UCLouvain

epl

École polytechnique de Louvain

# Air+Touch Gesture Recognition : Algorithms, Software, and Experiment

Author: **Romain NEUVILLE**
Supervisor: **Jean VANDERDONCKT**
Readers: **Arthur SLUŸTERS, Mehdi OUSMER**
Academic year 2020–2021
Master[120] in Computer Science

**Abstract**

Human Computer Interaction became omnipresent in recent years, especially gesture recognition. Thanks to major advances in machine learning, gesture recognition has become an everyday tool. Currently, almost everybody has mastered 2D gestures with the use of smartphones. However, limits of these 2D gestures have emerged which has led researchers to focus on 3D gesture recognition. These two types of gesture enable to create Air+Touch recognition. It simply consists in an environment where gestures can be 2D or 3D. We have decided to deal with Air+Touch gesture recognition for this thesis. There is a clear lack of research on this subject. A problem related to this lack is that devices that can sense and retrieve data for the two types of gestures, are quite rare. The 3DTouchpad [34] device can sense Air+Touch gestures. Therefore, we can build our own gestures set based on this device and on all the Air+Touch gestures gathered from the literature. Once it was defined, we integrated the 3DTouchpad inside the QuantumLeap [50] framework where we tested all their different recognizers to see which one fits the best with Air+Touch gestures. Each test was performed according to two scenarios: User-Independent, User-Dependent. We also analyzed two gestures elicitation studies based on the 3DTouchpad to confirm that our Air+Touch assumptions were corresponding to 60 participants feelings. To conclude, we are going to discuss about the benefits and contributions of our work before suggesting some future promising works.

# Acknowledgements

First of all, I would like to thank Prof. Jean Vanderdonckt and Arthur Sluÿters who guided me and helped me a lot all along the thesis. This thesis wouldn't have been possible without them.

A special thank to Mr. Vanderdonckt, Mr. Sluÿters and Mr. Ousmer for the reading of this thesis.

I would also like to thank all the researchers who have been mentioned and through their work have made this thesis possible.

A huge thanks to each people who made the two gesture elicitation studies and all their participants.

I would to thank all the teachers who took the time to teach us the magic of computers sciences and all administrations staffs from EPL and UCLouvain.

Finally, thank you to all my friends and family without them I could not have achieved all this.

# Contents

# Chapter 1

# Introduction

## 1.1 Context Of The Problem

Over the last decade, Human Computer Interaction has shown a great and a deep interest in the gesture recognition. There are two main types of gesture. They are the 2D gestures and 3D gestures.

Nowadays, almost everyone has mastered these 2D gestures thanks to the increasing numbers of smartphones or by the use the pad of a laptop. They are present everywhere in our lives but this has not always been the case. If you go back to the beginning of the millennium, they were not so famous. Researchers have worked hard to standardize the 2D gestures [58, 7, 60, 46]. However, there is one major issue related to these gestures: there is a huge lack of expressiveness.

The second main type of gesture is called mid-air(3D) gestures. Mid-air gestures often consist in performing a gesture where the data is recorded by a 3D sensor or one or more depth(s) camera(s). They are often defined with the XYZ spaces variables [23, 19] whereas the 2D gestures are defined by only using XY. This additional space variable solves the main issue from the 2D gestures. But unfortunately mid-gesture are suffering from other major problems:

- Mid-air raw data is less accurate as it is harder to collect and interpret it.
- A gesture can easily be confused with other gestures due to the freedom of expression.
- A gesture can also be unintentionally performed. Imagine if you have to scratch your nose or your hand, the recognizer could detect a gesture and start an action. It can't know that gesture wasn't wanted.
- In comparison to the 2D gestures, the recognition rate of 3D gestures is lower.

All these issues are probably going to be reduced with future advances in machine learning. However, it will take some time.

Some interesting research [10, 33, 32, 42] has focused on defining an environment which is working with 2D gestures and 3D gestures mixed together. Their results were promising. The benefit of using a mixed gestures set is that it reduces considerably the problems associated with each type. For instance, you can define important action which needs to be accurate by a 2D gesture and action which can be scale depending on gesture distance by a 3D gesture. Air+Touch gestures set offers an incredible potential for combinations. However, it's also facing some problems. They are often technical problems. Recognizers as $3 [25], protactor3D [26], or $P family [56] are often implemented to be relevant for 2D data or 3D data but not the two types together. So they have to be adapted according to the situation. There is also a clear lack of research on the subject when we look at its possible potential. The major problem that hinders its development concerns devices. Today, we are witnessing a multiplication of 3D devices for the acquisition of gestures. But, almost all of these devices are designed for 3D gesture recognition only as the leapMotion [44], the microsoft Kinect [24] and many others. Only fews devices can retrieve the mixed data. Researchers often create some devices combinations to be able to recognize the two types of gestures. It can be a combination of camera sensor(s)/camera(s)/touch surface. For example, the Nintendo Wii [48] is using a remote controller which retrieves touching interaction associated with a 3D sensor which is retrieving the controller space position. We can also take the devices used for the VR games as an example. These "kind" of devices are often costly and not easy to use.

Fortunately, there are some dedicated Air+Touch devices which are cheap and easy to use. One is called the 3DTouchpad [34]. It is a device developed by Microchip [35]. It can sense 2D and also 3D data. It is composed of five 3D sensors and one touching sensor on its whole physical surface which is allowing (multi-)finger(s) interaction. There is a free and public Windows Software Development Kit(SDK) [34] for the 3DTouchpad. It is implemented in the C language. Therefore, it is possible to implement our own script for interacting with touchpad and retrieve the raw data needed from the device. Thus, it is possible to build our own gestures set definition associated to the 3DTouchpad. This gestures set definition contains 2D and 3D gestures. It can be very interesting to send 3DTouchpad data inside the QuantumLeap [50]. We will be able to test all our gestures set with the different recognizers from the framework. We will also be able to study about the different possible combinations and to understand which gestures are fitting well with other gestures. These previous interesting points will be the subject of this thesis.

## 1.2 Mission Statement

### 1.2.1 Statement

In order to mitigate the lack of studies on Air+Touch gestures, the central objective of this master thesis is **first to create an Air+Touch gestures set definition based on the 3Dtouchpad device, on two gesture elicitation studies and on all Air+Touch gestures found in the literature, then to provide a way to test the 3DTouchpad's data inside QuantumLeap and to see how the recognizers, which were trained with our gestures samples, are reacting to Air+Touch data**.

### 1.2.2 Research Questions

The main focus of this thesis is Air+Touch gestures. There are several steps detailed in each chapters in which their analysis and results answered the following research questions:

- Is the framework enough reusable and generic to be used with new devices?
- Is it possible to use multiple sensors at the same time with the framework?
- Which recognizer is performing the best with our Air+Touch gestures set definition? (Recognition Rate and Execution Timing)
- Is Air+Touch gestures solving the different problems related to 2D gestures and 3D gestures?
- Is Air+Touch gestures combinations interesting?
- Is our gestures set definition representative of the different *gesture elicitation studies* results based on the 3DTouchpad?

### 1.2.3 Working Hypotheses

We assume the following working hypotheses for this thesis:

**Hypothesis 1.** *Gestures set definition*: Our gestures set definition was build according to the touchpad performance. A device which is acquiring data the same way as the touchpad can be taken into account but not other devices.

**Hypothesis 2.** *Context of use*: The device must be used in the same environment as when samples were collected and tested. It must also be used in the same position in front of the user.

**Hypothesis 3.** *Benchmarks of recognizers*: The PC performance does not influence the execution time and recognition rate for each recognizer at each iteration.

**Hypothesis 4.** *Recognition rate*: Samples from each dataset were performed by different users. Therefore, a sample may vary according to the speed and the user understanding about the gesture.

**Hypothesis 5.** *Dataset*: We supposed that in each dataset, there should be enough different templates to ensure variation depending on each user. Everyone can therefore be considered as the target audience.

### 1.2.4 Research Method

This section is a brief overview of the research methodology of this thesis. All the different point mentioned here are going to be focus in more details in the next chapters.

1. *State of art*: The first thing to do in this thesis is to collect the maximum amount of information in papers, books and articles which are related to Air+Touch gesture recognition. With all these information gathered, we can draw up a sort of catalog of Air+Touch gestures from the literature. The goal is to base our gestures set on this catalog to provide the optimal Air+Touch gestures set definition as possible.

2. *Framework requirements*: Continue to follow the different phases defined by the IEEE Standard Glossary of Software Engineering Terminology [2] when we integrate our code related to the 3DTouchpad inside QuantumLeap [50]. It's important to follow the line on which the framework was built and not reduce its quality.

3. *Gestures Set Definition*: Define a gestures set which is based on the state of art and which exploits the touchpad as much as possible. Then, record the gestures with left-handed and right-handed participants to represent the global population.

4. *Gesture Elicitation Study*: Analyze results of GES. Some trends defined by a majority of participants about the touchpad gesture, will probably emerge. This is important not to restrict to our ideas only.

5. *Benchmark and Evaluation Methodology*: Follow a strict benchmarking procedure and comparing recognizers results together with exactly the same parameters and the same testing scenario. Their recognition rate and execution time rate are principally compared.

## 1.3 Outline of the thesis

To achieve the central goal of this thesis, the manuscript is separated into several chapters. Each chapter is resumed in tiny explanations below to provide a better overview.

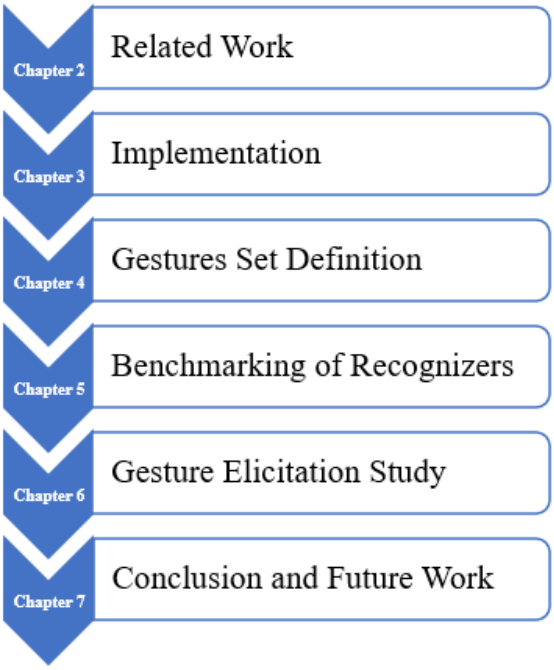



Figure 1.1: Chapters overview

- **Chapter 2: Related Work** presents past work about the objective mentions in this introduction. The chapter starts by describing the 3DTouchpad [34] and presents the basic Air+Touch gestures already defined in the SDK. Then, we briefly mention how the framework is working and what are the different functionalities already present. In this section, we also explore the Allen's interval algebra [3] as it offers some very interesting temporal combinations. Finally, we have a look through all the papers, articles which are talking about Air+Touch gestures found in the literature. We briefly explain their research, results and the device(s) used. We also draw up a catalog including all the 2D and 3D gestures from these resources.

- **Chapter 3: Implementation** is covering all our implementations for the thesis. It includes how we manage to acquire data from the touchpad and send it to the QuantumLeap Framework [50] or send it to our dataset recorder. It is also exploring all the different steps we made to integrate the touchpad

inside the framework. The last point of the chapter details different problems related to the use of multiple sensors at the same time inside QuantumLeap and how we managed to fixed them.

- **Chapter 4: Gestures Set Definition** overviews all the different 2D and 3D gestures which we decided to include inside our gestures set definition. A gesture description, a small drawing and the reference(s) where we took our inspiration, are associated to each gesture. After, we mention and analyze all the people who recorded the gestures to populate our different datasets. Finally, we talk about the different possible combinations that our gestures set offers. Especially, with allen's interval algebra mentioned in chapter 2.

- **Chapter 5: Benchmarking of Recognizers** tests the different recognizers implemented inside QuantumLeap [50] according to different parameters. All the parameters are explained in details. The chapter focus on the analysis of the recognizers results depending on two different testing scenarios: *User-Independent(UI)* and *User-Dependent(UD)*. The last point of the chapter is the confusion matrix and confusion wheel which help us to understand what are the common mistakes made by the recognizers.

- **Chapter 6: Gesture Elicitation Study** presents two GES performed by students and based on the 3DTouchpad. We explain how the two experiments were processed and what are the emerging results based on a majority of participants.

- **Chapter 7: Conclusion and Future Work** overviews the different contributions of the thesis. It also identifies all the points that could be interesting to focus in the future.

# Chapter 2

# Related Work

## 2.1 Overview

In this chapter, we are going to introduce the 3DTouchpad from Microchip [35] and then we are going to explain in details how it is working and what is its actual gestures set in section 2.2. Then in section 2.3, we are going to introduce QuantumLeap and then we are going to discuss briefly about all its different modules. Section 2.4 explains and mentions the 13 time relations of allen's interval algebra. Finally, section 2.5 is an overview of all the resources related to Air+Touch gestures found in the literature. A table regrouping all the different gestures from each resource is also built in the section.

## 2.2 The 3DTouchpad From Microchip

The 3DTouchpad [34] is one of the first development device that is able to acquire Air+Touch gestures for the PC market. It consists in an innovative mixed two technologies designed by Microchip [35]. They combine their 2D projected-capacitive (PCAP) touch solutions and their patented GestIC® technology for 3D gestures recognition [37].

Figure 2.1: Picture of 3DTouchpad[34]

The 3DTouchpad is an USB device that is able to acquire both 2D gestures and 3D gestures sequentially. It offers multi-fingers tracking, surface gestures and also a free space 3D gestures above the surface. The 3DTouchpad provides all the different features of a classical touch pad. Moreover, it combines all these features with 3D gestures. This results in a more efficient and more productive touch pad in comparison to the classical touch pad. Therefore, the device can retrieve Air+Touch data. The device is composed of five different electrodes. An electrode can be defined as a sensor responsible for gesture recognition. The device is composed of a central electrode which is also responsible for touch gesture recognition. In the case of a touch gesture, the electrode retrieves positions of each finger that is touching the device. The four other electrodes are placed according to the four cardinal points: East, West, North, South. The figure 2.2 shows how each electrode is placed according to the device.

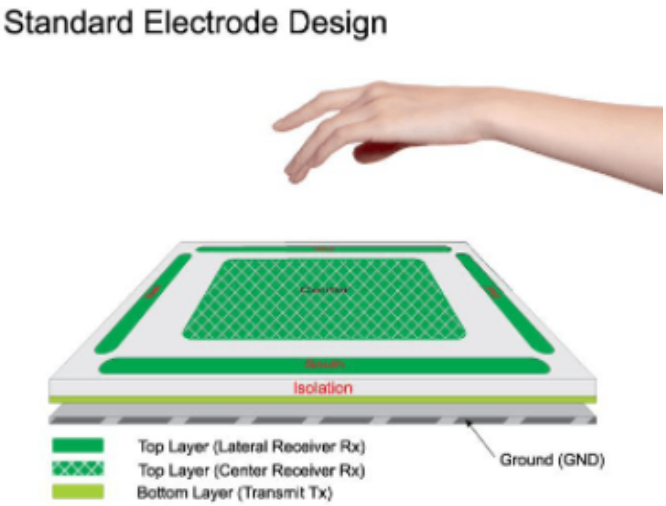



Figure 2.2: Electrodes of the 3DTouchpad [37]

These electrodes use frequencies and wavelength to compute the 3D data. When a finger enters in the "sensing" area, the electrical field distribution becomes distorted [37]. Thanks to the conductivity present in the human body, the different waves from each electrodes are modified. Therefore, each electrode can interpret these changes. The touchpad can gather all the data coming from each electrode and performs a triangulation [20] with all these data to compute the coordinates of the fingertip in space. The figure 2.3 shows how a finger can interfere with the electrodes.

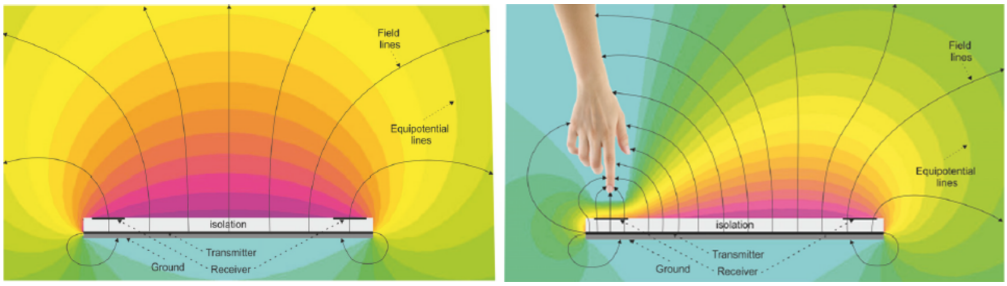
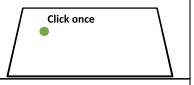
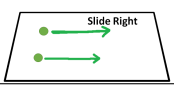



Figure 2.3: Sensing area of the 3DTouchpad [37]

Microchip implemented a Windows *Software Development Kit* (SDK) with the 3DTouchpad. It is implemented in C/C++ language. This SDK contains a Windows and Linux API which is called **HMI-API**. This api offers the possibility to developers to create their own C script to interact with the touchpad device. Therefore, it is possible to only retrieve the needed raw data. The raw data that we need for this thesis is the XYZ coordinates in the case of a 3D gesture is performed. Otherwise, we retrieve the XY coordinates from each finger touching the surface. In the 3D mode, each position is equal to a value between **0** and **65534** as it is encoded in a 16 bits integer. Whereas in the 2D mode, each x position is a value between **0** and **580** and each y position is a value between **0** and **380**. The values 380 and 580 represent the limits according to the dimensions of the touchpad (125mm x 95mm).

The SDK also provides some basics Air+Touch gestures. All these gestures are defined in table 2.1. The green color represents a 2D Touch gesture whereas the red color means that gesture is performed above the surface.

| Gesture Name | Dimension | Description | Drawing |
|---|---|---|---|
| One-Finger Movement | 2D | Move everywhere on the touching surface while you're still touching it. |  |
| One-Finger Tap | 2D | Single Click anywhere on the touchpad. |  |
| One-Finger Double Tap | 2D | Double Click anywhere on the touchpad. |  |
| Two-Finger Tap | 2D | Click once with two different fingers at the same time. |  |
| Two-Finger slide (Up/Down/Left/Right) | 2D | Swipe in one of the fourth cardinal direction with two fingers simultaneously. |  |

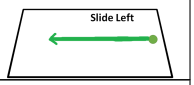
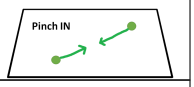
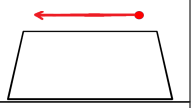
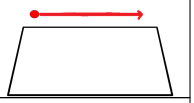
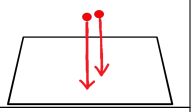
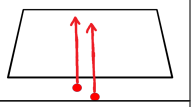
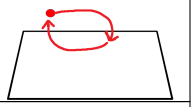
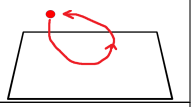
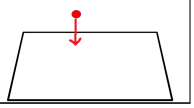
| | | | |
|---|---|---|---|
| Swipe from Right Edge | 2D | Swipe with one finger from East to West. |   |
| Swipe from Top | 2D | Swipe with one finger from North to South. |   |
| Pinch-to-Zoom | 2D | Pinch In on the touchpad with two fingers. |   |
| East -> West Flick | 3D | Swipe from East to West above the 3DTouchpad. |  |
| West -> East Flick | 3D | Swipe from West to East above the 3DTouchpad. |  |
| Double North -> South Flick | 3D | Swipe from South to North twice in a row above the touchpad. |  |
| Double South -> North Flick | 3D | Swipe from North to South twice in a row above the touchpad. |  |
| Airwheel Clockwise | 3D | Draw a circle in the clockwise sense above the touchpad. |  |
| Airwheel Counterclockwise | 3D | Draw a circle in the anti-clockwise sense above the touchpad. |  |
| Wake-up on Approach | 3D | Make an Air Swipe Down above the 3DTouchpad. |  |

Table 2.1: System-defined gestures [34]

As you can see on the table, the system-defined gestures are very limited and very basic gestures. It will be useful to create our own dedicated gestures set definition according to the 3DTouchpad. Our gestures set definition has to exploit the 3DTouchpad potential as much as possible and has to be based on several users.

The device retrieves both 2D and 3D data sequentially. Therefore, it is not possible to perform a 2D Touch gesture and a 3D gesture at the same time. Fortunately, we got the device for this thesis twice. So, with these two devices

working separately[1], it is still possible to perform both gestures simultaneously. This point is very important for the Air+Touch combinations.

## 2.3 QuantumLeap

*QuantumLeap Framework* [50] is designed as the separation layer between sensors and the application level. It was implemented in Javascript language by Arthur Sluÿters. Initially, QuantumLeap was developed for the leap-motion sensor [44]. But as it satisfies the ISO/IEC 25010 [1], it should not be a problem to integrate our 3DTouchpad inside. In our case, it seems very interesting to use a framework. Everything that is related to gesture recognition and other things needed, must be present inside QuantumLeap. On the paper, QuantumLeap provides a reusable software environment [45] for the different sensors which can acquire gesture data. At this time, there is only the leap Motion sensor integrated inside the framework. But, in the future, they will be probably plenty of different sensors present inside QuantumLeap as Leap Motion, 3DTouchpad, Smart-Ring, Microsoft Kinect and many others. The figure 2.4 represents the overall architecture of QuantumLeap Framework

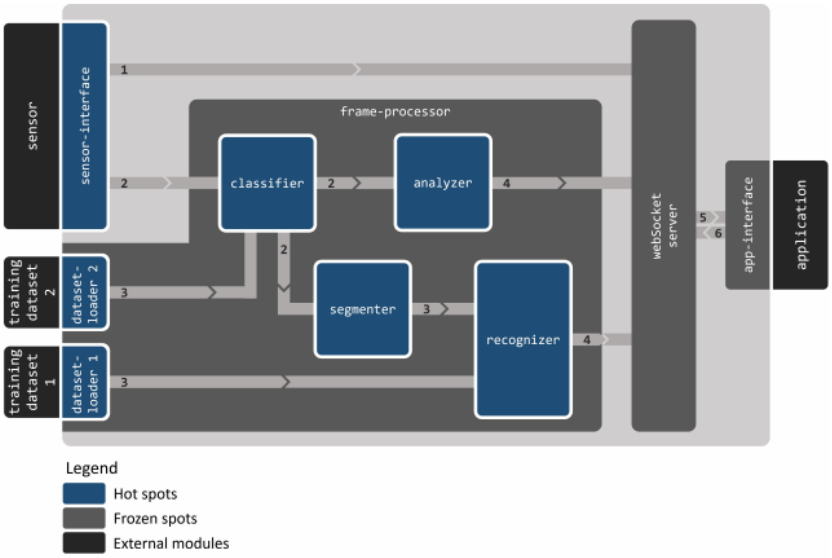



Figure 2.4: Architecture of QuantumLeap [50]

There are several hot spots inside the architecture. An *Hot spot* is a variable aspect of a parameter inside the domain [49]. Each one is defined with one or several

[1]Linux operating system doesn't allow to use the device twice. Therefore, it is only possible on Windows.

variations. As variations, we mean that there are several different recognizers, segmenters, analyzers and classifiers implementations inside the framework. Whereas the frozen spots, are completely fixed to increase the reusability of QuantumLeap. They do not have to change according to the configuration.

To make a clear and understandable description of all the different spots, We resume them in two different tables. Table 2.2 describes each frozen spot with a short description. Table 2.3 describes all the hot spots with all their possible variations.

## Frozen Spots

| **Name Spot** | **Description** |
|---|---|
| Frame | A frame represents standardized data extracted from the sensor. |
| Gestures and Gestures Sets | We can use any dataset inside the framework if it is adequately converted. *(StrokeData or PoseData)* |
| Websocket Server | Built to send response message to the application after treating data inside QuantumLeap |
| Frame Processor | Responsible for the organization and decide how the module has to extract the data. Also it determines if it is a pose or a gesture. |
| Application Interface | Used to make the transition between the application and the framework. |

Table 2.2: Frozen Spots modules from QuantumLeap.

## Hot Spots

| **Name Spot** | **Description** | **Variability** |
|---|---|---|
| Sensors | Collects the data from the selected device and treats it correctly inside QuantumLeap. | **3DTouchpad**[34]<br>**LeapMotion**[44] |
| Dataset | Transforms original data of a device from a dataset into an adequate form of the data to be treated inside QuantumLeap. | **Touchpad Dataset**<br>**LMC dataset** |
| Classifiers | Determines the hand position depending on data. | **$P3+** [55]<br>**GPSDa** |
| Analyzers | Responsible for the extraction of the data frame when a pose is identified. | **basic-analyzer** |

| | | |
|---|---|---|
| Segmenters | Identifies if the frames receives correspond to a gesture. | **Left-Hand Segmenter Zoning Segmenter Window Segmenter** |
| Recognizers | Depending on the data received, they have to determine the name of the gesture performed. | **3 cent[8, 9] Jacknife[52] $P3 $P3+ $P3+X $Q3 [56]** |

Table 2.3: Hot Spots modules from QuantumLeap.

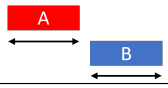
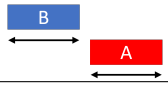
After having gone through all these workings of QuantumLeap, we can say that the framework is very useful because we won't need to implement the different modules to handle correctly the Air+Touch gesture recognition. Once several devices are integrated, it will be really interesting to see the different possible combinations between them. It can also be helpful for the future *Air+Touch* studies as it will be easy to combine 2D device and 3D device together and interpret their data.

If you are interested in deeper information about QuantumLeap, I suggest you to read the work of Arthur Sluÿters [50].

## 2.4 Allen's interval algebra

In this new section of this chapter, we are going to introduce the *Allen's interval Algebra*.

Allen's interval Algebra is an interval-based temporal logic. It was introduced by James F. Allen in 1998 in his original paper [3]. As he mentions in the paper, "the problem of representing knowledge and temporal reasoning arises in a wide range of disciplines, including computer science, philosophy, psychology and linguistics".As you can imagine, the focus is on the computer field and to be even more precise on Air+Touch gestures combinations. Allen defines **thirteen** time relations between two different events. For our thesis, these events can be seen as two different gestures. For the example on table 2.4, we can imagine that the event **A** is a *Swipe Left* gesture and the event **B** is an *Air Circle Right* gesture.

| Relation | Illustration | Interpretation |
|---|---|---|
| A $<$ B |  | A *before* B |
| A $>$ B |  | A *after* B |

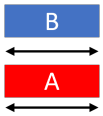
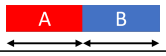
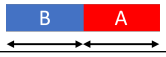
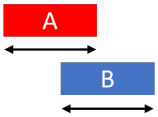
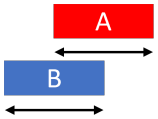
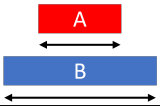
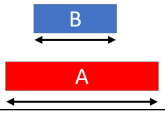
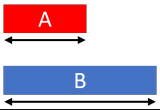
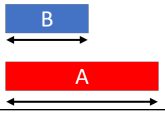
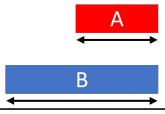
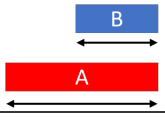
| | | |
|---|---|---|
| A = B |  | A *equal* B |
| A **m** B |  | A *meets* B |
| A **mi** B |  | A *met-by* B |
| A **o** B |  | A *overlaps* B |
| A **oi** B |  | A *overlapped-by* B |
| A **d** B |  | A *during* B |
| A **di** B |  | A *contains* B |
| A **s** B |  | A *starts* B |
| A **si** B |  | A *started by* B |
| A **f** B |  | A *finishes* B |
| A **fi** B |  | A *finished-by* B |

Table 2.4: The thirteen temporal relation defined by James Allen. [3]

As you can see, this definition has huge amount of possible combinations. As example, if we have just four different gestures, it results in **78** different temporal combinations. This is why the Allen's interval Algebra is very useful to define our possible Air+Touch combinations based on our gestures set definition.

