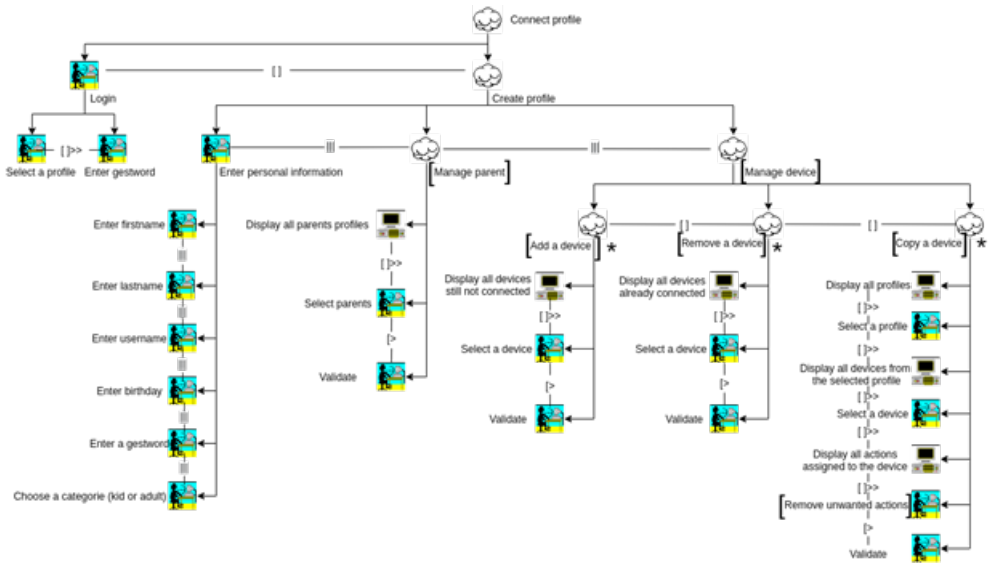

Figure A.5: Task with high complexity

A.2 Organization Modeling

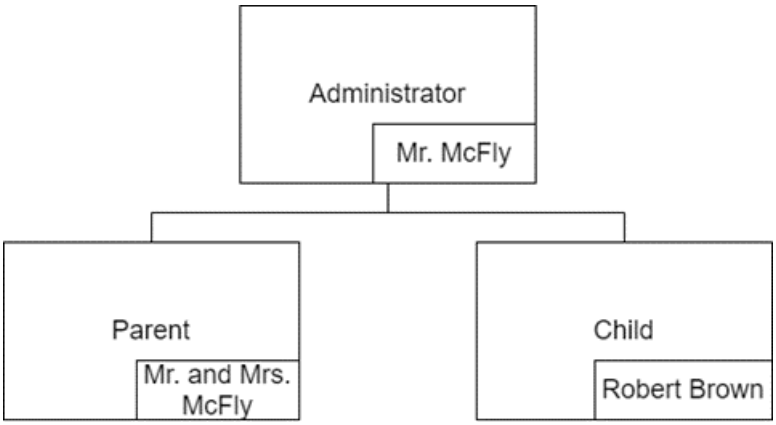

Figure A.6: Organization modeling

- **Administrator**: Person with full rights in QuantumLeap. He can choose the restrictions of the child accounts. In the case of this scenario, Mr. McFly has the role of administrator.
- **Parent**: The parent account has access to all features while in their profile. However, they cannot delete profiles.
- **Child**: This is a child account represented by Robert Brown. This account can perform many tasks, but it does not have access to certain actions decided by the administrator. For example, if a child decides to turn on the lights in a time slot where they do not have permission, then the action will not be carried out.

Task ID	1	2	3	4	5	6	7	8	9	10	11	12	13
Adminis-trator	X	X	X	X	X	X	X	X	X	X	X	X	X
Parent	X	X	X			X	X		X	X	X	X	X
Child	X	X	X			X	X		X	X	X	X	X

Task ID	14	15	16	17	18	19	20	21	22	23	24	25	26
Adminis-trator	X	X	X	X	X	X	X	X	X	X		X	X
Parent	X	X	X	X	X	X	X	X	X	X		X	X
Child	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure A.7: Job and user model

A.3 Workflow Modeling

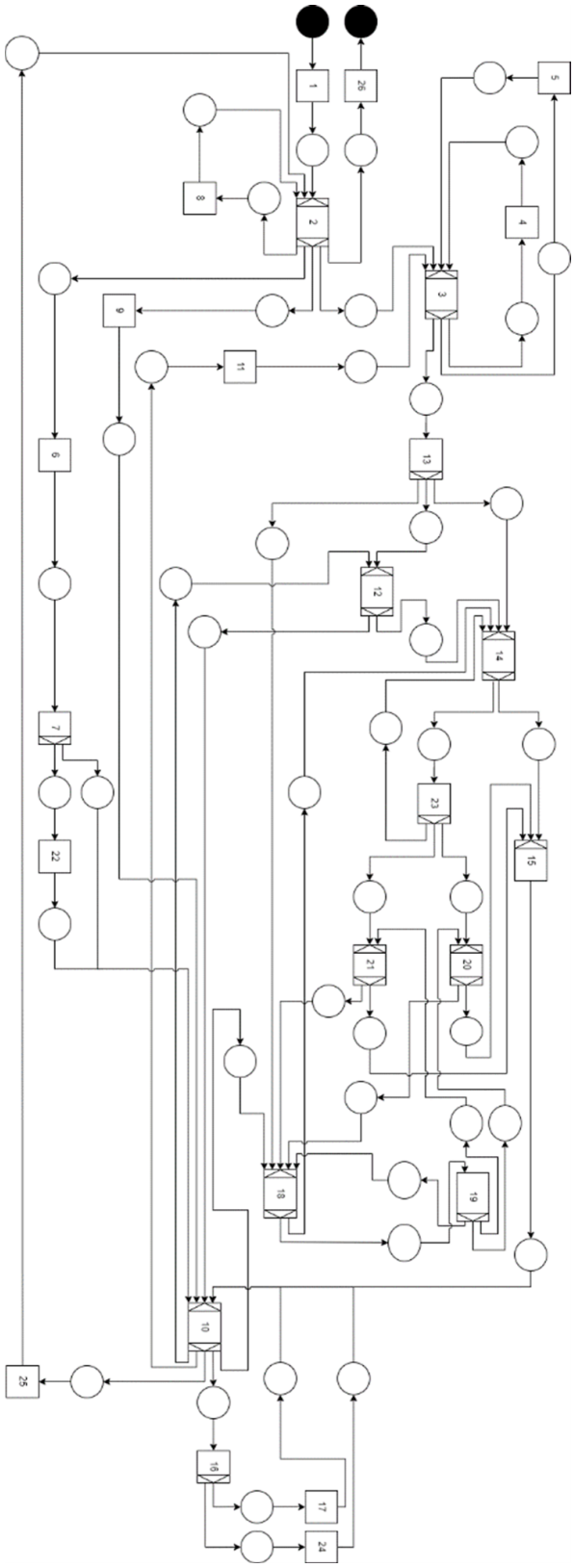

Figure A.8: Workflow model

Tasks	Pattern	Observations
T1	RBA DBOM RIEA	- Every user can launch the application. - Every user has the possibility to launch the application. - A user decides to launch the application.
T2	RBA DBOM RIEA	- Every user can arrive to the main menu. - Every user has the possibility to arrive at the main menu. - A user decides to arrive to the main menu.
T3	RBA DBOM RIA	- Every user can access the list of connected devices. - Every user has the possibility to access to the list of connected devices. - A user can access the list of connected devices when he wants to.
T4	RA, OA DBAS RIEA	- Only the administrator has the authorization to add a device - The administrator is the only person who can add a device. - A user decides to add a device.
T5	RA, OA DBAS RIEA	- Only the administrator has the authorization to remove a device - The administrator is the only person who can remove a device. - A user decides to remove a device.
T6	RBA DBOM RIA	- Every user can create a profile. - Every user has the possibility to create a profile. - A user can create a profile when he wants to.
T7	RBA, CH DBOS RIA	- Everyone is allowed to select the different devices he wishes to use. - This task is often processed by the same person. - The application allows the user who created the account to select devices. - The user can select all the devices he wants to use.
T8	RA DBOS RIA	- Only the administrator is allowed to delete a profile. - The application allows only the administrator to delete an account. - The user can delete a profile.
T9	RA, CBA DBOM RIA	- It is either the user who knows the gest word or an administrator who are allowed to connect to a profile. - Every user has the possibility to connect to his profile. - The user can connect to a profile when he wants to.
T10	RBA DBOM RIA	- Every user can arrive to the profile menu. - Every user has the possibility to arrive to the main menu. - The user can arrive to the profile menu.
T11	RBA	- Every user can add a new gesture.