Allen's interval algebra was already used in gesture recognition domain. In this paper [17], they defined gestures combinations with Allen's algebra. They used two smart rings. They analyzed the *2-ring* dataset coming from this elicitation study [16]. The result was that the participants were principally making gestures as *equal* for the Allen's algebra. This equal combination represents 69.9% of the dataset. Thus, it seems that starting the two gestures at the same time and finishing them also at the same time are the more common for the user. This result is interesting to put forward if we want to use a 3DTouchpad associated to another device (LeapMotion, SmartRing, ..) or simply to another 3DTouchpad.

## 2.5 History of Air + Touch

In this new section, we are going through all the different studies found in the literature which were focusing about Air+Touch gesture recognition. After this discussion, a kind of "*Air+Touch gestures*" catalog is dressed. This table resumes all the different gestures used in an *Air+Touch context* from the literature.

*S. Malik and Joseph Laszlo* published **Visual touchpad: a two-handed gestural input device** [32] in 2004. In their paper, they present the Visual Touchpad. It is a low-cost vision-based input device which captures two-handed gestures. It's a combination of two stereos hand tracking system which are capturing data on and above the touching surface. It also allows multi-touch gesture. For the 3D gestures, they defined some classical *pointing gestures* and some postures as *five-fingers* gesture(One or two handed). They combine these gestures with 2D touch gestures as *swipe, single/double tap, L-postures, PinchIn, PinchOut* and so on.

*Alexander Schick et al.* published **Extending Touch: Towards Interaction with Large-Scale Surfaces** [47] in 2009. Touch gestures for large-scale surfaces are not suitable as you have to make huge movements. Sometimes, this is not even possible because the top of the surface is too high. Thus, they developed a new extension to the touching surface based on 3D reconstruction with RGB camera [62]. Users can therefore interact in a touch and air environment with the device. They implement different gestures for their experiment as *touching swipe(2D)* and *Pointing finger movement(3D)*. They also make a comparison study about the feelings of the participants on *Touch-only*, *Pointing-Only* and *Touch-and-pointing* gestures. They preferred the touching and pointing because it was more intuitive, easy to use, more accurate than *Pointing-Only* and because they could

reach all parts of the screen easily and with less efforts. The mixed environment makes the use less difficult and more comfortable for the different participants.

*Michael Ortega* and *Laurence Nigay* published **AirMouse: Finger Gesture for 2D and 3D Interaction** [42] in 2009. They are analyzing the *AirMouse* technique. It consists in the use of fingers over the keyboard of a laptop. *AirMouse* allows interacting in 2D and/or 3D. It uses two cameras to track the different fingers positions. The device captures only the fingers positions but not the palm. They implemented only some pointing gestures to test their devices in 2D and 3D. They made an interesting statement about their work: AirMouse looks promising in comparison with the *visual Touchpad* [32] which is explained before.

*Jörg Muller et al.* published **MirrorTouch: Combining Touch and Mid-air Gestures for Public Displays** [40] in 2014. They focused on combining *Air+Touch gestures* on huge terminal that we usually found on the train station or fast-food. They developed a touching screen which was able to detect touching gestures and mid-air gestures of the participants. They defined very simple gestures as the *Single Tap, Swipe while touching* for the touch gestures. On the display, the participants could see their body representation. *MirrorTouch* were proposing some games where you have to move the different parts of your body for example to catch a falling cube to win some points. They used principally pointing gestures for mid-air interactions. They put several displays in different places without any using tutorial. The people were intriguing by the displays. So, they often started to touch before performing mid-air gestures, even if this was not what the researchers hoped for. The writers also point out how the combinations of the two environments are complementary and very interesting.

*Xiang 'Anthony' Chen et al.* published **Air+Touch: Interweaving Touch & In-Air Gestures** [11] in 2014. The goal of the paper was to create an Air+Touch gestures vocabulary and to show how complementary they are. They used a smartphone associated to a depth camera. Touch gestures were used to select targets whereas mid-air gestures to move the targets. Air gestures add some expressiveness to the touch gestures. If they were used alone, they would loose a lot of meaning. They defined three ways of making a gesture in their gestures set. They are called *Before Touch, Between Touch, After Touch.* This allows recognition engine to search for a smaller window. *Before Touch* starts with an in-air gesture which is finishes by a touch. *Between Touch* starts with a touch gesture, follow with an in-air gesture and then finishes by another touch gesture. *After Touch* starts with an in-air gesture and finishes with a touch gesture. They defined five ways to interact with their device: *corner, circle, pigtail, zigzag, spike.* This leads users to more powerful and more expressive interactions.

*Hansaem Lee* and *Juseok Park* published **Hand Gesture Recognition in Multi-space of 2D/3D** [27] in 2015. They proposed surface gesture recognition

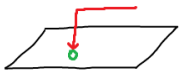
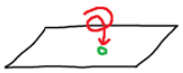
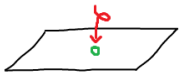
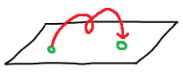
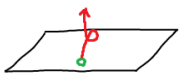
using touch and mid-air gestures. They used a touch screen combined with a depth sensor which are collecting 3D spatial information. They focused on the *Pinch In, Pinch Out* gestures. They defined a mixed-space pinch. It consists in touching the surface with two separate fingers. Then, you pinch in above the surface (mid-air). This gesture can be done in the opposite sense. After that, they focused on the recognition rate which shows interesting result. The mixed-space pinch has a better rate than 3D only but not than 2D Only. This lead us to think that *Air+Touch* can have a better accuracy than mid-air gestures only.

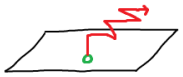
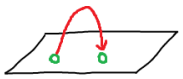
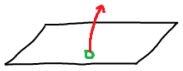
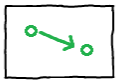
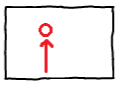
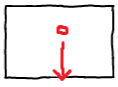
*Andrew Bragdon et al.* published **Code Space: Touch + Air Gesture Hybrid Interactions for Supporting Developer Meetings** [6] in 2011. They focused on Air+Touch interactions in a meeting environment. They used multi-touch devices as a smartphone combined with a Microsoft Kinect sensor [61]. They used in-air pointing as *Open palm, Open palm with spreads fingers, Pinch gesture, Pointing with a finger*. They combine these Air gestures with classical smartphone gestures as *single Tap, Flick in a direction, Pinch gestures*. The participant were showing interest about Code Space and their results looked promising. The *Air+Touch* once mastered could really increase the comfort and the efficiency of a meeting. Unfortunately, the study were limited to only one company. Thus, it's not representative of every company.

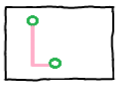
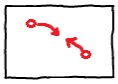
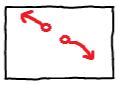
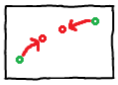
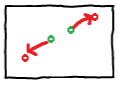
*Carole Plasson et al.* published **3D Tabletop AR: A Comparison of Mid-Air, Touch and Touch+Mid-Air Interaction** [43] in 2020. In this paper, they used a *Tabletop Augmented Reality*(AR) system. It combines touch interactions and 3D modeling. Their goal was to compare 3D mid-air interactions with mixed touch and mid-air gestures. They designed three different way to interact with device, but only one is interesting for this thesis. They called it the *Mixed* technique. A gesture is composed in two parts. The first one consist in a simple touch to select an object. Then the user can move the selected object by moving his finger above the tabletop. After comparing the results of the three scenarios, the mixed scenario is more accurate and faster than the mid-air only scenario. Air+Touch gestures combine the advantages of each environment but also reduce some of their respective limitations.

*Nicolai Marquardt et al.* published **The Continuous Interaction Space: Interaction Techniques Unifying Touch and Gesture on and above a Digital Surface** [33] in 2011. They introduced the continuous interaction space term. This consist in the combinations of (multi-)touch and hand gestures above the surface. They wanted people to consider the both space working together where they could easily move from a touch to a hand gesture above the surface without loosing expressiveness. They also provided some new interaction techniques merging both modalities. All the different gestures defined in the paper are listed in the table 2.5.

All the different papers described their own gestures associated with their device. Some gestures are quite unique whereas some are common to several different papers. All the gestures defined are listed in table 2.5. Each gesture is defined by their name, by the dimension in which the gesture is performed (2D, 3D, 2D+3D), by a brief description, by an explicit drawing and finally by the reference(s) where the gesture was found. The green color in the drawing represents a 2D Touch gesture whereas the red colors represents a 3D gesture.

| Gesture Name | Space | Description | Draw | Article |
|---|---|---|---|---|
| Corner Before Touch | 2D+3D | Finger contains a 90-degree angle in air before going down to touch the surface. |  | [10] |
| Corner Between Touch | 2D+3D | Finger contains a 90-degree angle in air after going up from surface and then goes down to touch the surface. |  | [10] |
| Corner After Touch | 2D+3D | Finger contains a 90-degree angle in air after going up in air from the surface. |  | [10] |
| Circle Before Touch | 2D+3D | Finger goes down to touch the surface after drawing cyclical paths in air. |  | [10] |
| Circle Between Touch | 2D+3D | Finger goes up from the surface and then cyclical paths in air. Finally, finger goes down to touch the surface. |  | [10] |
| Circle After Touch | 2D+3D | Finger goes up from the touch surface and draws cyclical paths in air. |  | [10] |
| Pigtail Before Touch | 2D+3D | Finger draws a small loop along its in-air trajectory. Then goes down to touch the surface. |  | [10] |
| Pigtail Between Touch | 2D+3D | Finger goes up from the surface and then draws a small loop along its in-air trajectory. Finally, finger goes down to touch the surface. |  | [10] |
| Pigtail After Touch | 2D+3D | Finger goes up and then draws a small loop along its in- air trajectory. |  | [10] |

| | | | | |
|---|---|---|---|---|
| Zigzag Before Touch | 2D+3D | Finger makes sharp 'turns' in air and then goes down to touch the surface. |  | [10] |
| Zigzag Between Touch | 2D+3D | Finger goes up from touch surface and then make sharp 'turns' in air. |  | [10] |
| Zigzag After Touch | 2D+3D | Finger goes up from the touch surface and then makes sharp 'turns' in air. |  | [10] |
| Spikes Before Touch | 2D+3D | Finger reaches a 'special' air position during its movement and then goes down to touch the surface. |  | [10] |
| Spikes Between Touch | 2D+3D | Finger reaches a 'special' air position during its movement and then goes down to touch the surface. Finally, it goes down to touch the surface. |  | [10] |
| Spike After Touch | 2D+3D | Finger goes up from the touch surface and then reaches a special air position during its movement. |  | [10] |
| Single Tap | 2D | "Click" on the touching surface. |  | [47, 32, 40] [42, 6, 43] [33] |
| Swipe | 2D | Move the touching finger from a position to a new one. |  | [47, 32, 40] [42, 6] [33] |
| Pointing with finger | 3D | Pointing in front of the 'device' without moving finger for a few times. |  | [47, 32, 40] [42, 6, 43] [33] |
| Swipe While Pointing | 3D | Move the pointing finger from an initial position to another one. |  | [47, 32, 40] [42, 6, 33] |
| Air Swipe Up | 3D | Withdraw the finger from the device and so increase Z axis value. |  | [47, 32, 43] [33] |

| | | | | |
|---|---|---|---|---|
| Double tap | 2D | Tap twice successively on the touching surface. |  | [32] |
| L-Posture with two fingers | 2D | Touch the surface with the index finger and the thumb as making a kind of L with both fingers. |  | [32, 33] |
| Zoom Pinch In | 2D/3D | Pinch in the thumb and index together on/above the surface. |  | [32, 27, 6] [33] |
| Zoom Pinch Out | 2D/3D | Pinch out the thumb and index together on/above the surface. |  | [32, 27, 6] |
| Cross | 2D | Draws an 'X' on the surface. |  | [32] |
| Clockwise Rotation | 2D | Rotations of two fingers in clockwise sense. |  | [32] |
| Five-finger Gesture with Both Hand | 3D | Spread your fingers from the two hands and stay above the surface. |  | [32] |
| Mixed-Space Pinch In | 2D+3D | Pinch In gesture starting on the surface and finish above the surface(3D). This can be done also started in 3D and finished in 2D. |  | [27] |
| Mixed-Space Pinch Out | 2D+3D | Pinch Out gesture starting on the surface and finish above the surface(3D). This can be done also started in 3D and finished in 2D. |  | [27] |
| Open Palm | 3D | Open the palm of one hand while the hand was pointing. |  | [6] |
| Open Palm Swiping | 3D | Move your hand in a wanted direction while your palm is still open. |  | [6] |

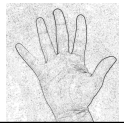
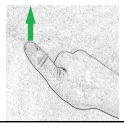
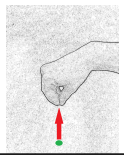
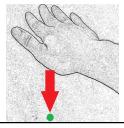
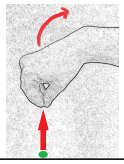
| | | | | |
|---|---|---|---|---|
| Open Palm and Spread Fingers | 3D | First, open the palm and then spread the fingers. |  | [6] |
| Flick Up/Down With Thumb | 2D | Flick up or down with the thumb on the touching surface. |  | [6] |
| Hand Waving | 3D | Wave your hand above the surface. |  | [33] |
| Pick-Up | 2D+3D | Pinch two fingers and touch the surface. Then, lift your hand above the surface as you were picking up something. |  | [33] |
| Dropping | 2D+3D | Bring your hand closer to the surface as an Air Swipe Down. Then, touch the surface |  | [33] |
| Pouring | 2D+3D | Make a pick up gesture. Then made a movement as you were pouring water from a water bucket. |  | [33] |

Table 2.5: All the gestures found in the literature mentioned earlier

After reading all these articles and analyzing their gestures, we can see some emerging facts:

- *Air+Touch is more accurate than mid-air only.*
- *Air+Touch reduces 2D limitations and 3D limitations.*
- *Air+Touch results are very promising.*
- *Air+Touch is more suitable for users than for the mid-air.*

However, there are still points to improve in order to increase the amount of work related to this thematic. Researchers often use a combination of devices, i.e. a 2D device associated with a 3D device. There is a clear lack of devices that are able to acquire the two different types of gestures. This is why the choice of

the 3DTouchpad is really interesting and very promising. Trough our thesis, we may prove that it could be very interesting to work on the development of new Air+Touch devices.

## 2.6 Conclusion

We went through a lot of different Air+Touch studies. They are very interesting and their results seem very promising. But as mentioned in the introduction, there are some real issues about devices. Each study describes complex devices which are costly and not easy to use. This strengthens the purpose of this thesis as we are using a cheap and easy device. We are going to integrate all gestures from figure 2.5 that are properly fitting with our device in our gestures set definition.

# Chapter 3

# Implementation

## 3.1 Overview

The chapter is divided into four sections. Section 3.2 overviews how we managed to retrieve the data from the 3DTouchpad device. Then section 3.3, explains in details how we created our C websocket to send the touchpad data. Section 3.4 overviews a JavaScript script which is saving data to create gestures samples. Finally, section 3.5 explains how we managed to integrate 3DTouchpad inside QuantumLeap. It also mentions some issues related to the use of multiple sensors at the same time and how me managed to fix these issues.

In appendix B, you can find all the different relevant implementations to this thesis and also a user tutorial to use our 3DTouchpad script on Windows OS.

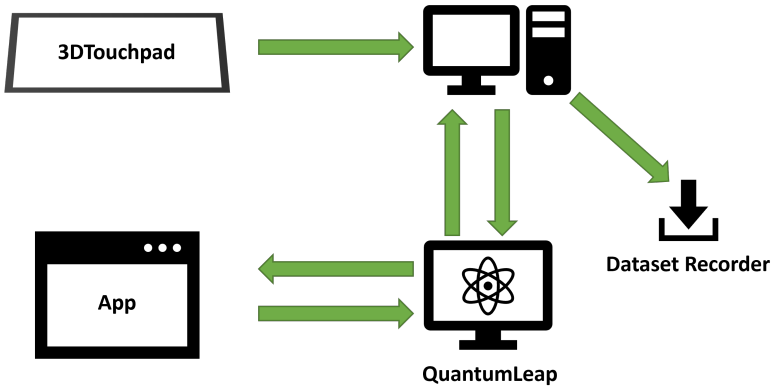



Figure 3.1: Visual representation of the implementation architecture. 1)Retrieve the touchpad data using a c script. 2)Send the data trough a webSocket to QuantumLeap or to a *Dataset Recorder*. 3)QuantumLeap analyzes and interprets the data. Then, it send the recognized gestures to the application if one was detected.

## 3.2 3DTouchpad Implementation

The first step of the implementation is to configure the Touchpad according to the HMI-API present in the SDK [34]. It is important to pay attention to these configurations that allow us to keep the relevant ones. This leads to the effect of not overloading the touchpad hardware and optimize the communication channel.


Figure 3.2: Touchpad Implementation

### 3.2.1 Initialisation

The touchpad has to be set-up in its mixed mode. It enables 2D and 3D signal processing. We just need to retrieve the raw data of the 2D/3D sensors. So, only these functions are required for a clean initialisation:

- **hmi_initialize**: initializes all variables and required ressources of hmi.
- **hmi_open**: opens connection to the device.
- **hmi2d_set_com_mask(hmi, com_flags, hmi2d_com_all)**: enables 2D/3D message communication. "com_flags" is an int value corresponding to the mixed mode in our case.
- **hmi2d_set_active_mask(hmi, hmi2d_active_gesture_recognition, hmi2d_active_all)**: disables all active features.
- **hmi2d_set_operation_mode(hmi, hmi2d_mixed_mode)**: sets 2D/3D mixed operation mode.
- **hmi3d_set_auto_calibration(hmi, 1)**, **hmi3d_select_frequencies(hmi, hmi3d_all_freq)**, **hmi3d_set_touch_detection(hmi, touch_detect)**, **hmi3d_set_approach_detection(hmi, 0)**: the four functions are necessary to reset the device to default state before processing.
- **hmi3d_set_output_enable_mask(hmi, stream_flags, 0, hmi3d_DataOutConfigMask_OutputAll)**: sets output-mask for streaming.

### 3.2.2 Retrieving the data

Once the touchpad is well configured, we can start to retrieve the data. There are two type of data: the 2D and the 3D.

#### 3D Data

While 3D data is detected and the Touchpad is not processing other data at the same time, it stores the different **X, Y, Z** in "pos" struct as you can see on the following code. The count value is used as the timestamp of the data.

```
char data[100];

while (count < iterations_number) {
 while (!hmi3d_retrieve_data(hmi, 0) && count < iterations_number) {
   if (pos != NULL) {
       //  Output the position
       int x = pos->x;
       int y = pos->y;
       int z = pos->z;
       Sleep(30);
```

```
11          count = count + 1;
12          }
13    }
14    while (hmi2d_retrieve_data(hmi) == HMI_NO_ERROR &&count <
       iterations_number) {
15    // ... 2D data
16    }
17 }
```

#### 2D data

If someone touches the touchpad, it starts sensing on the 2D surface. Therefore, the second while statement is true and all raw data needed are inside the "fingers" global variable. It contains all **X, Y** positions of each finger. The *output_fingers* function separates each data position associated to its relevant finger into a table. We limited the number of maximum fingers to three for this thesis as it has no sense to use it with more fingers on a so small surface.

```
1  char data[100];
2
3  while (count < iterations_number) {
4   while (!hmi3d_retrieve_data(hmi, 0) && count < iterations_number) {
5      // .... 3D Data
6   }
7   while (hmi2d_retrieve_data(hmi) == HMI_NO_ERROR && count <
       iterations_number) {
8     if (fingers != NULL) {
9         output_fingers( fingers, bool, count, s);
10        Sleep(30);
11        count = count + 1;
12    }
13  }
14 }
```

### 3.2.3 End of execution

It is important to reset the device and free all the resources allocated at the end of the execution. Otherwise, it can create hardware problems. A nice way to end the execution is with these functions:

- **hmi2d_set_com_mask(hmi, hmi2d_com_default, hmi2d_com_all)**: resets the default communication mode.
- **hmi2d_set_active_mask(hmi, hmi2d_active_default, hmi2d_active_all)**: resets the default features.

- **hmi2d_set_operation_mode(hmi, hmi2d_mixed_mode)**: resets to mixed operation mode.
- **hmi_close(hmi)**: closes connection to device.
- **hmi_cleanup(hmi)** and **hmi_free(hmi)**: releases further resources that were used by hmi

## 3.3 C Websocket

After retrieving the raw data, we need to send it to QuantumLeap or to our dataset recorder. We have implemented the 3DTouchpad script on Windows and Linux. Therefore, we need to use two different c libraries according to the operating system to build our websocket. We use the *winsock2.h* [13] library for Windows or *Libwebsockets.h* [18] on Linux. The sent data are exactly the same through each socket.

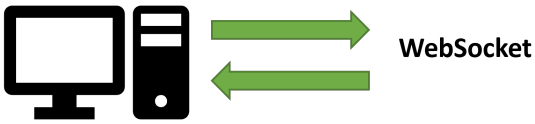



Figure 3.3: C WebSocket

### 3.3.1 Sent Data format

The sent data to the socket is a string composed of a Json structure. If the type of the data is *3D*. The Json string is such as:

```
1 {"type":"3D","count":15,"x":695,"y":456,"z":4898}
```

The field *"type"* mentions if it is a 3D or 2D data. "count" can be defined as the timestamps of the data. The three *"XYZ"* positions are the values recorded by the sensor and it can go to a maximum of 65534.
The 2D data structure is almost the same. However, it depends on how many fingers are touching the device. The three following lines represent the three different ways of sending 2D data according to the number of finger(s) used.

```
1 {"type":"2DTouch1","count":48,"x":[19],"y":[167]}
2 {"type":"2DTouch2","count":48,"x":[19,56],"y":[167,87]}
3 {"type":"2DTouch3","count":48,"x":[19,56,49],"y":[167,87,74]}
```

The *"type"* is the concatenation between the string "2DTouch" and the amount of fingers which are touching the device. This structure makes it easier to process the data later as we directly know how many fingers are touching the device. The *count* is the data timestamps. The *"XY"* fields are now a table composed of the different values of each finger. For example, *x[0]* and *y[0]* represent the XY positions of the first finger, *x[1],y[1]* are the positions of the second finger.

### 3.3.2 Linux

Now that we retrieved the data and standardized it, we still have to send it inside a webSocket. For the Linux operating system, we decide to use the Libwebsockets.h [18] which is a free software and open-source using the MIT licence [30]. It is also known as Lws. This library is used for implementing modern network protocols using nonblocking event loop. It provides an interesting abstraction of the lower layer. Thanks to Lws, we can implement both a websocket server and a websocket client. But in our case, we only need to implement a client. To create our websocket client, we can decompose the code into three important steps:

- Configuration of the connection information variable
- Callback function
- Sending data

#### Connection information

Below this paragraph, you can find the code example used which is setting-up our client. All the parameters are quite complex if you are not familiar with it. The address and the port variables are the main interest in this code example. "Address" is equal to *localhost* in our example but it can be equal to any address. It is the same for the "port" variable but I suggest you to not use the ports which are already reserved [12].

```
1  struct lws_context *context_socket = lws_create_context( &info_socket
       );
2
3  struct lws_client_connect_info ccinfo = {0};
4    ccinfo.context = context_socket;
5    ccinfo.address = "localhost";
6    ccinfo.port = 8081;
7    ccinfo.path = "/";
8    ccinfo.host = lws_canonical_hostname( context_socket );
9    ccinfo.origin = "origin";
10   ccinfo.protocol = protocols[PROTOCOL_EXAMPLE].name;
11   web_socket = lws_client_connect_via_info(&ccinfo);
```

```
}
```

### Callback function

A callback function is a function which is inside an other function as an argument. It increases the code reusability and clearness. It avoids to handle the 5 different events in our code example. Callback function is called each time an event is triggered or when there is a timeout. The event is stored inside the *reason* variable. It allows the code to jump easily from one case to another one. This is very important for the communication process. It provides an optimized way to handle websocket status and to send data.

```
static int callback( struct lws *wsi, enum lws_callback_reasons
    reason, void *user, void *in, size_t len )
{
    switch(reason){
        case LWS_CALLBACK_CLIENT_ESTABLISHED:
        // Connection with the server is establish
        case LWS_CALLBACK_CLIENT_RECEIVE:
        // Handle the different incomming  message
        case LWS_CALLBACK_CLIENT_WRITEABLE:
        // Writing to the socket is allowed
        case LWS_CALLBACK_CLOSED:
        // Connection is closed
        case LWS_CALLBACK_CLIENT_CONNECTION_ERROR:
        // Can't connect to the server
        default:
            break;
    }
    return 0;
}
```

### Sending data

The c code example below is showing how to send data in the best way. First we have to ask if we can write on the socket. If our request is accepted, we initialize the buffer and the char* with enough bits. We are using the *pre-pradding* and the *post-padding* to make sure to have enough bits and not produce a buffer overflows [28]. This can lead to serious vulnerability problems. Then, we can send our data thanks to **lws_write** function. The other function **lws_service** is a timeout waiting for "x" milliseconds to avoid segmentation fault problems. After all these steps, the data is correctly sent through our dedicated websocket and is waiting to be received by a websocket server.

```
lws_callback_on_writable(web_socket); //Ask if we can write
```

```
2
3 unsigned char buf[LWS_SEND_BUFFER_PRE_PADDING +
    EXAMPLE_RX_BUFFER_BYTES + LWS_SEND_BUFFER_POST_PADDING];
4 unsigned char *p = &buf[LWS_SEND_BUFFER_PRE_PADDING];
5 size_t n = sprintf( (char *)p, "{\"type\":\"3D\",\"count\":%d,\"x\":%
    d,\"y\":%d,\"z\":%d}",count,pos->x,pos->y,pos->z);
6 lws_write( web_socket, p, n, LWS_WRITE_TEXT );
7
8 lws_service( context_socket, /* timeout_ms = */ 10);
```

### 3.3.3 Windows

Unfortunately, *Libwebsockets* [18] is not supported on visual studio [21] which we had to use to handle c script on Windows. However, There is an other library called *Winsock2.h* [13]. This library can build websocket client and server. To create our own websocket client, we can decompose the implementation into the three following steps:

- Initialize the socket
- Fill the socket information
- Send the data

#### Initialize the socket

We have to create and allocate the different necessary resources to build our websocket client. *WSAStartup* is used to initiate and to launch the **WS2_32.dll** which contains all the needed library structures and implementations. Once the dynamic link library is ready to be used, we can create the socket as the following code example. The AF_INET means that the communication socket uses IPV4 address only.

```
1 WSADATA wsa;
2 SOCKET s;
3 char* message;
4
5 printf("\nInitialising Winsock...");
6 if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0){
7     printf("Failed. Error Code : %d", WSAGetLastError());
8     return 1;
9 }
10
11 printf("Initialised.\n");
12
13 //Create a socket
```

```
14 if ((s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == INVALID_SOCKET
       ){
15     printf("Could not create socket : %d", WSAGetLastError());
16 }
17
18 printf("Socket created.\n");
```

### Socket Information

Once we have created the socket, we have to fill the server information. We need to define the three fields of the *sockaddr_in server* variable: the server address family, the server ip address and the port number on which the server is listening. Once everything is correctly filled, we can connect our client socket to the server socket.

```
1 struct sockaddr_in server;
2
3 server.sin_family = AF_INET;
4 server.sin_port = htons(port_number);
5 server.sin_addr.s_addr = inet_addr("127.0.0.1");
6
7 //Connect to remote server
8 if (connect(s, (struct sockaddr*)&server, sizeof(server)) < 0){
9     puts("connect error");
10    return 1;
11 }
12
13 puts("Connected");
```

### Send data

To send our data buffer, we just have to use the *send* function with these three arguments: **s** is the socket structure, **data** is our string buffer and **n** is the number of bits to send. We can iterate over these function to send data periodically.

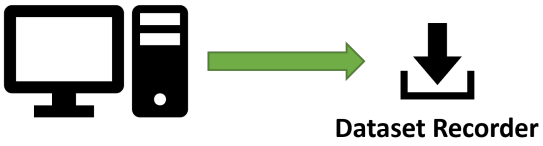
```
1 int n = sprintf(data, "{\"type\":\"3D\",\"count\":%d,\"x\":%d,\"y
    \":%d,\"z\":%d}", count, pos->x, pos->y, pos->z);
2 send(s, data, n,0);
```

Once the while iteration ends, we have to free all the allocated resources. It can easily be done by these two code lines:

```
1 closesocket(s);
2 WSACleanup();
```

## 3.4 Dataset Recorder

The goal of the dataset recorder is to save the raw data sent by the websocket client. It records all the data received in a JSON file and classifies it in a dedicated folder. It's important because these samples are necessary to train the different recognizers and therefore to have better results.