	DBOM RIA	- Every user has the possibility to add a new gesture. - The user can add a gesture when he wants to.
T12	RBA DBOM RIA	- Every user can access the catalogue of gestures. - Every user has the possibility to access to the Catalogue of gestures. - The user can access the catalogue of gestures when he wants to.
T13	RBA, CH DBOS RIEA	- Every user can choose a functionality from all the possibilities of the chosen device - This task is often processed by the same person. - The application allows the user who want to create a new gesture to choose a functionality. - The user has to choose a functionality.
T14	RBA, CH DBOS RIA	- Every user can create and assign a sequence of gestures to an action - This task is often processed by the same person. - The application allows the user who want to create a new gesture to create a gesture and assign to an action. - The user can create and assign a gesture when he wants to.
T15	RBA, CH, CBA DBOS RIEA	- Every user who knows the initial gesture can repeat 5 times this one. - This task is often processed by the same person. - The application allows the user who want to create a new gesture to repeat 5 times the same gesture. - The user has to repeat the gesture 5 times.
T16	RBA DBOM RIA REDO	- Every user can perform a created sequence of gestures to realize an action - Every user has the possibility to perform a gesture. - Every user can perform a gesture when he wants to. - A user has the possibility to execute an infinite time some gestures which are not recognize.
T17	AE DBAS, DE RIEA	- QuantumLeap automatically displays a loading bar on the screen to see the progression of the action - Only QuantumLeap has the possibility to display a loading bar at the moment the gesture is recognized. - QuantumLeap has to display a loading bar on the screen to see the progression of the action.
T18	RBA DBOM RIA	- Every user can view all his previously created gestures - Every user has the possibility to access

		his personal catalogue. - The user can access the catalogue of personal gestures when he wants to.
T19	RBA DBOM RIEA	- Every user can choose a gesture from the personal catalogue. - Every user has the possibility to choose a personal gesture. - The user has to choose a gesture in his personal catalogue.
T20	RA, CBA DBOM RIEA	- It is either the user who knows the gestword or an administrator who are allowed to delete a gesture from the personal catalogue. - Every user who knows the gestword has the possibility to delete a personal gesture. - The user has to delete a personal gesture when he wants to.
T21	RA, CBA DBOM RIEA	- It is either the user who knows the gestword or an administrator who are allowed to modify a gesture from the personal catalogue. - Every user who knows the gestword has the possibility to modify a personal gesture. - The user has to modify a personal gesture when he wants to.
T22	RBA, CH DBOS RIA	- Every user can copy some gestures of actions from other users to use them on the created account. - This task is often processed by the same person. - The application allows the user who has created the account to copy gestures of actions from other users. - The user can copy some gestures of actions from other users when he wants to.
T23	AE DBAS, DE RIEA	- QuantumLeap automatically displays an error about the fact that the user has created a gesture which was already assigned in his personal catalogue of gestures. - Only QuantumLeap has the possibility to display an error message when the user has created a gesture which was already assigned. - QuantumLeap has to display an error message on the screen when the user has created a gesture which was already assigned.
T24	AE DBAS, DE RIEA	- QuantumLeap automatically displays a message saying that the child doesn't have permission to perform this action. - Only QuantumLeap has the possibility to display an error message when a child doesn't have permission to perform an action.

		- QuantumLeap has to display an error saying that the child doesn't have permission to perform this action.
T25	RBA DBOM RIA	- Every user can disconnect from the profile. - Every user has the possibility to disconnect to a profile. - The user can disconnect from his profile when he wants to.
T26	RBA DBOM RIA	- Every user can log out the application. - Every user has the possibility to log out the application. - The user can log out the application when he wants to.

A.4 Workflow Analysis

In the case of our analysis, it is difficult for us to imagine possible improvements in our work since our study is based on an application that is already "To be". Therefore, our program is supposed to be already optimized.

However, it is possible to imagine an even more advanced technology than the one presented here, which would allow us to avoid and skip certain tasks in the software. Among these possible improvements, one can imagine a technology with facial recognition that would allow each user to use their account simultaneously. Taking this idea, a step further, one could also imagine technology that recognizes users' gestures when they are in a room (such as a camera or a Kinect). This would avoid the need for users to stand in front of the computer each time to use the various connected devices. It would require more effort to set up the software, but afterwards it would be much easier to use. A final feature that might be relevant would be the creation of a display device that would indicate to the user, if a gesture is made but not recognized, to indicate the gestures closest to it. This would save the user from having to go to their gesture catalogue every time they forgot the exact gesture to perform an action.