We decide to use the **'net'** [29] library to implement our Javascript websocket server. Net library provides asynchronous network API for creating stream-based TCP servers or clients. In our case, we only consider the server websocket. The first step is to create the server (CreateServer) that listens on a dedicated port. In our following code example, the port number is equal to 5000. The ip address is equal to localhost address by default. Now that the socket is created, the script is waiting until we receive any touchpad data. The data is stored in a **JSON format**. When the C websocket closes the connections, our script creates a file in the right folder containing all the received sensed data.

```
net = require('net');
const { table } = require('console');
var fs = require('fs');
var dataset = {
    table: []
}
var count = 0;

net.createServer(function (socket) {

  socket.on('data', function (data) {
        console.log(JSON.parse(data));
        dataset.table.push(JSON.parse(data))
  });
  socket.on('error', function(e){
        console.log(e);
  });

  socket.on('close', function(e){
        var json = JSON.stringify(dataset)
        fs.writeFile('droitier/participant7/AirSwipeUp.json'
        , json, function(err) {
```

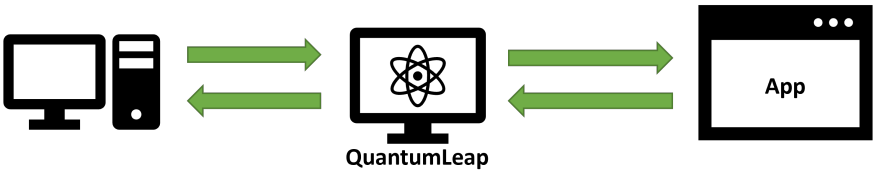
```
            if (err) throw err;
            console.log('Record number${count}');
            }
        );
        dataset.table = []
        count = count + 1
  });

}).listen(5000);

console.log("Chat server running at port 5000\n");
```

## 3.5 QuantumLeap Integration

Now that we can send the 3DTouchpad data on a websocket, we can discuss about the different integrations needed inside QuantumLeap [50] and how the data is processed.




As a quick reminder, QuantumLeap is a framework on gesture recognition. It was initially implemented for the LeapMotion [44] sensor. QuantumLeap is also the *PhD thesis* of Arthur Sluÿters. The framework is constantly evolving. It is becoming more and more generic but also more intuitive. I have had the honor of working on it and understanding every little thing about it. Over time, this can become revolutionary for gesture recognition. This will save a lot of time as everything is centralized inside the framework. As the touchpad was only the second device to be integrated, we managed to solve some bugs or parts which were too representative of the LeapMotion.

### 3.5.1 3DTouchpad Integration

The integration of our device in QuantumLeap can be separated into four parts. We mention each part in this subsection.

### Articulations Definition

Articulations are used to separate the different types of data from a device which have not to be processed together inside QuantumLeap. In our case, it is necessary to separate data depending on if it is a 2D or 3D data. We also separate 2D data according to the amount of fingers used. Therefore, we decide to create this four different articulations for the 3DTouchpad inside the framework:

- *3D*: Articulation containing only the sensed 3D data.
- *2DTouch1*: Articulation containing only the sensed 2D data when it is touched by only one finger.
- *2DTouch2*: Articulation containing only the sensed 2D data when it is touched by only two fingers.
- *2DTouch3*: Articulation containing only the sensed 2D data when it is touched by only three fingers.

### 3DTouchpad Dataset

The first step of the integration is to provide an adequate 3DTouchpad dataset with enough templates for each gesture. The next chapter details all the chosen gestures for our gestures set. Therefore, we won't discuss about it in this chapter. However, you can find all our samples associated to the touchpad **here** on github.

### 3DTouchpad Dataset Loader

This 3DTouchpad dataset loader consists in:

- Retrieving all the present samples inside the dataset folder.
- Separating each sample point by point
- Sending each point of a sample to its dedicated articulation.
- Each gesture articulation has to have exactly the same number of points. Therefore, we add some basic point (X:10,y:10,Z:10,t:0) to the other articulations to avoid recognizers errors.
- Returning the Gesture set variable that gathers all the gesture samples and their articulations.

The loader implementation can be found in appendix B.4.1 . The recognizers train themselves on the received data from this loader. Each recognizer easily processes the data as each loader from QuantumLeap has to be built this way.

### 3DTouchpad Sensor

Our sensor implementation can be find in appendix B.4.2. Here on github, you can find all the relevant information to the sensor configuration. The index.js file defines how the sensor is working inside QuantumLeap. We can separate the code into two main parts: the websocket client creation and how data is sent inside the framework.

The websocket client is built using the *.net library* [29] as presented in a previous section. The creation occurs inside the connected function. The goal is to listen to the dedicated port defined by the user or by default. Each time a data has been received, function updates the **this.lastframe** variable with received data.

```
connect(){
    this.net.createServer( (socket) => {
            console.log('TouchPad␣Connected')
            socket.on('data', (data) => {
                try{
                this.lastframe = JSON.parse(data);
            }
            catch(error){
            }
        });
        socket.on('error', function(e){
            console.log('3DTouchpad␣socket␣is␣disconnected');
    });

   }).listen(this.port);
}
```

Now, we are going to discuss about the getPoint function, which is the second important part. The framework is calling this function every x seconds. This function returns the following JSON:

```
1 { hasData: true, points: points, appData: {}};
```

HasData is set to true if points variable contains some value in its table. If no data were received between two calls and so this.lastframe is still undefined. The following JSON is returned:

```
1 { hasData: false, points: [], appData: {}};
```

So, the framework knows exactly when it has to process the data. We also need to use this small **addMissingPoints** "const function" inside the getPoint function. Its goal is to complete the other articulations with basics points where the received data are not relevant to them. Thus, it is avoiding the recognizers problems when samples do not have the same amount of points in each articulation.

```
const addMissingPoints = (field,points) => {
```

```
    let basic_point = new Point(10,10,10,0);
    for(let i = 0; i < articulation_field.length; i++){
        if (field !== articulation_field[i]){
            points.push({
                name : '${articulation_field[i]}',
                point : basic_point
            })
        }
    }
}
```

The last part is about the explanations of how the different points are pushed inside the table. Inside *this.lastframe*, we get the type field from our data. It can be 3D, 2D with one finger, 2D with 2 fingers or 2D with 3 fingers data. The code has a different shape according to these fields and can be explained as follow:

- **type="3D"**: We receive one 3D point in lastframe. We push this point inside 3D articulation. Then, we push one basic point to each articulation.
- **type="2DTouch1"**: We receive one 2D point in lastframe as it was touched with 1 finger. We push this point inside 2DTouch1 articulation. Then, we push one basic point to each articulation.
- **type="2DTouch2"**: We receive two different 2D points in lastframe as it was touched with 2 fingers. We push these points inside 2DTouch2 articulation. Then, we push **two** basic points to each articulation.
- **type="2DTouch3"**: We receive three different 2D points in lastframe as it was touched with 3 fingers. We push these points inside 2DTouch3 articulation. Then, we push **three** basic points to each articulation.

This is how our getPoint function is implemented for sending the received data depending on their articulations:

```
getPoints(timestamp){
let points = [];
if(this.lastframe.type === "3D"){
    points.push({
        name: '3D',
        point: new Point(this.lastframe.x,this.lastframe.y
        ,this.lastframe.z,this.lastframe.count)
    })
    addMissingPoints("3D",points)
}
else if(this.lastframe.type ==="2DTouch1"){
    points.push({
        name : '2DTouch1',
        point: new Point(this.lastframe.x[0],
```

```
            this.lastframe.y[0],0,this.lastframe.count)
        })
        addMissingPoints("2DTouch1",points)
    }
    else if (this.lastframe.type === "2DTouch2"){
        points.push({
            name : '2DTouch2',
            point: new Point(this.lastframe.x[0],
            this.lastframe.y[0],0,this.lastframe.count)
        })
        addMissingPoints("2DTouch2",points)
        points.push({
            name : '2DTouch2',
            point: new Point(this.lastframe.x[1],
            this.lastframe.y[1],0,this.lastframe.count)
        })
        addMissingPoints("2DTouch2",points)
    }
    else if (this.lastframe.type === "2DTouch3"){
        points.push({
            name : '2DTouch3',
            point: new Point(this.lastframe.x[0],
            this.lastframe.y[0],0,this.lastframe.count)
        })
        addMissingPoints("2DTouch3",points)

        points.push({
            name : '2DTouch3',
            point: new Point(this.lastframe.x[1]
            ,this.lastframe.y[1],0,this.lastframe.count)
        })
        addMissingPoints("2DTouch3",points)

        points.push({
            name : '2DTouch3',
            point: new Point(this.lastframe.x[2]
            ,this.lastframe.y[2],0,this.lastframe.count)
        })
        addMissingPoints("2DTouch3",points)
    }

    this.lastframe = undefined
```

### 3.5.2 Use of QuantumLeap

Now that the touchpad is integrated inside QuantumLeap framework, there is still one missing step. We need to build an application that has to be connected

to the framework and must request the recognized gestures by recognizer. For instance, the application can receive *SwipeLeft2Touch* or *AirSwipeLeft* gesture. Arthur Sluÿters developed a JS library that can you can find **here**. This library can be used inside a react project. This library can be useful if we have to test QuantumLeap with a new sensor.

### 3.5.3 Multiple Sensors inside QuantumLeap

We have two 3DTouchpads in our possession for this thesis. So, we started to be interested in using the two devices together inside QuantumLeap. The use of multiple devices has been implemented inside QuantumLeap but it was never tested before. However, we faced different problems during the testing:

1. It is not possible to connect two touchpads in Linux. So, we only use Windows OS.

2. There are some coordination problems related to the articulations inside the core system of QuantumLeap

3. The segmentation structure is not adequate.

All these problems are going to be explained and solved in the next sections.

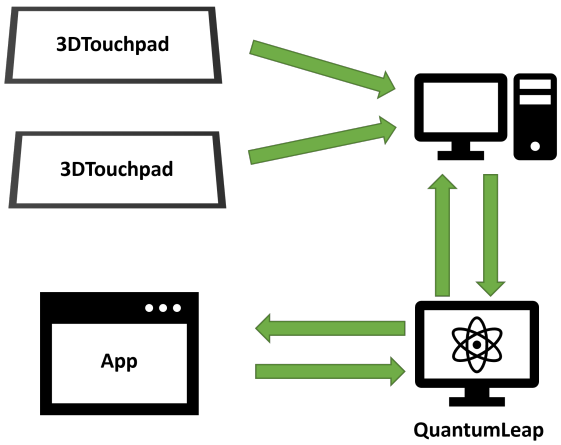



Figure 3.4: Overall structure of QuantumLeap with multiple sensors.

#### Articulations Design

The two sensors, even if they are similar device, have to be clearly separated inside the framework core design. Each device must have their own dedicated articulations. The articulations of the two touchpads are differentiated by their sensor id. The

port defined by the config.js file can't be the same. The loader is responsible for generating trainable data twice for each gesture coming from the touchpad dataset but without mixing with articulation from the other sensor id. As example, these are the articulations of each touchpad:

- **Touchpad 1** : 3D_id1, 2DTouch1_id1, 2DTouch2_id1, 2DTouch3_id1
- **Touchpad 2** : 3D_id2, 2DTouch1_id2, 2DTouch2_id2, 2DTouch3_id2

This separation is necessary for QuantumLeap framework. Otherwise, the data from both devices is mixed together. And so, each recognizer do not receive interpretable data and loose lots of accuracy.

### Articulation Design Problem

We found that the *articulation* architecture design was not fitting with what we imagined.

The first problem is that when the recognizer receives the data, it must contain exactly the same amount of points for each articulation. However, when the loader of a sensor or a sensor sends their data to the recognizer, some error happens. The received data in the recognizer contains points only relevant to their sensor articulations. Other sensor can't know other articulations from an other sensor. This results in undefined errors as sample does not have all the defined articulations. Unfortunately, a quick fix is not possible. We had to change several things to solve this first problem. The initial file that had to be changed was utils.js which is located inside the back-end framework folder. Its goals is to parse the different names of all the articulations defined from each sensor. Initially, the function was returning a table containing all the articulations names that had to be treated inside QuantumLeap. Now, the script consists in storing articulations from a sensor inside a table. So, if we use multiple sensors, there is an "articulation table" for each sensor. And all these tables are stored inside one big table. Therefore, the function returns a table containing tables. For instance, **pointsNames[0]** returns all the articulations from the first defined sensor. Thanks to this fix, recognizers can know which articulations are related to what sensors. This is a small example of how the function was returning data before and after the fix.

- **Before**: ["2DTouch1_id1", "3D_id1", "2DTouch1_id2", "3D_id2"]
- **After**: [ ["2DTouch1_id1", "3D_id1"] , ["2DTouch1_id2", "3D_id2"]]

The **parsePointsNames** function is now implemented as:

```
function parsePointsNames(selectedPoints) {
  let pointsNames = [];
  Object.keys(selectedPoints).forEach(sensorName => {
    let sensor_point = []
    sensor_point.push.apply(sensor_point, selectedPoints[sensorName]
        .map(pointName => `${pointName}_${sensorName}`))
    pointsNames.push(sensor_point)
  });
  return pointsNames;
}
```

Now that we clearly separated each articulation from each sensor, we can go through the second step to fix the error. It consists in selecting for each recognizer the right articulations from one sensor according to the received data. To solve the problem, we just had to add the following code at the beginning of the convert function from each recognizer. **SelectedPoints** contains the table returned by parsePointsNames. The goal of these code lines is to see if the sample received by the recognizer is associated to which sensor. Once we have discovered from which sensor the data is coming, we reduced the **SelectedPoints** variable to a table containing only relevant articulations for this sensor.

```
for(let i = 0 ; i < Object.keys(sample.paths).length; i ++){
    for(let y = 0; y < selectedPoints.length ; y++){
      if(sample.paths[selectedPoints[y][i]] !== undefined){
        selectedPoints = selectedPoints[y]
        y = selectedPoints.length
        i = sample.paths.length
      }
    }
 }
```

The problem related to the articulation design is now solved and each recognizer is now able to train and process data as wanted in a multi-sensors architecture.

### Segmentation Problem

The other major problem is the segmentation architecture. There are two steps to solve the problem. The **First one** is inside the sensor-group.js. It is located inside the sensor folder which is located inside the modules folder. The problem comes from the **process-frame** function. Initially, the function was collecting the data returned by sensors from each getPoint function. But, it was mixing the data from different sensors together. Therefore, the recognizers received this mixed data and as it was not trained with this "mixed" data, it reduced considerably its recognition rate for each sensor. But, we managed to solve this problem by the following code which represents the core of the processFrame function with our fix. Now, QuantumLeap is collecting data from each sensor separately and is directly

sending the data for segmentation. The data from multiple sensors are not mixed anymore.

```
let appData = {};
// Get points from each sensor
this.sensors.forEach(({sensor, id}) => {

    let hasData = false;

    let timestamp = Date.now();
    // Initialize the frame and appData
    let frame = new Frame(timestamp);

    let sensorData = sensor.getPoints(timestamp);
    hasData = hasData || sensorData.hasData;
    sensorData.points.forEach(({name, point}) => {
      let pointName = id ? `${name}_${id}` : name;
      frame.addArticulation(new Articulation(pointName, point));
    });

    appData = {
      ...appData,
      ...sensorData.appData
    };

    frame.hasData = hasData;
    // Callback only if data was sensed by a sensor
    if (hasData) {
      callback(frame, appData);
    }
```

The **second problem** is inside the **abstract-segmenter.js** and to be more precise inside the **segment function**. The segment is responsible for returning if the processed frame is corresponding to a dynamic gesture. If it detects the dynamic gesture, it returns a *StrokeData* Object. Otherwise, it returns null. The idea to solve the problem is similar to the previous one. We have to separate articulations from each sensor. If the actual code lines are swapped with the following one, the sensor data will always be transmitted separately to the recognizer. Therefore, the recognizer accuracy of each sensor is such as they were used alone.

```
segment(frame) {

// Get raw segments
let rawSegments = this.computeSegments(frame);
// Convert segments
let segments = [];
rawSegments.forEach(frames => {
  // Group all points by articulation
  let articulationsPoints = {}
```

```
  frames.forEach(frame => {

    frame.articulations.forEach(articulation => {
      if (articulationsPoints[articulation.label]) {
        articulationsPoints[articulation.label]
            .push(articulation.point);
      } else {
        articulationsPoints[articulation.label]
            = [articulation.point];
      }

    });

  })

  for(let w = 0; w < this.motionArticulations.length; w++){

  let articulationsPointsFromSelectedPoints = {}
  for(const articulationLabel of this.motionArticulations[w]){
    articulationsPointsFromSelectedPoints[articulationLabel]
        = articulationsPoints[articulationLabel]
  }

   let strokeData = new StrokeData();

   if (isMotion(articulationsPointsFromSelectedPoints,
   this.motionThreshold, this.motionArticulations[w])) {

      for (const articulationLabel of this.motionArticulations[w]) {

        let path = new Path(articulationLabel);
        strokeData.addPath(articulationLabel, path);
        let stroke = new Stroke();
        path.addStroke(stroke);
        stroke.points =
            articulationsPointsFromSelectedPoints[articulationLabel];
       }
       segments.push(strokeData);
    }
  }
});
return segments;
 }
```

## 3.6 Conclusion

In this chapter, we overviewed all our codes implementations for this thesis. We

managed to retrieve the 3DTouchpad data in the most efficient way. We also integrated the touchpad inside QuantumLeap. This is very interesting because we have shown that it was possible to use easily the framework with new devices. Also our modifications and our new files helped the framework to be more reusable and more generic. We also proved that it is possible to use the framework with multiple devices at the same time.

# Chapter 4

# Gestures Set Definition

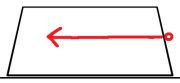
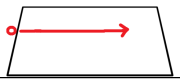
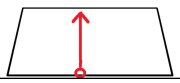
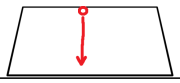
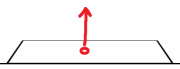
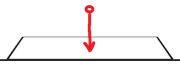
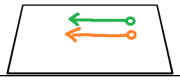
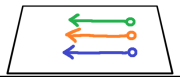
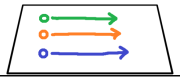
## 4.1 Overview

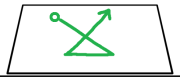
This section is divided into three main parts. The first section 4.2 presents all the gestures that we decided to include in our gestures set. You can find some explanations about each gesture and where we took our inspirations. And then in section 4.3, we asked some people to collect their gestures using the 3DTouchpad to populate our dataset. Finally, section 4.4 overviews possible Air+Touch combinations that seem very promising.

## 4.2 Gestures Set

For this thesis, we decided to define **34 gestures** associated with the 3DTouchpad. There are **8** 3D gestures and **26** 2D gestures. This difference comes from the fact that the touchpad senses only one triangulated point in the 3D mode. Thus, it is very restrictive for us because we can't use multi-fingers gestures or (multi-)hand gestures in the 3D mode. The table 4.1 presents every gesture included inside our gestures set definition. There are four colors used for the drawing of gestures. Each one has a different meaning:

- **Red**: Represents a gesture performed above the surface.(3D)
- **Green**: Represents the 2D gesture performed by the first finger.
- **Orange**: Represents the 2D gesture performed by the second finger.
- **Blue**: Represents the 2D gesture performed by the third finger.

| Gesture Name | Space | Description | Draw | Reference |
|---|---|---|---|---|
| AirCircleLeft | 3D | Draw circular path above the touchpad in an anti-clockwise direction. |  | [34, 10, 33] |
| AirCircleRight | 3D | Draw circular path above the touchpad in a clockwise direction. |  | [34, 10, 33] |
| AirSwipeLeft | 3D | Swipe From East to West above the touchpad. |  | [34, 33, 40, 42, 6] |
| AirSwipeRight | 3D | Swipe From West to East above the touchpad. |  | [34, 33, 40, 42, 6] |
| AirSwipeForward | 3D | Swipe from the South to North above the device. |  | [34, 33, 40, 42, 6] |
| AirSwipeBackward | 3D | Swipe from the North to South above the device. |  | [34, 33, 40, 42, 6] |
| AirSwipeUp | 3D | Elevate your finger above the device. |  | [47, 43, 32, 33] |
| AirSwipeDown | 3D | Lower your finger closer above the device. |  | [47, 43, 32, 33] |
| SwipeLeft1Touch | 2D | Swipe from East to West with **one** finger which is touching the surface. |  | [34, 47, 40, 42] [6, 33] |
| SwipeLeft2Touch | 2D | Swipe from East to West with **two** fingers which are touching the surface. |  | [34] |
| SwipeLeft3Touch | 2D | Swipe from East to West with **three** fingers which are touching the surface. |  | / |
| SwipeRight1Touch | 2D | Swipe from West to East with **one** finger which is touching the surface. |  | [34, 47, 40, 42] [6, 33] |
| SwipeRight2Touch | 2D | Swipe from West to East with **two** fingers which are touching the surface. |  | [34] |
| SwipeRight3Touch | 2D | Swipe from West to East with **three** fingers which are touching the surface. |  | / |
| SwipeUp1Touch | 2D | Swipe from South to North with **one** finger which is touching the surface. |  | [34, 47, 40, 42] [6, 33] |

| | | | | |
|---|---|---|---|---|
| SwipeUp2Touch | 2D | Swipe from South to North with **two** fingers which are touching the surface. |  | [34] |
| SwipeUp3Touch | 2D | Swipe from South to North with **three** fingers which are touching the surface. |  | / |
| SwipeDown1Touch | 2D | Swipe from North to South with **one** finger which is touching the surface. |  | [34, 47, 40, 42] [6, 33] |
| SwipeDown2Touch | 2D | Swipe from North to South with **two** fingers which are touching the surface. |  | [34] |
| SwipeDown3Touch | 2D | Swipe from North to South with **three** fingers which are touching the surface. |  | / |
| CircleLeft1Touch | 2D | Draw anti-clockwise circle using **one** finger on the surface. |  | [?] |
| CircleLeft2Touch | 2D | Draw anti-clockwise circle using **two** fingers on the surface. |  | / |
| CircleLeft3Touch | 2D | Draw anti-clockwise circle using **three** fingers on the surface. |  | / |
| CircleRight1Touch | 2D | Draw clockwise circle using **one** finger on the surface. |  | [?] |
| CircleLeft2Touch | 2D | Draw clockwise circle using **two** fingers on the surface. |  | / |
| CircleRight3Touch | 2D | Draw clockwise circle using **three** fingers on the surface. |  | / |
| Cross1Touch | 2D | Draw a kind of 'X' on the surface using **one** finger |  | [?] |
| Cross2Touch | 2D | Draw a kind of 'X' on the surface using **two** fingers |  | / |
| Cross3Touch | 2D | Draw a kind of 'X' on the surface using **three** fingers |  | / |
| ZoomPinchIn2Touch | 2D | Pinch in the **two** touching fingers together. |  | [32, 33, 27, 6] |

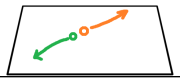
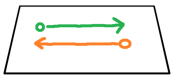
| | | | | |
|---|---|---|---|---|
| ZoomPinchOut2Touch | 2D | Pinch out the **two** touching fingers together. |  | [32, 33, 27, 6] |
| SwipeLeft&Right2Touch | 2D | Swipe from West to East on the surface with the **first** finger whereas the **second** finger is swiping from East to West at the same time. |  | / |
| Hold&SwipeLeft2Touch | 2D | Swipe from East to West on the surface with the **first** finger whereas the **second** finger keeps its touching position |  | / |
| Hold&SwipeRight2Touch | 2D | Swipe from West to East on the surface with the **first** finger whereas the **second** finger keeps its touching position |  | / |

Table 4.1: Our gestures set definition

The different gestures where their reference column is filled with "/", are often based on a gesture but performed with less fingers. Our gestures set contains 28 gestures at the first iteration. Six new gestures were added thanks to gestures elicitation studies results from chapter 6: *Cross1Touch, Cross2Touch, Cross3Touch, SwipeLeft&Right2Touch, Hold&SwipeLeft2Touch, Hold&SwipeRight2Touch.*

Now that our gestures set is defined, we can clearly see that 3D gestures offer much more freedom as we got more space to perform gestures. For example, if we take *CircleRight1Touch* and *AirCircleRight* gestures, they are exactly the same gesture but one is in 2D and the other 3D. If you performing the two together, you're more comfortable with 3D as it offers more expressiveness in space. Based on our gestures set definition, we can somehow affirm that Air+Touch gesture recognition can reduce the problem related to the freedom and expressiveness from 2D gestures.

## 4.3 Collect Samples

Now that we have defined our gestures set, we have to record some gestures samples. To stay fair and be representative of the population, we asked several participants to record these gestures. As our gestures set is basically based on gesture only using one hand, a left-handed gesture can be different from a right-handed gesture. The goal is to record 8 right-handed participants making each gesture twice and to record 4 left-handed participants making each gesture four times. We chose

to record only 4 left-handed participants because they are more difficult to find. Therefore, we are able to define three different datasets:

- **Right-Handed**: dataset with all the right-handed samples
- **Left-Handed**: dataset with all the left-handed samples
- **Whole**: dataset with all the left-handed and right-handed samples

The way to build a coherent dataset that fits perfectly with our dataset loader that is implemented inside QuantumLeap is composed of three steps:

- Save the record in a JSON file and call it with its gesture name and its ID sample. The two arguments have to be separated by a dash as follow: **"AirSwipeUp-1.json"**.
- Put all samples from a participant in a folder named with the participant numeric id.
- Put all the participants folders named with their numeric id in one folder named with a relevant name for the dataset as **Right-handed Touchpad dataset**

### 4.3.1 Context Of The Gesture Collect

We interviewed every participant in the same conditions for the collect. Each one received a 10 minutes introduction to familiar themselves with the device. All gestures were asked in the same order for everyone. The participants were seated at a table and the touchpad was exactly at the same position. Each participant could remake each gesture if they were not convinced by the performed gesture. Therefore, the only factor that is still present is the "human factor" which changes according to each participant.

### 4.3.2 Participants Information

All the participants who performed the recorded gestures were asked to give some personal informations. They gave us **their initials**[1]**, their age, their natural hand, their status and their field of work**. All their informations are resumed in table 4.2. Thanks to these informations, we can dress some statistics that take place in the next subsection.

[1]To preserve the confidentiality of the participants

| ID | Initials | Gender | Age | Hand | Status | Field Of Work |
|---|---|---|---|---|---|---|
| 1 | N.V. | Woman | 25 | Left-handed | Worker | Psychology |
| 2 | R.G. | Man | 22 | Left-handed | Student | Physiotherapist |
| 3 | C.I. | Woman | 53 | Left-handed | Worker | Economy |
| 4 | H.M. | Woman | 23 | Left-handed | Worker | Physiotherapist |
| 5 | F.R. | Man | 23 | Right-handed | Student | Physiotherapist |
| 6 | R.A. | Man | 23 | Right-handed | Student | Physical Education |
| 7 | S.A | Man | 22 | Right-handed | Student | Physiotherapist |
| 8 | Z.E | Woman | 22 | Right-handed | Student | Psychology |
| 9 | B.M. | Man | 24 | Right-handed | Student | Economy |
| 10 | T.T. | Man | 23 | Right-handed | Worker | Civil Engineer |
| 11 | N.P. | Man | 55 | Right-handed | Worker | Social Sciences |
| 12 | D.C. | Woman | 23 | Right-handed | Student | Psychology |

Table 4.2: Personal Information of each participant.

### 4.3.3 Participants Statistics

An important point when you populate a dataset is to choose a group of people from different sectors to be as diverse as possible. The more diverse it is, the more representative it is. So, the test results are representative of the population as much as possible .

We have made some statistical graphs to show you that the dataset is enough diversified. The two graphs in figure 4.1 represent two different proportions inside the dataset. The first proportion is the amount of **students** compared with the amount of **workers** among the participants. The second proportion is the amount of **women** compared with the amount of **men** among the participants. The proportion value is **42%** for the workers and women and **58%** for men and students. These proportions show equity.

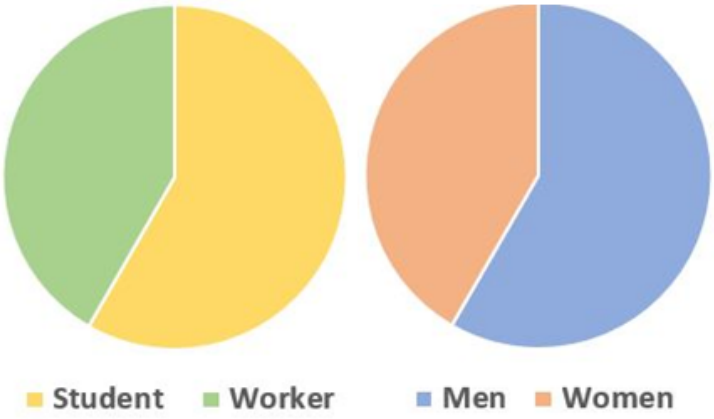



Figure 4.1: 1) Graphic representing the proportion of students and workers inside the dataset 2)Graphic representing the proportion of men and women inside the dataset

An other interesting graphic can be made. It is one relating to the different fields of work from all the participants. The following figure 4.2 is this graphic. As you can see, participants were coming from different fields: **25%** psychology, **33%** physiotherapist, **8%** physical education, **8%** civil engineer, **8%** social sciences, **17%** economy. In our opinion, there are maybe too many physiotherapists. But the point is that the 4 main fields are represented: Medical, Scientific, Economic and Social.

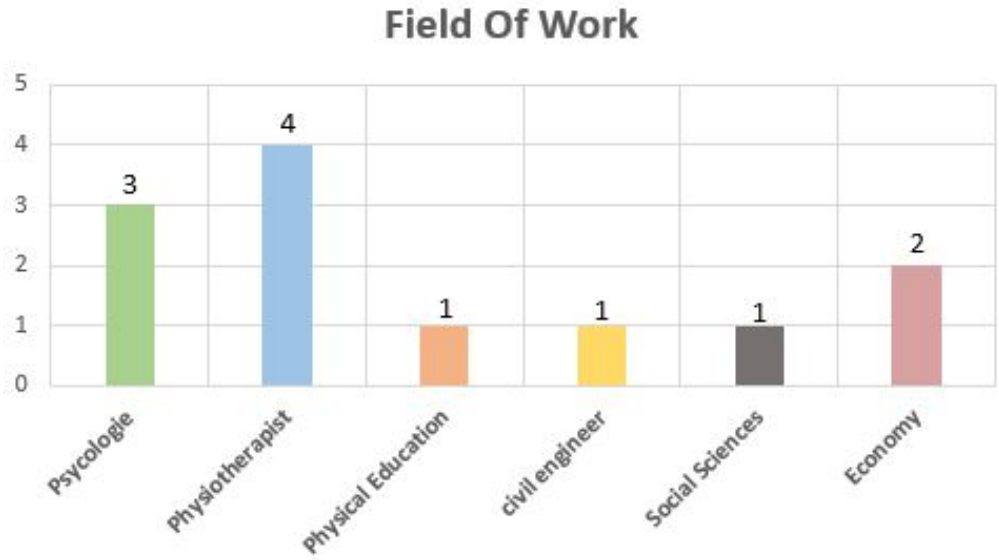



Figure 4.2: Graph representing all the different fields of work from the participants.

The last statistic is about the average age. The average age is equal to **28-29** years old. It would have been better if it was higher. But, considering the situation at the time of the collect, it was hard to find older people.

After analyzing all these statistics, all the participants were enough different to say that the overall dataset satisfies the diversification. So, future test will be very interesting and promising.

### 4.3.4 Participants Feelings

At the end of each interview, we asked the participants about their feelings from the gestures they performed. Everyone was intrigued and interested in 3D gestures. They found them cool to record. However, they were also more unfamiliar with them, they had more difficulties to make them right. Whereas for the 2D gestures, they were more familiar with it. Everyone quickly understood and performed the required gestures. They mastered the gestures with one and two fingers, as everyone "uses" them on their smartphone. But, it was more difficult to process 2D gestures with 3 fingers. These feelings are in line with the Air+Touch studies that we analyzed in the related work chapter.

## 4.4 Air+Touch Combinations

Now that we have defined an interesting gestures set and collected samples from multiple participants, we can focus on the possible combinations. By combination, we mean that an application can request multiple gestures to process an action. For example: "Imagine we are using a smart Tv, with the touchpad we can do an AirSwipeDown to say to the application that we want to interact with the smart TV. Then, we can process a SwipeLeft1Touch to say that we want to zap to the previous channel." This example can be seen as an Air+Touch combination.

There is a huge number of possible combinations according to the combinations requirements. The requirements can, for example, be: the **number** of successive gestures to make an action, gesture **type** defined ( the first one of the combination can only be a 3D gesture, the second is a 2D), is **repetitions** allowed. This is where it becomes interesting. Table 4.3 represents the amount of possible combinations based on our gestures set definition according to the requirements. The table is limited to two gestures combinations and repetitions are allowed. The amount of combinations has already reached 1156 as maximum value. This is already huge!

| First gesture requirement | second gesture requirement | calculation | Result |
|---|---|---|---|
| Every possible gestures | Every possible gestures | 34x34 | 1156 |
| Only 3D | Every possible gestures | 8x34 | 272 |

| | | | |
|---|---|---|---|
| Only 3D | Only 2D | 8x26 | 208 |
| Only 2D | Every possible gestures | 26x34 | 884 |

Table 4.3: 1) **34** = 2D+3D gestures 2) **26** = 2D gestures 3) **8** = 3D gestures

The previous paragraph considers only combinations as the second gesture starting when the first one is finished. Now, let's consider the Allen's interval algebra [3]. We already defined this topic in details in chapter 2. In his work, Allen defined 13 possible time relations. Thanks to the fact we solved the multi-sensors problems inside QuantumLeap, we can consider the thirteen time relation using **two touchpads**. This is very interesting because the number of possible combinations is massively increasing. The table 4.4 contains the same information as table 4.3 but with the thirteen time relations included.

| **First gesture requirement** | **Second gesture requirement** | **Calculation** | **Result** |
|---|---|---|---|
| Every gesture possible | Every possible gestures | 13x34x34 | 15 028 |
| Only 3D | Every possible gestures | 13x8x34 | 3536 |
| Only 3D | Only 2D | 13x8x26 | 2704 |
| Only 2D | Every possible gestures | 13x26x34 | 11 492 |

Table 4.4: 1) **34** = 2D+3D gestures 2) **26** = 2D gestures 3) **8** = 3D gestures 4) **13** = possible time interval

So with combinations of only two gestures, we reach **15 028** possible combinations. This is enormous! Using an application with so many combinations doesn't make any sense. But there is a huge advantage with this amount of combinations, we have the luxury to choose the gestures that:

- are the most comfortable.
- are easily mastered.
- are the most accurate and the least confusing with each other.

This is very interesting in customers point of view. Therefore, using Air+Touch combinations also means that we can use 3D gestures to express actions that can scale in space according to the distances and speeds of gesture. For instance, these actions can be *"Increase/Decrease the TV volume, Zap channel"*. However, 2D gestures can be used with actions which need to be accurate as *"Turn off/TV, Start Recording"*. So, Air+Touch combinations can be interesting and complementary.

## 4.5 Conclusion

In this chapter, we defined our own Air+Touch gestures set definition. It is composed of 34 gestures. We also populated our different datasets with many participants samples. We have tried to make them as diverse as possible to improve the results of this thesis. In the end, we mentioned the Air+Touch Combinations. These are looking very promising as it seems to be easy to swap from 2D to 3D and vice-versa. With Allen's interval algebra, we also showed that Air+Touch can be really interesting as it offers plenty of possible combinations.

# Chapter 5

# Benchmarking Of Recognizers

## 5.1 Overview

This chapter compares the different recognizers on their results on each dataset defined in chapter 4. Section 5.2 describes all the different parameters and criteria about the testing. Section 5.3 presents in details the two different testing scenarios: User-Independent(UI) and User-Dependent(UD). Section 5.4 analyzes all results from the raw data from each recognizer based on the two different scenarios.

## 5.2 Benchmark

A benchmarking of recognizers consists in comparing the behaviour of different recognizers. We are going to compare, for each gesture, their recognition rate and their execution time rate. QuantumLeap Framework [50] already contains an implementation of benchmarking. However, this implementation does not fit with our testing requirements. So, we had to adapt it to our own implementation. Our testing implementation can be found in appendix B.4.3.

### 5.2.1 Criteria

To judge if a recognizer is enough efficient to be tested, we determine different criteria that have to be satisfied:

- **Recognition Rate:** It is the total number of gestures correctly identified by the recognizer divided by the overall number of tested gestures. Each recognizer must recognize correctly almost all of the gestures.
- **Execution Time Rate:** It is the overall time taken by the recognizer for each gesture divided by the number of tested gesture. Each recognizer must

recognize gestures quickly.

- **Space variability:** Each recognizer must recognize 2D gestures almost as well as 3D gestures and vice versa.

Unfortunately, many of the recognizers implemented inside QuantumLeap are not satisfying these criteria. Just to give an idea, we tested the $P3+ [56], $\mu$V, GSPDa but their results were not enough satisfying. The results are detailed inside the table 5.1 and the figure 5.1 below. Each recognizer was trained on the right-handed dataset with 4 templates and in a user-independent scenario. As you can see, there are two recognizers that performed much better than the others. They are the **$\mu$F** recognizer and the **Jacknife** recognizer. We also performed a t-test which compared the five recognizers results. The figure 5.2 is showing the result of the t-test between Jacknife and the other recognizers. On this figure, we can clearly see that the t-test between Jacknife and $\mu$F is the only one which is not resulting in significantly different means. Therefore, it proves that Jacknife and $\mu$F are the two most interesting recognizers. Thus, we are going to consider only these two recognizers for the rest of the chapter.

| Name | Recognition Rate | Std deviation[1] | Exec Time |
|---|---|---|---|
| $P3+ | 0.22706 | 0.13095 | 1.8754 ms |
| GSPDa | 0.29235 | 0.19437 | 1.0044 ms |
| $\mu$V | 0.34353 | 0.32841 | 1.0468 ms |
| $\mu$F | 0.68317 | 0.22 | 1.7422 ms |
| Jacknife | 0.73093 | 0.23911 | 1.125 ms |

Table 5.1: Averaged Recongition Rate, Standard Deviation and Execution time for each recognizer.

[1]Standard deviation of the recognitionrate for each recognizer

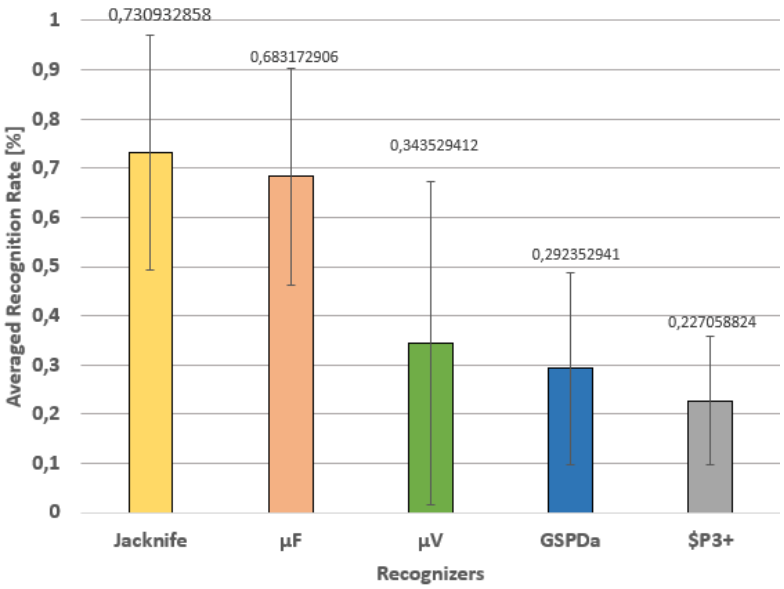



Figure 5.1: Averaged recognition rate of each recognizer

| t-Test: Paired Two Sample for Means | | |
|---|---|---|
| | *Jacknife* | *µF* |
| Mean | 0,730933 | 0,683173 |
| Variance | 0,058908 | 0,049888 |
| Observations | 34 | 34 |
| Pearson Correlation | 0,731599 | |
| Hypothesized Mean Difference | 0 | |
| df | 33 | |
| t Stat | 1,622102 | |
| P(T<=t) one-tail | 0,057149 | |
| t Critical one-tail | 1,69236 | |
| P(T<=t) two-tail | 0,114298 | |
| t Critical two-tail | 2,034515 | |

| t-Test: Paired Two Sample for Means | | |
|---|---|---|
| | *Jacknife* | *µV* |
| Mean | 0,730932858 | 0,343529 |
| Variance | 0,05890758 | 0,11112 |
| Observations | 34 | 34 |
| Pearson Correlation | 0,595111113 | |
| Hypothesized Mean Difference | 0 | |
| df | 33 | |
| t Stat | 8,319099615 | |
| P(T<=t) one-tail | 6,55103E-10 | |
| t Critical one-tail | 1,692360309 | |
| P(T<=t) two-tail | 1,31021E-09 | |
| t Critical two-tail | 2,034515297 | |

| t-Test: Paired Two Sample for Means | | |
|---|---|---|
| | *Jacknife* | *GSPDa* |
| Mean | 0,730933 | 0,292353 |
| Variance | 0,058908 | 0,039152 |
| Observations | 34 | 34 |
| Pearson Correlation | 0,311903 | |
| Hypothesized Mean Difference | 0 | |
| df | 33 | |
| t Stat | 9,79963 | |
| P(T<=t) one-tail | 1,34E-11 | |
| t Critical one-tail | 1,69236 | |
| P(T<=t) two-tail | 2,68E-11 | |
| t Critical two-tail | 2,034515 | |

| t-Test: Paired Two Sample for Means | | |
|---|---|---|
| | *Jacknife* | *$P3+* |
| Mean | 0,730932858 | 0,227059 |
| Variance | 0,05890758 | 0,01767 |
| Observations | 34 | 34 |
| Pearson Correlation | 0,497965118 | |
| Hypothesized Mean Difference | 0 | |
| df | 33 | |
| t Stat | 13,93623326 | |
| P(T<=t) one-tail | 1,10793E-15 | |
| t Critical one-tail | 1,692360309 | |
| P(T<=t) two-tail | 2,21585E-15 | |
| t Critical two-tail | 2,034515297 | |

Figure 5.2: Paired T-test for means comparing two recognizers together

### 5.2.2 Recognizers Definition

As we limited our analysis to only two recognizers, we are only defining these two recognizers.

**Jacknife** [56] is an algorithm which is able to recognize dynamic gestures from continuous data. The data can be retrieved from different devices. It can be a 3D sensor, a 2D touchscreen and many others. Therefore, it fits well with the 3DTouchpad [34]. It doesn't need a lot of training samples to perform an high recognition rate. It is also using a Dynamic Time Warping [41] to measure distance between two samples. DTW is a well-known technique which consists in finding an optimal alignment between two time-dependent templates.

$\mu\mathbf{F}^2$ [50] means multi-features. It is an algorithm developed to recognize 2D and 3D gestures. Gestures are sampled by a touch-sensitive surface or advanced computer vision techniques. They are characterized by one or several-paths. The recognizer has 4 different weights to determine the similarity between a candidate and a set of templates. They are called Alpha, Beta ,Gamma and Delta. Alpha is the weight for the Between-points Vector Distance metric. Beta is the weight for the Between-shapes distance metric. Gamma is the weight for the Between-Shapes Shape Distance metric. And delta is the weight for the Between-M -dimensional-Shape Distance metric. Beta is insensitive to variations in rotation whereas all the other weights are invariant to variations along Scale and Translation.

### 5.2.3 Testing Parameters

There are a lot of parameters associated to the testing. Some are fixed and some are changing at each specific test. We decided to fix the following parameters:

- **Sampling Points**: We fixed it to the value **16**. We based this value on the following article: *The Effect of Sampling Rate on the Performance of Template-based Gesture Recognizers* [54]. In this paper, authors prove that the value 16 is the optimal one. If we choose an higher value, the recognition rate is probably the same and maybe even lower.

- **Repetition Factor**: The value is fixed at **100**. This means that each gesture is tested 100 times.

- **$\mu$F Parameters**: *(Alpha, Beta, Gamma, Delta) metrics* is set to (1,4,0,0) for 3D gestures and (3,1.5,1,2) for 2D gestures.

Nevertheless, these following parameters can change depending on the test performed:

- **Test scenario**: can be *User-dependent(UD)* or *User-Independent(UI).*

- **Dataset**: we can choose to test samples depending on our three different datasets defined in chapter 4..

[2] $\mu$F recognizer is relating from the work of Nathan Magrofuoco

- **Number of Training Templates**: defined with how many different templates each recognizers trained itself.

## 5.3 Testing Scenario

For this thesis, we mainly focused on two different scenarios for testing the recognizers. They are called User-Independent and User-Dependent.

### 5.3.1 Scenario A: User-Independent

A User-Independent scenario for a single gesture can be defined according to the following steps:

1. Create the empty table **MarkedTemplates**.
2. Choose randomly a sample called **Candidate A** inside the dataset.
3. Push the **Candidate A** inside the **MarkedTemplates**.
4. Choose randomly a sample called **Training** which is not performed by the same user as **Candidate A** and which is not inside **MarkedTemplates**.
5. Train recognizer(s) with **Training** and push it inside **MarkedTemplates**.
6. Repeat the two previous steps until recognizer(s) is/are trained with enough templates to satisfy the amount of defined training templates.
7. Test recognizer(s) with the **Candidate A**.

These steps are repeated hundred times for each gestures to meet the repetition factor requirements. In a pseudo-code, this scenario can define as:

**Algorithm 1:** User-Independent Scenario

```
Data: Dataset,Recognizer(s)
Result: Test recognizer(s) for each gestures from the dataset according to
        the user-independent scenario
N ← TrainingTemplate;
for r ← 0 to 100 by 1 do
    //Return a random list of candidate, one for each gesture;
    candidates ← SelectCandidates(dataset);
    MarkedTemplates ← mark the templates present in candidates;
    for t← 0 to N do
        index ← 0;
        foreach gestureClass do
            training ← -1;
            //Check if training is set to a new value, if the value is not
             already used and if the user is not the same as candidate;
            while training ← -1 OR MarkedTemplates.include(training) OR
             MarkedTemplates[user] = gestureClass[training].user do
                training ← getRandomNumber(0,gestureClass.length);
            end
            MarkedTemplates[index].push(training);
            recognizer ← Train(gestureClass[training]);
            index++;
        end
        index ← 0;
        foreach gestureClass do
            toBeTested ← gestureClass[candidate[index]];
            result ← recognizer.recognize(toBeTested);
            index++;
        end
    end
end
```

### 5.3.2 Scenario B: User-Dependent

This second scenario is different from the previous. This scenario for a single gesture can be defined as follow:

1. Choose a user inside the dataset which is called **User A**
2. Create the empty table **MarkedTemplates**.

3. Choose randomly a sample called **Candidate A** performed by the **User A**.
4. Push **Candidate A** inside **MarkedTemplates**.
5. Choose randomly a sample called **Training A** which is performed by **User A** and which is not inside **MarkedTemplates**.
6. Train recognizer(s) with **Training A** and push it inside **MarkedTemplates**.
7. Repeat the two previous steps until recognizer(s) is/are trained with enough templates to satisfy the amount of defined training templates.
8. Test recognizer(s) with the **Candidate A**.

These steps are repeated for each gesture from each user. This is repeated until the repetition factor is met. For example: "If we have 5 different users, the gesture of each user is repeated 20 times to meet the repetition factor requirement which is equal to 100". The pseudo-code of this scenario is basically the same than the user-independent scenario with new steps described above. It can be defined as:

**Algorithm 2:** User-Dependent Scenario

```
Data: Dataset,recognizer
Result: Test recognizer(s) for each gestures from the dataset according to
        the user-dependent scenario
N ← TrainingTemplate;
userTab ← table containing all the user ID;
for r ← 0 to 100/userTab.length do
    foreach user inside UserTab do
        //Return a random list of candidate all performed by the same user;
        candidates ← SelectCandidatesOfUser(dataset,user);
        MarkedTemplates ← mark the templates present in candidates;
        for t← 0 to N do
            index ← 0;
            foreach gestureClass do
                training ← -1;
                //Check if training is set to a new value, if the value is not
                 already used and if the user is the same as candidate;
                while training ← -1 OR MarkedTemplates.include(training)
                 OR MarkedTemplates[user]
                 ≠ gestureClass[training].user do
                    training ← getRandomNumber(0,gestureClass.length);
                end
                MarkedTemplates[index].push(training);
                recognizer ← Train(gestureClass[training]);
                index++;
            end
        end
        index ← 0;
        foreach gestureClass do
            toBeTested ← gestureClass[candidate[index]];
            result ← recognizer.recognize(toBeTested);
            index++;
        end
    end
end
```

### 5.3.3 Multiple Recognizers

As we had to perform these two scenarios with multiple recognizers, we adapted the implementation to always train each recognizer with the same random templates

and to test each recognizer with the same random candidate for each iteration. So, we can execute a scenario with the same data for each recognizer at each iteration. This improves our result quality. The table 5.2 is showing that the **ID Sample** and the **ID User** are exactly the same for each recognizer at each iteration.

### 5.3.4 Export Result

The **result** variable from each scenario contains the execution time and the guessed gesture of the recognizer. We need to store these values and put them in a file for analysis. We chose to use a .csv file as it was the easiest. The file are updated with a new line at each iteration. Each line from the csv file contains the information as in table 5.2. All the saved information represent the raw data on which our analysis is based.

| R | Recognizer | Gesture Class | Sample | User | Time | Correct | Guessed Gesture |
|---|---|---|---|---|---|---|---|
| 1 | jacknife | AirCircleLeft | 2 | 11 | 5 | 1 | AirCircleLeft |
| 2 | $\mu$F | AirCircleLeft | 2 | 11 | 7 | 1 | AirCircleLeft |
| 3 | jacknife | AirCircleRight | 3 | 7 | 6 | 1 | AirCircleLeft |
| 4 | $\mu$F | AirCircleRight | 3 | 7 | 8 | 0 | AirSwipeBackward |

Table 5.2: Raw Data stored inside .csv file

## 5.4 Result

We are going to test our different datasets now we implemented the testing tools. As a quick reminder we have these three datasets:

- **Right-Handed Dataset**: Eight right-handed users performed twice each gesture from our gestures set.
- **Left-Handed Dataset**: Four left-handed users performed four times each gesture from our gestures set.
- **Whole Dataset**: Simply the combinations of the two previous datasets.

So, in the user-independent scenario, a recognizer can be trained with 14 templates(**right**), 12 templates(**left**) and 26 templates(**whole**). We restricted the amount of the maximum training templates to **8** as it is too computational intensive with more templates.

For the user-dependent test, a recognizer can be trained with 1 template(**right**) and 3 templates(**left**). To keep some equity between the analysis between the two datasets, we restricted the amount of the maximum training template to **1**. For this scenario, we are not going to discuss about the whole dataset.

### 5.4.1 User-Independent Test

#### Recognition Rate Per Dataset

The figure 5.3, figure A.1 and figure A.2 are the graphical representation of the recognition rate for each gesture. Figure 5.3 is based on the right-handed dataset, Figure A.1 on the left-handed and Figure A.2 on the whole dataset. The grey columns represent the recognition rate of $\mu$F for each gesture whereas the green is for Jacknife. Each recognizer was trained with 8 templates. The table 5.3 contains the recognition rate of each gesture for each recognizer and dataset. The values are related to the mentioned figures.

| Dataset | Recognition Rate Jacknife | Recognition Rate $\mu$F |
|---|---|---|
| Left | 0.8065 | 0.7144 |
| Right | 0.7774 | 0.7318 |
| Whole | 0.7947 | 0.7209 |

Table 5.3: Recognition Rate of each gesture associated to its dataset and recognizer.

There is no information emerging from table 5.3. The recognition rate stays more or less the same with the different datasets. One interesting point is that the $\mu$f has a better recognition rate with the right dataset whereas it is not the case for Jacknife. This can be interesting to analyze in details why it occurs. This can simply be explained by the gestures recorded by users. A last point on these recognition rate, is that it doesn't decrease even if recognizers are trained with both left-handed and right-handed samples. Therefore, there is no visible difference for the gestures if it is performed by a left-handed or right-handed user.

Now that we have discussed about the overall recognition rate for each recognizer, we can focus on the recognition rate for each gesture. The figures 5.3, A.1 and A.2 are showing that recognizers performed better with simple gestures. By simple gestures, we mean all gestures which are simply a (multi-)finger(s) swipe or a 3D gesture. Each participant who recorded these gestures performed those gestures almost the same way. Whereas, when they had to record, for instance *Hold&Swipe*, *ZoomPinch*[3], they were feeling more uncomfortable and unfamiliar with these gestures. So, this results in a lower accuracy.

[3]For the Pinch gesture the 3DTouchpad surface do not fit well for these one hand gesture. Each one touch touched with their nails and so their gesture lose in accuracy

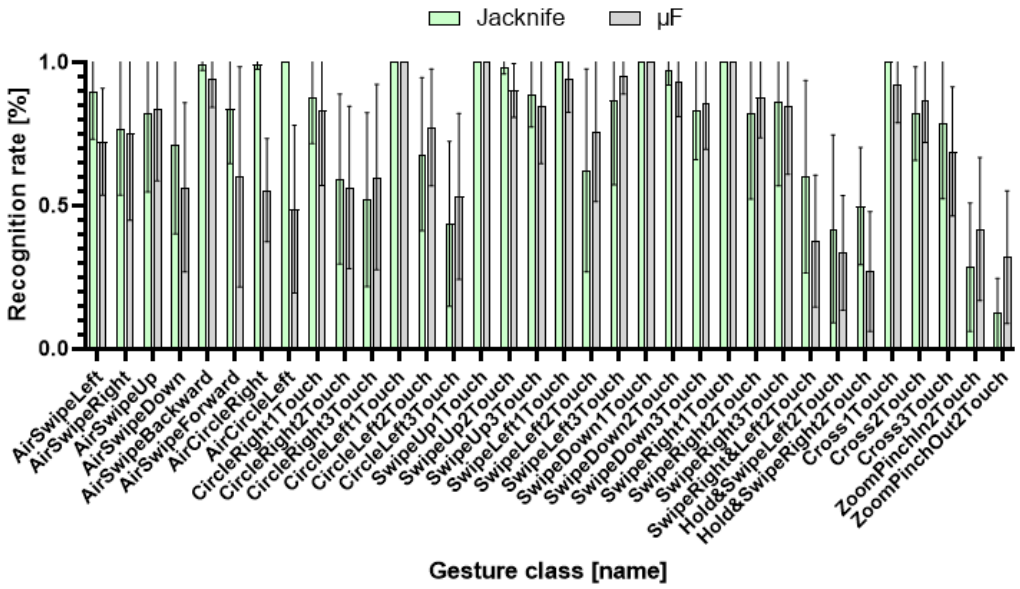



Figure 5.3: Recognition Rate of the two recognizers for each gesture based on the Right-hand Dataset

### Recognition Rate Depending On The Number Of Training Templates

The figure 5.4, figure A.3 and figure A.4 are showing how the recognition rate for each recognizer is converging as the number of training templates is increasing. On figure 5.4, $\mu$F recognizer **interquatile range(IQR)** is always lower than the jacknife one and it is also converging faster than jacknife. We can also see that the median, when the two recognizers are trained with only one template, is higher for $\mu$F recognizer. But, when we use more templates for training, Jacknife's median seems to be always better.