As you can see, it is obviously possible to imagine many advanced features and technologies that would im-prove the use of our tool. Among the examples we have given above, these are ideas that may be possible soon but there are many technical difficulties to overcome in order to achieve them.

Appendix B

ANTLR Code

B.1 ANTLR Code

```
grammar g;


ACTION:'Turn on '|'Turn off '|'Increase '|'Decrease '|'Pause '|'Play '|'
    Preheat '|'Mute ';
MACRO:'Macro ';
DEVICE:'the television '|'the audio player '|'the car '|'the radio '|'the
    air conditioner '|'the lights ';
PARAMETER: ( 'the volume on ' | 'until ' | 'in '  | 'on ' ) Number | '
    Favorite ' | Number ;
ENVIRONMENT: 'in the kitchen '| 'in the living room' |' in the playroom '|'
     in the bedroom '|'in the garage ';
Number : '0'..'9'+' '; // plusieurs chiffres avec digit

instruction :macroInstruction | basicInstruction | commonInstruction| (
    composedInstruction )+;
composedInstruction :  (macroInstruction | basicInstruction |
    commonInstruction)+ ;


macroInstruction:MACRO ;
basicInstruction : (BI1|BI2|BI3|BI4|BI5|BI6|BI7|BI8|BI9|BI10|BI11|BI12|BI13
    |BI14|BI15|BI16|BI17|BI18|BI19|BI20|BI21|BI22|BI23|BI24)+;
BI1 :ACTION DEVICE PARAMETER* ENVIRONMENT*;
BI2 :DEVICE ACTION PARAMETER* ENVIRONMENT*;
BI3 :DEVICE ACTION ENVIRONMENT* PARAMETER*;
BI4 :ACTION DEVICE ENVIRONMENT* PARAMETER*;
BI5 :ACTION PARAMETER* DEVICE ENVIRONMENT*;
BI6 :ACTION PARAMETER* ENVIRONMENT* DEVICE;
BI7 :ACTION ENVIRONMENT* PARAMETER* DEVICE;
BI8 :ACTION ENVIRONMENT* DEVICE PARAMETER*;
BI9 :DEVICE PARAMETER* ACTION ENVIRONMENT*;
BI10:DEVICE PARAMETER* ENVIRONMENT* ACTION;
BI11:DEVICE ENVIRONMENT* ACTION PARAMETER*;
BI12:DEVICE ENVIRONMENT* PARAMETER* ACTION;
BI13:ENVIRONMENT* ACTION PARAMETER* DEVICE;
BI14:ENVIRONMENT* ACTION DEVICE PARAMETER*;
BI15:ENVIRONMENT* DEVICE ACTION PARAMETER*;
BI16:ENVIRONMENT* DEVICE PARAMETER* ACTION;
BI17:ENVIRONMENT* PARAMETER* DEVICE ACTION;
```

```
BI18:ENVIRONMENT* PARAMETER* ACTION DEVICE;
BI19:PARAMETER* ACTION ENVIRONMENT* DEVICE;
BI20:PARAMETER* ACTION DEVICE ENVIRONMENT*;
BI21:PARAMETER* DEVICE ACTION ENVIRONMENT*;
BI22:PARAMETER* DEVICE ENVIRONMENT* ACTION;
BI23:PARAMETER* ENVIRONMENT* DEVICE ACTION;
BI24:PARAMETER* ENVIRONMENT* ACTION DEVICE;

commonInstruction:(CI1 |CI2|CI3|CI4|CI5|CI6|CI7|CI8|CI9|CI10|CI11|CI12|CI13
    |CI14|CI15|CI16|CI17|CI18|CI19|CI20|CI21|CI22|CI23|CI24)+;
CI1 :ACTION DEVICE* PARAMETER* ENVIRONMENT*;
CI2 :DEVICE* ACTION PARAMETER* ENVIRONMENT*;
CI3 :DEVICE* ACTION ENVIRONMENT* PARAMETER*;
CI4 :ACTION DEVICE* ENVIRONMENT* PARAMETER*;
CI5 :ACTION PARAMETER* DEVICE* ENVIRONMENT*;
CI6 :ACTION PARAMETER* ENVIRONMENT* DEVICE*;
CI7 :ACTION ENVIRONMENT* PARAMETER* DEVICE*;
CI8 :ACTION ENVIRONMENT* DEVICE* PARAMETER*;
CI9 :DEVICE* PARAMETER* ACTION ENVIRONMENT*;
CI10:DEVICE* PARAMETER* ENVIRONMENT* ACTION;
CI11:DEVICE* ENVIRONMENT* ACTION PARAMETER*;
CI12:DEVICE* ENVIRONMENT* PARAMETER* ACTION;
CI13:ENVIRONMENT* ACTION PARAMETER* DEVICE*;
CI14:ENVIRONMENT* ACTION DEVICE* PARAMETER*;
CI15:ENVIRONMENT* DEVICE* ACTION PARAMETER*;
CI16:ENVIRONMENT* DEVICE* PARAMETER* ACTION;
CI17:ENVIRONMENT* PARAMETER* DEVICE* ACTION;
CI18:ENVIRONMENT* PARAMETER* ACTION DEVICE*;
CI19:PARAMETER* ACTION ENVIRONMENT* DEVICE*;
CI20:PARAMETER* ACTION DEVICE* ENVIRONMENT*;
CI21:PARAMETER* DEVICE* ACTION ENVIRONMENT*;
CI22:PARAMETER* DEVICE* ENVIRONMENT* ACTION;
CI23:PARAMETER* ENVIRONMENT* DEVICE* ACTION;
CI24:PARAMETER* ENVIRONMENT* ACTION DEVICE*;
```

Bibliography

[1] C. Appert and S. Zhai, "Using strokes as command shortcuts: Cognitive benefits and toolkit support," p. 2289–2298, 2009. [Online]. Available: https://doi.org/10.1145/1518701.1519052

[2] D. M. Lane, H. A. Napier, S. C. Peres, and A. Sandor, "Hidden costs of graphical user interfaces: Failure to make the transition from menus and icon toolbars to keyboard shortcuts," *Int. J. Hum. Comput. Interact.*, vol. 18, no. 2, pp. 133–144, 2005. [Online]. Available: https://doi.org/10.1207/s15327590ijhc1802_1

[3] W. Delamare, C. Silpasuwanchai, S. Sarcar, T. Shiraki, and X. Ren, "On gesture combination: An exploration of a solution to augment gesture interaction," 11 2019, pp. 135–146.

[4] J. O. Wobbrock, M. R. Morris, and A. D. Wilson, "User-defined gestures for surface computing," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1083–1092. [Online]. Available: https://doi.org/10.1145/1518701.1518866

[5] D. Lane, H. Napier, S. Peres, and A. Sandor, "Hidden costs of graphical user interfaces: Failure to make the transition from menus and icon toolbars to keyboard shortcuts," *Int. J. Hum. Comput. Interaction*, vol. 18, pp. 133–144, 05 2005.

[6] Y. Rogers, "Evaluating the meaningfulness of icon sets to represent command operations," in *Proceedings of the Second Conference of the British Computer Society, Human Computer Interaction Specialist Group on People and Computers: Designing for Usability*. USA: Cambridge University Press, 1986, p. 586–603.

[7] P. Morrel-Samuels, "Clarifying the distinction between lexical and gestural commands," *Int. J. Man-Mach. Stud.*, vol. 32, no. 5, p. 581–590, may 1990. [Online]. Available: https://doi.org/10.1016/S0020-7373(05)80034-3

[8] F. Craik and R. Lockhart, "Levels of processing: A framework for memory research," *Journal of Verbal Learning and Verbal Behavior*, vol. 11, pp. 671–, 12 1972.

[9] T. Grossman, P. Dragicevic, and R. Balakrishnan, "Strategies for accelerating on-line learning of hotkeys," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 1591–1600. [Online]. Available: https://doi.org/10.1145/1240624.1240865

[10] J. Kela, P. Korpipää, J. Mäntyjärvi, S. Kallio, G. Savino, L. Jozzo, and S. Marca, "Accelerometer-based gesture control for a design environment," *Personal and Ubiquitous Computing*, vol. 10, pp. 285–299, 08 2006.

[11] C. Kühnel, T. Westermann, F. Hemmert, S. Kratz, A. Müller, and S. Möller, "I'm home: Defining and evaluating a gesture set for smart-home control," *International Journal of Human-Computer Studies*, vol. 69, no. 11, pp. 693–704, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1071581911000668

[12] P. Vogiatzidakis and P. Koutsabasis, "Frame-based elicitation of mid-air gestures for a smart home device ecosystem," *Informatics*, vol. 6, no. 2, 2019. [Online]. Available: https://www.mdpi.com/2227-9709/6/2/23

[13] M. Hoffman, P. Varcholik, and J. J. LaViola, "Breaking the status quo: Improving 3d gesture recognition with spatially convenient input devices," in *2010 IEEE Virtual Reality Conference (VR)*, 2010, pp. 59–66.