However, all the previous analysis were only based on the right-handed samples and the figure 5.4. They are not necessarily true for the left-handed dataset and the whole dataset. The median for the left-handed dataset and whole dataset is strictly higher at each point for jacknife. We can also see that figure A.3 is converging much faster than the two other figures. Jacknife is especially converging fast and this time even faster than $\mu$F. This phenomena is probably coming from the fact that less users were interviewed to record gestures for this dataset and each one recorded four times each gesture whereas it was only two for the right-handed dataset. Therefore, there is less variation between samples as you can train the recognizer with four samples performed by the same user. This is why the results are much more higher than the other datasets. We can also see that jacknife is converging near 100%. Maybe, if we increase the number of training templates,

jacknife will be almost able to reach the 100% recognition rate.

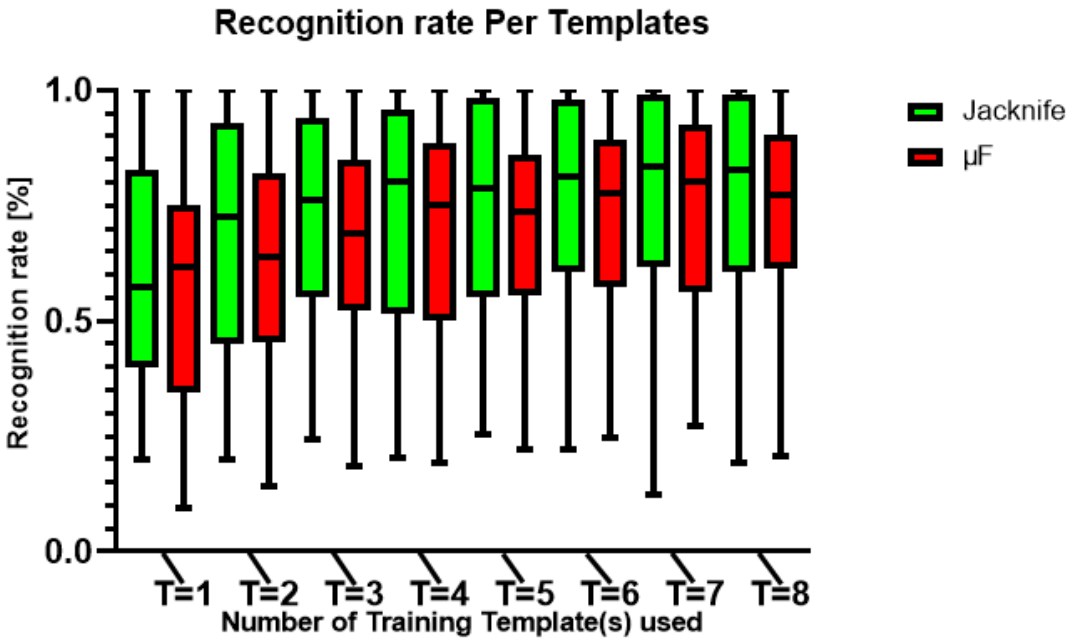



Figure 5.4: Box Plot of the recognition for each recognizer depending on their training templates based on the Right-Handed Dataset.

### Execution Time Rate

The time execution rate of each recognizer is the same according to the three different datasets. This rate increases according to the amount of used training templates. On the figure 5.5 and figure 5.6, we can see the distribution of the amount of training templates. The two box plots of each recognizer have two different shapes. The Jacknife's time execution rate increases slowly when we increase the amount of training templates. With 8 training templates, the median has increased from **0.12 ms**. Whereas $\mu$F's execution time rate has increased exponentially according to the amount of training templates. With 7 training templates, the rate has increased from 1.85ms. Therefore, if we used this recognizer with a huge amount of training templates, this recognizer is not scalable. A recognizer has to process data quickly especially for gesture recognition. However, for this thesis, $\mu$F recognizer is still very efficient as we don't have to test with more than 8 training templates.

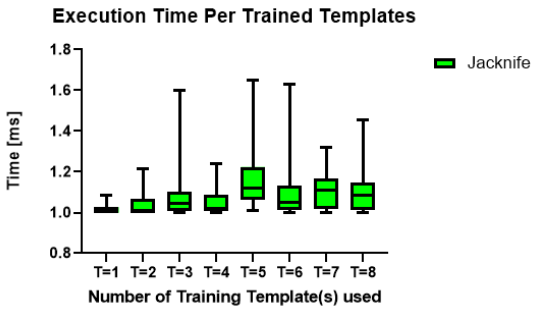



Figure 5.5: Boxplot based on the execution time of each gesture with Jacknife recognizer.

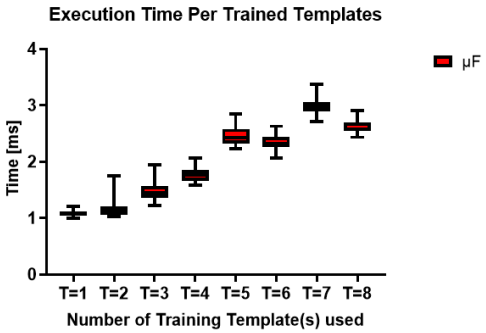



Figure 5.6: Boxplot based on the execution time of each gesture with $\mu$F recognizer.

Figures A.6, A.5, and A.7 are showing the time execution for each gesture from each dataset. The data are based when each recognizer is trained with 8 different templates. We can easily see on these three graphics that the time execution is the same for all the different gestures according to their recognizer. We can also determine that the $\mu$F execution with 8 training templates is about 2 or 3 times the execution from jacknife.

### Mixed The Two Recognizers

After comparing the two recognizers in a User-Independant scenario, it could be interesting to mix the two recognizers and to analyze its results. There is a way to perform this combination inside *QuantumLeap* [50], it is called Multiple Recognizers. It gets the result of a gesture from each recognizer. It returns the one which seems to be the best for the framework. But, we are still exposed to some problems. For example:"If one recognizer gets the right gesture but the other gets a wrong gesture, the framework can choose the one with the wrong gesture". In the other way, it can also solve a wrong gesture. On the figure 5.7, we can clearly see the difference between the two recognizers and the mixed recognizer. The mixed recognizer is often better than $\mu$F but never better than jacknife. However, we can see that the **min** and **max** are lower than the two separated recognizers. This seems to be the only good points in the comparison with Jacknife. An another bad point with this mixed recognizer is related to the execution time. This mixed form computes the test with each recognizer. Therefore, it results in the addition of the execution time from each recognizer. As discussed in the previous subsubsection, we can easily know that this mixed recognizer is not scalable with an high number of training templates due to $\mu$F recognizer. With all the different points mentioned in this paragraph, we can determine that is not interesting to mix the two recognizers

together.

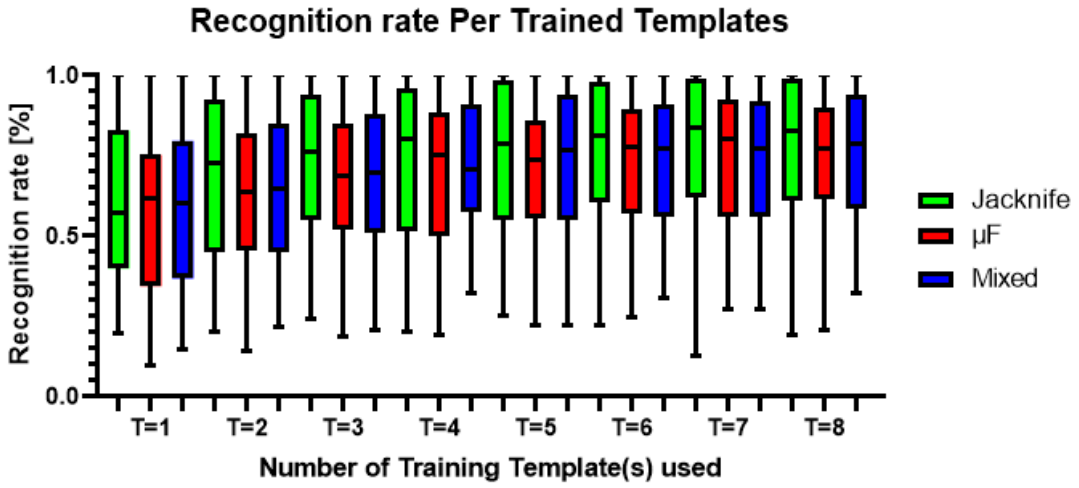



Figure 5.7: Box Plot comparing recognition rate from each recognizer and when they are associated. Their rates depend on the number of training templates

### 5.4.2 User-Dependent Scenario

Now that we have defined all the different results related to the user-independent testing, let's analyze the user-dependent testing. We are not going to discuss about the execution time as it is exactly the same observation as the one detailed in the previous subsection.

#### Recognition Rate

Figure 5.8 is the box plot based on the right-handed dataset. Unfortunately as we only got two records from each user for this dataset, we could train each recognizer with only one template. We can directly see that the recognition rate and the median are very high compared with results from the user-independent scenario. Jacknife recognition rate is equal to **79.81%** and $\mu$F recognition rate is equal to **70.31%**. These results were expected as they were trained with gestures performed by the same user. It can be really interesting to see this testing scenario with plenty of templates from the same user. The converging can probably achieve a very high recognition rate

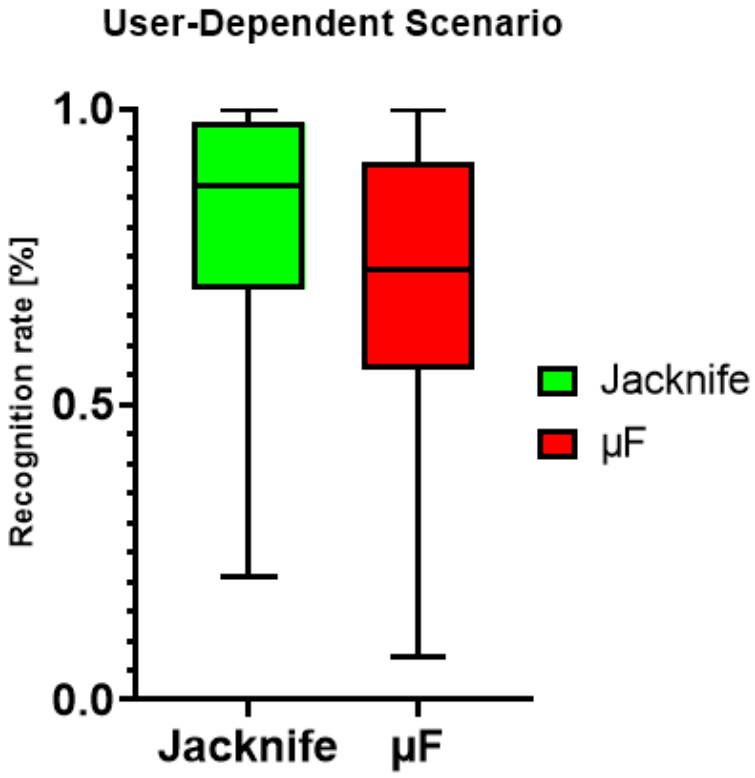



Figure 5.8: Box Plot comparing recognition rate from each recognizer in a user-dependent scenario.

### 5.4.3 Confusion Matrix and Confusion Wheel

Now that we have detailed the previous results, let's focus on recognizers errors. It could be interesting to analyze with which other gestures a recognizer is wrong. The confusion matrix [51] and confusion wheel [22] are two tools that allow to understand the link between errors. The figure A.8 and the figure 5.9 are representing the confusion matrix and confusion wheel based on our gestures set. These two different models were built using this online website: **Link**. All relevant informations that are used are defined in this paper [53]. The confusion matrix data were obtained from the jacknife recognizer trained with 8 different samples in a UI scenario. Thanks to the figures, we can easily determine which gestures are not optimal together. For example, the recognizer often confuses the two PinchIn and PinchOut gestures together. The 2D gestures with 3 fingers have also more errors as it is harder to process data. We can also see that the recognizer fails with the Hold&Swipe gestures. It often recognizes classical swipe. These tools are very important, if

we need a very high recognition rate for an application. Based on the confusion matrix and wheel, we can optimize our dataset and only choose gestures which never fail together. It could also be interesting to understand why they failed to try to improve recognizers.

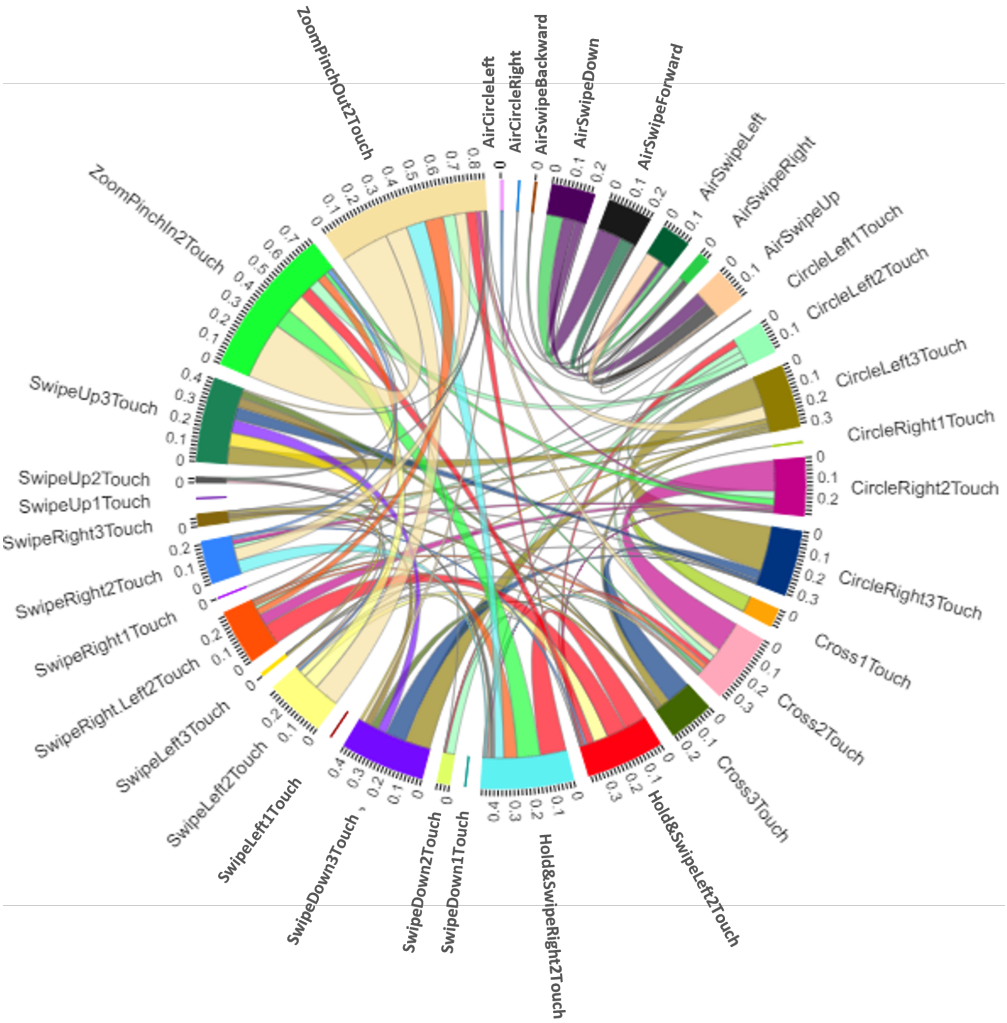



Figure 5.9: Confusion Wheel from the jacknife result on our gestures set.

## 5.5 Conclusion

In this chapter, we observed how recognizers reacted to Air+Touch data. The results for $\mu$F and Jacknife were pretty good. The two recognizers met our criteria we fixed at the beginning of the chapter. Jacknife looks to be the best recognizer

inside QuantumLeap with our gestures set definition. It always provides the highest recognition rate and the lowest execution time rate in each tested scenario. $\mu$F results were not bad but are still a step below. In the last part of the section, we also showed what gestures were not working together thanks to the confusion tools. This chapter is very important for this thesis as we proved that Air+Touch gestures can have a high rate. And at the same time, we proved the reliability of our gestures set definition.

# Chapter 6

# Gesture Elicitation Study

## 6.1 Overview

This chapter analyses two gesture elicitation studies about the 3DTouchpad. They were realized by two groups of students. One group realized as part of the **LINFO1311-Interface Home Machine** course, the second one as part of the **LSMG2008-Informatique de gestion** course. Each *GES* were supervised by Pr. Jean Vanderdonckt and his assistants. Section 6.2 introduces the point of gesture elicitation study and its purpose. Section 6.3 explains the different contexts of experiment defined by each group. Section 6.4 mentions all the interesting results emerging from the two gesture elicitation studies.

## 6.2 Gesture Elicitation Study

First, it is important to answer this question: *What is a "gesture elicitation study"?*
A gesture elicitation study can be defined with the following steps [59, 31, 16]:

1. Gather participants in a similar room.
2. Participants have to complete the consent form.
3. Participants have to complete another form which collects their general skills with some different technologies.
4. Ask to participants to initiate a gesture which is the most appropriate to a specific action in daily life such as "Turn on Tv, Turn light off". These specific actions are called referent. The order of referents has to be randomly mixed for each participant.

5. Each time an action is performed, the gesture is filmed and the time taken by each participant for each gesture is recorded. Device raw data can also be saved.

6. Ask participants to fill some questionnaires and a feedback form relevant to the study subject.

If these requirements are satisfied, the empirical results of the study can be very interesting. We are able to see some emerging results for each referent. Sometimes a referent may not obtain a low majority. This is also interesting to understand why this result occurs. We can also analyze some correlation between the different participants. For example, we can imagine that the students prefer a gesture whereas workers prefer another gesture. Therefore, we can analyze cluster based on their age, their situation and their gender.

A complete and detailed analysis of these statistics are important. It allows to see the different trends related to each user specification. In the industrial world, we could define a gestures set for each referent specifically adapted to each user. In the point of our thesis, it helps us to see what are the trend, which gesture are preferred and which gesture seems the most natural. This lead us to think about different points of view without being too biased by our opinion.

## 6.3 Experiment

This section compares the similarity, the difference and the context of the experiment between the two studies.

### 6.3.1 Referent Definition

The first step in the experiment is to define a coherent list of referents. They defined **19** referents in the two studies. Each referent was performed by **30** participants. We dressed the following table to resume all the referents defined by the two different groups.

| ID | LINFO1311 | LSMG2008 |
|---|---|---|
| 1 | Select | Turn TV on |
| 2 | Scroll Up | Turn TV off |
| 3 | Scroll Down | Start player |
| 4 | Turn Up the Volume | Turn Up The Volume |
| 5 | Turn Down the volume | Turn Down The Volume |
| 6 | Brightness up | Go to the next item in the list |
| 7 | Brightness down | Go to the previous item in the list |

| | | |
|---|---|---|
| 8 | Switch page right | Turn air conditioning on |
| 9 | Switch page left | Turn air conditioning off |
| 10 | Shut down computer | Turn light on |
| 11 | Change window of opened app | Turn light off |
| 12 | Zoom in | Brighten lights |
| 13 | Zoom out | Dim lights |
| 14 | Open sub-option menu | Turn heat on |
| 15 | Go back | Turn heat off |
| 16 | Go forth | Turn alarm on |
| 17 | Copy | Turn alarm off |
| 18 | Paste | Answer phone call |
| 19 | Select many | End phone call |

Table 6.1: List of referents

As you can, see some referent are equivalent as the *Turn up the volume on/off, Brightness up/down*. But in general, they are quite different. We will be able to see if some different referents are resulting in the same gesture. We also will analyze if the 4 similar referents from the two studies result in the same gesture.

### 6.3.2 Pre-Study context

There is another point that differs from the two studies. It's the explanations before the study. The group from LINFO1311 course let each participant play with the touchpad and the GUI [36] provided by Microchip [35]. So, these explanations allow each participant to be more familiar with the touchpad and they understand that the touchpad can sense only one point in 3D mode. Whereas the LSMG2008's group didn't mention this point. So people were making some hand pose gesture or gesture with two fingers in air. Unfortunately, 3DTouchpad can't acquire these gestures. This fact is bit annoying for the comparison. But, this choice releases another interesting point: *What gesture are performed if the 3D sensor was improved?* By improved we mean that touchpad can sense multi-fingers and pose hand gestures with the 3D sensors.

## 6.4 Result

The two studies produced interesting results. We are going to detail them in several subsections where we are going to talk about the two **GES**. The graphics from the two studies are displayed together. To avoid any understanding problems, we

defined that the left one is always the one corresponding to the **LINFO1311** group whereas the right one is the one from **LSMG2008** study. To be less redundant, we also renamed the LINFO1311 to the study number one and the LSMG2008 to the study number two.

### 6.4.1 Analysis of participants' specifications

The first statistic is the one related to the gender of the participants. The more the proportion is balanced the more the result is interesting. For the first study, the ratio is quite unbalanced. Only **9** women were participants(**30%**). And so **21** men performed the experiment. Therefore, the men ratio is equal to **70%**. However in the second study, the ratio is very interesting. The men ratio is equal to **53%**(16 participants) and the women ratio is equal to **47%**(14 participants).

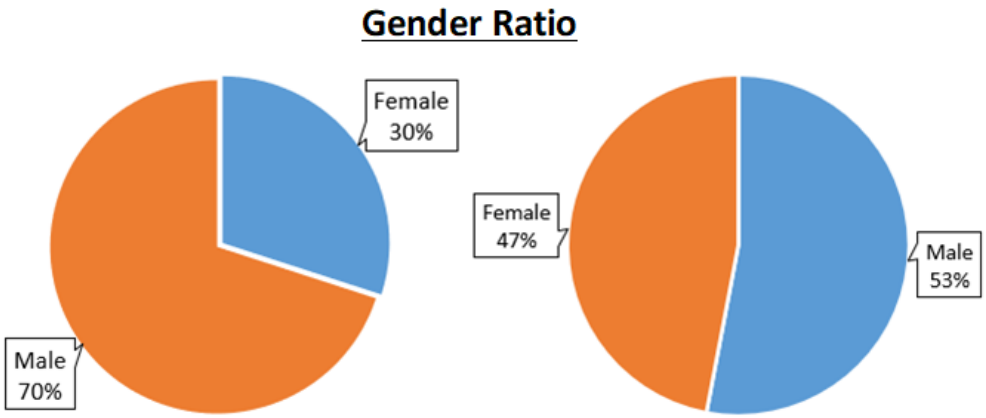



Figure 6.1: Graph representing the ratio between women and men who performed the two studies.

The second statistic about participants personal information is the profession ratio. The **profession ratio** is defined as the amount of participants who are **Student, Employed, Executive, Other, Retired and Independent**. Here, the ratio of each study is quite similar. The only difference is that there are four more student in the first than the second.

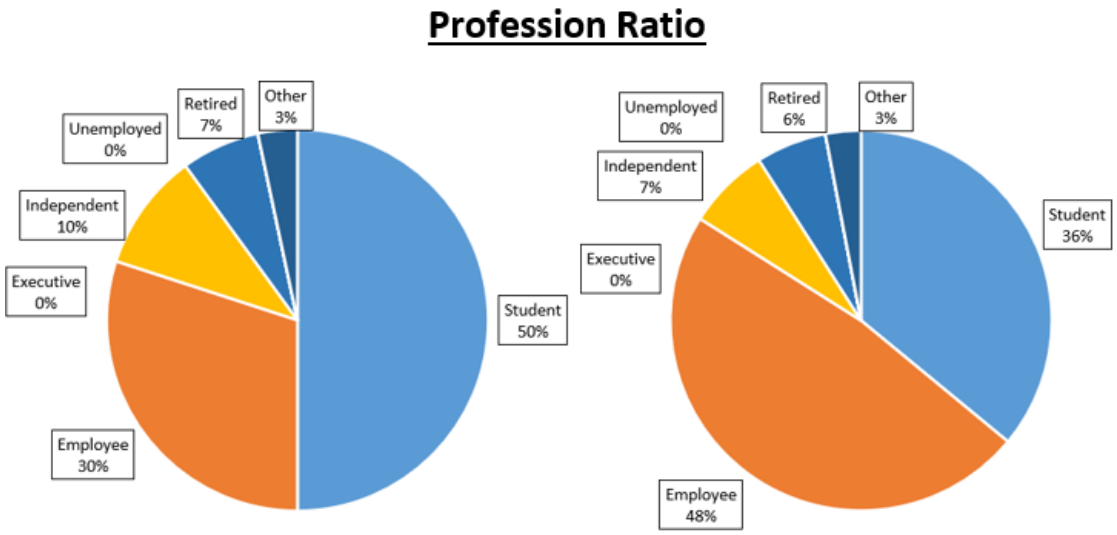



Figure 6.2: Graph representing the profession ratio of participants who performed the two studies.

The third statistic is related to the age of each participant. The figure 6.3 represents all the participants grouped by a "10 years" cluster and divided by their gender. The average age of each study is **34-35** years. This average is pretty good.

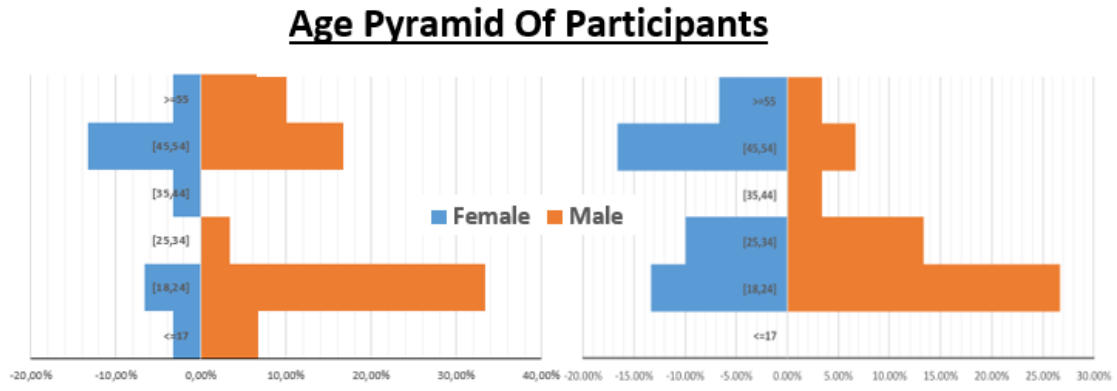



Figure 6.3: Pyramidal representation of "age" cluster. Each "age" cluster is differentiated by the participant's gender.

The last graph of the section represents the usage frequency with some device as the Microsoft Kinect, a traditional computer, a tablet, ... Each participant has to choose a value between 1 to 7. The value 7 means that the participant is very comfortable with the device and uses it often. Whereas value 1 means that the participant never used it. The participants often felt conformable with

the touchpad. This come from the fact that the computer and smartphone usage frequency is high. The touchpad could look like a computer pad or a smartphone surface.

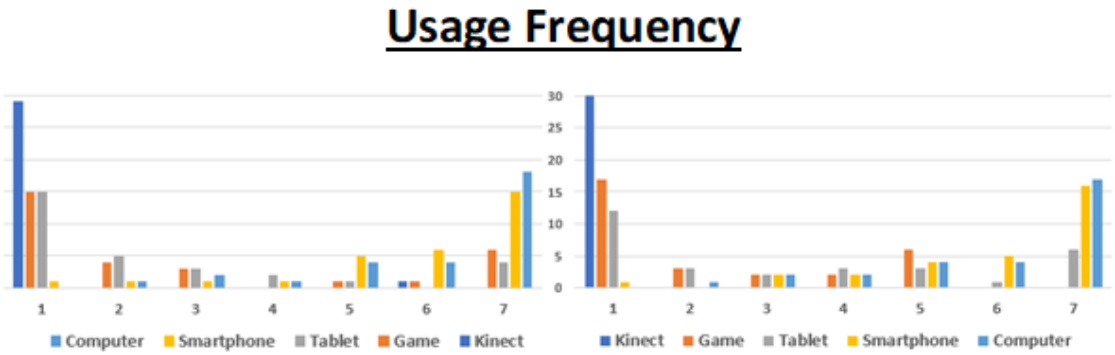



Figure 6.4: Usage Frequency proportion for each device.

The last interesting point concerns the **natural hand** of each participant. In the first study, **28** participants were right-handed(**93%**) and only **2** were ambidextrous(**7%**). In the second study, **24** participants were right-handed(**80%**) and **6** were left-handed(**20%**). The distribution of the first study is quite annoying because there is no left-handed participant. As mention earlier, gesture can be different depending on the used hand. So, it could have been an interesting point to analyze the distribution.