[14] S. Fothergill, H. Mentis, P. Kohli, and S. Nowozin, "Instructing people for training gestural interactive systems," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1737–1746. [Online]. Available: https://doi.org/10.1145/2207676.2208303

[15] R. Walter, G. Bailly, and J. Müller, "Strikeapose: Revealing mid-air gestures on public displays," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 841–850. [Online]. Available: https://doi.org/10.1145/2470654.2470774

[16] R.-D. Vatavu, "A comparative study of user-defined handheld vs. freehand gestures for home entertainment environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 5, pp. 187–211, 03 2013.

[17] ——, "The dissimilarity-consensus approach to agreement analysis in gesture elicitation studies," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–13. [Online]. Available: https://doi.org/10.1145/3290605.3300454

[18] W. Delamare, C. Coutrix, and L. Nigay, "Designing guiding systems for gesture-based interaction," in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ser. EICS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 44–53. [Online]. Available: https://doi.org/10.1145/2774225.2774847

[19] G. Rovelo, D. Degraen, D. Vanacken, K. Luyten, and K. Coninx, "Gestu-wan - an intelligible mid-air gesture guidance system for walk-up-and-use displays," in *Human-Computer Interaction – INTERACT 2015*, J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, and M. Winckler, Eds. Cham: Springer International Publishing, 2015, pp. 368–386.

[20] R.-D. Vatavu, C.-M. Chera, and W.-T. Tsai, "Gesture profile for web services: An event-driven architecture to support gestural interfaces for smart environments," in *Ambient Intelligence*, F. Paternò, B. de Ruyter, P. Markopoulos, C. Santoro, E. van Loenen, and K. Luyten, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 161–176.

[21] R.-D. Vatavu, "Nomadic gestures: A technique for reusing gesture commands for frequent ambient interactions," *Journal of Ambient Intelligence and Smart Environments*, vol. 4, pp. 79–93, 04 2012.

[22] F. Loch, "Hierarchical gestures: ggestural shortcuts for touchscreen devices," 2012. [Online]. Available: http://essay.utwente.nl/61931/

[23] A. M. Rahman, M. A. Hossain, J. Parra, and A. El Saddik, "Motion-path based gesture interaction with smart home services," in *Proceedings of the 17th ACM International Conference on Multimedia*, ser. MM '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 761–764. [Online]. Available: https://doi.org/10.1145/1631272.1631407

[24] R. Aigner, D. Wigdor, H. Benko, M. Haller, D. Lindlbauer, A. Ion, S. Zhao, and J. Koh, "Understanding mid-air hand gestures: A study of human preferences in usage of gesture types for hci," 11 2012.

[25] M. Karam and m. Schraefel, "A taxonomy of gestures in human computer interactions," *Electronics and Computer Science*, 01 2005.

[26] W. T. Freeman and C. D. Weissman, "Television control by hand gestures," in *International Workshop on Automatic Face and Gesture Recognition*, 1995, pp. 179–183.

[27] R.-D. Vatavu and I.-A. Zaiti, "Leap gestures for tv: Insights from an elicitation study," in *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*, ser. TVX '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 131–138. [Online]. Available: https://doi.org/10.1145/2602299.2602316

[28] F. Portet, M. Vacher, C. Golanski, C. Roux, and B. Meillon, "Design and evaluation of a smart home voice interface for the elderly [U+2015] acceptability and objection aspects," *Personal and Ubiquitous Computing*, vol. 17, pp. 1–18, 01 2012.

[29] R.-D. Vatavu, "User-defined gestures for free-hand tv control," in *Proceedings of the 10th European Conference on Interactive TV and Video*, ser. EuroITV '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 45–48. [Online]. Available: https://doi.org/10.1145/2325616.2325626

[30] R.-D. Vatavu and S.-G. Pentiuc, "Interactive coffee tables: Interfacing tv within an intuitive, fun and shared experience," in *Changing Television Environments*, M. Tscheligi, M. Obrist, and A. Lugmayr, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 183–187.

[31] D. Dave, A. Chowriappa, and T. Kesavadas, "Gesture interface for 3d cad modeling using kinect," *Computer-Aided Design and Applications*, vol. 10, 01 2012.