#### Conclusion

The four analyzed figures are very interesting. One of the goal of a gesture elicitation study is to choose a selection of participants where the diversification is maximized. The two studies have shown that they have sought to meet this objective. The first study has two limited distribution: the natural hand and gender distribution. But we wish that they will not affect the study result quality.

### 6.4.2 Thinking Time Analysis

As almost all referents are different according to the study, the subsection is divided into two parts. In each part, we analyze the average thinking time for each referent (figures 6.5,6.7) per each participant (figures 6.6,6.8).

#### First Study

Participants take less time to think about the referent when it is a familiar one. *Select, Scroll, Zoom* referent are familiar because they are used everyday. Unfortu-

nately, we think that each referent was asked in the same order, which strongly reduces the interpretation of this graph. For example, there are some linked referents like *Scroll Up* and *Scroll Down*. So, if you always make the first one before the second one, you will probably make the same gesture but in opposite direction. So the thinking time is more reduced for the second referent than for the first referent. In order to alleviate this problem, we can give each referent in a random order. On the second figure, we can see that people that are more than 45 years old take more time than people who are less than 45 years old.

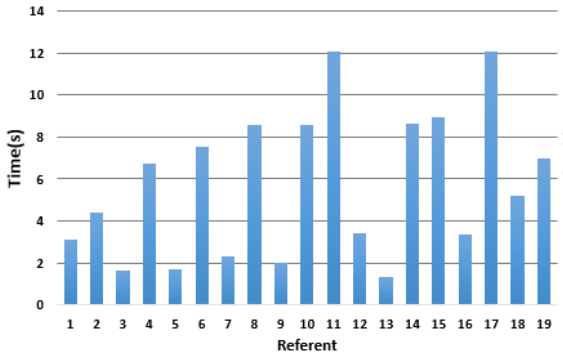



Figure 6.5: Average thinking time for each referent.

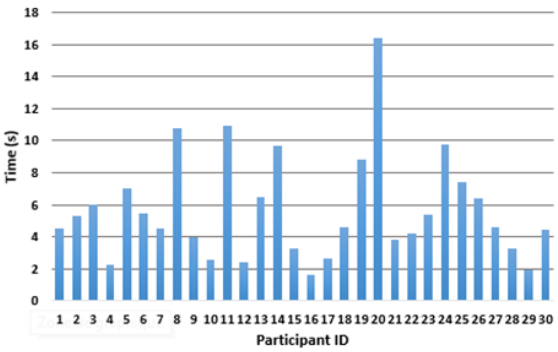



Figure 6.6: Average thinking time for each participant.

### Second Study

First of all, they presented each referent in a random order to the participants. So we don't have the same problem with the linked "referents" as mentioned in the first study. They also chose referents which seems less familiar with the touchpad use as *turn off TV, light on*. The time thinking variation is less present between the referent unlike the previous study. However, the uncommon referent as brightness up, turn on air conditioning need more thinking time. This is probably because these referents are rarely done for participants in their daily life. The figure 6.8 is showing that older participants need more times.

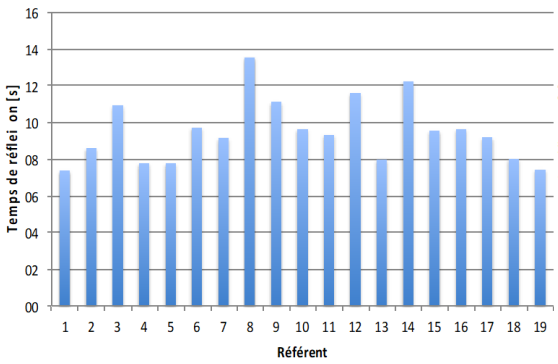



Figure 6.7: Average thinking time for each referent.

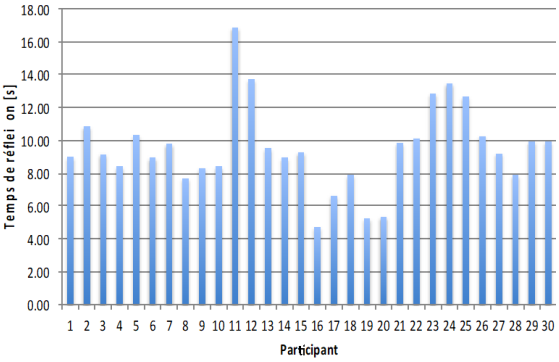



Figure 6.8: Average thinking time for each participant.

### 6.4.3 Agreement Rate

In this subsection we are going to discuss about the agreement rate for each referent. The agreement rate is defined as a measure of how the participants are agreeing on the gestures performed for a specific referent. If the rate is high, a majority of participants provided the same gesture. Whereas, if the rate is low, the participants often proposed quite different gestures for a same referent. The agreement rate based on Vatavu et al. [57] can be defined with the following formulas:

$$AR(r) = \frac{\sum_{i<j} \delta_{i,j}}{n \cdot (n-1)/2} \tag{6.1}$$

The **n** is the amount of participants. $\delta_{i,j}$ is equal to 1 if the participant i and the participant j are agreeing over referent r. Otherwise, the value is equal to 0. The section is divided into two parts for each study. Each part contains a table which is filled with the referent id, the most used gesture and its agreement rate. Each subsection contains a graphical representation of the agreement for each referent.

#### First Study

Thanks to the table 6.2, we can easily see that group who made the experiments were unfortunately a bit too vague about their gestures name. They often reported a swipe with multiple fingers. It would have been interesting to have the direction of each finger and their numbers. Apart from that, the participants seem to widely prefer 2D gestures. Agreement rate for some referents is huge. Swipe with multiple fingers were often performed. In our gestures set definition, we get a lot of swipe with single/multiple finger(s). We also got other gestures performed as 2D circle, 3D Swipe. This shows that our gestures set is representative of users.

Participants also performed other gestures as making a *cross* on the 3DTouchpad, the *hold&Swipe* gesture, the *SwipeLeft&Right* with two fingers. These gestures looked promising and so we decided to include them in our gestures set definition. This shows again how a GES could step up our work. It is important not to restrict ourselves to our ideas.

| ID | Gesture Proposed | Agreement Rate |
|---|---|---|
| 1 | One or more taps | 0.755 |
| 2 | Swipe with Multiple Fingers | 0.755 |
| 3 | Swipe with Multiple Fingers | 0.701 |
| 4 | Swipe with Multiple Fingers | 0.194 |
| 5 | Swipe with Multiple Fingers | 0.232 |
| 6 | Swipe with Multiple Fingers | 0.303 |
| 7 | Swipe with Multiple Fingers | 0.374 |
| 8 | Swipe with Multiple Fingers | 0.316 |
| 9 | Swipe with Multiple Fingers | 0.323 |
| 10 | 2D Circle with one Finger | 0.11 |
| 11 | Swipe with Multiple Fingers | 0.166 |
| 12 | Multiple Fingers on the Pad and Then Open | 0.701 |
| 13 | Multiple Fingers on the Pad and Then Close | 0.701 |
| 14 | One or more taps | 0.456 |
| 15 | Swipe with Multiple Fingers | 0.181 |
| 16 | 3D Swipe | 0.144 |
| 17 | One or more taps | 0.099 |
| 18 | One or more taps | 0.105 |
| 19 | 2D Swipe then Single Tap Tap | 0.196 |

Table 6.2: Agreement Rate and associated gesture for each referent.

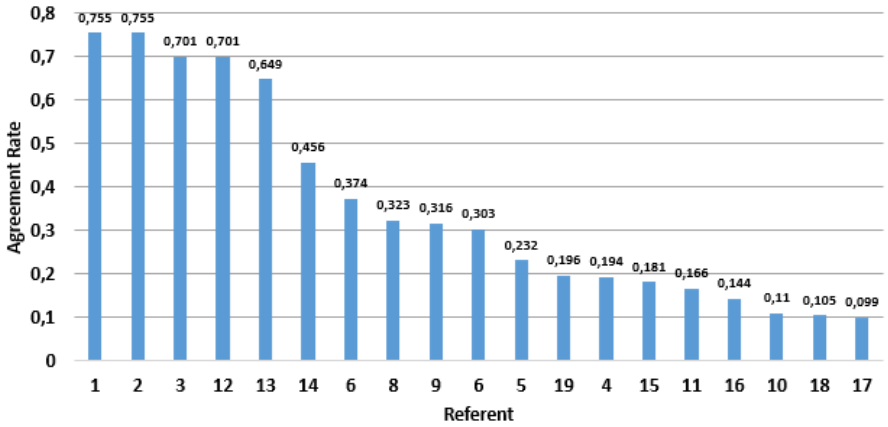



Figure 6.9: Agreement rate for each referent.

### Second Study

The second study is more representative than the first one. They defined the swipes performed by each participant in details. And once again, we observe that swipe gestures have a high agreement rate for some referents. Again, a lot of gesture performed by participants are defined inside our gestures set. The 2D gestures were widely preferred even if there are more 3D gestures in the table 6.3 than in table 6.2. Unfortunately, some of the performed 3D gestures aren't possible with the touchpad as the open palm, close fist. So, it leads us to think that it can be really promising to evolve the 3D sensing of the touchpad. It may be probably done by combining it with another device. The reflection about the comparison between the same referents that we introduced earlier in the chapter, are not relevant because the first group was too vague.

| ID | Gesture Proposed | Agreement Rate |
|---|---|---|
| 1 | Tap with One Finger | 0.090 |
| 2 | 3D Swipe From North to South | 0.069 |
| 3 | Tap with One Finger | 0.163 |
| 4 | Clockwise 2D Circle | 0.255 |
| 5 | Anti-clockwise 2D Circle | 0.237 |
| 6 | Swipe from North to South | 0.303 |
| 7 | Swipe from South to North | 0.283 |
| 8 | 2D Pinch In with Two Fingers | 0.046 |
| 9 | 2D Swipe from West to East | 0.034 |
| 10 | Open Palm | 0.149 |
| 11 | Close Fist | 0.159 |
| 12 | 3D Anti-clockwise Circle with hand | 0.108 |

| | | |
|---|---|---|
| 13 | 3D Anti-clockwise Circle with One Finger | 0.106 |
| 14 | turn a valve Above touchpad | 0.099 |
| 15 | 3D Swipe Up with Palm | 0.076 |
| 16 | Snap your fingers | 0.034 |
| 17 | Knock | 0.030 |
| 18 | 2D Swipe From West to East | 0.172 |
| 19 | 2D Swipe From East to West | 0.071 |

Table 6.3: Agreement Rate and associated gesture for each referent.

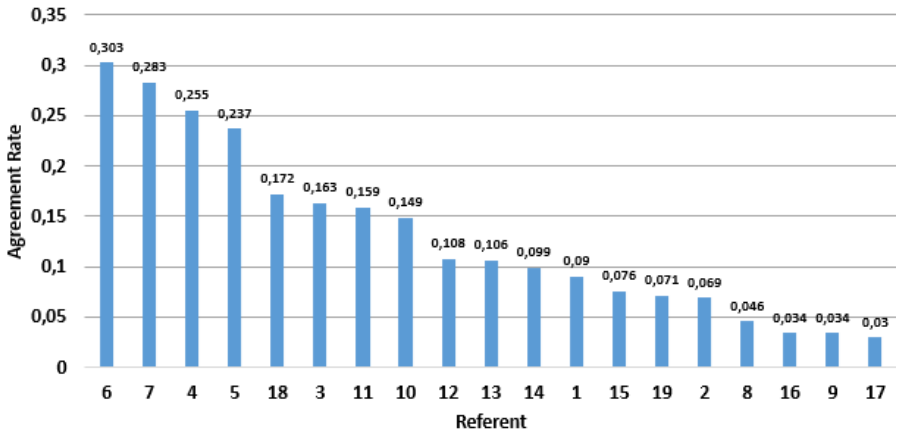



Figure 6.10: Agreement rate for each referent.

### 6.4.4 Goodness Of Fit

After a gesture was performed, each participant was asked to rate his gesture. This rating must be between 0 and 10. If they thought that the gesture they performed for a referent was absolutely not fitting with the touchpad, they simply rated it with the zero value. Whereas, if they thought that the gesture they performed was perfectly fitting with the context, they could rate it with a 10 value. This is what goodness of fit [39] means. On figure 6.11, we can see that the goodness of fit average for each referent is between **6** and **9**. This means that participants were often satisfied with their gestures. They found it adequate to the situation.

Now that we get the goodness of fit and the agreement rate for each referent, we can see that gestures from our gestures set look relevant. For example, *2D Swipe Up* has a good agreement rate and a good goodness of fit. This analysis can also be done for other gestures: *AirCirle, 2DCircle, 2DSwipe and many others.* Once again, this proves that our gestures set is interesting.

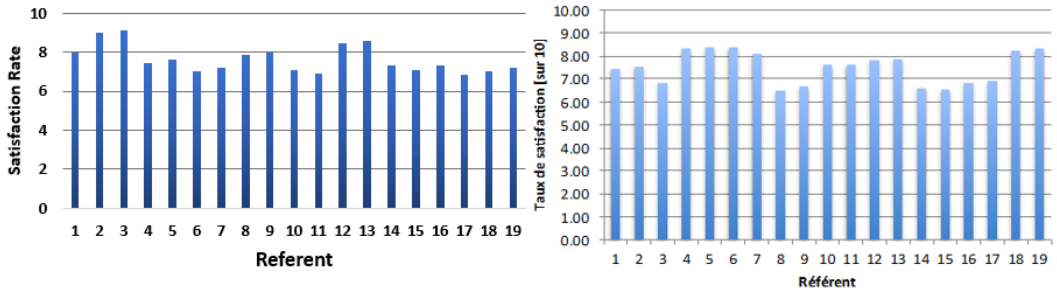



Figure 6.11: Goodness of fit for each referent

### 6.4.5 PSSUQ

PSSUQ means the **Post-Study System Usability Questionnaire** [15]. It is a questionnaire which is collecting answers for 16 questions. Each question is answered with a value between one to seven. The value one means that the participant "Strongly disagree with the question". The value seven means that the participant "Strongly agree with the question". Unfortunately, the first study lacks a lot of answers on these question. So we won't discuss about the first study here. The table 6.4 enumerate the 16 questions.

| ID | Question |
|---|---|
| 1 | Overall, I am satisfied with the ease of use of this system |
| 2 | The system was simple to use |
| 3 | I was able to complete the tasks and scenarios quickly with this system |
| 4 | I felt comfortable with the system |
| 5 | The system was easy to learn |
| 6 | I expect to become productive quickly using this system |
| 7 | The system provided me with clear error messages to solve the problems |
| 8 | I was able to correct every mistake simply and quickly |
| 9 | Information provided by the system, such as online help , messages, documentation, was clearly provided by the system |
| 10 | I easily found the information I needed |
| 11 | The information was useful to me in completing the tasks and scenarios |
| 12 | Information was clearly presented on the screen |
| 13 | The system interface was nice |
| 14 | I liked using the system interface |
| 15 | The system had all the functions I wanted |
| 16 | Overall, I am satisfied with the system |

Table 6.4: The 16 Questions of the PSSUQ.

The figure 6.12 and 6.13 are the global score average by participants and by questions. The average score for each question is between 4 and 5. So that means that participants often agree with *PSSUQ*. On the second graph, we can see that some participants are below the mean. Unfortunately, there is not a specific characteristic to understand these low scores.


Figure 6.12: Global score per questions

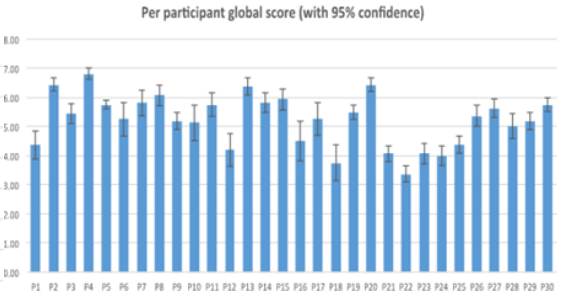



Figure 6.13: Global score per participants

## 6.5 Conclusion

In this chapter, we analyzed two *Gesture Elicitation Studies* about the 3DTouchpad [34]. We analyzed their results and compared them together. It was really interesting to have the point of view of 60 different participants. We also understood how participants that have never worked with the touchpad were interacting with it. We took inspiration from these studies as we added 6 gestures to our gestures set. As expected, participants were also feeling more confident with the 2D gestures than with the 3D. This chapter is useful for this thesis as it confirm the reliability of our gestures set definition.

# Chapter 7

# Conclusion and Future Work

## 7.1 Final Conclusion

In this chapter, we are going to mention all the different contributions which were discussed on each chapter of this thesis.

In Chapter 1, we posed our context problem and all the relevant hypothesis, questions, research methods to our subject. We have also pointed out that there is a lack of content in the literature for Air+Touch recognition.

In Chapter 2, we gathered all papers, articles, books relating to our subject. We analyzed them in details. With all these resources, we built a huge Air+Touch gestures catalog. Each kind of found gestures were described inside this catalog. This was necessary to build a relevant gestures set definition.

In Chapter 3, we overviewed all our implementation choices and works. We managed to retrieve the raw data from the 3DTouchpad. We integrated the touchpad inside QuantumLeap. We proved that it was possible to use the framework with other devices than LeapMotion sensor. We also demonstrated that the actual implementation is problematic with Multiple-sensors working at the same time. Therefore, we proposed a way to fix it. So now, new devices can be added and used together.

In Chapter 4, we defined our own gestures set definition based on the catalog from chapter 2 and the two gesture elicitation studies results in chapter 6. We defined and populated our three datasets thanks to 12 participants. We also described the different possible Air+Touch combinations. They seemed to be very interesting.

In Chapter 5, we realized a benchmarking between the different implemented recognizers inside QuantumLeap. We quickly saw that $\mu$F and Jacknife were performing much better than the others. Therefore, we restricted our tests to these two recognizers only. We tested them according to two scenarios: User-Independent,

User-Dependent. The results for our gestures set were very good. It showed that Air+Touch gestures can have an high recognition rate.

In Chapter 6, we analyzed two different gesture elicitation studies based on the 3DTouchpad. It confirmed the reliability of our gestures set definition. We also expanded our gestures set based on these results. These results were based on over 60 participants.

## 7.2 Future Work

During the whole thesis, we travelled through a lot of different steps. Some of them led us think about what can be done to expand our work. These future prospects can be envisaged to continue the work done in this thesis:

- **Framework Integration**: Continue the framework development. Continue to integrate new devices as smart ring, kinect and many others.
- **Framework Multiple-Sensors**: As we proved that it was possible to use multiple-sensors together inside the framework, the next step is to combine multiple devices as leapMotion, touchpad together. Therefore, it will be easy to add some new 2D device and 3D device. This provides an easy way to continue the Air+Touch recognition research.
- **Air+Touch Application**: Create a dedicated application which is designed for Air+Touch combinations as we mentioned in chapter 4 and perform gesture elicitation study to optimize the combinations. Make a comparative study about Air combinations only, Touch combinations only and Air+Touch combinations. This could prove that Air+Touch combinations are more suitable than the two other types alone.
- **Expand our Gestures Set and our Dataset**: Try to expand our gestures set with some new interesting gestures. Collect plenty of new samples about each gesture. With these new things, testing can be really promising. It will be also interesting to see if new recognizers are fitting with the Air+Touch data. An other point for the future, is to perform several gesture elicitation studies dedicated to Air+Touch gestures.

# List of Figures

# List of Tables

# Bibliography


[1] Iso. iso/iec 25010 - software quality product standard. standard, international standard organization, geneva, 2019.

[2] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[3] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[4] Hrvoje Benko, Edward W Ishak, and Steven Feiner. Cross-dimensional gestural interaction techniques for hybrid immersive environments. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.*, pages 209–216. IEEE, 2005.

[5] BMW. Gesture Controls | BMW Genius How-To. `https://www.youtube.com/watch?v=wqvAPskg_k0`, 2015.

[6] Andrew Bragdon, Rob DeLine, Ken Hinckley, and Meredith Ringel Morris. Code space: touch+ air gesture hybrid interactions for supporting developer meetings. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 212–221, 2011.

[7] Andrew Bragdon, Eugene Nelson, Yang Li, and Ken Hinckley. Experimental analysis of touch-screen gesture designs in mobile environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 403–412, 2011.

[8] Fabio Marco Caputo, Pietro Prebianca, Alessandro Carcangiu, Lucio Davide Spano, and Andrea Giachetti. A 3 cent recognizer: Simple and effective retrieval and classification of mid-air gestures from single 3d traces. In *STAG*, pages 9–15, 2017.

[9] Alessandro Carcangiu, Lucio Davide Spano, and Andrea Giachetti. A 3 cent recognizer: Simple and effective retrieval and classification of mid-air gestures from single 3d traces.

[10] Xiang 'Anthony' Chen, Julia Schwarz, Chris Harrison, Jennifer Mankoff, and Scott E. Hudson. Air+touch: Interweaving touch & in-air gestures. UIST '14, page 519–525, New York, NY, USA, 2014. Association for Computing Machinery.

[11] Xiang'Anthony' Chen, Julia Schwarz, Chris Harrison, Jennifer Mankoff, and Scott E Hudson. Air+ touch: interweaving touch & in-air gestures. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 519–525, 2014.

[12] Liste des ports logiciels. `https://fr.wikipedia.org/wiki/Liste_de_ports_logiciels`.

[13] Docs.microsoft.com. Winsock2.h header - win32 apps. `https://docs.microsoft.com/en-us/windows/win32/api/winsock2/`.

[14] ExtremeTech. Gestic brings 3d, camera-free gesture recognition to low-cost devices, `https://www.extremetech.com/extreme/140286-gestic-brings-3d-camera-free-gesture-recognition-to-low-cost-devices`.

[15] Ann Fruhling and Sang Lee. Assessing the reliability, validity and adaptability of pssuq. *AMCIS 2005 proceedings*, page 378, 2005.

[16] Bogdan-Florin Gheran, Jean Vanderdonckt, and Radu-Daniel Vatavu. Gestures for smart rings: empirical results, insights, and design implications. In *Proceedings of the 2018 Designing Interactive Systems Conference*, pages 623–635, 2018.

[17] Bogdan-Florin Gheran, Radu-Daniel Vatavu, and Jean Vanderdonckt. Ring x2: Designing gestures for smart rings using temporal calculus. In *Proceedings of the 2018 ACM Conference Companion Publication on Designing Interactive Systems*, pages 117–122, 2018.

[18] Andy Green. Libwesocket.org. `https://libwebsockets.org/`.

[19] Georg Hackenberg, Rod McCall, and Wolfgang Broll. Lightweight palm and finger tracking for real-time 3d gesture control. In *2011 IEEE Virtual Reality Conference*, pages 19–26. IEEE, 2011.

[20] Franz Huber, Oliver Arold, Florian Willomitzer, Svenja Ettl, and Gerd Häusler. 3d body scanning with flying triangulation. *DGaO Proceedings 2011*, page P30, 2011.

[21] Microsoft Visual Studio IDE. `https://visualstudio.microsoft.com/fr/vs/`.

[22] Eva Kaderabek and Panittha Suwannajang. Confusion matrix viz.

[23] Cem Keskin, Ayse Erkan, and Lale Akarun. Real time hand tracking and 3d gesture recognition for interactive interfaces using hmm. *Icann/Iconipp*, 2003:26–29, 2003.

[24] Anthony R Khoury. Motion capture for telemedicine: a review of nintendo wii, microsoft kinect, and playstation move. *Journal of the International Society for Telemedicine and eHealth*, 6:e14–1, 2018.

[25] Sven Kratz and Michael Rohs. A $3 gesture recognizer: simple gesture recognition for devices equipped with 3d acceleration sensors. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 341–344, 2010.

[26] Sven Kratz and Michael Rohs. Protractor3d: a closed-form solution to rotation-invariant 3d gestures. In *Proceedings of the 16th international conference on Intelligent user interfaces*, pages 371–374, 2011.

[27] Hansaem Lee and Junseok Park. Hand gesture recognition in multi-space of 2d/3d. *Int. J. Comput. Sci. Netw. Secur*, 15:12–16, 2015.

[28] Kyung-Suk Lhee and Steve J Chapin. Buffer overflow and format string overflow vulnerabilities. *Software: practice and experience*, 33(5):423–460, 2003.

[29] Net library. `https://nodejs.org/api/net.html`.

[30] MIT License. `https://libwebsockets.org/git/libwebsockets/tree/LICENSE`.

[31] Nathan Magrofuoco and Jean Vanderdonckt. Gelicit: A cloud platform for distributed gesture elicitation studies. *Proceedings of the ACM on Human-Computer Interaction*, 3(EICS):1–41, 2019.

[32] S. Malik and Joseph Laszlo. Visual touchpad: a two-handed gestural input device. In *ICMI '04*, 2004.

[33] Nicolai Marquardt, Ricardo Jota, Saul Greenberg, and Joaquim A Jorge. The continuous interaction space: interaction techniques unifying touch and gesture on and above a digital surface. In *IFIP Conference on Human-Computer Interaction*, pages 461–476. Springer, 2011.

[34] Microchip. 3DTouchPad Sell Sheet. `https://ww1.microchip.com/downloads/en/DeviceDoc/00001835A.pdf`.

[35] Microchip. https://www.microchip.com/.

[36] Microchip. 3dtouchpad: Product details. https://www.microchip.com/DevelopmentTools/ProductDetails/DM160225.

[37] Microchip. Gestic technology. https://www.microchip.com/en-us/solutions/touch-and-gesture-technologies/gestic-technology-basics.

[38] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(3):311–324, 2007.

[39] Stanley A Mulaik, Larry R James, Judith Van Alstine, Nathan Bennett, Sherri Lind, and C Dean Stilwell. Evaluation of goodness-of-fit indices for structural equation models. *Psychological bulletin*, 105(3):430, 1989.

[40] Jörg Müller, Gilles Bailly, Thor Bossuyt, and Niklas Hillgren. Mirrortouch: combining touch and mid-air gestures for public displays. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*, pages 319–328, 2014.

[41] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[42] Michael Ortega and Laurence Nigay. Airmouse: Finger gesture for 2d and 3d interaction. In *IFIP Conference on Human-Computer Interaction*, pages 214–227. Springer, 2009.

[43] Carole Plasson, Dominique Cunin, Yann Laurillau, and Laurence Nigay. 3d tabletop ar: A comparison of mid-air, touch and touch+ mid-air interaction. In *Proceedings of the International Conference on Advanced Visual Interfaces*, pages 1–5, 2020.

[44] Leigh Ellen Potter, Jake Araullo, and Lewis Carter. The leap motion controller: a view on sign language. In *Proceedings of the 25th Australian computer-human interaction conference: augmentation, application, innovation, collaboration*, pages 175–178, 2013.

[45] Wolfgang Pree. Hot-spot-driven framework development. *Framework*, 2:B1, 2000.

[46] Napa Sae-Bae, Nasir Memon, Katherine Isbister, and Kowsar Ahmed. Multitouch gesture-based authentication. *IEEE transactions on information forensics and security*, 9(4):568–582, 2014.

[47] Alexander Schick, Florian van de Camp, Joris Ijsselmuiden, and Rainer Stiefelhagen. Extending touch: towards interaction with large-scale surfaces. In *Proceedings of the ACM international conference on interactive tabletops and surfaces*, pages 117–124, 2009.

[48] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14, 2008.

[49] Han Albrecht Schmid. Systematic framework design by generalization. *Communications of the ACM*, 40(10):48–51, 1997.

[50] Arthur Sluÿters. A framework for engineering gesture-based user interfaces using the leap motion controller : development and applications, `http://hdl.handle.net/2078.1/thesis:26507`.

[51] Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.

[52] Eugene M Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R Pittman, and Joseph J LaViola Jr. Jackknife: A reliable recognizer with few samples and many modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 5850–5861, 2017.

[53] Andreas Theissler, Simon Vollert, Patrick Benz, Laurentius A Meerhoff, and Marc Fernandes. Ml-modelexplorer: An explorative model-agnostic approach to evaluate and compare multi-class classifiers. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 281–300. Springer, 2020.

[54] Radu-Daniel Vatavu. The effect of sampling rate on the performance of template-based gesture recognizers. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 271–278, 2011.

[55] Radu-Daniel 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*, pages 4667–4679, 2017.

[56] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. Gestures as point clouds: a $ p recognizer for user interface prototypes. In *Proceedings of the 14th ACM international conference on Multimodal interaction*, pages 273–280, 2012.

[57] Radu-Daniel Vatavu and Jacob O Wobbrock. Between-subjects elicitation studies: Formalization and tool support. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3390–3402, 2016.

[58] Craig Villamor, Dan Willis, and Luke Wroblewski. Touch gesture reference guide. *Touch Gesture Reference Guide*, 2010.

[59] Santiago Villarreal-Narvaez, Jean Vanderdonckt, Radu-Daniel Vatavu, and Jacob O Wobbrock. A systematic review of gesture elicitation studies: What can we learn from 216 studies? In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, pages 855–872, 2020.

[60] Andrew D Wilson. Touchlight: an imaging touch screen and display for gesture-based interaction. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 69–76, 2004.

[61] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[62] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. State of the art on 3d reconstruction with rgb-d cameras. In *Computer graphics forum*, volume 37, pages 625–652. Wiley Online Library, 2018.

# Appendix A

# Benchmarking Figures

## A.1 Recognition Rate

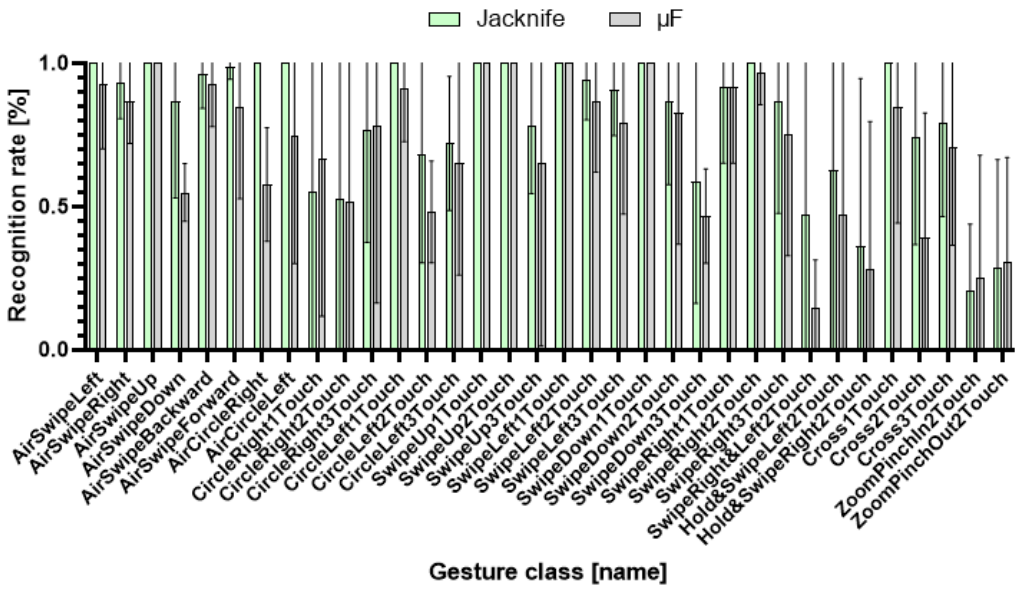



Figure A.1: Recognition Rate of the two recognizers for each gesture based on the Left-hand Dataset

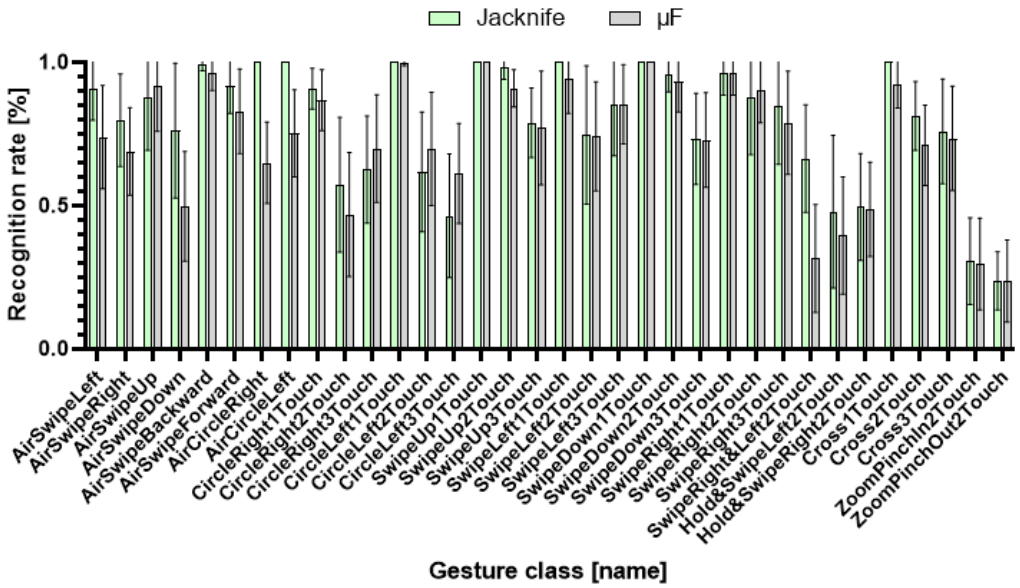



Figure A.2: Recognition Rate of the two recognizers for each gesture based on the whole Dataset

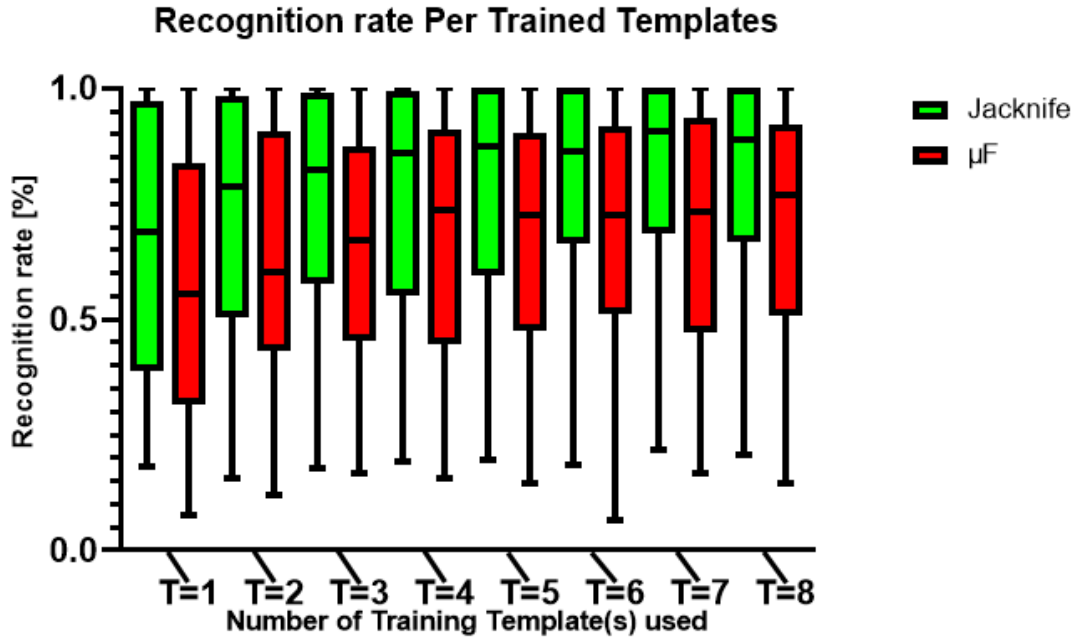



Figure A.3: Box Plot of the recognition for each recognizer depending on their training templates based on Left-Handed Dataset.

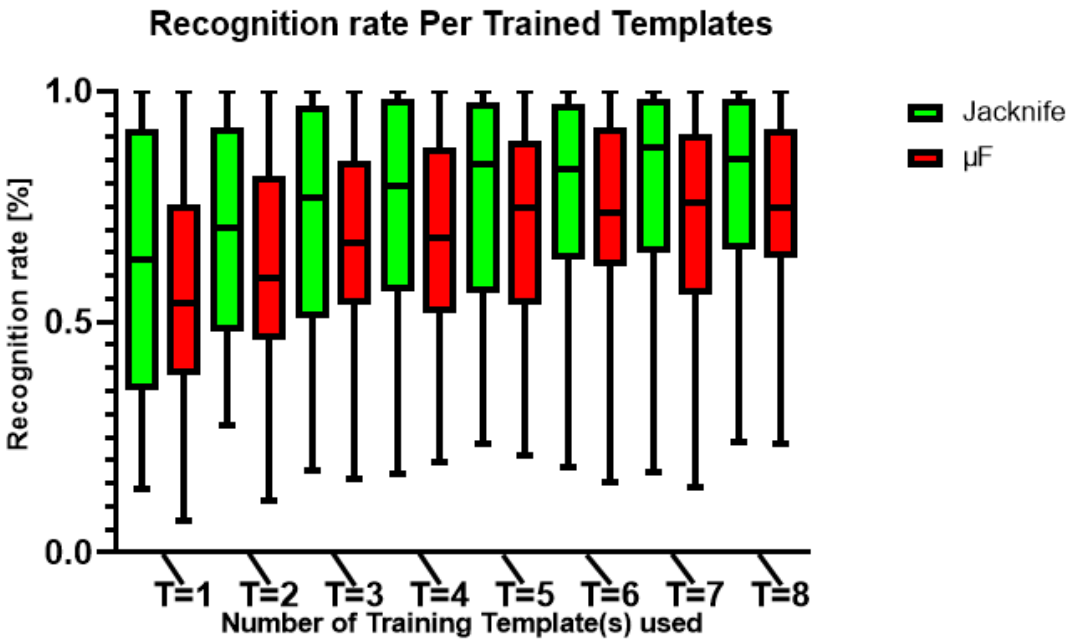



Figure A.4: Box Plot of the recognition for each recognizer depending on their training templates based Whole-Handed Dataset.

## A.2 Execution Time Rate

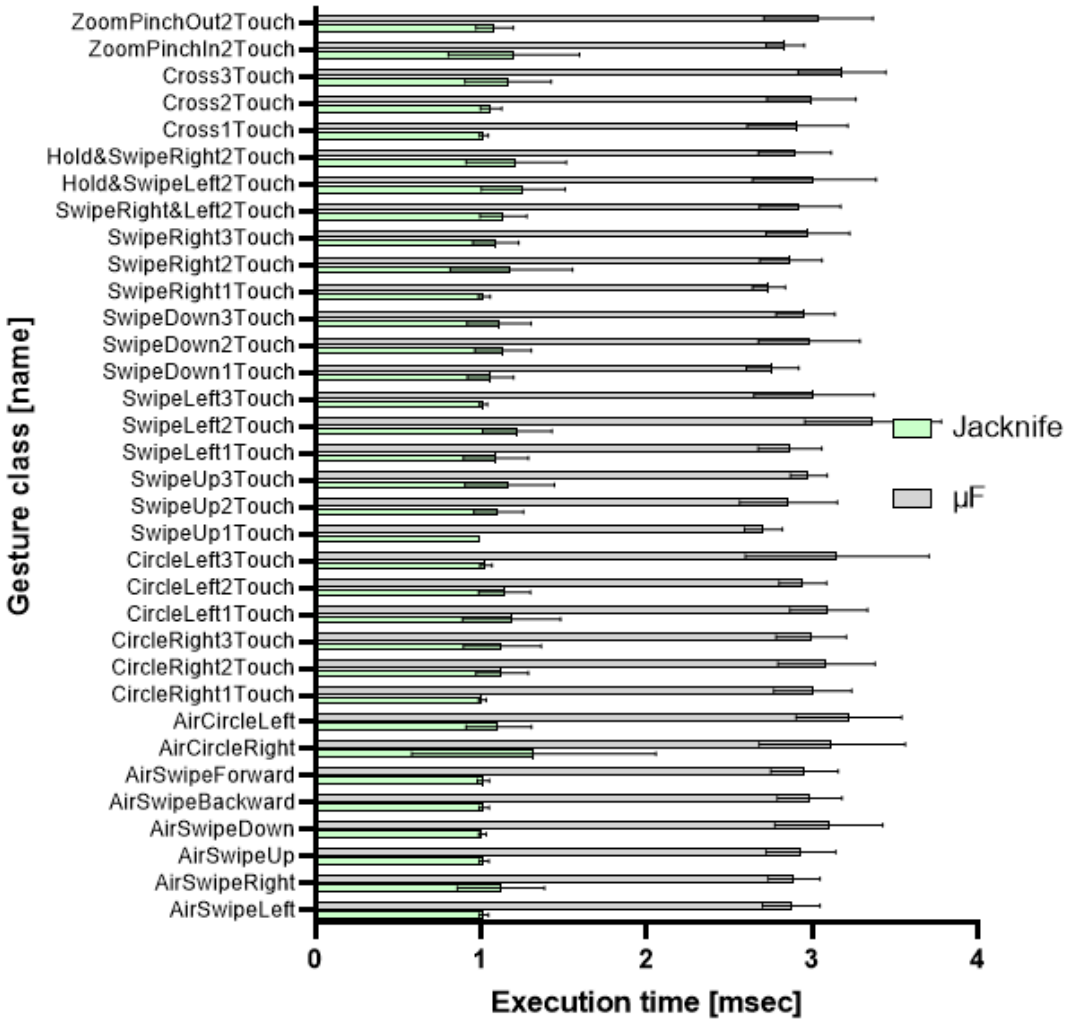



Figure A.5: Execution Time rate for each gesture from the right-handed dataset

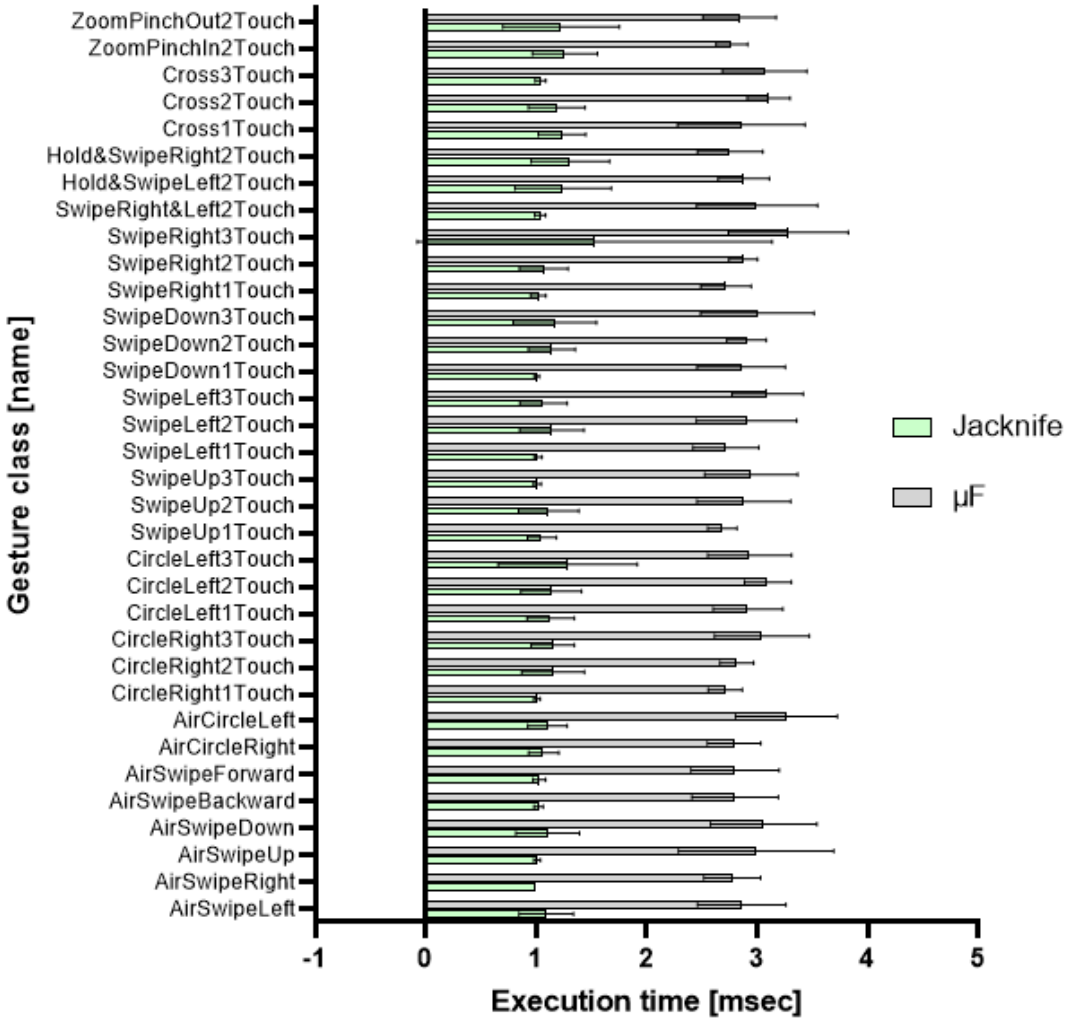



Figure A.6: Execution Time rate for each gesture from the left-hand dataset

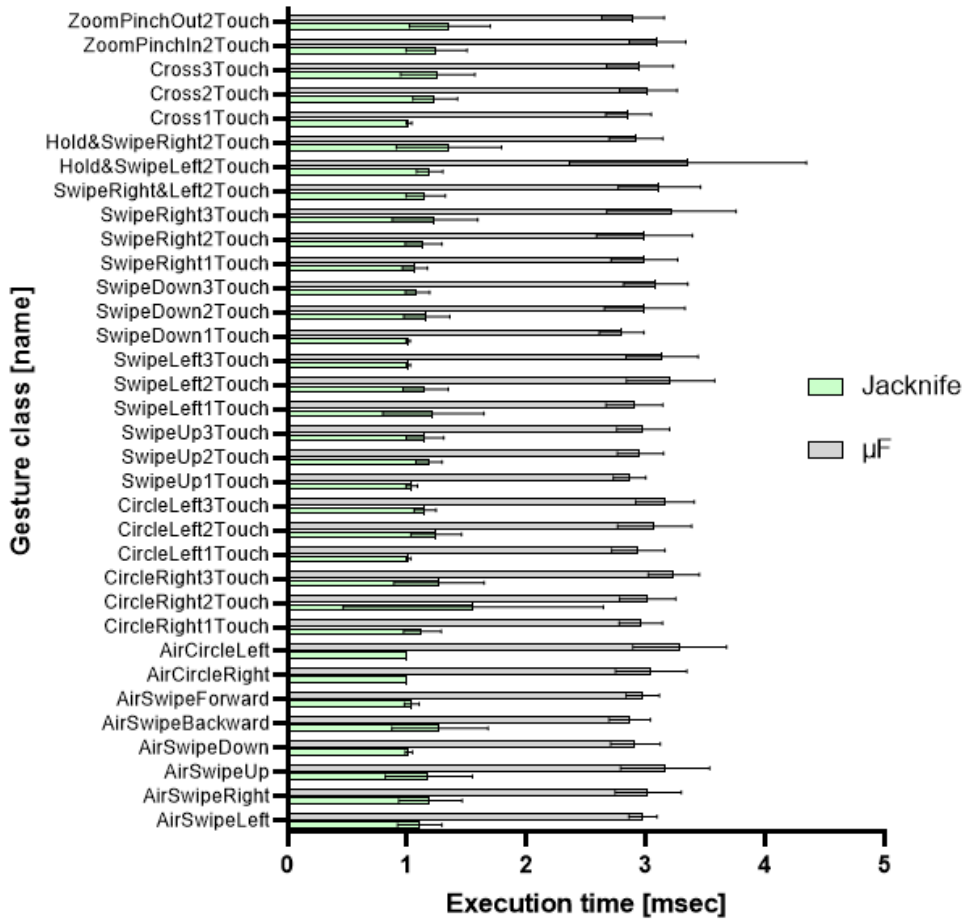



Figure A.7: Execution Time rate for each gesture from the whole dataset

## A.3 Confusion Matrix

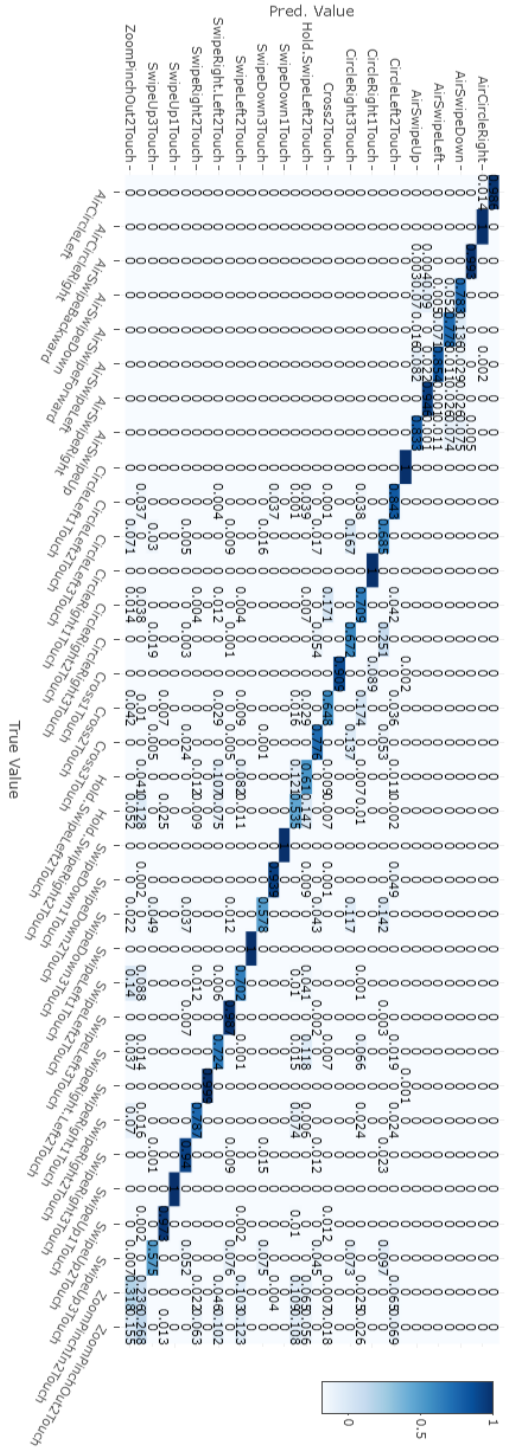



Figure A.8: Confusion Matrix of our gestures set from Jacknife recognizer.

# Appendix B

# Code Implementation

## B.1 User Guide of 3DTouchpad

If you want to use the 3DTouchpad on windows please follow these following steps:

1. Download 3DTouchpad SDK.
2. Download Visual Studio[21] with two extensions: Development Desktop in c++ and Development Linux in c++.
3. Get the touchpad.exe from our c script.
4. Always put the .exe file in a folder where *mchp_hmi.dll* file from touchpad SDK is present.
5. Everything is ok and you can use our implementation

## B.2 Windows 3DTouchpad C Script

```
1
2 #include <winsock2.h>
3
4 #include <hmi_api.h>
5 #include <stdio.h>
6 #include <io.h>
7 #include <string.h>
8 #include <Windows.h>
9
10 #pragma comment(lib, "WS2_32.lib")
11
12
```

```
13
14 void output_fingers(hmi2d_finger_pos_list_t* fingers,int bool, int
       count, SOCKET s)
15 {
16     int i;
17     int finger_nbr;
18
19     if (fingers->count >= 4) {
20         finger_nbr = 3;
21     }
22     else {
23         finger_nbr = fingers->count;
24     }
25     if (finger_nbr == 1) {
26
27         char data[100];
28         size_t n = sprintf(data, "{\"type\":\"2DTouch%d\",\"count\":%
       d,\"x\":[%d],\"y\":[%d]}", finger_nbr, count, fingers->entry[0].x,
        fingers->entry[0].y);
29         if (bool == 1) {
30             puts(data);
31         }
32         send(s, data, n, 0);
33
34     }
35     else if (finger_nbr == 2) {
36         char data[100];
37         size_t n = sprintf(data, "{\"type\":\"2DTouch%d\",\"count\":%
       d,\"x\":[%d,%d],\"y\":[%d,%d]}", finger_nbr, count, fingers->entry
       [0].x, fingers->entry[1].x, fingers->entry[0].y, fingers->entry
       [1].y);
38         if (bool == 1) {
39             puts(data);
40         }
41         send(s, data, n, 0);
42     }
43     else if (finger_nbr == 3) {
44         char data[100];
45         size_t n = sprintf(data, "{\"type\":\"2DTouch%d\",\"count\":%
       d,\"x\":[%d,%d,%d],\"y\":[%d,%d,%d]}", finger_nbr, count, fingers
       ->entry[0].x, fingers->entry[1].x, fingers->entry[2].x, fingers->
       entry[0].y, fingers->entry[1].y, fingers->entry[2].y);
46         if (bool == 1) {
47             puts(data);
48         }
49         send(s, data, n, 0);
50     }
51 }
52
```

```
53 int main(int argc, char** argv) {
54
55     printf("Enter a Port Number: ");
56
57     int port_number = 0;
58     scanf("%d", &port_number);
59
60     if ( port_number < 1 || 65552 < port_number ) {
61         puts("Invalid_Port Number");
62         return;
63     }
64
65     int iterations_number = 0;
66     printf("Enter a number for iteration value: ");
67
68     scanf("%d", &iterations_number);
69
70
71     if (iterations_number < 0) {
72         puts("Invalid_Port Number");
73         return;
74     }
75
76     int bool = 0;
77
78     char* show_info = malloc(sizeof(char*)*16);
79     printf("Do you want to display the data information? (y or n): ")
    ;
80
81     scanf("%s", show_info);
82     if (strlen(show_info) != 1 ) {
83         puts("Invalid Value Entered, information will not be
    displayed");
84     }
85     else if (show_info[0] == 'y') {
86         bool = 1;
87     }
88
89     free(show_info);
90
91
92
93     int i;
94
95     hmi_t* hmi = hmi_create();
96
97
98     WSADATA wsa;
99     SOCKET s;
```

```
100     char* message;
101
102     printf("\nInitialising Winsock...");
103     if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
104     {
105         printf("Failed. Error Code : %d", WSAGetLastError());
106         return 1;
107     }
108
109     printf("Initialised.\n");
110
111     //Create a socket
112     if ((s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
    INVALID_SOCKET)
113     {
114         printf("Could not create socket : %d", WSAGetLastError());
115     }
116
117     printf("Socket created.\n");
118
119
120
121
122     struct sockaddr_in server;
123
124     server.sin_family = AF_INET;
125     server.sin_port = htons(port_number);
126     server.sin_addr.s_addr = inet_addr("127.0.0.1");
127    //bind(s, (struct sockaddr*)&server, sizeof(server));
128     //Connect to remote server
129
130
131
132     if (connect(s, (struct sockaddr*)&server, sizeof(server)) < 0)
133     {
134         puts("connect error");
135         return 1;
136     }
137
138     puts("Connected");
139
140
141     const int stream_flags = hmi3d_DataOutConfigMask_TouchInfo |
    hmi3d_DataOutConfigMask_xyzPosition |
    hmi3d_DataOutConfigMask_SDData;
142     const int com_flags = hmi2d_com_finger_positions |
    hmi2d_com_3d_messages;
143     const hmi3d_signal_t* sd = NULL;
144     const hmi3d_position_t* pos = NULL;
```

```
145     const hmi2d_finger_pos_list_t* fingers = NULL;
146
147      //Initialize all variables and required resources of hmi
148     hmi_initialize(hmi);
149
150     sd = hmi3d_get_sd(hmi);
151     pos = hmi3d_get_position(hmi);
152     fingers = hmi2d_get_finger_positions(hmi);
153     // Open connection to the device
154     if (hmi_open(hmi) < 0) {
155         fprintf(stderr, "Could not open connection to device.n");
156         return -1;
157     }
158
159     // Enable 2D/3D message communication only
160     if (hmi2d_set_com_mask(hmi, com_flags, hmi2d_com_all) !=
    HMI_NO_ERROR)
161     {
162         fprintf(stderr, "Could not set communication mode.n");
163         return -1;
164     }
165
166     //  Disable all active features
167     if (hmi2d_set_active_mask(hmi, hmi2d_active_gesture_recognition,
    hmi2d_active_all) != HMI_NO_ERROR) {
168         fprintf(stderr, "Could not set active mask.n");
169         return -1;
170     }
171
172     /* Set 2D/3D mixed operation mode */
173     if (hmi2d_set_operation_mode(hmi, hmi2d_mixed_mode) !=
    HMI_NO_ERROR) {
174         fprintf(stderr, "Could not set 3D only operation mode.n");
175         return -1;
176     }
177     int touch_detect = 1;
178     if (hmi3d_set_auto_calibration(hmi, 1) < 0 ||
179         hmi3d_select_frequencies(hmi, hmi3d_all_freq) < 0 ||
180         hmi3d_set_touch_detection(hmi, touch_detect) < 0 ||
181         hmi3d_set_approach_detection(hmi, 0) < 0)
182     {
183         fprintf(stderr, "Could not reset device to default state.n");
184         return -1;
185     }
186     if (hmi3d_set_output_enable_mask(hmi, stream_flags, 0,
187         hmi3d_DataOutConfigMask_OutputAll) < 0)
188     {
189         fprintf(stderr, "Could not set output-mask for streaming.n");
190         return -1;
```

```
191     }
192     int count = 0;
193 
194     int while_count = 0;
195 
196 
197     char data[100];
198 
199     while (count < iterations_number) {
200         while (!hmi3d_retrieve_data(hmi, 0) && count <
    iterations_number) {
201             if (pos != NULL) {
202                 //  Output the position
203                 while_count = while_count + 1;
204                 if (while_count % 5 == 0) {
205                     int x = pos->x;
206                     int y = pos->y;
207                     int z = pos->z;
208                     int n = sprintf(data, "{\"type\":\"3D\",\"count
    \":%d,\"x\":%d,\"y\":%d,\"z\":%d}", count, pos->x, pos->y, pos->z)
    ;
209                     if (bool == 1) {
210                         puts(data);
211                     }
212                     send(s, data, n,0);
213                     Sleep(30);
214                     count = count + 1;
215                 }
216             }
217         }
218         while (hmi2d_retrieve_data(hmi) == HMI_NO_ERROR &&count <
    iterations_number) {
219 
220             if (fingers != NULL) {
221                 while_count = while_count + 1;
222                 if (while_count % 5 == 0) {
223                     output_fingers( fingers, bool, count, s);
224                     Sleep(30);
225                     count = count + 1;
226                 }
227             }
228         }
229     }
230     // Reset default communication mode
231     if (hmi2d_set_com_mask(hmi, hmi2d_com_default, hmi2d_com_all) !=
    HMI_NO_ERROR)
232     {
233         fprintf(stderr, "Could not reset default communication mode.n
    ");
```

```
234     }
235
236     // Reset default features
237     if (hmi2d_set_active_mask(hmi, hmi2d_active_default,
    hmi2d_active_all) != HMI_NO_ERROR)
238     {
239         fprintf(stderr, "Could not reset default active mask.n");
240     }
241
242     // Reset to mixed operation mode
243     if (hmi2d_set_operation_mode(hmi, hmi2d_mixed_mode) !=
    HMI_NO_ERROR) {
244         fprintf(stderr, "Could not restore mixed operation mode.n");
245     }
246
247     // Close connection to device
248     hmi_close(hmi);
249
250     //Release further resources that were used by hmi
251     hmi_cleanup(hmi);
252     hmi_free(hmi);
253
254     closesocket(s);
255     WSACleanup();
256
257
258     printf("END of the Execution\n");
259     printf("Shutting Down\n");
260     Sleep(1000);
261     return 0;
262 }
```