[32] E. A. Kirchner, J. de Gea Fernandez, P. Kampmann, M. Schröer, J. H. Metzen, and F. Kirchner, *Intuitive Interaction with Robots – Technical Approaches and Challenges*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, pp. 224–248. [Online]. Available: https://doi.org/10.1007/978-3-658-09994-7_8

[33] G. Kurtenbach, T. P. Moran, and W. Buxton, "Contextual animation of gestural commands," *Computer Graphics Forum*, vol. 13, no. 5, pp. 305–314, 1994. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.1350305

[34] Q. Roy, S. Malacria, Y. Guiard, E. Lecolinet, and J. Eagan, "Augmented letters: Mnemonic gesture-based shortcuts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 2325–2328. [Online]. Available: https://doi.org/10.1145/2470654.2481321

[35] M. Serrano, E. Lecolinet, and Y. Guiard, "Bezel-tap gestures: Quick activation of commands from sleep mode on tablets," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 3027–3036. [Online]. Available: https://doi.org/10.1145/2470654.2481421

[36] J. O. Wobbrock, "$1 unistroke recognizer," 2022. [Online]. Available: https://depts.washington.edu/acelab/proj/dollar/index.html

[37] ——, "$n multistroke recognizer," 2022. [Online]. Available: https://depts.washington.edu/acelab/proj/dollar/ndollar.html

[38] S. Zhai, P. O. Kristensson, C. Appert, T. H. Anderson, and X. Cao, "Foundational issues in touch-surface stroke gesture design — an integrative review," *Foundations and Trends® in Human–Computer Interaction*, vol. 5, no. 2, pp. 97–205, 2012. [Online]. Available: http://dx.doi.org/10.1561/1100000012

[39] A. Authors, "Towards congruent and hierarchical 2d stroke gestures as commands," 2021.

[40] G. Kurtenbach, G. Fitzmaurice, T. Baudel, and B. Buxton, "The design of a gui paradigm based on tablets, two-hands, and transparency," in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 35–42. [Online]. Available: https://doi.org/10.1145/258549.258574

[41] H. Hse, M. Shilman, and A. R. Newton, "Robust sketched symbol fragmentation using templates," in *Proceedings of the 9th International Conference on Intelligent User Interfaces*, ser. IUI '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 156–160. [Online]. Available: https://doi.org/10.1145/964442.964472

[42] N. Magrofuoco, P. Roselli, and J. Vanderdonckt, "v: An articulation, rotation, scaling, and translation invariant (arst) multi-stroke gesture recognizer," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, no. EICS, jun 2022. [Online]. Available: https://doi.org/10.1145/3532200

[43] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes," in *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 159–168. [Online]. Available: https://doi.org/10.1145/1294211.1294238

[44] L. Anthony and J. O. Wobbrock, "A lightweight multistroke recognizer for user interface prototypes," in *Proceedings of Graphics Interface 2010*, ser. GI '10. CAN: Canadian Information Processing Society, 2010, p. 245–252.

[45] R.-D. Vatavu, L. Anthony, and J. Wobbrock, "Gestures as point clouds: A $p recognizer for user interface prototypes," 10 2012, pp. 273–280.

[46] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock, "Gestures as point clouds: A $p recognizer for user interface prototypes," in *Proceedings of the 14th ACM International Conference on Multimodal Interaction*, ser. ICMI '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 273–280. [Online]. Available: https://doi.org/10.1145/2388676.2388732

[47] R.-D. Vatavu, "Improving gesture recognition accuracy on touch screens for users with low vision," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 4667–4679. [Online]. Available: https://doi.org/10.1145/3025453.3025941

[48] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock, "$q: A super-quick, articulation-invariant stroke-gesture recognizer for low-resource devices," in *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3229434.3229465

[49] M. Ousmer, A. Sluÿters, N. Magrofuoco, P. Roselli, and J. Vanderdonckt, "Recognizing 3d trajectories as 2d multi-stroke gestures," *Proc. ACM Hum.-Comput. Interact.*, vol. 4, no. ISS, nov 2020. [Online]. Available: https://doi.org/10.1145/3427326

[50] F. M. Caputo, P. Prebianca, A. Carcangiu, L. D. Spano, and A. Giachetti, "A 3 Cent Recognizer: Simple and Effective Retrieval and Classification of Mid-air Gestures from Single 3D Traces," in *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, A. Giachetti, P. Pingi, and F. Stanco, Eds. The Eurographics Association, 2017.