## B.3 Dataset Recorder

```
net = require('net');
const { table } = require('console');
var fs = require('fs');
var dataset = {
    table: []
}
var count = 0;

net.createServer(function (socket) {

  socket.on('data', function (data) {
        console.log(JSON.parse(data));
        dataset.table.push(JSON.parse(data))
  });
```

```
  socket.on('error', function(e){
        console.log(e);
  });

  socket.on('close', function(e){
        var json = JSON.stringify(dataset)
        fs.writeFile(`droitier/participant7/
        SwipeRight1Touch-${count}.json`, json, function(err) {
            if (err) throw err;
            console.log(`Record n ${count}`);
            }
        );
        dataset.table = []
        count = count + 1
  });

}).listen(5000);

console.log("Chat server running at port 5000\n");
```

## B.4 QuantumLeap Integration

### B.4.1 3DTouchpad Loader

```
const fs = require('fs');
const path = require('path');

const GestureSet = require('../../../../framework/gestures/gesture-set')
.GestureSet;
const GestureClass = require('../../../../framework/gestures/gesture-class')
.GestureClass;
const StrokeData = require('../../../../framework/gestures/stroke-data')
.StrokeData;
const Stroke = require('../../../../framework/gestures/stroke-data')
.Stroke;
const Path = require('../../../../framework/gestures/stroke-data')
.Path;
const Point3D = require('../../../../framework/gestures/point')
.Point3D;


function loadDataset(name, datasetPath, identifier, sensorPointsNames){
    let gestureSet = new GestureSet(name);
    let dirPath = datasetPath;
    let gestureIndex = 0;
```

```
fs.readdirSync(dirPath, {withFileTypes: true}).filter(dirent =>
!dirent.isFile()).map(dirent => dirent.name).forEach((user_dir) => {

    let gestureClassDirPath = path.join(dirPath,user_dir);

    fs.readdirSync(gestureClassDirPath).forEach((sample) =>{

        let rawGesturePath = path.join(gestureClassDirPath,sample);
        let rawGestureData = JSON.parse(fs.readFileSync(rawGesturePath));

        let filename = sample.split(".")[0].split("-");
        let gestureName = filename[0].split("#")[0];

        let infosupp = undefined;
        if (filename[0].split("#").length > 1) {
            infosupp = filename[0].split("#")[1];
        }
        let id = filename[1];

        let gestureData = new StrokeData(parseInt(user_dir),id,undefined);

        let string3D = "3D_"
        let string2D1 = "2DTouch1_"
        let string2D2 = "2DTouch2_"
        let string2D3 = "2DTouch3_"


        let strokePath3D = new Path(string3D.concat(identifier));
        let strokePath2D1 = new Path(string2D1.concat(identifier));
        let strokePath2D2 = new Path(string2D2.concat(identifier));
        let strokePath2D3 = new Path(string2D3.concat(identifier));
        gestureData.addPath(string3D.concat(identifier),strokePath3D);
        gestureData.addPath(string2D1.concat(identifier),strokePath2D1);
        gestureData.addPath(string2D2.concat(identifier),strokePath2D2);
        gestureData.addPath(string2D3.concat(identifier),strokePath2D3);


        let stroke3D = new Stroke(string3D.concat(identifier));
        let stroke2D1 = new Stroke(string2D1.concat(identifier));
        let stroke2D2 = new Stroke(string2D2.concat(identifier));
        let stroke2D3 = new Stroke(string2D3.concat(identifier));

        for(let i = 0 ; i < rawGestureData.table.length ; i++ ) {

            let basic_point = new Point3D(10,10,10,10);

            var value = rawGestureData.table

            if(value[i].type === "2DTouch1"){
```

```
                stroke2D1.addPoint(new Point3D(value[i].x[0],
                value[i].y[0],0,value[i].count));

                stroke3D.addPoint(basic_point);
                stroke2D2.addPoint(basic_point);
                stroke2D3.addPoint(basic_point);
            }
            if(value[i].type === "2DTouch2"){
                stroke2D2.addPoint(new Point3D(value[i].x[0],
                value[i].y[0],0,value[i].count));

                stroke3D.addPoint(basic_point);
                stroke2D1.addPoint(basic_point);
                stroke2D3.addPoint(basic_point);

                stroke2D2.addPoint(new Point3D(value[i].x[1],
                value[i].y[1],0,value[i].count));

                stroke3D.addPoint(basic_point);
                stroke2D1.addPoint(basic_point);
                stroke2D3.addPoint(basic_point);
            }
            if(value[i].type === "2DTouch3"){
                stroke2D3.addPoint(new Point3D(value[i].x[0],
                value[i].y[0],0,value[i].count));

                stroke3D.addPoint(basic_point);
                stroke2D1.addPoint(basic_point);
                stroke2D2.addPoint(basic_point);

                stroke2D3.addPoint(new Point3D(value[i].x[1],
                value[i].y[1],0,value[i].count));

                stroke3D.addPoint(basic_point);
                stroke2D1.addPoint(basic_point);
                stroke2D2.addPoint(basic_point);

                stroke2D3.addPoint(new Point3D(value[i].x[2],
                value[i].y[2],0,value[i].count));

                stroke3D.addPoint(basic_point);
                stroke2D1.addPoint(basic_point);
                stroke2D2.addPoint(basic_point);
            }
            else if(value[i].type === "3D"){

                stroke3D.addPoint(new Point3D(value[i].x,
                value[i].y,value[i].z,value[i].count));
```

```
                    stroke2D2.addPoint(basic_point);
                    stroke2D1.addPoint(basic_point);
                    stroke2D3.addPoint(basic_point);
                }

            }
            strokePath2D1.addStroke(stroke2D1);
            strokePath2D2.addStroke(stroke2D2);
            strokePath2D3.addStroke(stroke2D3);
            strokePath3D.addStroke(stroke3D);

            if (gestureSet.getGestureClasses().has(gestureName)) {
                gestureSet.getGestureClasses().get(gestureName)
                .addSample(gestureData);
            } else {
                let gestureClass = new GestureClass(gestureName, gestureIndex);
                gestureIndex += 1;
                gestureClass.addSample(gestureData);
                gestureSet.addGestureClass(gestureClass);
            }



        });

    })

    return gestureSet;
}
module.exports = {
    loadDataset
};
```

### B.4.2 3DTouchpad-Sensor

```
const AbstractSensor = require('../../../framework/modules/sensors/abstract-sensor')
.AbstractSensor
const Point = require('../../../framework/gestures/point').Point3D;


var articulation_field = ["3D","2DTouch1","2DTouch2","2DTouch3"]



class Sensor extends AbstractSensor{

    constructor(options){
        super(`3DTouchPad`);
        this.port = options.port
```

```
        this.lastframe = undefined
        this.net = require('net');
    }

    getPoints(timestamp){
        let points = [];

        if(this.lastframe === undefined){
            return {
                hasData: false,
                points: points,
                appData: {}
              };
        }

        const addMissingPoints = (field,points) => {
            let basic_point = new Point(10,10,10,10);
            for(let i = 0; i < articulation_field.length; i++){
                if (field !== articulation_field[i]){
                    points.push({
                        name : `${articulation_field[i]}`,
                        point : basic_point
                    })
                }
            }
        }
        if(this.lastframe.type === "3D"){
            points.push({
                name: `3D`,
                point: new Point(this.lastframe.x,this.lastframe.y,
                this.lastframe.z,this.lastframe.count)
            })
            addMissingPoints("3D",points)


        }
        else if(this.lastframe.type ==="2DTouch1"){
            points.push({
                name : `2DTouch1`,
                point: new Point(this.lastframe.x[0],this.lastframe.y[0],
                0,this.lastframe.count)
            })
            addMissingPoints("2DTouch1",points)
        }
        else if (this.lastframe.type === "2DTouch2"){
            points.push({
                name : `2DTouch2`,
                point: new Point(this.lastframe.x[0],this.lastframe.y[0],
                0,this.lastframe.count)
```

```
            })
            addMissingPoints("2DTouch2",points)

            points.push({
                name : '2DTouch2',
                point: new Point(this.lastframe.x[1],this.lastframe.y[1],
                0,this.lastframe.count)
            })
            addMissingPoints("2DTouch2",points)
        }
        else if (this.lastframe.type === "2DTouch3"){

            points.push({
                name : '2DTouch3',
                point: new Point(this.lastframe.x[0],this.lastframe.y[0],
                0,this.lastframe.count)
            })
            addMissingPoints("2DTouch3",points)

            points.push({
                name : '2DTouch3',
                point: new Point(this.lastframe.x[1],this.lastframe.y[1],
                0,this.lastframe.count)
            })
            addMissingPoints("2DTouch3",points)

            points.push({
                name : '2DTouch3',
                point: new Point(this.lastframe.x[2],this.lastframe.y[2],
                0,this.lastframe.count)
            })
            addMissingPoints("2DTouch3",points)
        }

        this.lastframe = undefined



        return {
            hasData: true,
            points: points,
            appData: {}
          };
    }



    connect(){
```

```
        this.net.createServer( (socket) => {
                        console.log("TouchPad Connected")
                        socket.on('data', (data) => {
                                try{
                    this.lastframe = JSON.parse(data);
                }
                catch(error){
                }
                        });

                        socket.on('error', function(e){
                                console.log(`3DTouchpad socket is disconnected`);
                        });

                }).listen(this.port);

    }

    disconnect(){

    }

}

module.exports = Sensor;
```

### B.4.3 Testing.js

```
const path = require('path');
const fs = require('graceful-fs');
const fs_original = require('fs');
const GestureSet = require('./gestures/gesture-set').GestureSet;
const GestureClass = require('./gestures/gesture-class').GestureClass;
const stringify = require("json-stringify-pretty-compact");

// Important values
const computeNextT = x => x + 1; // Function that computes the next number of trainin

class Testing {
  constructor(recognizerType, config) {
    this.recognizerType = recognizerType;
    // Get datasets and recognizers
    this.datasets = config.datasets[recognizerType];
    this.recognizers = config.recognizers[recognizerType];
    // Get testing parameters
    this.minT = config.general.testingParams.minT;
    this.maxT = config.general.testingParams.maxT;
    this.r = config.general.testingParams.r;
    // Get global parameters for the recognizers
```

```
    this.n = config.general.globalParams.samplingPoints;
    this.selectedPoints = config.general.globalParams.points;
  }

  run() {
    let results = [];
    for (let i = 0; i < this.datasets.modules.length; i++) {
      let dataset = loadDataset(this.recognizerType, this.datasets);
      let datasetResults = {
        r: this.r,
        dataset: dataset.name,
        gestures: Array.from(dataset.getGestureClasses().keys()),
        data: []
      };
      let res = this.testRecognizer(dataset, this.recognizers);
      console.log(datasetResults)
      results.push(datasetResults);
    }
    console.log('end')
  }


  testRecognizer(dataset, recognizerModule) {
    throw new Error('You have to implement this function');
  }
}

class UserIndependentTesting extends Testing {
  constructor(recognizerType, config) {
    super(recognizerType, config);
  }

  testRecognizer(dataset, recognizers) {

    let count = 1


    let results = [];
    // Perform the test for each size of training set
    for (let trainingSetSize = this.minT; trainingSetSize <=
    Math.min(dataset.getMinTemplate(), this.maxT);
    trainingSetSize = computeNextT(trainingSetSize)) {
      if(this.maxT !== trainingSetSize){
        fs.writeFile(`./test_result/UserIndependantTestResult_
        ${trainingSetSize}TrainingTemplate.csv`, `r = ${this.r},
        Reconaisseurs,Classes de Geste,ID Samples,ID Utilisateur,
        Temps d execution, true or false, Guessed Gesture\n`, (err) => {
          if (err) throw err;
        });
```

```
    }

    // Repeat the test this.r times
    for (let r = 0; r < this.r; r++) {
      // Initialize the recognizer and select the candidates
      let recognizer_tab = []

      for(let i = 0; i < this.recognizers.modules.length ; i++){
        let to_store = this.recognizers.modules[i]
        let recognizer = new to_store.module(to_store.moduleSettings)
        recognizer_tab[i] = recognizer
      }

      let candidates = selectCandidates(dataset);

      // For each gesture class, mark the templates that cannot be reused
      let markedTemplates = [];
      candidates.forEach(candidate => {
        markedTemplates.push([candidate]);
      });
      // Train the recognizer
      for (let t = 0; t < trainingSetSize; t++) { // Add trainingSetSize strokeData
        // Add one sample for each gesture class
        let index = 0;
        dataset.getGestureClasses().forEach((gestureClass) => {
          // Select a valid training template
          let training = -1;
          while (training == -1 ||
          markedTemplates[index].includes(training) ||
          gestureClass.getSamples()[training].user ==
          gestureClass.getSamples()[markedTemplates[index][0]].user) {
            training = getRandomNumber(0, gestureClass.getSamples().length);
          }

          // Mark the training template
          markedTemplates[index].push(training);
          // Train the recognizer
          //console.log(gestureClass.name)

          /
          for(let i = 0; i < this.recognizers.modules.length ; i++){
            recognizer_tab[i].addGesture(gestureClass.name,
            gestureClass.getSamples()[training]);
          }
          index++;
        });
      }
      // Test the recognizer
      let index = 0;
```

```
        dataset.getGestureClasses().forEach((gestureClass) => {

          // Retrieve the testing sample
          let toBeTested = gestureClass.getSamples()[candidates[index]];

          // Attempt recognition
          //console.log(gestureClass.name, toBeTested.id)
          for(let id = 0; id < recognizer_tab.length ; id++){
            if (this.recognizerType === 'dynamic') {
              var result = recognizer_tab[id].recognize(toBeTested);
            } else {
              var result = recognizer_tab[id].recognize(toBeTested.frame);
            }
            //console.log(result.name)
            // Update the confusion matrix
            if (dataset.getGestureClasses().has(result.name)) {
              let resultIndex = dataset.getGestureClasses()
              .get(result.name).index;
              res.confusionMatrix[gestureClass.index][resultIndex] += 1;
            }
            if(result.time === 0){
              result.time += 1;
            }
            res.time += result.time;

            let value = 0
            if(result.name === gestureClass.name){
              value = 1
            }
            if(this.maxT !== trainingSetSize){
            fs.appendFile(`./test_result/UserIndependantTestResult_$
            {trainingSetSize}TrainingTemplate.csv`,`${count},
            ${recognizers.modules[id].module.name},
            ${gestureClass.name},${parseInt(toBeTested.id)+1},
            ${toBeTested.user},${result.time},${value},
            ${result.name}\n`,(err) => {
                if (err) throw err;
              });
            }
            count ++
          }
          index++;
        });
      }

    }

    return results;
  }
```

```
}


class UserDependentTesting extends Testing {
  constructor(recognizerType, config) {
    super(recognizerType, config);
  }

  testRecognizer(dataset, recognizers) {
    let count = 1
    let userIdTab = []

    dataset.getGestureClasses().forEach((gestureClass) => {
      for(let i = 0 ; i < gestureClass.getSamples().length ; i++ ){
        if(!userIdTab.includes(gestureClass.getSamples()[i].user)){
          userIdTab.push(gestureClass.getSamples()[i].user)
        }
      }
    })

    let results = [];
    // Perform the test for each size of training set
    for (let trainingSetSize = this.minT; trainingSetSize <=
    Math.min(dataset.getMinTemplate(), this.maxT); trainingSetSize =
    computeNextT(trainingSetSize)) {
      if(this.maxT !== trainingSetSize || this.maxT === this.minT){
        fs.writeFile('./test_result/UserDependantTestResult_
        ${trainingSetSize}TrainingTemplate.csv', 'r=${this.r},
        Reconaisseurs,Classes de Geste,ID Samples,
        ID Utilisateur, Temps d execution, true or false,
        Guessed Gesture\n', (err) => {
          if (err) throw err;
        });
      }

      // Repeat the test this.r times
      for (let r = 0; r < this.r; r++) {
        // Initialize the recognizer and select the candidates
        for(let user_count = 0 ; user_count < userIdTab.length ;
        user_count++){
          let recognizer_tab = []

          for(let i = 0; i < this.recognizers.modules.length ; i++){
            let to_store = this.recognizers.modules[i]
            let recognizer =
            new to_store.module(to_store.moduleSettings)
            recognizer_tab[i] = recognizer
          }
```

```
        let candidates =
        selectCandidatesOfUser(dataset,userIdTab[user_count]);

        // For each gesture class, mark the templates that cannot be reused
        let markedTemplates = [];
        candidates.forEach(candidate => {
          markedTemplates.push([candidate]);
        });

        // Train the recognizer
        for (let t = 0; t < trainingSetSize; t++) {
        // Add trainingSetSize strokeData per gestureClass
          // Add one sample for each gesture class
          let index = 0;
          dataset.getGestureClasses().forEach((gestureClass) => {
            // Select a valid training template
            let training = -1;
            while (training == -1 ||
            markedTemplates[index].includes(training) ||
            gestureClass.getSamples()[training].user != gestureClass.
            getSamples()[markedTemplates[index][0]].user) {
              training =
              getRandomNumber(0, gestureClass.getSamples().length);
            }
            // Mark the training template
            markedTemplates[index].push(training);
            // Train the recognizer
            //console.log(gestureClass.name)
            for(let i = 0; i < this.recognizers.modules.length ; i++){
              recognizer_tab[i].addGesture(gestureClass.name,
              gestureClass.getSamples()[training]);
            }
            index++;
          });
        }
        // Test the recognizer
        let index = 0;
        dataset.getGestureClasses().forEach((gestureClass) => {
          // Retrieve the testing sample
        // console.log(gestureClass.getSamples())
          let toBeTested = gestureClass.getSamples()[candidates[index]];
          // Attempt recognition
          //console.log(gestureClass.name, toBeTested.id)
          for(let id = 0; id < recognizer_tab.length ; id++){
            if (this.recognizerType === 'dynamic') {
              var result = recognizer_tab[id].recognize(toBeTested);
            } else {
              var result = recognizer_tab[id].recognize(toBeTested.frame);
            }
```

```
                let value = 0
                if(result.name === gestureClass.name){
                  value = 1
                }
                if(this.maxT !== trainingSetSize || this.maxT === this.minT){
                  fs.appendFile(`./test_result/UserDependantTestResult_
                  ${trainingSetSize}TrainingTemplate.csv`,`${count},
                  ${recognizers.modules[id].module.name},
                  ${gestureClass.name},${parseInt(toBeTested.id)+1+1},
                  ${toBeTested.user},${result.time},${value},
                  ${result.name}\n`,(err) => {
                      if (err) throw err;
                  });
                }
                count ++
              }
              index++;
            });
          }
        }
      }

      return results;
    }

}

class AutoMatchingTesting extends Testing {
  constructor(recognizerType, config) {
    super(recognizerType, config);
  }

  testRecognizer(dataset, recognizers) {

    let count = 1


    let results = [];
    // Perform the test for each size of training set
    for (let trainingSetSize = this.minT; trainingSetSize <=
    Math.min(dataset.getMinTemplate(), this.maxT);
    trainingSetSize = computeNextT(trainingSetSize)) {
      if(this.maxT !== trainingSetSize){
        fs.writeFile(`./test_result/AutoMatchingTestResult_
        ${trainingSetSize}TrainingTemplate.csv`,
        `r = ${this.r},Reconaisseurs,Classes de Geste,
        ID Samples,ID Utilisateur, Temps d execution,
        true or false, Guessed Gesture\n`, (err) => {
```

```
        if (err) throw err;
      });
    }

    // Repeat the test this.r times
    for (let r = 0; r < this.r; r++) {
      // Initialize the recognizer and select the candidates
      let recognizer_tab = []

      for(let i = 0; i < this.recognizers.modules.length ; i++){
        let to_store = this.recognizers.modules[i]
        let recognizer = new to_store.module(to_store.moduleSettings)
        recognizer_tab[i] = recognizer
      }

      let candidates = selectCandidates(dataset);
      // For each gesture class, mark the templates that cannot be reused
      let markedTemplates = [];
      candidates.forEach(candidate => {
        markedTemplates.push([candidate]);
      });

      let index_out = 0;
      dataset.getGestureClasses().forEach((gestureClass) => {

        let training = -1;

        while (training == -1 ||
        !markedTemplates[index_out].includes(training)){
          training = getRandomNumber(0, gestureClass.getSamples().length);
        }
        markedTemplates[index_out].push(training);
        for(let i = 0; i < this.recognizers.modules.length ; i++){
          recognizer_tab[i].addGesture(gestureClass.name,
          gestureClass.getSamples()[training]);
        }
        index_out++;
      });


      // Train the recognizer
      for (let t = 0; t < trainingSetSize - 1; t++) {
      // Add trainingSetSize strokeData per gestureClass
        // Add one sample for each gesture class
        let index = 0;
        dataset.getGestureClasses().forEach((gestureClass) => {
          // Select a valid training template
          let training = -1;
          while (training == -1 ||
```

```
        markedTemplates[index].includes(training)) {
          training =
          getRandomNumber(0, gestureClass.getSamples().length);
        }

        // Mark the training template
        markedTemplates[index].push(training);
        // Train the recognizer
        //console.log(gestureClass.name)

        for(let i = 0; i < this.recognizers.modules.length ; i++){
          recognizer_tab[i].addGesture(gestureClass.name,
          gestureClass.getSamples()[training]);
        }
        index++;
      });
    }
    // Test the recognizer
    let index = 0;
    dataset.getGestureClasses().forEach((gestureClass) => {

      // Retrieve the testing sample
      let toBeTested = gestureClass.getSamples()[candidates[index]];

      // Attempt recognition
      //console.log(gestureClass.name, toBeTested.id)
      for(let id = 0; id < recognizer_tab.length ; id++){
        if (this.recognizerType === 'dynamic') {
          var result = recognizer_tab[id].recognize(toBeTested);
        } else {
          var result = recognizer_tab[id].recognize(toBeTested.frame);
        }
        res.time += result.time;

        let value = 0
        if(result.name === gestureClass.name){
          value = 1
        }
        if(this.maxT !== trainingSetSize){
        fs.appendFile(`./test_result/AutoMatchingTestResult_
        ${trainingSetSize}TrainingTemplate.csv`,`${count},
        ${recognizers.modules[id].module.name},
        ${gestureClass.name},${parseInt(toBeTested.id)+1},
        ${toBeTested.user},${result.time},${value},
        ${result.name}\n`,(err) => {
            if (err) throw err;
          });
        }
        count ++
```

```
          }
          index++;
        });
      }

    }

    return results;
  }
}

// HELPER FUNCTIONS

/**
 * Return a random list of candidate gestures, 1 candidate per gesture class.
 */
function selectCandidates(dataset) {
  let candidates = [];
  dataset.getGestureClasses().forEach((value) => {
    candidates.push(getRandomNumber(0, value.getSamples().length));
  });
  return candidates;
};

function selectCandidatesOfUser(dataset,user){
  let candidates = [];
  dataset.getGestureClasses().forEach((value) => {
    let sample = -1
    while(sample == -1 || value.getSamples()[sample].user != user){
      sample = getRandomNumber(0, value.getSamples().length);
    }
    candidates.push(sample)
  });
  return candidates
}

/**
 * Return a random number between min and max.
 */
function getRandomNumber(min, max) {
  min = Math.ceil(min);
  max = Math.floor(max);
  return Math.floor(Math.random() * (max - min)) + min;
};

/**
 * Load a gesture dataset
 */
function loadDataset(type, datasetsConfig) {
```

```
  // Load the dataset
  let datasetLoaderModule = datasetsConfig.modules[0];
  let datasetLoader = datasetLoaderModule.module;
  let identifier = datasetLoaderModule.additionalSettings.id;
  let datasetName = datasetLoaderModule.additionalSettings.datasets[0];
  let datasetPath =
  path.resolve(__dirname, '../datasets', type, datasetName);
  let dataset =
  datasetLoader.loadDataset(datasetName, datasetPath, identifier, [])
  // Select/aggregate/rename classes of the dataset if required
  if (datasetsConfig.aggregateClasses &&
  datasetsConfig.aggregateClasses.length != 0) {
    let newDataset = new GestureSet(dataset.name);
    datasetsConfig.aggregateClasses.forEach((aggregate, index) => {
      // Aggregate gesture class
      let newClass = new GestureClass(aggregate.name, index);
      let templates = [];
      // Fuse the classes into a new aggregate class

      for (const className of aggregate.gestureClasses) {
        let oldClass = dataset.getGestureClasses().get(className);
        templates = templates.concat(templates, oldClass.getSamples());
      }
      // Add the templates to the new gesture class
      for (template of templates) {
        newClass.addSample(template);
      }
      // Add the aggregate class to the new dataset
      newDataset.addGestureClass(newClass);
    });
    return newDataset
  } else {
    return dataset;
  }
}

module.exports = {
  Testing,
  UserDependentTesting,
  UserIndependentTesting,
  AutoMatchingTesting
}
```

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