[51] J. Herold and T. F. Stahovich, "The One Cent Recognizer: A Fast, Accurate, and Easy-to-Implement Handwritten Gesture Recognition Technique," in *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, K. Singh and L. B. Kara, Eds. The Eurographics Association, 2012.

[52] P. Kristensson and L. Denby, "Continuous Recognition and Visualization of Pen Strokes and Touch-Screen Gestures," in *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, T. Hammond and A. Nealen, Eds. The Eurographics Association, 2011.

[53] P.-O. Kristensson and S. Zhai, "Shark2: A large vocabulary shorthand writing system for pen-based computers," in *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 43–52. [Online]. Available: https://doi.org/10.1145/1029632.1029640

[54] J. Reaver, T. F. Stahovich, and J. Herold, "How to make a quick$: Using hierarchical clustering to improve the efficiency of the dollar recognizer," in *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, ser. SBIM '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 103–108. [Online]. Available: https://doi.org/10.1145/2021164.2021183

[55] R.-D. Vatavu, "The impact of motion dimensionality and bit cardinality on the design of 3d gesture recognizers," *International Journal of Human-Computer Studies*, vol. 71, no. 4, pp. 387–409, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1071581912002108

[56] N. Magrofuoco, P. Roselli, and J. Vanderdonckt, "Two-dimensional stroke gesture recognition: A survey," *ACM Comput. Surv.*, vol. 54, no. 7, jul 2021. [Online]. Available: https://doi.org/10.1145/3465400

[57] R.-D. Vatavu, "Improving gesture recognition accuracy on touch screens for users with low vision," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 4667–4679. [Online]. Available: https://doi.org/10.1145/3025453.3025941

[58] N. H. Villaroman, D. C. Rowe, and B. Swan, "Teaching natural user interaction using openni and the microsoft kinect sensor," pp. 227–232, 2011. [Online]. Available: https://doi.org/10.1145/2047594.2047654

[59] UltraLeap, "Leap motion controller." [Online]. Available: https://www.ultraleap.com/product/leap-motion-controller

[60] P. Vogiatzidakis and P. Koutsabasis, "'address and command': Two-handed mid-air interactions with multiple home devices," *International Journal of Human-Computer Studies*, vol. 159, p. 102755, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1071581921001737

[61] V. Bellotti, M. Back, W. K. Edwards, R. E. Grinter, A. Henderson, and C. Lopes, "Making sense of sensing systems: Five questions for designers and researchers," p. 415–422, 2002. [Online]. Available: https://doi.org/10.1145/503376.503450

[62] T. Parr and K. Fisher, "Ll(*): the foundation of the ANTLR parser generator," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, M. W. Hall and D. A. Padua, Eds. ACM, 2011, pp. 425–436. [Online]. Available: https://doi.org/10.1145/1993498.1993548

[63] P. Lech, "Wifi multi access point smart home iot architecture," in *Automation Control Theory Perspectives in Intelligent Systems - Proceedings of the 5th Computer Science On-line Conference 2016 (CSOC2016), Vol 3*, ser. Advances in Intelligent Systems and Computing, R. Silhavy, R. Senkerik, Z. K. Oplatková, P. Silhavy, and Z. Prokopova, Eds., vol. 466. Springer, 2016, pp. 247–254. [Online]. Available: https://doi.org/10.1007/978-3-319-33389-2_24

[64] R. Bruno, M. Conti, and E. Gregori, "Bluetooth: Architecture, protocols and scheduling algorithms," *Clust. Comput.*, vol. 5, no. 2, pp. 117–131, 2002. [Online]. Available: https://doi.org/10.1023/A:1013989524865

[65] M. Chellappa, S. Madasamy, and R. Prabakaran, "Study on zigbee technology," 04 2011, pp. 297–301.

[66] A. Sluÿters, M. Ousmer, P. Roselli, and J. Vanderdonckt, "Quantumleap, a framework for engineering gestural user interfaces based on the leap motion controller," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, no. EICS, jun 2022. [Online]. Available: https://doi.org/10.1145/3532211

[67] M. Beigl, "Point & click - interaction in smart environments," in *Handheld and Ubiquitous Computing, First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*, ser. Lecture Notes in Computer Science, H. Gellersen, Ed., vol. 1707. Springer, 1999, pp. 311–313. [Online]. Available: https://doi.org/10.1007/3-540-48157-5_31

[68] A. Authors, "The agile coach," 2022. [Online]. Available: https://www.atlassian.com/agile#:~:text=Agile%20is%20an%20iterative%20approach,small%2C%20but%20consumable%2C%20increments.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**